

# POTATOES

Generated by Doxygen 1.7.1

Thu Oct 28 2010 14:49:30



# Contents

<b>1</b>	<b>Bug List</b>	<b>1</b>
<b>2</b>	<b>Data Structure Index</b>	<b>3</b>
2.1	Data Structures . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Data Structure Documentation</b>	<b>9</b>
4.1	address Struct Reference . . . . .	9
4.1.1	Detailed Description . . . . .	9
4.1.2	Field Documentation . . . . .	9
4.1.2.1	cyl . . . . .	9
4.1.2.2	head . . . . .	9
4.1.2.3	sector . . . . .	9
4.2	block_buffer Struct Reference . . . . .	10
4.2.1	Detailed Description . . . . .	10
4.2.2	Field Documentation . . . . .	10
4.2.2.1	block_nr . . . . .	10
4.2.2.2	cache . . . . .	10
4.3	command_t Struct Reference . . . . .	10
4.3.1	Field Documentation . . . . .	10
4.3.1.1	desc . . . . .	10
4.3.1.2	exec . . . . .	10
4.3.1.3	name . . . . .	11
4.4	cpu_state_t Struct Reference . . . . .	11
4.4.1	Detailed Description . . . . .	11
4.4.2	Field Documentation . . . . .	12
4.4.2.1	cs . . . . .	12

---

4.4.2.2	ds	12
4.4.2.3	eax	12
4.4.2.4	ebp	12
4.4.2.5	ebx	12
4.4.2.6	ecx	12
4.4.2.7	edi	12
4.4.2.8	edx	12
4.4.2.9	eflags	12
4.4.2.10	eip	12
4.4.2.11	err_code	12
4.4.2.12	es	12
4.4.2.13	esi	12
4.4.2.14	esp	12
4.4.2.15	fs	12
4.4.2.16	gs	12
4.4.2.17	int_no	12
4.4.2.18	ss	12
4.4.2.19	useresp	12
4.5	d_inode Struct Reference	12
4.5.1	Detailed Description	13
4.5.2	Field Documentation	13
4.5.2.1	i_create_ts	13
4.5.2.2	i_direct_pointer	13
4.5.2.3	i_double_indirect_pointer	13
4.5.2.4	i_mode	13
4.5.2.5	i_modify_ts	13
4.5.2.6	i_single_indirect_pointer	13
4.5.2.7	i_size	13
4.6	device_t Struct Reference	14
4.6.1	Detailed Description	14
4.6.2	Field Documentation	14
4.6.2.1	close	14
4.6.2.2	data	14
4.6.2.3	fd	14
4.6.2.4	name	15
4.6.2.5	next	15

---

4.6.2.6	open	15
4.6.2.7	read	15
4.6.2.8	seek	15
4.6.2.9	write	15
4.7	dir_entry Struct Reference	15
4.7.1	Detailed Description	15
4.7.2	Field Documentation	16
4.7.2.1	inode	16
4.7.2.2	name	16
4.8	file Struct Reference	16
4.8.1	Detailed Description	16
4.8.2	Field Documentation	16
4.8.2.1	f_count	16
4.8.2.2	f_desc	16
4.8.2.3	f_inode	16
4.8.2.4	f_mode	16
4.8.2.5	f_name	17
4.9	file_info Struct Reference	17
4.9.1	Field Documentation	17
4.9.1.1	create_ts	17
4.9.1.2	mode	17
4.9.1.3	modify_ts	17
4.9.1.4	name	17
4.9.1.5	num_links	17
4.9.1.6	size	18
4.10	gdt_entry Struct Reference	18
4.10.1	Detailed Description	18
4.10.2	Field Documentation	18
4.10.2.1	access	18
4.10.2.2	base_high	18
4.10.2.3	base_low	18
4.10.2.4	base_middle	18
4.10.2.5	granularity	19
4.10.2.6	limit_low	19
4.11	gdt_pointer Struct Reference	19
4.11.1	Detailed Description	19

---

4.11.2	Field Documentation	19
4.11.2.1	base	19
4.11.2.2	limit	19
4.12	hd_info Struct Reference	19
4.12.1	Detailed Description	20
4.12.2	Field Documentation	20
4.12.2.1	apparent_capacity	20
4.12.2.2	apparent_cyl	20
4.12.2.3	apparent_head	21
4.12.2.4	apparent_sector_per_track	21
4.12.2.5	buffer_size	21
4.12.2.6	buffer_type	21
4.12.2.7	bytes_per_sector	21
4.12.2.8	bytes_per_track	21
4.12.2.9	config_word	21
4.12.2.10	dw_io_flg	21
4.12.2.11	firmware	21
4.12.2.12	lba_dma_flg	21
4.12.2.13	manufacturer1	22
4.12.2.14	manufacturer2	22
4.12.2.15	mode_multi_dma	22
4.12.2.16	mode_single_dma	22
4.12.2.17	num_cyl	22
4.12.2.18	num_ecc_bytes	22
4.12.2.19	num_head	22
4.12.2.20	num_lba_sectors	22
4.12.2.21	reserved1	22
4.12.2.22	reserved2	22
4.12.2.23	reserved3	22
4.12.2.24	reserved4	22
4.12.2.25	reserved5	22
4.12.2.26	rw_multiple_flg	22
4.12.2.27	sector_per_track	22
4.12.2.28	sectors_per_int	22
4.12.2.29	serial	22
4.12.2.30	timingmode_dma	22

---

4.12.2.31	timingmode_pio	22
4.12.2.32	type	22
4.13	heap_t Struct Reference	22
4.13.1	Field Documentation	23
4.13.1.1	end	23
4.13.1.2	max_addr	23
4.13.1.3	readonly	23
4.13.1.4	start	23
4.13.1.5	supervisor	23
4.14	idt_entry Struct Reference	23
4.14.1	Detailed Description	24
4.14.2	Field Documentation	24
4.14.2.1	flags	24
4.14.2.2	high_offset	24
4.14.2.3	low_offset	24
4.14.2.4	selector	24
4.14.2.5	separator	24
4.15	idt_pointer Struct Reference	24
4.15.1	Detailed Description	25
4.15.2	Field Documentation	25
4.15.2.1	maxsize	25
4.15.2.2	start	25
4.16	line Struct Reference	25
4.16.1	Field Documentation	25
4.16.1.1	linewidth	25
4.16.1.2	next	25
4.16.1.3	num_chars	25
4.16.1.4	offset	26
4.16.1.5	prev	26
4.17	m_inode Struct Reference	26
4.17.1	Detailed Description	26
4.17.2	Field Documentation	26
4.17.2.1	i_adr	26
4.17.2.2	i_create_ts	26
4.17.2.3	i_direct_pointer	26
4.17.2.4	i_double_indirect_pointer	27

4.17.2.5	<code>i_mode</code>	27
4.17.2.6	<code>i_modify_ts</code>	27
4.17.2.7	<code>i_num</code>	27
4.17.2.8	<code>i_single_indirect_pointer</code>	27
4.17.2.9	<code>i_size</code>	27
4.18	<code>mm_header</code> Struct Reference	27
4.18.1	Detailed Description	28
4.18.2	Field Documentation	28
4.18.2.1	<code>name</code>	28
4.18.2.2	<code>next</code>	28
4.18.2.3	<code>prev</code>	28
4.18.2.4	<code>size</code>	28
4.19	<code>multiboot</code> Struct Reference	28
4.19.1	Detailed Description	29
4.19.2	Field Documentation	29
4.19.2.1	<code>addr</code>	29
4.19.2.2	<code>apm_table</code>	29
4.19.2.3	<code>boot_device</code>	29
4.19.2.4	<code>boot_loader_name</code>	29
4.19.2.5	<code>cmdline</code>	29
4.19.2.6	<code>config_table</code>	29
4.19.2.7	<code>drives_addr</code>	29
4.19.2.8	<code>drives_length</code>	29
4.19.2.9	<code>flags</code>	29
4.19.2.10	<code>mem_lower</code>	29
4.19.2.11	<code>mem_upper</code>	29
4.19.2.12	<code>mmap_addr</code>	30
4.19.2.13	<code>mmap_length</code>	30
4.19.2.14	<code>mods_addr</code>	30
4.19.2.15	<code>mods_count</code>	30
4.19.2.16	<code>num</code>	30
4.19.2.17	<code>shndx</code>	30
4.19.2.18	<code>size</code>	30
4.19.2.19	<code>vbe_control_info</code>	30
4.19.2.20	<code>vbe_interface_len</code>	30
4.19.2.21	<code>vbe_interface_off</code>	30



---

4.19.2.22	vbe_interface_seg	30
4.19.2.23	vbe_mode	30
4.19.2.24	vbe_mode_info	30
4.20	page Struct Reference	30
4.20.1	Field Documentation	31
4.20.1.1	accessed	31
4.20.1.2	avail	31
4.20.1.3	dirty	31
4.20.1.4	frame	31
4.20.1.5	present	31
4.20.1.6	res	31
4.20.1.7	res2	31
4.20.1.8	rw	31
4.20.1.9	user	31
4.21	page_directory Struct Reference	31
4.21.1	Field Documentation	31
4.21.1.1	physicalAddr	31
4.21.1.2	tables	31
4.21.1.3	tablesPhysical	32
4.22	page_table Struct Reference	32
4.22.1	Field Documentation	32
4.22.1.1	pages	32
4.23	potatoes_dir_entry Struct Reference	32
4.23.1	Field Documentation	32
4.23.1.1	inode	32
4.23.1.2	name	32
4.24	potatoes_file_info Struct Reference	33
4.24.1	Field Documentation	33
4.24.1.1	create_ts	33
4.24.1.2	mode	33
4.24.1.3	modify_ts	33
4.24.1.4	name	33
4.24.1.5	num_links	33
4.24.1.6	size	33
4.25	potatoes_time Struct Reference	33
4.25.1	Field Documentation	34

4.25.1.1	century	34
4.25.1.2	day	34
4.25.1.3	hour	34
4.25.1.4	min	34
4.25.1.5	month	34
4.25.1.6	sec	34
4.25.1.7	weekday	34
4.25.1.8	year	34
4.26	PotatoesDisk Struct Reference	34
4.26.1	Field Documentation	34
4.26.1.1	data	34
4.26.1.2	size	34
4.27	proc_file Struct Reference	35
4.27.1	Detailed Description	35
4.27.2	Field Documentation	35
4.27.2.1	pf_desc	35
4.27.2.2	pf_f_desc	35
4.27.2.3	pf_pos	35
4.28	process_t Struct Reference	35
4.28.1	Detailed Description	36
4.28.2	Field Documentation	36
4.28.2.1	addr	36
4.28.2.2	context	36
4.28.2.3	name	37
4.28.2.4	next	37
4.28.2.5	pft	37
4.28.2.6	pid	37
4.28.2.7	priority	37
4.28.2.8	remaining_timeslices	37
4.28.2.9	stack_start	37
4.28.2.10	state	37
4.28.2.11	stdin	38
4.28.2.12	timeslice	38
4.28.2.13	vmonitor	38
4.29	ring_fifo Struct Reference	38
4.29.1	Detailed Description	38

---

4.29.2	Field Documentation	39
4.29.2.1	data	39
4.29.2.2	end	39
4.29.2.3	len	39
4.29.2.4	size	39
4.29.2.5	start	39
4.30	sc_close_args_t Struct Reference	39
4.30.1	Detailed Description	39
4.30.2	Field Documentation	40
4.30.2.1	fd	40
4.30.2.2	success	40
4.31	sc_malloc_args_t Struct Reference	40
4.31.1	Detailed Description	40
4.31.2	Field Documentation	40
4.31.2.1	mem	40
4.31.2.2	size	40
4.32	sc_open_args_t Struct Reference	40
4.32.1	Detailed Description	41
4.32.2	Field Documentation	41
4.32.2.1	fd	41
4.32.2.2	mode	41
4.32.2.3	oflag	41
4.32.2.4	path	41
4.33	sc_read_write_args_t Struct Reference	41
4.33.1	Detailed Description	42
4.33.2	Field Documentation	42
4.33.2.1	buf	42
4.33.2.2	fd	42
4.33.2.3	rw_count	42
4.33.2.4	size	42
4.34	sc_seek_args_t Struct Reference	42
4.34.1	Detailed Description	42
4.34.2	Field Documentation	42
4.34.2.1	fd	42
4.34.2.2	offset	43
4.34.2.3	pos	43

4.34.2.4	whence	43
4.35	sc_stat_args_t Struct Reference	43
4.35.1	Detailed Description	43
4.35.2	Field Documentation	43
4.35.2.1	buf	43
4.35.2.2	path	43
4.35.2.3	success	43
4.36	sc_unlink_args_t Struct Reference	44
4.36.1	Detailed Description	44
4.36.2	Field Documentation	44
4.36.2.1	path	44
4.36.2.2	success	44
4.37	Semaphore Struct Reference	44
4.37.1	Field Documentation	44
4.37.1.1	count	44
4.37.1.2	lock	44
4.38	shell_cmd_t Struct Reference	45
4.38.1	Detailed Description	45
4.38.2	Field Documentation	45
4.38.2.1	cmd	45
4.38.2.2	desc	45
4.38.2.3	name	45
4.39	shortcut Struct Reference	45
4.39.1	Detailed Description	46
4.39.2	Field Documentation	46
4.39.2.1	ch	46
4.39.2.2	control	46
4.39.2.3	func	46
4.39.2.4	super	46
4.40	super_block Struct Reference	46
4.40.1	Field Documentation	46
4.40.1.1	s_bmap	46
4.40.1.2	s_bmap_blocks	47
4.40.1.3	s_dirt	47
4.40.1.4	s_first_data_block	47
4.40.1.5	s_HD_size	47

---

4.40.1.6	s_iroot	47
4.40.1.7	s_magic_number	47
4.40.1.8	s_max_file_size	47
4.40.1.9	s_modify_ts	47
4.40.1.10	s_read_only	47
4.41	time Struct Reference	47
4.41.1	Detailed Description	48
4.41.2	Field Documentation	48
4.41.2.1	century	48
4.41.2.2	day	48
4.41.2.3	hour	48
4.41.2.4	min	48
4.41.2.5	month	48
4.41.2.6	sec	48
4.41.2.7	weekday	48
4.41.2.8	year	49
4.42	Tone Struct Reference	49
4.42.1	Detailed Description	49
4.42.2	Field Documentation	49
4.42.2.1	duration	49
4.42.2.2	frequency	49
4.43	virt_monitor Struct Reference	49
4.43.1	Detailed Description	50
4.43.2	Field Documentation	50
4.43.2.1	begin	50
4.43.2.2	disable_refresh	50
4.43.2.3	name	50
4.43.2.4	offset	51
4.43.2.5	pid	51
4.43.2.6	scrolldown_limit	51
4.43.2.7	scrollup_limit	51
4.43.2.8	size	51
4.43.2.9	vis_begin	51
<b>5</b>	<b>File Documentation</b>	<b>53</b>
5.1	src/apps/apps.h File Reference	53
5.1.1	Detailed Description	53

---

5.1.2	Function Documentation . . . . .	53
5.1.2.1	shell_cmd_memview . . . . .	53
5.1.2.2	shell_cmd_snapshot . . . . .	54
5.1.2.3	shell_cmd_speed . . . . .	54
5.2	src/apps/brainfuck_interpreter.c File Reference . . . . .	54
5.2.1	Detailed Description . . . . .	55
5.2.2	Function Documentation . . . . .	55
5.2.2.1	init_bf . . . . .	55
5.2.2.2	interpret_bf . . . . .	55
5.2.2.3	reset_bf . . . . .	55
5.2.3	Variable Documentation . . . . .	55
5.2.3.1	bf_buf_offset . . . . .	55
5.2.3.2	bf_buf_position . . . . .	56
5.2.3.3	bf_buffer . . . . .	56
5.2.3.4	bf_ptr . . . . .	56
5.2.3.5	bf_start . . . . .	56
5.2.3.6	jumpover . . . . .	56
5.2.3.7	loop_marks . . . . .	56
5.2.3.8	loopdepth . . . . .	56
5.2.3.9	maxlength . . . . .	56
5.2.3.10	maxloopdepth . . . . .	56
5.2.3.11	STDERR . . . . .	56
5.2.3.12	store_lock . . . . .	56
5.2.3.13	store_lock_level . . . . .	56
5.2.3.14	temploopdepth . . . . .	57
5.3	src/apps/brainfuck_interpreter.h File Reference . . . . .	57
5.3.1	Detailed Description . . . . .	57
5.3.2	Define Documentation . . . . .	57
5.3.2.1	BRAINFUCK . . . . .	57
5.3.3	Function Documentation . . . . .	57
5.3.3.1	init_bf . . . . .	57
5.3.3.2	interpret_bf . . . . .	58
5.3.3.3	reset_bf . . . . .	58
5.4	src/apps/editor.c File Reference . . . . .	58
5.4.1	Detailed Description . . . . .	58
5.4.2	Function Documentation . . . . .	59

---

5.4.2.1	resize_buf	59
5.4.2.2	shell_cmd_speed	59
5.4.2.3	speed	59
5.5	src/apps/editor.h File Reference	59
5.5.1	Detailed Description	59
5.5.2	Define Documentation	60
5.5.2.1	DUMP_LINES	60
5.5.2.2	NEXT_LINE	60
5.5.2.3	PREV_LINE	60
5.5.3	Typedef Documentation	61
5.5.3.1	line	61
5.6	src/apps/games.h File Reference	61
5.6.1	Detailed Description	62
5.6.2	Define Documentation	63
5.6.2.1	CURSOR_DOWN	63
5.6.2.2	CURSOR_LEFT	63
5.6.2.3	CURSOR_RIGHT	63
5.6.2.4	CURSOR_UP	63
5.6.2.5	DRAW_GLYPH	63
5.6.2.6	DRAW_PADDLE	63
5.6.2.7	EAT_APPLE_SOUND	63
5.6.2.8	ENTER	63
5.6.2.9	ESCAPE	63
5.6.2.10	HIT_LPADDLE_SOUND	63
5.6.2.11	HIT_PADDLE	64
5.6.2.12	HIT_HPADDLE_SOUND	64
5.6.2.13	HIT_SIDE_SOUND	64
5.6.2.14	KEY_A	64
5.6.2.15	KEY_D	64
5.6.2.16	KEY_E	64
5.6.2.17	KEY_F	64
5.6.2.18	KEY_G	64
5.6.2.19	KEY_H	64
5.6.2.20	KEY_I	64
5.6.2.21	KEY_J	64
5.6.2.22	KEY_K	65

5.6.2.23	KEY_L	65
5.6.2.24	KEY_MINUS	65
5.6.2.25	KEY_PLUS	65
5.6.2.26	KEY_Q	65
5.6.2.27	KEY_R	65
5.6.2.28	KEY_S	65
5.6.2.29	KEY_T	65
5.6.2.30	KEY_U	65
5.6.2.31	KEY_W	65
5.6.2.32	KEY_Z	65
5.6.2.33	LIMIT	65
5.6.2.34	PADDLE_DEFLECTION	65
5.6.2.35	SET_PIXEL	66
5.6.2.36	SHARP	66
5.6.2.37	SNAKE_DOWN	66
5.6.2.38	SNAKE_LEFT	66
5.6.2.39	SNAKE_RIGHT	66
5.6.2.40	SNAKE_UP	66
5.6.3	Typedef Documentation	66
5.6.3.1	glyph_t	66
5.6.4	Function Documentation	66
5.6.4.1	keydown	66
5.6.4.2	shell_cmd_pong	66
5.6.4.3	shell_cmd_snake	66
5.6.4.4	shell_cmd_synth	66
5.7	src/apps/memview.c File Reference	67
5.7.1	Detailed Description	68
5.7.2	Define Documentation	69
5.7.2.1	MEMVIEW_MAX_DUMMY_BLOCKS	69
5.7.3	Function Documentation	69
5.7.3.1	allocate_dummy_block	69
5.7.3.2	free_all_dummy_blocks	69
5.7.3.3	free_dummy_block	69
5.7.3.4	get_free_dummy_slot	70
5.7.3.5	heap_get_size	70
5.7.3.6	keydown	70



---

5.7.3.7	mark_visual_block	70
5.7.3.8	memview_main	71
5.7.3.9	mv_do_benchmark	71
5.7.3.10	mv_show_stats	71
5.7.3.11	mv_switch_to_graphicsmode	71
5.7.3.12	mv_switch_to_textmode	71
5.7.3.13	shell_cmd_memview	71
5.7.3.14	update_view	71
5.7.4	Variable Documentation	72
5.7.4.1	mv_allocated_blocks	72
5.7.4.2	mv_bytes_per_block	72
5.7.4.3	mv_disp	72
5.7.4.4	mv_framebuf	72
5.7.4.5	mv_total_mem	72
5.8	src/apps/pong.c File Reference	72
5.8.1	Detailed Description	73
5.8.2	Function Documentation	73
5.8.2.1	keydown	73
5.8.2.2	shell_cmd_pong	73
5.8.3	Variable Documentation	74
5.8.3.1	STDIN	74
5.9	src/apps/shell_cmds.c File Reference	74
5.9.1	Detailed Description	76
5.9.2	Function Documentation	76
5.9.2.1	fs_init	76
5.9.2.2	fs_shutdown	77
5.9.2.3	mem_dump	77
5.9.2.4	pm_dump	77
5.9.2.5	reset_bf	77
5.9.2.6	shell_cmd_bf	77
5.9.2.7	shell_cmd_cat	77
5.9.2.8	shell_cmd_cd	77
5.9.2.9	shell_cmd_clear	78
5.9.2.10	shell_cmd_cmdlist	78
5.9.2.11	shell_cmd_cp	78
5.9.2.12	shell_cmd_date	78

---

5.9.2.13	shell_cmd_echo	79
5.9.2.14	shell_cmd_exec	79
5.9.2.15	shell_cmd_exit	79
5.9.2.16	shell_cmd_kill	79
5.9.2.17	shell_cmd_ls	79
5.9.2.18	shell_cmd_memdump	80
5.9.2.19	shell_cmd_mkdir	80
5.9.2.20	shell_cmd_nice	80
5.9.2.21	shell_cmd_ps	80
5.9.2.22	shell_cmd_pwd	81
5.9.2.23	shell_cmd_rm	81
5.9.2.24	shell_cmd_sync	81
5.9.2.25	shell_cmd_test	81
5.9.2.26	shell_cmd_touch	81
5.9.2.27	shell_cmd_write	82
5.9.3	Variable Documentation	82
5.9.3.1	shell_cmds	82
5.10	src/apps/shell_main.c File Reference	82
5.10.1	Detailed Description	83
5.10.2	Function Documentation	83
5.10.2.1	new_shell	83
5.10.2.2	shell_autocomplete	83
5.10.2.3	shell_handle_command	84
5.10.2.4	shell_main	84
5.10.3	Variable Documentation	84
5.10.3.1	cwd	84
5.10.3.2	path_buf	84
5.11	src/apps/shell_main.h File Reference	84
5.11.1	Detailed Description	85
5.11.2	Typedef Documentation	86
5.11.2.1	shell_cmd_func	86
5.11.2.2	shell_cmd_t	86
5.11.3	Function Documentation	86
5.11.3.1	new_shell	86
5.11.3.2	shell_handle_command	86
5.11.4	Variable Documentation	86

---

5.11.4.1	<code>cwd</code>	86
5.11.4.2	<code>path_buf</code>	86
5.11.4.3	<code>shell_cmds</code>	87
5.12	<code>src/apps/shell_utils.c</code> File Reference	87
5.12.1	Detailed Description	88
5.12.2	Function Documentation	88
5.12.2.1	<code>_fgetch</code>	88
5.12.2.2	<code>_fgets</code>	88
5.12.2.3	<code>_fputch</code>	89
5.12.2.4	<code>_fputs</code>	89
5.12.2.5	<code>_printf</code>	89
5.12.2.6	<code>halt</code>	90
5.12.2.7	<code>shell_makepath</code>	90
5.12.3	Variable Documentation	90
5.12.3.1	<code>STDIN</code>	90
5.12.3.2	<code>STDOUT</code>	90
5.13	<code>src/apps/shell_utils.h</code> File Reference	90
5.13.1	Detailed Description	91
5.13.2	Function Documentation	92
5.13.2.1	<code>_fgetch</code>	92
5.13.2.2	<code>_fgets</code>	92
5.13.2.3	<code>_fputch</code>	92
5.13.2.4	<code>_fputs</code>	92
5.13.2.5	<code>_printf</code>	93
5.13.2.6	<code>shell_makepath</code>	93
5.13.3	Variable Documentation	93
5.13.3.1	<code>STDIN</code>	93
5.13.3.2	<code>STDOUT</code>	93
5.14	<code>src/apps/snake.c</code> File Reference	93
5.14.1	Detailed Description	94
5.14.2	Function Documentation	94
5.14.2.1	<code>body_collision</code>	94
5.14.2.2	<code>draw_snake</code>	94
5.14.2.3	<code>shell_cmd_snake</code>	95
5.14.2.4	<code>snake</code>	95
5.14.3	Variable Documentation	95

5.14.3.1	STDIN	95
5.15	src/apps/snapshot.c File Reference	95
5.15.1	Detailed Description	96
5.15.2	Function Documentation	96
5.15.2.1	make_snapshot	96
5.15.2.2	shell_cmd_snapshot	96
5.15.2.3	shell_makepath	96
5.15.2.4	snapshot	97
5.15.3	Variable Documentation	97
5.15.3.1	snap_buffer	97
5.15.3.2	STDIN	97
5.15.3.3	STDOUT	97
5.16	src/apps/synthesizer.c File Reference	97
5.16.1	Detailed Description	98
5.16.2	Function Documentation	98
5.16.2.1	shell_cmd_synth	98
5.16.2.2	synth	98
5.17	src/kernel/fs/fs_block_dev.c File Reference	98
5.17.1	Detailed Description	99
5.17.2	Function Documentation	100
5.17.2.1	cache_block	100
5.17.2.2	clear_block	100
5.17.2.3	enlarge_file	100
5.17.2.4	get_data_block	101
5.17.2.5	rd_block	101
5.17.2.6	wrt_block	101
5.17.2.7	wrt_cache	102
5.18	src/kernel/fs/fs_block_dev.h File Reference	102
5.18.1	Detailed Description	102
5.18.2	Function Documentation	103
5.18.2.1	cache_block	103
5.18.2.2	clear_block	103
5.18.2.3	enlarge_file	103
5.18.2.4	get_data_block	104
5.18.2.5	rd_block	104
5.18.2.6	wrt_block	104

---

5.18.2.7	wrt_cache	105
5.19	src/kernel/fs/fs_bmap.c File Reference	105
5.19.1	Detailed Description	106
5.19.2	Function Documentation	106
5.19.2.1	alloc_block	106
5.19.2.2	dump_bmap	106
5.19.2.3	get_free_block	107
5.19.2.4	get_hdsize	107
5.19.2.5	init_bmap	107
5.19.2.6	is_allocated_block	107
5.19.2.7	load_bmap	107
5.19.2.8	malloc_bmap	108
5.19.2.9	mark_block	108
5.19.2.10	write_bmap	108
5.20	src/kernel/fs/fs_bmap.h File Reference	108
5.20.1	Detailed Description	109
5.20.2	Function Documentation	110
5.20.2.1	alloc_block	110
5.20.2.2	dump_bmap	110
5.20.2.3	get_free_block	110
5.20.2.4	init_bmap	110
5.20.2.5	is_allocated_block	110
5.20.2.6	load_bmap	111
5.20.2.7	malloc_bmap	111
5.20.2.8	mark_block	111
5.20.2.9	write_bmap	111
5.20.3	Variable Documentation	112
5.20.3.1	bmap	112
5.20.3.2	first_data_block	112
5.20.3.3	num_bmap_blocks	112
5.21	src/kernel/fs/fs_buf.c File Reference	112
5.21.1	Detailed Description	112
5.21.2	Function Documentation	113
5.21.2.1	clear_buffer	113
5.21.2.2	clear_cache	113
5.22	src/kernel/fs/fs_buf.h File Reference	113

---

5.22.1	Detailed Description	114
5.22.2	Function Documentation	114
5.22.2.1	clear_buffer	114
5.22.2.2	clear_cache	115
5.22.3	Variable Documentation	115
5.22.3.1	addr_cache	115
5.22.3.2	d_inode_cache	115
5.22.3.3	dir_cache	115
5.22.3.4	m_inode_cache	115
5.22.3.5	read_buffer	115
5.22.3.6	read_cache	115
5.22.3.7	write_buffer	115
5.22.3.8	write_cache	116
5.23	src/kernel/fs/fs_const.h File Reference	116
5.23.1	Detailed Description	117
5.23.2	Define Documentation	117
5.23.2.1	ADDR_SIZE	117
5.23.2.2	ADDRS_PER_BLOCK	117
5.23.2.3	BLOCK_SIZE	117
5.23.2.4	BOOT_BLOCK	117
5.23.2.5	BYTES_DIRECT	117
5.23.2.6	BYTES_DOUBLE_INDIRECT	117
5.23.2.7	BYTES_SINGLE_INDIRECT	117
5.23.2.8	CLEAN	118
5.23.2.9	DATA_FILE	118
5.23.2.10	DIR_ENTRIES_PER_BLOCK	118
5.23.2.11	DIR_ENTRY_SIZE	118
5.23.2.12	DIRECTORY	118
5.23.2.13	DIRTY	118
5.23.2.14	DISK_INODE_SIZE	118
5.23.2.15	FIRST_BMAP_BLOCK	118
5.23.2.16	INODES_PER_BLOCK	118
5.23.2.17	MAGIC_NUMBER	118
5.23.2.18	MEM_INODE_SIZE	118
5.23.2.19	NAME_SIZE	118
5.23.2.20	NOT_EXISTENT	119

---

5.23.2.21	NOT_FOUND	119
5.23.2.22	NOT_POSSIBLE	119
5.23.2.23	NUM_DIRECT_POINTER	119
5.23.2.24	NUM_FILES	119
5.23.2.25	NUM_INODES	119
5.23.2.26	NUM_PROC_FILES	119
5.23.2.27	ROOT_INODE	119
5.23.2.28	ROOT_INODE_BLOCK	119
5.23.2.29	SUPER_BLOCK	119
5.23.2.30	SUPER_SIZE	119
5.24	src/kernel/fs/fs_create_delete.c File Reference	120
5.24.1	Detailed Description	120
5.24.2	Function Documentation	121
5.24.2.1	fs_create	121
5.24.2.2	fs_create_delete	121
5.24.2.3	fs_delete	122
5.24.2.4	fs_truncate	122
5.25	src/kernel/fs/fs_dir.c File Reference	122
5.25.1	Detailed Description	123
5.25.2	Function Documentation	124
5.25.2.1	contains_filename	124
5.25.2.2	create_entry	124
5.25.2.3	delete_entry	124
5.25.2.4	delete_file_from_dir	125
5.25.2.5	find_filename	125
5.25.2.6	get_filename	125
5.25.2.7	get_path	126
5.25.2.8	insert_file_into_dir	126
5.25.2.9	rfsearch	126
5.25.2.10	search_file	127
5.26	src/kernel/fs/fs_dir.h File Reference	127
5.26.1	Detailed Description	128
5.26.2	Function Documentation	128
5.26.2.1	contains_filename	128
5.26.2.2	create_entry	128
5.26.2.3	delete_entry	129

5.26.2.4	delete_file_from_dir . . . . .	129
5.26.2.5	find_filename . . . . .	129
5.26.2.6	get_filename . . . . .	130
5.26.2.7	get_path . . . . .	130
5.26.2.8	insert_file_into_dir . . . . .	130
5.26.2.9	rfsearch . . . . .	131
5.26.2.10	search_file . . . . .	131
5.27	src/kernel/fs/fs_file_table.c File Reference . . . . .	131
5.27.1	Detailed Description . . . . .	133
5.27.2	Function Documentation . . . . .	133
5.27.2.1	alloc_file . . . . .	133
5.27.2.2	alloc_proc_file . . . . .	133
5.27.2.3	contains_file . . . . .	134
5.27.2.4	dump_file . . . . .	134
5.27.2.5	dump_files . . . . .	134
5.27.2.6	dump_proc_file . . . . .	134
5.27.2.7	dump_proc_files . . . . .	135
5.27.2.8	free_file . . . . .	135
5.27.2.9	free_proc_file . . . . .	135
5.27.2.10	get_file . . . . .	135
5.27.2.11	get_file_info . . . . .	136
5.27.2.12	get_proc_file . . . . .	136
5.27.2.13	inc_count . . . . .	136
5.27.2.14	init_file_table . . . . .	136
5.27.2.15	init_proc_file_table . . . . .	137
5.27.2.16	inode2desc . . . . .	137
5.27.2.17	insert_file . . . . .	137
5.27.2.18	insert_proc_file . . . . .	137
5.27.2.19	lseek . . . . .	138
5.27.2.20	name2desc . . . . .	138
5.28	src/kernel/fs/fs_file_table.h File Reference . . . . .	138
5.28.1	Detailed Description . . . . .	140
5.28.2	Define Documentation . . . . .	140
5.28.2.1	NIL_FILE . . . . .	140
5.28.2.2	NIL_PROC_FILE . . . . .	140
5.28.3	Function Documentation . . . . .	140



---

5.28.3.1	alloc_file	140
5.28.3.2	alloc_proc_file	141
5.28.3.3	contains_file	141
5.28.3.4	dump_file	141
5.28.3.5	dump_files	141
5.28.3.6	dump_proc_file	142
5.28.3.7	dump_proc_files	142
5.28.3.8	free_file	142
5.28.3.9	free_proc_file	142
5.28.3.10	get_file	142
5.28.3.11	get_proc_file	143
5.28.3.12	inc_count	143
5.28.3.13	init_file_table	143
5.28.3.14	init_proc_file_table	143
5.28.3.15	inode2desc	143
5.28.3.16	insert_file	144
5.28.3.17	insert_proc_file	144
5.28.3.18	lseek	144
5.28.3.19	name2desc	145
5.28.4	Variable Documentation	145
5.28.4.1	gft	145
5.29	src/kernel/fs/fs_inode_table.c File Reference	145
5.29.1	Detailed Description	147
5.29.2	Function Documentation	147
5.29.2.1	alloc_inode	147
5.29.2.2	cpy_dinode_to_minode	147
5.29.2.3	cpy_minode_to_dinode	147
5.29.2.4	create_root	148
5.29.2.5	dump_dinode	148
5.29.2.6	dump_inode	148
5.29.2.7	dump_inodes	148
5.29.2.8	free_inode	149
5.29.2.9	get_inode	149
5.29.2.10	init_inode_table	149
5.29.2.11	load_root	149
5.29.2.12	new_minode	149

5.29.2.13	read_dinode	150
5.29.2.14	read_minode	150
5.29.2.15	write_inode	150
5.29.2.16	write_inodes	151
5.29.2.17	write_root	151
5.30	src/kernel/fs/fs_inode_table.h File Reference	151
5.30.1	Detailed Description	152
5.30.2	Define Documentation	153
5.30.2.1	NIL_INODE	153
5.30.3	Function Documentation	153
5.30.3.1	alloc_inode	153
5.30.3.2	cpy_dinode_to_minode	153
5.30.3.3	cpy_minode_to_dinode	153
5.30.3.4	create_root	154
5.30.3.5	dump_inode	154
5.30.3.6	dump_inodes	154
5.30.3.7	free_inode	154
5.30.3.8	get_inode	154
5.30.3.9	init_inode_table	155
5.30.3.10	load_root	155
5.30.3.11	new_minode	155
5.30.3.12	read_dinode	155
5.30.3.13	read_minode	156
5.30.3.14	write_inode	156
5.30.3.15	write_inodes	156
5.30.3.16	write_root	156
5.30.4	Variable Documentation	156
5.30.4.1	inode_table	156
5.30.4.2	root	156
5.31	src/kernel/fs/fs_io_functions.h File Reference	157
5.31.1	Detailed Description	158
5.31.2	Define Documentation	158
5.31.2.1	CREATE	158
5.31.2.2	DELETE	158
5.31.3	Function Documentation	158
5.31.3.1	fs_close	158

---

5.31.3.2	<a href="#">fs_create</a>	158
5.31.3.3	<a href="#">fs_create_delete</a>	159
5.31.3.4	<a href="#">fs_delete</a>	159
5.31.3.5	<a href="#">fs_open</a>	160
5.31.3.6	<a href="#">fs_read</a>	160
5.31.3.7	<a href="#">fs_truncate</a>	160
5.31.3.8	<a href="#">fs_write</a>	161
5.31.3.9	<a href="#">lseek</a>	161
5.31.3.10	<a href="#">read_dinode</a>	161
5.31.3.11	<a href="#">read_minode</a>	162
5.31.3.12	<a href="#">write_inode</a>	162
5.32	<a href="#">src/kernel/fs/fs_main.c File Reference</a>	162
5.32.1	<a href="#">Detailed Description</a>	163
5.32.2	<a href="#">Function Documentation</a>	164
5.32.2.1	<a href="#">create_fs</a>	164
5.32.2.2	<a href="#">do_close</a>	164
5.32.2.3	<a href="#">do_close_pf</a>	164
5.32.2.4	<a href="#">do_create</a>	164
5.32.2.5	<a href="#">do_file_exists</a>	164
5.32.2.6	<a href="#">do_lseek</a>	164
5.32.2.7	<a href="#">do_mkdir</a>	164
5.32.2.8	<a href="#">do_mkfile</a>	164
5.32.2.9	<a href="#">do_open</a>	165
5.32.2.10	<a href="#">do_read</a>	165
5.32.2.11	<a href="#">do_remove</a>	165
5.32.2.12	<a href="#">do_write</a>	165
5.32.2.13	<a href="#">dump_consts</a>	165
5.32.2.14	<a href="#">fs_init</a>	165
5.32.2.15	<a href="#">fs_shutdown</a>	165
5.32.2.16	<a href="#">load_fs</a>	166
5.32.2.17	<a href="#">panic</a>	166
5.32.2.18	<a href="#">search_file</a>	166
5.33	<a href="#">src/kernel/fs/fs_main.h File Reference</a>	166
5.33.1	<a href="#">Detailed Description</a>	167
5.33.2	<a href="#">Function Documentation</a>	167
5.33.2.1	<a href="#">create_fs</a>	167

5.33.2.2	do_close	167
5.33.2.3	do_close_pf	168
5.33.2.4	do_create	168
5.33.2.5	do_file_exists	168
5.33.2.6	do_lseek	168
5.33.2.7	do_mkdir	168
5.33.2.8	do_mkfile	168
5.33.2.9	do_open	168
5.33.2.10	do_read	168
5.33.2.11	do_remove	168
5.33.2.12	do_write	169
5.33.2.13	dump_consts	169
5.33.2.14	fs_init	169
5.33.2.15	fs_shutdown	169
5.33.2.16	get_file_info	169
5.33.2.17	load_fs	169
5.34	src/kernel/fs/fs_open_close.c File Reference	169
5.34.1	Detailed Description	170
5.34.2	Function Documentation	170
5.34.2.1	fs_close	170
5.34.2.2	fs_open	171
5.35	src/kernel/fs/fs_read_write.c File Reference	171
5.35.1	Detailed Description	172
5.35.2	Function Documentation	172
5.35.2.1	fs_read	172
5.35.2.2	fs_write	173
5.36	src/kernel/fs/fs_super.c File Reference	173
5.36.1	Detailed Description	174
5.36.2	Function Documentation	174
5.36.2.1	dump_super	174
5.36.2.2	get_hdsiz	174
5.36.2.3	init_super_block	175
5.36.2.4	load_super_block	175
5.36.2.5	write_super_block	175
5.36.3	Variable Documentation	175
5.36.3.1	super	175

---

5.37	src/kernel/fs/fs_super.h File Reference	175
5.37.1	Detailed Description	176
5.37.2	Define Documentation	176
5.37.2.1	NIL_SUPER	176
5.37.3	Function Documentation	177
5.37.3.1	__attribute__	177
5.37.3.2	dump_super	177
5.37.3.3	init_super_block	177
5.37.3.4	load_super_block	177
5.37.3.5	write_super_block	177
5.37.4	Variable Documentation	177
5.37.4.1	root	177
5.37.4.2	s_bmap	178
5.37.4.3	s_bmap_blocks	178
5.37.4.4	s_dirt	178
5.37.4.5	s_first_data_block	178
5.37.4.6	s_HD_size	178
5.37.4.7	s_iroot	178
5.37.4.8	s_magic_number	178
5.37.4.9	s_max_file_size	178
5.37.4.10	s_modify_ts	178
5.37.4.11	s_read_only	178
5.38	src/kernel/fs/fs_tests.c File Reference	178
5.38.1	Detailed Description	179
5.38.2	Function Documentation	179
5.38.2.1	__ls	179
5.38.2.2	mem_dump	179
5.38.2.3	run_FS_tests	179
5.38.2.4	test_bmap	180
5.38.2.5	test_close	180
5.38.2.6	test_create	180
5.38.2.7	test_delete	180
5.38.2.8	test_error	180
5.38.2.9	test_file_table	180
5.38.2.10	test_inode_table	180
5.38.2.11	test_ls	180

5.38.2.12	test_open_close	180
5.38.2.13	test_PM	180
5.38.2.14	test_reload	181
5.38.2.15	test_rw_qualitative	181
5.38.2.16	test_rw_quantitative	181
5.38.2.17	test_sync	181
5.39	src/kernel/fs/fs_types.h File Reference	181
5.39.1	Detailed Description	182
5.39.2	Typedef Documentation	183
5.39.2.1	block_buffer	183
5.39.2.2	block_cache	183
5.39.2.3	block_nr	183
5.39.2.4	file	183
5.39.2.5	file_info_t	183
5.39.2.6	file_nr	183
5.39.2.7	inode_nr	183
5.39.2.8	m_inode	183
5.39.2.9	proc_file	183
5.39.3	Function Documentation	183
5.39.3.1	__attribute__	183
5.39.4	Variable Documentation	184
5.39.4.1	i_create_ts	184
5.39.4.2	i_direct_pointer	184
5.39.4.3	i_double_indirect_pointer	184
5.39.4.4	i_mode	184
5.39.4.5	i_modify_ts	184
5.39.4.6	i_single_indirect_pointer	184
5.39.4.7	i_size	184
5.39.4.8	inode	184
5.39.4.9	name	184
5.40	src/kernel/include/assert.h File Reference	184
5.40.1	Detailed Description	184
5.40.2	Define Documentation	185
5.40.2.1	ASSERT	185
5.41	src/kernel/include/const.h File Reference	185
5.41.1	Detailed Description	185

---

5.41.2	Define Documentation	186
5.41.2.1	EOF	186
5.41.2.2	FALSE	186
5.41.2.3	FREQUENCY	186
5.41.2.4	NULL	186
5.41.2.5	TRUE	186
5.41.2.6	VGA_DISPLAY	186
5.42	src/kernel/include/debug.h File Reference	187
5.42.1	Detailed Description	187
5.42.2	Define Documentation	187
5.42.2.1	dprint_separator	187
5.42.2.2	dprintf	187
5.42.2.3	fs_dprintf	188
5.42.2.4	SHORTCUT_CTRL	188
5.42.2.5	SHORTCUT_CTRL_SUPER	188
5.42.2.6	SHORTCUT_SUPER	188
5.43	src/kernel/include/init.h File Reference	188
5.43.1	Detailed Description	190
5.43.2	Function Documentation	190
5.43.2.1	__attribute__	190
5.43.2.2	do_tests	190
5.43.2.3	fs_init	190
5.43.2.4	fs_shutdown	190
5.43.2.5	io_init	190
5.43.2.6	mm_init	191
5.43.2.7	mm_init2	191
5.43.2.8	panic	191
5.43.2.9	pm_init	191
5.43.3	Variable Documentation	191
5.43.3.1	addr	191
5.43.3.2	apm_table	192
5.43.3.3	boot_device	192
5.43.3.4	boot_loader_name	192
5.43.3.5	cmdline	192
5.43.3.6	config_table	192
5.43.3.7	drives_addr	192

5.43.3.8	drives_length	192
5.43.3.9	flags	192
5.43.3.10	g_mboot_ptr	192
5.43.3.11	mem_lower	192
5.43.3.12	mem_upper	192
5.43.3.13	mmap_addr	192
5.43.3.14	mmap_length	192
5.43.3.15	mods_addr	192
5.43.3.16	mods_count	192
5.43.3.17	num	192
5.43.3.18	shndx	193
5.43.3.19	size	193
5.43.3.20	vbe_control_info	193
5.43.3.21	vbe_interface_len	193
5.43.3.22	vbe_interface_off	193
5.43.3.23	vbe_interface_seg	193
5.43.3.24	vbe_mode	193
5.43.3.25	vbe_mode_info	193
5.44	src/kernel/include/limits.h File Reference	193
5.44.1	Detailed Description	193
5.44.2	Define Documentation	194
5.44.2.1	SINT16_MAX	194
5.44.2.2	SINT16_MIN	194
5.44.2.3	SINT32_MAX	194
5.44.2.4	SINT32_MIN	194
5.44.2.5	SINT8_MAX	194
5.44.2.6	SINT8_MIN	194
5.44.2.7	UINT16_MAX	194
5.44.2.8	UINT32_MAX	194
5.44.2.9	UINT8_MAX	194
5.45	src/kernel/include/ringbuffer.h File Reference	194
5.45.1	Detailed Description	195
5.45.2	Function Documentation	196
5.45.2.1	rf_alloc	196
5.45.2.2	rf_clear	196
5.45.2.3	rf_copy	196



---

5.45.2.4	<a href="#">rf_dump</a>	196
5.45.2.5	<a href="#">rf_free</a>	197
5.45.2.6	<a href="#">rf_getlength</a>	197
5.45.2.7	<a href="#">rf_isempty</a>	197
5.45.2.8	<a href="#">rf_isfull</a>	197
5.45.2.9	<a href="#">rf_read</a>	198
5.45.2.10	<a href="#">rf_write</a>	198
5.45.3	<a href="#">Variable Documentation</a>	198
5.45.3.1	<a href="#">__attribute__</a>	198
5.46	<a href="#">src/kernel/include/stdarg.h File Reference</a>	198
5.46.1	<a href="#">Detailed Description</a>	199
5.46.2	<a href="#">Define Documentation</a>	199
5.46.2.1	<a href="#">va_arg</a>	199
5.46.2.2	<a href="#">va_end</a>	199
5.46.2.3	<a href="#">va_start</a>	199
5.46.3	<a href="#">Typedef Documentation</a>	199
5.46.3.1	<a href="#">va_list</a>	199
5.47	<a href="#">src/kernel/include/stdio.h File Reference</a>	199
5.47.1	<a href="#">Detailed Description</a>	201
5.47.2	<a href="#">Enumeration Type Documentation</a>	201
5.47.2.1	<a href="#">colors</a>	201
5.47.3	<a href="#">Function Documentation</a>	202
5.47.3.1	<a href="#">cputchar</a>	202
5.47.3.2	<a href="#">cputs</a>	202
5.47.3.3	<a href="#">hd_read_sector</a>	202
5.47.3.4	<a href="#">hd_write_sector</a>	202
5.47.3.5	<a href="#">monitor_cputc</a>	202
5.47.3.6	<a href="#">monitor_cputs</a>	203
5.47.3.7	<a href="#">monitor_putc</a>	203
5.47.3.8	<a href="#">monitor_puthex</a>	203
5.47.3.9	<a href="#">monitor_puti</a>	203
5.47.3.10	<a href="#">monitor_puts</a>	204
5.47.3.11	<a href="#">monitor_scrolldown</a>	204
5.47.3.12	<a href="#">monitor_scrollup</a>	204
5.47.3.13	<a href="#">printf</a>	204
5.47.3.14	<a href="#">putchar</a>	204

---

5.47.3.15	puts	204
5.47.3.16	snprintf	205
5.47.3.17	vsnprintf	205
5.47.4	Variable Documentation	205
5.47.4.1	maxaddr	205
5.48	src/kernel/include/stdlib.h File Reference	205
5.48.1	Detailed Description	206
5.48.2	Function Documentation	206
5.48.2.1	calloc	206
5.48.2.2	callocn	206
5.48.2.3	free	206
5.48.2.4	free_memory	207
5.48.2.5	malloc	207
5.48.2.6	mallocn	207
5.48.2.7	mem_dump	207
5.48.2.8	rand	207
5.48.2.9	realloc	207
5.48.2.10	srand	208
5.49	src/kernel/include/string.h File Reference	208
5.49.1	Detailed Description	209
5.49.2	Function Documentation	209
5.49.2.1	atoi	209
5.49.2.2	bzero	210
5.49.2.3	isspace	210
5.49.2.4	itoa	210
5.49.2.5	memcpy	210
5.49.2.6	memmove	211
5.49.2.7	memset	211
5.49.2.8	strcat	212
5.49.2.9	strchr	212
5.49.2.10	strcmp	212
5.49.2.11	strcpy	212
5.49.2.12	strdup	213
5.49.2.13	strlen	213
5.49.2.14	strncat	213
5.49.2.15	strncpy	214

5.49.2.16	strreverse	214
5.49.2.17	strsep	214
5.49.2.18	strtol	215
5.50	src/kernel/include/types.h File Reference	216
5.50.1	Detailed Description	216
5.50.2	Define Documentation	216
5.50.2.1	MAX	216
5.50.2.2	MIN	217
5.50.3	Typedef Documentation	217
5.50.3.1	bool	217
5.50.3.2	float32	217
5.50.3.3	float64	217
5.50.3.4	sint16	217
5.50.3.5	sint32	217
5.50.3.6	sint8	217
5.50.3.7	size_t	217
5.50.3.8	time_t	217
5.50.3.9	uint16	217
5.50.3.10	uint32	217
5.50.3.11	uint8	217
5.51	src/kernel/include/util.h File Reference	217
5.51.1	Detailed Description	217
5.51.2	Function Documentation	218
5.51.2.1	sleep	218
5.51.2.2	sleep_ticks	218
5.52	src/kernel/init/main.c File Reference	218
5.52.1	Detailed Description	219
5.52.2	Function Documentation	219
5.52.2.1	main	219
5.52.2.2	panic	219
5.52.2.3	shell_main	220
5.52.3	Variable Documentation	220
5.52.3.1	end	220
5.52.3.2	g_mboot_ptr	220
5.52.3.3	start	220
5.53	src/kernel/init/tests.c File Reference	220

---

5.53.1 Detailed Description . . . . .	223
5.53.2 Define Documentation . . . . .	223
5.53.2.1 MBOOT_INFO . . . . .	223
5.53.3 Function Documentation . . . . .	223
5.53.3.1 assert_test . . . . .	223
5.53.3.2 atoi_test . . . . .	224
5.53.3.3 create_fs . . . . .	224
5.53.3.4 do_tests . . . . .	224
5.53.3.5 draw_test . . . . .	224
5.53.3.6 fgetch . . . . .	224
5.53.3.7 fgets . . . . .	224
5.53.3.8 fs_tests . . . . .	224
5.53.3.9 grubstruct_test . . . . .	224
5.53.3.10 hd_stressread_test . . . . .	225
5.53.3.11 hd_stresswrite_test . . . . .	225
5.53.3.12 hd_test . . . . .	225
5.53.3.13 hd_write_test . . . . .	225
5.53.3.14 isr_test . . . . .	225
5.53.3.15 make_snapshot . . . . .	225
5.53.3.16 malloc_test . . . . .	225
5.53.3.17 mm_pagefault_test . . . . .	225
5.53.3.18 new_fs . . . . .	225
5.53.3.19 nullptr_test . . . . .	225
5.53.3.20 p . . . . .	225
5.53.3.21 paralleleModellierung . . . . .	226
5.53.3.22 print_time . . . . .	226
5.53.3.23 printf_test . . . . .	226
5.53.3.24 ralph_wiggum . . . . .	226
5.53.3.25 reboot . . . . .	226
5.53.3.26 run_FS_tests . . . . .	226
5.53.3.27 sema_init . . . . .	226
5.53.3.28 sleep_test . . . . .	226
5.53.3.29 strings_test . . . . .	226
5.53.3.30 strsep_test . . . . .	227
5.53.3.31 syscall_test . . . . .	227
5.53.3.32 syscall_test_thread . . . . .	227

---

5.53.3.33	test_batch_files	227
5.53.3.34	threadA	227
5.53.3.35	threadA_test	227
5.53.3.36	threadB	227
5.53.3.37	threadB_test	227
5.53.3.38	threadC	227
5.53.3.39	threadC_test	228
5.53.3.40	threadConsumer	228
5.53.3.41	threadConsumer_test	228
5.53.3.42	threadD	228
5.53.3.43	threadD_test	228
5.53.3.44	threadFabrik	228
5.53.3.45	threadGeschaeft1	228
5.53.3.46	threadGeschaeft2	228
5.53.3.47	threadLastwagen1	228
5.53.3.48	threadLastwagen2	229
5.53.3.49	threadMitarbeiter	229
5.53.3.50	threadProducer	229
5.53.3.51	threadProducer_test	229
5.53.3.52	v	229
5.53.4	Variable Documentation	229
5.53.4.1	betten	229
5.53.4.2	elements	229
5.53.4.3	g1_access	229
5.53.4.4	g1_free	229
5.53.4.5	g1_full	229
5.53.4.6	g1betten	229
5.53.4.7	g2_access	230
5.53.4.8	g2b_free	230
5.53.4.9	g2b_full	230
5.53.4.10	g2betten	230
5.53.4.11	g2s_free	230
5.53.4.12	g2s_full	230
5.53.4.13	g2schraenke	230
5.53.4.14	lager_access	230
5.53.4.15	lager_betten	230

5.53.4.16	lager_free	230
5.53.4.17	lager_schraenke	230
5.53.4.18	mutex	230
5.53.4.19	rampe_access	230
5.53.4.20	rampe_betten	230
5.53.4.21	rampe_free	230
5.53.4.22	rampe_schraenke	230
5.53.4.23	rbetten	230
5.53.4.24	rschraenke	230
5.53.4.25	schraenke	230
5.53.4.26	sema_access	231
5.53.4.27	sema_free	231
5.53.4.28	sema_full	231
5.54	src/kernel/io/int_handler.h File Reference	231
5.54.1	Detailed Description	231
5.54.2	Function Documentation	231
5.54.2.1	hd_handler	231
5.54.2.2	kb_handler	232
5.54.2.3	timer_handler	232
5.55	src/kernel/io/int_idt.c File Reference	232
5.55.1	Detailed Description	233
5.55.2	Function Documentation	233
5.55.2.1	__attribute__	233
5.55.2.2	idt_fill_entry	234
5.55.2.3	idt_flush	234
5.55.2.4	idt_init	234
5.55.3	Variable Documentation	234
5.55.3.1	flags	234
5.55.3.2	high_offset	234
5.55.3.3	idt	234
5.55.3.4	idtp	234
5.55.3.5	low_offset	234
5.55.3.6	maxsize	235
5.55.3.7	selector	235
5.55.3.8	separator	235
5.55.3.9	start	235

---

5.56	src/kernel/io/int_irq.c File Reference	235
5.56.1	Detailed Description	236
5.56.2	Function Documentation	236
5.56.2.1	idt_fill_entry	236
5.56.2.2	irq0	236
5.56.2.3	irq1	237
5.56.2.4	irq10	237
5.56.2.5	irq11	237
5.56.2.6	irq12	237
5.56.2.7	irq13	237
5.56.2.8	irq14	237
5.56.2.9	irq15	237
5.56.2.10	irq2	237
5.56.2.11	irq3	237
5.56.2.12	irq4	237
5.56.2.13	irq5	237
5.56.2.14	irq6	238
5.56.2.15	irq7	238
5.56.2.16	irq8	238
5.56.2.17	irq9	238
5.56.2.18	irq_handler	238
5.56.2.19	irq_init	238
5.56.2.20	outb	238
5.56.2.21	pic_remap	238
5.56.2.22	reactivate_pic	239
5.56.3	Variable Documentation	239
5.56.3.1	hw_messages	239
5.57	src/kernel/io/int_isr.c File Reference	239
5.57.1	Detailed Description	241
5.57.2	Function Documentation	241
5.57.2.1	idt_fill_entry	241
5.57.2.2	incoming_syscall	241
5.57.2.3	isr0	241
5.57.2.4	isr1	241
5.57.2.5	isr10	241
5.57.2.6	isr11	242

---

5.57.2.7	isr12	242
5.57.2.8	isr13	242
5.57.2.9	isr14	242
5.57.2.10	isr15	242
5.57.2.11	isr16	242
5.57.2.12	isr17	242
5.57.2.13	isr18	242
5.57.2.14	isr19	242
5.57.2.15	isr2	242
5.57.2.16	isr20	242
5.57.2.17	isr21	243
5.57.2.18	isr22	243
5.57.2.19	isr23	243
5.57.2.20	isr24	243
5.57.2.21	isr25	243
5.57.2.22	isr26	243
5.57.2.23	isr27	243
5.57.2.24	isr28	243
5.57.2.25	isr29	243
5.57.2.26	isr3	243
5.57.2.27	isr30	243
5.57.2.28	isr31	244
5.57.2.29	isr4	244
5.57.2.30	isr5	244
5.57.2.31	isr6	244
5.57.2.32	isr7	244
5.57.2.33	isr8	244
5.57.2.34	isr9	244
5.57.2.35	isr_handler	244
5.57.2.36	isr_init	244
5.57.3	Variable Documentation	245
5.57.3.1	ex_messages	245
5.58	src/kernel/io/io.h File Reference	245
5.58.1	Detailed Description	246
5.58.2	Function Documentation	246
5.58.2.1	add_shortcut	246



---

5.58.2.2	clear_interrupts	247
5.58.2.3	end_beep	247
5.58.2.4	get_ticks	247
5.58.2.5	halt	247
5.58.2.6	hd_init	247
5.58.2.7	idt_init	247
5.58.2.8	inb	247
5.58.2.9	irq_init	248
5.58.2.10	isr_init	248
5.58.2.11	keyboard_init	248
5.58.2.12	make_syscall	248
5.58.2.13	monitor_init	248
5.58.2.14	monitor_invert	248
5.58.2.15	mutex_lock	248
5.58.2.16	mutex_unlock	248
5.58.2.17	outb	248
5.58.2.18	reactivate_pic	249
5.58.2.19	repinsw	249
5.58.2.20	repoutsw	249
5.58.2.21	set_disp	249
5.58.2.22	set_interrupts	249
5.58.2.23	start_beep	249
5.58.2.24	timer_init	249
5.59	src/kernel/io/io_harddisk.c File Reference	250
5.59.1	Detailed Description	251
5.59.2	Function Documentation	251
5.59.2.1	dump_hd1	251
5.59.2.2	get_hdsize	251
5.59.2.3	hd_handler	251
5.59.2.4	hd_init	252
5.59.2.5	hd_read_sector	252
5.59.2.6	hd_write_sector	252
5.59.2.7	itoaddr	252
5.59.2.8	select_masterdrive	253
5.59.2.9	wait_on_hd_interrupt	253
5.59.3	Variable Documentation	253

5.59.3.1	hd1	253
5.59.3.2	hd_interrupt	253
5.60	src/kernel/io/io_harddisk.h File Reference	253
5.60.1	Detailed Description	255
5.60.2	Define Documentation	255
5.60.2.1	HDALTBASE	255
5.60.2.2	HDALTREG_ADDR	256
5.60.2.3	HDALTREG_STAT	256
5.60.2.4	HDBASE	256
5.60.2.5	HDCMD_EXEC_DRIVE_DIAG	256
5.60.2.6	HDCMD_FLUSH_CACHE	256
5.60.2.7	HDCMD_IDENTIFY_DEVICE	256
5.60.2.8	HDCMD_READ	256
5.60.2.9	HDCMD_WRITE	256
5.60.2.10	HDREG_CMD	256
5.60.2.11	HDREG_COUNT	256
5.60.2.12	HDREG_CYL_HIGH	256
5.60.2.13	HDREG_CYL_LOW	256
5.60.2.14	HDREG_DATA	257
5.60.2.15	HDREG_DRIVE	257
5.60.2.16	HDREG_ERR	257
5.60.2.17	HDREG_SEC	257
5.60.2.18	HDREG_STAT	257
5.60.2.19	HDSTATE_FLOATINGBUS	257
5.60.2.20	HDSTATE_NOTREADY	257
5.60.2.21	MASTERDRIVE	257
5.60.2.22	SLAVEDRIVE	257
5.60.3	Function Documentation	257
5.60.3.1	__attribute__	257
5.60.3.2	get_hdsiz	257
5.60.4	Variable Documentation	258
5.60.4.1	__attribute__	258
5.60.4.2	apparent_capacity	260
5.60.4.3	apparent_cyl	260
5.60.4.4	apparent_head	260
5.60.4.5	apparent_sector_per_track	260

---

5.60.4.6	buffer_size	260
5.60.4.7	buffer_type	260
5.60.4.8	bytes_per_sector	260
5.60.4.9	bytes_per_track	260
5.60.4.10	config_word	260
5.60.4.11	dw_io_flg	260
5.60.4.12	firmware	260
5.60.4.13	lba_dma_flg	260
5.60.4.14	manufacturer1	260
5.60.4.15	manufacturer2	260
5.60.4.16	mode_multi_dma	260
5.60.4.17	mode_single_dma	260
5.60.4.18	num_cyl	260
5.60.4.19	num_ecc_bytes	260
5.60.4.20	num_head	260
5.60.4.21	num_lba_sectors	260
5.60.4.22	reserved1	260
5.60.4.23	reserved2	260
5.60.4.24	reserved3	260
5.60.4.25	reserved4	260
5.60.4.26	reserved5	260
5.60.4.27	rw_multiple_flg	260
5.60.4.28	sector_per_track	260
5.60.4.29	sectors_per_int	260
5.60.4.30	serial	260
5.60.4.31	timingmode_dma	260
5.60.4.32	timingmode_pio	260
5.60.4.33	type	260
5.61	src/kernel/io/io_keyboard.c File Reference	260
5.61.1	Detailed Description	261
5.61.2	Define Documentation	262
5.61.2.1	SHORTCUTS_ARRAY_SIZE	262
5.61.3	Function Documentation	262
5.61.3.1	add_shortcut	262
5.61.3.2	kb_handler	262
5.61.4	Variable Documentation	263

---

5.61.4.1	echo	263
5.61.4.2	keyboard_state	263
5.61.4.3	shcut_num	263
5.61.4.4	shortcuts	263
5.62	src/kernel/io/io_keyboard.h File Reference	263
5.62.1	Detailed Description	264
5.62.2	Define Documentation	265
5.62.2.1	ALT	265
5.62.2.2	CTRL	265
5.62.2.3	CURSOR_DOWN	265
5.62.2.4	CURSOR_LEFT	265
5.62.2.5	CURSOR_RIGHT	265
5.62.2.6	CURSOR_UP	265
5.62.2.7	ENTER	265
5.62.2.8	ESCAPE	265
5.62.2.9	KB_PORT	265
5.62.2.10	KEY_PRESSED	265
5.62.2.11	KEY_RELEASED	265
5.62.2.12	LSHIFT	265
5.62.2.13	RSHIFT	265
5.62.2.14	SCROLL_DOWN	265
5.62.2.15	SCROLL_UP	266
5.62.2.16	SUPER	266
5.62.3	Function Documentation	266
5.62.3.1	cursor_move	266
5.62.4	Variable Documentation	266
5.62.4.1	alt	266
5.62.4.2	ctrl	266
5.62.4.3	kb_alt_map	266
5.62.4.4	kb_map	266
5.62.4.5	kb_shift_map	266
5.62.4.6	shift	266
5.62.4.7	super_button	266
5.63	src/kernel/io/io_main.c File Reference	267
5.63.1	Detailed Description	267
5.63.2	Function Documentation	267

5.63.2.1	io_init	267
5.63.3	Variable Documentation	268
5.63.3.1	keyboard_state	268
5.64	src/kernel/io/io_monitor.c File Reference	268
5.64.1	Detailed Description	269
5.64.2	Function Documentation	269
5.64.2.1	monitor_cputc	269
5.64.2.2	monitor_cputs	269
5.64.2.3	monitor_invert	269
5.64.2.4	monitor_putc	270
5.64.2.5	monitor_puthex	270
5.64.2.6	monitor_puti	270
5.64.2.7	monitor_puts	270
5.64.2.8	set_disp	270
5.65	src/kernel/io/io_rtc.c File Reference	271
5.65.1	Detailed Description	271
5.65.2	Function Documentation	272
5.65.2.1	bcd2bin	272
5.65.2.2	calculate_weekday	272
5.65.2.3	rtc_init	272
5.65.2.4	rtc_update	272
5.65.2.5	time2str	272
5.66	src/kernel/io/io_rtc.h File Reference	273
5.66.1	Detailed Description	274
5.66.2	Define Documentation	274
5.66.2.1	RTC_ADDR	274
5.66.2.2	RTC_ALERTHOUR	274
5.66.2.3	RTC_ALERTMIN	274
5.66.2.4	RTC_ALERTSEC	274
5.66.2.5	RTC_CENTURY	274
5.66.2.6	RTC_DATA	274
5.66.2.7	RTC_DAY	275
5.66.2.8	RTC_HOUR	275
5.66.2.9	RTC_MIN	275
5.66.2.10	RTC_MONTH	275
5.66.2.11	RTC_SEC	275

5.66.2.12	RTC_STATA	275
5.66.2.13	RTC_STATB	275
5.66.2.14	RTC_STATC	275
5.66.2.15	RTC_STATD	275
5.66.2.16	RTC_WEEKDAY	275
5.66.2.17	RTC_YEAR	275
5.66.3	Function Documentation	275
5.66.3.1	bcd2bin	275
5.66.3.2	rtc_init	276
5.66.3.3	rtc_update	276
5.66.3.4	time2str	276
5.66.4	Variable Documentation	276
5.66.4.1	time	276
5.67	src/kernel/io/io_sound.c File Reference	276
5.67.1	Detailed Description	277
5.67.2	Function Documentation	277
5.67.2.1	end_beep	277
5.67.2.2	start_beep	277
5.67.3	Variable Documentation	277
5.67.3.1	note_names	277
5.67.3.2	notes	278
5.68	src/kernel/io/io_sound.h File Reference	278
5.68.1	Detailed Description	281
5.68.2	Define Documentation	281
5.68.2.1	IO_SPEAKER_PORT	281
5.68.2.2	NOTE_A	281
5.68.2.3	NOTE_A0	281
5.68.2.4	NOTE_A1	281
5.68.2.5	NOTE_A2	281
5.68.2.6	NOTE_A3	281
5.68.2.7	NOTE_A4	281
5.68.2.8	NOTE_A5	281
5.68.2.9	NOTE_A6	281
5.68.2.10	NOTE_A7	281
5.68.2.11	NOTE_ASH	281
5.68.2.12	NOTE_ASH0	282

---

5.68.2.13 NOTE_ASH1 . . . . .	282
5.68.2.14 NOTE_ASH2 . . . . .	282
5.68.2.15 NOTE_ASH3 . . . . .	282
5.68.2.16 NOTE_ASH4 . . . . .	282
5.68.2.17 NOTE_ASH5 . . . . .	282
5.68.2.18 NOTE_ASH6 . . . . .	282
5.68.2.19 NOTE_ASH7 . . . . .	282
5.68.2.20 NOTE_B . . . . .	282
5.68.2.21 NOTE_B0 . . . . .	282
5.68.2.22 NOTE_B1 . . . . .	282
5.68.2.23 NOTE_B2 . . . . .	282
5.68.2.24 NOTE_B3 . . . . .	282
5.68.2.25 NOTE_B4 . . . . .	282
5.68.2.26 NOTE_B5 . . . . .	282
5.68.2.27 NOTE_B6 . . . . .	282
5.68.2.28 NOTE_B7 . . . . .	282
5.68.2.29 NOTE_C . . . . .	282
5.68.2.30 NOTE_C0 . . . . .	283
5.68.2.31 NOTE_C1 . . . . .	283
5.68.2.32 NOTE_C2 . . . . .	283
5.68.2.33 NOTE_C3 . . . . .	283
5.68.2.34 NOTE_C4 . . . . .	283
5.68.2.35 NOTE_C5 . . . . .	283
5.68.2.36 NOTE_C6 . . . . .	283
5.68.2.37 NOTE_C7 . . . . .	283
5.68.2.38 NOTE_CSH . . . . .	283
5.68.2.39 NOTE_CSH0 . . . . .	283
5.68.2.40 NOTE_CSH1 . . . . .	283
5.68.2.41 NOTE_CSH2 . . . . .	283
5.68.2.42 NOTE_CSH3 . . . . .	283
5.68.2.43 NOTE_CSH4 . . . . .	283
5.68.2.44 NOTE_CSH5 . . . . .	283
5.68.2.45 NOTE_CSH6 . . . . .	283
5.68.2.46 NOTE_CSH7 . . . . .	283
5.68.2.47 NOTE_D . . . . .	283
5.68.2.48 NOTE_D0 . . . . .	284

---

5.68.2.49 NOTE_D1 . . . . .	284
5.68.2.50 NOTE_D2 . . . . .	284
5.68.2.51 NOTE_D3 . . . . .	284
5.68.2.52 NOTE_D4 . . . . .	284
5.68.2.53 NOTE_D5 . . . . .	284
5.68.2.54 NOTE_D6 . . . . .	284
5.68.2.55 NOTE_D7 . . . . .	284
5.68.2.56 NOTE_DSH . . . . .	284
5.68.2.57 NOTE_DSH0 . . . . .	284
5.68.2.58 NOTE_DSH1 . . . . .	284
5.68.2.59 NOTE_DSH2 . . . . .	284
5.68.2.60 NOTE_DSH3 . . . . .	284
5.68.2.61 NOTE_DSH4 . . . . .	284
5.68.2.62 NOTE_DSH5 . . . . .	284
5.68.2.63 NOTE_DSH6 . . . . .	284
5.68.2.64 NOTE_DSH7 . . . . .	284
5.68.2.65 NOTE_E . . . . .	284
5.68.2.66 NOTE_E0 . . . . .	285
5.68.2.67 NOTE_E1 . . . . .	285
5.68.2.68 NOTE_E2 . . . . .	285
5.68.2.69 NOTE_E3 . . . . .	285
5.68.2.70 NOTE_E4 . . . . .	285
5.68.2.71 NOTE_E5 . . . . .	285
5.68.2.72 NOTE_E6 . . . . .	285
5.68.2.73 NOTE_E7 . . . . .	285
5.68.2.74 NOTE_F . . . . .	285
5.68.2.75 NOTE_F0 . . . . .	285
5.68.2.76 NOTE_F1 . . . . .	285
5.68.2.77 NOTE_F2 . . . . .	285
5.68.2.78 NOTE_F3 . . . . .	285
5.68.2.79 NOTE_F4 . . . . .	285
5.68.2.80 NOTE_F5 . . . . .	285
5.68.2.81 NOTE_F6 . . . . .	285
5.68.2.82 NOTE_F7 . . . . .	285
5.68.2.83 NOTE_FSH . . . . .	285
5.68.2.84 NOTE_FSH0 . . . . .	286



---

5.68.2.85	NOTE_FSH1	286
5.68.2.86	NOTE_FSH2	286
5.68.2.87	NOTE_FSH3	286
5.68.2.88	NOTE_FSH4	286
5.68.2.89	NOTE_FSH5	286
5.68.2.90	NOTE_FSH6	286
5.68.2.91	NOTE_FSH7	286
5.68.2.92	NOTE_G	286
5.68.2.93	NOTE_G0	286
5.68.2.94	NOTE_G1	286
5.68.2.95	NOTE_G2	286
5.68.2.96	NOTE_G3	286
5.68.2.97	NOTE_G4	286
5.68.2.98	NOTE_G5	286
5.68.2.99	NOTE_G6	286
5.68.2.100	NOTE_G7	286
5.68.2.101	NOTE_GSH	286
5.68.2.102	NOTE_GSH0	287
5.68.2.103	NOTE_GSH1	287
5.68.2.104	NOTE_GSH2	287
5.68.2.105	NOTE_GSH3	287
5.68.2.106	NOTE_GSH4	287
5.68.2.107	NOTE_GSH5	287
5.68.2.108	NOTE_GSH6	287
5.68.2.109	NOTE_GSH7	287
5.68.3	Variable Documentation	287
5.68.3.1	note_names	287
5.68.3.2	notes	287
5.69	src/kernel/io/io_timer.c File Reference	287
5.69.1	Detailed Description	288
5.69.2	Function Documentation	288
5.69.2.1	get_ticks	288
5.69.2.2	sleep	288
5.69.2.3	sleep_ticks	288
5.69.2.4	timer_handler	289
5.69.2.5	timer_init	289

5.70	src/kernel/io/io_timer.h File Reference	289
5.70.1	Detailed Description	289
5.70.2	Define Documentation	289
5.70.2.1	PIT_CONTROL	289
5.70.2.2	PIT_COUNTER0	290
5.70.2.3	PIT_COUNTER1	290
5.70.2.4	PIT_COUNTER2	290
5.70.2.5	PIT_INIT_CMD	290
5.70.2.6	PIT_SOUND_CMD	290
5.71	src/kernel/io/io_virtual.c File Reference	290
5.71.1	Detailed Description	291
5.71.2	Function Documentation	292
5.71.2.1	free_virt_monitor	292
5.71.2.2	get_color_tag	292
5.71.2.3	new_virt_monitor	292
5.71.2.4	update_virt_monitor	292
5.71.2.5	virt_cursor_move	293
5.71.2.6	virt_monitor_cputc	293
5.71.2.7	virt_monitor_cputs	293
5.71.2.8	virt_monitor_invert	294
5.71.2.9	virt_monitor_putc	294
5.71.2.10	virt_monitor_puts	294
5.71.2.11	virt_monitor_scrolldown	294
5.71.2.12	virt_monitor_scrollup	295
5.72	src/kernel/io/io_virtual.h File Reference	295
5.72.1	Detailed Description	296
5.72.2	Define Documentation	297
5.72.2.1	CURSOR_OFFSET	297
5.72.2.2	LINE_WIDTH	297
5.72.2.3	MONITOR_START	297
5.72.2.4	VIRTUAL_MONITOR_SIZE	297
5.72.2.5	VMONITOR_HEIGHT	297
5.72.3	Function Documentation	297
5.72.3.1	free_virt_monitor	297
5.72.3.2	get_active_virt_monitor	298
5.72.3.3	get_active_virt_monitor_name	298

---

5.72.3.4	<a href="#">init_vmonitors</a>	298
5.72.3.5	<a href="#">new_virt_monitor</a>	298
5.72.3.6	<a href="#">start_vmonitor</a>	298
5.72.3.7	<a href="#">switch_monitor_down</a>	299
5.72.3.8	<a href="#">switch_monitor_up</a>	299
5.72.3.9	<a href="#">update_virt_monitor</a>	299
5.72.3.10	<a href="#">virt_cursor_move</a>	299
5.72.3.11	<a href="#">virt_monitor_cputc</a>	299
5.72.3.12	<a href="#">virt_monitor_cputs</a>	300
5.72.3.13	<a href="#">virt_monitor_invert</a>	300
5.72.3.14	<a href="#">virt_monitor_putc</a>	300
5.72.3.15	<a href="#">virt_monitor_puts</a>	300
5.72.3.16	<a href="#">virt_monitor_scrolldown</a>	301
5.72.3.17	<a href="#">virt_monitor_scrollup</a>	301
5.72.3.18	<a href="#">virt_printf</a>	301
5.72.4	<a href="#">Variable Documentation</a>	301
5.72.4.1	<a href="#">active_monitor</a>	301
5.72.4.2	<a href="#">maxvmonitor</a>	301
5.72.4.3	<a href="#">vmonitors</a>	301
5.73	<a href="#">src/kernel/io/io_virtual_monitors.c File Reference</a>	302
5.73.1	<a href="#">Detailed Description</a>	302
5.73.2	<a href="#">Function Documentation</a>	303
5.73.2.1	<a href="#">get_active_virt_monitor</a>	303
5.73.2.2	<a href="#">get_active_virt_monitor_name</a>	303
5.73.2.3	<a href="#">init_vmonitors</a>	303
5.73.2.4	<a href="#">start_vmonitor</a>	303
5.73.2.5	<a href="#">switch_monitor_down</a>	304
5.73.2.6	<a href="#">switch_monitor_up</a>	304
5.73.3	<a href="#">Variable Documentation</a>	304
5.73.3.1	<a href="#">active_monitor</a>	304
5.73.3.2	<a href="#">maxvmonitor</a>	304
5.73.3.3	<a href="#">num_vmonitor_limit</a>	304
5.73.3.4	<a href="#">vmonitors</a>	304
5.74	<a href="#">src/kernel/lib/ringbuffer.c File Reference</a>	304
5.74.1	<a href="#">Detailed Description</a>	305
5.74.2	<a href="#">Function Documentation</a>	306

---

5.74.2.1	rf_alloc	306
5.74.2.2	rf_clear	306
5.74.2.3	rf_copy	306
5.74.2.4	rf_dump	307
5.74.2.5	rf_free	307
5.74.2.6	rf_getlength	307
5.74.2.7	rf_isempty	307
5.74.2.8	rf_isfull	308
5.74.2.9	rf_read	308
5.74.2.10	rf_write	308
5.75	src/kernel/lib/stdio.c File Reference	309
5.75.1	Detailed Description	309
5.75.2	Define Documentation	310
5.75.2.1	SELECT_VMONITOR	310
5.75.3	Function Documentation	310
5.75.3.1	cputchar	310
5.75.3.2	cputs	310
5.75.3.3	printf	310
5.75.3.4	putchar	310
5.75.3.5	puts	311
5.75.3.6	snprintf	311
5.75.3.7	vsprintf	311
5.76	src/kernel/lib/stdlib.c File Reference	311
5.76.1	Detailed Description	312
5.76.2	Function Documentation	312
5.76.2.1	calloc	312
5.76.2.2	callocn	313
5.76.2.3	free	313
5.76.2.4	free_memory	313
5.76.2.5	malloc	313
5.76.2.6	mallocn	314
5.76.2.7	mem_dump	314
5.76.2.8	rand	314
5.76.2.9	realloc	314
5.76.2.10	srand	315
5.77	src/kernel/lib/string.c File Reference	315

---

5.77.1	Detailed Description	316
5.77.2	Function Documentation	316
5.77.2.1	atoi	316
5.77.2.2	bzero	317
5.77.2.3	isspace	317
5.77.2.4	itoa	317
5.77.2.5	memcpy	318
5.77.2.6	memmove	318
5.77.2.7	memset	319
5.77.2.8	streat	319
5.77.2.9	strchr	319
5.77.2.10	strcmp	320
5.77.2.11	strcpy	320
5.77.2.12	strdup	320
5.77.2.13	strlen	321
5.77.2.14	strncat	321
5.77.2.15	strncpy	321
5.77.2.16	strreverse	321
5.77.2.17	strsep	322
5.77.2.18	strtol	322
5.78	src/kernel/mm/mm.h File Reference	323
5.78.1	Detailed Description	325
5.78.2	Define Documentation	325
5.78.2.1	HEAP_EXPAND_STEP_SIZE	325
5.78.2.2	HEAP_MIN_SIZE	325
5.78.2.3	KHEAP_INITIAL_SIZE	325
5.78.2.4	KHEAP_START	325
5.78.3	Typedef Documentation	325
5.78.3.1	mm_header	325
5.78.4	Function Documentation	325
5.78.4.1	__attribute__	325
5.78.4.2	create_heap	326
5.78.4.3	gdt_init	326
5.78.4.4	heap_free	326
5.78.4.5	heap_mallocn	326
5.78.4.6	heap_mem_dump	327

---

5.78.4.7	mm_init	327
5.78.4.8	mm_init_output	327
5.78.4.9	mm_move_block	327
5.78.5	Variable Documentation	327
5.78.5.1	access	327
5.78.5.2	base	327
5.78.5.3	base_high	327
5.78.5.4	base_low	327
5.78.5.5	base_middle	327
5.78.5.6	gdt	327
5.78.5.7	gp	328
5.78.5.8	granularity	328
5.78.5.9	kernel_heap	328
5.78.5.10	limit	328
5.78.5.11	limit_low	328
5.79	src/kernel/mm/mm_bitmap.c File Reference	328
5.79.1	Detailed Description	329
5.79.2	Function Documentation	329
5.79.2.1	alloc_frame	329
5.79.2.2	bitset_test	329
5.79.2.3	clear_frame	329
5.79.2.4	first_free_frame	330
5.79.2.5	free_frame	330
5.79.2.6	set_frame	330
5.79.2.7	test_frame	330
5.80	src/kernel/mm/mm_bitmap.h File Reference	331
5.80.1	Detailed Description	331
5.80.2	Function Documentation	331
5.80.2.1	alloc_frame	331
5.80.2.2	bitset_test	332
5.80.2.3	clear_frame	332
5.80.2.4	first_free_frame	332
5.80.2.5	free_frame	332
5.80.2.6	set_frame	333
5.80.2.7	test_frame	333
5.81	src/kernel/mm/mm_gdt.c File Reference	333

---

5.81.1	Detailed Description	333
5.81.2	Function Documentation	334
5.81.2.1	gdt_add_entry	334
5.81.2.2	gdt_flush	334
5.81.2.3	gdt_init	334
5.82	src/kernel/mm/mm_heap.c File Reference	335
5.82.1	Detailed Description	336
5.82.2	Function Documentation	336
5.82.2.1	_printf	336
5.82.2.2	create_heap	336
5.82.2.3	heap_contract	336
5.82.2.4	heap_expand	337
5.82.2.5	heap_free	337
5.82.2.6	heap_get_size	337
5.82.2.7	heap_mallocn	338
5.82.2.8	heap_mem_dump	338
5.82.2.9	heap_setup_block	338
5.83	src/kernel/mm/mm_main.c File Reference	338
5.83.1	Detailed Description	339
5.83.2	Function Documentation	340
5.83.2.1	get_page	340
5.83.2.2	kmalloc	340
5.83.2.3	mm_init	340
5.83.2.4	mm_init_output	340
5.83.2.5	switch_page_dir	341
5.84	src/kernel/mm/mm_paging.h File Reference	341
5.84.1	Detailed Description	342
5.84.2	Typedef Documentation	342
5.84.2.1	page_directory_t	342
5.84.2.2	page_t	342
5.84.2.3	page_table_t	342
5.84.3	Function Documentation	342
5.84.3.1	get_page	342
5.84.3.2	kmalloc	342
5.84.3.3	page_fault	343
5.84.3.4	read_from_cr0	343

---

5.84.3.5	read_from_cr3	343
5.84.3.6	switch_page_dir	343
5.84.3.7	write_to_cr0	343
5.84.3.8	write_to_cr3	343
5.84.4	Variable Documentation	343
5.84.4.1	current_dir	343
5.84.4.2	frames	343
5.84.4.3	kernel_dir	343
5.84.4.4	nframes	344
5.84.4.5	placement_addr	344
5.85	src/kernel/pm/dev_brainfuck.c File Reference	344
5.85.1	Detailed Description	344
5.85.2	Function Documentation	345
5.85.2.1	dev_brainfuck_close	345
5.85.2.2	dev_brainfuck_open	345
5.85.2.3	dev_brainfuck_read	345
5.85.2.4	dev_brainfuck_seek	345
5.85.2.5	dev_brainfuck_write	345
5.85.3	Variable Documentation	345
5.85.3.1	dev_brainfuck	345
5.86	src/kernel/pm/dev_clock.c File Reference	345
5.86.1	Detailed Description	346
5.86.2	Function Documentation	346
5.86.2.1	dev_clock_close	346
5.86.2.2	dev_clock_open	346
5.86.2.3	dev_clock_read	346
5.86.2.4	dev_clock_seek	346
5.86.2.5	dev_clock_write	346
5.86.3	Variable Documentation	346
5.86.3.1	dev_clock	346
5.87	src/kernel/pm/dev_framebuffer.c File Reference	347
5.87.1	Detailed Description	347
5.87.2	Function Documentation	348
5.87.2.1	dev_framebuffer_close	348
5.87.2.2	dev_framebuffer_open	348
5.87.2.3	dev_framebuffer_read	348



---

5.87.2.4	<a href="#">dev_framebuffer_seek</a>	348
5.87.2.5	<a href="#">dev_framebuffer_write</a>	348
5.87.3	<a href="#">Variable Documentation</a>	348
5.87.3.1	<a href="#">dev_framebuffer</a>	348
5.87.3.2	<a href="#">dev_framebuffer_count</a>	348
5.88	<a href="#">src/kernel/pm/dev_keyboard.c File Reference</a>	348
5.88.1	<a href="#">Detailed Description</a>	349
5.88.2	<a href="#">Function Documentation</a>	349
5.88.2.1	<a href="#">dev_keyboard_close</a>	349
5.88.2.2	<a href="#">dev_keyboard_open</a>	349
5.88.2.3	<a href="#">dev_keyboard_read</a>	349
5.88.2.4	<a href="#">dev_keyboard_seek</a>	350
5.88.2.5	<a href="#">dev_keyboard_write</a>	350
5.88.3	<a href="#">Variable Documentation</a>	350
5.88.3.1	<a href="#">dev_keyboard</a>	350
5.88.3.2	<a href="#">keyboard_state</a>	350
5.89	<a href="#">src/kernel/pm/dev_null.c File Reference</a>	350
5.89.1	<a href="#">Detailed Description</a>	351
5.89.2	<a href="#">Function Documentation</a>	351
5.89.2.1	<a href="#">dev_null_close</a>	351
5.89.2.2	<a href="#">dev_null_open</a>	351
5.89.2.3	<a href="#">dev_null_read</a>	351
5.89.2.4	<a href="#">dev_null_seek</a>	351
5.89.2.5	<a href="#">dev_null_write</a>	351
5.89.3	<a href="#">Variable Documentation</a>	351
5.89.3.1	<a href="#">dev_null</a>	351
5.90	<a href="#">src/kernel/pm/dev_stdin.c File Reference</a>	352
5.90.1	<a href="#">Detailed Description</a>	352
5.90.2	<a href="#">Function Documentation</a>	353
5.90.2.1	<a href="#">dev_stdin_close</a>	353
5.90.2.2	<a href="#">dev_stdin_open</a>	353
5.90.2.3	<a href="#">dev_stdin_read</a>	353
5.90.2.4	<a href="#">dev_stdin_seek</a>	353
5.90.2.5	<a href="#">dev_stdin_write</a>	353
5.90.3	<a href="#">Variable Documentation</a>	353
5.90.3.1	<a href="#">dev_stdin</a>	353

---

5.91	src/kernel/pm/dev_stdout.c File Reference	353
5.91.1	Detailed Description	354
5.91.2	Function Documentation	354
5.91.2.1	dev_stdout_close	354
5.91.2.2	dev_stdout_open	354
5.91.2.3	dev_stdout_read	354
5.91.2.4	dev_stdout_seek	354
5.91.2.5	dev_stdout_write	354
5.91.3	Variable Documentation	354
5.91.3.1	dev_stdout	354
5.92	src/kernel/pm/pm_devices.c File Reference	355
5.92.1	Detailed Description	355
5.92.2	Function Documentation	356
5.92.2.1	pm_fd2device	356
5.92.2.2	pm_name2device	356
5.92.2.3	pm_register_device	356
5.92.3	Variable Documentation	357
5.92.3.1	devices_head	357
5.93	src/kernel/pm/pm_devices.h File Reference	357
5.93.1	Detailed Description	358
5.93.2	Define Documentation	358
5.93.2.1	MAX_DEVICES	358
5.93.3	Typedef Documentation	358
5.93.3.1	dev_close_func	358
5.93.3.2	dev_open_func	358
5.93.3.3	dev_read_func	358
5.93.3.4	dev_seek_func	358
5.93.3.5	dev_write_func	358
5.93.3.6	device_t	358
5.93.4	Function Documentation	358
5.93.4.1	pm_fd2device	358
5.93.4.2	pm_name2device	359
5.93.4.3	pm_register_device	359
5.94	src/kernel/pm/pm_input.c File Reference	359
5.94.1	Detailed Description	360
5.94.2	Function Documentation	360

---

5.94.2.1	pm_handle_input	360
5.95	src/kernel/pm/pm_input.h File Reference	360
5.95.1	Detailed Description	361
5.95.2	Function Documentation	361
5.95.2.1	pm_handle_input	361
5.96	src/kernel/pm/pm_main.c File Reference	361
5.96.1	Detailed Description	363
5.96.2	Function Documentation	363
5.96.2.1	aprintf	363
5.96.2.2	getpid	364
5.96.2.3	pm_create_thread	364
5.96.2.4	pm_destroy_thread	364
5.96.2.5	pm_dump	365
5.96.2.6	pm_get_proc	365
5.96.2.7	pm_init	365
5.96.2.8	pm_kill_proc	365
5.96.2.9	pm_schedule	366
5.96.2.10	pm_set_focus_proc	366
5.96.2.11	pm_set_thread_priority	366
5.96.3	Variable Documentation	366
5.96.3.1	active_proc	366
5.96.3.2	dev_brainfuck	367
5.96.3.3	dev_clock	367
5.96.3.4	dev_framebuffer	367
5.96.3.5	dev_keyboard	367
5.96.3.6	dev_null	367
5.96.3.7	dev_stdin	367
5.96.3.8	dev_stdout	367
5.96.3.9	focus_proc	367
5.96.3.10	kernel_proc	367
5.96.3.11	next_pid	367
5.96.3.12	procs_head	367
5.97	src/kernel/pm/pm_main.h File Reference	367
5.97.1	Detailed Description	369
5.97.2	Define Documentation	369
5.97.2.1	PSTATE_ALIVE	369

5.97.2.2	PSTATE_DEAD	370
5.97.2.3	PSTATE_STDINSLEEP	370
5.97.2.4	STDIN_QUEUE_SIZE	370
5.97.3	Typedef Documentation	370
5.97.3.1	process_t	370
5.97.4	Function Documentation	370
5.97.4.1	_syscall	370
5.97.4.2	getpid	370
5.97.4.3	pm_create_thread	370
5.97.4.4	pm_destroy_thread	371
5.97.4.5	pm_get_proc	371
5.97.4.6	pm_init	372
5.97.4.7	pm_kill_proc	372
5.97.4.8	pm_schedule	372
5.97.4.9	pm_set_focus_proc	372
5.97.4.10	pm_set_thread_priority	373
5.97.5	Variable Documentation	373
5.97.5.1	active_proc	373
5.97.5.2	focus_proc	373
5.97.5.3	kernel_proc	373
5.97.5.4	procs_head	373
5.98	src/kernel/pm/pm_syscalls.c File Reference	373
5.98.1	Detailed Description	375
5.98.2	Define Documentation	375
5.98.2.1	SYSCALL_TRACE	375
5.98.3	Function Documentation	376
5.98.3.1	pm_syscall	376
5.98.3.2	sys_close	376
5.98.3.3	sys_exit	376
5.98.3.4	sys_free	376
5.98.3.5	sys_getpid	376
5.98.3.6	sys_kill	377
5.98.3.7	sys_log	377
5.98.3.8	sys_malloc	377
5.98.3.9	sys_open	377
5.98.3.10	sys_read	377

---

5.98.3.11	sys_seek	377
5.98.3.12	sys_stat	377
5.98.3.13	sys_unlink	378
5.98.3.14	sys_write	378
5.98.4	Variable Documentation	378
5.98.4.1	syscall_table	378
5.99	src/kernel/pm/pm_syscalls.h File Reference	378
5.99.1	Detailed Description	380
5.99.2	Typedef Documentation	380
5.99.2.1	syscall_handler	380
5.99.3	Function Documentation	380
5.99.3.1	sys_close	380
5.99.3.2	sys_exit	380
5.99.3.3	sys_free	381
5.99.3.4	sys_getpid	381
5.99.3.5	sys_kill	381
5.99.3.6	sys_log	381
5.99.3.7	sys_malloc	381
5.99.3.8	sys_open	381
5.99.3.9	sys_read	381
5.99.3.10	sys_seek	382
5.99.3.11	sys_stat	382
5.99.3.12	sys_unlink	382
5.99.3.13	sys_write	382
5.99.4	Variable Documentation	382
5.99.4.1	syscall_table	382
5.100	src/kernel/pm/syscalls_cli.c File Reference	382
5.100.1	Detailed Description	383
5.100.2	Function Documentation	384
5.100.2.1	_close	384
5.100.2.2	_exit	384
5.100.2.3	_free	384
5.100.2.4	_getpid	385
5.100.2.5	_kill	385
5.100.2.6	_log	385
5.100.2.7	_malloc	385

5.100.2.8	<code>_open</code>	386
5.100.2.9	<code>_read</code>	386
5.100.2.10	<code>_seek</code>	387
5.100.2.11	<code>_stat</code>	387
5.100.2.12	<code>_unlink</code>	388
5.100.2.13	<code>_write</code>	388
5.101	<code>src/kernel/pm/syscalls_cli.h</code> File Reference	388
5.101.1	Detailed Description	390
5.101.2	Function Documentation	390
5.101.2.1	<code>_close</code>	390
5.101.2.2	<code>_exit</code>	390
5.101.2.3	<code>_fgetch</code>	391
5.101.2.4	<code>_fgets</code>	391
5.101.2.5	<code>_fputch</code>	391
5.101.2.6	<code>_fputs</code>	392
5.101.2.7	<code>_free</code>	392
5.101.2.8	<code>_getpid</code>	392
5.101.2.9	<code>_kill</code>	392
5.101.2.10	<code>_log</code>	393
5.101.2.11	<code>_malloc</code>	393
5.101.2.12	<code>_open</code>	393
5.101.2.13	<code>_printf</code>	394
5.101.2.14	<code>_read</code>	394
5.101.2.15	<code>_seek</code>	394
5.101.2.16	<code>_stat</code>	395
5.101.2.17	<code>_unlink</code>	395
5.101.2.18	<code>_write</code>	396
5.102	<code>src/kernel/pm/syscalls_shared.h</code> File Reference	396
5.102.1	Detailed Description	398
5.102.2	Define Documentation	398
5.102.2.1	<code>MAX_SYSCALL</code>	398
5.102.2.2	<code>O_CREAT</code>	398
5.102.2.3	<code>O_OPEN</code>	399
5.102.2.4	<code>SEEK_CUR</code>	399
5.102.2.5	<code>SEEK_END</code>	399
5.102.2.6	<code>SEEK_SET</code>	399

---

5.102.2.7	SYS_CLOSE	399
5.102.2.8	SYS_EXIT	399
5.102.2.9	SYS_FREE	399
5.102.2.10	SYS_GETPID	399
5.102.2.11	SYS_KILL	399
5.102.2.12	SYS_LOG	399
5.102.2.13	SYS_MALLOC	400
5.102.2.14	SYS_OPEN	400
5.102.2.15	SYS_READ	400
5.102.2.16	SYS_SEEK	400
5.102.2.17	SYS_STAT	400
5.102.2.18	SYS_UNLINK	400
5.102.2.19	SYS_WRITE	400
5.102.2.20	WAIT_FOR_INTERRUPT	400
5.102.3	Typedef Documentation	400
5.102.3.1	sc_close_args_t	400
5.102.3.2	sc_malloc_args_t	400
5.102.3.3	sc_open_args_t	400
5.102.3.4	sc_read_write_args_t	401
5.102.3.5	sc_seek_args_t	401
5.102.3.6	sc_stat_args_t	401
5.102.3.7	sc_unlink_args_t	401
5.102.3.8	stat	401
5.103	src/tools/bin2c/bin2c.c File Reference	401
5.103.1	Detailed Description	402
5.103.2	Define Documentation	402
5.103.2.1	ITEMSPERLINE	402
5.103.2.2	VAREND	402
5.103.2.3	VARFMT	402
5.103.2.4	VARHDR	402
5.103.2.5	VARTAB	403
5.103.3	Function Documentation	403
5.103.3.1	main	403
5.104	src/tools/chips/chips.c File Reference	403
5.104.1	Function Documentation	403
5.104.1.1	chips_usage	403

---

5.104.1.2 main	403
5.105src/tools/chips/chips.h File Reference	404
5.105.1 Define Documentation	405
5.105.1.1 POTATOES_BLOCK_SIZE	405
5.105.1.2 POTATOES_DATA_FILE	405
5.105.1.3 POTATOES_DIR_ENTRIES_PER_BLOCK	405
5.105.1.4 POTATOES_DIR_ENTRY_SIZE	405
5.105.1.5 POTATOES_DIRECTORY	405
5.105.1.6 POTATOES_NAME_SIZE	406
5.105.2 Typedef Documentation	406
5.105.2.1 potatoes_block_nr	406
5.105.2.2 potatoes_bool	406
5.105.2.3 potatoes_file_info_t	406
5.105.2.4 potatoes_file_nr	406
5.105.2.5 potatoes_inode_nr	406
5.105.2.6 potatoes_sint16	406
5.105.2.7 potatoes_sint32	406
5.105.2.8 potatoes_sint8	406
5.105.2.9 potatoes_size_t	406
5.105.2.10potatoes_time_t	406
5.105.2.11potatoes_uint16	406
5.105.2.12potatoes_uint32	406
5.105.2.13potatoes_uint8	406
5.105.2.14potatpes_float32	406
5.105.2.15potatpes_float64	406
5.105.3 Function Documentation	406
5.105.3.1 __attribute__	406
5.105.3.2 do_close	406
5.105.3.3 do_open	406
5.105.3.4 do_read	407
5.105.3.5 do_write	407
5.105.3.6 dump_consts	407
5.105.3.7 fs_create	407
5.105.3.8 fs_delete	407
5.105.3.9 fs_init	408
5.105.3.10fs_shutdown	408



---

5.105.3.1	lfs_truncate	408
5.105.3.2	get_file_info	408
5.105.3.3	potatoes_disc_create	408
5.105.3.4	potatoes_disc_destroy	408
5.105.3.5	potatoes_set_current_disk	408
5.105.4	Variable Documentation	408
5.105.4.1	inode	408
5.105.4.2	name	408
5.106	src/tools/chips/chipsfs.c File Reference	408
5.106.1	Define Documentation	409
5.106.1.1	FUSE_USE_VERSION	409
5.106.2	Function Documentation	409
5.106.2.1	chips_create	409
5.106.2.2	chips_mkdir	409
5.106.2.3	chips_unlink	409
5.106.2.4	main	409
5.107	src/tools/chips/fs_wrappers.c File Reference	409
5.107.1	Function Documentation	410
5.107.1.1	potatoes_bzero	410
5.107.1.2	potatoes_disc_create	411
5.107.1.3	potatoes_disc_destroy	411
5.107.1.4	potatoes_free	411
5.107.1.5	potatoes_get_hdsiz	411
5.107.1.6	potatoes_hd_read_sector	411
5.107.1.7	potatoes_hd_write_sector	411
5.107.1.8	potatoes_itoa	411
5.107.1.9	potatoes_malloc	411
5.107.1.10	potatoes_mallocn	411
5.107.1.11	potatoes_mem_dump	411
5.107.1.12	potatoes_memcpy	411
5.107.1.13	potatoes_panic	411
5.107.1.14	potatoes_printf	411
5.107.1.15	potatoes_set_current_disk	412
5.107.1.16	potatoes_strcat	412
5.107.1.17	potatoes_strcmp	412
5.107.1.18	potatoes_strdup	412

---

5.107.1.19	potatoes_strlen	412
5.107.1.20	potatoes_strncat	412
5.107.1.21	potatoes_strncpy	412
5.107.1.22	potatoes_strsep	412
5.107.1.23	potatoes_time2str	412
5.107.2	Variable Documentation	412
5.107.2.1	potatoes_current_disc	412
5.108	src/tools/chips/fs_wrappers.h File Reference	412
5.108.1	Define Documentation	413
5.108.1.1	bzero	413
5.108.1.2	cputs	413
5.108.1.3	free	413
5.108.1.4	get_hdsize	413
5.108.1.5	hd_read_sector	413
5.108.1.6	hd_write_sector	414
5.108.1.7	itoa	414
5.108.1.8	mallocn	414
5.108.1.9	mem_dump	414
5.108.1.10	nemcpy	414
5.108.1.11	panic	414
5.108.1.12	printf	414
5.108.1.13	printf	414
5.108.1.14	putchar	414
5.108.1.15	puts	415
5.108.1.16	snprintf	415
5.108.1.17	strcat	415
5.108.1.18	strcmp	415
5.108.1.19	strdup	415
5.108.1.20	strlen	415
5.108.1.21	strncpy	415
5.108.1.22	strsep	415
5.108.1.23	time2str	415
5.108.1.24	vsprintf	415

# Chapter 1

## Bug List

**Global `memmove(void *dest, void *src, size_t count)`** There has to be a better way, ie one that does not use dynamic memory.

**Global `puts(char *s)`** Does not write a terminating newline character. A change in behavior breaks other things (namely `printf()`). Please leave it like that for now.

**Global `strsep(char **str_ptr, char *delims)`** The current implementation does not handle multiple delimiters (as specified in the libc manual). Only the first character in `*delims` is used for tokenizing the input string.



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">address</a> (Address data for the hard disk packed in a struct) . . . . .	9
<a href="#">block_buffer</a> (A buffer for one block) . . . . .	10
<a href="#">command_t</a> . . . . .	10
<a href="#">cpu_state_t</a> (Represents the state of all cpu registers at a given time) . . . . .	11
<a href="#">d_inode</a> (The inode on disk) . . . . .	12
<a href="#">device_t</a> (Describes a single device) . . . . .	14
<a href="#">dir_entry</a> (The directory entry) . . . . .	15
<a href="#">file</a> (The global file descriptor) . . . . .	16
<a href="#">file_info</a> . . . . .	17
<a href="#">gdt_entry</a> (The structure of one GDT entry) . . . . .	18
<a href="#">gdt_pointer</a> (The structure of the GDT pointer which tells the processor where to find our GDT) . . . . .	19
<a href="#">hd_info</a> (This struct is filled by the IDENTIFY DRIVE command) . . . . .	19
<a href="#">heap_t</a> . . . . .	22
<a href="#">idt_entry</a> (The structure of one IDT entry) . . . . .	23
<a href="#">idt_pointer</a> (The structure of the IDT pointer which tells the processor where to find our IDT) . . . . .	24
<a href="#">line</a> . . . . .	25
<a href="#">m_inode</a> (The inode in memory) . . . . .	26
<a href="#">mm_header</a> (Structure of a header of an occupied memory block) . . . . .	27
<a href="#">multiboot</a> (Multiboot structure) . . . . .	28
<a href="#">page</a> . . . . .	30
<a href="#">page_directory</a> . . . . .	31
<a href="#">page_table</a> . . . . .	32
<a href="#">potatoes_dir_entry</a> . . . . .	32
<a href="#">potatoes_file_info</a> . . . . .	33
<a href="#">potatoes_time</a> . . . . .	33
<a href="#">PotatoesDisk</a> . . . . .	34
<a href="#">proc_file</a> (The process file descriptor) . . . . .	35
<a href="#">process_t</a> (Structure describing a single process) . . . . .	35
<a href="#">ring_fifo</a> (A static circular FIFO buffer) . . . . .	38
<a href="#">sc_close_args_t</a> (Arguments for the CLOSE syscall) . . . . .	39
<a href="#">sc_malloc_args_t</a> (Arguments for the MALLOC syscall) . . . . .	40
<a href="#">sc_open_args_t</a> (Arguments for the OPEN syscall) . . . . .	40
<a href="#">sc_read_write_args_t</a> (Arguments for the READ and WRITE syscalls) . . . . .	41

---

<a href="#">sc_seek_args_t</a> (Arguments for the SEEK syscall) . . . . .	42
<a href="#">sc_stat_args_t</a> (Arguments for the STAT syscall) . . . . .	43
<a href="#">sc_unlink_args_t</a> (Arguments for the UNLINK syscall) . . . . .	44
<a href="#">Semaphore</a> . . . . .	44
<a href="#">shell_cmd_t</a> (A single shell commmand) . . . . .	45
<a href="#">shortcut</a> (Structure of a shortcut) . . . . .	45
<a href="#">super_block</a> . . . . .	46
<a href="#">time</a> (Global time struct) . . . . .	47
<a href="#">Tone</a> (Stores a frequency and a duration) . . . . .	49
<a href="#">virt_monitor</a> (Structure that represents a virtual monitor) . . . . .	49

# Chapter 3

## File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

src/apps/apps.h (Header file for the applications ) . . . . .	53
src/apps/brainfuck_interpreter.c (Interprets brainfuck source code ) . . . . .	54
src/apps/brainfuck_interpreter.h (Header file for the brainfuck interpreter ) . . . . .	57
src/apps/editor.c (Simple Potatoes/Etios EDitor ) . . . . .	58
src/apps/editor.h (Simple Potatoes/Etios EDitor header ) . . . . .	59
src/apps/games.h (Header file for the games ) . . . . .	61
src/apps/memview.c (Main source file of the memory viewer tool "memview" ) . . . . .	67
src/apps/pong.c (Pong game ) . . . . .	72
src/apps/shell_cmds.c (Implementations of the shell commands ) . . . . .	74
src/apps/shell_main.c (EtiOS shell entry point and main source file ) . . . . .	82
src/apps/shell_main.h (EtiOS shell main header file ) . . . . .	84
src/apps/shell_utils.c (Shell utility functions ) . . . . .	87
src/apps/shell_utils.h (Shell utility functions ) . . . . .	90
src/apps/snake.c (Snake game ) . . . . .	93
src/apps/snapshot.c (Programm to make and view screenshots ) . . . . .	95
src/apps/synthesizer.c (Synthesizer tool ) . . . . .	97
src/kernel/fs/fs_block_dev.c (Basic block based functions ) . . . . .	98
src/kernel/fs/fs_block_dev.h (Basic definitions of all block based functions ) . . . . .	102
src/kernel/fs/fs_bmap.c (Functions to manage the block bitmap (bmap) ) . . . . .	105
src/kernel/fs/fs_bmap.h (Basic definitions concerning the block bitmap (bmap) ) . . . . .	108
src/kernel/fs/fs_buf.c (Basic buffer functions ) . . . . .	112
src/kernel/fs/fs_buf.h (Basic buffer/cache definitions ) . . . . .	113
src/kernel/fs/fs_const.h (Basic constant definitions ) . . . . .	116
src/kernel/fs/fs_create_delete.c (Functions concerning syscalls "create" and "delete" ) . . . . .	120
src/kernel/fs/fs_dir.c (Functions on directories ) . . . . .	122
src/kernel/fs/fs_dir.h (Basic functions concerning directories ) . . . . .	127
src/kernel/fs/fs_file_table.c (The file descriptor table functions ) . . . . .	131
src/kernel/fs/fs_file_table.h (The file descriptor table ) . . . . .	138
src/kernel/fs/fs_inode_table.c (The inode table ) . . . . .	145
src/kernel/fs/fs_inode_table.h (The inode table ) . . . . .	151
src/kernel/fs/fs_io_functions.h (Functions definitions concerning read, write, open, close, create, delete ) . . . . .	157
src/kernel/fs/fs_main.c (This is the central program dealing as an interface between FS and PM ) . . . . .	162

src/kernel/fs/fs_main.h (Basic definitions of all functions concerning work on files ) . . . . .	166
src/kernel/fs/fs_open_close.c (Basic definitions concerning syscalls "open" and "close" ) . . . . .	169
src/kernel/fs/fs_read_write.c (Functions concerning syscalls "read" and "write" ) . . . . .	171
src/kernel/fs/fs_super.c (Basic functions of the super block ) . . . . .	173
src/kernel/fs/fs_super.h (The superblock table ) . . . . .	175
src/kernel/fs/fs_tests.c (Some fs test-functions ) . . . . .	178
src/kernel/fs/fs_types.h (Basic type definitions ) . . . . .	181
src/kernel/include/assert.h (Definition of the ASSERT() macro ) . . . . .	184
src/kernel/include/const.h (Basic constant definitions ) . . . . .	185
src/kernel/include/debug.h (Debugging utility functions ) . . . . .	187
src/kernel/include/init.h (Basic definitions for functions used in the main()-function of the kernel ) . . . . .	188
src/kernel/include/limits.h (Definition of the maximum and minimum values of different types ) . . . . .	193
src/kernel/include/ringbuffer.h (Ringbuffer definitions ) . . . . .	194
src/kernel/include/stdarg.h (Macros dealing with variable argument lists ) . . . . .	198
src/kernel/include/stdio.h (Standard I/O functions ) . . . . .	199
src/kernel/include/stdlib.h (Includes some general help functions for type conversion, memory allocation, process control and other purposes ) . . . . .	205
src/kernel/include/string.h (Headers for string.c ) . . . . .	208
src/kernel/include/types.h (Basic type definitions ) . . . . .	216
src/kernel/include/util.h (Useful function headers ) . . . . .	217
src/kernel/init/main.c (Main kernel file ) . . . . .	218
src/kernel/init/tests.c (Basic definitions for functions used in the main()-function of the kernel ) . . . . .	220
src/kernel/io/int_handler.h (Header-file for specific hardware interrupt-handlers ) . . . . .	231
src/kernel/io/int_idt.c (Builds and initializes the interrupt descriptor table ) . . . . .	232
src/kernel/io/int_irq.c (Handler for hardware-interrupts ) . . . . .	235
src/kernel/io/int_isr.c (Handler for exceptions ) . . . . .	239
src/kernel/io/io.h (The global IO functions header ) . . . . .	245
src/kernel/io/io_harddisk.c (Harddisk-handler ) . . . . .	250
src/kernel/io/io_harddisk.h (Header file for the hard disk driver ) . . . . .	253
src/kernel/io/io_keyboard.c (Keyboard-handler ) . . . . .	260
src/kernel/io/io_keyboard.h (Header for the keyboard-handler ) . . . . .	263
src/kernel/io/io_main.c (IO-init function for main ) . . . . .	267
src/kernel/io/io_monitor.c (Functions to print things on the monitor ) . . . . .	268
src/kernel/io/io_rtc.c (The real-time clock ) . . . . .	271
src/kernel/io/io_rtc.h (The real-time clock constants and the time struct ) . . . . .	273
src/kernel/io/io_sound.c (Sound driver ) . . . . .	276
src/kernel/io/io_sound.h (Sound driver header ) . . . . .	278
src/kernel/io/io_timer.c (Timer-handler ) . . . . .	287
src/kernel/io/io_timer.h (Header file for the timer-handler ) . . . . .	289
src/kernel/io/io_virtual.c (Function used by virtual outputs ) . . . . .	290
src/kernel/io/io_virtual.h (Header for the virtual monitor structure ) . . . . .	295
src/kernel/io/io_virtual_monitors.c (Shell & monitor interface ) . . . . .	302
src/kernel/lib/ringbuffer.c (Ring buffer implementation ) . . . . .	304
src/kernel/lib/stdio.c (Standard I/O functions ) . . . . .	309
src/kernel/lib/stdlib.c (Functions to allocate and free memory ) . . . . .	311
src/kernel/lib/string.c (Basic functions for string manipulation ) . . . . .	315
src/kernel/mm/mm.h (Definitions for the functions and variables used in the memory management ) . . . . .	323
src/kernel/mm/mm_bitmap.c (All functions concerning the allocation and deallocation of frames ) . . . . .	328
src/kernel/mm/mm_bitmap.h (Declarations of the functions needed for frame allocation ) . . . . .	331
src/kernel/mm/mm_gdt.c (Creates and initializes the global descriptor table ) . . . . .	333
src/kernel/mm/mm_heap.c (Declarations for the functions and variables needed for paging ) . . . . .	335
src/kernel/mm/mm_main.c (Initializes memory management ) . . . . .	338
src/kernel/mm/mm_paging.h (Declarations for the functions and variables needed for paging ) . . . . .	341



---

src/kernel/pm/dev_brainfuck.c (The brainfuck device ) . . . . .	344
src/kernel/pm/dev_clock.c (The clock device ) . . . . .	345
src/kernel/pm/dev_framebuffer.c (The framebuffer device ) . . . . .	347
src/kernel/pm/dev_keyboard.c (The keyboard device ) . . . . .	348
src/kernel/pm/dev_null.c (The null device ) . . . . .	350
src/kernel/pm/dev_stdin.c (The STDIN device ) . . . . .	352
src/kernel/pm/dev_stdout.c (The STDOUT device ) . . . . .	353
src/kernel/pm/pm_devices.c (Implementation of the device subsystem ) . . . . .	355
src/kernel/pm/pm_devices.h (Definitions for the device subsystem ) . . . . .	357
src/kernel/pm/pm_input.c (Process management code that handles keyboard input ) . . . . .	359
src/kernel/pm/pm_input.h (Input management header ) . . . . .	360
src/kernel/pm/pm_main.c (Process management main source file ) . . . . .	361
src/kernel/pm/pm_main.h (Process management main header file ) . . . . .	367
src/kernel/pm/pm_syscalls.c (This is the kernel side implementation of the syscalls ) . . . . .	373
src/kernel/pm/pm_syscalls.h (Header file for the kernel side syscall implementations ) . . . . .	378
src/kernel/pm/syscalls_cli.c (This contains all the syscall definitions for the "client" side ) . . . . .	382
src/kernel/pm/syscalls_cli.h (Header file for the "client" side syscall magic ) . . . . .	388
src/kernel/pm/syscalls_shared.h (This holds definitions and things which need to be shared between the kernel side of the syscall subsystem and the client side ) . . . . .	396
src/tools/bin2c/bin2c.c (Reads a file and dumps its contents into a C byte array ) . . . . .	401
src/tools/chips/chips.c . . . . .	403
src/tools/chips/chips.h . . . . .	404
src/tools/chips/chipsfs.c . . . . .	408
src/tools/chips/fs_wrappers.c . . . . .	409
src/tools/chips/fs_wrappers.h . . . . .	412



# Chapter 4

## Data Structure Documentation

### 4.1 address Struct Reference

Address data for the hard disk packed in a struct.

```
#include <io_harddisk.h>
```

#### Data Fields

- [uint16 cyl](#)
- [uint8 head](#)
- [uint8 sector](#)

#### 4.1.1 Detailed Description

Address data for the hard disk packed in a struct.

#### 4.1.2 Field Documentation

##### 4.1.2.1 uint16 address::cyl

Referenced by `hd_read_sector()`, `hd_write_sector()`, and `itoaddr()`.

##### 4.1.2.2 uint8 address::head

Referenced by `hd_read_sector()`, `hd_write_sector()`, and `itoaddr()`.

##### 4.1.2.3 uint8 address::sector

Referenced by `hd_read_sector()`, `hd_write_sector()`, and `itoaddr()`.

The documentation for this struct was generated from the following file:

- [src/kernel/io/io\\_harddisk.h \(r/r22\)](#)

## 4.2 `block_buffer` Struct Reference

A buffer for one block.

```
#include <fs_types.h>
```

### Data Fields

- [block\\_nr](#) `block_nr`
- [uint8](#) `cache` [BLOCK\_SIZE]

### 4.2.1 Detailed Description

A buffer for one block.

### 4.2.2 Field Documentation

#### 4.2.2.1 `block_nr` `block_buffer::block_nr`

Referenced by `cache_block()`, `clear_block()`, `clear_cache()`, `wrt_block()`, and `wrt_cache()`.

#### 4.2.2.2 `uint8` `block_buffer::cache`[BLOCK\_SIZE]

Referenced by `cache_block()`, `clear_block()`, `clear_cache()`, `fs_read()`, `fs_write()`, `rd_block()`, `wrt_block()`, and `wrt_cache()`.

The documentation for this struct was generated from the following file:

- [src/kernel/fs/fs\\_types.h](#) (r/r22)

## 4.3 `command_t` Struct Reference

### Data Fields

- `int(* exec)(int argc, char **argv)`
- `char const * name`
- `char const * desc`

### 4.3.1 Field Documentation

#### 4.3.1.1 `char const*` `command_t::desc`

#### 4.3.1.2 `int(* command\_t::exec)(int argc, char **argv)`

Referenced by `main()`.

### 4.3.1.3 `char const* command_t::name`

The documentation for this struct was generated from the following file:

- `src/tools/chips/chips.c` ([r/r22](#))

## 4.4 `cpu_state_t` Struct Reference

Represents the state of all cpu registers at a given time.

```
#include <pm_main.h>
```

### Data Fields

- unsigned int `gs`
- unsigned int `fs`
- unsigned int `es`
- unsigned int `ds`
- unsigned int `edi`
- unsigned int `esi`
- unsigned int `ebp`
- unsigned int `esp`
- unsigned int `ebx`
- unsigned int `edx`
- unsigned int `ecx`
- unsigned int `eax`
- unsigned int `int_no`
- unsigned int `err_code`
- unsigned int `eip`
- unsigned int `cs`
- unsigned int `eflags`
- unsigned int `useresp`
- unsigned int `ss`

### 4.4.1 Detailed Description

Represents the state of all cpu registers at a given time. Primarily used to provide additional information and error locations to the user in the exception handlers.

## 4.4.2 Field Documentation

- 4.4.2.1 unsigned int `cpu_state_t::cs`
- 4.4.2.2 unsigned int `cpu_state_t::ds`
- 4.4.2.3 unsigned int `cpu_state_t::eax`
- 4.4.2.4 unsigned int `cpu_state_t::ebp`
- 4.4.2.5 unsigned int `cpu_state_t::ebx`
- 4.4.2.6 unsigned int `cpu_state_t::ecx`
- 4.4.2.7 unsigned int `cpu_state_t::edi`
- 4.4.2.8 unsigned int `cpu_state_t::edx`
- 4.4.2.9 unsigned int `cpu_state_t::eflags`
- 4.4.2.10 unsigned int `cpu_state_t::eip`
- 4.4.2.11 unsigned int `cpu_state_t::err_code`
- 4.4.2.12 unsigned int `cpu_state_t::es`
- 4.4.2.13 unsigned int `cpu_state_t::esi`
- 4.4.2.14 unsigned int `cpu_state_t::esp`
- 4.4.2.15 unsigned int `cpu_state_t::fs`
- 4.4.2.16 unsigned int `cpu_state_t::gs`
- 4.4.2.17 unsigned int `cpu_state_t::int_no`

Referenced by `isr_handler()`.

- 4.4.2.18 unsigned int `cpu_state_t::ss`
- 4.4.2.19 unsigned int `cpu_state_t::useresp`

The documentation for this struct was generated from the following file:

- [src/kernel/pm/pm\\_main.h \(r/r22\)](#)

## 4.5 `d_inode` Struct Reference

The inode on disk.

```
#include <fs_types.h>
```

## Data Fields

- [uint16 i\\_mode](#)
- [uint32 i\\_size](#)
- [time\\_t i\\_create\\_ts](#)
- [time\\_t i\\_modify\\_ts](#)
- [block\\_nr i\\_direct\\_pointer](#) [NUM\_DIRECT\_POINTER]
- [block\\_nr i\\_single\\_indirect\\_pointer](#)
- [block\\_nr i\\_double\\_indirect\\_pointer](#)

### 4.5.1 Detailed Description

The inode on disk.

### 4.5.2 Field Documentation

#### 4.5.2.1 `time_t d_inode::i_create_ts`

Referenced by `cpy_dinode_to_minode()`, and `cpy_minode_to_dinode()`.

#### 4.5.2.2 `block_nr d_inode::i_direct_pointer`[NUM\_DIRECT\_POINTER]

Referenced by `cpy_dinode_to_minode()`, `cpy_minode_to_dinode()`, and `dump_dinode()`.

#### 4.5.2.3 `block_nr d_inode::i_double_indirect_pointer`

Referenced by `cpy_dinode_to_minode()`, `cpy_minode_to_dinode()`, and `dump_dinode()`.

#### 4.5.2.4 `uint16 d_inode::i_mode`

Referenced by `cpy_dinode_to_minode()`, `cpy_minode_to_dinode()`, and `dump_dinode()`.

#### 4.5.2.5 `time_t d_inode::i_modify_ts`

Referenced by `cpy_dinode_to_minode()`, and `cpy_minode_to_dinode()`.

#### 4.5.2.6 `block_nr d_inode::i_single_indirect_pointer`

Referenced by `cpy_dinode_to_minode()`, `cpy_minode_to_dinode()`, and `dump_dinode()`.

#### 4.5.2.7 `uint32 d_inode::i_size`

Referenced by `cpy_dinode_to_minode()`, and `cpy_minode_to_dinode()`.

The documentation for this struct was generated from the following file:

- [src/kernel/fs/fs\\_types.h \(r/r22\)](#)

## 4.6 device\_t Struct Reference

Describes a single device.

```
#include <pm_devices.h>
```

### Data Fields

- char [name](#) [20]  
*Device name, exported to the file system.*
- int [fd](#)  
*The devices file descriptor.*
- void \* [data](#)  
*The data pointer can be used by a device to save state information which is unique to the device instance.*
- [dev\\_open\\_func](#) open
- [dev\\_close\\_func](#) close
- [dev\\_read\\_func](#) read
- [dev\\_write\\_func](#) write
- [dev\\_seek\\_func](#) seek
- struct [device\\_t](#) \* [next](#)  
*Linked list next pointer.*

### 4.6.1 Detailed Description

Describes a single device.

### 4.6.2 Field Documentation

#### 4.6.2.1 dev\_close\_func device\_t::close

Referenced by `sys_close()`.

#### 4.6.2.2 void\* device\_t::data

The data pointer can be used by a device to save state information which is unique to the device instance.

#### 4.6.2.3 int device\_t::fd

The devices file descriptor.

Note that this is global and does not change from process to process. Make sure it is unique.

Referenced by `pm_fd2device()`, and `pm_register_device()`.



#### 4.6.2.4 `char device_t::name[20]`

Device name, exported to the file system.

Referenced by `pm_name2device()`, and `pm_register_device()`.

#### 4.6.2.5 `struct device_t* device_t::next`

Linked list next pointer.

Referenced by `pm_fd2device()`, `pm_name2device()`, and `pm_register_device()`.

#### 4.6.2.6 `dev_open_func device_t::open`

Referenced by `sys_open()`.

#### 4.6.2.7 `dev_read_func device_t::read`

Referenced by `sys_read()`.

#### 4.6.2.8 `dev_seek_func device_t::seek`

Referenced by `sys_seek()`.

#### 4.6.2.9 `dev_write_func device_t::write`

Referenced by `sys_write()`.

The documentation for this struct was generated from the following file:

- [src/kernel/pm/pm\\_devices.h \(r/r22\)](#)

## 4.7 `dir_entry` Struct Reference

The directory entry.

```
#include <fs_types.h>
```

### Data Fields

- [block\\_nr](#) `inode`
- `char name` [NAME\_SIZE]

### 4.7.1 Detailed Description

The directory entry.

## 4.7.2 Field Documentation

### 4.7.2.1 `block_nr dir_entry::inode`

### 4.7.2.2 `char dir_entry::name[NAME_SIZE]`

The documentation for this struct was generated from the following file:

- [src/kernel/fs/fs\\_types.h \(r/r22\)](#)

## 4.8 file Struct Reference

The global file descriptor.

```
#include <fs_types.h>
```

### Data Fields

- [file\\_nr f\\_desc](#)
- [m\\_inode \\* f\\_inode](#)
- [char \\* f\\_name](#)
- [uint8 f\\_mode](#)
- [uint16 f\\_count](#)

### 4.8.1 Detailed Description

The global file descriptor.

### 4.8.2 Field Documentation

#### 4.8.2.1 `uint16 file::f_count`

Referenced by `alloc_file()`, `dump_file()`, `free_file()`, `get_file_info()`, and `inc_count()`.

#### 4.8.2.2 `file_nr file::f_desc`

Referenced by `alloc_file()`, `dump_file()`, `free_file()`, `fs_close()`, `init_file_table()`, `inode2desc()`, `insert_file()`, and `name2desc()`.

#### 4.8.2.3 `m_inode* file::f_inode`

Referenced by `dump_file()`, `free_file()`, `fs_close()`, `fs_truncate()`, `get_file_info()`, `inode2desc()`, `insert_file()`, `sys_seek()`, `test_rw_qualitative()`, and `test_rw_quantitative()`.

#### 4.8.2.4 `uint8 file::f_mode`

Referenced by `dump_file()`, `get_file_info()`, and `insert_file()`.

#### 4.8.2.5 char\* file::f\_name

Referenced by `dump_file()`, `free_file()`, `fs_close()`, `get_file_info()`, and `insert_file()`.

The documentation for this struct was generated from the following file:

- `src/kernel/fs/fs_types.h` ([r/r22](#))

## 4.9 file\_info Struct Reference

```
#include <fs_types.h>
```

### Data Fields

- char `name` [NAME\_SIZE]
- uint16 `mode`
- uint32 `size`
- time\_t `create_ts`
- time\_t `modify_ts`
- uint16 `num_links`

### 4.9.1 Field Documentation

#### 4.9.1.1 time\_t file\_info::create\_ts

Referenced by `get_file_info()`.

#### 4.9.1.2 uint16 file\_info::mode

Referenced by `do_read()`, `get_file_info()`, `shell_cmd_ls()`, `sys_unlink()`, and `sys_write()`.

#### 4.9.1.3 time\_t file\_info::modify\_ts

Referenced by `get_file_info()`, and `shell_cmd_ls()`.

#### 4.9.1.4 char file\_info::name[NAME\_SIZE]

Referenced by `get_file_info()`.

#### 4.9.1.5 uint16 file\_info::num\_links

Referenced by `get_file_info()`.

#### 4.9.1.6 uint32 file\_info::size

Referenced by `do_read()`, `get_file_info()`, `shell_cmd_ls()`, and `sys_unlink()`.

The documentation for this struct was generated from the following file:

- `src/kernel/fs/fs_types.h` ([r/r22](#))

## 4.10 gdt\_entry Struct Reference

The structure of one GDT entry.

```
#include <mm.h>
```

### Data Fields

- [uint16 limit\\_low](#)
- [uint16 base\\_low](#)
- [uint8 base\\_middle](#)
- [uint8 access](#)
- [uint8 granularity](#)
- [uint8 base\\_high](#)

### 4.10.1 Detailed Description

The structure of one GDT entry. access byte: 7 | 6 5 | 4 | 3 0 | present | ring (0-3) | descriptor type | segment type (1010:code;0010:data) | granularity byte: 7 | 6 | 5 | 4 | 3 0 | granularity | operand size | always 0 | available (always 0) | segment length (bits 19 - 16) | granularity bit: 0 -> 1 byte; 1 -> 4 kbyte operand size: 0 -> 16bit; 1 -> 32bit

### 4.10.2 Field Documentation

#### 4.10.2.1 uint8 gdt\_entry::access

Referenced by `gdt_add_entry()`.

#### 4.10.2.2 uint8 gdt\_entry::base\_high

Referenced by `gdt_add_entry()`.

#### 4.10.2.3 uint16 gdt\_entry::base\_low

Referenced by `gdt_add_entry()`.

#### 4.10.2.4 uint8 gdt\_entry::base\_middle

Referenced by `gdt_add_entry()`.

#### 4.10.2.5 `uint8 gdt_entry::granularity`

Referenced by `gdt_add_entry()`.

#### 4.10.2.6 `uint16 gdt_entry::limit_low`

Referenced by `gdt_add_entry()`.

The documentation for this struct was generated from the following file:

- [src/kernel/mm/mm.h \(r/r22\)](#)

## 4.11 `gdt_pointer` Struct Reference

The structure of the GDT pointer which tells the processor where to find our GDT.

```
#include <mm.h>
```

### Data Fields

- [uint16 limit](#)
- [uint32 base](#)

### 4.11.1 Detailed Description

The structure of the GDT pointer which tells the processor where to find our GDT.

### 4.11.2 Field Documentation

#### 4.11.2.1 `uint32 gdt_pointer::base`

Referenced by `gdt_init()`.

#### 4.11.2.2 `uint16 gdt_pointer::limit`

Referenced by `gdt_init()`.

The documentation for this struct was generated from the following file:

- [src/kernel/mm/mm.h \(r/r22\)](#)

## 4.12 `hd_info` Struct Reference

This struct is filled by the IDENTIFY DRIVE command.

```
#include <io_harddisk.h>
```

## Data Fields

- uint16 config\_word
- uint16 num\_cyl
- uint16 reserved1
- uint16 num\_head
- uint16 bytes\_per\_track
- uint16 bytes\_per\_sector
- uint16 sector\_per\_track
- uint16 manufacturer1 [3]
- uint16 serial [10]
- uint16 buffer\_type
- uint16 buffer\_size
- uint16 num\_ecc\_bytes
- uint16 firmware [4]
- uint16 type [20]
- uint16 rw\_multiple\_flg
- uint16 dw\_io\_flg
- uint16 lba\_dma\_flg
- uint16 reserved2
- uint16 timingmode\_pio
- uint16 timingmode\_dma
- uint16 reserved3
- uint16 apparent\_cyl
- uint16 apparent\_head
- uint16 apparent\_sector\_per\_track
- uint16 apparent\_capacity [2]
- uint16 sectors\_per\_int
- uint16 num\_lba\_sectors [2]
- uint16 mode\_single\_dma
- uint16 mode\_multi\_dma
- uint16 reserved4 [64]
- uint16 manufacturer2 [32]
- uint16 reserved5 [96]

### 4.12.1 Detailed Description

This struct is filled by the IDENTIFY DRIVE command. It contains hard disk geometry data and more.

### 4.12.2 Field Documentation

#### 4.12.2.1 uint16 hd\_info::apparent\_capacity[2]

Referenced by `get_hdsz()`.

#### 4.12.2.2 uint16 hd\_info::apparent\_cyl

Referenced by `dump_hd1()`.

#### 4.12.2.3 uint16 hd\_info::apparent\_head

Referenced by dump\_hd1(), and itoaddr().

#### 4.12.2.4 uint16 hd\_info::apparent\_sector\_per\_track

Referenced by dump\_hd1(), and itoaddr().

#### 4.12.2.5 uint16 hd\_info::buffer\_size

Referenced by dump\_hd1().

#### 4.12.2.6 uint16 hd\_info::buffer\_type

Referenced by dump\_hd1().

#### 4.12.2.7 uint16 hd\_info::bytes\_per\_sector

Referenced by dump\_hd1().

#### 4.12.2.8 uint16 hd\_info::bytes\_per\_track

#### 4.12.2.9 uint16 hd\_info::config\_word

#### 4.12.2.10 uint16 hd\_info::dw\_io\_flg

#### 4.12.2.11 uint16 hd\_info::firmware[4]

#### 4.12.2.12 uint16 hd\_info::lba\_dma\_flg

Referenced by dump\_hd1().

- 4.12.2.13 `uint16 hd_info::manufacturer1[3]`
- 4.12.2.14 `uint16 hd_info::manufacturer2[32]`
- 4.12.2.15 `uint16 hd_info::mode_multi_dma`
- 4.12.2.16 `uint16 hd_info::mode_single_dma`
- 4.12.2.17 `uint16 hd_info::num_cyl`
- 4.12.2.18 `uint16 hd_info::num_ecc_bytes`
- 4.12.2.19 `uint16 hd_info::num_head`
- 4.12.2.20 `uint16 hd_info::num_lba_sectors[2]`
- 4.12.2.21 `uint16 hd_info::reserved1`
- 4.12.2.22 `uint16 hd_info::reserved2`
- 4.12.2.23 `uint16 hd_info::reserved3`
- 4.12.2.24 `uint16 hd_info::reserved4[64]`
- 4.12.2.25 `uint16 hd_info::reserved5[96]`
- 4.12.2.26 `uint16 hd_info::rw_multiple_flg`
- 4.12.2.27 `uint16 hd_info::sector_per_track`
- 4.12.2.28 `uint16 hd_info::sectors_per_int`
- 4.12.2.29 `uint16 hd_info::serial[10]`
- 4.12.2.30 `uint16 hd_info::timingmode_dma`
- 4.12.2.31 `uint16 hd_info::timingmode_pio`
- 4.12.2.32 `uint16 hd_info::type[20]`

The documentation for this struct was generated from the following file:

- [src/kernel/io/io\\_harddisk.h \(r/r22\)](#)

## 4.13 `heap_t` Struct Reference

```
#include <mm.h>
```

### Data Fields

- `mm_header * start`



- [mm\\_header \\* end](#)
- [uint32 max\\_addr](#)
- [uint8 supervisor](#)
- [uint8 readonly](#)

### 4.13.1 Field Documentation

#### 4.13.1.1 mm\_header\* heap\_t::end

Referenced by `create_heap()`, `heap_contract()`, `heap_expand()`, `heap_get_size()`, `heap_mallocn()`, `heap_mem_dump()`, `malloc_test()`, `memview_main()`, `mm_init_output()`, `mv_show_stats()`, and `update_view()`.

#### 4.13.1.2 uint32 heap\_t::max\_addr

Referenced by `create_heap()`.

#### 4.13.1.3 uint8 heap\_t::readonly

Referenced by `create_heap()`, and `heap_expand()`.

#### 4.13.1.4 mm\_header\* heap\_t::start

Referenced by `create_heap()`, `heap_contract()`, `heap_expand()`, `heap_get_size()`, `heap_mallocn()`, `heap_mem_dump()`, `malloc_test()`, `mark_visual_block()`, `memview_main()`, `mm_init_output()`, `mv_show_stats()`, and `update_view()`.

#### 4.13.1.5 uint8 heap\_t::supervisor

Referenced by `create_heap()`, and `heap_expand()`.

The documentation for this struct was generated from the following file:

- `src/kernel/mm/mm.h (r/r22)`

## 4.14 idt\_entry Struct Reference

The structure of one IDT entry.

### Data Fields

- [uint16 low\\_offset](#)  
*Lower 16 bit of the interrupt handler's code address.*
- [uint16 selector](#)  
*Code segment selector in the GDT.*
- [uint8 separator](#)

*Unused.*

- [uint8 flags](#)
- [uint16 high\\_offset](#)

*Upper 16 bit of the interrupt handler's code address.*

### 4.14.1 Detailed Description

The structure of one IDT entry.

### 4.14.2 Field Documentation

#### 4.14.2.1 `uint8 idt_entry::flags`

Referenced by `idt_fill_entry()`.

#### 4.14.2.2 `uint16 idt_entry::high_offset`

Upper 16 bit of the interrupt handler's code address.

Referenced by `idt_fill_entry()`.

#### 4.14.2.3 `uint16 idt_entry::low_offset`

Lower 16 bit of the interrupt handler's code address.

Referenced by `idt_fill_entry()`.

#### 4.14.2.4 `uint16 idt_entry::selector`

Code segment selector in the GDT.

Referenced by `idt_fill_entry()`.

#### 4.14.2.5 `uint8 idt_entry::separator`

Unused.

Referenced by `idt_fill_entry()`.

The documentation for this struct was generated from the following file:

- `src/kernel/io/int_idt.c (r/r22)`

## 4.15 `idt_pointer` Struct Reference

The structure of the IDT pointer which tells the processor where to find our IDT.

## Data Fields

- [uint16 maxsize](#)
- [uint32 start](#)

### 4.15.1 Detailed Description

The structure of the IDT pointer which tells the processor where to find our IDT.

### 4.15.2 Field Documentation

#### 4.15.2.1 uint16 idt\_pointer::maxsize

Referenced by `idt_init()`.

#### 4.15.2.2 uint32 idt\_pointer::start

Referenced by `idt_init()`.

The documentation for this struct was generated from the following file:

- [src/kernel/io/int\\_idt.c \(r/r22\)](#)

## 4.16 line Struct Reference

```
#include <editor.h>
```

## Data Fields

- int [num\\_chars](#)
- int [offset](#)
- unsigned char [linewidth](#)
- struct [line](#) \* [next](#)
- struct [line](#) \* [prev](#)

### 4.16.1 Field Documentation

#### 4.16.1.1 unsigned char line::linewidth

#### 4.16.1.2 struct line\* line::next

Referenced by `speed()`.

#### 4.16.1.3 int line::num\_chars

Referenced by `speed()`.

#### 4.16.1.4 `int line::offset`

Referenced by `speed()`.

#### 4.16.1.5 `struct line* line::prev`

Referenced by `speed()`.

The documentation for this struct was generated from the following file:

- [src/apps/editor.h \(r/r22\)](#)

## 4.17 `m_inode` Struct Reference

The inode in memory.

```
#include <fs_types.h>
```

### Data Fields

- [inode\\_nr i\\_num](#)
- [block\\_nr i\\_adr](#)
- [uint16 i\\_mode](#)
- [uint32 i\\_size](#)
- [time\\_t i\\_create\\_ts](#)
- [time\\_t i\\_modify\\_ts](#)
- [block\\_nr i\\_direct\\_pointer](#) [NUM\_DIRECT\_POINTER]
- [block\\_nr i\\_single\\_indirect\\_pointer](#)
- [block\\_nr i\\_double\\_indirect\\_pointer](#)

### 4.17.1 Detailed Description

The inode in memory.

### 4.17.2 Field Documentation

#### 4.17.2.1 `block_nr m_inode::i_adr`

Referenced by `dump_inode()`, `free_file()`, `get_data_block()`, `load_root()`, `new_minode()`, `read_minode()`, `rfsearch()`, `search_file()`, and `write_inode()`.

#### 4.17.2.2 `time_t m_inode::i_create_ts`

Referenced by `cpy_dinode_to_minode()`, `cpy_minode_to_dinode()`, and `new_minode()`.

#### 4.17.2.3 `block_nr m_inode::i_direct_pointer`[NUM\_DIRECT\_POINTER]

Referenced by `cpy_dinode_to_minode()`, `cpy_minode_to_dinode()`, `dump_inode()`, `fs_truncate()`, `get_data_block()`, and `new_minode()`.

#### 4.17.2.4 block\_nr m\_inode::i\_double\_indirect\_pointer

Referenced by `cpy_dinode_to_minode()`, `cpy_minode_to_dinode()`, `dump_inode()`, `fs_truncate()`, `get_data_block()`, and `new_minode()`.

#### 4.17.2.5 uint16 m\_inode::i\_mode

Referenced by `cpy_dinode_to_minode()`, `cpy_minode_to_dinode()`, `delete_file_from_dir()`, `dump_inode()`, `fs_open()`, `insert_file_into_dir()`, and `new_minode()`.

#### 4.17.2.6 time\_t m\_inode::i\_modify\_ts

Referenced by `cpy_dinode_to_minode()`, `cpy_minode_to_dinode()`, `delete_file_from_dir()`, `fs_write()`, `insert_file_into_dir()`, and `new_minode()`.

#### 4.17.2.7 inode\_nr m\_inode::i\_num

Referenced by `alloc_inode()`, `create_root()`, `dump_inode()`, `free_file()`, `free_inode()`, `init_inode_table()`, `inode2desc()`, `load_root()`, `new_minode()`, and `test_inode_table()`.

#### 4.17.2.8 block\_nr m\_inode::i\_single\_indirect\_pointer

Referenced by `cpy_dinode_to_minode()`, `cpy_minode_to_dinode()`, `dump_inode()`, `fs_truncate()`, `get_data_block()`, and `new_minode()`.

#### 4.17.2.9 uint32 m\_inode::i\_size

Referenced by `cpy_dinode_to_minode()`, `cpy_minode_to_dinode()`, `delete_file_from_dir()`, `fs_read()`, `fs_truncate()`, `fs_write()`, `insert_file_into_dir()`, `new_minode()`, and `sys_seek()`.

The documentation for this struct was generated from the following file:

- [src/kernel/fs/fs\\_types.h \(r/r22\)](#)

## 4.18 mm\_header Struct Reference

the structure of a header of an occupied memory block

```
#include <mm.h>
```

### Data Fields

- struct [mm\\_header](#) \* `prev`
- struct [mm\\_header](#) \* `next`
- char `name` [32]
- [uint32](#) `size`

### 4.18.1 Detailed Description

the structure of a header of an occupied memory block

### 4.18.2 Field Documentation

#### 4.18.2.1 char mm\_header::name[32]

Referenced by `create_heap()`, `heap_contract()`, `heap_expand()`, `heap_mem_dump()`, `heap_setup_block()`, and `realloc()`.

#### 4.18.2.2 struct mm\_header\* mm\_header::next

Referenced by `create_heap()`, `heap_contract()`, `heap_expand()`, `heap_mallocn()`, `heap_mem_dump()`, `heap_setup_block()`, `mv_show_stats()`, `realloc()`, and `update_view()`.

#### 4.18.2.3 struct mm\_header\* mm\_header::prev

Referenced by `create_heap()`, `heap_contract()`, `heap_expand()`, `heap_mallocn()`, `heap_setup_block()`, and `mv_show_stats()`.

#### 4.18.2.4 uint32 mm\_header::size

Referenced by `create_heap()`, `heap_contract()`, `heap_expand()`, `heap_mem_dump()`, `heap_setup_block()`, `mv_show_stats()`, `realloc()`, and `update_view()`.

The documentation for this struct was generated from the following file:

- [src/kernel/mm/mm.h \(r/r22\)](#)

## 4.19 multiboot Struct Reference

Multiboot structure.

```
#include <init.h>
```

### Data Fields

- [uint32 flags](#)
- [uint32 mem\\_lower](#)
- [uint32 mem\\_upper](#)
- [uint32 boot\\_device](#)
- [uint32 cmdline](#)
- [uint32 mods\\_count](#)
- [uint32 mods\\_addr](#)
- [uint32 num](#)
- [uint32 size](#)
- [uint32 addr](#)

- [uint32 shndx](#)
- [uint32 mmap\\_length](#)
- [uint32 mmap\\_addr](#)
- [uint32 drives\\_length](#)
- [uint32 drives\\_addr](#)
- [uint32 config\\_table](#)
- [uint32 boot\\_loader\\_name](#)
- [uint32 apm\\_table](#)
- [uint32 vbe\\_control\\_info](#)
- [uint32 vbe\\_mode\\_info](#)
- [uint32 vbe\\_mode](#)
- [uint32 vbe\\_interface\\_seg](#)
- [uint32 vbe\\_interface\\_off](#)
- [uint32 vbe\\_interface\\_len](#)

### 4.19.1 Detailed Description

Multiboot structure.

See also

<http://www.gnu.org/software/grub/manual/multiboot/multiboot.html>

### 4.19.2 Field Documentation

**4.19.2.1 uint32 multiboot::addr**

**4.19.2.2 uint32 multiboot::apm\_table**

**4.19.2.3 uint32 multiboot::boot\_device**

**4.19.2.4 uint32 multiboot::boot\_loader\_name**

**4.19.2.5 uint32 multiboot::cmdline**

**4.19.2.6 uint32 multiboot::config\_table**

**4.19.2.7 uint32 multiboot::drives\_addr**

**4.19.2.8 uint32 multiboot::drives\_length**

**4.19.2.9 uint32 multiboot::flags**

**4.19.2.10 uint32 multiboot::mem\_lower**

**4.19.2.11 uint32 multiboot::mem\_upper**

Referenced by `main()`.

- 4.19.2.12 `uint32 multiboot::mmap_addr`
- 4.19.2.13 `uint32 multiboot::mmap_length`
- 4.19.2.14 `uint32 multiboot::mods_addr`
- 4.19.2.15 `uint32 multiboot::mods_count`
- 4.19.2.16 `uint32 multiboot::num`
- 4.19.2.17 `uint32 multiboot::shndx`
- 4.19.2.18 `uint32 multiboot::size`
- 4.19.2.19 `uint32 multiboot::vbe_control_info`
- 4.19.2.20 `uint32 multiboot::vbe_interface_len`
- 4.19.2.21 `uint32 multiboot::vbe_interface_off`
- 4.19.2.22 `uint32 multiboot::vbe_interface_seg`
- 4.19.2.23 `uint32 multiboot::vbe_mode`
- 4.19.2.24 `uint32 multiboot::vbe_mode_info`

The documentation for this struct was generated from the following file:

- [src/kernel/include/init.h \(r/r22\)](#)

## 4.20 page Struct Reference

```
#include <mm_paging.h>
```

### Data Fields

- `uint32 present`: 1
- `uint32 rw`: 1
- `uint32 user`: 1
- `uint32 res`: 2
- `uint32 accessed`: 1
- `uint32 dirty`: 1
- `uint32 res2`: 2
- `uint32 avail`: 3
- `uint32 frame`: 20



## 4.20.1 Field Documentation

### 4.20.1.1 uint32 page::accessed

### 4.20.1.2 uint32 page::avail

### 4.20.1.3 uint32 page::dirty

### 4.20.1.4 uint32 page::frame

Referenced by alloc\_frame(), and free\_frame().

### 4.20.1.5 uint32 page::present

Referenced by alloc\_frame().

### 4.20.1.6 uint32 page::res

### 4.20.1.7 uint32 page::res2

### 4.20.1.8 uint32 page::rw

Referenced by alloc\_frame().

### 4.20.1.9 uint32 page::user

Referenced by alloc\_frame().

The documentation for this struct was generated from the following file:

- [src/kernel/mm/mm\\_paging.h \(r/r22\)](#)

## 4.21 page\_directory Struct Reference

```
#include <mm_paging.h>
```

### Data Fields

- [page\\_table\\_t \\* tables](#) [1024]
- [uint32 tablesPhysical](#) [1024]
- [uint32 physicalAddr](#)

## 4.21.1 Field Documentation

### 4.21.1.1 uint32 page\_directory::physicalAddr

### 4.21.1.2 page\_table\_t\* page\_directory::tables[1024]

Referenced by get\_page().

### 4.21.1.3 uint32 page\_directory::tablesPhysical[1024]

Referenced by `get_page()`.

The documentation for this struct was generated from the following file:

- [src/kernel/mm/mm\\_paging.h \(r/r22\)](#)

## 4.22 page\_table Struct Reference

```
#include <mm_paging.h>
```

### Data Fields

- [page\\_t pages](#) [1024]

### 4.22.1 Field Documentation

#### 4.22.1.1 page\_t page\_table::pages[1024]

Referenced by `get_page()`.

The documentation for this struct was generated from the following file:

- [src/kernel/mm/mm\\_paging.h \(r/r22\)](#)

## 4.23 potatoes\_dir\_entry Struct Reference

```
#include <chips.h>
```

### Data Fields

- [potatoes\\_block\\_nr](#) inode
- char [name](#) [POTATOES\_NAME\_SIZE]

### 4.23.1 Field Documentation

#### 4.23.1.1 potatoes\_block\_nr potatoes\_dir\_entry::inode

#### 4.23.1.2 char potatoes\_dir\_entry::name[POTATOES\_NAME\_SIZE]

The documentation for this struct was generated from the following file:

- [src/tools/chips/chips.h \(r/r22\)](#)

## 4.24 potatoes\_file\_info Struct Reference

```
#include <chips.h>
```

### Data Fields

- char [name](#) [POTATOES\_NAME\_SIZE]
- [potatoes\\_uint16 mode](#)
- [potatoes\\_uint32 size](#)
- [potatoes\\_time\\_t create\\_ts](#)
- [potatoes\\_time\\_t modify\\_ts](#)
- [potatoes\\_uint16 num\\_links](#)

### 4.24.1 Field Documentation

#### 4.24.1.1 potatoes\_time\_t potatoes\_file\_info::create\_ts

#### 4.24.1.2 potatoes\_uint16 potatoes\_file\_info::mode

Referenced by [chips\\_unlink\(\)](#).

#### 4.24.1.3 potatoes\_time\_t potatoes\_file\_info::modify\_ts

#### 4.24.1.4 char potatoes\_file\_info::name[POTATOES\_NAME\_SIZE]

#### 4.24.1.5 potatoes\_uint16 potatoes\_file\_info::num\_links

#### 4.24.1.6 potatoes\_uint32 potatoes\_file\_info::size

The documentation for this struct was generated from the following file:

- [src/tools/chips/chips.h \(r/r22\)](#)

## 4.25 potatoes\_time Struct Reference

```
#include <chips.h>
```

### Data Fields

- [potatoes\\_uint8 sec](#)
- [potatoes\\_uint8 min](#)
- [potatoes\\_uint8 hour](#)
- [potatoes\\_uint8 weekday](#)
- [potatoes\\_uint8 day](#)
- [potatoes\\_uint8 month](#)
- [potatoes\\_uint8 year](#)
- [potatoes\\_uint8 century](#)

### 4.25.1 Field Documentation

4.25.1.1 potatoes\_uint8 potatoes\_time::century

4.25.1.2 potatoes\_uint8 potatoes\_time::day

4.25.1.3 potatoes\_uint8 potatoes\_time::hour

4.25.1.4 potatoes\_uint8 potatoes\_time::min

4.25.1.5 potatoes\_uint8 potatoes\_time::month

4.25.1.6 potatoes\_uint8 potatoes\_time::sec

4.25.1.7 potatoes\_uint8 potatoes\_time::weekday

4.25.1.8 potatoes\_uint8 potatoes\_time::year

The documentation for this struct was generated from the following file:

- [src/tools/chips/chips.h \(r/r22\)](#)

## 4.26 PotatoesDisk Struct Reference

```
#include <chips.h>
```

### Data Fields

- unsigned char \* [data](#)
- unsigned int [size](#)

### 4.26.1 Field Documentation

4.26.1.1 unsigned char\* PotatoesDisk::data

Referenced by potatoes\_disc\_create(), potatoes\_disc\_destroy(), potatoes\_hd\_read\_sector(), and potatoes\_hd\_write\_sector().

4.26.1.2 unsigned int PotatoesDisk::size

Referenced by potatoes\_disc\_create(), potatoes\_disc\_destroy(), potatoes\_get\_hdsizes(), potatoes\_hd\_read\_sector(), and potatoes\_hd\_write\_sector().

The documentation for this struct was generated from the following file:

- [src/tools/chips/chips.h \(r/r22\)](#)

## 4.27 `proc_file` Struct Reference

The process file descriptor.

```
#include <fs_types.h>
```

### Data Fields

- [file\\_nr pf\\_desc](#)
- [file\\_nr pf\\_f\\_desc](#)
- [uint32 pf\\_pos](#)

### 4.27.1 Detailed Description

The process file descriptor.

### 4.27.2 Field Documentation

#### 4.27.2.1 `file_nr proc_file::pf_desc`

Referenced by `alloc_proc_file()`, `dump_proc_file()`, `free_proc_file()`, and `insert_proc_file()`.

#### 4.27.2.2 `file_nr proc_file::pf_f_desc`

Referenced by `do_close_pf()`, `dump_proc_file()`, `free_proc_file()`, `insert_proc_file()`, `sys_read()`, `sys_seek()`, `sys_write()`, and `test_PM()`.

#### 4.27.2.3 `uint32 proc_file::pf_pos`

Referenced by `dump_proc_file()`, `free_proc_file()`, `insert_proc_file()`, `lseek()`, `sys_read()`, `sys_seek()`, and `sys_write()`.

The documentation for this struct was generated from the following file:

- [src/kernel/fs/fs\\_types.h \(r/r22\)](#)

## 4.28 `process_t` Struct Reference

Structure describing a single process.

```
#include <pm_main.h>
```

### Data Fields

- `char * name`  
*a readable name string*
- `uint32 pid`

*process id*

- [uint8 state](#)  
*process state: running, dead, ...*
- [uint32 context](#)  
*the memory area which constitutes the stack*
- [void \\* stack\\_start](#)  
*pointer to the beginning of the memory that holds the stack*
- [void \\* addr](#)  
*memory address*
- [uint32 timeslice](#)  
*process timeslice*
- [ring\\_fifo \\* stdin](#)  
*STDIN queue.*
- [proc\\_file pft \[NUM\\_PROC\\_FILES\]](#)  
*process file table*
- [virt\\_monitor \\* vmonitor](#)  
*the virtual monitor this process is attached to*
- [struct process\\_t \\* next](#)  
*linked list next ptr*
- [uint32 priority](#)  
*number of consecutive timeslices the process is offered everytime it becomes active*
- [uint32 remaining\\_timeslices](#)  
*decremented every timer interrupt.*

### 4.28.1 Detailed Description

Structure describing a single process. TODO: Maybe move this into pm\_process.h

### 4.28.2 Field Documentation

#### 4.28.2.1 void\* process\_t::addr

memory address

#### 4.28.2.2 uint32 process\_t::context

the memory area which constitutes the stack

Referenced by pm\_create\_thread(), pm\_dump(), and pm\_schedule().

#### 4.28.2.3 char\* process\_t::name

a readable name string

Referenced by pm\_create\_thread(), pm\_destroy\_thread(), pm\_dump(), pm\_init(), snake(), speed(), and sys\_malloc().

#### 4.28.2.4 struct process\_t\* process\_t::next

linked list next ptr

Referenced by pm\_create\_thread(), pm\_dump(), pm\_get\_proc(), pm\_init(), and pm\_schedule().

#### 4.28.2.5 proc\_file process\_t::pft[NUM\_PROC\_FILES]

process file table

Referenced by pm\_create\_thread(), pm\_init(), sys\_close(), sys\_open(), sys\_read(), sys\_seek(), sys\_stat(), and sys\_write().

#### 4.28.2.6 uint32 process\_t::pid

process id

Referenced by \_getpid(), getpid(), pm\_create\_thread(), pm\_destroy\_thread(), pm\_dump(), pm\_get\_proc(), and pm\_init().

#### 4.28.2.7 uint32 process\_t::priority

number of consecutive timeslices the process is offered everytime it becomes active

Referenced by pm\_create\_thread(), pm\_dump(), pm\_init(), pm\_schedule(), and pm\_set\_thread\_priority().

#### 4.28.2.8 uint32 process\_t::remaining\_timeslices

decremented every timer interrupt.

The process becomes inactive when remaining\_timeslices is 0

Referenced by pm\_create\_thread(), pm\_init(), and pm\_schedule().

#### 4.28.2.9 void\* process\_t::stack\_start

pointer to the beginning of the memory that holds the stack

Referenced by pm\_create\_thread(), and pm\_destroy\_thread().

#### 4.28.2.10 uint8 process\_t::state

process state: running, dead, ...

Referenced by dev\_stdin\_read(), pm\_create\_thread(), pm\_handle\_input(), pm\_init(), pm\_kill\_proc(), pm\_schedule(), and sys\_exit().

#### 4.28.2.11 ring\_fifo\* process\_t::stdin

STDIN queue.

Referenced by `dev_stdin_read()`, `dev_stdin_write()`, `pm_create_thread()`, `pm_destroy_thread()`, `pm_handle_input()`, and `pm_init()`.

#### 4.28.2.12 uint32 process\_t::timeslice

process timeslice

#### 4.28.2.13 virt\_monitor\* process\_t::vmonitor

the virtual monitor this process is attached to

Referenced by `aprintf()`, `dev_stdout_write()`, `free_virt_monitor()`, `pm_create_thread()`, `pm_destroy_thread()`, and `pm_init()`.

The documentation for this struct was generated from the following file:

- [src/kernel/pm/pm\\_main.h \(r/r22\)](#)

## 4.29 ring\_fifo Struct Reference

A static circular FIFO buffer.

```
#include <ringbuffer.h>
```

### Data Fields

- [uint8 \\* data](#)  
*the data pointer*
- [uint32 size](#)  
*size of data (max value of len)*
- [uint32 start](#)  
*read position*
- [uint32 end](#)  
*write position*
- [uint32 len](#)  
*number of used bytes*

#### 4.29.1 Detailed Description

A static circular FIFO buffer.



## 4.29.2 Field Documentation

### 4.29.2.1 `uint8* ring_fifo::data`

the data pointer

Referenced by `rf_alloc()`, `rf_copy()`, `rf_dump()`, `rf_free()`, `rf_read()`, and `rf_write()`.

### 4.29.2.2 `uint32 ring_fifo::end`

write position

Referenced by `rf_clear()`, `rf_copy()`, `rf_dump()`, and `rf_write()`.

### 4.29.2.3 `uint32 ring_fifo::len`

number of used bytes

Referenced by `rf_clear()`, `rf_copy()`, `rf_dump()`, `rf_getlength()`, `rf_isempty()`, `rf_isfull()`, `rf_read()`, and `rf_write()`.

### 4.29.2.4 `uint32 ring_fifo::size`

size of data (max value of len)

Referenced by `rf_alloc()`, `rf_copy()`, `rf_dump()`, `rf_isfull()`, `rf_read()`, and `rf_write()`.

### 4.29.2.5 `uint32 ring_fifo::start`

read position

Referenced by `rf_clear()`, `rf_copy()`, `rf_dump()`, and `rf_read()`.

The documentation for this struct was generated from the following file:

- `src/kernel/include/ringbuffer.h` ([r/r22](#))

## 4.30 `sc_close_args_t` Struct Reference

Arguments for the CLOSE syscall.

```
#include <syscalls_shared.h>
```

### Data Fields

- `int success`
- `int fd`

### 4.30.1 Detailed Description

Arguments for the CLOSE syscall.

## 4.30.2 Field Documentation

### 4.30.2.1 `int sc_close_args_t::fd`

Referenced by `_close()`, and `sys_close()`.

### 4.30.2.2 `int sc_close_args_t::success`

Referenced by `_close()`, and `sys_close()`.

The documentation for this struct was generated from the following file:

- [src/kernel/pm/syscalls\\_shared.h \(r/r22\)](#)

## 4.31 `sc_malloc_args_t` Struct Reference

Arguments for the MALLOC syscall.

```
#include <syscalls_shared.h>
```

### Data Fields

- `void * mem`
- `int size`

### 4.31.1 Detailed Description

Arguments for the MALLOC syscall.

### 4.31.2 Field Documentation

#### 4.31.2.1 `void* sc_malloc_args_t::mem`

Referenced by `_malloc()`, and `sys_malloc()`.

#### 4.31.2.2 `int sc_malloc_args_t::size`

Referenced by `_malloc()`, and `sys_malloc()`.

The documentation for this struct was generated from the following file:

- [src/kernel/pm/syscalls\\_shared.h \(r/r22\)](#)

## 4.32 `sc_open_args_t` Struct Reference

Arguments for the OPEN syscall.

```
#include <syscalls_shared.h>
```

## Data Fields

- int `fd`
- char \* `path`
- int `oflag`
- int `mode`

### 4.32.1 Detailed Description

Arguments for the OPEN syscall.

### 4.32.2 Field Documentation

#### 4.32.2.1 int `sc_open_args_t::fd`

Referenced by `_open()`, and `sys_open()`.

#### 4.32.2.2 int `sc_open_args_t::mode`

Referenced by `_open()`, and `sys_open()`.

#### 4.32.2.3 int `sc_open_args_t::oflag`

Referenced by `_open()`, and `sys_open()`.

#### 4.32.2.4 char\* `sc_open_args_t::path`

Referenced by `_open()`, and `sys_open()`.

The documentation for this struct was generated from the following file:

- `src/kernel/pm/syscalls_shared.h` ([r/r22](#))

## 4.33 `sc_read_write_args_t` Struct Reference

Arguments for the READ and WRITE syscalls.

```
#include <syscalls_shared.h>
```

## Data Fields

- int `rw_count`
- int `fd`
- void \* `buf`
- int `size`

### 4.33.1 Detailed Description

Arguments for the READ and WRITE syscalls.

### 4.33.2 Field Documentation

#### 4.33.2.1 `void* sc_read_write_args_t::buf`

Referenced by `_read()`, `_write()`, `sys_read()`, and `sys_write()`.

#### 4.33.2.2 `int sc_read_write_args_t::fd`

Referenced by `_read()`, `_write()`, `sys_read()`, and `sys_write()`.

#### 4.33.2.3 `int sc_read_write_args_t::rw_count`

Referenced by `_read()`, `_write()`, `sys_read()`, and `sys_write()`.

#### 4.33.2.4 `int sc_read_write_args_t::size`

Referenced by `_read()`, `_write()`, `sys_read()`, and `sys_write()`.

The documentation for this struct was generated from the following file:

- [src/kernel/pm/syscalls\\_shared.h \(r/r22\)](#)

## 4.34 `sc_seek_args_t` Struct Reference

Arguments for the SEEK syscall.

```
#include <syscalls_shared.h>
```

### Data Fields

- `int pos`
- `int fd`
- `int offset`
- `int whence`

### 4.34.1 Detailed Description

Arguments for the SEEK syscall.

### 4.34.2 Field Documentation

#### 4.34.2.1 `int sc_seek_args_t::fd`

Referenced by `_seek()`, and `sys_seek()`.

#### 4.34.2.2 `int sc_seek_args_t::offset`

Referenced by `_seek()`, and `sys_seek()`.

#### 4.34.2.3 `int sc_seek_args_t::pos`

Referenced by `_seek()`, and `sys_seek()`.

#### 4.34.2.4 `int sc_seek_args_t::whence`

Referenced by `_seek()`, and `sys_seek()`.

The documentation for this struct was generated from the following file:

- [src/kernel/pm/syscalls\\_shared.h \(r/r22\)](#)

## 4.35 `sc_stat_args_t` Struct Reference

Arguments for the STAT syscall.

```
#include <syscalls_shared.h>
```

### Data Fields

- `int success`
- `stat * buf`
- `char * path`

### 4.35.1 Detailed Description

Arguments for the STAT syscall.

### 4.35.2 Field Documentation

#### 4.35.2.1 `stat* sc_stat_args_t::buf`

Referenced by `_stat()`, and `sys_stat()`.

#### 4.35.2.2 `char* sc_stat_args_t::path`

Referenced by `_stat()`, and `sys_stat()`.

#### 4.35.2.3 `int sc_stat_args_t::success`

Referenced by `_stat()`, and `sys_stat()`.

The documentation for this struct was generated from the following file:

- [src/kernel/pm/syscalls\\_shared.h \(r/r22\)](#)

## 4.36 `sc_unlink_args_t` Struct Reference

Arguments for the UNLINK syscall.

```
#include <syscalls_shared.h>
```

### Data Fields

- int [success](#)
- char \* [path](#)

### 4.36.1 Detailed Description

Arguments for the UNLINK syscall.

### 4.36.2 Field Documentation

#### 4.36.2.1 `char* sc_unlink_args_t::path`

Referenced by `_unlink()`, and `sys_unlink()`.

#### 4.36.2.2 `int sc_unlink_args_t::success`

Referenced by `_unlink()`, and `sys_unlink()`.

The documentation for this struct was generated from the following file:

- [src/kernel/pm/syscalls\\_shared.h \(r/r22\)](#)

## 4.37 Semaphore Struct Reference

### Data Fields

- int [lock](#)
- int [count](#)

### 4.37.1 Field Documentation

#### 4.37.1.1 `int Semaphore::count`

Referenced by `p()`, `sema_init()`, and `v()`.

#### 4.37.1.2 `int Semaphore::lock`

Referenced by `p()`, `sema_init()`, and `v()`.

The documentation for this struct was generated from the following file:

- [src/kernel/init/tests.c \(r/r22\)](#)

## 4.38 shell\_cmd\_t Struct Reference

A single shell commmand.

```
#include <shell_main.h>
```

### Data Fields

- char [name](#) [16]
- [shell\\_cmd\\_func](#) cmd
- char [desc](#) [100]

### 4.38.1 Detailed Description

A single shell commmand.

### 4.38.2 Field Documentation

#### 4.38.2.1 shell\_cmd\_func shell\_cmd\_t::cmd

Referenced by [shell\\_autocomplete\(\)](#), [shell\\_handle\\_command\(\)](#), and [shell\\_main\(\)](#).

#### 4.38.2.2 char shell\_cmd\_t::desc[100]

Referenced by [shell\\_cmd\\_cmdlist\(\)](#).

#### 4.38.2.3 char shell\_cmd\_t::name[16]

Referenced by [shell\\_autocomplete\(\)](#), and [shell\\_cmd\\_cmdlist\(\)](#).

The documentation for this struct was generated from the following file:

- [src/apps/shell\\_main.h \(r/r22\)](#)

## 4.39 shortcut Struct Reference

Structure of a shortcut.

```
#include <io_keyboard.h>
```

### Data Fields

- void(\* [func](#) )()
- [bool](#) control
- [bool](#) super
- char [ch](#)

### 4.39.1 Detailed Description

Structure of a shortcut.

### 4.39.2 Field Documentation

#### 4.39.2.1 char shortcut::ch

Referenced by `add_shortcut()`.

#### 4.39.2.2 bool shortcut::control

Referenced by `add_shortcut()`.

#### 4.39.2.3 void(\* shortcut::func)()

Referenced by `add_shortcut()`, and `kb_handler()`.

#### 4.39.2.4 bool shortcut::super

Referenced by `add_shortcut()`.

The documentation for this struct was generated from the following file:

- [src/kernel/io/io\\_keyboard.h \(r/r22\)](#)

## 4.40 super\_block Struct Reference

```
#include <fs_super.h>
```

### Data Fields

- [block\\_nr s\\_HD\\_size](#)
- [uint16 s\\_bmap\\_blocks](#)
- [block\\_nr s\\_first\\_data\\_block](#)
- [uint32 s\\_max\\_file\\_size](#)
- [uint8 \\* s\\_bmap](#)
- [m\\_inode \\* s\\_iroot](#)
- [time\\_t s\\_modify\\_ts](#)
- [bool s\\_read\\_only](#)
- [uint16 s\\_dirt](#)
- [uint32 s\\_magic\\_number](#)

### 4.40.1 Field Documentation

#### 4.40.1.1 uint8\* super\_block::s\_bmap

Referenced by `dump_super()`, and `init_super_block()`.



**4.40.1.2 uint16 super\_block::s\_bmap\_blocks**

Referenced by `dump_super()`, and `init_super_block()`.

**4.40.1.3 uint16 super\_block::s\_dirt**

Referenced by `init_super_block()`.

**4.40.1.4 block\_nr super\_block::s\_first\_data\_block**

Referenced by `dump_super()`, and `init_super_block()`.

**4.40.1.5 block\_nr super\_block::s\_HD\_size**

Referenced by `dump_super()`, and `init_super_block()`.

**4.40.1.6 m\_inode\* super\_block::s\_iroot**

Referenced by `dump_super()`, and `init_super_block()`.

**4.40.1.7 uint32 super\_block::s\_magic\_number**

Referenced by `dump_super()`, and `init_super_block()`.

**4.40.1.8 uint32 super\_block::s\_max\_file\_size**

Referenced by `dump_super()`, and `init_super_block()`.

**4.40.1.9 time\_t super\_block::s\_modify\_ts**

Referenced by `init_super_block()`.

**4.40.1.10 bool super\_block::s\_read\_only**

Referenced by `init_super_block()`.

The documentation for this struct was generated from the following file:

- [src/kernel/fs/fs\\_super.h \(r/r22\)](#)

## 4.41 time Struct Reference

Global time struct.

```
#include <io_rtc.h>
```

## Data Fields

- [uint8 sec](#)
- [uint8 min](#)
- [uint8 hour](#)
- [uint8 weekday](#)
- [uint8 day](#)
- [uint8 month](#)
- [uint8 year](#)
- [uint8 century](#)

### 4.41.1 Detailed Description

Global time struct.

### 4.41.2 Field Documentation

#### 4.41.2.1 `uint8 time::century`

Referenced by `time2str()`.

#### 4.41.2.2 `uint8 time::day`

Referenced by `calculate_weekday()`, and `time2str()`.

#### 4.41.2.3 `uint8 time::hour`

Referenced by `time2str()`.

#### 4.41.2.4 `uint8 time::min`

Referenced by `time2str()`.

#### 4.41.2.5 `uint8 time::month`

Referenced by `calculate_weekday()`, and `time2str()`.

#### 4.41.2.6 `uint8 time::sec`

Referenced by `time2str()`.

#### 4.41.2.7 `uint8 time::weekday`

Referenced by `calculate_weekday()`, and `time2str()`.

#### 4.41.2.8 uint8 time::year

Referenced by `calculate_weekday()`, and `time2str()`.

The documentation for this struct was generated from the following file:

- [src/kernel/io/io\\_rtc.h \(r/r22\)](#)

## 4.42 Tone Struct Reference

Stores a frequency and a duration.

### Data Fields

- int [frequency](#)
- int [duration](#)

#### 4.42.1 Detailed Description

Stores a frequency and a duration.

#### 4.42.2 Field Documentation

##### 4.42.2.1 int Tone::duration

Referenced by `shell_cmd_synth()`.

##### 4.42.2.2 int Tone::frequency

The documentation for this struct was generated from the following file:

- [src/apps/synthesizer.c \(r/r22\)](#)

## 4.43 virt\_monitor Struct Reference

Structure that represents a virtual monitor.

```
#include <io_virtual.h>
```

### Data Fields

- [uint16 \\* begin](#)  
*Pointer to the start of the allocated memory.*
- [char \\* name](#)  
*VMonitor name.*

- **uint32 size**  
*Size of the allocated memory.*
- **uint16 \* vis\_begin**  
*Pointer to the start of the visible pane.*
- **uint32 offset**  
*Position of the cursor on the visible pane.*
- **uint32 scrolldown\_limit**  
*Number of lines beneath the visible pane.*
- **uint32 scrollup\_limit**  
*Number of lines above the visible pane.*
- **bool disable\_refresh**  
*For framebuffer access.*
- **uint32 pid**  
*The PID of the process that owns this virtual monitor.*

### 4.43.1 Detailed Description

Structure that represents a virtual monitor.

### 4.43.2 Field Documentation

#### 4.43.2.1 uint16\* virt\_monitor::begin

Pointer to the start of the allocated memory.

Referenced by `free_virt_monitor()`, `new_virt_monitor()`, `update_virt_monitor()`, `virt_monitor_cputc()`, `virt_monitor_scrolldown()`, and `virt_monitor_scrollup()`.

#### 4.43.2.2 bool virt\_monitor::disable\_refresh

For framebuffer access.

If this is false, the virtual monitor will not be painted.

Referenced by `dev_framebuffer_close()`, `dev_framebuffer_open()`, `new_virt_monitor()`, `shell_cmd_snapshot()`, `update_virt_monitor()`, and `virt_cursor_move()`.

#### 4.43.2.3 char\* virt\_monitor::name

VMonitor name.

Referenced by `free_virt_monitor()`, `get_active_virt_monitor_name()`, `new_virt_monitor()`, and `start_vmonitor()`.

#### 4.43.2.4 uint32 virt\_monitor::offset

Position of the cursor on the visible pane.

Referenced by `new_virt_monitor()`, `update_virt_monitor()`, `virt_cursor_move()`, and `virt_monitor_cputc()`.

#### 4.43.2.5 uint32 virt\_monitor::pid

The PID of the process that owns this virtual monitor.

Referenced by `free_virt_monitor()`, and `new_virt_monitor()`.

#### 4.43.2.6 uint32 virt\_monitor::scrolldown\_limit

Number of lines beneath the visible pane.

Referenced by `new_virt_monitor()`, `virt_monitor_scrolldown()`, and `virt_monitor_scrollup()`.

#### 4.43.2.7 uint32 virt\_monitor::scrollup\_limit

Number of lines above the visible pane.

Referenced by `new_virt_monitor()`, `virt_monitor_cputc()`, `virt_monitor_scrolldown()`, and `virt_monitor_scrollup()`.

#### 4.43.2.8 uint32 virt\_monitor::size

Size of the allocated memory.

Referenced by `new_virt_monitor()`, `update_virt_monitor()`, `virt_monitor_cputc()`, `virt_monitor_scrolldown()`, and `virt_monitor_scrollup()`.

#### 4.43.2.9 uint16\* virt\_monitor::vis\_begin

Pointer to the start of the visible pane.

Referenced by `new_virt_monitor()`, `update_virt_monitor()`, `virt_monitor_cputc()`, `virt_monitor_invert()`, `virt_monitor_scrolldown()`, and `virt_monitor_scrollup()`.

The documentation for this struct was generated from the following file:

- [src/kernel/io/io\\_virtual.h \(r/r22\)](#)



# Chapter 5

## File Documentation

### 5.1 src/apps/apps.h File Reference

Header file for the applications.

#### Functions

- void [shell\\_cmd\\_snapshot](#) (int argc, char \*argv[ ])
- void [shell\\_cmd\\_speed](#) (int argc, char \*argv[ ])
- void [shell\\_cmd\\_memview](#) (int argc, char \*argv[ ])

*Shell command wrapper for the memview tool.*

#### 5.1.1 Detailed Description

Header file for the applications.

##### Author

Dmitriy Traytel

##### LastChangedBy:

dbader

##### Version

##### Rev:

16

#### 5.1.2 Function Documentation

##### 5.1.2.1 void `shell_cmd_memview` ( int *argc*, char \* *argv*[ ] )

Shell command wrapper for the memview tool.

References `memview_main()`, and `pm_create_thread()`.

### 5.1.2.2 `void shell_cmd_snapshot ( int argc, char * argv[] )`

References `_close()`, `_open()`, `_printf()`, `_read()`, `virt_monitor::disable_refresh`, `ESCAPE`, `get_active_virt_monitor()`, `keydown()`, `shell_makepath()`, `STDIN`, `STDOUT`, and `VGA_DISPLAY`.

### 5.1.2.3 `void shell_cmd_speed ( int argc, char * argv[] )`

References `_close()`, `_open()`, `_printf()`, `O_CREAT`, `pm_create_thread()`, `shell_makepath()`, and `speed()`.

## 5.2 `src/apps/brainfuck_interpreter.c` File Reference

Interprets brainfuck source code.

```
#include "../kernel/include/types.h"
#include "../kernel/include/const.h"
#include "../kernel/include/string.h"
#include "../kernel/include/debug.h"
#include "../kernel/include/assert.h"
#include "../kernel/io/io.h"
#include "../kernel/pm/syscalls_cli.h"
#include "../kernel/pm/syscalls_shared.h"
#include "brainfuck_interpreter.h"
```

### Functions

- void `init_bf ()`
- void `reset_bf ()`
- void `interpret_bf (char ch)`

### Variables

- int `STDERR = 1`
- int `maxlength = 10000`
- const int `maxloopdepth = 50`
- char \* `bf_start`
- char \* `bf_ptr`
- char \* `bf_buffer = NULL`
- int `bf_buf_offset = 0`
- int `bf_buf_position = -1`
- int `loop_marks [50][2]`
- bool `jumpover = FALSE`
- bool `store_lock = FALSE`
- int `loopdepth = 0`



- int `temploopdepth` = 0
- int `store_lock_level` = 0

### 5.2.1 Detailed Description

Interprets brainfuck source code.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.2.2 Function Documentation

#### 5.2.2.1 void `init_bf` ( )

References `_close()`, `_malloc()`, `_open()`, `_write()`, `ASSERT`, `bf_ptr`, `bf_start`, `maxlength`, `NULL`, `O_CREAT`, and `strlen`.

Referenced by `pm_init()`.

#### 5.2.2.2 void `interpret_bf` ( char *ch* )

References `_free()`, `_malloc()`, `_read()`, `_write()`, `ASSERT`, `bf_buf_offset`, `bf_buf_position`, `bf_buffer`, `bf_ptr`, `bf_start`, `end`, `halt()`, `interpret_bf()`, `jumpover`, `loop_marks`, `loopdepth`, `maxlength`, `memcpy`, `NULL`, `start`, `STDERR`, `store_lock`, `store_lock_level`, `temploopdepth`, and `TRUE`.

Referenced by `dev_brainfuck_read()`, `dev_brainfuck_write()`, and `interpret_bf()`.

#### 5.2.2.3 void `reset_bf` ( )

References `_free()`, `_malloc()`, `ASSERT`, `bf_buf_offset`, `bf_buf_position`, `bf_buffer`, `bf_ptr`, `bf_start`, `jumpover`, `loopdepth`, `maxlength`, `NULL`, `store_lock`, `store_lock_level`, and `temploopdepth`.

Referenced by `shell_cmd_bf()`.

### 5.2.3 Variable Documentation

#### 5.2.3.1 int `bf_buf_offset` = 0

Referenced by `interpret_bf()`, and `reset_bf()`.

**5.2.3.2 int bf\_buf\_position = -1**

Referenced by `interpret_bf()`, and `reset_bf()`.

**5.2.3.3 char\* bf\_buffer = NULL**

Referenced by `interpret_bf()`, and `reset_bf()`.

**5.2.3.4 char\* bf\_ptr**

Referenced by `init_bf()`, `interpret_bf()`, and `reset_bf()`.

**5.2.3.5 char\* bf\_start**

Referenced by `init_bf()`, `interpret_bf()`, and `reset_bf()`.

**5.2.3.6 bool jumcover = FALSE**

Referenced by `interpret_bf()`, and `reset_bf()`.

**5.2.3.7 int loop\_marks[50][2]**

Referenced by `interpret_bf()`.

**5.2.3.8 int loopdepth = 0**

Referenced by `interpret_bf()`, and `reset_bf()`.

**5.2.3.9 int maxlength = 10000**

Referenced by `init_bf()`, `interpret_bf()`, and `reset_bf()`.

**5.2.3.10 const int maxloopdepth = 50****5.2.3.11 int STDERR = 1**

Referenced by `interpret_bf()`.

**5.2.3.12 bool store\_lock = FALSE**

Referenced by `interpret_bf()`, and `reset_bf()`.

**5.2.3.13 int store\_lock\_level = 0**

Referenced by `interpret_bf()`, and `reset_bf()`.

#### 5.2.3.14 int temploopdepth = 0

Referenced by `interpret_bf()`, and `reset_bf()`.

## 5.3 src/apps/brainfuck\_interpreter.h File Reference

Header file for the brainfuck interpreter.

### Defines

- `#define BRAINFUCK 5`

### Functions

- void `interpret_bf` (char ch)
- void `init_bf` ()
- void `reset_bf` ()

#### 5.3.1 Detailed Description

Header file for the brainfuck interpreter.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

#### 5.3.2 Define Documentation

##### 5.3.2.1 `#define BRAINFUCK 5`

#### 5.3.3 Function Documentation

##### 5.3.3.1 void `init_bf` ( )

References `_close()`, `_malloc()`, `_open()`, `_write()`, `ASSERT`, `bf_ptr`, `bf_start`, `maxlength`, `NULL`, `O_CREAT`, and `strlen`.

Referenced by `pm_init()`.

### 5.3.3.2 void interpret\_bf ( char ch )

References `_free()`, `_malloc()`, `_read()`, `_write()`, `ASSERT`, `bf_buf_offset`, `bf_buf_position`, `bf_buffer`, `bf_ptr`, `bf_start`, `end`, `halt()`, `interpret_bf()`, `jumpover`, `loop_marks`, `loopdepth`, `maxlength`, `memcpy`, `NULL`, `start`, `STDERR`, `store_lock`, `store_lock_level`, `temploopdepth`, and `TRUE`.

Referenced by `dev_brainfuck_read()`, `dev_brainfuck_write()`, and `interpret_bf()`.

### 5.3.3.3 void reset\_bf ( )

## 5.4 src/apps/editor.c File Reference

Simple Potatoes/Etios EDitor.

```
#include "../kernel/pm/syscalls_cli.h"
#include "../kernel/pm/syscalls_shared.h"
#include "../kernel/pm/pm_main.h"
#include "shell_utils.h"
#include "games.h"
#include "editor.h"
#include "../kernel/include/const.h"
#include "../kernel/include/string.h"
```

### Functions

- char \* [resize\\_buf](#) (int newsize, char \*buf)
- void [speed](#) ()
- void [shell\\_cmd\\_speed](#) (int argc, char \*argv[ ])

### 5.4.1 Detailed Description

Simple Potatoes/Etios EDitor.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

## 5.4.2 Function Documentation

### 5.4.2.1 `char* resize_buf ( int newsize, char * buf )`

References `_free()`, `_malloc()`, `bzero`, and `memcpy`.

Referenced by `speed()`.

### 5.4.2.2 `void shell_cmd_speed ( int argc, char * argv[] )`

References `_close()`, `_open()`, `_printf()`, `O_CREAT`, `pm_create_thread()`, `shell_makepath()`, and `speed()`.

### 5.4.2.3 `void speed ( )`

References `_close()`, `_exit()`, `_fgetch()`, `_free()`, `_malloc()`, `_open()`, `_printf()`, `_read()`, `_seek()`, `_unlink()`, `_write()`, `active_proc`, `bzero`, `ESCAPE`, `keydown()`, `process_t::name`, `line::next`, `NEXT_LINE`, `line::num_`, `chars`, `O_CREAT`, `line::offset`, `line::prev`, `PREV_LINE`, `resize_buf()`, `SEEK_END`, `SEEK_SET`, `size`, and `strlen`.

Referenced by `shell_cmd_speed()`.

## 5.5 src/apps/editor.h File Reference

Simple Potatoes/Etios EDitor header.

```
#include "../kernel/include/debug.h"
```

### Data Structures

- struct [line](#)

### Defines

- #define [NEXT\\_LINE](#)
- #define [PREV\\_LINE](#)
- #define [DUMP\\_LINES](#)

### Typedefs

- typedef struct [line](#) [line](#)

### 5.5.1 Detailed Description

Simple Potatoes/Etios EDitor header.

#### Author

Dmitriy Traytel

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

**5.5.2 Define Documentation****5.5.2.1 #define DUMP\_LINES****Value:**

```
dprintf("NUM_CHARS\tOFFSET\t\tTYPE\n");\
for(line* temp = startline; temp != NULL; temp = temp->next) {\
    dprintf("%d\t\t%d\t\t%s\n",\
        temp->num_chars,\
        temp->offset,\
        (temp->prev->linewidth == 80)? "\\n" : "\\t");\
}
```

**5.5.2.2 #define NEXT\_LINE****Value:**

```
actualline->next = _malloc(sizeof(line));\
if(str[pos] == '\n') {\
    actualline->linewidth = 80;\
    if(actualline->prev->linewidth == 8) {\
        actualline->offset = actualline->prev->offset;\
    }\
} else {\
    actualline->linewidth = 8;\
    actualline->offset = actualline->prev->offset + actualline->num_c\
hars + 8 - actualline->num_chars % 8;\
}\
actualline->next->prev = actualline;\
actualline = actualline->next;\
actualline->num_chars = 0;\
actualline->offset = 0;\
actualline->next = NULL;
```

Referenced by speed().

**5.5.2.3 #define PREV\_LINE****Value:**

```
line *temp = actualline;\
actualline = actualline->prev;\
actualline->next = temp->next;\
if(temp->next!=NULL) {\
    temp->next->prev = actualline;\
}
```

```
    }\n    _free(temp);\n    if(actualline->linewidth == 8) {\n        actualline->offset=0;\n    }\n    for(int i = 0;\n        i< actualline->linewidth - actualline->offset - (actualline->num_chars %\nactualline->linewidth); i++) {\n        _printf("\\b");\n    }\n}
```

Referenced by speed().

### 5.5.3 Typedef Documentation

#### 5.5.3.1 typedef struct line line

## 5.6 src/apps/games.h File Reference

Header file for the games.

```
#include "../kernel/include/types.h"\n#include "../kernel/io/io_sound.h"\n#include "../kernel/io/io_sound.h"
```

### Defines

- #define [CURSOR\\_UP](#) 0x48
- #define [CURSOR\\_DOWN](#) 0x50
- #define [CURSOR\\_LEFT](#) 0x4B
- #define [CURSOR\\_RIGHT](#) 0x4D
- #define [ESCAPE](#) 0x01
- #define [ENTER](#) 0x1C
- #define [SHARP](#) 0x2B
- #define [KEY\\_Q](#) 0x10
- #define [KEY\\_W](#) 0x11
- #define [KEY\\_E](#) 0x12
- #define [KEY\\_R](#) 0x13
- #define [KEY\\_T](#) 0x14
- #define [KEY\\_Z](#) 0x15
- #define [KEY\\_U](#) 0x16
- #define [KEY\\_I](#) 0x17
- #define [KEY\\_A](#) 0x1E
- #define [KEY\\_S](#) 0x1F
- #define [KEY\\_D](#) 0x20
- #define [KEY\\_F](#) 0x21
- #define [KEY\\_G](#) 0x22
- #define [KEY\\_H](#) 0x23
- #define [KEY\\_J](#) 0x24
- #define [KEY\\_K](#) 0x25
- #define [KEY\\_L](#) 0x26

- #define `KEY_PLUS` 0x1B
- #define `KEY_MINUS` 0x35
- #define `SET_PIXEL`(x, y, cl) `bbuf[(y)*80+(x)] = cl;`
- #define `DRAW_PADDLE`(x, y, cl)
- #define `LIMIT`(x, min, max) `if (x<min) x = min; if (x>max) x = max;`
- #define `HIT_PADDLE`(paddley, y) `(paddley <= (y) && paddley + 5 >= (y))`
- #define `PADDLE_DEFLECTION`(paddley, hit) `(paddley - (hit)) * 10`
- #define `HIT_RPADDLE_SOUND` `NOTE_B4`
- #define `HIT_LPADDLE_SOUND` `NOTE_C4`
- #define `HIT_SIDE_SOUND` `NOTE_E4`
- #define `DRAW_GLYPH`(x, y, idx, cl)
- #define `SNAKE_RIGHT` 1
- #define `SNAKE_LEFT` 2
- #define `SNAKE_UP` 3
- #define `SNAKE_DOWN` 4
- #define `EAT_APPLE_SOUND` `NOTE_C5`

## Typedefs

- typedef `uint8` `glyph_t` [4 \*5]

## Functions

- void `shell_cmd_pong` (int argc, char \*argv[ ])
- void `shell_cmd_snake` (int argc, char \*argv[ ])
- void `shell_cmd_synth` (int argc, char \*argv[ ])
- bool `keydown` (char key, int fd)

### 5.6.1 Detailed Description

Header file for the games.

#### Author

Daniel Bader  
Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12



## 5.6.2 Define Documentation

### 5.6.2.1 #define CURSOR\_DOWN 0x50

Referenced by kb\_handler(), shell\_cmd\_pong(), and snake().

### 5.6.2.2 #define CURSOR\_LEFT 0x4B

Referenced by kb\_handler(), and snake().

### 5.6.2.3 #define CURSOR\_RIGHT 0x4D

Referenced by kb\_handler(), and snake().

### 5.6.2.4 #define CURSOR\_UP 0x48

Referenced by kb\_handler(), shell\_cmd\_pong(), and snake().

### 5.6.2.5 #define DRAW\_GLYPH( x, y, idx, cl )

#### Value:

```
for (int px = 0; px < sizeof(glyph_t); px++) \
    SET_PIXEL(x + px % 4, y + px / 4, (font[idx][px] ? cl : 0));
```

Referenced by shell\_cmd\_pong().

### 5.6.2.6 #define DRAW\_PADDLE( x, y, cl )

#### Value:

```
SET_PIXEL(x, y, cl); SET_PIXEL(x, y+1, cl); SET_PIXEL(x, y+2, cl); \
    SET_PIXEL(x, y+3, cl); SET_PIXEL(x, y+4, cl);
```

Referenced by shell\_cmd\_pong().

### 5.6.2.7 #define EAT\_APPLE\_SOUND NOTE\_C5

Referenced by snake().

### 5.6.2.8 #define ENTER 0x1C

### 5.6.2.9 #define ESCAPE 0x01

Referenced by kb\_handler(), memview\_main(), shell\_cmd\_pong(), shell\_cmd\_snapshot(), snake(), speed(), and synth().

### 5.6.2.10 #define HIT\_LPADDLE\_SOUND NOTE\_C4

Referenced by shell\_cmd\_pong().

**5.6.2.11 #define HIT\_PADDLE( *paddley*, *y* ) (paddley <= (*y*) && paddley + 5 >= (*y*))**

Referenced by shell\_cmd\_pong().

**5.6.2.12 #define HIT\_RPADDLE\_SOUND NOTE\_B4**

Referenced by shell\_cmd\_pong().

**5.6.2.13 #define HIT\_SIDE\_SOUND NOTE\_E4**

Referenced by shell\_cmd\_pong().

**5.6.2.14 #define KEY\_A 0x1E**

Referenced by shell\_cmd\_pong(), snake(), and synth().

**5.6.2.15 #define KEY\_D 0x20**

Referenced by snake(), and synth().

**5.6.2.16 #define KEY\_E 0x12**

Referenced by synth().

**5.6.2.17 #define KEY\_F 0x21**

Referenced by synth().

**5.6.2.18 #define KEY\_G 0x22**

Referenced by synth().

**5.6.2.19 #define KEY\_H 0x23**

Referenced by synth().

**5.6.2.20 #define KEY\_I 0x17**

Referenced by snake().

**5.6.2.21 #define KEY\_J 0x24**

Referenced by snake(), and synth().

**5.6.2.22 #define KEY\_K 0x25**

Referenced by shell\_cmd\_pong(), snake(), and synth().

**5.6.2.23 #define KEY\_L 0x26**

Referenced by shell\_cmd\_pong(), and snake().

**5.6.2.24 #define KEY\_MINUS 0x35**

Referenced by synth().

**5.6.2.25 #define KEY\_PLUS 0x1B**

Referenced by synth().

**5.6.2.26 #define KEY\_Q 0x10****5.6.2.27 #define KEY\_R 0x13****5.6.2.28 #define KEY\_S 0x1F**

Referenced by shell\_cmd\_pong(), snake(), and synth().

**5.6.2.29 #define KEY\_T 0x14**

Referenced by synth().

**5.6.2.30 #define KEY\_U 0x16**

Referenced by synth().

**5.6.2.31 #define KEY\_W 0x11**

Referenced by snake(), and synth().

**5.6.2.32 #define KEY\_Z 0x15**

Referenced by synth().

**5.6.2.33 #define LIMIT( *x*, *min*, *max* ) if (x<min) x = min; if (x>max) x = max;**

Referenced by shell\_cmd\_pong().

**5.6.2.34 #define PADDLE\_DEFLECTION( *paddley*, *hit* ) (paddley - (hit)) \* 10**

Referenced by shell\_cmd\_pong().

**5.6.2.35** `#define SET_PIXEL( x, y, cl ) bbuf[(y)*80+(x)] = cl;`

Referenced by `draw_snake()`, `shell_cmd_pong()`, and `snake()`.

**5.6.2.36** `#define SHARP 0x2B`

**5.6.2.37** `#define SNAKE_DOWN 4`

Referenced by `snake()`.

**5.6.2.38** `#define SNAKE_LEFT 2`

Referenced by `snake()`.

**5.6.2.39** `#define SNAKE_RIGHT 1`

Referenced by `snake()`.

**5.6.2.40** `#define SNAKE_UP 3`

Referenced by `snake()`.

## 5.6.3 Typedef Documentation

**5.6.3.1** `typedef uint8 glyph_t[4 *5]`

## 5.6.4 Function Documentation

**5.6.4.1** `bool keydown ( char key, int fd )`

Referenced by `memview_main()`, `shell_cmd_pong()`, `shell_cmd_snapshot()`, `snake()`, `speed()`, and `synth()`.

**5.6.4.2** `void shell_cmd_pong ( int argc, char * argv[] )`

References `_close()`, `_fgetch()`, `_open()`, `_printf()`, `_read()`, `_write()`, `BLACK`, `BLUE`, `CURSOR_DOWN`, `CURSOR_UP`, `DRAW_GLYPH`, `DRAW_PADDLE`, `end_beep()`, `ESCAPE`, `halt()`, `HIT_LPADDLE_SOUND`, `HIT_PADDLE`, `HIT_HPADDLE_SOUND`, `HIT_SIDE_SOUND`, `KEY_A`, `KEY_K`, `KEY_L`, `KEY_S`, `keydown()`, `LIMIT`, `memset()`, `PADDLE_DEFLECTION`, `RED`, `SET_PIXEL`, `start_beep()`, `STDIN`, `strcmp`, `TRUE`, `WHITE`, and `YELLOW`.

**5.6.4.3** `void shell_cmd_snake ( int argc, char * argv[] )`

References `_free()`, `_malloc()`, `name`, `pm_create_thread()`, `snake()`, `strcmp`, and `strcpy()`.

**5.6.4.4** `void shell_cmd_synth ( int argc, char * argv[] )`

References `_open()`, `_printf()`, `_read()`, `_seek()`, `Tone::duration`, `free`, `malloc()`, `pm_create_thread()`, `SEEK_END`, `SEEK_SET`, `shell_makepath()`, `size`, and `synth()`.

## 5.7 src/apps/memview.c File Reference

Main source file of the memory viewer tool "memview".

```
#include "../kernel/pm/syscalls_cli.h"
#include "../kernel/include/stdio.h"
#include "../kernel/include/string.h"
#include "../kernel/include/const.h"
#include "../kernel/io/io.h"
#include "../kernel/pm/pm_main.h"
#include "../kernel/mm/mm.h"
#include "games.h"
```

### Defines

- `#define MEMVIEW_MAX_DUMMY_BLOCKS 10000`  
*The maximum amount of dummy blocks that can be allocated.*

### Functions

- `uint32 heap_get_size (heap_t *heap)`  
*Returns the total size of a heap.*
- `bool keydown (char key, int fd)`
- `int get_free_dummy_slot ()`  
*Finds a free slot for a new dummy block.*
- `void free_all_dummy_blocks ()`  
*Releases all dummy blocks we allocated.*
- `bool allocate_dummy_block (uint32 size)`  
*Allocates a new dummy memory block of the specified size.*
- `bool free_dummy_block ()`  
*Frees the first dummy block found in allocated\_blocks.*
- `void mark_visual_block (uint32 start, uint32 size)`  
*Mark all 'visual' blocks that lie within the memory block at address 'start' with length 'size'.*
- `void update_view ()`  
*Updates the memview tool display.*
- `void mv_switch_to_textmode ()`  
*Disables the framebuffer and clears the textmode screen.*
- `void mv_switch_to_graphicsmode ()`

*Enables the framebuffer.*

- void `mv_show_stats` ()  
*Displays some memory statistics.*
- void `mv_do_benchmark` ()  
*Conducts a simple memory benchmark.*
- void `memview_main` (void)  
*The memory viewer tool entry function.*
- void `shell_cmd_memview` (int argc, char \*argv[ ])  
*Shell command wrapper for the memview tool.*

## Variables

- void \* `mv_allocated_blocks` [MEMVIEW\_MAX\_DUMMY\_BLOCKS]  
*Keeps track of all dummy memory blocks we allocate.*
- uint8 `mv_disp` [25 \*80]  
*Display backbuffer.*
- int `mv_framebuf`  
*/dev/framebuffer file descriptor*
- unsigned int `mv_total_mem`  
*Size of the kernel heap.*
- unsigned int `mv_bytes_per_block`  
*The 'resolution' of one visual block (ie one pixel in the graphical memory view).*

### 5.7.1 Detailed Description

Main source file of the memory viewer tool "memview". "memview" graphically displays the POTATOES memory layout. It can be used to visualize the effects of different memory allocation strategies.

Example:

- start memview
- allocate some blocks by pressing 'a'
- now free some blocks via 'f'
- allocate some more blocks again -> you will see that the memory manager uses a first fit strategy because freed spaces are filled again from 'left to right'.

Every function and variable in this file should be prefixed with 'mv' (for 'memview') to avoid cluttering the global namespace and nasty errors resulting of that.

**Author**

Daniel Bader

**LastChangedBy:****Version****Rev:**

12

**5.7.2 Define Documentation****5.7.2.1 #define MEMVIEW\_MAX\_DUMMY\_BLOCKS 10000**

The maximum amount of dummy blocks that can be allocated.

**5.7.3 Function Documentation****5.7.3.1 bool allocate\_dummy\_block ( uint32 size )**

Allocates a new dummy memory block of the specified size.

**Parameters**

*size* size of the block in bytes

**Returns**

TRUE if a block of the given size was allocated, FALSE if not

References `_malloc()`, `get_free_dummy_slot()`, and `mv_allocated_blocks`.

Referenced by `memview_main()`, and `mv_do_benchmark()`.

**5.7.3.2 void free\_all\_dummy\_blocks ( )**

Releases all dummy blocks we allocated.

References `_free()`, and `mv_allocated_blocks`.

Referenced by `memview_main()`, and `mv_do_benchmark()`.

**5.7.3.3 bool free\_dummy\_block ( )**

Frees the first dummy block found in `allocated_blocks`.

**Returns**

TRUE if a block was released, FALSE if no active blocks were found

References `_free()`, and `mv_allocated_blocks`.

Referenced by `memview_main()`, and `mv_do_benchmark()`.

#### 5.7.3.4 `int get_free_dummy_slot ( )`

Finds a free slot for a new dummy block.

Returns -1 if all slots are occupied.

##### Returns

index of the free slot or -1 if no free slot could be found.

References `mv_allocated_blocks`.

Referenced by `allocate_dummy_block()`.

#### 5.7.3.5 `uint32 heap_get_size ( heap_t * heap )`

Returns the total size of a heap.

##### Parameters

*heap* the heap

##### Returns

size of the heap

References `heap_t::end`, and `heap_t::start`.

Referenced by `heap_contract()`, `heap_expand()`, `heap_mallocn()`, and `update_view()`.

#### 5.7.3.6 `bool keydown ( char key, int fd )`

References `_read()`.

#### 5.7.3.7 `void mark_visual_block ( uint32 start, uint32 size )`

Mark all 'visual' blocks that lie within the memory block at address 'start' with length 'size'.

##### Parameters

*start* the starting address of the block

*size* the size of the block in bytes

References `kernel_heap`, `mv_bytes_per_block`, `mv_disp`, and `heap_t::start`.

Referenced by `update_view()`.



**5.7.3.8 void memview\_main ( void )**

The memory viewer tool entry function.

References `_close()`, `_exit()`, `_fgetch()`, `_open()`, `_printf()`, `allocate_dummy_block()`, `heap_t::end`, `ESCAPE`, `free_all_dummy_blocks()`, `free_dummy_block()`, `halt()`, `kernel_heap`, `keydown()`, `memset()`, `mv_allocated_blocks`, `mv_bytes_per_block`, `mv_do_benchmark()`, `mv_framebuf`, `mv_show_stats()`, `mv_switch_to_graphicsmode()`, `mv_switch_to_textmode()`, `mv_total_mem`, `heap_t::start`, and `update_view()`.

Referenced by `shell_cmd_memview()`.

**5.7.3.9 void mv\_do\_benchmark ( )**

Conducts a simple memory benchmark.

FYI this was generated by a script... ;)

References `_printf()`, `allocate_dummy_block()`, `free_all_dummy_blocks()`, and `free_dummy_block()`.

Referenced by `memview_main()`.

**5.7.3.10 void mv\_show\_stats ( )**

Displays some memory statistics.

References `_fgetch()`, `_open()`, `_printf()`, `heap_t::end`, `kernel_heap`, `mv_bytes_per_block`, `mv_total_mem`, `mm_header::next`, `mm_header::prev`, `mm_header::size`, and `heap_t::start`.

Referenced by `memview_main()`.

**5.7.3.11 void mv\_switch\_to\_graphicsmode ( )**

Enables the framebuffer.

References `_open()`, and `mv_framebuf`.

Referenced by `memview_main()`.

**5.7.3.12 void mv\_switch\_to\_textmode ( )**

Disables the framebuffer and clears the textmode screen.

References `_close()`, `_printf()`, and `mv_framebuf`.

Referenced by `memview_main()`.

**5.7.3.13 void shell\_cmd\_memview ( int argc, char \* argv[] )**

Shell command wrapper for the memview tool.

References `memview_main()`, and `pm_create_thread()`.

**5.7.3.14 void update\_view ( )**

Updates the memview tool display.

We iterate through all memory manager block descriptors and mark affected 'visual' blocks accordingly. This is a bit of a hack as we access internal structures directly.

References `_write()`, `DARKGREY`, `heap_t::end`, `heap_get_size()`, `kernel_heap`, `mark_visual_block()`, `memset()`, `mv_bytes_per_block`, `mv_disp`, `mv_framebuf`, `mv_total_mem`, `mm_header::next`, `mm_header::size`, and `heap_t::start`.

Referenced by `memview_main()`.

## 5.7.4 Variable Documentation

### 5.7.4.1 `void* mv_allocated_blocks[MEMVIEW_MAX_DUMMY_BLOCKS]`

Keeps track of all dummy memory blocks we allocate.

Referenced by `allocate_dummy_block()`, `free_all_dummy_blocks()`, `free_dummy_block()`, `get_free_dummy_slot()`, and `memview_main()`.

### 5.7.4.2 `unsigned int mv_bytes_per_block`

The 'resolution' of one visual block (ie one pixel in the graphical memory view).

Referenced by `mark_visual_block()`, `memview_main()`, `mv_show_stats()`, and `update_view()`.

### 5.7.4.3 `uint8 mv_disp[25 * 80]`

Display backbuffer.

Referenced by `mark_visual_block()`, and `update_view()`.

### 5.7.4.4 `int mv_framebuf`

`/dev/framebuffer` file descriptor

Referenced by `memview_main()`, `mv_switch_to_graphicsmode()`, `mv_switch_to_textmode()`, and `update_view()`.

### 5.7.4.5 `unsigned int mv_total_mem`

Size of the kernel heap.

Careful, this changes all the time as the heap grows and contracts as needed.

Referenced by `memview_main()`, `mv_show_stats()`, and `update_view()`.

## 5.8 `src/apps/pong.c` File Reference

Pong game.

```
#include "games.h"
#include "../kernel/pm/syscalls_cli.h"
#include "../kernel/include/stdio.h"
```

```
#include "../kernel/include/string.h"
#include "../kernel/include/const.h"
#include "../kernel/io/io.h"
```

## Functions

- `bool` `keydown` (char key, int fd)
- `void` `shell_cmd_pong` (int argc, char \*argv[])

## Variables

- `int` `STDIN`  
*The shell's STDIN file descriptor.*

### 5.8.1 Detailed Description

Pong game.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.8.2 Function Documentation

#### 5.8.2.1 `bool` `keydown` ( `char` *key*, `int` *fd* )

References `_read`().

Referenced by `memview_main`(), `shell_cmd_pong`(), `shell_cmd_snapshot`(), `snake`(), `speed`(), and `synth`().

#### 5.8.2.2 `void` `shell_cmd_pong` ( `int` *argc*, `char` \* *argv*[] )

References `_close`(), `_fgetch`(), `_open`(), `_printf`(), `_read`(), `_write`(), `BLACK`, `BLUE`, `CURSOR_DOWN`, `CURSOR_UP`, `DRAW_GLYPH`, `DRAW_PADDLE`, `end_beep`(), `ESCAPE`, `halt`(), `HIT_LPADDLE_SOUND`, `HIT_PADDLE`, `HIT_RPADDLE_SOUND`, `HIT_SIDE_SOUND`, `KEY_A`, `KEY_K`, `KEY_L`, `KEY_S`, `keydown`(), `LIMIT`, `memset`(), `PADDLE_DEFLECTION`, `RED`, `SET_PIXEL`, `start_beep`(), `STDIN`, `strcmp`, `TRUE`, `WHITE`, and `YELLOW`.

### 5.8.3 Variable Documentation

#### 5.8.3.1 int STDIN

The shell's STDIN file descriptor.

Used by all internal commands.

Referenced by shell\_cmd\_pong(), shell\_cmd\_snapshot(), shell\_main(), and snake().

## 5.9 src/apps/shell\_cmds.c File Reference

Implementations of the shell commands.

```
#include "../kernel/io/io.h"
#include "../kernel/include/string.h"
#include "../kernel/include/const.h"
#include "../kernel/include/limits.h"
#include "../kernel/include/stdlib.h"
#include "../kernel/include/stdio.h"
#include "../kernel/fs/fs_types.h"
#include "../kernel/fs/fs_const.h"
#include "../kernel/pm/syscalls_shared.h"
#include "../kernel/pm/syscalls_cli.h"
#include "../kernel/pm/pm_main.h"
#include "games.h"
#include "apps.h"
#include "shell_main.h"
#include "shell_utils.h"
```

### Functions

- void [shell\\_cmd\\_test](#) (int argc, char \*argv[ ])  
*A simple test command that prints all of its arguments to the screen.*
- void [shell\\_cmd\\_cmdlist](#) (int argc, char \*argv[ ])  
*Lists all available arguments.*
- void [shell\\_cmd\\_echo](#) (int argc, char \*argv[ ])  
*Prints its arguments to the screen.*
- void [shell\\_cmd\\_ls](#) (int argc, char \*argv[ ])  
*Lists directory contents.*
- void [shell\\_cmd\\_touch](#) (int argc, char \*argv[ ])

*Creates an empty file.*

- void [shell\\_cmd\\_mkdir](#) (int argc, char \*argv[ ])  
*Creates a new directory.*
- void [shell\\_cmd\\_cat](#) (int argc, char \*argv[ ])  
*Prints file contents.*
- void [shell\\_cmd\\_write](#) (int argc, char \*argv[ ])  
*Writes text into a file.*
- void [shell\\_cmd\\_cd](#) (int argc, char \*argv[ ])  
*Changes the current directory.*
- void [shell\\_cmd\\_clear](#) (int argc, char \*argv[ ])  
*Clears the screen.*
- void [fs\\_shutdown](#) ()  
*Shuts the file system down and writes all important information to HD.*
- void [fs\\_init](#) ()  
*Initializes the file system.*
- void [shell\\_cmd\\_sync](#) (int argc, char \*argv[ ])  
*Flushes all fs memory buffers to disk.*
- void [mem\\_dump](#) ()
- void [shell\\_cmd\\_memdump](#) (int argc, char \*argv[ ])  
*Dumps memory stats and allocated blocks.*
- void [shell\\_cmd\\_pwd](#) (int argc, char \*argv[ ])  
*Prints the current working directory.*
- void [shell\\_cmd\\_cp](#) (int argc, char \*argv[ ])  
*Copies a file.*
- void [pm\\_dump](#) ()  
*Prints some status information about all processes.*
- void [shell\\_cmd\\_ps](#) (int argc, char \*argv[ ])  
*Prints info about running processes.*
- void [reset\\_bf](#) ()
- void [shell\\_cmd\\_bf](#) (int argc, char \*argv[ ])  
*Interface to the brainfuck interpreter device.*
- void [shell\\_cmd\\_exit](#) (int argc, char \*argv[ ])  
*Exits the shell.*
- void [shell\\_cmd\\_date](#) (int argc, char \*argv[ ])

*Prints the current time and date.*

- void `shell_cmd_rm` (int argc, char \*argv[ ])  
*Removes a file.*
- void `shell_cmd_exec` (int argc, char \*argv[ ])  
*Removes a file.*
- void `shell_cmd_kill` (int argc, char \*argv[ ])  
*Kills a process.*
- void `shell_cmd_nice` (int argc, char \*argv[ ])  
*Modifies a process' priority.*

## Variables

- struct `shell_cmd_t shell_cmds` [ ]  
*The shell command table.*

### 5.9.1 Detailed Description

Implementations of the shell commands. TODO: nice to have:

proper tab completion repeat last command globbing rm mv kill df exit exec

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

21

### 5.9.2 Function Documentation

#### 5.9.2.1 void fs\_init ( )

Initializes the file system.

Referenced by `main()`, and `shell_cmd_sync()`.

### 5.9.2.2 void fs\_shutdown ( )

Shuts the file system down and writes all important information to HD.

Referenced by shell\_cmd\_sync(), and test\_sync().

### 5.9.2.3 void mem\_dump ( )

### 5.9.2.4 void pm\_dump ( )

Prints some status information about all processes.

Can be used as a crude form of unix's "ps" command.

References aprintf(), process\_t::context, process\_t::name, process\_t::next, p(), process\_t::pid, and process\_t::priority.

Referenced by shell\_cmd\_ps().

### 5.9.2.5 void reset\_bf ( )

References \_free(), \_malloc(), ASSERT, bf\_buf\_offset, bf\_buf\_position, bf\_buffer, bf\_ptr, bf\_start, jumprover, loopdepth, maxlength, NULL, store\_lock, store\_lock\_level, and temploopdepth.

Referenced by shell\_cmd\_bf().

### 5.9.2.6 void shell\_cmd\_bf ( int argc, char \* argv[] )

Interface to the brainfuck interpreter device.

#### Parameters

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References \_close(), \_open(), \_printf(), \_read(), \_write(), get\_ticks(), reset\_bf(), shell\_makepath(), strcmp, strlen, and UINT32\_MAX.

### 5.9.2.7 void shell\_cmd\_cat ( int argc, char \* argv[] )

Prints file contents.

#### Parameters

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References \_close(), \_fputc(), \_open(), \_printf(), \_read(), \_write(), shell\_makepath(), and STDOUT.

### 5.9.2.8 void shell\_cmd\_cd ( int argc, char \* argv[] )

Changes the current directory.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_close()`, `_open()`, `_printf()`, `cwd`, `shell_makepath()`, and `strcpy()`.

**5.9.2.9 void shell\_cmd\_clear ( int argc, char \* argv[] )**

Clears the screen.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_fputs()`, `_free()`, `_malloc()`, `memset()`, and `STDOUT`.

**5.9.2.10 void shell\_cmd\_cmdlist ( int argc, char \* argv[] )**

Lists all available arguments.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_printf()`, `shell_cmd_t::desc`, `shell_cmd_t::name`, `NULL`, `shell_cmds`, and `STDOUT`.

**5.9.2.11 void shell\_cmd\_cp ( int argc, char \* argv[] )**

Copies a file.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_close()`, `_open()`, `_printf()`, `_read()`, `_write()`, `O_CREAT`, and `shell_makepath()`.

**5.9.2.12 void shell\_cmd\_date ( int argc, char \* argv[] )**

Prints the current time and date.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_close()`, `_open()`, `_printf()`, and `_read()`.



**5.9.2.13 void shell\_cmd\_echo ( int argc, char \* argv[] )**

Prints its arguments to the screen.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_printf()`.

**5.9.2.14 void shell\_cmd\_exec ( int argc, char \* argv[] )**

Removes a file.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_close()`, `_free()`, `_malloc()`, `_open()`, `_printf()`, `_read()`, `bzero`, `shell_handle_command()`, `shell_makepath()`, and `strlen`.

**5.9.2.15 void shell\_cmd\_exit ( int argc, char \* argv[] )**

Exits the shell.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_exit()`, `_free()`, and `_printf()`.

**5.9.2.16 void shell\_cmd\_kill ( int argc, char \* argv[] )**

Kills a process.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_kill()`, `_printf()`, and `atoi()`.

**5.9.2.17 void shell\_cmd\_ls ( int argc, char \* argv[] )**

Lists directory contents.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_close()`, `_open()`, `_printf()`, `_read()`, `_stat()`, `bzero`, `cwd`, `DIRECTORY`, `inode`, `file_info::mode`, `file_info::modify_ts`, `name`, `NULL`, `shell_makepath()`, `file_info::size`, `strcat`, `strlen`, `strncpy`, and `time2str`.

#### 5.9.2.18 void shell\_cmd\_memdump ( int argc, char \* argv[] )

Dumps memory stats and allocated blocks.

See also

[mem\\_dump](#)

#### Parameters

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `mem_dump`.

#### 5.9.2.19 void shell\_cmd\_mkdir ( int argc, char \* argv[] )

Creates a new directory.

#### Parameters

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_close()`, `_open()`, `_printf()`, `O_CREAT`, `shell_makepath()`, `STDOUT`, `strcat`, and `strlen`.

#### 5.9.2.20 void shell\_cmd\_nice ( int argc, char \* argv[] )

Modifies a process' priority.

#### Parameters

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_printf()`, `atoi()`, and `pm_set_thread_priority()`.

#### 5.9.2.21 void shell\_cmd\_ps ( int argc, char \* argv[] )

Prints info about running processes.

#### Parameters

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `pm_dump()`.

**5.9.2.22 void shell\_cmd\_pwd ( int argc, char \* argv[] )**

Prints the current working directory.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_printf()`, and `cwd`.

**5.9.2.23 void shell\_cmd\_rm ( int argc, char \* argv[] )**

Removes a file.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_printf()`, `_unlink()`, and `shell_makepath()`.

**5.9.2.24 void shell\_cmd\_sync ( int argc, char \* argv[] )**

Flushes all fs memory buffers to disk.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `fs_init()`, and `fs_shutdown()`.

**5.9.2.25 void shell\_cmd\_test ( int argc, char \* argv[] )**

A simple test command that prints all of its arguments to the screen.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_printf()`.

**5.9.2.26 void shell\_cmd\_touch ( int argc, char \* argv[] )**

Creates an empty file.

**Parameters**

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_close()`, `_open()`, `_printf()`, `O_CREAT`, and `shell_makepath()`.

### 5.9.2.27 void shell\_cmd\_write ( int argc, char \* argv[] )

Writes text into a file.

#### Parameters

*argc* the number of argument strings in argv

*argv* the argument vector. Contains all arguments of the command.

References `_close()`, `_open()`, `_printf()`, `_write()`, `shell_makepath()`, and `strlen`.

## 5.9.3 Variable Documentation

### 5.9.3.1 struct shell\_cmd\_t shell\_cmds[]

The shell command table.

Every shell command must be registered here to be accessible.

Referenced by `shell_autocomplete()`, `shell_cmd_cmdlist()`, and `shell_handle_command()`.

## 5.10 src/apps/shell\_main.c File Reference

etiOS shell entry point and main source file.

```
#include "../kernel/include/string.h"
#include "../kernel/include/const.h"
#include "../kernel/pm/syscalls_cli.h"
#include "../kernel/pm/pm_main.h"
#include "shell_main.h"
#include "shell_utils.h"
```

### Functions

- void [shell\\_handle\\_command](#) (char \*cmd)  
*Parses and executes a given shell command.*
- void [shell\\_autocomplete](#) (char \*partial)  
*Attempts to autocomplete a given partial shell command.*
- void [shell\\_main](#) ()  
*The shell entry point and main loop.*
- void [new\\_shell](#) ()  
*Starts a new shell in an own thread.*

## Variables

- char `cwd` [255]  
*The current working directory.*
- char `path_buf` [sizeof(`cwd`)]  
*A buffer for `shell_makepath()`.*

### 5.10.1 Detailed Description

etiOS shell entry point and main source file.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.10.2 Function Documentation

#### 5.10.2.1 void `new_shell` ( )

Starts a new shell in an own thread.

References `_free()`, `_malloc()`, `itoa`, `name`, `pm_create_thread()`, `shell_main()`, `strcpy()`, and `strncat()`.

Referenced by `do_tests()`, and `main()`.

#### 5.10.2.2 void `shell_autocomplete` ( `char * partial` )

Attempts to autocomplete a given partial shell command.

#### Parameters

*partial* the partial command string. Will be extended by the first valid completion found.

References `_fputs()`, `shell_cmd_t::cmd`, `memset()`, `shell_cmd_t::name`, `NULL`, `shell_cmds`, `STDOUT`, `strcat`, `strcmp`, `strlen`, and `strncpy`.

Referenced by `shell_main()`.

### 5.10.2.3 void shell\_handle\_command ( char \* cmd )

Parses and executes a given shell command.

#### Parameters

*cmd* string to containing the shell command

References `_free()`, `_printf()`, `shell_cmd_t::cmd`, `name`, `NULL`, `shell_cmds`, `strcmp`, `strdup`, `strlen`, and `strsep`.

Referenced by `shell_cmd_exec()`, and `shell_main()`.

### 5.10.2.4 void shell\_main ( )

The shell entry point and main loop.

This is where the shell prompt gets displayed and user typed commands are dispatched.

References `_close()`, `_exit()`, `_fgets()`, `_open()`, `_printf()`, `shell_cmd_t::cmd`, `cwd`, `memset()`, `shell_autocomplete()`, `shell_handle_command()`, `STDIN`, `STDOUT`, `strcpy()`, and `strlen`.

Referenced by `new_shell()`.

## 5.10.3 Variable Documentation

### 5.10.3.1 char cwd[255]

The current working directory.

All relative paths are relative to this.

Referenced by `shell_cmd_cd()`, `shell_cmd_ls()`, `shell_cmd_pwd()`, `shell_main()`, and `shell_makepath()`.

### 5.10.3.2 char path\_buf[sizeof(cwd)]

A buffer for [shell\\_makepath\(\)](#).

Referenced by `shell_makepath()`.

## 5.11 src/apps/shell\_main.h File Reference

etiOS shell main header file.

### Data Structures

- struct [shell\\_cmd\\_t](#)

*A single shell commmand.*

## Typedefs

- typedef void(\* [shell\\_cmd\\_func](#) )(int argc, char \*argv[ ])  
*A shell command function pointer.*
- typedef struct [shell\\_cmd\\_t](#) [shell\\_cmd\\_t](#)  
*A single shell commmand.*

## Functions

- void [new\\_shell](#) ()  
*Starts a new shell in an own thread.*
- void [shell\\_handle\\_command](#) (char \*cmd)  
*Parses and executes a given shell command.*

## Variables

- struct [shell\\_cmd\\_t](#) [shell\\_cmds](#) [ ]  
*The shell command table.*
- char [cwd](#) [255]  
*The current working directory.*
- char [path\\_buf](#) [sizeof(cwd)]  
*A buffer for [shell\\_makepath](#)().*

### 5.11.1 Detailed Description

etiOS shell main header file.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

## 5.11.2 Typedef Documentation

### 5.11.2.1 typedef void(\* shell\_cmd\_func)(int argc, char \*argv[ ])

A shell command function pointer.

Entry point of all shell commands.

### 5.11.2.2 typedef struct shell\_cmd\_t shell\_cmd\_t

A single shell commmand.

## 5.11.3 Function Documentation

### 5.11.3.1 void new\_shell ( )

Starts a new shell in an own thread.

References `_free()`, `_malloc()`, `itoa`, `name`, `pm_create_thread()`, `shell_main()`, `strcpy()`, and `strncat()`.

Referenced by `do_tests()`, and `main()`.

### 5.11.3.2 void shell\_handle\_command ( char \* cmd )

Parses and executes a given shell command.

#### Parameters

*cmd* string to containing the shell command

References `_free()`, `_printf()`, `shell_cmd_t::cmd`, `name`, `NULL`, `shell_cmds`, `strcmp`, `strdup`, `strlen`, and `strsep`.

Referenced by `shell_cmd_exec()`, and `shell_main()`.

## 5.11.4 Variable Documentation

### 5.11.4.1 char cwd[255]

The current working directory.

All relative paths are relative to this.

Referenced by `shell_cmd_cd()`, `shell_cmd_ls()`, `shell_cmd_pwd()`, `shell_main()`, and `shell_makepath()`.

### 5.11.4.2 char path\_buf[sizeof(cwd)]

A buffer for `shell_makepath()`.

Referenced by `shell_makepath()`.



### 5.11.4.3 struct shell\_cmd\_t shell\_cmds[]

The shell command table.

Every shell command must be registered here to be accessible.

Referenced by shell\_autocomplete(), shell\_cmd\_cmdlist(), and shell\_handle\_command().

## 5.12 src/apps/shell\_utils.c File Reference

Shell utility functions.

```
#include "../kernel/include/types.h"
#include "../kernel/include/const.h"
#include "../kernel/pm/syscalls_cli.h"
#include "../kernel/include/string.h"
#include "../kernel/include/stdarg.h"
#include "../kernel/include/stdio.h"
#include "shell_main.h"
```

### Functions

- int [\\_fputc](#) (char ch, int fd)  
*Writes a character to the given file.*
- void [halt](#) ()
- int [\\_fgetch](#) (int fd)  
*Waits until a character from the given file could be read and returns it.*
- char \* [\\_fgets](#) (char \*s, int n, int fd)  
*Reads a string from a file descriptor into a buffer.*
- int [\\_fputs](#) (char \*s, int fd)  
*Writes a string into a given file.*
- void [\\_printf](#) (char \*fmt,...)  
*Prints formatted output to STDOUT.*
- char \* [shell\\_makepath](#) (char \*path)  
*Makes a given path absolute if needed.*

### Variables

- int [STDIN](#) = -1  
*The shell's STDIN file descriptor.*
- int [STDOUT](#) = -1

*The shell's STDOUT file descriptor.*

### 5.12.1 Detailed Description

Shell utility functions.

#### Author

Daniel Bader

#### LastChangedBy:

dbader

#### Version

#### Rev:

16

### 5.12.2 Function Documentation

#### 5.12.2.1 `int _fgetch ( int fd )`

Waits until a character from the given file could be read and returns it.

#### Parameters

*fd* the file descriptor

#### Returns

the character that was read

References `_read()`, and `halt()`.

Referenced by `_fgets()`, `memview_main()`, `mv_show_stats()`, `shell_cmd_pong()`, `snake()`, and `speed()`.

#### 5.12.2.2 `char* _fgets ( char * s, int n, int fd )`

Reads a string from a file descriptor into a buffer.

#### Parameters

*s* the string buffer

*n* the maximum number of bytes to read (ie the buffer size)

*fd* the file descriptor

#### Returns

the string buffer

References `_fgetch()`, `_fputch()`, `start`, and `STDOUT`.

Referenced by `shell_main()`, and `snapshot()`.

### 5.12.2.3 `int _fputc ( char ch, int fd )`

Writes a character to the given file.

#### Parameters

*ch* the character to write

*fd* the file descriptor

#### Returns

the number of bytes written, -1 on error

References `_write()`.

Referenced by `_fgets()`, and `shell_cmd_cat()`.

### 5.12.2.4 `int _fputs ( char * s, int fd )`

Writes a string into a given file.

#### Parameters

*s* the string to write

*fd* the file descriptor

#### Returns

the number of bytes written, -1 on error

References `_write()`, and `strlen`.

Referenced by `shell_autocomplete()`, and `shell_cmd_clear()`.

### 5.12.2.5 `void _printf ( char * fmt, ... )`

Prints formatted output to STDOUT.

#### See also

[printf](#) This exists as a stub to ease the separation of the shell from the kernel code (as of now, the shell could simply call the kernel [printf](#)).

References `_write()`, `STDOUT`, `va_end`, `va_start`, and `vsnprintf`.

Referenced by `heap_mem_dump()`, `memview_main()`, `mv_do_benchmark()`, `mv_show_stats()`, `mv_switch_to_textmode()`, `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cd()`, `shell_cmd_cmdlist()`, `shell_cmd_cp()`, `shell_cmd_date()`, `shell_cmd_echo()`, `shell_cmd_exec()`, `shell_cmd_exit()`, `shell_cmd_kill()`, `shell_cmd_ls()`, `shell_cmd_mkdir()`, `shell_cmd_nice()`, `shell_cmd_pong()`, `shell_cmd_pwd()`, `shell_cmd_rm()`, `shell_cmd_snapshot()`, `shell_cmd_speed()`, `shell_cmd_synth()`, `shell_cmd_test()`, `shell_cmd_touch()`, `shell_cmd_write()`, `shell_handle_command()`, `shell_main()`, `snake()`, `snapshot()`, `speed()`, `threadA()`, `threadB()`, `threadConsumer()`, `threadFabrik()`, `threadGeschaeft1()`, `threadGeschaeft2()`, `threadLastwagen1()`, `threadLastwagen2()`, `threadMitarbeiter()`, and `threadProducer()`.

### 5.12.2.6 void halt ( )

Referenced by `_fgetch()`, `dev_stdin_read()`, `fgetch()`, `interpret_bf()`, `main()`, `memview_main()`, `p()`, `panic()`, `shell_cmd_pong()`, `sleep()`, `sleep_ticks()`, `snake()`, `threadA()`, `threadB()`, `threadC()`, `threadConsumer()`, `threadD()`, `threadFabrik()`, `threadGeschaeft1()`, `threadGeschaeft2()`, `threadLastwagen1()`, `threadLastwagen2()`, `threadMitarbeiter()`, `threadProducer()`, and `wait_on_hd_interrupt()`.

### 5.12.2.7 char\* shell\_makepath ( char \* path )

Makes a given path absolute if needed.

`shell_makepath` checks for a leading slash in `path` to decide whether a given path is already absolute. If the path is not absolute it will be appended to the current working directory. Calling this will invalidate the last result.

#### Parameters

*path* the path to make absolute

#### Returns

the absolute path. This pointer is only valid until the next call to `shell_makepath()`.

References `cwd`, `path_buf`, `strcat`, `strcpy()`, and `strlen`.

Referenced by `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cd()`, `shell_cmd_cp()`, `shell_cmd_exec()`, `shell_cmd_ls()`, `shell_cmd_mkdir()`, `shell_cmd_rm()`, `shell_cmd_snapshot()`, `shell_cmd_speed()`, `shell_cmd_synth()`, `shell_cmd_touch()`, and `shell_cmd_write()`.

## 5.12.3 Variable Documentation

### 5.12.3.1 int STDIN = -1

The shell's STDIN file descriptor.

Used by all internal commands.

Referenced by `shell_cmd_pong()`, `shell_cmd_snapshot()`, `shell_main()`, and `snake()`.

### 5.12.3.2 int STDOUT = -1

The shell's STDOUT file descriptor.

Used by all internal commands.

Referenced by `_fgets()`, `_printf()`, `shell_autocomplete()`, `shell_cmd_cat()`, `shell_cmd_clear()`, `shell_cmd_cmdlist()`, `shell_cmd_mkdir()`, `shell_cmd_snapshot()`, and `shell_main()`.

## 5.13 src/apps/shell\_utils.h File Reference

Shell utility functions.

## Functions

- int `_fputc` (char ch, int fd)  
*Writes a character to the given file.*
- int `_fgetc` (int fd)  
*Waits until a character from the given file could be read and returns it.*
- char \* `_fgets` (char \*s, int n, int fd)  
*Reads a string from a file descriptor into a buffer.*
- int `_fputs` (char \*s, int fd)  
*Writes a string into a given file.*
- void `_printf` (char \*fmt,...)  
*Prints formatted output to STDOUT.*
- char \* `shell_makepath` (char \*path)  
*Makes a given path absolute if needed.*

## Variables

- int `STDIN`  
*The shell's STDIN file descriptor.*
- int `STDOUT`  
*The shell's STDOUT file descriptor.*

### 5.13.1 Detailed Description

Shell utility functions.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

## 5.13.2 Function Documentation

### 5.13.2.1 `int _fgetc ( int fd )`

Waits until a character from the given file could be read and returns it.

#### Parameters

*fd* the file descriptor

#### Returns

the character that was read

### 5.13.2.2 `char* _fgets ( char * s, int n, int fd )`

Reads a string from a file descriptor into a buffer.

#### Parameters

*s* the string buffer

*n* the maximum number of bytes to read (ie the buffer size)

*fd* the file descriptor

#### Returns

the string buffer

### 5.13.2.3 `int _fputc ( char ch, int fd )`

Writes a character to the given file.

#### Parameters

*ch* the character to write

*fd* the file descriptor

#### Returns

the number of bytes written, -1 on error

### 5.13.2.4 `int _fputs ( char * s, int fd )`

Writes a string into a given file.

#### Parameters

*s* the string to write

*fd* the file descriptor

#### Returns

the number of bytes written, -1 on error

### 5.13.2.5 void \_printf ( char \* *fmt*, ... )

Prints formatted output to STDOUT.

#### See also

[printf](#) This exists as a stub to ease the separation of the shell from the kernel code (as of now, the shell could simply call the kernel [printf](#)).

### 5.13.2.6 char\* shell\_makepath ( char \* *path* )

Makes a given path absolute if needed.

shell\_makepath checks for a leading slash in path to decide whether a given path is already absolute. If the path is not absolute it will be appended to the current working directory. Calling this will invalidate the last result.

#### Parameters

*path* the path to make absolute

#### Returns

the absolute path. This pointer is only valid until the next call to [shell\\_makepath\(\)](#).

## 5.13.3 Variable Documentation

### 5.13.3.1 int STDIN

The shell's STDIN file descriptor.

Used by all internal commands.

### 5.13.3.2 int STDOUT

The shell's STDOUT file descriptor.

Used by all internal commands.

## 5.14 src/apps/snake.c File Reference

Snake game.

```
#include "games.h"
#include "../kernel/pm/syscalls_cli.h"
#include "../kernel/pm/pm_main.h"
#include "../kernel/include/stdio.h"
#include "../kernel/include/string.h"
#include "../kernel/include/stdlib.h"
#include "../kernel/include/const.h"
```

```
#include "../kernel/io/io.h"
#include "../kernel/include/ringbuffer.h"
```

## Functions

- `uint16 draw_snake` (`ring_fifo *snake`, `uint8 color`, `uint8 headcolor`, `uint8 *bbuf`)
- `bool body_collision` (`ring_fifo *snake`, `uint16 pos`)
- `void snake` ()
- `void shell_cmd_snake` (`int argc`, `char *argv[]`)

## Variables

- `int STDIN`  
*The shell's STDIN file descriptor.*

### 5.14.1 Detailed Description

Snake game.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.14.2 Function Documentation

#### 5.14.2.1 `bool body_collision ( ring_fifo * snake, uint16 pos )`

References `rf_copy()`, `rf_free()`, `rf_getlength()`, and `rf_read()`.

Referenced by `snake()`.

#### 5.14.2.2 `uint16 draw_snake ( ring_fifo * snake, uint8 color, uint8 headcolor, uint8 * bbuf )`

References `rf_copy()`, `rf_free()`, `rf_getlength()`, `rf_read()`, and `SET_PIXEL`.

Referenced by `snake()`.



**5.14.2.3 void shell\_cmd\_snake ( int argc, char \* argv[] )**

References `_free()`, `_malloc()`, `name`, `pm_create_thread()`, `snake()`, `strcmp`, and `strcpy()`.

**5.14.2.4 void snake ( )**

References `_close()`, `_exit()`, `_fgetch()`, `_open()`, `_printf()`, `_read()`, `_write()`, `active_proc`, `BLACK`, `BLUE`, `body_collision()`, `CURSOR_DOWN`, `CURSOR_LEFT`, `CURSOR_RIGHT`, `CURSOR_UP`, `draw_snake()`, `EAT_APPLE_SOUND`, `end_beep()`, `ESCAPE`, `FALSE`, `get_ticks()`, `GREEN`, `halt()`, `KEY_A`, `KEY_D`, `KEY_I`, `KEY_J`, `KEY_K`, `KEY_L`, `KEY_S`, `KEY_W`, `keydown()`, `LIGHTBLUE`, `LIGHTGREEN`, `memset()`, `process_t::name`, `ORANGE`, `rand()`, `RED`, `rf_alloc()`, `rf_free()`, `rf_getlength()`, `rf_read()`, `rf_write()`, `SET_PIXEL`, `SNAKE_DOWN`, `SNAKE_LEFT`, `SNAKE_RIGHT`, `SNAKE_UP`, `srand()`, `start_beep()`, `STDIN`, `strlen`, and `TRUE`.

Referenced by `shell_cmd_snake()`.

**5.14.3 Variable Documentation****5.14.3.1 int STDIN**

The shell's STDIN file descriptor.

Used by all internal commands.

**5.15 src/apps/snapshot.c File Reference**

Programm to make and view screenshots.

```
#include "../kernel/pm/syscalls_cli.h"
#include "../kernel/pm/syscalls_shared.h"
#include "../kernel/include/const.h"
#include "../kernel/include/string.h"
#include "../kernel/include/stdlib.h"
#include "../kernel/io/io.h"
#include "../kernel/io/io_virtual.h"
#include "../kernel/pm/pm_main.h"
#include "games.h"
```

**Functions**

- void `snapshot ()`
- char \* `shell_makepath (char *path)`  
*Makes a given path absolute if needed.*
- void `shell_cmd_snapshot (int argc, char *argv[])`
- void `make_snapshot ()`

## Variables

- char `snap_buffer` [2 \*25 \*80]
- int `STDIN`  
*The shell's STDIN file descriptor.*
- int `STDOUT`  
*The shell's STDOUT file descriptor.*

### 5.15.1 Detailed Description

Programm to make and view screenshots.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.15.2 Function Documentation

#### 5.15.2.1 void `make_snapshot` ( )

References `memcpy`, `pm_create_thread`(), `snap_buffer`, `snapshot`(), and `VGA_DISPLAY`.

Referenced by `do_tests`().

#### 5.15.2.2 void `shell_cmd_snapshot` ( int *argc*, char \* *argv*[ ] )

References `_close`(), `_open`(), `_printf`(), `_read`(), `virt_monitor::disable_refresh`, `ESCAPE`, `get_active_virt_monitor`(), `keydown`(), `shell_makepath`(), `STDIN`, `STDOUT`, and `VGA_DISPLAY`.

#### 5.15.2.3 char\* `shell_makepath` ( char \* *path* )

Makes a given path absolute if needed.

`shell_makepath` checks for a leading slash in `path` to decide whether a given path is already absolute. If the path is not absolute it will be appended to the current working directory. Calling this will invalidate the last result.

#### Parameters

*path* the path to make absolute

**Returns**

the absolute path. This pointer is only valid until the next call to [shell\\_makepath\(\)](#).

References `cwd`, `path_buf`, `strcat`, `strcpy()`, and `strlen`.

Referenced by `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cd()`, `shell_cmd_cp()`, `shell_cmd_exec()`, `shell_cmd_ls()`, `shell_cmd_mkdir()`, `shell_cmd_rm()`, `shell_cmd_snapshot()`, `shell_cmd_speed()`, `shell_cmd_synth()`, `shell_cmd_touch()`, and `shell_cmd_write()`.

**5.15.2.4 void snapshot ( )**

References `_close()`, `_exit()`, `_fgets()`, `_free()`, `_malloc()`, `_open()`, `_printf()`, `_write()`, `memset()`, `O_CREAT`, `snap_buffer`, `strcpy()`, and `strlen`.

Referenced by `make_snapshot()`.

**5.15.3 Variable Documentation****5.15.3.1 char snap\_buffer[2 \*25 \*80]**

Referenced by `make_snapshot()`, and `snapshot()`.

**5.15.3.2 int STDIN**

The shell's STDIN file descriptor.

Used by all internal commands.

**5.15.3.3 int STDOUT**

The shell's STDOUT file descriptor.

Used by all internal commands.

Referenced by `_fgets()`, `_printf()`, `shell_autocomplete()`, `shell_cmd_cat()`, `shell_cmd_clear()`, `shell_cmd_cmdlist()`, `shell_cmd_mkdir()`, `shell_cmd_snapshot()`, and `shell_main()`.

**5.16 src/apps/synthesizer.c File Reference**

Synthesizer tool.

```
#include "../kernel/io/io.h"
#include "../kernel/pm/pm_main.h"
#include "../kernel/pm/syscalls_cli.h"
#include "games.h"
#include "shell_utils.h"
#include "../kernel/include/string.h"
#include "../kernel/include/stdlib.h"
```

## Data Structures

- struct [Tone](#)  
*Stores a frequency and a duration.*

## Functions

- void [synth](#) ()
- void [shell\\_cmd\\_synth](#) (int argc, char \*argv[ ])

### 5.16.1 Detailed Description

Synthesizer tool.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.16.2 Function Documentation

#### 5.16.2.1 void shell\_cmd\_synth ( int argc, char \* argv[ ] )

References [\\_open\(\)](#), [\\_printf\(\)](#), [\\_read\(\)](#), [\\_seek\(\)](#), [Tone::duration](#), [free](#), [malloc\(\)](#), [pm\\_create\\_thread\(\)](#), [SEEK\\_END](#), [SEEK\\_SET](#), [shell\\_makepath\(\)](#), [size](#), and [synth\(\)](#).

#### 5.16.2.2 void synth ( )

References [\\_close\(\)](#), [\\_exit\(\)](#), [\\_open\(\)](#), [ESCAPE](#), [KEY\\_A](#), [KEY\\_D](#), [KEY\\_E](#), [KEY\\_F](#), [KEY\\_G](#), [KEY\\_H](#), [KEY\\_J](#), [KEY\\_K](#), [KEY\\_MINUS](#), [KEY\\_PLUS](#), [KEY\\_S](#), [KEY\\_T](#), [KEY\\_U](#), [KEY\\_W](#), [KEY\\_Z](#), [keydown\(\)](#), [NOTE\\_A](#), [NOTE\\_ASH](#), [NOTE\\_B](#), [NOTE\\_C](#), [NOTE\\_CSH](#), [NOTE\\_D](#), [NOTE\\_DSH](#), [NOTE\\_E](#), [NOTE\\_F](#), [NOTE\\_FSH](#), [NOTE\\_G](#), and [NOTE\\_GSH](#).

Referenced by [shell\\_cmd\\_synth\(\)](#).

## 5.17 src/kernel/fs/fs\_block\_dev.c File Reference

Basic block based functions.

```
#include "../include/const.h"
```

```
#include "../include/types.h"
#include "../include/debug.h"
#include "fs_const.h"
#include "fs_types.h"
#include "fs_buf.h"
#include "fs_bmap.h"
#include "fs_block_dev.h"
#include "fs_inode_table.h"
#include "../include/string.h"
#include "../include/stdio.h"
#include "../io/io_harddisk.h"
```

## Functions

- void `rd_block` (void \*buf, `block_nr` blk\_nr, `size_t` num\_bytes)  
*Read a block from HD into 'buf' and cache it into the read\_cache.*
- void `cache_block` (`block_nr` blk\_nr, `size_t` num\_bytes)  
*Load a block from HD into temporary read\_cache.*
- void `wrt_block` (`block_nr` blk\_nr, void \*buf, `size_t` num\_bytes)  
*Write the buffer's content to HD.*
- void `wrt_cache` (`block_cache` \*cache, `size_t` num\_bytes)  
*Write a block cache to HD.*
- void `clear_block` (`block_nr` blk\_nr)  
*Clear a block from HD by resetting it with zeros.*
- `block_nr` `get_data_block` (`m_inode` \*inode, `uint32` pos, `bool` allow\_enlargement)  
*Determine block for given position within an inode.*
- `block_nr` `enlarge_file` (`block_nr` \*blk\_ptr, `block_nr` alloc\_start)  
*Enlarge the file.*

### 5.17.1 Detailed Description

Basic block based functions.

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

**Version****Rev:**

12

**5.17.2 Function Documentation****5.17.2.1 void cache\_block ( block\_nr *blk\_nr*, size\_t *num\_bytes* )**

Load a block from HD into temporary read\_cache.

**Parameters**

*blk\_nr* block number to be read

*num\_bytes* number of bytes to be read

References block\_buffer::block\_nr, block\_buffer::cache, clear\_cache(), hd\_read\_sector, and read\_cache.

Referenced by fs\_read(), fs\_write(), and rd\_block().

**5.17.2.2 void clear\_block ( block\_nr *blk\_nr* )**

Clear a block from HD by resetting it with zeros.

**Parameters**

*blk\_nr* block number to be reset

References block\_buffer::block\_nr, BLOCK\_SIZE, bzero, block\_buffer::cache, rd\_block(), read\_cache, and wrt\_cache().

Referenced by enlarge\_file(), and insert\_file\_into\_dir().

**5.17.2.3 block\_nr enlarge\_file ( block\_nr \* *blk\_ptr*, block\_nr *alloc\_start* )**

Enlarge the file.

**Parameters**

*blk\_ptr* pointer to inode block pointer

*alloc\_start* start block for block allocation search

**Returns**

block\_nr number of newly allocated block

References alloc\_block(), and clear\_block().

Referenced by get\_data\_block().

#### 5.17.2.4 `block_nr get_data_block ( m_inode * inode, uint32 pos, bool allow_enlargement )`

Determine block for given position within an inode.

##### Parameters

- inode* file's inode
- pos* position/byte within the file/inode
- allow\_enlargement* enlarge the file if pos > EOF?

##### Returns

block number containing the desired position

References `addr_cache`, `ADDRS_PER_BLOCK`, `BLOCK_SIZE`, `BYTES_DIRECT`, `BYTES_DOUBLE_INDIRECT`, `BYTES_SINGLE_INDIRECT`, `enlarge_file()`, `fs_dprintf`, `m_inode::i_adr`, `m_inode::i_direct_pointer`, `m_inode::i_double_indirect_pointer`, `m_inode::i_single_indirect_pointer`, `NULL`, `rd_block()`, `write_inode()`, and `wrt_block()`.

Referenced by `delete_file_from_dir()`, `fs_read()`, `fs_truncate()`, `fs_write()`, and `insert_file_into_dir()`.

#### 5.17.2.5 `void rd_block ( void * buf, block_nr blk_nr, size_t num_bytes )`

Read a block from HD into 'buf' and cache it into the read\_cache.

##### Parameters

- \*buf* pointer to the dest. buffer
- blk\_nr* block number to be read
- num\_bytes* number of bytes to be read

References `block_buffer::cache`, `cache_block()`, `memcpy`, and `read_cache`.

Referenced by `clear_block()`, `fs_truncate()`, `get_data_block()`, `load_bmap()`, `load_super_block()`, and `read_dinode()`.

#### 5.17.2.6 `void wrt_block ( block_nr blk_nr, void * buf, size_t num_bytes )`

Write the buffer's content to HD.

##### Parameters

- blk\_nr* destination block number
- buf* buffer with content to be written
- num\_bytes* number of bytes to be written

References `block_buffer::block_nr`, `block_buffer::cache`, `clear_cache()`, `hd_write_sector`, `memcpy`, and `write_cache`.

Referenced by `delete_file_from_dir()`, `fs_truncate()`, `get_data_block()`, `insert_file_into_dir()`, `mark_block()`, `write_bmap()`, `write_inode()`, `write_super_block()`, and `wrt_cache()`.

### 5.17.2.7 void wrt\_cache ( block\_cache \* cache, size\_t num\_bytes )

Write a block cache to HD.

#### Parameters

- cache* cache with content
- num\_bytes* number of bytes to be written

References block\_buffer::block\_nr, block\_buffer::cache, and wrt\_block().

Referenced by clear\_block(), and fs\_write().

## 5.18 src/kernel/fs/fs\_block\_dev.h File Reference

Basic definitions of all block based functions.

### Functions

- void [rd\\_block](#) (void \*buf, [block\\_nr](#) blk\_nr, [size\\_t](#) num\_bytes)  
*Read a block from HD into 'buf' and cache it into the read\_cache.*
- void [cache\\_block](#) ([block\\_nr](#) blk\_nr, [size\\_t](#) num\_bytes)  
*Load a block from HD into temporary read\_cache.*
- void [wrt\\_block](#) ([block\\_nr](#) blk\_nr, void \*buf, [size\\_t](#) num\_bytes)  
*Write the buffer's content to HD.*
- void [wrt\\_cache](#) ([block\\_cache](#) \*cache, [size\\_t](#) num\_bytes)  
*Write a block cache to HD.*
- void [clear\\_block](#) ([block\\_nr](#) blk\_nr)  
*Clear a block from HD by resetting it with zeros.*
- [block\\_nr](#) [get\\_data\\_block](#) ([m\\_inode](#) \*inode, [uint32](#) pos, [bool](#) allow\_enlargement)  
*Determine block for given position within an inode.*
- [block\\_nr](#) [enlarge\\_file](#) ([block\\_nr](#) \*blk\_ptr, [block\\_nr](#) alloc\_start)  
*Enlarge the file.*

### 5.18.1 Detailed Description

Basic definitions of all block based functions.

#### Author

Vincenz Doelle



**LastChangedBy:**

dtraytel

**Version****Rev:**

12

## 5.18.2 Function Documentation

### 5.18.2.1 void cache\_block ( block\_nr *blk\_nr*, size\_t *num\_bytes* )

Load a block from HD into temporary read\_cache.

**Parameters**

*blk\_nr* block number to be read  
*num\_bytes* number of bytes to be read

References block\_buffer::block\_nr, block\_buffer::cache, clear\_cache(), hd\_read\_sector, and read\_cache.  
Referenced by fs\_read(), fs\_write(), and rd\_block().

### 5.18.2.2 void clear\_block ( block\_nr *blk\_nr* )

Clear a block from HD by resetting it with zeros.

**Parameters**

*blk\_nr* block number to be reset

References block\_buffer::block\_nr, BLOCK\_SIZE, bzero, block\_buffer::cache, rd\_block(), read\_cache, and wrt\_cache().  
Referenced by enlarge\_file(), and insert\_file\_into\_dir().

### 5.18.2.3 block\_nr enlarge\_file ( block\_nr \* *blk\_ptr*, block\_nr *alloc\_start* )

Enlarge the file.

**Parameters**

*blk\_ptr* pointer to inode block pointer  
*alloc\_start* start block for block allocation search

**Returns**

block\_nr number of newly allocated block

References alloc\_block(), and clear\_block().  
Referenced by get\_data\_block().

#### 5.18.2.4 `block_nr get_data_block ( m_inode * inode, uint32 pos, bool allow_enlargement )`

Determine block for given position within an inode.

##### Parameters

- inode* file's inode
- pos* position/byte within the file/inode
- allow\_enlargement* enlarge the file if pos > EOF?

##### Returns

block number containing the desired position

References `addr_cache`, `ADDRS_PER_BLOCK`, `BLOCK_SIZE`, `BYTES_DIRECT`, `BYTES_DOUBLE_INDIRECT`, `BYTES_SINGLE_INDIRECT`, `enlarge_file()`, `fs_dprintf`, `m_inode::i_adr`, `m_inode::i_direct_pointer`, `m_inode::i_double_indirect_pointer`, `m_inode::i_single_indirect_pointer`, `NULL`, `rd_block()`, `write_inode()`, and `wrt_block()`.

Referenced by `delete_file_from_dir()`, `fs_read()`, `fs_truncate()`, `fs_write()`, and `insert_file_into_dir()`.

#### 5.18.2.5 `void rd_block ( void * buf, block_nr blk_nr, size_t num_bytes )`

Read a block from HD into 'buf' and cache it into the read\_cache.

##### Parameters

- \*buf* pointer to the dest. buffer
- blk\_nr* block number to be read
- num\_bytes* number of bytes to be read

References `block_buffer::cache`, `cache_block()`, `memcpy`, and `read_cache`.

Referenced by `clear_block()`, `fs_truncate()`, `get_data_block()`, `load_bmap()`, `load_super_block()`, and `read_dinode()`.

#### 5.18.2.6 `void wrt_block ( block_nr blk_nr, void * buf, size_t num_bytes )`

Write the buffer's content to HD.

##### Parameters

- blk\_nr* destination block number
- buf* buffer with content to be written
- num\_bytes* number of bytes to be written

References `block_buffer::block_nr`, `block_buffer::cache`, `clear_cache()`, `hd_write_sector`, `memcpy`, and `write_cache`.

Referenced by `delete_file_from_dir()`, `fs_truncate()`, `get_data_block()`, `insert_file_into_dir()`, `mark_block()`, `write_bmap()`, `write_inode()`, `write_super_block()`, and `wrt_cache()`.

### 5.18.2.7 void wrt\_cache ( block\_cache \* cache, size\_t num\_bytes )

Write a block cache to HD.

#### Parameters

*cache* cache with content

*num\_bytes* number of bytes to be written

References block\_buffer::block\_nr, block\_buffer::cache, and wrt\_block().

Referenced by clear\_block(), and fs\_write().

## 5.19 src/kernel/fs/fs\_bmap.c File Reference

Functions to manage the block bitmap (bmap).

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/debug.h"
#include "../include/assert.h"
#include "../include/string.h"
#include "../include/stdlib.h"
#include "fs_const.h"
#include "fs_types.h"
#include "fs_super.h"
#include "fs_bmap.h"
#include "fs_block_dev.h"
```

### Functions

- [uint32 get\\_hdsize \(\)](#)  
*Returns the size of the hard disk.*
- [size\\_t malloc\\_bmap \(\)](#)  
*Allocate the block bitmap in memory.*
- void [init\\_bmap \(\)](#)  
*Resets the block bitmap.*
- void [load\\_bmap \(\)](#)  
*Loads the block bitmap from HD.*
- void [write\\_bmap \(\)](#)  
*Writes the block bitmap to HD.*
- [block\\_nr get\\_free\\_block \(block\\_nr start\)](#)

*Find a new unused block.*

- void `mark_block` (`block_nr` blk\_nr, `bool` flag)  
*Function to mark a block number as free (flag = 0) or used (flag = 1).*
- `bool` `is_allocated_block` (`block_nr` blk\_nr)  
*Checks whether a block was already allocated before.*
- `block_nr` `alloc_block` (`block_nr` start)  
*Allocates a new block.*
- void `dump_bmap` ()  
*Dumps out the block bitmap.*

### 5.19.1 Detailed Description

Functions to manage the block bitmap (bmap).

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.19.2 Function Documentation

#### 5.19.2.1 `block_nr` `alloc_block` ( `block_nr` *start* )

Allocates a new block.

#### Parameters

*start* search offset (start to search linear from block number "start")

References `get_free_block()`, `mark_block()`, and `TRUE`.

Referenced by `enlarge_file()`, `insert_file_into_dir()`, and `test_bmap()`.

#### 5.19.2.2 void `dump_bmap` ( )

Dumps out the block bitmap.

References `bmap`, `BOOT_BLOCK`, `fs_dprintf`, `get_hdsiz`, `is_allocated_block()`, and `NULL`.

Referenced by `test_bmap()`, `test_delete()`, and `test_sync()`.

### 5.19.2.3 `block_nr get_free_block ( block_nr start )`

Find a new unused block.

Search linear from the "start" block number.

#### Parameters

*start* search offset (start to search linear from block number "start")

#### Returns

block number of the unused block

References FALSE, get\_hdsiz, and is\_allocated\_block().

Referenced by alloc\_block(), and test\_bmap().

### 5.19.2.4 `uint32 get_hdsiz ( )`

Returns the size of the hard disk.

#### Returns

size of the master hard disk in sectors

### 5.19.2.5 `void init_bmap ( )`

Resets the block bitmap.

References ASSERT, first\_data\_block, fs\_dprintf, malloc\_bmap(), mark\_block(), size, and TRUE.

Referenced by create\_fs().

### 5.19.2.6 `bool is_allocated_block ( block_nr blk_nr )`

Checks whether a block was already allocated before.

#### Parameters

*blk\_nr* block number to check

#### Returns

allocation status

References bmap.

Referenced by dump\_bmap(), get\_free\_block(), and test\_bmap().

### 5.19.2.7 `void load_bmap ( )`

Loads the block bitmap from HD.

References ASSERT, BLOCK\_SIZE, bmap, FIRST\_BMAP\_BLOCK, fs\_dprintf, malloc\_bmap(), num\_bmap\_blocks, rd\_block(), and size.

Referenced by load\_fs().

### 5.19.2.8 `size_t malloc_bmap ( )`

Allocate the block bitmap in memory.

Functions on the block bitmap.

#### Returns

size of the block bitmap

References `BLOCK_SIZE`, `bmap`, `bzero`, `first_data_block`, `fs_dprintf`, `get_hdsizes`, `mallocn`, `NULL`, `num_bmap_blocks`, `ROOT_INODE_BLOCK`, and `size`.

Referenced by `init_bmap()`, and `load_bmap()`.

### 5.19.2.9 `void mark_block ( block_nr blk_nr, bool flag )`

Function to mark a block number as free (`flag = 0`) or used (`flag = 1`).

#### Parameters

*blk\_nr* block number which should be marked

*flag* value with whom the block should be marked (1 | 0)

References `BLOCK_SIZE`, `bmap`, `FIRST_BMAP_BLOCK`, `fs_dprintf`, `TRUE`, and `wrt_block()`.

Referenced by `alloc_block()`, `delete_entry()`, `fs_truncate()`, `init_bmap()`, `insert_file_into_dir()`, and `test_bmap()`.

### 5.19.2.10 `void write_bmap ( )`

Writes the block bitmap to HD.

References `BLOCK_SIZE`, `bmap`, `FIRST_BMAP_BLOCK`, `fs_dprintf`, `num_bmap_blocks`, and `wrt_block()`.

Referenced by `fs_shutdown()`.

## 5.20 `src/kernel/fs/fs_bmap.h` File Reference

Basic definitions concerning the block bitmap (`bmap`).

### Functions

- [size\\_t malloc\\_bmap \( \)](#)  
*Functions on the block bitmap.*
- [void init\\_bmap \( \)](#)  
*Resets the block bitmap.*
- [void load\\_bmap \( \)](#)  
*Loads the block bitmap from HD.*

- void `write_bmap ()`  
*Writes the block bitmap to HD.*
- `block_nr get_free_block (block_nr start)`  
*Find a new unused block.*
- void `mark_block (block_nr blk_nr, bool flag)`  
*Function to mark a block number as free (flag = 0) or used (flag = 1).*
- `block_nr alloc_block (block_nr start)`  
*Allocates a new block.*
- `bool is_allocated_block (block_nr blk_nr)`  
*Checks whether a block was already allocated before.*
- void `dump_bmap ()`  
*Dumps out the block bitmap.*

## Variables

- `uint8 * bmap`  
*Central block bitmap structure.*
- `uint32 num_bmap_blocks`
- `uint32 first_data_block`

### 5.20.1 Detailed Description

Basic definitions concerning the block bitmap (bmap).

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

## 5.20.2 Function Documentation

### 5.20.2.1 `block_nr alloc_block ( block_nr start )`

Allocates a new block.

#### Parameters

*start* search offset (start to search linear from block number "start")

References `get_free_block()`, `mark_block()`, and `TRUE`.

Referenced by `enlarge_file()`, `insert_file_into_dir()`, and `test_bmap()`.

### 5.20.2.2 `void dump_bmap ( )`

Dumps out the block bitmap.

References `bmap`, `BOOT_BLOCK`, `fs_dprintf`, `get_hdsizes`, `is_allocated_block()`, and `NULL`.

Referenced by `test_bmap()`, `test_delete()`, and `test_sync()`.

### 5.20.2.3 `block_nr get_free_block ( block_nr start )`

Find a new unused block.

Search linear from the "start" block number.

#### Parameters

*start* search offset (start to search linear from block number "start")

#### Returns

block number of the unused block

References `FALSE`, `get_hdsizes`, and `is_allocated_block()`.

Referenced by `alloc_block()`, and `test_bmap()`.

### 5.20.2.4 `void init_bmap ( )`

Resets the block bitmap.

References `ASSERT`, `first_data_block`, `fs_dprintf`, `malloc_bmap()`, `mark_block()`, `size`, and `TRUE`.

Referenced by `create_fs()`.

### 5.20.2.5 `bool is_allocated_block ( block_nr blk_nr )`

Checks whether a block was already allocated before.

#### Parameters

*blk\_nr* block number to check



**Returns**

allocation status

References bmap.

Referenced by dump\_bmap(), get\_free\_block(), and test\_bmap().

**5.20.2.6 void load\_bmap ( )**

Loads the block bitmap from HD.

References ASSERT, BLOCK\_SIZE, bmap, FIRST\_BMAP\_BLOCK, fs\_dprintf, malloc\_bmap(), num\_bmap\_blocks, rd\_block(), and size.

Referenced by load\_fs().

**5.20.2.7 size\_t malloc\_bmap ( )**

Functions on the block bitmap.

Functions on the block bitmap.

**Returns**

size of the block bitmap

References BLOCK\_SIZE, bmap, bzero, first\_data\_block, fs\_dprintf, get\_hdsiz, mallocn, NULL, num\_bmap\_blocks, ROOT\_INODE\_BLOCK, and size.

Referenced by init\_bmap(), and load\_bmap().

**5.20.2.8 void mark\_block ( block\_nr blk\_nr, bool flag )**

Function to mark a block number as free (flag = 0) or used (flag = 1).

**Parameters**

*blk\_nr* block number which should be marked

*flag* value with whom the block should be marked (1 | 0)

References BLOCK\_SIZE, bmap, FIRST\_BMAP\_BLOCK, fs\_dprintf, TRUE, and wrt\_block().

Referenced by alloc\_block(), delete\_entry(), fs\_truncate(), init\_bmap(), insert\_file\_into\_dir(), and test\_bmap().

**5.20.2.9 void write\_bmap ( )**

Writes the block bitmap to HD.

References BLOCK\_SIZE, bmap, FIRST\_BMAP\_BLOCK, fs\_dprintf, num\_bmap\_blocks, and wrt\_block().

Referenced by fs\_shutdown().

### 5.20.3 Variable Documentation

#### 5.20.3.1 uint8\* bmap

Central block bitmap structure.

a bit larger than NUM\_BLOCKS\_ON\_HD / 8

Referenced by dump\_bmap(), fs\_init(), fs\_shutdown(), init\_super\_block(), is\_allocated\_block(), load\_bmap(), malloc\_bmap(), mark\_block(), and write\_bmap().

#### 5.20.3.2 uint32 first\_data\_block

Referenced by init\_bmap(), init\_super\_block(), and malloc\_bmap().

#### 5.20.3.3 uint32 num\_bmap\_blocks

Referenced by init\_super\_block(), load\_bmap(), malloc\_bmap(), and write\_bmap().

## 5.21 src/kernel/fs/fs\_buf.c File Reference

Basic buffer functions.

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/string.h"
#include "fs_const.h"
#include "fs_types.h"
#include "fs_buf.h"
#include "fs_block_dev.h"
```

### Functions

- void `clear_cache` (`block_cache *cache`)  
*Clears a struct of type 'block\_cache'.*
- void `clear_buffer` (`uint8 buffer[BLOCK_SIZE]`)  
*Clears a array of bytes = a buffer.*

### 5.21.1 Detailed Description

Basic buffer functions. Types of temporary caches (= [addr][buffer]) used: 1) read\_cache 2) write\_cache 3) addr\_cache 4) d\_inode\_cache 5) m\_inode\_cache

#### Author

Vincenz Doelle

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

## 5.21.2 Function Documentation

### 5.21.2.1 void clear\_buffer ( uint8 *buffer*[BLOCK\_SIZE] )

Clears a array of bytes = a buffer.

**Parameters**

*buffer* pointer to the array/buffer

References bzero.

### 5.21.2.2 void clear\_cache ( block\_cache \* *cache* )

Clears a struct of type 'block\_cache'.

**Parameters**

*cache* pointer to a cache

References block\_buffer::block\_nr, BLOCK\_SIZE, bzero, and block\_buffer::cache.

Referenced by cache\_block(), and wrt\_block().

## 5.22 src/kernel/fs/fs\_buf.h File Reference

Basic buffer/cache definitions.

### Functions

- void [clear\\_cache](#) (block\_cache \*cache)  
*Clears a struct of type 'block\_cache'.*
- void [clear\\_buffer](#) (uint8 buffer[BLOCK\_SIZE])  
*Clears a array of bytes = a buffer.*

## Variables

- `uint8 write_buffer [BLOCK_SIZE]`  
*A flat buffer/ byte array.*
- `uint8 read_buffer [BLOCK_SIZE]`  
*A flat buffer/ byte array.*
- `block_cache read_cache`  
*A cache for reading a block.*
- `block_cache write_cache`  
*A cache for writing a block.*
- `d_inode d_inode_cache`  
*A disk inode cache.*
- `m_inode m_inode_cache`  
*A memory inode cache.*
- `dir_entry dir_cache [DIR_ENTRIES_PER_BLOCK]`  
*A directory cache.*
- `block_nr addr_cache [ADDRS_PER_BLOCK]`

### 5.22.1 Detailed Description

Basic buffer/cache definitions. (cache = [addr][buffer])

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.22.2 Function Documentation

#### 5.22.2.1 void clear\_buffer ( uint8 buffer[BLOCK\_SIZE] )

Clears a array of bytes = a buffer.

#### Parameters

*buffer* pointer to the array/buffer

References bzero.

### 5.22.2.2 void clear\_cache ( block\_cache \* cache )

Clears a struct of type 'block\_cache'.

#### Parameters

*cache* pointer to a cache

References block\_buffer::block\_nr, BLOCK\_SIZE, bzero, and block\_buffer::cache.

Referenced by cache\_block(), and wrt\_block().

## 5.22.3 Variable Documentation

### 5.22.3.1 block\_nr addr\_cache[ADDRS\_PER\_BLOCK]

Referenced by fs\_truncate(), and get\_data\_block().

### 5.22.3.2 d\_inode d\_inode\_cache

A disk inode cache.

Referenced by read\_minode(), and write\_inode().

### 5.22.3.3 dir\_entry dir\_cache[DIR\_ENTRIES\_PER\_BLOCK]

A directory cache.

(4 bytes for block\_nr, NAME\_SIZE bytes for the name)

Referenced by delete\_file\_from\_dir(), insert\_file\_into\_dir(), and rfsearch().

### 5.22.3.4 m\_inode m\_inode\_cache

A memory inode cache.

Referenced by rfsearch().

### 5.22.3.5 uint8 read\_buffer[BLOCK\_SIZE]

A flat buffer/ byte array.

### 5.22.3.6 block\_cache read\_cache

A cache for reading a block.

Referenced by cache\_block(), clear\_block(), fs\_read(), fs\_write(), and rd\_block().

### 5.22.3.7 uint8 write\_buffer[BLOCK\_SIZE]

A flat buffer/ byte array.

### 5.22.3.8 block\_cache write\_cache

A cache for writing a block.

Referenced by wrt\_block().

## 5.23 src/kernel/fs/fs\_const.h File Reference

Basic constant definitions.

### Defines

- #define `BLOCK_SIZE` 512
- #define `NAME_SIZE` 28
- #define `DIR_ENTRY_SIZE` sizeof (struct `dir_entry`)
- #define `DISK_INODE_SIZE` sizeof (struct `d_inode`)
- #define `MEM_INODE_SIZE` sizeof (struct `m_inode`)
- #define `SUPER_SIZE` sizeof (struct `super_block`)
- #define `INODES_PER_BLOCK` 1
- #define `DIR_ENTRIES_PER_BLOCK` ((`BLOCK_SIZE`)/(`DIR_ENTRY_SIZE`))
- #define `NUM_FILES` 10
- #define `NUM_PROC_FILES` 20
- #define `NUM_INODES` 10
- #define `ADDR_SIZE` 4
- #define `ADDRS_PER_BLOCK` ((`BLOCK_SIZE`)/(`ADDR_SIZE`))
- #define `NUM_DIRECT_POINTER` 30
- #define `BYTES_DIRECT` ((`NUM_DIRECT_POINTER`) \* (`BLOCK_SIZE`))
- #define `BYTES_SINGLE_INDIRECT` ((`ADDRS_PER_BLOCK`) \* (`BLOCK_SIZE`))
- #define `BYTES_DOUBLE_INDIRECT` ((`ADDRS_PER_BLOCK`) \* (`ADDRS_PER_BLOCK`) \* (`BLOCK_SIZE`))
- #define `BOOT_BLOCK` ((`block_nr`) 0)
- #define `SUPER_BLOCK` ((`block_nr`) 1)
- #define `ROOT_INODE` ((`inode_nr`) 0)
- #define `ROOT_INODE_BLOCK` ((`block_nr`) 2)
- #define `FIRST_BMAP_BLOCK` ((`block_nr`) 3)
- #define `DATA_FILE` 1
- #define `DIRECTORY` 2
- #define `CLEAN` 0
- #define `DIRTY` 1
- #define `NOT_EXISTENT` -1
- #define `NOT_FOUND` -1
- #define `NOT_POSSIBLE` -1
- #define `MAGIC_NUMBER` 280187

### 5.23.1 Detailed Description

Basic constant definitions. This file defines constants used throughout the file system.

**Author**

Vincenz Doelle

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

### 5.23.2 Define Documentation

#### 5.23.2.1 `#define ADDR_SIZE 4`

#### 5.23.2.2 `#define ADDRS_PER_BLOCK ((BLOCK_SIZE)/(ADDR_SIZE))`

Referenced by `dump_consts()`, and `get_data_block()`.

#### 5.23.2.3 `#define BLOCK_SIZE 512`

Referenced by `clear_block()`, `clear_cache()`, `fs_read()`, `fs_truncate()`, `fs_write()`, `get_data_block()`, `init_super_block()`, `load_bmap()`, `malloc_bmap()`, `mark_block()`, and `write_bmap()`.

#### 5.23.2.4 `#define BOOT_BLOCK ((block_nr) 0)`

Referenced by `dump_bmap()`.

#### 5.23.2.5 `#define BYTES_DIRECT ((NUM_DIRECT_POINTER) * (BLOCK_SIZE))`

Referenced by `dump_consts()`, and `get_data_block()`.

#### 5.23.2.6 `#define BYTES_DOUBLE_INDIRECT ((ADDRS_PER_BLOCK) * (ADDRS_PER_BLOCK) * (BLOCK_SIZE))`

Referenced by `dump_consts()`, and `get_data_block()`.

#### 5.23.2.7 `#define BYTES_SINGLE_INDIRECT ((ADDRS_PER_BLOCK) * (BLOCK_SIZE))`

Referenced by `dump_consts()`, and `get_data_block()`.

**5.23.2.8 #define CLEAN 0****5.23.2.9 #define DATA\_FILE 1**

Referenced by do\_mkfile(), test\_file\_table(), test\_ls(), test\_rw\_qualitative(), and test\_rw\_quantitative().

**5.23.2.10 #define DIR\_ENTRIES\_PER\_BLOCK ((BLOCK\_SIZE)/(DIR\_ENTRY\_SIZE))**

Referenced by do\_read(), and dump\_consts().

**5.23.2.11 #define DIR\_ENTRY\_SIZE sizeof (struct dir\_entry)****5.23.2.12 #define DIRECTORY 2**

Referenced by create\_root(), delete\_file\_from\_dir(), do\_mkdir(), do\_read(), insert\_file\_into\_dir(), shell\_cmd\_ls(), sys\_unlink(), sys\_write(), test\_create(), test\_delete(), test\_file\_table(), test\_ls(), and test\_open\_close().

**5.23.2.13 #define DIRTY 1****5.23.2.14 #define DISK\_INODE\_SIZE sizeof (struct d\_inode)**

Referenced by dump\_consts().

**5.23.2.15 #define FIRST\_BMAP\_BLOCK ((block\_nr) 3)**

Referenced by load\_bmap(), mark\_block(), and write\_bmap().

**5.23.2.16 #define INODES\_PER\_BLOCK 1**

Referenced by dump\_consts().

**5.23.2.17 #define MAGIC\_NUMBER 280187**

Referenced by load\_fs().

**5.23.2.18 #define MEM\_INODE\_SIZE sizeof (struct m\_inode)**

Referenced by dump\_consts().

**5.23.2.19 #define NAME\_SIZE 28**

Referenced by create\_entry(), and delete\_entry().



**5.23.2.20 #define NOT\_EXISTENT -1****5.23.2.21 #define NOT\_FOUND -1**

Referenced by contains\_filename(), delete\_file\_from\_dir(), do\_file\_exists(), fs\_create\_delete(), fs\_open(), fs\_read(), fs\_write(), insert\_file(), insert\_file\_into\_dir(), rfsearch(), sys\_stat(), and sys\_unlink().

**5.23.2.22 #define NOT\_POSSIBLE -1**

Referenced by delete\_file\_from\_dir(), fs\_create\_delete(), insert\_file\_into\_dir(), sys\_open(), test\_file\_table(), test\_rw\_qualitative(), and test\_rw\_quantitative().

**5.23.2.23 #define NUM\_DIRECT\_POINTER 30**

Referenced by init\_super\_block().

**5.23.2.24 #define NUM\_FILES 10**

Referenced by dump\_consts().

**5.23.2.25 #define NUM\_INODES 10**

Referenced by dump\_consts().

**5.23.2.26 #define NUM\_PROC\_FILES 20**

Referenced by dump\_consts().

**5.23.2.27 #define ROOT\_INODE ((inode\_nr) 0)**

Referenced by create\_root(), and load\_root().

**5.23.2.28 #define ROOT\_INODE\_BLOCK ((block\_nr) 2)**

Referenced by create\_root(), delete\_file\_from\_dir(), dump\_consts(), free\_file(), insert\_file\_into\_dir(), load\_root(), and malloc\_bmap().

**5.23.2.29 #define SUPER\_BLOCK ((block\_nr) 1)**

Referenced by load\_super\_block(), and write\_super\_block().

**5.23.2.30 #define SUPER\_SIZE sizeof (struct super\_block)**

Referenced by dump\_consts().

## 5.24 src/kernel/fs/fs\_create\_delete.c File Reference

Functions concerning syscalls "create" and "delete".

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/string.h"
#include "../include/stdlib.h"
#include "../include/debug.h"
#include "fs_const.h"
#include "fs_types.h"
#include "fs_block_dev.h"
#include "fs_buf.h"
#include "fs_bmap.h"
#include "fs_inode_table.h"
#include "fs_file_table.h"
#include "fs_dir.h"
#include "fs_io_functions.h"
```

### Functions

- [bool fs\\_create](#) (char \*abs\_path, int data\_type)  
*Creates a file from absolute path by inserting the name into the containing directory, creating a new inode and writing it to HD.*
- [bool fs\\_delete](#) (char \*abs\_path)  
*Deletes a file by removing it from the containing directory.*
- [bool fs\\_create\\_delete](#) (char \*abs\_path, int mode, int data\_type)  
*Handles the creation and deletion process.*
- [bool fs\\_truncate](#) (char \*abs\_path, uint32 size)  
*Set the size of file abs\_path to be exactly size, deleting or allocating blocks as necessary.*

### 5.24.1 Detailed Description

Functions concerning syscalls "create" and "delete".

#### Author

Vincenz Doelle

#### LastChangedBy:

wrtlprnft

## Version

### Rev:

14

## 5.24.2 Function Documentation

### 5.24.2.1 `bool fs_create ( char * abs_path, int data_type )`

Creates a file from absolute path by inserting the name into the containing directory, creating a new inode and writing it to HD.

#### Parameters

*abs\_path* absolute file path

*data\_type* DATA\_FILE | DIRECTORY (

#### See also

[fs\\_const.h](#))

#### Returns

result status of the create operation

References CREATE, and fs\_create\_delete().

Referenced by chips\_create(), chips\_mkdir(), do\_create(), do\_mkdir(), test\_create(), test\_delete(), test\_open\_close(), test\_rw\_qualitative(), and test\_rw\_quantitative().

### 5.24.2.2 `bool fs_create_delete ( char * abs_path, int mode, int data_type )`

Handles the creation and deletion process.

#### Parameters

*abs\_path* absolute file path

*mode* CREATE | DELETE (

#### See also

[fs\\_const.h](#))

#### Parameters

*data\_type* DATA\_FILE | DIRECTORY (

#### See also

[fs\\_const.h](#))

#### Returns

result status of the create/delete operation

References CREATE, DELETE, delete\_file\_from\_dir(), FALSE, free, fs\_dprintf, fs\_truncate(), get\_filename(), get\_path(), inode, insert\_file\_into\_dir(), new\_minode(), NOT\_FOUND, NOT\_POSSIBLE, NULL, printf, RED, search\_file(), strcmp, and write\_inode().

Referenced by fs\_create(), and fs\_delete().

### 5.24.2.3 bool fs\_delete ( char \* abs\_path )

Deletes a file by removing it from the containing directory.

#### Parameters

*abs\_path* absolute file path

#### Returns

result status of the delete operation

References DELETE, fs\_create\_delete(), and NULL.

Referenced by chips\_unlink(), do\_remove(), and test\_delete().

### 5.24.2.4 bool fs\_truncate ( char \* abs\_path, uint32 size )

Set the size of file *abs\_path* to be exactly *size*, deleting or allocating blocks as necessary.

References addr\_cache, BLOCK\_SIZE, file::f\_inode, FALSE, fs\_close(), fs\_dprintf, fs\_open(), get\_data\_block(), get\_file(), m\_inode::i\_direct\_pointer, m\_inode::i\_double\_indirect\_pointer, m\_inode::i\_single\_indirect\_pointer, m\_inode::i\_size, i\_size, inode, mark\_block(), MAX, NULL, rd\_block(), TRUE, write\_inode(), and wrt\_block().

Referenced by fs\_create\_delete().

## 5.25 src/kernel/fs/fs\_dir.c File Reference

Functions on directories.

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/string.h"
#include "../include/stdlib.h"
#include "../include/debug.h"
#include "fs_const.h"
#include "fs_types.h"
#include "fs_dir.h"
#include "fs_block_dev.h"
#include "fs_bmap.h"
#include "fs_buf.h"
#include "fs_inode_table.h"
```

```
#include "fs_io_functions.h"
```

## Functions

- [block\\_nr find\\_filename](#) ([dir\\_entry](#) file\_list[DIR\_ENTRIES\_PER\_BLOCK], char \*name)  
*Find a filename within a block of directory entries.*
- [block\\_nr insert\\_file\\_into\\_dir](#) ([block\\_nr](#) dir\_inode\_blk, char \*name)  
*Insert a new filename into a directory.*
- [block\\_nr delete\\_file\\_from\\_dir](#) ([block\\_nr](#) dir\_inode\_blk, char \*name)  
*Delete a new filename from a directory.*
- [uint32 create\\_entry](#) ([dir\\_entry](#) file\_list[DIR\_ENTRIES\_PER\_BLOCK], [block\\_nr](#) blk\_nr, char \*name)  
*Inserts a new file into a directory.*
- [uint32 delete\\_entry](#) ([dir\\_entry](#) file\_list[DIR\_ENTRIES\_PER\_BLOCK], char \*name)  
*Removes a file from a directory.*
- [bool contains\\_filename](#) ([dir\\_entry](#) file\_list[DIR\_ENTRIES\_PER\_BLOCK], char \*name)  
*Checks whether a directory contains a special filename.*
- [block\\_nr search\\_file](#) (char \*path)  
*Prepare recursive search according to strtok() definition.*
- [block\\_nr rfsearch](#) ([block\\_nr](#) crt\_dir, char \*path, char \*tok, char delim[ ])  
*Recursive search for files.*
- [char \\* get\\_filename](#) (char \*abs\_path)  
*Extract the filename WITHOUT the path from the absolute path.*
- [char \\* get\\_path](#) (char \*abs\_path)  
*Extract the path WITHOUT the filename from the absolute path.*

### 5.25.1 Detailed Description

Functions on directories.

#### Author

Vincenz Doelle

#### LastChangedBy:

wrtlprnft

#### Version

#### Rev:

14

## 5.25.2 Function Documentation

### 5.25.2.1 `bool contains_filename ( dir_entry file_list[DIR_ENTRIES_PER_BLOCK], char * name )`

Checks whether a directory contains a special filename.

#### Parameters

*file\_list* directory's file list of type `dir_entry`  
*name* file name which should be found

#### Returns

filename found?

References `find_filename()`, and `NOT_FOUND`.

Referenced by `delete_file_from_dir()`, and `insert_file_into_dir()`.

### 5.25.2.2 `uint32 create_entry ( dir_entry file_list[DIR_ENTRIES_PER_BLOCK], block_nr blk_nr, char * name )`

Inserts a new file into a directory.

#### Parameters

*file\_list* file list of type `dir_entry`  
*blk\_nr* block number of the new file  
*name* name of the new file

#### Returns

insert position

References `memcpy`, `NAME_SIZE`, and `strcmp`.

Referenced by `insert_file_into_dir()`.

### 5.25.2.3 `uint32 delete_entry ( dir_entry file_list[DIR_ENTRIES_PER_BLOCK], char * name )`

Removes a file from a directory.

#### Parameters

*file\_list* file list of type `dir_entry`  
*name* name of the file

#### Returns

status of remove operation

References `bzero`, `FALSE`, `fs_dprintf`, `inode`, `mark_block()`, `NAME_SIZE`, and `strcmp`.

Referenced by `delete_file_from_dir()`.

#### 5.25.2.4 `block_nr delete_file_from_dir ( block_nr dir_inode_blk, char * name )`

Delete a new filename from a directory.

##### Parameters

*dir\_inode\_blk* block number of the directory inode  
*name* file name which should be deleted

##### Returns

block number of deleted file's inode

References contains\_filename(), delete\_entry(), dir\_cache, DIRECTORY, FALSE, free, fs\_dprintf, fs\_read(), get\_data\_block(), m\_inode::i\_mode, m\_inode::i\_modify\_ts, m\_inode::i\_size, mallocn, NOT\_FOUND, NOT\_POSSIBLE, printf, read\_minode(), RED, root, ROOT\_INODE\_BLOCK, write\_inode(), and wrt\_block().

Referenced by fs\_create\_delete().

#### 5.25.2.5 `block_nr find_filename ( dir_entry file_list[DIR_ENTRIES_PER_BLOCK], char * name )`

Find a filename within a block of directory entries.

Format of an entry: (struct [dir\\_entry](#)): [block\_nr, name].

##### Parameters

*file\_list* directory's file list of type [dir\\_entry](#)  
*name* file name which should be found

##### Returns

block\_nr of the file if found

References fs\_dprintf, inode, and strcmp.

Referenced by contains\_filename(), and rfsearch().

#### 5.25.2.6 `char* get_filename ( char * abs_path )`

Extract the filename WITHOUT the path from the absolute path.

##### Parameters

*abs\_path* absolute path to file

##### Returns

only the filename

References bzero, fs\_dprintf, mallocn, memcpy, NULL, and strlen.

Referenced by fs\_create\_delete().

**5.25.2.7 char\* get\_path ( char \* abs\_path )**

Extract the path WITHOUT the filename from the absolute path.

**Parameters**

*abs\_path* absolute path to file

**Returns**

only the path

References bzero, fs\_dprintf, mallocn, memcpy, NULL, and strlen.

Referenced by fs\_create\_delete().

**5.25.2.8 block\_nr insert\_file\_into\_dir ( block\_nr dir\_inode\_blk, char \* name )**

Insert a new filename into a directory.

**Parameters**

*dir\_inode\_blk* block number of the directory inode

*name* file name which should be inserted

**Returns**

block number of the newly allocated inode block for 'name'

References alloc\_block(), clear\_block(), contains\_filename(), create\_entry(), dir\_cache, DIRECTORY, FALSE, free, fs\_dprintf, fs\_read(), get\_data\_block(), m\_inode::i\_mode, m\_inode::i\_modify\_ts, m\_inode::i\_size, mallocn, mark\_block(), NOT\_FOUND, NOT\_POSSIBLE, printf, read\_minode(), RED, root, ROOT\_INODE\_BLOCK, TRUE, write\_inode(), and wrt\_block().

Referenced by fs\_create\_delete().

**5.25.2.9 block\_nr rfsearch ( block\_nr crt\_dir, char \* path, char \* tok, char delim[] )**

Recursive search for files.

**Parameters**

*crt\_dir* current directory to search in

*path* absolute path to file

*tok* a token of the path

*delim* mostly '/'

**Returns**

block number of the file's inode (if found)

References dir\_cache, FALSE, find\_filename(), fs\_dprintf, fs\_read(), m\_inode::i\_adr, m\_inode\_cache, NOT\_FOUND, NULL, read\_minode(), rfsearch(), root, and strsep.

Referenced by rfsearch(), and search\_file().



### 5.25.2.10 block\_nr search\_file ( char \* path )

Prepare recursive search according to strtok() definition.

#### Parameters

*path* absolute path to file

#### Returns

block number of the file's inode (if found)

References free, fs\_dprintf, m\_inode::i\_adr, rfsearch(), root, strcmp, strdup, and strsep.

Referenced by do\_file\_exists(), fs\_create\_delete(), and fs\_open().

## 5.26 src/kernel/fs/fs\_dir.h File Reference

Basic functions concerning directories.

### Functions

- [block\\_nr find\\_filename](#) (dir\_entry file\_list[DIR\_ENTRIES\_PER\_BLOCK], char \*name)  
*Find a filename within a block of directory entries.*
- [bool contains\\_filename](#) (dir\_entry file\_list[DIR\_ENTRIES\_PER\_BLOCK], char \*name)  
*Checks whether a directory contains a special filename.*
- [block\\_nr search\\_file](#) (char \*path)  
*Prepare recursive search according to strtok() definition.*
- [block\\_nr rfsearch](#) (block\_nr crt\_dir, char \*path, char \*tok, char delim[ ])  
*Recursive search for files.*
- [block\\_nr insert\\_file\\_into\\_dir](#) (block\_nr dir\_inode\_blk, char \*name)  
*Insert a new filename into a directory.*
- [uint32 create\\_entry](#) (dir\_entry file\_list[DIR\_ENTRIES\_PER\_BLOCK], block\_nr blk\_nr, char \*name)  
*Inserts a new file into a directory.*
- [block\\_nr delete\\_file\\_from\\_dir](#) (block\_nr dir\_inode\_blk, char \*name)  
*Delete a new filename from a directory.*
- [uint32 delete\\_entry](#) (dir\_entry file\_list[DIR\_ENTRIES\_PER\_BLOCK], char \*name)  
*Removes a file from a directory.*
- char \* [get\\_filename](#) (char \*abs\_path)  
*Extract the filename WITHOUT the path from the absolute path.*
- char \* [get\\_path](#) (char \*abs\_path)  
*Extract the path WITHOUT the filename from the absolute path.*

## 5.26.1 Detailed Description

Basic functions concerning directories.

### Author

Vincenz Doelle

### LastChangedBy:

dtraytel

### Version

### Rev:

12

## 5.26.2 Function Documentation

### 5.26.2.1 `bool contains_filename ( dir_entry file_list[DIR_ENTRIES_PER_BLOCK], char * name )`

Checks whether a directory contains a special filename.

#### Parameters

*file\_list* directory's file list of type `dir_entry`  
*name* file name which should be found

#### Returns

filename found?

References `find_filename()`, and `NOT_FOUND`.

Referenced by `delete_file_from_dir()`, and `insert_file_into_dir()`.

### 5.26.2.2 `uint32 create_entry ( dir_entry file_list[DIR_ENTRIES_PER_BLOCK], block_nr blk_nr, char * name )`

Inserts a new file into a directory.

#### Parameters

*file\_list* file list of type `dir_entry`  
*blk\_nr* block number of the new file  
*name* name of the new file

#### Returns

insert position

References `memcpy`, `NAME_SIZE`, and `strcmp`.

Referenced by `insert_file_into_dir()`.

### 5.26.2.3 uint32 delete\_entry ( dir\_entry file\_list[DIR\_ENTRIES\_PER\_BLOCK], char \* name )

Removes a file from a directory.

#### Parameters

*file\_list* file list of type [dir\\_entry](#)  
*name* name of the file

#### Returns

status of remove operation

References bzero, FALSE, fs\_dprintf, inode, mark\_block(), NAME\_SIZE, and strcmp.

Referenced by delete\_file\_from\_dir().

### 5.26.2.4 block\_nr delete\_file\_from\_dir ( block\_nr dir\_inode\_blk, char \* name )

Delete a new filename from a directory.

#### Parameters

*dir\_inode\_blk* block number of the directory inode  
*name* file name which should be deleted

#### Returns

block number of deleted file's inode

References contains\_filename(), delete\_entry(), dir\_cache, DIRECTORY, FALSE, free, fs\_dprintf, fs\_read(), get\_data\_block(), m\_inode::i\_mode, m\_inode::i\_modify\_ts, m\_inode::i\_size, mallocn, NOT\_FOUND, NOT\_POSSIBLE, printf, read\_mininode(), RED, root, ROOT\_INODE\_BLOCK, write\_inode(), and wrt\_block().

Referenced by fs\_create\_delete().

### 5.26.2.5 block\_nr find\_filename ( dir\_entry file\_list[DIR\_ENTRIES\_PER\_BLOCK], char \* name )

Find a filename within a block of directory entries.

Format of an entry: (struct [dir\\_entry](#)): [block\_nr, name].

#### Parameters

*file\_list* directory's file list of type [dir\\_entry](#)  
*name* file name which should be found

#### Returns

block\_nr of the file if found

References fs\_dprintf, inode, and strcmp.

Referenced by contains\_filename(), and rfsearch().

#### 5.26.2.6 char\* get\_filename ( char \* *abs\_path* )

Extract the filename WITHOUT the path from the absolute path.

##### Parameters

*abs\_path* absolute path to file

##### Returns

only the filename

References bzero, fs\_dprintf, mallocn, memcpy, NULL, and strlen.

Referenced by fs\_create\_delete().

#### 5.26.2.7 char\* get\_path ( char \* *abs\_path* )

Extract the path WITHOUT the filename from the absolute path.

##### Parameters

*abs\_path* absolute path to file

##### Returns

only the path

References bzero, fs\_dprintf, mallocn, memcpy, NULL, and strlen.

Referenced by fs\_create\_delete().

#### 5.26.2.8 block\_nr insert\_file\_into\_dir ( block\_nr *dir\_inode\_blk*, char \* *name* )

Insert a new filename into a directory.

##### Parameters

*dir\_inode\_blk* block number of the directory inode

*name* file name which should be inserted

##### Returns

block number of the newly allocated inode block for 'name'

References alloc\_block(), clear\_block(), contains\_filename(), create\_entry(), dir\_cache, DIRECTORY, FALSE, free, fs\_dprintf, fs\_read(), get\_data\_block(), m\_inode::i\_mode, m\_inode::i\_modify\_ts, m\_inode::i\_size, mallocn, mark\_block(), NOT\_FOUND, NOT\_POSSIBLE, printf, read\_minode(), RED, root, ROOT\_INODE\_BLOCK, TRUE, write\_inode(), and wrt\_block().

Referenced by fs\_create\_delete().

### 5.26.2.9 block\_nr rfsearch ( block\_nr crt\_dir, char \* path, char \* tok, char delim[] )

Recursive search for files.

#### Parameters

*crt\_dir* current directory to search in  
*path* absolute path to file  
*tok* a token of the path  
*delim* mostly '/'

#### Returns

block number of the file's inode (if found)

References `dir_cache`, `FALSE`, `find_filename()`, `fs_dprintf`, `fs_read()`, `m_inode::i_adr`, `m_inode_cache`, `NOT_FOUND`, `NULL`, `read_minode()`, `rfsearch()`, `root`, and `strsep`.

Referenced by `rfsearch()`, and `search_file()`.

### 5.26.2.10 block\_nr search\_file ( char \* path )

Prepare recursive search according to `strtok()` definition.

#### Parameters

*path* absolute path to file

#### Returns

block number of the file's inode (if found)

## 5.27 src/kernel/fs/fs\_file\_table.c File Reference

The file descriptor table functions.

```
#include "../include/const.h"  
#include "../include/types.h"  
#include "../include/string.h"  
#include "../include/stdlib.h"  
#include "../include/debug.h"  
#include "fs_const.h"  
#include "fs_types.h"  
#include "fs_file_table.h"  
#include "fs_inode_table.h"
```

#### Functions

- void [init\\_file\\_table\(\)](#)

*Initialize the global file table with NULL elements.*

- void `init_proc_file_table` (`proc_file` pft[`NUM_PROC_FILES`])  
*Initialize the process file table with NULL elements.*
- `file_nr` `insert_file` (`m_inode` \*inode, char \*name, `uint8` mode)  
*Insert a new file to the global file table.*
- `file_nr` `insert_proc_file` (`proc_file` pft[`NUM_PROC_FILES`], `file_nr` glo\_fd)  
*Insert a new file to a given process file table.*
- `file` \* `alloc_file` ()  
*Allocate an unused file in the global file table.*
- `proc_file` \* `alloc_proc_file` (`proc_file` pft[`NUM_PROC_FILES`])  
*Allocate an unused file in the process file table.*
- `file` \* `get_file` (`file_nr` fd)  
*Find a file in the global file table.*
- `proc_file` \* `get_proc_file` (`proc_file` pft[`NUM_PROC_FILES`], `file_nr` fd)  
*Find a file in the process file table.*
- void `free_file` (`file_nr` fd)  
*Reset a file.*
- void `free_proc_file` (`proc_file` pft[`NUM_PROC_FILES`], `file_nr` fd)  
*Reset a process file.*
- void `inc_count` (`file_nr` fd)  
*Increment the reference counter of a file.*
- `size_t` `lseek` (`proc_file` pft[`NUM_PROC_FILES`], `file_nr` fd, `sint32` offset, `uint32` origin)  
*Move file position to current file position + offset.*
- `file_nr` `name2desc` (char \*name)  
*Find the descriptor of a file with the file name.*
- `file_nr` `inode2desc` (`m_inode` \*inode)  
*Find the descriptor of a file via its inode (number).*
- `bool` `contains_file` (`file_nr` fd)  
*Look whether the global file table contains a special file descriptor.*
- `file_info_t` \* `get_file_info` (`file_nr` fd, `file_info_t` \*info)  
*Get file further information.*
- void `dump_file` (`file` \*f)  
*Print out a file for debug purposes.*

- void `dump_files` ()  
*Print out file table for debug purposes.*
- void `dump_proc_file` (`proc_file *pf`)  
*Print out a process file for debug purposes.*
- void `dump_proc_files` (`proc_file pft[NUM_PROC_FILES]`)  
*Print out process file table for debug purposes.*

### 5.27.1 Detailed Description

The file descriptor table functions.

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.27.2 Function Documentation

#### 5.27.2.1 `file* alloc_file ( )`

Allocate an unused file in the global file table.

#### Returns

pointer to the allocated file

References `file::f_count`, `file::f_desc`, `gft`, and `NIL_FILE`.

Referenced by `insert_file()`.

#### 5.27.2.2 `proc_file* alloc_proc_file ( proc_file pft[NUM_PROC_FILES] )`

Allocate an unused file in the process file table.

#### Parameters

*pft* process file table

#### Returns

pointer to the allocated file

References `NIL_PROC_FILE`, and `proc_file::pf_desc`.

Referenced by `insert_proc_file()`.

### 5.27.2.3 `bool contains_file ( file_nr fd )`

Look whether the global file table contains a special file descriptor.

#### Parameters

*fd* file descriptor

#### Returns

operation status

References `get_file()`, and `NULL`.

Referenced by `test_file_table()`.

### 5.27.2.4 `void dump_file ( file * f )`

Print out a file for debug purposes.

#### Parameters

*f* file to be printed

References `file::f_count`, `file::f_desc`, `file::f_inode`, `file::f_mode`, `file::f_name`, and `fs_dprintf`.

Referenced by `dump_files()`.

### 5.27.2.5 `void dump_files ( )`

Print out file table for debug purposes.

References `dump_file()`, `fs_dprintf`, and `gft`.

Referenced by `fs_close()`, `test_close()`, `test_file_table()`, and `test_open_close()`.

### 5.27.2.6 `void dump_proc_file ( proc_file * pf )`

Print out a process file for debug purposes.

#### Parameters

*pf* file to be printed

References `fs_dprintf`, `proc_file::pf_desc`, `proc_file::pf_f_desc`, and `proc_file::pf_pos`.

Referenced by `dump_proc_files()`.



**5.27.2.7 void dump\_proc\_files ( proc\_file *pft*[NUM\_PROC\_FILES] )**

Print out process file table for debug purposes.

References dump\_proc\_file(), and fs\_dprintf.

Referenced by test\_close().

**5.27.2.8 void free\_file ( file\_nr *fd* )**

Reset a file.

Free memory if possible.

**Parameters**

*fd* file descriptor

References file::f\_count, file::f\_desc, file::f\_inode, file::f\_name, free, free\_inode(), get\_file(), m\_inode::i\_adr, m\_inode::i\_num, NULL, and ROOT\_INODE\_BLOCK.

Referenced by fs\_close(), and test\_file\_table().

**5.27.2.9 void free\_proc\_file ( proc\_file *pft*[NUM\_PROC\_FILES], file\_nr *fd* )**

Reset a process file.

**Parameters**

*pft* process file table

*fd* the file

References get\_proc\_file(), NULL, proc\_file::pf\_desc, proc\_file::pf\_f\_desc, and proc\_file::pf\_pos.

Referenced by do\_close\_pf(), and test\_open\_close().

**5.27.2.10 file\* get\_file ( file\_nr *fd* )**

Find a file in the global file table.

**Parameters**

*fd* file descriptor

**Returns**

pointer to the found file

References gft.

Referenced by contains\_file(), do\_read(), do\_write(), free\_file(), fs\_close(), fs\_truncate(), get\_file\_info(), inc\_count(), insert\_proc\_file(), sys\_seek(), sys\_unlink(), test\_rw\_qualitative(), and test\_rw\_quantitative().

**5.27.2.11 file\_info\_t\* get\_file\_info ( file\_nr *fd*, file\_info\_t \* *info* )**

Get file further information.

**Parameters**

*fd* file descriptor

**Returns**

file information

References file\_info::create\_ts, file::f\_count, file::f\_inode, file::f\_mode, file::f\_name, get\_file(), i\_create\_ts, i\_modify\_ts, i\_size, file\_info::mode, file\_info::modify\_ts, file\_info::name, file\_info::num\_links, file\_info::size, and strcmpy.

Referenced by chips\_unlink(), do\_read(), sys\_stat(), sys\_unlink(), and sys\_write().

**5.27.2.12 proc\_file\* get\_proc\_file ( proc\_file *pft*[NUM\_PROC\_FILES], file\_nr *fd* )**

Find a file in the process file table.

**Parameters**

*pft* process file table

*fd* file descriptor

**Returns**

pointer to the file found

Referenced by do\_close\_pf(), free\_proc\_file(), lseek(), sys\_read(), sys\_seek(), sys\_write(), and test\_PM().

**5.27.2.13 void inc\_count ( file\_nr *fd* )**

Increment the reference counter of a file.

**Parameters**

*fd* file descriptor

References file::f\_count, and get\_file().

Referenced by insert\_proc\_file().

**5.27.2.14 void init\_file\_table ( )**

Initialize the global file table with NULL elements.

References file::f\_desc, and gft.

Referenced by create\_fs(), load\_fs(), and test\_file\_table().

**5.27.2.15 void init\_proc\_file\_table ( proc\_file *pft*[NUM\_PROC\_FILES] )**

Initialize the process file table with NULL elements.

**Parameters**

*pft* process file table

References fs\_dprintf.

Referenced by pm\_create\_thread(), pm\_init(), test\_close(), test\_open\_close(), and test\_PM().

**5.27.2.16 file\_nr inode2desc ( m\_inode \* *inode* )**

Find the descriptor of a file via its inode (number).

**Parameters**

*inode* the file's inode

**Returns**

found file descriptor

References file::f\_desc, file::f\_inode, gft, and m\_inode::i\_num.

Referenced by test\_file\_table().

**5.27.2.17 file\_nr insert\_file ( m\_inode \* *inode*, char \* *name*, uint8 *mode* )**

Insert a new file to the global file table.

**Parameters**

*inode* corresponding inode

*name* filename

*mode* DATA\_FILE | DIRECTORY

**Returns**

assigned file descriptor

References alloc\_file(), file::f\_desc, file::f\_inode, file::f\_mode, file::f\_name, free, fs\_dprintf, name2desc(), NOT\_FOUND, NULL, and strdup.

Referenced by fs\_open(), and test\_file\_table().

**5.27.2.18 file\_nr insert\_proc\_file ( proc\_file *pft*[NUM\_PROC\_FILES], file\_nr *glo\_fd* )**

Insert a new file to a given process file table.

**Parameters**

*pft* process file table

*glo\_fd* global (for all processes) file descriptor

### Returns

assigned file descriptor

References `alloc_proc_file()`, `fs_dprintf`, `get_file()`, `inc_count()`, `NULL`, `proc_file::pf_desc`, `proc_file::pf_f_desc`, and `proc_file::pf_pos`.

Referenced by `sys_open()`, `sys_stat()`, `test_close()`, `test_open_close()`, and `test_PM()`.

#### 5.27.2.19 `size_t lseek ( proc_file pft[NUM_PROC_FILES], file_nr fd, sint32 offset, uint32 origin )`

Move file position to current file position + offset.

### Parameters

*pft* process file table

*fd* file descriptor

*offset* offset

*origin* original position

### Returns

new position or -1 if failed

References `get_proc_file()`, `NULL`, and `proc_file::pf_pos`.

Referenced by `do_lseek()`.

#### 5.27.2.20 `file_nr name2desc ( char * name )`

Find the descriptor of a file with the file name.

### Parameters

*name* file name

### Returns

found file descriptor or `NOT_FOUND`

References `file::f_desc`, `fs_dprintf`, `gft`, `NULL`, and `strcmp`.

Referenced by `fs_open()`, `insert_file()`, `test_file_table()`, `test_rw_qualitative()`, and `test_rw_quantitative()`.

## 5.28 `src/kernel/fs/fs_file_table.h` File Reference

The file descriptor table.

## Defines

- #define `NIL_FILE` -1
- #define `NIL_PROC_FILE` -1

## Functions

- void `init_file_table` ()  
*Initialize the global file table with NULL elements.*
- void `init_proc_file_table` (`proc_file` `proc_filp` [ ])
- void `dump_file` (`file` \*`f`)  
*Print out a file for debug purposes.*
- void `dump_files` ()  
*Print out file table for debug purposes.*
- void `dump_proc_file` (`proc_file` \*`pf`)  
*Print out a process file for debug purposes.*
- void `dump_proc_files` ()
- `file_nr` `insert_file` (`m_inode` \*`inode`, `char` \*`name`, `uint8` `mode`)  
*Insert a new file to the global file table.*
- `file_nr` `insert_proc_file` (`proc_file` `pft`[`NUM_PROC_FILES`], `file_nr` `glo_fd`)  
*Insert a new file to a given process file table.*
- `file` \* `alloc_file` ()  
*Allocate an unused file in the global file table.*
- `proc_file` \* `alloc_proc_file` (`proc_file` `pft`[`NUM_PROC_FILES`])  
*Allocate an unused file in the process file table.*
- `file` \* `get_file` (`file_nr` `fd`)  
*Find a file in the global file table.*
- `proc_file` \* `get_proc_file` (`proc_file` `pft`[`NUM_PROC_FILES`], `file_nr` `fd`)  
*Find a file in the process file table.*
- void `free_file` (`file_nr` `fd`)  
*Reset a file.*
- void `free_proc_file` (`proc_file` `pft`[`NUM_PROC_FILES`], `file_nr` `fd`)  
*Reset a process file.*
- `bool` `contains_file` (`file_nr` `fd`)  
*Look whether the global file table contains a special file descriptor.*
- `file_nr` `name2desc` (`char` \*`name`)  
*Find the descriptor of a file with the file name.*

- `file_nr inode2desc (m_inode *inode)`  
*Find the descriptor of a file via its inode (number).*
- `void inc_count (file_nr fd)`  
*Increment the reference counter of a file.*
- `size_t lseek (proc_file pft[NUM_PROC_FILES], file_nr fd, sint32 offset, uint32 origin)`  
*Move file position to current file position + offset.*

## Variables

- `file gft [NUM_FILES]`

### 5.28.1 Detailed Description

The file descriptor table.

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.28.2 Define Documentation

#### 5.28.2.1 #define NIL\_FILE -1

Referenced by `alloc_file()`, and `fs_close()`.

#### 5.28.2.2 #define NIL\_PROC\_FILE -1

Referenced by `alloc_proc_file()`.

### 5.28.3 Function Documentation

#### 5.28.3.1 file\* alloc\_file ( )

Allocate an unused file in the global file table.

**Returns**

pointer to the allocated file

References file::f\_count, file::f\_desc, gft, and NIL\_FILE.

Referenced by insert\_file().

**5.28.3.2 proc\_file\* alloc\_proc\_file ( proc\_file *pft*[NUM\_PROC\_FILES] )**

Allocate an unused file in the process file table.

**Parameters**

*pft* process file table

**Returns**

pointer to the allocated file

References NIL\_PROC\_FILE, and proc\_file::pf\_desc.

Referenced by insert\_proc\_file().

**5.28.3.3 bool contains\_file ( file\_nr *fd* )**

Look whether the global file table contains a special file descriptor.

**Parameters**

*fd* file descriptor

**Returns**

operation status

References get\_file(), and NULL.

Referenced by test\_file\_table().

**5.28.3.4 void dump\_file ( file \* *f* )**

Print out a file for debug purposes.

**Parameters**

*f* file to be printed

References file::f\_count, file::f\_desc, file::f\_inode, file::f\_mode, file::f\_name, and fs\_dprintf.

Referenced by dump\_files().

**5.28.3.5 void dump\_files ( )**

Print out file table for debug purposes.

References dump\_file(), fs\_dprintf, and gft.

Referenced by fs\_close(), test\_close(), test\_file\_table(), and test\_open\_close().

**5.28.3.6 void dump\_proc\_file ( proc\_file \* *pf* )**

Print out a process file for debug purposes.

**Parameters**

*pf* file to be printed

References fs\_dprintf, proc\_file::pf\_desc, proc\_file::pf\_f\_desc, and proc\_file::pf\_pos.

Referenced by dump\_proc\_files().

**5.28.3.7 void dump\_proc\_files ( )****5.28.3.8 void free\_file ( file\_nr *fd* )**

Reset a file.

Free memory if possible.

**Parameters**

*fd* file descriptor

References file::f\_count, file::f\_desc, file::f\_inode, file::f\_name, free, free\_inode(), get\_file(), m\_inode::i\_adr, m\_inode::i\_num, NULL, and ROOT\_INODE\_BLOCK.

Referenced by fs\_close(), and test\_file\_table().

**5.28.3.9 void free\_proc\_file ( proc\_file *pft*[NUM\_PROC\_FILES], file\_nr *fd* )**

Reset a process file.

**Parameters**

*pft* process file table

*fd* the file

References get\_proc\_file(), NULL, proc\_file::pf\_desc, proc\_file::pf\_f\_desc, and proc\_file::pf\_pos.

Referenced by do\_close\_pf(), and test\_open\_close().

**5.28.3.10 file\* get\_file ( file\_nr *fd* )**

Find a file in the global file table.

**Parameters**

*fd* file descriptor

**Returns**

pointer to the found file

References gft.

Referenced by contains\_file(), do\_read(), do\_write(), free\_file(), fs\_close(), fs\_truncate(), get\_file\_info(), inc\_count(), insert\_proc\_file(), sys\_seek(), sys\_unlink(), test\_rw\_qualitative(), and test\_rw\_quantitative().



**5.28.3.11** `proc_file* get_proc_file ( proc_file pft[NUM_PROC_FILES], file_nr fd )`

Find a file in the process file table.

**Parameters**

*pft* process file table

*fd* file descriptor

**Returns**

pointer to the file found

Referenced by `do_close_pf()`, `free_proc_file()`, `lseek()`, `sys_read()`, `sys_seek()`, `sys_write()`, and `test_PM()`.

**5.28.3.12** `void inc_count ( file_nr fd )`

Increment the reference counter of a file.

**Parameters**

*fd* file descriptor

References `file::f_count`, and `get_file()`.

Referenced by `insert_proc_file()`.

**5.28.3.13** `void init_file_table ( )`

Initialize the global file table with NULL elements.

References `file::f_desc`, and `gft`.

Referenced by `create_fs()`, `load_fs()`, and `test_file_table()`.

**5.28.3.14** `void init_proc_file_table ( proc_file proc_filp[] )`**5.28.3.15** `file_nr inode2desc ( m_inode * inode )`

Find the descriptor of a file via its inode (number).

**Parameters**

*inode* the file's inode

**Returns**

found file descriptor

References `file::f_desc`, `file::f_inode`, `gft`, and `m_inode::i_num`.

Referenced by `test_file_table()`.

**5.28.3.16 file\_nr insert\_file ( m\_inode \* inode, char \* name, uint8 mode )**

Insert a new file to the global file table.

**Parameters**

*inode* corresponding inode  
*name* filename  
*mode* DATA\_FILE | DIRECTORY

**Returns**

assigned file descriptor

References alloc\_file(), file::f\_desc, file::f\_inode, file::f\_mode, file::f\_name, free, fs\_dprintf, name2desc(), NOT\_FOUND, NULL, and strdup.

Referenced by fs\_open(), and test\_file\_table().

**5.28.3.17 file\_nr insert\_proc\_file ( proc\_file pft[NUM\_PROC\_FILES], file\_nr glo\_fd )**

Insert a new file to a given process file table.

**Parameters**

*pft* process file table  
*glo\_fd* global (for all processes) file descriptor

**Returns**

assigned file descriptor

References alloc\_proc\_file(), fs\_dprintf, get\_file(), inc\_count(), NULL, proc\_file::pf\_desc, proc\_file::pf\_f\_desc, and proc\_file::pf\_pos.

Referenced by sys\_open(), sys\_stat(), test\_close(), test\_open\_close(), and test\_PM().

**5.28.3.18 size\_t lseek ( proc\_file pft[NUM\_PROC\_FILES], file\_nr fd, sint32 offset, uint32 origin )**

Move file position to current file position + offset.

**Parameters**

*pft* process file table  
*fd* file descriptor  
*offset* offset  
*origin* original position

**Returns**

new position or -1 if failed

### 5.28.3.19 file\_nr name2desc ( char \* name )

Find the descriptor of a file with the file name.

#### Parameters

*name* file name

#### Returns

found file descriptor or NOT\_FOUND

References file::f\_desc, fs\_dprintf, gft, NULL, and strcmp.

Referenced by fs\_open(), insert\_file(), test\_file\_table(), test\_rw\_qualitative(), and test\_rw\_quantitative().

## 5.28.4 Variable Documentation

### 5.28.4.1 file gft[NUM\_FILES]

Referenced by alloc\_file(), dump\_files(), fs\_shutdown(), get\_file(), init\_file\_table(), inode2desc(), and name2desc().

## 5.29 src/kernel/fs/fs\_inode\_table.c File Reference

The inode table.

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/string.h"
#include "../include/stdlib.h"
#include "../include/debug.h"
#include "../include/assert.h"
#include "fs_const.h"
#include "fs_types.h"
#include "fs_buf.h"
#include "fs_block_dev.h"
#include "fs_inode_table.h"
```

### Functions

- void [init\\_inode\\_table](#) ()  
*Initialize all inodes with i\_num = NIL\_INODE = -1 -> sign that inode is unused.*
- void [create\\_root](#) ()  
*Create a new root inode.*

- void `load_root ()`  
*Load root inode from HD to inode table.*
- void `write_root ()`  
*Write root to HD.*
- `m_inode *` `get_inode (inode_nr i_num)`  
*Returns a pointer to an inode with inode\_nr = i\_num.*
- void `read_dinode (d_inode *inode, block_nr inode_blk)`  
*Read a inode from HD into a disk inode.*
- void `read_minode (m_inode *inode, block_nr inode_blk)`  
*Read a inode from HD into a memory inode.*
- void `write_inode (m_inode *inode)`  
*Write a memory inode to disk.*
- void `write_inodes ()`  
*Write all inodes from inode table to HD.*
- void `cpy_minode_to_dinode (d_inode *di, m_inode *mi)`  
*Copy common content of a memory inode to a disk inode.*
- void `cpy_dinode_to_minode (m_inode *mi, d_inode *di)`  
*Copy common content of a disk inode to a memory inode.*
- `m_inode *` `new_minode (block_nr adr, int mode, bool to_inode_table)`  
*Create a new memory inode.*
- void `free_inode (inode_nr i_num)`  
*Sets an inode as unused.*
- `m_inode *` `alloc_inode (void)`  
*Returns an unused inode.*
- void `dump_inode (m_inode *mi)`  
*Print out a inode's attributes for debug purposes.*
- void `dump_dinode (d_inode *di)`  
*Print out a disk inode's attributes for debug purposes.*
- void `dump_inodes ()`  
*Print out inode table for debug purposes.*

## 5.29.1 Detailed Description

The inode table. This table contains the currently opened inodes. This table is only existent in memory.

### Author

Vincenz Doelle

### LastChangedBy:

dtraytel

### Version

### Rev:

12

## 5.29.2 Function Documentation

### 5.29.2.1 `m_inode* alloc_inode ( void )`

Returns an unused inode.

#### Returns

Pointer to the inode

References `m_inode::i_num`, `inode_table`, and `NIL_INODE`.

Referenced by `fs_open()`, `new_minode()`, and `test_inode_table()`.

### 5.29.2.2 `void cpy_dinode_to_minode ( m_inode * mi, d_inode * di )`

Copy common content of a disk inode to a memory inode.

#### Parameters

*mi* memory inode

*di* disk inode

References `d_inode::i_create_ts`, `m_inode::i_create_ts`, `d_inode::i_direct_pointer`, `m_inode::i_direct_pointer`, `d_inode::i_double_indirect_pointer`, `m_inode::i_double_indirect_pointer`, `d_inode::i_mode`, `m_inode::i_mode`, `d_inode::i_modify_ts`, `m_inode::i_modify_ts`, `d_inode::i_single_indirect_pointer`, `m_inode::i_single_indirect_pointer`, `d_inode::i_size`, and `m_inode::i_size`.

Referenced by `read_minode()`.

### 5.29.2.3 `void cpy_minode_to_dinode ( d_inode * di, m_inode * mi )`

Copy common content of a memory inode to a disk inode.

**Parameters***di* disk inode*mi* memory inode

References `m_inode::i_create_ts`, `d_inode::i_create_ts`, `m_inode::i_direct_pointer`, `d_inode::i_direct_pointer`, `m_inode::i_double_indirect_pointer`, `d_inode::i_double_indirect_pointer`, `m_inode::i_mode`, `d_inode::i_mode`, `m_inode::i_modify_ts`, `d_inode::i_modify_ts`, `m_inode::i_single_indirect_pointer`, `d_inode::i_single_indirect_pointer`, `m_inode::i_size`, and `d_inode::i_size`.

Referenced by `write_inode()`.

**5.29.2.4 void create\_root ( )**

Create a new root inode.

References `ASSERT`, `DIRECTORY`, `FALSE`, `m_inode::i_num`, `inode_table`, `memcpy`, `new_minode()`, `NIL_INODE`, `NULL`, `root`, `ROOT_INODE`, and `ROOT_INODE_BLOCK`.

Referenced by `create_fs()`.

**5.29.2.5 void dump\_dinode ( d\_inode \* di )**

Print out a disk inode's attributes for debug purposes.

**Parameters***di* inode to be printed

References `fs_dprintf`, `d_inode::i_direct_pointer`, `d_inode::i_double_indirect_pointer`, `d_inode::i_mode`, and `d_inode::i_single_indirect_pointer`.

**5.29.2.6 void dump\_inode ( m\_inode \* mi )**

Print out a inode's attributes for debug purposes.

**Parameters***mi* inode to be printed

References `fs_dprintf`, `m_inode::i_adr`, `m_inode::i_direct_pointer`, `m_inode::i_double_indirect_pointer`, `m_inode::i_mode`, `m_inode::i_num`, and `m_inode::i_single_indirect_pointer`.

Referenced by `dump_inodes()`, and `test_sync()`.

**5.29.2.7 void dump\_inodes ( )**

Print out inode table for debug purposes.

References `dump_inode()`, and `inode_table`.

Referenced by `test_file_table()`, and `test_inode_table()`.

### 5.29.2.8 void free\_inode ( inode\_nr i\_num )

Sets an inode as unused.

#### Parameters

*i\_num* The inode number

References get\_inode(), and m\_inode::i\_num.

Referenced by free\_file(), and test\_inode\_table().

### 5.29.2.9 m\_inode\* get\_inode ( inode\_nr i\_num )

Returns a pointer to an inode with inode\_nr = i\_num.

#### Parameters

*i\_num* The inode number of the inode

#### Returns

Pointer to the inode.

References inode\_table.

Referenced by free\_inode().

### 5.29.2.10 void init\_inode\_table ( )

Initialize all inodes with i\_num = NIL\_INODE = -1 -> sign that inode is unused.

References fs\_dprintf, m\_inode::i\_num, and inode\_table.

Referenced by create\_fs(), and load\_fs().

### 5.29.2.11 void load\_root ( )

Load root inode from HD to inode table.

References ASSERT, fs\_dprintf, m\_inode::i\_adr, m\_inode::i\_num, inode\_table, NULL, read\_minode(), root, ROOT\_INODE, and ROOT\_INODE\_BLOCK.

Referenced by load\_fs().

### 5.29.2.12 m\_inode\* new\_minode ( block\_nr adr, int mode, bool to\_inode\_table )

Create a new memory inode.

#### Parameters

*adr* block address

*mode* DATA\_FILE | DIRECTORY (

**See also**

[fs\\_const.h](#))

**Parameters**

*to\_inode\_table* should the new inode be inserted into the inode table?

**Returns**

pointer to new inode

References `alloc_inode()`, `bzero`, `m_inode::i_adr`, `m_inode::i_create_ts`, `m_inode::i_direct_pointer`, `m_inode::i_double_indirect_pointer`, `m_inode::i_mode`, `m_inode::i_modify_ts`, `m_inode::i_num`, `m_inode::i_single_indirect_pointer`, `m_inode::i_size`, `mallocn`, and `NULL`.

Referenced by `create_root()`, `fs_create_delete()`, and `test_file_table()`.

**5.29.2.13 void read\_dinode ( d\_inode \* inode, block\_nr inode\_blk )**

Read a inode from HD into a disk inode.

**Parameters**

*inode* destination

*inode\_blk* source block on HD

References `rd_block()`.

Referenced by `read_minode()`.

**5.29.2.14 void read\_minode ( m\_inode \* inode, block\_nr inode\_blk )**

Read a inode from HD into a memory inode.

**Parameters**

*inode* destination

*inode\_blk* source block on HD

References `bzero`, `cpy_dinode_to_minode()`, `d_inode_cache`, `m_inode::i_adr`, and `read_dinode()`.

Referenced by `delete_file_from_dir()`, `fs_open()`, `insert_file_into_dir()`, `load_root()`, and `rfsearch()`.

**5.29.2.15 void write\_inode ( m\_inode \* inode )**

Write a memory inode to disk.

**Parameters**

*inode* pointer to the memory inode which should be written

**Returns**

boolean status of operation



References `cpy_minode_to_dinode()`, `d_inode_cache`, `m_inode::i_adr`, and `wrt_block()`.

Referenced by `delete_file_from_dir()`, `fs_close()`, `fs_create_delete()`, `fs_truncate()`, `fs_write()`, `get_data_block()`, `insert_file_into_dir()`, `write_inodes()`, and `write_root()`.

#### 5.29.2.16 void write\_inodes ( )

Write all inodes from inode table to HD.

References `inode_table`, and `write_inode()`.

#### 5.29.2.17 void write\_root ( )

Write root to HD.

References `ASSERT`, `mallocn`, `NULL`, `root`, and `write_inode()`.

Referenced by `fs_shutdown()`.

## 5.30 src/kernel/fs/fs\_inode\_table.h File Reference

The inode table.

### Defines

- #define `NIL_INODE` -1

### Functions

- void `init_inode_table` ()  
*Initialize all inodes with `i_num = NIL_INODE = -1` -> sign that inode is unused.*
- void `dump_inode` (`m_inode *i`)  
*Print out a inode's attributes for debug purposes.*
- void `dump_inodes` ()  
*Print out inode table for debug purposes.*
- void `load_root` ()  
*Load root inode from HD to inode table.*
- void `write_root` ()  
*Write root to HD.*
- void `create_root` ()  
*Create a new root inode.*
- `m_inode *` `get_inode` (`inode_nr i_num`)  
*Returns a pointer to an inode with `inode_nr = i_num`.*

- void `read_dinode` (`d_inode *inode`, `block_nr inode_blk`)  
*Read a inode from HD into a disk inode.*
- void `read_minode` (`m_inode *inode`, `block_nr inode_blk`)  
*Read a inode from HD into a memory inode.*
- void `write_inode` (`m_inode *inode`)  
*Write a memory inode to disk.*
- void `write_inodes` ()  
*Write all inodes from inode table to HD.*
- void `cpy_minode_to_dinode` (`d_inode *di`, `m_inode *mi`)  
*Copy common content of a memory inode to a disk inode.*
- void `cpy_dinode_to_minode` (`m_inode *mi`, `d_inode *di`)  
*Copy common content of a disk inode to a memory inode.*
- `m_inode * new_minode` (`block_nr adr`, `int mode`, `bool to_inode_table`)  
*Create a new memory inode.*
- void `free_inode` (`inode_nr i_num`)  
*Sets an inode as unused.*
- `m_inode * alloc_inode` ()  
*Returns an unused inode.*

## Variables

- `m_inode inode_table` [`NUM_INODES`]  
*The (memory-) inode table.*
- `m_inode * root`  
*The root inode.*

### 5.30.1 Detailed Description

The inode table. This table contains the currently opened inodes. This table is only existent in memory.

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

**Rev:**

12

**5.30.2 Define Documentation****5.30.2.1 #define NIL\_INODE -1**

Referenced by `alloc_inode()`, and `create_root()`.

**5.30.3 Function Documentation****5.30.3.1 m\_inode\* alloc\_inode ( void )**

Returns an unused inode.

**Returns**

Pointer to the inode

References `m_inode::i_num`, `inode_table`, and `NIL_INODE`.

Referenced by `fs_open()`, `new_minode()`, and `test_inode_table()`.

**5.30.3.2 void cpy\_dinode\_to\_minode ( m\_inode \* mi, d\_inode \* di )**

Copy common content of a disk inode to a memory inode.

**Parameters**

*mi* memory inode

*di* disk inode

References `d_inode::i_create_ts`, `m_inode::i_create_ts`, `d_inode::i_direct_pointer`, `m_inode::i_direct_pointer`, `d_inode::i_double_indirect_pointer`, `m_inode::i_double_indirect_pointer`, `d_inode::i_mode`, `m_inode::i_mode`, `d_inode::i_modify_ts`, `m_inode::i_modify_ts`, `d_inode::i_single_indirect_pointer`, `m_inode::i_single_indirect_pointer`, `d_inode::i_size`, and `m_inode::i_size`.

Referenced by `read_minode()`.

**5.30.3.3 void cpy\_minode\_to\_dinode ( d\_inode \* di, m\_inode \* mi )**

Copy common content of a memory inode to a disk inode.

**Parameters**

*di* disk inode

*mi* memory inode

References `m_inode::i_create_ts`, `d_inode::i_create_ts`, `m_inode::i_direct_pointer`, `d_inode::i_direct_pointer`, `m_inode::i_double_indirect_pointer`, `d_inode::i_double_indirect_pointer`, `m_inode::i_mode`, `d_inode::i_mode`, `m_inode::i_modify_ts`, `d_inode::i_modify_ts`, `m_inode::i_single_indirect_pointer`, `d_inode::i_single_indirect_pointer`, `m_inode::i_size`, and `d_inode::i_size`.

Referenced by `write_inode()`.

#### 5.30.3.4 void create\_root ( )

Create a new root inode.

References ASSERT, DIRECTORY, FALSE, m\_inode::i\_num, inode\_table, memcpy, new\_minode(), NIL\_INODE, NULL, root, ROOT\_INODE, and ROOT\_INODE\_BLOCK.

Referenced by create\_fs().

#### 5.30.3.5 void dump\_inode ( m\_inode \* mi )

Print out a inode's attributes for debug purposes.

##### Parameters

*mi* inode to be printed

References fs\_dprintf, m\_inode::i\_adr, m\_inode::i\_direct\_pointer, m\_inode::i\_double\_indirect\_pointer, m\_inode::i\_mode, m\_inode::i\_num, and m\_inode::i\_single\_indirect\_pointer.

Referenced by dump\_inodes(), and test\_sync().

#### 5.30.3.6 void dump\_inodes ( )

Print out inode table for debug purposes.

References dump\_inode(), and inode\_table.

Referenced by test\_file\_table(), and test\_inode\_table().

#### 5.30.3.7 void free\_inode ( inode\_nr i\_num )

Sets an inode as unused.

##### Parameters

*i\_num* The inode number

References get\_inode(), and m\_inode::i\_num.

Referenced by free\_file(), and test\_inode\_table().

#### 5.30.3.8 m\_inode\* get\_inode ( inode\_nr i\_num )

Returns a pointer to an inode with inode\_nr = i\_num.

##### Parameters

*i\_num* The inode number of the inode

##### Returns

Pointer to the inode.

References inode\_table.

Referenced by free\_inode().

### 5.30.3.9 void init\_inode\_table ( )

Initialize all inodes with `i_num = NIL_INODE = -1` -> sign that inode is unused.

References `fs_dprintf`, `m_inode::i_num`, and `inode_table`.

Referenced by `create_fs()`, and `load_fs()`.

### 5.30.3.10 void load\_root ( )

Load root inode from HD to inode table.

References `ASSERT`, `fs_dprintf`, `m_inode::i_adr`, `m_inode::i_num`, `inode_table`, `NULL`, `read_minode()`, `root`, `ROOT_INODE`, and `ROOT_INODE_BLOCK`.

Referenced by `load_fs()`.

### 5.30.3.11 m\_inode\* new\_minode ( block\_nr adr, int mode, bool to\_inode\_table )

Create a new memory inode.

#### Parameters

*adr* block address

*mode* DATA\_FILE | DIRECTORY (

#### See also

[fs\\_const.h](#))

#### Parameters

*to\_inode\_table* should the new inode be inserted into the inode table?

#### Returns

pointer to new inode

References `alloc_inode()`, `bzero`, `m_inode::i_adr`, `m_inode::i_create_ts`, `m_inode::i_direct_pointer`, `m_inode::i_double_indirect_pointer`, `m_inode::i_mode`, `m_inode::i_modify_ts`, `m_inode::i_num`, `m_inode::i_single_indirect_pointer`, `m_inode::i_size`, `mallocn`, and `NULL`.

Referenced by `create_root()`, `fs_create_delete()`, and `test_file_table()`.

### 5.30.3.12 void read\_dinode ( d\_inode \* inode, block\_nr inode\_blk )

Read a inode from HD into a disk inode.

#### Parameters

*inode* destination

*inode\_blk* source block on HD

**5.30.3.13 void read\_inode ( m\_inode \* inode, block\_nr inode\_blk )**

Read a inode from HD into a memory inode.

**Parameters**

*inode* destination

*inode\_blk* source block on HD

**5.30.3.14 void write\_inode ( m\_inode \* inode )**

Write a memory inode to disk.

**Parameters**

*inode* pointer to the memory inode which should be written

**Returns**

boolean status of operation

**5.30.3.15 void write\_inodes ( )**

Write all inodes from inode table to HD.

References inode\_table, and write\_inode().

**5.30.3.16 void write\_root ( )**

Write root to HD.

References ASSERT, mallocn, NULL, root, and write\_inode().

Referenced by fs\_shutdown().

**5.30.4 Variable Documentation****5.30.4.1 m\_inode inode\_table[NUM\_INODES]**

The (memory-) inode table.

Referenced by alloc\_inode(), create\_root(), dump\_inodes(), get\_inode(), init\_inode\_table(), load\_root(), and write\_inodes().

**5.30.4.2 m\_inode\* root**

The root inode.

extern deklaration of root inode

Referenced by create\_root(), delete\_file\_from\_dir(), fs\_open(), init\_super\_block(), insert\_file\_into\_dir(), load\_root(), rfsearch(), search\_file(), test\_file\_table(), test\_sync(), and write\_root().

## 5.31 src/kernel/fs/fs\_io\_functions.h File Reference

Functions definitions concerning read, write, open, close, create, delete.

### Defines

- #define `CREATE` 1
- #define `DELETE` 2

### Functions

- `size_t fs_read` (void \*buf, `m_inode` \*inode, `size_t` num\_bytes, `uint32` pos, `bool` allow\_enlargement)  
*Read content from HD.*
- `size_t fs_write` (`m_inode` \*inode, void \*buf, `size_t` num\_bytes, `uint32` pos, `bool` allow\_enlargement)  
*Write content to HD.*
- `file_nr fs_open` (char \*abs\_path)  
*Opens a file by searching the inode's block number, loading the inode from HD, inserting the inode to the inode table and registering the file in the global/process file table.*
- `bool fs_close` (`file_nr` fd)  
*Closes a file by writing the inode to HD and freeing the file descriptor entry.*
- `bool fs_create` (char \*path, int data\_type)  
*Creates a file from absolute path by inserting the name into the containing directory, creating a new inode and writing it to HD.*
- `bool fs_delete` (char \*path)  
*Deletes a file by removing it from the containing directory.*
- `bool fs_create_delete` (char \*abs\_path, int mode, int data\_type)  
*Handles the creation and deletion process.*
- `bool fs_truncate` (char \*abs\_path, `uint32` size)  
*Set the size of file abs\_path to be exactly size, deleting or allocating blocks as necessary.*
- `size_t lseek` (`proc_file` pft[`NUM_PROC_FILES`], `file_nr` fd, `sint32` offset, `uint32` originf)  
*Move file position to current file position + offset.*
- void `read_dinode` (`d_inode` \*inode, `block_nr` inode\_blk)  
*Read a inode from HD into a disk inode.*
- void `read_minode` (`m_inode` \*inode, `block_nr` inode\_blk)  
*Read a inode from HD into a memory inode.*
- void `write_inode` (`m_inode` \*inode)  
*Write a memory inode to disk.*

### 5.31.1 Detailed Description

Functions definitions concerning read, write, open, close, create, delete.

**Author**

Vincenz Doelle

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

### 5.31.2 Define Documentation

#### 5.31.2.1 #define CREATE 1

Referenced by fs\_create(), and fs\_create\_delete().

#### 5.31.2.2 #define DELETE 2

Referenced by fs\_create\_delete(), and fs\_delete().

### 5.31.3 Function Documentation

#### 5.31.3.1 bool fs\_close ( file\_nr *fd* )

Closes a file by writing the inode to HD and freeing the file descriptor entry.

**Parameters**

*fd* process file descriptor.

**Returns**

operation status

References dump\_files(), file::f\_desc, file::f\_inode, file::f\_name, free\_file(), fs\_dprintf, get\_file(), NIL\_FILE, NULL, and write\_inode().

Referenced by do\_close(), fs\_shutdown(), fs\_truncate(), and test\_open\_close().

#### 5.31.3.2 bool fs\_create ( char \* *abs\_path*, int *data\_type* )

Creates a file from absolute path by inserting the name into the containing directory, creating a new inode and writing it to HD.



**Parameters**

*abs\_path* absolute file path  
*data\_type* DATA\_FILE | DIRECTORY (

**See also**

[fs\\_const.h](#))

**Returns**

result status of the create operation

**5.31.3.3 bool fs\_create\_delete ( char \* *abs\_path*, int *mode*, int *data\_type* )**

Handles the creation and deletion process.

**Parameters**

*abs\_path* absolute file path  
*mode* CREATE | DELETE (

**See also**

[fs\\_const.h](#))

**Parameters**

*data\_type* DATA\_FILE | DIRECTORY (

**See also**

[fs\\_const.h](#))

**Returns**

result status of the create/delete operation

References CREATE, DELETE, delete\_file\_from\_dir(), FALSE, free, fs\_dprintf, fs\_truncate(), get\_filename(), get\_path(), inode, insert\_file\_into\_dir(), new\_minode(), NOT\_FOUND, NOT\_POSSIBLE, NULL, printf, RED, search\_file(), strcmp, and write\_inode().

Referenced by fs\_create(), and fs\_delete().

**5.31.3.4 bool fs\_delete ( char \* *abs\_path* )**

Deletes a file by removing it from the containing directory.

**Parameters**

*abs\_path* absolute file path

**Returns**

result status of the delete operation

### 5.31.3.5 file\_nr fs\_open ( char \* *abs\_path* )

Opens a file by searching the inode's block number, loading the inode from HD, inserting the inode to the inode table and registering the file in the global/process file table.

#### Parameters

*abs\_path* absolute file path

#### Returns

process file descriptor

References alloc\_inode(), fs\_dprintf, m\_inode::i\_mode, inode, insert\_file(), name2desc(), NOT\_FOUND, NULL, read\_minode(), root, search\_file(), and strcmp.

Referenced by do\_open(), fs\_truncate(), test\_open\_close(), test\_rw\_qualitative(), and test\_rw\_quantitative().

### 5.31.3.6 size\_t fs\_read ( void \* *buf*, m\_inode \* *inode*, size\_t *num\_bytes*, uint32 *pos*, bool *allow\_enlargement* )

Read content from HD.

#### Parameters

*buf* destination buffer

*inode* source inode

*num\_bytes* number of bytes desired to read

*pos* start position

*allow\_enlargement* enlarge the file if EOF is reached?

#### Returns

number of bytes read

References BLOCK\_SIZE, block\_buffer::cache, cache\_block(), fs\_dprintf, fs\_read(), get\_data\_block(), m\_inode::i\_size, memcpy, NOT\_FOUND, and read\_cache.

Referenced by delete\_file\_from\_dir(), do\_read(), fs\_read(), insert\_file\_into\_dir(), rfsearch(), test\_rw\_qualitative(), and test\_rw\_quantitative().

### 5.31.3.7 bool fs\_truncate ( char \* *abs\_path*, uint32 *size* )

Set the size of file *abs\_path* to be exactly *size*, deleting or allocating blocks as necessary.

References addr\_cache, BLOCK\_SIZE, file::f\_inode, FALSE, fs\_close(), fs\_dprintf, fs\_open(), get\_data\_block(), get\_file(), m\_inode::i\_direct\_pointer, m\_inode::i\_double\_indirect\_pointer, m\_inode::i\_single\_indirect\_pointer, m\_inode::i\_size, i\_size, inode, mark\_block(), MAX, NULL, rd\_block(), TRUE, write\_inode(), and wrt\_block().

Referenced by fs\_create\_delete().

**5.31.3.8** `size_t fs_write ( m_inode * inode, void * buf, size_t num_bytes, uint32 pos, bool allow_scaling )`

Write content to HD.

**Parameters**

*inode* destination inode  
*buf* source buffer  
*num\_bytes* number of bytes desired to write  
*pos* start position  
*allow\_scaling* enlarge the file if EOF is reached?

**Returns**

number of bytes written

References BLOCK\_SIZE, block\_buffer::cache, cache\_block(), free, fs\_dprintf, fs\_write(), get\_data\_block(), m\_inode::i\_modify\_ts, m\_inode::i\_size, mallocn, memcpy, NOT\_FOUND, read\_cache, time2str, write\_inode(), and wrt\_cache().

Referenced by do\_write(), fs\_write(), test\_rw\_qualitative(), and test\_rw\_quantitative().

**5.31.3.9** `size_t lseek ( proc_file pft[NUM_PROC_FILES], file_nr fd, sint32 offset, uint32 origin )`

Move file position to current file position + offset.

**Parameters**

*pft* process file table  
*fd* file descriptor  
*offset* offset  
*origin* original position

**Returns**

new position or -1 if failed

References get\_proc\_file(), NULL, and proc\_file::pf\_pos.

Referenced by do\_lseek().

**5.31.3.10** `void read_dinode ( d_inode * inode, block_nr inode_blk )`

Read a inode from HD into a disk inode.

**Parameters**

*inode* destination  
*inode\_blk* source block on HD

References rd\_block().

Referenced by read\_minode().

### 5.31.3.11 void read\_minode ( m\_inode \* inode, block\_nr inode\_blk )

Read a inode from HD into a memory inode.

#### Parameters

*inode* destination

*inode\_blk* source block on HD

References bzero, cpy\_dinode\_to\_minode(), d\_inode\_cache, m\_inode::i\_adr, and read\_dinode().

Referenced by delete\_file\_from\_dir(), fs\_open(), insert\_file\_into\_dir(), load\_root(), and rfsearch().

### 5.31.3.12 void write\_inode ( m\_inode \* inode )

Write a memory inode to disk.

#### Parameters

*inode* pointer to the memory inode which should be written

#### Returns

boolean status of operation

References cpy\_minode\_to\_dinode(), d\_inode\_cache, m\_inode::i\_adr, and wrt\_block().

Referenced by delete\_file\_from\_dir(), fs\_close(), fs\_create\_delete(), fs\_truncate(), fs\_write(), get\_data\_block(), insert\_file\_into\_dir(), write\_inodes(), and write\_root().

## 5.32 src/kernel/fs/fs\_main.c File Reference

This is the central program dealing as an interface between FS and PM.

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/debug.h"
#include "../include/stdlib.h"
#include "fs_const.h"
#include "fs_types.h"
#include "fs_file_table.h"
#include "fs_inode_table.h"
#include "fs_io_functions.h"
#include "fs_super.h"
#include "fs_bmap.h"
#include "fs_main.h"
```

## Functions

- void `panic` (char \*msg)  
*Kernel panic function.*
- void `fs_init` ()  
*Initializes the file system.*
- void `fs_shutdown` ()  
*Shuts the file system down and writes all important information to HD.*
- bool `load_fs` ()  
*Loads the file system from HD.*
- bool `create_fs` ()  
*Creates a new file system.*
- `size_t do_read` (file\_nr fd, void \*buf, size\_t count, uint32 pos)  
*Interface functions to handle the system calls.*
- `size_t do_write` (file\_nr fd, void \*buf, size\_t count, uint32 pos)
- bool `do_create` (char \*abs\_path, uint8 mode)
- bool `do_mkdir` (char \*abs\_path)
- bool `do_mkfile` (char \*abs\_path)
- int `do_remove` (char \*abs\_path)
- file\_nr `do_open` (char \*abs\_path)
- int `do_close` (file\_nr fd)
- int `do_close_pf` (proc\_file pft[NUM\_PROC\_FILES], file\_nr pfd)
- `size_t do_lseek` (proc\_file pft[NUM\_PROC\_FILES], file\_nr fd, sint32 offset, uint32 origin)
- block\_nr `search_file` (char \*path)  
*Prepare recursive search according to strtok() definition.*
- bool `do_file_exists` (char \*path)
- void `dump_consts` ()  
*Prints out all important constants concerning the file system.*

### 5.32.1 Detailed Description

This is the central program dealing as an interface between FS and PM.

**Author**

Vincenz Doelle

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

## 5.32.2 Function Documentation

### 5.32.2.1 `bool create_fs ( )`

Creates a new file system.

References `create_root()`, `dump_super()`, `init_bmap()`, `init_file_table()`, `init_inode_table()`, `init_super_block()`, `printf`, and `write_super_block()`.

Referenced by `fs_init()`, and `new_fs()`.

### 5.32.2.2 `int do_close ( file_nr fd )`

References `fs_close()`.

Referenced by `__ls()`, `chips_unlink()`, `do_close_pf()`, and `sys_unlink()`.

### 5.32.2.3 `int do_close_pf ( proc_file pft[NUM_PROC_FILES], file_nr pfd )`

References `do_close()`, `free_proc_file()`, `get_proc_file()`, and `proc_file::pf_f_desc`.

Referenced by `sys_close()`, `sys_stat()`, `test_close()`, and `test_PM()`.

### 5.32.2.4 `bool do_create ( char * abs_path, uint8 mode )`

References `fs_create()`.

Referenced by `do_mkfile()`, `pm_register_device()`, `test_ls()`, and `test_PM()`.

### 5.32.2.5 `bool do_file_exists ( char * path )`

References `NOT_FOUND`, and `search_file()`.

Referenced by `pm_init()`, and `pm_register_device()`.

### 5.32.2.6 `size_t do_lseek ( proc_file pft[NUM_PROC_FILES], file_nr fd, sint32 offset, uint32 origin )`

References `lseek()`.

### 5.32.2.7 `bool do_mkdir ( char * abs_path )`

References `DIRECTORY`, and `fs_create()`.

Referenced by `pm_init()`, `sys_open()`, `test_close()`, `test_error()`, and `test_PM()`.

### 5.32.2.8 `bool do_mkfile ( char * abs_path )`

References `DATA_FILE`, and `do_create()`.

Referenced by `sys_open()`, and `test_close()`.

**5.32.2.9 file\_nr do\_open ( char \* *abs\_path* )**

References fs\_open().

Referenced by \_\_ls(), chips\_unlink(), sys\_open(), sys\_stat(), sys\_unlink(), test\_close(), test\_ls(), and test\_PM().

**5.32.2.10 size\_t do\_read ( file\_nr *fd*, void \* *buf*, size\_t *count*, uint32 *pos* )**

Interface functions to handle the system calls.

References DIR\_ENTRIES\_PER\_BLOCK, DIRECTORY, FALSE, fs\_dprintf, fs\_read(), get\_file(), get\_file\_info(), file\_info::mode, and file\_info::size.

Referenced by \_\_ls(), sys\_read(), and test\_PM().

**5.32.2.11 int do\_remove ( char \* *abs\_path* )**

References fs\_delete().

Referenced by sys\_unlink().

**5.32.2.12 size\_t do\_write ( file\_nr *fd*, void \* *buf*, size\_t *count*, uint32 *pos* )**

References fs\_write(), get\_file(), and TRUE.

Referenced by sys\_write().

**5.32.2.13 void dump\_consts ( )**

Prints out all important constants concerning the file system.

For debug purposes only.

References ADDRS\_PER\_BLOCK, BYTES\_DIRECT, BYTES\_DOUBLE\_INDIRECT, BYTES\_SINGLE\_INDIRECT, DIR\_ENTRIES\_PER\_BLOCK, DISK\_INODE\_SIZE, INODES\_PER\_BLOCK, MEM\_INODE\_SIZE, NUM\_FILES, NUM\_INODES, NUM\_PROC\_FILES, printf, ROOT\_INODE\_BLOCK, and SUPER\_SIZE.

**5.32.2.14 void fs\_init ( )**

Initializes the file system.

References bmap, create\_fs(), dprint\_separator, dprintf, free, load\_fs(), panic, and printf.

Referenced by main(), and shell\_cmd\_sync().

**5.32.2.15 void fs\_shutdown ( )**

Shuts the file system down and writes all important information to HD.

References bmap, free, fs\_close(), gft, printf, write\_bmap(), write\_root(), and write\_super\_block().

Referenced by shell\_cmd\_sync(), and test\_sync().

### 5.32.2.16 `bool load_fs ( )`

Loads the file system from HD.

References `init_file_table()`, `init_inode_table()`, `load_bmap()`, `load_root()`, `load_super_block()`, `MAGIC_NUMBER`, and `printf`.

Referenced by `fs_init()`, and `test_sync()`.

### 5.32.2.17 `void panic ( char * msg )`

Kernel panic function.

Displays an error message and enters an infinite loop thus effectively halting the kernel. This should only be called when a non-recoverable ("fatal") error occurs inside the kernel.

#### Parameters

*msg* Message to display.

### 5.32.2.18 `block_nr search_file ( char * path )`

Prepare recursive search according to `strtok()` definition.

#### Parameters

*path* absolute path to file

#### Returns

block number of the file's inode (if found)

References `free`, `fs_dprintf`, `m_inode::i_adr`, `rfsearch()`, `root`, `strcmp`, `strdup`, and `strsep`.

Referenced by `do_file_exists()`, `fs_create_delete()`, and `fs_open()`.

## 5.33 `src/kernel/fs/fs_main.h` File Reference

Basic definitions of all functions concerning work on files.

### Functions

- void `fs_init ()`  
*Initializes the file system.*
- void `fs_shutdown ()`  
*Shuts the file system down and writes all important information to HD.*
- bool `load_fs ()`  
*Loads the file system from HD.*



- `bool create_fs ()`  
*Creates a new file system.*
  
- `bool do_create (char *abs_path, uint8 mode)`
- `bool do_mkdir (char *abs_path)`
- `bool do_mkfile (char *abs_path)`
- `int do_remove (char *abs_path)`
- `file_nr do_open (char *abs_path)`
- `int do_close (file_nr fd)`
- `int do_close_pf (proc_file pft[NUM_PROC_FILES], file_nr pfd)`
- `size_t do_read (file_nr fd, void *buf, size_t count, uint32 pos)`  
*Interface functions to handle the system calls.*
  
- `size_t do_write (file_nr fd, void *buf, size_t count, uint32 pos)`
- `size_t do_lseek (proc_file pft[NUM_PROC_FILES], file_nr fd, sint32 offset, uint32 origin)`
- `file_info_t * get_file_info (file_nr fd, file_info_t *info)`  
*Get file further information.*
  
- `bool do_file_exists (char *path)`
- `void dump_consts ()`  
*Prints out all important constants concerning the file system.*

### 5.33.1 Detailed Description

Basic definitions of all functions concerning work on files.

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.33.2 Function Documentation

#### 5.33.2.1 `bool create_fs ( )`

Creates a new file system.

#### 5.33.2.2 `int do_close ( file_nr fd )`

References `fs_close()`.

Referenced by `__ls()`, `chips_unlink()`, `do_close_pf()`, and `sys_unlink()`.

**5.33.2.3 int do\_close\_pf ( proc\_file pft[*NUM\_PROC\_FILES*], file\_nr pfd )**

References do\_close(), free\_proc\_file(), get\_proc\_file(), and proc\_file::pf\_f\_desc.

Referenced by sys\_close(), sys\_stat(), test\_close(), and test\_PM().

**5.33.2.4 bool do\_create ( char \* abs\_path, uint8 mode )**

References fs\_create().

Referenced by do\_mkfile(), pm\_register\_device(), test\_ls(), and test\_PM().

**5.33.2.5 bool do\_file\_exists ( char \* path )**

References NOT\_FOUND, and search\_file().

Referenced by pm\_init(), and pm\_register\_device().

**5.33.2.6 size\_t do\_lseek ( proc\_file pft[*NUM\_PROC\_FILES*], file\_nr fd, sint32 offset, uint32 origin )**

References lseek().

**5.33.2.7 bool do\_mkdir ( char \* abs\_path )**

References DIRECTORY, and fs\_create().

Referenced by pm\_init(), sys\_open(), test\_close(), test\_error(), and test\_PM().

**5.33.2.8 bool do\_mkfile ( char \* abs\_path )**

References DATA\_FILE, and do\_create().

Referenced by sys\_open(), and test\_close().

**5.33.2.9 file\_nr do\_open ( char \* abs\_path )****5.33.2.10 size\_t do\_read ( file\_nr fd, void \* buf, size\_t count, uint32 pos )**

Interface functions to handle the system calls.

References DIR\_ENTRIES\_PER\_BLOCK, DIRECTORY, FALSE, fs\_dprintf, fs\_read(), get\_file(), get\_file\_info(), file\_info::mode, and file\_info::size.

Referenced by \_\_ls(), sys\_read(), and test\_PM().

**5.33.2.11 int do\_remove ( char \* abs\_path )**

References fs\_delete().

Referenced by sys\_unlink().

**5.33.2.12** `size_t do_write ( file_nr fd, void * buf, size_t count, uint32 pos )`

References `fs_write()`, `get_file()`, and `TRUE`.

Referenced by `sys_write()`.

**5.33.2.13** `void dump_consts ( )`

Prints out all important constants concerning the file system.

For debug purposes only.

**5.33.2.14** `void fs_init ( )`

Initializes the file system.

**5.33.2.15** `void fs_shutdown ( )`

Shuts the file system down and writes all important information to HD.

**5.33.2.16** `file_info_t* get_file_info ( file_nr fd, file_info_t * info )`

Get file further information.

**Parameters**

*fd* file descriptor

**Returns**

file information

References `file_info::create_ts`, `file::f_count`, `file::f_inode`, `file::f_mode`, `file::f_name`, `get_file()`, `i_create_ts`, `i_modify_ts`, `i_size`, `file_info::mode`, `file_info::modify_ts`, `file_info::name`, `file_info::num_links`, `file_info::size`, and `strncpy`.

Referenced by `chips_unlink()`, `do_read()`, `sys_stat()`, `sys_unlink()`, and `sys_write()`.

**5.33.2.17** `bool load_fs ( )`

Loads the file system from HD.

References `init_file_table()`, `init_inode_table()`, `load_bmap()`, `load_root()`, `load_super_block()`, `MAGIC_NUMBER`, and `printf`.

Referenced by `fs_init()`, and `test_sync()`.

## 5.34 src/kernel/fs/fs\_open\_close.c File Reference

Basic definitions concerning syscalls "open" and "close".

```
#include "../include/const.h"
```

```
#include "../include/types.h"
#include "../include/stdlib.h"
#include "../include/string.h"
#include "../include/debug.h"
#include "fs_const.h"
#include "fs_types.h"
#include "fs_io_functions.h"
#include "fs_file_table.h"
#include "fs_inode_table.h"
#include "fs_buf.h"
#include "fs_block_dev.h"
#include "fs_dir.h"
```

## Functions

- [file\\_nr fs\\_open](#) (char \*abs\_path)  
*Opens a file by searching the inode's block number, loading the inode from HD, inserting the inode to the inode table and and registering the file in the global/process file table.*
- [bool fs\\_close](#) (file\_nr fd)  
*Closes a file by writing the inode to HD and freeing the file descriptor entry.*

### 5.34.1 Detailed Description

Basic definitions concerning syscalls "open" and "close".

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.34.2 Function Documentation

#### 5.34.2.1 bool fs\_close ( file\_nr fd )

Closes a file by writing the inode to HD and freeing the file descriptor entry.

**Parameters**

*fd* process file descriptor.

**Returns**

operation status

References `dump_files()`, `file::f_desc`, `file::f_inode`, `file::f_name`, `free_file()`, `fs_dprintf`, `get_file()`, `NIL_FILE`, `NULL`, and `write_inode()`.

Referenced by `do_close()`, `fs_shutdown()`, `fs_truncate()`, and `test_open_close()`.

**5.34.2.2 file\_nr fs\_open ( char \* abs\_path )**

Opens a file by searching the inode's block number, loading the inode from HD, inserting the inode to the inode table and registering the file in the global/process file table.

**Parameters**

*abs\_path* absolute file path

**Returns**

process file descriptor

References `alloc_inode()`, `fs_dprintf`, `m_inode::i_mode`, `inode`, `insert_file()`, `name2desc()`, `NOT_FOUND`, `NULL`, `read_minode()`, `root`, `search_file()`, and `strcmp`.

Referenced by `do_open()`, `fs_truncate()`, `test_open_close()`, `test_rw_qualitative()`, and `test_rw_quantitative()`.

**5.35 src/kernel/fs/fs\_read\_write.c File Reference**

Functions concerning syscalls "read" and "write".

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/string.h"
#include "../include/stdlib.h"
#include "../include/debug.h"
#include "fs_const.h"
#include "fs_types.h"
#include "fs_block_dev.h"
#include "fs_buf.h"
#include "fs_io_functions.h"
```

**Functions**

- `size_t fs_read` (void \*buf, `m_inode` \*inode, `size_t` num\_bytes, `uint32` pos, `bool` allow\_enlargement)

*Read content from HD.*

- `size_t fs_write (m_inode *inode, void *buf, size_t num_bytes, uint32 pos, bool allow_scaling)`

*Write content to HD.*

### 5.35.1 Detailed Description

Functions concerning syscalls "read" and "write".

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.35.2 Function Documentation

#### 5.35.2.1 `size_t fs_read ( void * buf, m_inode * inode, size_t num_bytes, uint32 pos, bool allow_enlargement )`

Read content from HD.

#### Parameters

*buf* destination buffer

*inode* source inode

*num\_bytes* number of bytes desired to read

*pos* start position

*allow\_enlargement* enlarge the file if EOF is reached?

#### Returns

number of bytes read

References BLOCK\_SIZE, block\_buffer::cache, cache\_block(), fs\_dprintf, fs\_read(), get\_data\_block(), m\_inode::i\_size, memcpy, NOT\_FOUND, and read\_cache.

Referenced by delete\_file\_from\_dir(), do\_read(), fs\_read(), insert\_file\_into\_dir(), rfsearch(), test\_rw\_qualitative(), and test\_rw\_quantitative().

### 5.35.2.2 `size_t fs_write ( m_inode * inode, void * buf, size_t num_bytes, uint32 pos, bool allow_scaling )`

Write content to HD.

#### Parameters

*inode* destination inode  
*buf* source buffer  
*num\_bytes* number of bytes desired to write  
*pos* start position  
*allow\_scaling* enlarge the file if EOF is reached?

#### Returns

number of bytes written

References BLOCK\_SIZE, block\_buffer::cache, cache\_block(), free, fs\_dprintf, fs\_write(), get\_data\_block(), m\_inode::i\_modify\_ts, m\_inode::i\_size, mallocn, memcpy, NOT\_FOUND, read\_cache, time2str, write\_inode(), and wrt\_cache().

Referenced by do\_write(), fs\_write(), test\_rw\_qualitative(), and test\_rw\_quantitative().

## 5.36 src/kernel/fs/fs\_super.c File Reference

Basic functions of the super block.

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/debug.h"
#include "../include/assert.h"
#include "fs_const.h"
#include "fs_types.h"
#include "fs_super.h"
#include "fs_block_dev.h"
#include "fs_bmap.h"
#include "../io/io_rtc.h"
```

#### Functions

- [uint32 get\\_hdsize \(\)](#)  
*Returns the size of the hard disk.*
- [void init\\_super\\_block \(\)](#)  
*Initializes the super block.*
- [void dump\\_super \(\)](#)

*Dump the main attributes.*

- void `load_super_block ()`  
*Load the super block from HD.*
- void `write_super_block ()`  
*Write the super block from memory to HD.*

## Variables

- struct `super_block * super`  
*Pointer to super block structure.*

### 5.36.1 Detailed Description

Basic functions of the super block.

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.36.2 Function Documentation

#### 5.36.2.1 void `dump_super ( )`

Dump the main attributes.

References `ASSERT`, `fs_dprintf`, `NIL_SUPER`, `super_block::s_bmap`, `super_block::s_bmap_blocks`, `super_block::s_first_data_block`, `super_block::s_HD_size`, `super_block::s_iroot`, `super_block::s_magic_number`, and `super_block::s_max_file_size`.

Referenced by `create_fs()`.

#### 5.36.2.2 uint32 `get_hdsz ( )`

Returns the size of the hard disk.

#### Returns

size of the master hard disk in sectors



### 5.36.2.3 void init\_super\_block ( )

Initializes the super block.

References BLOCK\_SIZE, bmap, first\_data\_block, get\_hdsiz, num\_bmap\_blocks, NUM\_DIRECT\_POINTER, root, super\_block::s\_bmap, super\_block::s\_bmap\_blocks, super\_block::s\_dirt, super\_block::s\_first\_data\_block, super\_block::s\_HD\_size, super\_block::s\_iroot, super\_block::s\_magic\_number, super\_block::s\_max\_file\_size, super\_block::s\_modify\_ts, and super\_block::s\_read\_only.

Referenced by create\_fs().

### 5.36.2.4 void load\_super\_block ( )

Load the super block from HD.

References ASSERT, fs\_dprintf, NIL\_SUPER, rd\_block(), and SUPER\_BLOCK.

Referenced by load\_fs().

### 5.36.2.5 void write\_super\_block ( )

Write the super block from memory to HD.

References SUPER\_BLOCK, and wrt\_block().

Referenced by create\_fs(), and fs\_shutdown().

## 5.36.3 Variable Documentation

### 5.36.3.1 struct super\_block\* super

Pointer to super block structure.

Referenced by kb\_handler().

## 5.37 src/kernel/fs/fs\_super.h File Reference

The superblock table.

### Data Structures

- struct [super\\_block](#)

### Defines

- #define [NIL\\_SUPER](#) (struct [super\\_block](#) \*) 0

### Functions

- struct [super\\_block](#) [\\_\\_attribute\\_\\_\(\(packed\)\)](#) [super\\_block](#)
- void [init\\_super\\_block](#) ( )

*Initializes the super block.*

- void `load_super_block ()`  
*Load the super block from HD.*
- void `write_super_block ()`  
*Write the super block from memory to HD.*
- void `dump_super ()`  
*Dump the main attributes.*

## Variables

- `m_inode * root`  
*extern deklaration of root inode*
- `block_nr s_HD_size`
- `uint16 s_bmap_blocks`
- `block_nr s_first_data_block`
- `uint32 s_max_file_size`
- `uint8 * s_bmap`
- `m_inode * s_iroot`
- `time_t s_modify_ts`
- `bool s_read_only`
- `uint16 s_dirt`
- `uint32 s_magic_number`

### 5.37.1 Detailed Description

The superblock table.

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.37.2 Define Documentation

#### 5.37.2.1 `#define NIL_SUPER (struct super_block *) 0`

Referenced by `dump_super()`, and `load_super_block()`.

### 5.37.3 Function Documentation

#### 5.37.3.1 struct super\_block \_\_attribute\_\_((packed))

#### 5.37.3.2 void dump\_super()

Dump the main attributes.

References ASSERT, fs\_dprintf, NIL\_SUPER, super\_block::s\_bmap, super\_block::s\_bmap\_blocks, super\_block::s\_first\_data\_block, super\_block::s\_HD\_size, super\_block::s\_iroot, super\_block::s\_magic\_number, and super\_block::s\_max\_file\_size.

Referenced by create\_fs().

#### 5.37.3.3 void init\_super\_block()

Initializes the super block.

References BLOCK\_SIZE, bmap, first\_data\_block, get\_hdsize, num\_bmap\_blocks, NUM\_DIRECT\_POINTER, root, super\_block::s\_bmap, super\_block::s\_bmap\_blocks, super\_block::s\_dirt, super\_block::s\_first\_data\_block, super\_block::s\_HD\_size, super\_block::s\_iroot, super\_block::s\_magic\_number, super\_block::s\_max\_file\_size, super\_block::s\_modify\_ts, and super\_block::s\_read\_only.

Referenced by create\_fs().

#### 5.37.3.4 void load\_super\_block()

Load the super block from HD.

References ASSERT, fs\_dprintf, NIL\_SUPER, rd\_block(), and SUPER\_BLOCK.

Referenced by load\_fs().

#### 5.37.3.5 void write\_super\_block()

Write the super block from memory to HD.

References SUPER\_BLOCK, and wrt\_block().

Referenced by create\_fs(), and fs\_shutdown().

### 5.37.4 Variable Documentation

#### 5.37.4.1 m\_inode\* root

extern declaration of root inode

Referenced by create\_root(), delete\_file\_from\_dir(), fs\_open(), init\_super\_block(), insert\_file\_into\_dir(), load\_root(), rfsearch(), search\_file(), test\_file\_table(), test\_sync(), and write\_root().

5.37.4.2 `uint8* s_bmap`

5.37.4.3 `uint16 s_bmap_blocks`

5.37.4.4 `uint16 s_dirt`

5.37.4.5 `block_nr s_first_data_block`

5.37.4.6 `block_nr s_HD_size`

5.37.4.7 `m_inode* s_iroot`

5.37.4.8 `uint32 s_magic_number`

5.37.4.9 `uint32 s_max_file_size`

5.37.4.10 `time_t s_modify_ts`

5.37.4.11 `bool s_read_only`

## 5.38 `src/kernel/fs/fs_tests.c` File Reference

Some fs test-functions.

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/string.h"
#include "../include/debug.h"
#include "../include/stdlib.h"
#include "../include/assert.h"
#include "fs_const.h"
#include "fs_types.h"
#include "fs_file_table.h"
#include "fs_inode_table.h"
#include "fs_io_functions.h"
#include "fs_super.h"
#include "fs_bmap.h"
#include "fs_main.h"
#include "fs_buf.h"
#include "fs_block_dev.h"
```

### Functions

- void [test\\_bmap](#) ()
- void [test\\_inode\\_table](#) ()

- void [test\\_file\\_table](#) ()
- void [test\\_reload](#) ()
- void [\\_\\_ls](#) (char \*path)
- void [test\\_create](#) ()
- void [mem\\_dump](#) ()
- void [test\\_rw\\_quantitative](#) ()
- void [test\\_rw\\_qualitative](#) ()
- void [test\\_PM](#) ()
- void [test\\_error](#) ()
- void [test\\_close](#) ()
- void [test\\_ls](#) ()
- void [test\\_sync](#) ()
- void [test\\_delete](#) ()
- void [test\\_open\\_close](#) ()
- void [run\\_FS\\_tests](#) ()

### 5.38.1 Detailed Description

Some fs test-functions.

#### Author

Vincenz Doelle

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.38.2 Function Documentation

#### 5.38.2.1 void [\\_\\_ls](#) ( char \* *path* )

References [bzero](#), [do\\_close](#)(), [do\\_open](#)(), [do\\_read](#)(), [dprintf](#), [inode](#), and [name](#).

Referenced by [test\\_create](#)(), [test\\_delete](#)(), and [test\\_ls](#)().

#### 5.38.2.2 void [mem\\_dump](#) ( )

#### 5.38.2.3 void [run\\_FS\\_tests](#) ( )

References [test\\_open\\_close](#)().

Referenced by [fs\\_tests](#)().

**5.38.2.4 void test\_bmap ( )**

References `alloc_block()`, `dprintf`, `dump_bmap()`, `FALSE`, `get_free_block()`, `is_allocated_block()`, `mark_block()`, and `TRUE`.

**5.38.2.5 void test\_close ( )**

References `do_close_pf()`, `do_mkdir()`, `do_mkfile()`, `do_open()`, `dump_files()`, `dump_proc_files()`, `init_proc_file_table()`, and `insert_proc_file()`.

**5.38.2.6 void test\_create ( )**

References `__ls()`, `bzero`, `DIRECTORY`, `dprintf`, `fs_create()`, `itoa`, and `strcat`.

**5.38.2.7 void test\_delete ( )**

References `__ls()`, `DIRECTORY`, `dprintf`, `dump_bmap()`, `fs_create()`, and `fs_delete()`.

**5.38.2.8 void test\_error ( )**

References `do_mkdir()`, and `dprintf`.

**5.38.2.9 void test\_file\_table ( )**

References `contains_file()`, `DATA_FILE`, `DIRECTORY`, `dprintf`, `dump_files()`, `dump_inodes()`, `FALSE`, `free_file()`, `init_file_table()`, `inode2desc()`, `insert_file()`, `name`, `name2desc()`, `new_minode()`, `NOT_POSSIBLE`, `root`, and `TRUE`.

**5.38.2.10 void test\_inode\_table ( )**

References `alloc_inode()`, `dprintf`, `dump_inodes()`, `free_inode()`, and `m_inode::i_num`.

**5.38.2.11 void test\_ls ( )**

References `__ls()`, `bzero`, `DATA_FILE`, `DIRECTORY`, `do_create()`, and `do_open()`.

**5.38.2.12 void test\_open\_close ( )**

References `DIRECTORY`, `dprintf`, `dump_files()`, `free_proc_file()`, `fs_close()`, `fs_create()`, `fs_open()`, `init_proc_file_table()`, `insert_proc_file()`, and `mallocn`.

Referenced by `run_FS_tests()`.

**5.38.2.13 void test\_PM ( )**

References `bzero`, `do_close_pf()`, `do_create()`, `do_mkdir()`, `do_open()`, `do_read()`, `dprintf`, `get_proc_file()`, `init_proc_file_table()`, `insert_proc_file()`, and `proc_file::pf_f_desc`.

#### 5.38.2.14 void test\_reload ( )

#### 5.38.2.15 void test\_rw\_qualitative ( )

References bzero, DATA\_FILE, dprintf, file::f\_inode, FALSE, fs\_create(), fs\_open(), fs\_read(), fs\_write(), get\_file(), name2desc(), NOT\_POSSIBLE, strcat, and TRUE.

#### 5.38.2.16 void test\_rw\_quantitative ( )

References bzero, DATA\_FILE, dprintf, file::f\_inode, FALSE, fs\_create(), fs\_open(), fs\_read(), fs\_write(), get\_file(), mem\_dump, name2desc(), NOT\_POSSIBLE, and TRUE.

#### 5.38.2.17 void test\_sync ( )

References dump\_bmap(), dump\_inode(), fs\_shutdown(), load\_fs(), and root.

## 5.39 src/kernel/fs/fs\_types.h File Reference

Basic type definitions.

```
#include "fs_const.h"
#include "../io/io_rtc.h"
#include "../io/io_rtc.h"
```

### Data Structures

- struct [dir\\_entry](#)  
*The directory entry.*
- struct [d\\_inode](#)  
*The inode on disk.*
- struct [m\\_inode](#)  
*The inode in memory.*
- struct [file](#)  
*The global file descriptor.*
- struct [proc\\_file](#)  
*The process file descriptor.*
- struct [block\\_buffer](#)  
*A buffer for one block.*
- struct [file\\_info](#)

## Typedefs

- typedef [uint32](#) [block\\_nr](#)
- typedef [sint16](#) [inode\\_nr](#)
- typedef [sint16](#) [file\\_nr](#)
- typedef struct [m\\_inode](#) [m\\_inode](#)  
*The inode in memory.*
- typedef struct [file](#) [file](#)  
*The global file descriptor.*
- typedef struct [proc\\_file](#) [proc\\_file](#)  
*The process file descriptor.*
- typedef struct [block\\_buffer](#) [block\\_buffer](#)  
*A buffer for one block.*
- typedef [block\\_buffer](#) [block\\_cache](#)  
*The cache for one generic block.*
- typedef struct [file\\_info](#) [file\\_info\\_t](#)

## Functions

- struct [dir\\_entry](#) [\\_\\_attribute\\_\\_\(\(packed\)\)](#) [dir\\_entry](#)  
*The directory entry.*

## Variables

- [block\\_nr](#) [inode](#)
- char [name](#) [[NAME\\_SIZE](#)]
- [uint16](#) [i\\_mode](#)
- [uint32](#) [i\\_size](#)
- [time\\_t](#) [i\\_create\\_ts](#)
- [time\\_t](#) [i\\_modify\\_ts](#)
- [block\\_nr](#) [i\\_direct\\_pointer](#) [[NUM\\_DIRECT\\_POINTER](#)]
- [block\\_nr](#) [i\\_single\\_indirect\\_pointer](#)
- [block\\_nr](#) [i\\_double\\_indirect\\_pointer](#)

### 5.39.1 Detailed Description

Basic type definitions. This file defines basic data types used throughout the file system. You should use these types whenever possible in order to avoid the possible ambiguity of the builtin types.

#### Author

Vincenz Doelle



**LastChangedBy:**

dtraytel

**Version****Rev:**

12

## 5.39.2 Typedef Documentation

### 5.39.2.1 typedef struct block\_buffer block\_buffer

A buffer for one block.

### 5.39.2.2 typedef block\_buffer block\_cache

The cache for one generic block.

### 5.39.2.3 typedef uint32 block\_nr

### 5.39.2.4 typedef struct file file

The global file descriptor.

### 5.39.2.5 typedef struct file\_info file\_info\_t

### 5.39.2.6 typedef sint16 file\_nr

### 5.39.2.7 typedef sint16 inode\_nr

### 5.39.2.8 typedef struct m\_inode m\_inode

The inode in memory.

### 5.39.2.9 typedef struct proc\_file proc\_file

The process file descriptor.

## 5.39.3 Function Documentation

### 5.39.3.1 struct dir\_entry \_\_attribute\_\_((packed))

The directory entry.

The inode on disk.

## 5.39.4 Variable Documentation

### 5.39.4.1 `time_t i_create_ts`

Referenced by `get_file_info()`.

### 5.39.4.2 `block_nr i_direct_pointer[NUM_DIRECT_POINTER]`

### 5.39.4.3 `block_nr i_double_indirect_pointer`

### 5.39.4.4 `uint16 i_mode`

### 5.39.4.5 `time_t i_modify_ts`

Referenced by `get_file_info()`.

### 5.39.4.6 `block_nr i_single_indirect_pointer`

### 5.39.4.7 `uint32 i_size`

Referenced by `fs_truncate()`, and `get_file_info()`.

### 5.39.4.8 `block_nr inode`

Referenced by `__ls()`, `delete_entry()`, `find_filename()`, `fs_create_delete()`, `fs_open()`, `fs_truncate()`, and `shell_cmd_ls()`.

### 5.39.4.9 `char name[NAME_SIZE]`

Referenced by `__ls()`, `chips_usage()`, `get_active_virt_monitor_name()`, `main()`, `new_shell()`, `new_virt_monitor()`, `shell_cmd_ls()`, `shell_cmd_snake()`, `shell_handle_command()`, and `test_file_table()`.

## 5.40 `src/kernel/include/assert.h` File Reference

Definition of the `ASSERT()` macro.

### Defines

- `#define ASSERT(x)`

### 5.40.1 Detailed Description

Definition of the `ASSERT()` macro.

### Author

Daniel Bader

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

## 5.40.2 Define Documentation

### 5.40.2.1 #define ASSERT( x )

**Value:**

```
if (!(x)) { \
    printf("ASSERTION FAILED: %s:%s():%d. Expr: %s\n", __FILE__, __FUNCTION__
, __LINE__, #x); }
```

Referenced by `assert_test()`, `create_root()`, `dump_super()`, `getpid()`, `hd_stressread_test()`, `hd_stresswrite_test()`, `hd_write_test()`, `heap_contract()`, `heap_expand()`, `init_bf()`, `init_bmap()`, `init_vmonitors()`, `interpret_bf()`, `load_bmap()`, `load_root()`, `load_super_block()`, `new_virt_monitor()`, `pm_create_thread()`, `reset_bf()`, `rf_clear()`, `rf_dump()`, `rf_free()`, `rf_getlength()`, `rf_isempty()`, `rf_isfull()`, `sys_exit()`, and `write_root()`.

## 5.41 src/kernel/include/const.h File Reference

Basic constant definitions.

### Defines

- #define `NULL` 0
- #define `TRUE` 1
- #define `FALSE` 0
- #define `FREQUENCY` 100
- #define `VGA_DISPLAY` 0xB8000
- #define `EOF` -1

### 5.41.1 Detailed Description

Basic constant definitions. This file defines constants used throughout the kernel.

**Author**

Daniel Bader

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

**5.41.2 Define Documentation****5.41.2.1 #define EOF -1**

Referenced by `sys_close()`.

**5.41.2.2 #define FALSE 0**

Referenced by `assert_test()`, `create_root()`, `delete_entry()`, `delete_file_from_dir()`, `do_read()`, `fs_create_delete()`, `fs_truncate()`, `get_free_block()`, `insert_file_into_dir()`, `io_init()`, `rfsearch()`, `snake()`, `test_bmap()`, `test_file_table()`, `test_rw_qualitative()`, and `test_rw_quantitative()`.

**5.41.2.3 #define FREQUENCY 100**

Referenced by `io_init()`, and `sleep()`.

**5.41.2.4 #define NULL 0**

Referenced by `atoi()`, `callocn()`, `contains_file()`, `create_root()`, `dump_bmap()`, `free_file()`, `free_proc_file()`, `fs_close()`, `fs_create_delete()`, `fs_delete()`, `fs_open()`, `fs_truncate()`, `get_data_block()`, `get_filename()`, `get_path()`, `getpid()`, `heap_mallocn()`, `init_bf()`, `insert_file()`, `insert_proc_file()`, `interpret_bf()`, `load_root()`, `lseek()`, `main()`, `malloc_bmap()`, `malloc_test()`, `name2desc()`, `new_minode()`, `pm_create_thread()`, `pm_destroy_thread()`, `pm_fd2device()`, `pm_handle_input()`, `pm_name2device()`, `pm_register_device()`, `pm_schedule()`, `potatoes_disc_create()`, `potatoes_disc_destroy()`, `potatoes_get_hdsizes()`, `potatoes_hd_read_sector()`, `potatoes_hd_write_sector()`, `realloc()`, `reset_bf()`, `rf_alloc()`, `rf_clear()`, `rf_dump()`, `rf_free()`, `rf_getlength()`, `rf_isempty()`, `rf_isfull()`, `rf_read()`, `rf_write()`, `rfsearch()`, `shell_autocomplete()`, `shell_cmd_cmdlist()`, `shell_cmd_ls()`, `shell_handle_command()`, `strdup()`, `strsep()`, `strsep_test()`, `strtoul()`, `sys_exit()`, `sys_getpid()`, `sys_open()`, `vsnprintf()`, and `write_root()`.

**5.41.2.5 #define TRUE 1**

Referenced by `alloc_block()`, `assert_test()`, `do_write()`, `fs_truncate()`, `init_bmap()`, `insert_file_into_dir()`, `interpret_bf()`, `mark_block()`, `shell_cmd_pong()`, `snake()`, `test_bmap()`, `test_file_table()`, `test_rw_qualitative()`, and `test_rw_quantitative()`.

**5.41.2.6 #define VGA\_DISPLAY 0xB8000**

Referenced by `dev_framebuffer_write()`, `make_snapshot()`, `monitor_cputc()`, `shell_cmd_snapshot()`, and `update_virt_monitor()`.

## 5.42 src/kernel/include/debug.h File Reference

Debugging utility functions.

```
#include "stdio.h"
#include "../io/io.h"
#include "../io/io.h"
```

### Defines

- #define [dprintf](#) printf
- #define [dprint\\_separator](#)( ) dprintf("#{GRE}-----  
-----##");
- #define [fs\\_dprintf](#)(a,...) ((void)0)
- #define [SHORTCUT\\_CTRL](#)(ch, func)
- #define [SHORTCUT\\_SUPER](#)(ch, func)
- #define [SHORTCUT\\_CTRL\\_SUPER](#)(ch, func)

### 5.42.1 Detailed Description

Debugging utility functions. This is work in progress so far. I plan on implementing a small toolbox of useful debugging functions, eg debug printf, stack crawling etc.

#### Author

dbader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.42.2 Define Documentation

**5.42.2.1** #define [dprint\\_separator](#)( ) dprintf("#{GRE}-----  
-----##");

Referenced by [do\\_tests](#)(), [fs\\_init](#)(), [io\\_init](#)(), [main](#)(), [mm\\_init\\_output](#)(), and [pm\\_init](#)().

#### 5.42.2.2 #define [dprintf](#) printf

Referenced by [\\_\\_ls](#)(), [atoi\\_test](#)(), [dev\\_framebuffer\\_open](#)(), [dev\\_null\\_read](#)(), [dev\\_null\\_write](#)(), [fs\\_init](#)(), [gdt\\_add\\_entry](#)(), [heap\\_free](#)(), [io\\_init](#)(), [mm\\_init\\_output](#)(), [pm\\_create\\_thread](#)(), [pm\\_destroy\\_thread](#)(), [pm\\_handle\\_input](#)(), [pm\\_init](#)(), [pm\\_register\\_device](#)(), [pm\\_set\\_thread\\_priority](#)(), [sys\\_exit](#)(), [sys\\_stat](#)(), [sys\\_unlink](#)(), [test\\_bmap](#)(), [test\\_create](#)(), [test\\_delete](#)(), [test\\_error](#)(), [test\\_file\\_table](#)(), [test\\_inode\\_table](#)(), [test\\_open\\_close](#)(), [test\\_PM](#)(), [test\\_rw\\_qualitative](#)(), and [test\\_rw\\_quantitative](#)().

### 5.42.2.3 #define fs\_dprintf( a, ... )((void)0)

Referenced by `delete_entry()`, `delete_file_from_dir()`, `do_read()`, `dump_bmap()`, `dump_dinode()`, `dump_file()`, `dump_files()`, `dump_inode()`, `dump_proc_file()`, `dump_proc_files()`, `dump_super()`, `find_filename()`, `fs_close()`, `fs_create_delete()`, `fs_open()`, `fs_read()`, `fs_truncate()`, `fs_write()`, `get_data_block()`, `get_filename()`, `get_path()`, `init_bmap()`, `init_inode_table()`, `init_proc_file_table()`, `insert_file()`, `insert_file_into_dir()`, `insert_proc_file()`, `load_bmap()`, `load_root()`, `load_super_block()`, `malloc_bmap()`, `mark_block()`, `name2desc()`, `rfsearch()`, `search_file()`, and `write_bmap()`.

### 5.42.2.4 #define SHORTCUT\_CTRL( ch, func )

#### Value:

```
add_shortcut(TRUE, FALSE, ch, func); \
    printf("\tCTRL + %c ==> %s()\n", ch, #func)
```

Referenced by `do_tests()`.

### 5.42.2.5 #define SHORTCUT\_CTRL\_SUPER( ch, func )

#### Value:

```
add_shortcut(TRUE, TRUE, ch, func); \
    printf("\tCTRL + SUPER + %c ==> %s()\n", ch, #func)
```

Referenced by `do_tests()`.

### 5.42.2.6 #define SHORTCUT\_SUPER( ch, func )

#### Value:

```
add_shortcut(FALSE, TRUE, ch, func); \
    printf("\tSUPER + %c ==> %s()\n", ch, #func)
```

## 5.43 src/kernel/include/init.h File Reference

Basic definitions for functions used in the `main()`-function of the kernel.

```
#include "types.h"
```

### Data Structures

- struct `multiboot`  
*Multiboot structure.*

### Functions

- struct `multiboot __attribute__((packed))`  
*Multiboot structure.*

- void `mm_init` (`uint32 start`, `uint32 end`)  
*initializes memory management (includeing the GDT)*
- void `mm_init2` ()
- void `io_init` ()
- void `pm_init` ()  
*Performs process management initializations.*
- void `fs_init` ()  
*Initializes the file system.*
- void `fs_shutdown` ()  
*Shuts the file system down and writes all important information to HD.*
- void `panic` (`char *msg`)  
*Kernel panic function.*
- void `do_tests` ()

## Variables

- `uint32 flags`
- `uint32 mem_lower`
- `uint32 mem_upper`
- `uint32 boot_device`
- `uint32 cmdline`
- `uint32 mods_count`
- `uint32 mods_addr`
- `uint32 num`
- `uint32 size`
- `uint32 addr`
- `uint32 shndx`
- `uint32 mmap_length`
- `uint32 mmap_addr`
- `uint32 drives_length`
- `uint32 drives_addr`
- `uint32 config_table`
- `uint32 boot_loader_name`
- `uint32 apm_table`
- `uint32 vbe_control_info`
- `uint32 vbe_mode_info`
- `uint32 vbe_mode`
- `uint32 vbe_interface_seg`
- `uint32 vbe_interface_off`
- `uint32 vbe_interface_len`
- `struct multiboot * g_mboot_ptr`  
*Global pointer to multiboot structure.*

### 5.43.1 Detailed Description

Basic definitions for functions used in the `main()`-function of the kernel.

**Author**

Dmitriy Traytel

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

### 5.43.2 Function Documentation

#### 5.43.2.1 `struct multiboot __attribute__ ( (packed) )`

Multiboot structure.

**See also**

<http://www.gnu.org/software/grub/manual/multiboot/multiboot.html>

#### 5.43.2.2 `void do_tests ( )`

References `_malloc()`, `atoi_test()`, `dprint_separator`, `make_snapshot()`, `new_shell()`, `paralleleModellierung()`, `printf`, `ralph_wiggum()`, `reboot()`, `sema_init()`, `SHORTCUT_CTRL`, `SHORTCUT_CTRL_SUPER`, `switch_monitor_down()`, `switch_monitor_up()`, `test_batch_files()`, `threadA_test()`, `threadB_test()`, `threadC_test()`, `threadConsumer_test()`, `threadD_test()`, and `threadProducer_test()`.

Referenced by `main()`.

#### 5.43.2.3 `void fs_init ( )`

Initializes the file system.

#### 5.43.2.4 `void fs_shutdown ( )`

Shuts the file system down and writes all important information to HD.

#### 5.43.2.5 `void io_init ( )`

References `dprint_separator`, `dprintf`, `FALSE`, `FREQUENCY`, `hd_init()`, `idt_init()`, `irq_init()`, `isr_init()`, `keyboard_state`, `memset()`, `set_interrupts()`, and `timer_init()`.

Referenced by `main()`.



### 5.43.2.6 void mm\_init ( uint32 start, uint32 end )

initializes memory management (includeing the GDT)

#### Parameters

*start* start address of the free memory (= end of kernel)

*end* end address of the available memory

References alloc\_frame(), bzero, create\_heap(), current\_dir, frames, get\_page(), kernel\_dir, kernel\_heap, KHEAP\_START, kmalloc(), nframes, placement\_addr, and switch\_page\_dir().

Referenced by main().

### 5.43.2.7 void mm\_init2 ( )

### 5.43.2.8 void panic ( char \* msg )

Kernel panic function.

Displays an error message and enters an infinite loop thus effectively halting the kernel. This should only be called when a non-recoverable ("fatal") error occurs inside the kernel.

#### Parameters

*msg* Message to display.

References BLACK, clear\_interrupts(), halt(), monitor\_cputs(), and RED.

### 5.43.2.9 void pm\_init ( )

Performs process management initializations.

Referenced by main().

## 5.43.3 Variable Documentation

### 5.43.3.1 uint32 addr

Referenced by grubstruct\_test().

**5.43.3.2 uint32 apm\_table**

**5.43.3.3 uint32 boot\_device**

**5.43.3.4 uint32 boot\_loader\_name**

**5.43.3.5 uint32 cmdline**

**5.43.3.6 uint32 config\_table**

**5.43.3.7 uint32 drives\_addr**

**5.43.3.8 uint32 drives\_length**

**5.43.3.9 uint32 flags**

Referenced by `grubstruct_test()`.

**5.43.3.10 struct multiboot\* g\_mboot\_ptr**

Global pointer to multiboot structure.

Global pointer to multiboot structure.

This should later be removed and only used in the kernel `main()`.

**5.43.3.11 uint32 mem\_lower**

Referenced by `grubstruct_test()`.

**5.43.3.12 uint32 mem\_upper**

Referenced by `grubstruct_test()`.

**5.43.3.13 uint32 mmap\_addr**

**5.43.3.14 uint32 mmap\_length**

**5.43.3.15 uint32 mods\_addr**

**5.43.3.16 uint32 mods\_count**

**5.43.3.17 uint32 num**

Referenced by `itoa()`.

#### 5.43.3.18 uint32 shndx

#### 5.43.3.19 uint32 size

Referenced by grubstruct\_test(), init\_bmap(), load\_bmap(), main(), malloc\_bmap(), shell\_cmd\_synth(), and speed().

#### 5.43.3.20 uint32 vbe\_control\_info

#### 5.43.3.21 uint32 vbe\_interface\_len

#### 5.43.3.22 uint32 vbe\_interface\_off

#### 5.43.3.23 uint32 vbe\_interface\_seg

#### 5.43.3.24 uint32 vbe\_mode

#### 5.43.3.25 uint32 vbe\_mode\_info

## 5.44 src/kernel/include/limits.h File Reference

Definition of the maximum and minimum values of different types.

### Defines

- #define [SINT8\\_MAX](#) 0x7F
- #define [UINT8\\_MAX](#) 0xFF
- #define [SINT16\\_MAX](#) 0x7FFF
- #define [UINT16\\_MAX](#) 0xFFFF
- #define [SINT32\\_MAX](#) 0x7FFFFFFF
- #define [UINT32\\_MAX](#) 0xFFFFFFFF
- #define [SINT8\\_MIN](#) 0x80
- #define [SINT16\\_MIN](#) 0x8000
- #define [SINT32\\_MIN](#) 0x80000000

### 5.44.1 Detailed Description

Definition of the maximum and minimum values of different types.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

Rev:

12

## 5.44.2 Define Documentation

5.44.2.1 **#define SINT16\_MAX 0x7FFF**

5.44.2.2 **#define SINT16\_MIN 0x8000**

5.44.2.3 **#define SINT32\_MAX 0x7FFFFFFF**

Referenced by `sleep()`, `sleep_ticks()`, and `timer_handler()`.

5.44.2.4 **#define SINT32\_MIN 0x80000000**

5.44.2.5 **#define SINT8\_MAX 0x7F**

5.44.2.6 **#define SINT8\_MIN 0x80**

5.44.2.7 **#define UINT16\_MAX 0xFFFF**

5.44.2.8 **#define UINT32\_MAX 0xFFFFFFFF**

Referenced by `shell_cmd_bf()`.

5.44.2.9 **#define UINT8\_MAX 0xFF**

## 5.45 src/kernel/include/ringbuffer.h File Reference

Ringbuffer definitions.

```
#include "../include/types.h"
```

### Data Structures

- struct [ring\\_fifo](#)  
*A static circular FIFO buffer.*

### Functions

- [ring\\_fifo \\* rf\\_alloc](#) ([uint32 size](#))  
*Allocates and initializes a new ring\_buffer.*
- [ring\\_fifo \\* rf\\_copy](#) ([ring\\_fifo \\*fifo](#))  
*Copies the given ring\_buffer into a new allocated copy.*
- void [rf\\_free](#) ([ring\\_fifo \\*fifo](#))

*Destroys a previously allocated ring\_buffer.*

- void [rf\\_clear](#) ([ring\\_fifo](#) \*fifo)  
*Clears a ring buffer.*
- [uint32 rf\\_getlength](#) ([ring\\_fifo](#) \*fifo)  
*Returns the buffer's number of used bytes.*
- [bool rf\\_isfull](#) ([ring\\_fifo](#) \*fifo)  
*Checks whether the given ring\_buffer is full.*
- [bool rf\\_isempty](#) ([ring\\_fifo](#) \*fifo)  
*Checks whether the given ring\_buffer is empty.*
- void [rf\\_dump](#) ([ring\\_fifo](#) \*fifo)  
*Prints information about the buffer to stdout.*
- [sint32 rf\\_write](#) ([ring\\_fifo](#) \*fifo, [uint8](#) \*buf, [uint32](#) count)  
*Writes data into a ring\_buffer.*
- [sint32 rf\\_read](#) ([ring\\_fifo](#) \*fifo, [uint8](#) \*buf, [uint32](#) count)  
*Reads data from a ring\_buffer.*

## Variables

- typedef [\\_\\_attribute\\_\\_](#)  
*The inode on disk.*

### 5.45.1 Detailed Description

Ringbuffer definitions.

#### Author

dbader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

## 5.45.2 Function Documentation

### 5.45.2.1 `ring_fifo* rf_alloc ( uint32 size )`

Allocates and initializes a new `ring_buffer`.

#### Parameters

*size* max size of the buffer

#### Returns

pointer to new `ring_buffer` or `NULL` on error

References `ring_fifo::data`, `mallocn`, `NULL`, `rf_clear()`, and `ring_fifo::size`.

Referenced by `pm_create_thread()`, `pm_init()`, `rf_copy()`, and `snake()`.

### 5.45.2.2 `void rf_clear ( ring_fifo * fifo )`

Clears a ring buffer.

#### Parameters

*fifo* the buffer

References `ASSERT`, `ring_fifo::end`, `ring_fifo::len`, `NULL`, and `ring_fifo::start`.

Referenced by `rf_alloc()`.

### 5.45.2.3 `ring_fifo* rf_copy ( ring_fifo * fifo )`

Copies the given `ring_buffer` into a new allocated copy.

#### Parameters

*fifo* the `ring_buffer` to copy from

#### Returns

pointer to the new copy of the given `ring_buffer`

References `ring_fifo::data`, `ring_fifo::end`, `ring_fifo::len`, `memcpy`, `rf_alloc()`, `ring_fifo::size`, and `ring_fifo::start`.

Referenced by `body_collision()`, and `draw_snake()`.

### 5.45.2.4 `void rf_dump ( ring_fifo * fifo )`

Prints information about the buffer to `stdout`.

#### Parameters

*fifo* the buffer

References `ASSERT`, `ring_fifo::data`, `ring_fifo::end`, `ring_fifo::len`, `NULL`, `printf`, `ring_fifo::size`, and `ring_fifo::start`.

#### 5.45.2.5 void rf\_free ( ring\_fifo \* fifo )

Destroys a previously allocated ring\_buffer.

##### Parameters

*fifo* the ring buffer to free

References ASSERT, ring\_fifo::data, free, and NULL.

Referenced by body\_collision(), draw\_snake(), pm\_destroy\_thread(), and snake().

#### 5.45.2.6 uint32 rf\_getlength ( ring\_fifo \* fifo )

Returns the buffer's number of used bytes.

##### Parameters

*fifo* the buffer

##### Returns

Number of used bytes

References ASSERT, ring\_fifo::len, and NULL.

Referenced by body\_collision(), dev\_stdin\_read(), draw\_snake(), and snake().

#### 5.45.2.7 bool rf\_isempty ( ring\_fifo \* fifo )

Checks whether the given ring\_buffer is empty.

##### Parameters

*fifo* the buffer to check

##### Returns

TRUE if the buffer contains at least one byte of data, false if not

References ASSERT, ring\_fifo::len, and NULL.

#### 5.45.2.8 bool rf\_isfull ( ring\_fifo \* fifo )

Checks whether the given ring\_buffer is full.

##### Parameters

*fifo* the buffer to check

##### Returns

TRUE if at least one byte of storage is available, FALSE if not.

References ASSERT, ring\_fifo::len, NULL, and ring\_fifo::size.

#### 5.45.2.9 `sint32 rf_read ( ring_fifo * fifo, uint8 * buf, uint32 count )`

Reads data from a `ring_buffer`.

Less than `count` bytes might be read if the buffer becomes empty.

##### Parameters

*fifo* `ring_buffer` to read from

*buf* buffer to write to

*count* number of bytes to write

##### Returns

Number of bytes read, -1 on error

References `ring_fifo::data`, `ring_fifo::len`, `NULL`, `ring_fifo::size`, and `ring_fifo::start`.

Referenced by `body_collision()`, `dev_stdin_read()`, `draw_snake()`, and `snake()`.

#### 5.45.2.10 `sint32 rf_write ( ring_fifo * fifo, uint8 * buf, uint32 count )`

Writes data into a `ring_buffer`.

If the buffer gets filled up less than `count` bytes might be written.

##### Parameters

*fifo* `ring_buffer` to write to

*buf* buffer to read from

*count* number of bytes to write

##### Returns

Number of bytes written, -1 on error.

References `ring_fifo::data`, `ring_fifo::end`, `ring_fifo::len`, `NULL`, `printf`, and `ring_fifo::size`.

Referenced by `dev_stdin_write()`, `pm_handle_input()`, and `snake()`.

### 5.45.3 Variable Documentation

#### 5.45.3.1 `struct gdt_pointer __attribute__`

The inode on disk.

The structure of the GDT pointer which tells the processor where to find our GDT.

The structure of the IDT pointer which tells the processor where to find our IDT.

## 5.46 `src/kernel/include/stdarg.h` File Reference

Macros dealing with variable argument lists.



## Defines

- #define [va\\_start](#)(v, l) `__builtin_va_start(v,l)`
- #define [va\\_end](#)(v) `__builtin_va_end(v)`
- #define [va\\_arg](#)(v, l) `__builtin_va_arg(v,l)`

## Typedefs

- typedef `__builtin_va_list` [va\\_list](#)

### 5.46.1 Detailed Description

Macros dealing with variable argument lists.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.46.2 Define Documentation

#### 5.46.2.1 #define [va\\_arg](#)( v, l ) `__builtin_va_arg(v,l)`

Referenced by `vsnprintf()`.

#### 5.46.2.2 #define [va\\_end](#)( v ) `__builtin_va_end(v)`

Referenced by `_printf()`, `aprntf()`, `potatoes_printf()`, `printf()`, and `snprintf()`.

#### 5.46.2.3 #define [va\\_start](#)( v, l ) `__builtin_va_start(v,l)`

Referenced by `_printf()`, `aprntf()`, `potatoes_printf()`, `printf()`, and `snprintf()`.

### 5.46.3 Typedef Documentation

#### 5.46.3.1 typedef `__builtin_va_list` [va\\_list](#)

## 5.47 src/kernel/include/stdio.h File Reference

Standard I/O functions.

```
#include "types.h"
#include "stdarg.h"
```

## Enumerations

- enum `colors` {  
    **BLACK** = 0x0, **BLUE** = 0x1, **GREEN** = 0x2, **CYAN** = 0x3,  
    **RED** = 0x4, **VIOLET** = 0x5, **ORANGE** = 0x6, **LIGHTGREY** = 0x7,  
    **DARKGREY** = 0x8, **LIGHTBLUE** = 0x9, **LIGHTGREEN** = 0xA, **TURQUOISE** = 0xB,  
    **PINK** = 0xC, **MAGENTA** = 0xD, **YELLOW** = 0xE, **WHITE** = 0xF }

## Functions

- void `monitor_cputc` (char ch, uint8 fg, uint8 bg)  
    *Writes a colored character to the display.*
- void `monitor_cputs` (char \*str, uint8 fg, uint8 bg)  
    *Writes a colored null-terminated string to the display.*
- void `monitor_putc` (char ch)  
    *Writes a character to the display.*
- void `monitor_puts` (char \*str)  
    *Writes a null-terminated string to the display.*
- void `monitor_puti` (sint32 x)  
    *Writes an integer to the display.*
- void `monitor_puthex` (uint8 ch)  
    *Writes a hex-byte to the display.*
- void `monitor_scrollup` ()
- void `monitor_scrolldown` ()
- int `putchar` (char c)  
    *Writes a character to stdout.*
- int `cputchar` (char c, uint8 fg, uint8 bg)
- int `puts` (char \*s)  
    *Writes a string to stdout.*
- int `cputs` (char \*s, uint8 fg, uint8 bg)
- int `vsnprintf` (char \*s, int n, char \*format, va\_list ap)
- int `snprintf` (char \*buf, int size, char \*fmt,...)
- void `printf` (char \*fmt,...)  
    *Prints formatted output.*
- void `hd_write_sector` (uint32 dest, void \*src)  
    *Writes a sector to the hard disk.*

- void `hd_read_sector` (void \*dest, uint32 src)  
*Reads a sector from the hard disk.*

## Variables

- `uint32 maxaddr`

### 5.47.1 Detailed Description

Standard I/O functions.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.47.2 Enumeration Type Documentation

#### 5.47.2.1 enum colors

##### Enumerator:

*BLACK*  
*BLUE*  
*GREEN*  
*CYAN*  
*RED*  
*VIOLET*  
*ORANGE*  
*LIGHTGREY*  
*DARKGREY*  
*LIGHTBLUE*  
*LIGHTGREEN*  
*TURQUOISE*  
*PINK*  
*MAGENTA*  
*YELLOW*  
*WHITE*

### 5.47.3 Function Documentation

#### 5.47.3.1 int cputchar ( char *c*, uint8 *fg*, uint8 *bg* )

References SELECT\_VMONITOR, and virt\_monitor\_cputc().

#### 5.47.3.2 int cputs ( char \* *s*, uint8 *fg*, uint8 *bg* )

References SELECT\_VMONITOR, and virt\_monitor\_cputs().

#### 5.47.3.3 void hd\_read\_sector ( void \* *dest*, uint32 *src* )

Reads a sector from the hard disk.

##### Parameters

*\*dest* the pointer where the read data should go to

*src* the address of the sector to be read from

References address::cyl, get\_hdsiz, hd\_read\_sector, HDALTBASE, HDALTREG\_STAT, HDBASE, HDCMD\_READ, HDREG\_CMD, HDREG\_COUNT, HDREG\_CYL\_HIGH, HDREG\_CYL\_LOW, HDREG\_DATA, HDREG\_SEC, HDSTATE\_NOTREADY, address::head, inb(), itoaddr(), outb(), panic, repinsw(), address::sector, select\_masterdrive(), and wait\_on\_hd\_interrupt().

#### 5.47.3.4 void hd\_write\_sector ( uint32 *dest*, void \* *src* )

Writes a sector to the hard disk.

##### Parameters

*dest* the address of the sector to be written to

*\*src* the pointer where data should be written from

The EEEPC needs a cache flush after a write operation.

References address::cyl, get\_hdsiz, HDALTBASE, HDALTREG\_STAT, HDBASE, HDCMD\_FLUSH\_CACHE, HDCMD\_WRITE, HDREG\_CMD, HDREG\_COUNT, HDREG\_CYL\_HIGH, HDREG\_CYL\_LOW, HDREG\_DATA, HDREG\_SEC, HDSTATE\_NOTREADY, address::head, inb(), itoaddr(), outb(), panic, repoutsw(), address::sector, select\_masterdrive(), and wait\_on\_hd\_interrupt().

#### 5.47.3.5 void monitor\_cputc ( char *ch*, uint8 *fg*, uint8 *bg* )

Writes a colored character to the display.

##### Parameters

*ch* character to be written

*fg* foreground-color

*bg* background color

References monitor\_cputc(), monitor\_invert(), sleep\_ticks(), and VGA\_DISPLAY.

Referenced by monitor\_cputc(), monitor\_cputs(), monitor\_putc(), monitor\_puthex(), and monitor\_puti().

**5.47.3.6 void monitor\_cputs ( char \* *str*, uint8 *fg*, uint8 *bg* )**

Writes a colored null-terminated string to the display.

**Parameters**

*\*str* pointer to the string

*fg* foreground-color

*bg* background color

References monitor\_cputc().

Referenced by draw\_test(), monitor\_puts(), panic(), and update\_virt\_monitor().

**5.47.3.7 void monitor\_putc ( char *ch* )**

Writes a character to the display.

**Parameters**

*ch* character to be written

References BLACK, monitor\_cputc(), and WHITE.

Referenced by monitor\_puti().

**5.47.3.8 void monitor\_puthex ( uint8 *ch* )**

Writes a hex-byte to the display.

**Parameters**

*ch* character to be written

References BLACK, GREEN, and monitor\_cputc().

Referenced by draw\_test().

**5.47.3.9 void monitor\_puti ( sint32 *x* )**

Writes an integer to the display.

**Parameters**

*x* integer to be written

References BLACK, monitor\_cputc(), monitor\_putc(), and RED.

Referenced by draw\_test().

**5.47.3.10 void monitor\_puts ( char \* *str* )**

Writes a null-terminated string to the display.

**Parameters**

*\*str* pointer to the string

References BLACK, monitor\_cputs(), and WHITE.

Referenced by draw\_test(), and grubstruct\_test().

**5.47.3.11 void monitor\_scrollup ( )****5.47.3.12 void monitor\_scrollup ( )****5.47.3.13 void printf ( char \* *fmt*, ... )**

Prints formatted output.

The following format specifiers are supported: %% - prints the % character. i - prints a signed integer. d - prints a signed integer. u - prints an unsigned integer. b - prints an unsigned integer in binary format (base 2) o - prints an unsigned integer in octal format (base 8). x - prints an unsigned integer in hexadecimal format (base 16). c - prints a single character. s - prints a string. "(null)" if argument is NULL. p - prints a pointer (base 16). %{ - prints the string until } colored All other format specifiers are ignored.

**Parameters**

*fmt* format string

... variable number of arguments

References puts, va\_end, va\_start, and vsnprintf.

**5.47.3.14 int putchar ( char *c* )**

Writes a character to stdout.

**Parameters**

*c* character to write

**Returns**

the character written

References SELECT\_VMONITOR, and virt\_monitor\_putc().

**5.47.3.15 int puts ( char \* *s* )**

Writes a string to stdout.

**Bug**

Does not write a terminating newline character. A change in behavior breaks other things (namely [printf\(\)](#)). Please leave it like that for now.

**Parameters**

*s* the string

**Returns**

the number of characters written

References BLACK, SELECT\_VMONITOR, virt\_monitor\_cputs(), and WHITE.

**5.47.3.16 int snprintf ( char \* buf, int size, char \* fmt, ... )**

References va\_end, va\_start, and vsnprintf.

**5.47.3.17 int vsnprintf ( char \* s, int n, char \* format, va\_list ap )**

References itoa, NULL, and va\_arg.

**5.47.4 Variable Documentation****5.47.4.1 uint32 maxaddr****5.48 src/kernel/include/stdlib.h File Reference**

includes some general help functions for type conversion, memory allocation, process control and other purposes.

```
#include "types.h"
```

**Functions**

- void [srand](#) (int seed)
- int [rand](#) ()
- void \* [malloc](#) (uint32 size)
- void \* [mallocn](#) (uint32 size, char \*name)
- void \* [calloc](#) (size\_t elements, size\_t size)
 

*Allocates space for n elements of the same size.*
- void \* [callocn](#) (size\_t elements, size\_t size, char \*name)
 

*Allocates space for n elements of the same size and additionally saves a name in the header of the block.*
- void [free](#) (void \*start)
 

*Frees a memory block.*
- void \* [realloc](#) (void \*pointer, size\_t size)
 

*Reallocates a memory block to 'size' bytes.*
- void [mem\\_dump](#) ()
- [uint32 free\\_memory](#) ()
 

*Function to return the free memory space.*

### 5.48.1 Detailed Description

includes some general help functions for type conversion, memory allocation, process control and other purposes.

**Author**

Johannes Schamburger

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

### 5.48.2 Function Documentation

#### 5.48.2.1 void\* calloc ( size\_t *n*, size\_t *size* )

Allocates space for *n* elements of the same size.

**Parameters**

*n* number of elements  
*size* size of each element

References calloc().

#### 5.48.2.2 void\* callocn ( size\_t *n*, size\_t *size*, char \* *name* )

Allocates space for *n* elements of the same size and additionally saves a name in the header of the block.

**Parameters**

*n* number of elements  
*size* size of each element  
*name* name of the memory block (mainly for debugging purposes)

References bzero, mallocn, and NULL.

Referenced by calloc(), init\_vmonitors(), and malloc\_test().

#### 5.48.2.3 void free ( void \* *start* )

Frees a memory block.

**Parameters**

*start* pointer to the start of the block that shall be freed

References heap\_free(), and kernel\_heap.



#### 5.48.2.4 uint32 free\_memory ( )

Function to return the free memory space.

##### Returns

free memory space in bytes

#### 5.48.2.5 void\* malloc ( uint32 size )

Referenced by `hd_test()`, `main()`, `potatoes_disc_create()`, `potatoes_malloc()`, `potatoes_mallocn()`, and `shell_cmd_synth()`.

#### 5.48.2.6 void\* mallocn ( uint32 size, char \* name )

#### 5.48.2.7 void mem\_dump ( )

References `heap_mem_dump()`.

#### 5.48.2.8 int rand ( )

Referenced by `snake()`.

#### 5.48.2.9 void\* realloc ( void \* pointer, size\_t size )

Reallocates a memory block to 'size' bytes.

ATTENTION: if `realloc` fails (i.e. there is not enough free space), the return value is (void\*) NULL. So this should always be tested! Especially `realloc()` shouldn't be used like this:

```
int* p = malloc(50);
p = realloc(100);
```

In this case, if `realloc` fails, `p` is overwritten by (void\*) NULL. So, the memory allocated for `p` is no longer accessible, which means that the data stored in `p` is lost and the memory allocated for `p` can't be used any more.

##### Parameters

*pointer* pointer to the old allocated space

*size* the new size

##### Returns

pointer to the reallocated space

References `free`, `mallocn`, `memmove()`, `mm_header::name`, `mm_header::next`, `NULL`, and `mm_header::size`.

Referenced by `malloc_test()`.

### 5.48.2.10 void srand ( int seed )

Referenced by snake().

## 5.49 src/kernel/include/string.h File Reference

Headers for [string.c](#).

```
#include "types.h"
```

### Functions

- [uint32 strlen](#) (char \*str)  
*Returns the length of a null-terminated string.*
- char \* [strcpy](#) (char \*dest, char \*src)  
*Copies string src to string dest (including terminating \0 character).*
- char \* [strncpy](#) (char \*dest, char \*src, [size\\_t](#) n)  
*Copies at most n characters from src to dest.*
- char \* [strchr](#) (char \*str, char ch)  
*Locates the first occurrence of ch in the string pointed to by str.*
- char \* [strcat](#) (char \*s1, char \*s2)  
*Concatenates two strings by appending a copy of s2 to the end of s1.*
- char \* [strncat](#) (char \*s1, char \*s2, [size\\_t](#) n)  
*Concatenates two strings by appending a copy of s2 to the end of s1.*
- char \* [strdup](#) (char \*str)  
*Duplicates a string.*
- char \* [strsep](#) (char \*\*str\_ptr, char \*delims)  
*Tokenizes a string.*
- [sint32 strcmp](#) (char \*s1, char \*s2)  
*Compares two strings.*
- int [isspace](#) (char c)  
*Test if a character represents whitespace.*
- void \* [memset](#) (void \*dest, [uint8](#) value, [size\\_t](#) count)  
*Writes count bytes of value value to the memory referenced by dest.*
- void [bzero](#) (void \*dest, [uint32](#) count)
- void \* [memcpy](#) (void \*dest, void \*src, [size\\_t](#) count)  
*Copies count bytes from src to dest.*

- void \* [memmove](#) (void \*dest, void \*src, [size\\_t](#) count)  
*Copies count bytes from src to dest.*
- char \* [strreverse](#) (char \*str)  
*Reverses a string in place.*
- char \* [itoa](#) (int n, char \*str, unsigned int [base](#))  
*Converts an integer into a string using an arbitrary base.*
- int [strtol](#) (char \*nptr, char \*\*endptr, int [base](#))  
*Converts a string into an integer.*
- int [atoi](#) (char \*str)  
*Converts a string into an integer (base 10 is assumed).*

### 5.49.1 Detailed Description

Headers for [string.c](#).

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.49.2 Function Documentation

#### 5.49.2.1 int atoi ( char \* str )

Converts a string into an integer (base 10 is assumed).

This function will skip whitespace and read a (possibly) signed number until it reaches a char that can't be part of the number.

#### Parameters

*str* the source string buffer

#### Returns

The conversion result

References [NULL](#), and [strtol\(\)](#).

Referenced by [atoi\\_test\(\)](#), [shell\\_cmd\\_kill\(\)](#), and [shell\\_cmd\\_nice\(\)](#).

**5.49.2.2 void bzero ( void \* *dest*, uint32 *count* )****5.49.2.3 int isspace ( char *c* )**

Test if a character represents whitespace.

**Parameters**

*c* the character to be tested

**Returns**

whether the character is to be considered whitespace

For the purposes of this function space, horizontal tab, newline, carriage return, form feed and vertical tab are considered whitespace.

Referenced by strtol().

**5.49.2.4 char\* itoa ( int *n*, char \* *str*, unsigned int *base* )**

Converts an integer into a string using an arbitrary base.

Make sure the buffer for the converted string is large enough. The smaller the base the more space is required, eg for converting a 32 bit integer you can expect 35 bytes to be sufficient (base 2: 32 bytes for digits + 1 byte for the terminator + some space just to feel safe and take into account future format changes). In other, GAD inspired words:  $\text{bufsize} \geq \text{ceil}(\log_2(\text{MAX\_INT}))$  which is equal to:  $\text{bufsize} \geq \text{sizeof}(\text{int}) * 8$

This is not a part of the C standard library.

**Parameters**

*n* the number to convert

*str* the target string buffer

*base* the base to use for the conversion

**Returns**

The original value of *str*

References num, and streverse().

**5.49.2.5 void\* memcpy ( void \* *dest*, void \* *src*, size\_t *count* )**

Copies *count* bytes from *src* to *dest*.

Undefined behaviour when *src* and *dest* are overlapping (use [memmove\(\)](#) instead).

**See also**

[memmove](#)

**Parameters**

*dest* Destination memory

*src* Source memory  
*count* Number of bytes to copy

### Returns

dest, the destination memory

#### 5.49.2.6 void\* memmove ( void \* dest, void \* src, size\_t count )

Copies count bytes from src to dest.

Unlike [memcpy\(\)](#), [memmove\(\)](#) supports copy operations where the blocks referenced by src and dest are overlapping. Should you not need such functionality, use [memcpy\(\)](#) for performance reasons.

### Bug

There has to be a better way, ie one that does not use dynamic memory.

### See also

[memcpy](#)

### Parameters

*dest* Destination memory  
*src* Source memory  
*count* Number of bytes to copy

### Returns

dest, the destination memory

References free, mallocn, and memcpy.

Referenced by realloc().

#### 5.49.2.7 void\* memset ( void \* dest, uint8 value, size\_t count )

Writes count bytes of value value to the memory referenced by dest.

### Parameters

*dest* Destination memory  
*value* Value dest is filled with  
*count* Number of bytes to write

### Returns

dest, the destination memory

Referenced by bzero(), dev\_null\_read(), free\_virt\_monitor(), get\_page(), io\_init(), memview\_main(), pm\_init(), shell\_autocomplete(), shell\_cmd\_clear(), shell\_cmd\_pong(), shell\_main(), snake(), snapshot(), start\_vmonitor(), syscall\_test\_thread(), and update\_view().

#### 5.49.2.8 char\* strcat ( char \* *s1*, char \* *s2* )

Concatenates two strings by appending a copy of *s2* to the end of *s1*.

##### Parameters

*s1* the "head"

*s2* the "tail"

##### Returns

the concatenated string

References strchr(), and strcpy().

#### 5.49.2.9 char\* strchr ( char \* *str*, char *ch* )

Locates the first occurrence of *ch* in the string pointed to by *str*.

The terminating \0 character is considered to be part of the string.

##### Parameters

*str* the string to search

*ch* the character to look for

##### Returns

the first occurrence of *ch* in *str* or NULL if not found

Referenced by strcat(), strncat(), and strsep().

#### 5.49.2.10 sint32 strcmp ( char \* *s1*, char \* *s2* )

Compares two strings.

##### Parameters

*s1* String

*s2* String

##### Returns

0 if both strings are equal, >0 if *s1* greater than *s2*, <0 if *s1* is less than *s2*

#### 5.49.2.11 char\* strcpy ( char \* *dest*, char \* *src* )

Copies string *src* to string *dest* (including terminating \0 character).

For stability and security reasons, try to use [strncpy\(\)](#) whenever possible.

##### See also

[strncpy](#)

**Parameters**

*dest* Destination string

*src* Source string

**Returns**

*dest*, the destination string

Referenced by `new_shell()`, `potatoes_time2str()`, `shell_cmd_cd()`, `shell_cmd_snake()`, `shell_main()`, `shell_-makepath()`, `snapshot()`, `strcat()`, `strdup()`, and `time2str()`.

**5.49.2.12 char\* strdup ( char \* str )**

Duplicates a string.

`strdup()` allocates sufficient memory for a copy of the string *str*, copies it and returns a pointer to the copied string. Strings returned by `strdup()` must be released by calling `free()`.

**Parameters**

*str* the string to duplicate

**Returns**

The pointer to the duplicated string or NULL on error

References `mallocn`, `NULL`, `strcpy()`, and `strlen`.

**5.49.2.13 uint32 strlen ( char \* str )**

Returns the length of a null-terminated string.

**Parameters**

*str* String to check

**Returns**

Number of characters preceding the terminating null character.

**5.49.2.14 char\* strncat ( char \* s1, char \* s2, size\_t n )**

Concatenates two strings by appending a copy of *s2* to the end of *s1*.

Not more than *n* characters are copied from *s2*.

**Parameters**

*s1* the "head"

*s2* the "tail"

*n* max number of characters to copy

**Returns**

the concatenated string

References `strchr()`, and `strncpy`.

Referenced by `new_shell()`, `potatoes_strncat()`, and `strings_test()`.

**5.49.2.15 char\* strncpy ( char \* *dest*, char \* *src*, size\_t *n* )**

Copies at most *n* characters from *src* to *dest*.

If *src* is less than *n* characters long, the remainder of *dest* is filled with `\0` characters. Otherwise, *dest* is not terminated.

**Parameters**

*dest* Destination string

*src* Source string

*n* Number of bytes to copy at most

**Returns**

*dest*, the destination string

**5.49.2.16 char\* strreverse ( char \* *str* )**

Reverses a string in place.

This is not a part of the C standard library.

**Parameters**

*str* the string to reverse

**Returns**

the reversed string

References `end`, `start`, and `strlen`.

Referenced by `itoa()`.

**5.49.2.17 char\* strsep ( char \*\* *str\_ptr*, char \* *delims* )**

Tokenizes a string.

Take note that `strsep()` will manipulate both the string pointer `**str_ptr` points at as well as the contents of the respective string.

Example:

```
char path[] = "/usr/share/bin/editor";
char delim[] = "/";
char *tok;
char *copy = strdup(path);
char *work_copy = copy;
```



```
do {
    printf("strsep(\"%s\") ", work_copy);
    tok = strsep(&work_copy, delim);
    printf("-> \"%s\"\n", tok);
} while (tok != NULL);

printf("\ncopy = %p\n", copy);
printf("work_copy = %p\n", work_copy);
puts("done.");

free(copy);
```

### Bug

The current implementation does not handle multiple delimiters (as specified in the libc manual). Only the first character in *\*delims* is used for tokenizing the input string.

### Parameters

*str\_ptr* Pointer to string to tokenize

*delims* String containing all delimiter characters

### Returns

The next token or NULL if the end of the input string was reached

References NULL, and strchr().

#### 5.49.2.18 int strtol ( char \* *nptr*, char \*\* *endptr*, int *base* )

Converts a string into an integer.

Bases in the range from 2 to 36 are handled, with a-z and A-Z being treated as the digits with values 10 to 35.

If *base* is 0, try to guess the base, if the string starts with "0x" it will be treated as base 16, if it starts with "0" it will be treated as base 8, otherwise base 10 is assumed.

No range check is performed, so if the value is greater than or equal to  $2^{31}-1$  or less than  $-2^{31}$  the results will be undefined.

### Parameters

*str* the source string buffer

*endptr* if this not equal to NULL, the address of the first character that was not parsed is written to this address

*base* the base to be used for conversion

### Returns

the result of the conversion

References isspace(), and NULL.

Referenced by atoi(), and atoi\_test().

## 5.50 src/kernel/include/types.h File Reference

Basic type definitions.

### Defines

- #define [MIN](#)(a, b) (a < b ? a : b)
- #define [MAX](#)(a, b) (a > b ? a : b)

### Typedefs

- typedef unsigned char [uint8](#)
- typedef signed char [sint8](#)
- typedef unsigned short [uint16](#)
- typedef signed short [sint16](#)
- typedef unsigned int [uint32](#)
- typedef signed int [sint32](#)
- typedef float [float32](#)
- typedef double [float64](#)
- typedef [uint8](#) [bool](#)
- typedef [uint32](#) [size\\_t](#)
- typedef struct [time](#) [time\\_t](#)

### 5.50.1 Detailed Description

Basic type definitions. This file defines basic data types used throughout the kernel. You should use these types whenever possible in order to avoid the possible ambiguity of the builtin types.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.50.2 Define Documentation

#### 5.50.2.1 #define [MAX](#)( *a*, *b* ) (a > b ? a : b)

Referenced by [fs\\_truncate\(\)](#).

5.50.2.2 `#define MIN( a, b ) (a < b ? a : b)`

### 5.50.3 Typedef Documentation

5.50.3.1 `typedef uint8 bool`

5.50.3.2 `typedef float float32`

5.50.3.3 `typedef double float64`

5.50.3.4 `typedef signed short sint16`

5.50.3.5 `typedef signed int sint32`

5.50.3.6 `typedef signed char sint8`

5.50.3.7 `typedef uint32 size_t`

5.50.3.8 `typedef struct time time_t`

5.50.3.9 `typedef unsigned short uint16`

5.50.3.10 `typedef unsigned int uint32`

5.50.3.11 `typedef unsigned char uint8`

## 5.51 src/kernel/include/util.h File Reference

Useful function headers.

```
#include "types.h"
```

### Functions

- void `sleep` (`sint32` sec)  
*Sleeps num seconds.*
- void `sleep_ticks` (`sint32` ticks)  
*Sleeps num ticks.*

### 5.51.1 Detailed Description

Useful function headers.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

## Version

### Rev:

12

## 5.51.2 Function Documentation

### 5.51.2.1 void sleep ( sint32 sec )

Sleeps num seconds.

References FREQUENCY, halt(), reactivate\_pic(), set\_interrupts(), SINT32\_MAX, and sleep().

Referenced by sleep(), sleep\_test(), and sleep\_ticks().

### 5.51.2.2 void sleep\_ticks ( sint32 ticks )

Sleeps num ticks.

References halt(), reactivate\_pic(), set\_interrupts(), SINT32\_MAX, and sleep().

Referenced by monitor\_cputc().

## 5.52 src/kernel/init/main.c File Reference

Main kernel file.

```
#include "../include/init.h"
#include "../include/const.h"
#include "../include/stdio.h"
#include "../include/debug.h"
#include "../../apps/shell_main.h"
#include "../io/io.h"
#include "../pm/pm_main.h"
#include "../mm/mm.h"
#include "../io/io_virtual.h"
```

## Functions

- void [panic](#) (char \*msg)  
*Kernel panic function.*
- void [shell\\_main](#) ()  
*The shell entry point and main loop.*
- int [main](#) (struct [multiboot](#) \*mboot\_ptr)

*C kernel entry point.*

## Variables

- struct `multiboot` \* `g_mboot_ptr`  
*Global pointer to multiboot structure As of now, this is only needed for `mboot_test()` in `tests.c`.*
- int `end`  
*The address of `end` is equal to the end of kernel code in memory + 1.*
- int `start`  
*The address of `start` is equal to the start address of the kernel code in memory.*

### 5.52.1 Detailed Description

Main kernel file. Includes the C code entry point.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.52.2 Function Documentation

#### 5.52.2.1 `int main ( struct multiboot * mboot_ptr )`

C kernel entry point.

#### Parameters

*mboot\_ptr* The multiboot struct passed by the bootloader (grub).

References `do_tests()`, `dprint_separator`, `end`, `fs_init()`, `get_ticks()`, `halt()`, `init_vmonitors()`, `io_init()`, `multiboot::mem_upper`, `mm_init()`, `mm_init_output()`, `new_shell()`, `pm_init()`, and `printf`.

#### 5.52.2.2 `void panic ( char * msg )`

Kernel panic function.

Displays an error message and enters an infinite loop thus effectively halting the kernel. This should only be called when a non-recoverable ("fatal") error occurs inside the kernel.

**Parameters**

*msg* Message to display.

References BLACK, clear\_interrupts(), halt(), monitor\_cputs(), and RED.

**5.52.2.3 void shell\_main ( )**

The shell entry point and main loop.

This is where the shell prompt gets displayed and user typed commands are dispatched.

References \_close(), \_exit(), \_fgets(), \_open(), \_printf(), shell\_cmd\_t::cmd, cwd, memset(), shell\_autocomplete(), shell\_handle\_command(), STDIN, STDOUT, strcpy(), and strlen.

Referenced by new\_shell().

**5.52.3 Variable Documentation****5.52.3.1 int end**

The address of end is equal to the end of kernel code in memory + 1.

This constant gets defined in the linker script link.ld.

Usage example:

```
mem_start = (uint32) &end;
```

Referenced by interpret\_bf(), main(), and strreverse().

**5.52.3.2 struct multiboot\* g\_mboot\_ptr**

Global pointer to multiboot structure As of now, this is only needed for mboot\_test() in tests.c.

Global pointer to multiboot structure.

This should later be removed and only used in the kernel main().

**5.52.3.3 int start**

The address of start is equal to the start address of the kernel code in memory.

Referenced by \_fgets(), interpret\_bf(), and strreverse().

**5.53 src/kernel/init/tests.c File Reference**

Basic difinitions for functions used in the main()-function of the kernel.

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/assert.h"
#include "../include/string.h"
```

```
#include "../include/stdio.h"
#include "../include/stdlib.h"
#include "../include/init.h"
#include "../include/util.h"
#include "../include/debug.h"
#include "../../apps/shell_main.h"
#include "../io/io_virtual.h"
#include "../io/io.h"
#include "../io/io_harddisk.h"
#include "../io/io_rtc.h"
#include "../mm/mm.h"
#include "../mm/mm_paging.h"
#include "../pm/pm_main.h"
#include "../pm/syscalls_cli.h"
#include "../pm/syscalls_shared.h"
```

## Data Structures

- struct [Semaphore](#)

## Defines

- #define [MBOOT\\_INFO](#)(x) printf("\t%s = %u\n", #x, mboot\_ptr->x);

## Functions

- void [draw\\_test](#) ()  
*output-testing*
- void [printf\\_test](#) ()
- void [strings\\_test](#) ()
- void [strsep\\_test](#) ()
- void [assert\\_test](#) ()
- void [grubstruct\\_test](#) (struct [multiboot](#) \*mboot\_ptr)
- void [malloc\\_test](#) ()
- void [mm\\_pagefault\\_test](#) ()
- void [sleep\\_test](#) ()
- void [hd\\_test](#) ()  
*Memcopy using hd ;-).*
- void [hd\\_write\\_test](#) ()
- void [hd\\_stresswrite\\_test](#) ()
- void [hd\\_stressread\\_test](#) ()
- int [fgetch](#) (int fd)

- char \* [fgets](#) (char \*s, int n, int fd)
- void [syscall\\_test\\_thread](#) ()
- void [syscall\\_test](#) ()
- void [ralph\\_wiggum](#) ()
- void [run\\_FS\\_tests](#) ()
- void [fs\\_tests](#) ()
- void [isr\\_test](#) ()
- void [threadA](#) ()
- void [threadB](#) ()
- void [threadA\\_test](#) ()
- void [threadB\\_test](#) ()
- void [threadC](#) ()
- void [threadD](#) ()
- void [threadC\\_test](#) ()
- void [threadD\\_test](#) ()
- void [sema\\_init](#) (struct [Semaphore](#) \*semaphore, int count)
- void [p](#) (struct [Semaphore](#) \*semaphore)
- void [v](#) (struct [Semaphore](#) \*semaphore)
- void [threadProducer](#) ()
- void [threadConsumer](#) ()
- void [threadConsumer\\_test](#) ()
- void [threadProducer\\_test](#) ()
- void [nullptr\\_test](#) ()
- void [print\\_time](#) ()
- void [test\\_batch\\_files](#) ()
- void [atoi\\_test](#) ()
- bool [create\\_fs](#) ()

*Creates a new file system.*

- void [new\\_fs](#) ()
- void [reboot](#) ()
- void [threadFabrik](#) ()
- void [threadMitarbeiter](#) ()
- void [threadLastwagen1](#) ()
- void [threadLastwagen2](#) ()
- void [threadGeschaeft1](#) ()
- void [threadGeschaeft2](#) ()
- void [paralleleModellierung](#) ()
- void [make\\_snapshot](#) ()
- void [do\\_tests](#) ()

## Variables

- int [mutex](#) = 0
- struct [Semaphore](#) \* [sema\\_free](#)
- struct [Semaphore](#) \* [sema\\_full](#)
- struct [Semaphore](#) \* [sema\\_access](#)
- int [elements](#) = 0
- int [betten](#) = 0
- int [schraenke](#) = 0



- struct Semaphore \* lager\_betten
- struct Semaphore \* lager\_schraenke
- struct Semaphore \* lager\_free
- struct Semaphore \* lager\_access
- int rbetten = 0
- int rschraenke = 0
- struct Semaphore \* rampe\_betten
- struct Semaphore \* rampe\_schraenke
- struct Semaphore \* rampe\_free
- struct Semaphore \* rampe\_access
- int g1betten = 0
- struct Semaphore \* g1\_full
- struct Semaphore \* g1\_free
- struct Semaphore \* g1\_access
- int g2betten = 0
- int g2schraenke = 0
- struct Semaphore \* g2b\_full
- struct Semaphore \* g2b\_free
- struct Semaphore \* g2s\_full
- struct Semaphore \* g2s\_free
- struct Semaphore \* g2\_access

### 5.53.1 Detailed Description

Basic difinitions for functions used in the `main()`-function of the kernel.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

21

### 5.53.2 Define Documentation

#### 5.53.2.1 #define MBOOT\_INFO( x ) printf("\t%s = %u\n", #x, mboot\_ptr->x);

Referenced by `grubstruct_test()`.

### 5.53.3 Function Documentation

#### 5.53.3.1 void assert\_test ( )

References `ASSERT`, `FALSE`, and `TRUE`.

### 5.53.3.2 void atoi\_test ( )

References atoi(), dprintf, and strtol().

Referenced by do\_tests().

### 5.53.3.3 bool create\_fs ( )

Creates a new file system.

References create\_root(), dump\_super(), init\_bmap(), init\_file\_table(), init\_inode\_table(), init\_super\_block(), printf, and write\_super\_block().

Referenced by fs\_init(), and new\_fs().

### 5.53.3.4 void do\_tests ( )

References \_malloc(), atoi\_test(), dprint\_separator, make\_snapshot(), new\_shell(), paralleleModellierung(), printf, ralph\_wiggum(), reboot(), sema\_init(), SHORTCUT\_CTRL, SHORTCUT\_CTRL\_SUPER, switch\_monitor\_down(), switch\_monitor\_up(), test\_batch\_files(), threadA\_test(), threadB\_test(), threadC\_test(), threadConsumer\_test(), threadD\_test(), and threadProducer\_test().

Referenced by main().

### 5.53.3.5 void draw\_test ( )

output-testing

References BLACK, BLUE, CYAN, DARKGREY, GREEN, LIGHTBLUE, LIGHTGREEN, LIGHTGREY, MAGENTA, monitor\_cputs(), monitor\_puthex(), monitor\_puti(), monitor\_puts(), ORANGE, PINK, RED, strlen, TURQUOISE, VIOLET, WHITE, and YELLOW.

### 5.53.3.6 int fgetch ( int *fd* )

References \_read(), and halt().

Referenced by fgets().

### 5.53.3.7 char\* fgets ( char \* *s*, int *n*, int *fd* )

References fgets().

Referenced by syscall\_test\_thread().

### 5.53.3.8 void fs\_tests ( )

References run\_FS\_tests().

### 5.53.3.9 void grubstruct\_test ( struct multiboot \* *mboot\_ptr* )

References addr, flags, MBOOT\_INFO, mem\_lower, mem\_upper, monitor\_puts(), printf, and size.

**5.53.3.10 void hd\_stressread\_test ( )**

References ASSERT, free, get\_hdsizes, hd\_read\_sector, malloc, and printf.

**5.53.3.11 void hd\_stresswrite\_test ( )**

References ASSERT, free, get\_hdsizes, hd\_write\_sector, malloc, and printf.

**5.53.3.12 void hd\_test ( )**

Memcopy using hd ;-).

References hd\_read\_sector, hd\_write\_sector, malloc(), and puts.

**5.53.3.13 void hd\_write\_test ( )**

References ASSERT, free, hd\_write\_sector, malloc, and printf.

**5.53.3.14 void isr\_test ( )**

References printf.

**5.53.3.15 void make\_snapshot ( )**

References memcpy, pm\_create\_thread(), snap\_buffer, snapshot(), and VGA\_DISPLAY.

Referenced by do\_tests().

**5.53.3.16 void malloc\_test ( )**

References calloc(), heap\_t::end, free, heap\_mem\_dump(), kernel\_heap, malloc, mem\_dump, NULL, printf, realloc(), and heap\_t::start.

**5.53.3.17 void mm\_pagefault\_test ( )****5.53.3.18 void new\_fs ( )**

References create\_fs().

**5.53.3.19 void nullptr\_test ( )**

References putchar.

**5.53.3.20 void p ( struct Semaphore \* semaphore )**

References Semaphore::count, halt(), Semaphore::lock, mutex\_lock(), and mutex\_unlock().

Referenced by `pm_dump()`, `pm_get_proc()`, `pm_kill_proc()`, `pm_set_thread_priority()`, `threadConsumer()`, `threadFabrik()`, `threadGeschaeft1()`, `threadGeschaeft2()`, `threadLastwagen1()`, `threadLastwagen2()`, `threadMitarbeiter()`, and `threadProducer()`.

#### 5.53.3.21 void `paralleleModellierung` ( )

References `_malloc()`, `pm_create_thread()`, `sema_init()`, `threadFabrik()`, `threadGeschaeft1()`, `threadGeschaeft2()`, `threadLastwagen1()`, `threadLastwagen2()`, and `threadMitarbeiter()`.

Referenced by `do_tests()`.

#### 5.53.3.22 void `print_time` ( )

References `free`, `mallocn`, `printf`, and `time2str`.

#### 5.53.3.23 void `printf_test` ( )

References `printf`.

#### 5.53.3.24 void `ralph_wiggum` ( )

References `printf`.

Referenced by `do_tests()`.

#### 5.53.3.25 void `reboot` ( )

References `outb()`.

Referenced by `do_tests()`.

#### 5.53.3.26 void `run_FS_tests` ( )

References `test_open_close()`.

Referenced by `fs_tests()`.

#### 5.53.3.27 void `sema_init` ( `struct Semaphore * semaphore`, `int count` )

References `Semaphore::count`, and `Semaphore::lock`.

Referenced by `do_tests()`, and `paralleleModellierung()`.

#### 5.53.3.28 void `sleep_test` ( )

References `printf`, and `sleep()`.

#### 5.53.3.29 void `strings_test` ( )

References `bzero`, `printf`, `strcat`, and `strncat()`.

**5.53.3.30 void strsep\_test ( )**

References free, NULL, printf, strdup, and strsep.

**5.53.3.31 void syscall\_test ( )**

References pm\_create\_thread(), and syscall\_test\_thread().

**5.53.3.32 void syscall\_test\_thread ( )**

References \_close(), \_exit(), \_free(), \_getpid(), \_log(), \_malloc(), \_open(), \_read(), \_seek(), \_write(), fgets(), memset(), printf, SEEK\_CUR, and strlen.

Referenced by syscall\_test().

**5.53.3.33 void test\_batch\_files ( )**

References \_close(), \_open(), \_write(), O\_CREAT, and strlen.

Referenced by do\_tests().

**5.53.3.34 void threadA ( )**

References \_free(), \_log(), \_malloc(), \_printf(), bcd2bin(), halt(), mutex, mutex\_lock(), mutex\_unlock(), and time2str.

Referenced by threadA\_test().

**5.53.3.35 void threadA\_test ( )**

References pm\_create\_thread(), and threadA().

Referenced by do\_tests().

**5.53.3.36 void threadB ( )**

References \_free(), \_log(), \_malloc(), \_printf(), bcd2bin(), halt(), mutex, mutex\_lock(), mutex\_unlock(), and time2str.

Referenced by threadB\_test().

**5.53.3.37 void threadB\_test ( )**

References pm\_create\_thread(), and threadB().

Referenced by do\_tests().

**5.53.3.38 void threadC ( )**

References \_log(), and halt().

Referenced by threadC\_test().

**5.53.3.39 void threadC\_test ( )**

References pm\_create\_thread(), and threadC().

Referenced by do\_tests().

**5.53.3.40 void threadConsumer ( )**

References \_printf(), elements, halt(), p(), and v().

Referenced by threadConsumer\_test().

**5.53.3.41 void threadConsumer\_test ( )**

References pm\_create\_thread(), and threadConsumer().

Referenced by do\_tests().

**5.53.3.42 void threadD ( )**

References \_log(), and halt().

Referenced by threadD\_test().

**5.53.3.43 void threadD\_test ( )**

References pm\_create\_thread(), pm\_set\_thread\_priority(), and threadD().

Referenced by do\_tests().

**5.53.3.44 void threadFabrik ( )**

References \_printf(), betten, halt(), p(), schraenke, and v().

Referenced by paralleleModellierung().

**5.53.3.45 void threadGeschaeft1 ( )**

References \_printf(), g1betten, halt(), p(), and v().

Referenced by paralleleModellierung().

**5.53.3.46 void threadGeschaeft2 ( )**

References \_printf(), g2betten, g2schraenke, halt(), p(), and v().

Referenced by paralleleModellierung().

**5.53.3.47 void threadLastwagen1 ( )**

References \_printf(), g1betten, halt(), p(), rbetten, rschraenke, and v().

Referenced by paralleleModellierung().

**5.53.3.48 void threadLastwagen2 ( )**

References `_printf()`, `g2betten`, `g2schraenke`, `halt()`, `p()`, `rbetten`, `rschraenke`, and `v()`.

Referenced by `paralleleModellierung()`.

**5.53.3.49 void threadMitarbeiter ( )**

References `_printf()`, `betten`, `halt()`, `p()`, `rbetten`, `rschraenke`, `schraenke`, and `v()`.

Referenced by `paralleleModellierung()`.

**5.53.3.50 void threadProducer ( )**

References `_printf()`, `elements`, `halt()`, `p()`, and `v()`.

Referenced by `threadProducer_test()`.

**5.53.3.51 void threadProducer\_test ( )**

References `pm_create_thread()`, and `threadProducer()`.

Referenced by `do_tests()`.

**5.53.3.52 void v ( struct Semaphore \* semaphore )**

References `Semaphore::count`, `Semaphore::lock`, `mutex_lock()`, and `mutex_unlock()`.

Referenced by `threadConsumer()`, `threadFabrik()`, `threadGeschaeft1()`, `threadGeschaeft2()`, `threadLastwagen1()`, `threadLastwagen2()`, `threadMitarbeiter()`, and `threadProducer()`.

**5.53.4 Variable Documentation****5.53.4.1 int betten = 0**

Referenced by `threadFabrik()`, and `threadMitarbeiter()`.

**5.53.4.2 int elements = 0**

Referenced by `threadConsumer()`, and `threadProducer()`.

**5.53.4.3 struct Semaphore\* g1\_access****5.53.4.4 struct Semaphore\* g1\_free****5.53.4.5 struct Semaphore\* g1\_full****5.53.4.6 int g1betten = 0**

Referenced by `threadGeschaeft1()`, and `threadLastwagen1()`.

**5.53.4.7 struct Semaphore\* g2\_access**

**5.53.4.8 struct Semaphore\* g2b\_free**

**5.53.4.9 struct Semaphore\* g2b\_full**

**5.53.4.10 int g2betten = 0**

Referenced by threadGeschaeft2(), and threadLastwagen2().

**5.53.4.11 struct Semaphore\* g2s\_free**

**5.53.4.12 struct Semaphore\* g2s\_full**

**5.53.4.13 int g2schraenke = 0**

Referenced by threadGeschaeft2(), and threadLastwagen2().

**5.53.4.14 struct Semaphore\* lager\_access**

**5.53.4.15 struct Semaphore\* lager\_betten**

**5.53.4.16 struct Semaphore\* lager\_free**

**5.53.4.17 struct Semaphore\* lager\_schraenke**

**5.53.4.18 int mutex = 0**

Referenced by threadA(), and threadB().

**5.53.4.19 struct Semaphore\* rampe\_access**

**5.53.4.20 struct Semaphore\* rampe\_betten**

**5.53.4.21 struct Semaphore\* rampe\_free**

**5.53.4.22 struct Semaphore\* rampe\_schraenke**

**5.53.4.23 int rbetten = 0**

Referenced by threadLastwagen1(), threadLastwagen2(), and threadMitarbeiter().

**5.53.4.24 int rschraenke = 0**

Referenced by threadLastwagen1(), threadLastwagen2(), and threadMitarbeiter().

**5.53.4.25 int schraenke = 0**

Referenced by threadFabrik(), and threadMitarbeiter().



5.53.4.26 struct Semaphore\* sema\_access

5.53.4.27 struct Semaphore\* sema\_free

5.53.4.28 struct Semaphore\* sema\_full

## 5.54 src/kernel/io/int\_handler.h File Reference

Header-file for specific hardware interrupt-handlers.

### Functions

- void [kb\\_handler](#) ()  
*Handles a keyboard interrupt by calling the PM (providing already the right character).*
- [uint32 timer\\_handler](#) (uint32 context)  
*Handles a timer-interrupt by incrementing the ticks-counter and calling the PM.*
- void [hd\\_handler](#) ()  
*Handles an hard disk interrupt by setting the hd\_interrupt flag.*

### 5.54.1 Detailed Description

Header-file for specific hardware interrupt-handlers.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.54.2 Function Documentation

#### 5.54.2.1 void [hd\\_handler](#) ( )

Handles an hard disk interrupt by setting the hd\_interrupt flag.

References [hd\\_interrupt](#), [HDBASE](#), [HDREG\\_STAT](#), [inb\(\)](#), and [panic](#).

Referenced by [irq\\_handler\(\)](#).

### 5.54.2.2 void kb\_handler ( )

Handles a keyboard interrupt by calling the PM (providing already the right character).

In echo-mode prints the typed character directly to the screen.

References alt, ALT, ctrl, CTRL, CURSOR\_DOWN, CURSOR\_LEFT, CURSOR\_RIGHT, CURSOR\_UP, echo, ESCAPE, shortcut::func, get\_active\_virt\_monitor(), inb(), kb\_alt\_map, kb\_map, KB\_PORT, kb\_shift\_map, KEY\_PRESSED, KEY\_RELEASED, keyboard\_state, LSHIFT, pm\_handle\_input(), RSHIFT, SCROLL\_DOWN, SCROLL\_UP, set\_interrupts(), shcut\_num, shift, super, SUPER, super\_button, virt\_cursor\_move(), virt\_monitor\_invert(), virt\_monitor\_putc(), virt\_monitor\_scrolldown(), and virt\_monitor\_scrollup().

Referenced by irq\_handler().

### 5.54.2.3 uint32 timer\_handler ( uint32 context )

Handles a timer-interrupt by incrementing the ticks-counter and calling the PM.

References get\_active\_virt\_monitor(), pm\_schedule(), rtc\_update(), SINT32\_MAX, and update\_virt\_monitor().

Referenced by irq\_handler().

## 5.55 src/kernel/io/int\_idt.c File Reference

Builds and initializes the interrupt descriptor table.

```
#include "../include/types.h"
```

### Data Structures

- struct [idt\\_entry](#)  
*The structure of one IDT entry.*
- struct [idt\\_pointer](#)  
*The structure of the IDT pointer which tells the processor where to find our IDT.*

### Functions

- struct [idt\\_entry \\_\\_attribute\\_\\_\(\(packed\)\)](#)  
*The structure of one IDT entry.*
- void [idt\\_fill\\_entry](#) (uint8 pos, uint32 offset, uint16 sel, uint8 flg)  
*Makes a new idt-entry.*
- void [idt\\_flush](#) (uint32)  
*Initializes a new idt with blank entries.*
- void [idt\\_init](#) ()

## Variables

- `uint16 low_offset`  
*Lower 16 bit of the interrupt handler's code address.*
- `uint16 selector`  
*Code segment selector in the GDT.*
- `uint8 separator`  
*Unused.*
- `uint8 flags`
- `uint16 high_offset`  
*Upper 16 bit of the interrupt handler's code address.*
- `uint16 maxsize`
- `uint32 start`  
*The address of start is equal to the start address of the kernel code in memory.*
- `struct idt_entry idt [256]`  
*Our idt with 256 descriptors.*
- `struct idt_pointer idtp`  
*The pointer to our idt.*

### 5.55.1 Detailed Description

Builds and initializes the interrupt descriptor table.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.55.2 Function Documentation

#### 5.55.2.1 `struct idt_entry __attribute__((packed))`

The structure of one IDT entry.

The structure of the IDT pointer which tells the processor where to find our IDT.

### 5.55.2.2 void idt\_fill\_entry ( uint8 pos, uint32 offset, uint16 sel, uint8 flg )

Makes a new idt-entry.

#### Parameters

- pos* Number of the new idt-entry
- offset* Offset to be added to the base address
- sel* Selector for the segment's base address
- flg* Deskriptor flags

References `idt_entry::flags`, `idt_entry::high_offset`, `idt`, `idt_entry::low_offset`, `idt_entry::selector`, and `idt_entry::separator`.

Referenced by `idt_init()`, `irq_init()`, and `isr_init()`.

### 5.55.2.3 void idt\_flush ( uint32 )

Initializes a new idt with blank entries.

Referenced by `idt_init()`.

### 5.55.2.4 void idt\_init ( )

References `idt`, `idt_fill_entry()`, `idt_flush()`, `idtp`, `idt_pointer::maxsize`, and `idt_pointer::start`.

Referenced by `io_init()`.

## 5.55.3 Variable Documentation

### 5.55.3.1 uint8 flags

### 5.55.3.2 uint16 high\_offset

Upper 16 bit of the interrupt handler's code address.

### 5.55.3.3 struct idt\_entry idt[256]

Our idt with 256 descriptors.

Referenced by `idt_fill_entry()`, and `idt_init()`.

### 5.55.3.4 struct idt\_pointer idtp

The pointer to our idt.

Referenced by `idt_init()`.

### 5.55.3.5 uint16 low\_offset

Lower 16 bit of the interrupt handler's code address.

### 5.55.3.6 uint16 maxsize

### 5.55.3.7 uint16 selector

Code segment selector in the GDT.

### 5.55.3.8 uint8 separator

Unused.

### 5.55.3.9 uint32 start

The address of start is equal to the start address of the kernel code in memory.

Referenced by `_fgets()`, `interpret_bf()`, and `strreverse()`.

## 5.56 src/kernel/io/int\_irq.c File Reference

Handler for hardware-interrupts.

```
#include "../include/types.h"
#include "../include/stdio.h"
#include "../io/int_handler.h"
#include "../pm/pm_main.h"
```

### Functions

- void `irq0` ()
- void `irq1` ()
- void `irq2` ()
- void `irq3` ()
- void `irq4` ()
- void `irq5` ()
- void `irq6` ()
- void `irq7` ()
- void `irq8` ()
- void `irq9` ()
- void `irq10` ()
- void `irq11` ()
- void `irq12` ()
- void `irq13` ()
- void `irq14` ()
- void `irq15` ()
- void `idt_fill_entry` (`uint8` pos, `uint32` offset, `uint16` sel, `uint8` flg)  
*Makes a new idt-entry.*
- void `outb` (`uint8` port, `uint8` value)
- void `pic_remap` ()

*Remaps the master and slave programmable interrupt controller to the idt-entries 32-47.*

- void `irq_init` ()  
*Inits IRQ-support.*
- void `reactivate_pic` (bool slave)  
*The programmable interrupt controller needs to be reactivated after every interrupt.*
- `uint32 irq_handler` (uint32 int\_no, uint32 context)  
*Distributes the interrupts to their handlers.*

## Variables

- char \* `hw_messages` []

### 5.56.1 Detailed Description

Handler for hardware-interrupts.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.56.2 Function Documentation

#### 5.56.2.1 void `idt_fill_entry` ( uint8 *pos*, uint32 *offset*, uint16 *sel*, uint8 *flg* )

Makes a new idt-entry.

#### Parameters

*pos* Number of the new idt-entry  
*offset* Offset to be added to the base address  
*sel* Selector for the segment's base address  
*flg* Deskriptor flags

#### 5.56.2.2 void `irq0` ( )

Referenced by `irq_init`().

**5.56.2.3 void irq1 ( )**

Referenced by irq\_init().

**5.56.2.4 void irq10 ( )**

Referenced by irq\_init().

**5.56.2.5 void irq11 ( )**

Referenced by irq\_init().

**5.56.2.6 void irq12 ( )**

Referenced by irq\_init().

**5.56.2.7 void irq13 ( )**

Referenced by irq\_init().

**5.56.2.8 void irq14 ( )**

Referenced by irq\_init().

**5.56.2.9 void irq15 ( )**

Referenced by irq\_init().

**5.56.2.10 void irq2 ( )**

Referenced by irq\_init().

**5.56.2.11 void irq3 ( )**

Referenced by irq\_init().

**5.56.2.12 void irq4 ( )**

Referenced by irq\_init().

**5.56.2.13 void irq5 ( )**

Referenced by irq\_init().

**5.56.2.14 void irq6 ( )**

Referenced by irq\_init().

**5.56.2.15 void irq7 ( )**

Referenced by irq\_init().

**5.56.2.16 void irq8 ( )**

Referenced by irq\_init().

**5.56.2.17 void irq9 ( )**

Referenced by irq\_init().

**5.56.2.18 uint32 irq\_handler ( uint32 *int\_no*, uint32 *context* )**

Distributes the interrupts to their handlers.

**Parameters**

*int\_no* the hardware interrupt number

*context* a snapshot of the cpu registers at the time of the interrupt.

**Returns**

the context of the new active process (after scheduling)

References hd\_handler(), kb\_handler(), reactivate\_pic(), and timer\_handler().

**5.56.2.19 void irq\_init ( )**

Inits IRQ-support.

References idt\_fill\_entry(), irq0(), irq1(), irq10(), irq11(), irq12(), irq13(), irq14(), irq15(), irq2(), irq3(), irq4(), irq5(), irq6(), irq7(), irq8(), irq9(), and pic\_remap().

Referenced by io\_init().

**5.56.2.20 void outb ( uint8 *port*, uint8 *value* )**

Referenced by end\_beep(), hd\_init(), hd\_read\_sector(), hd\_write\_sector(), pic\_remap(), reactivate\_pic(), reboot(), rtc\_init(), rtc\_update(), select\_masterdrive(), start\_beep(), timer\_init(), update\_virt\_monitor(), and virt\_cursor\_move().

**5.56.2.21 void pic\_remap ( )**

Remaps the master and slave programable interrupt controller to the idt-entries 32-47.



References `outb()`.

Referenced by `irq_init()`.

#### 5.56.2.22 void reactivate\_pic ( bool *slave* )

The programmable interrupt controller needs to be reactivated after every interrupt.

This procedure provides this.

##### Parameters

*slave* should the slave pic be remapped too? (0 if not)

References `outb()`.

Referenced by `irq_handler()`, `sleep()`, and `sleep_ticks()`.

### 5.56.3 Variable Documentation

#### 5.56.3.1 char\* hw\_messages[ ]

##### Initial value:

```
{
    "timer",
    "keyboard",
    "irq 9",
    "com 2,4,6,8",
    "com 1,3,5,7",
    "ltp 2",
    "floppy",
    "ltp 1",
    "rtc",
    "vga",
    "pci",
    "scsi",
    "ps/2",
    "coprocessor",
    "primary ide",
    "secondary ide"
}
```

## 5.57 src/kernel/io/int\_isr.c File Reference

Handler for exceptions.

```
#include "../include/types.h"
```

```
#include "../include/stdio.h"
```

```
#include "../include/init.h"
```

```
#include "../pm/pm_main.h"
```

### Functions

- void `isr0()`

- void [isr1](#) ()
- void [isr2](#) ()
- void [isr3](#) ()
- void [isr4](#) ()
- void [isr5](#) ()
- void [isr6](#) ()
- void [isr7](#) ()
- void [isr8](#) ()
- void [isr9](#) ()
- void [isr10](#) ()
- void [isr11](#) ()
- void [isr12](#) ()
- void [isr13](#) ()
- void [isr14](#) ()
- void [isr15](#) ()
- void [isr16](#) ()
- void [isr17](#) ()
- void [isr18](#) ()
- void [isr19](#) ()
- void [isr20](#) ()
- void [isr21](#) ()
- void [isr22](#) ()
- void [isr23](#) ()
- void [isr24](#) ()
- void [isr25](#) ()
- void [isr26](#) ()
- void [isr27](#) ()
- void [isr28](#) ()
- void [isr29](#) ()
- void [isr30](#) ()
- void [isr31](#) ()
- void [incoming\\_syscall](#) ()
- void [idt\\_fill\\_entry](#) (uint8 pos, uint32 offset, uint16 sel, uint8 flg)  
*Makes a new idt-entry.*
- void [isr\\_init](#) ()  
*Fills the IDT with our interrupt service routines.*
- void [isr\\_handler](#) (cpu\_state\_t \*cpu\_state)  
*Handles all ISRs (for now just "panicing").*

## Variables

- char \* [ex\\_messages](#) []

## 5.57.1 Detailed Description

Handler for exceptions.

### Author

Dmitriy Traytel

### LastChangedBy:

dtraytel

### Version

### Rev:

12

## 5.57.2 Function Documentation

### 5.57.2.1 void idt\_fill\_entry ( uint8 *pos*, uint32 *offset*, uint16 *sel*, uint8 *flag* )

Makes a new idt-entry.

#### Parameters

- pos* Number of the new idt-entry
- offset* Offset to be added to the base address
- sel* Selector for the segment's base address
- flag* Deskriptor flags

References `idt_entry::flags`, `idt_entry::high_offset`, `idt`, `idt_entry::low_offset`, `idt_entry::selector`, and `idt_entry::separator`.

Referenced by `idt_init()`, `irq_init()`, and `isr_init()`.

### 5.57.2.2 void incoming\_syscall ( )

Referenced by `isr_init()`.

### 5.57.2.3 void isr0 ( )

Referenced by `isr_init()`.

### 5.57.2.4 void isr1 ( )

Referenced by `isr_init()`.

### 5.57.2.5 void isr10 ( )

Referenced by `isr_init()`.

**5.57.2.6 void isr11 ( )**

Referenced by isr\_init().

**5.57.2.7 void isr12 ( )**

Referenced by isr\_init().

**5.57.2.8 void isr13 ( )**

Referenced by isr\_init().

**5.57.2.9 void isr14 ( )**

Referenced by isr\_init().

**5.57.2.10 void isr15 ( )**

Referenced by isr\_init().

**5.57.2.11 void isr16 ( )**

Referenced by isr\_init().

**5.57.2.12 void isr17 ( )**

Referenced by isr\_init().

**5.57.2.13 void isr18 ( )**

Referenced by isr\_init().

**5.57.2.14 void isr19 ( )**

Referenced by isr\_init().

**5.57.2.15 void isr2 ( )**

Referenced by isr\_init().

**5.57.2.16 void isr20 ( )**

Referenced by isr\_init().

**5.57.2.17 void isr21 ( )**

Referenced by isr\_init().

**5.57.2.18 void isr22 ( )**

Referenced by isr\_init().

**5.57.2.19 void isr23 ( )**

Referenced by isr\_init().

**5.57.2.20 void isr24 ( )**

Referenced by isr\_init().

**5.57.2.21 void isr25 ( )**

Referenced by isr\_init().

**5.57.2.22 void isr26 ( )**

Referenced by isr\_init().

**5.57.2.23 void isr27 ( )**

Referenced by isr\_init().

**5.57.2.24 void isr28 ( )**

Referenced by isr\_init().

**5.57.2.25 void isr29 ( )**

Referenced by isr\_init().

**5.57.2.26 void isr3 ( )**

Referenced by isr\_init().

**5.57.2.27 void isr30 ( )**

Referenced by isr\_init().

**5.57.2.28 void isr31 ( )**

Referenced by `isr_init()`.

**5.57.2.29 void isr4 ( )**

Referenced by `isr_init()`.

**5.57.2.30 void isr5 ( )**

Referenced by `isr_init()`.

**5.57.2.31 void isr6 ( )**

Referenced by `isr_init()`.

**5.57.2.32 void isr7 ( )**

Referenced by `isr_init()`.

**5.57.2.33 void isr8 ( )**

Referenced by `isr_init()`.

**5.57.2.34 void isr9 ( )**

Referenced by `isr_init()`.

**5.57.2.35 void isr\_handler ( *cpu\_state\_t* \* *cpu\_state* )**

Handles all ISRs (for now just "panicing").

**Parameters**

*cpu\_state* a snapshot of the cpu registers at the time of the interrupt.

References `ex_messages`, `cpu_state_t::int_no`, and `panic`.

**5.57.2.36 void isr\_init ( )**

Fills the IDT with our interrupt service routines.

References `idt_fill_entry()`, `incoming_syscall()`, `isr0()`, `isr1()`, `isr10()`, `isr11()`, `isr12()`, `isr13()`, `isr14()`, `isr15()`, `isr16()`, `isr17()`, `isr18()`, `isr19()`, `isr2()`, `isr20()`, `isr21()`, `isr22()`, `isr23()`, `isr24()`, `isr25()`, `isr26()`, `isr27()`, `isr28()`, `isr29()`, `isr3()`, `isr30()`, `isr31()`, `isr4()`, `isr5()`, `isr6()`, `isr7()`, `isr8()`, and `isr9()`.

Referenced by `io_init()`.

### 5.57.3 Variable Documentation

#### 5.57.3.1 char\* ex\_messages[]

Referenced by `isr_handler()`.

## 5.58 src/kernel/io/io.h File Reference

The global IO functions header.

```
#include "../include/types.h"
```

### Functions

- void `idt_init` ()
- void `isr_init` ()  
*Fills the IDT with our interrupt service routines.*
  
- void `irq_init` ()  
*Inits IRQ-support.*
  
- void `timer_init` (int hz)
- void `monitor_init` ()
- void `keyboard_init` ()
- void `hd_init` ()  
*Initializes the hard disk drive.*
  
- `sint32 get_ticks` ()  
*Returns the timer ticks.*
  
- void `set_disp` (uint32 addr)
- void `add_shortcut` (bool control\_flag, bool super\_flag, uint8 character, void(\*function)())  
*Adds a shortcut to the system.*
  
- void `mutex_lock` (int \*lock)
- void `mutex_unlock` (int \*lock)
- void `set_interrupts` ()  
*Permits interrupts.*
  
- void `clear_interrupts` ()  
*Forbids interrupts.*
  
- void `make_syscall` (uint16 num, void \*syscall\_struct)
- void `outb` (uint16 port, uint8 value)  
*Writes a byte to an io-port.*
  
- `uint8 inb` (uint16 port)  
*Reads a byte from an io-port.*

- void `repinsw` (`uint16` port, `uint16` \*dest, `uint32` num)  
*Reads words from an io-port.*
- void `repoutsw` (`uint16` port, `uint16` \*src, `uint32` num)  
*Writes words to an io-port.*
- void `halt` ()  
*Gives the CPU some rest ;-).*
- void `reactivate_pic` ()
- void `monitor_invert` ()  
*Inverts the monitor.*
- void `start_beep` (`uint32` freq)
- void `end_beep` ()

### 5.58.1 Detailed Description

The global IO functions header.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.58.2 Function Documentation

#### 5.58.2.1 void `add_shortcut` ( `bool` *control\_flag*, `bool` *super\_flag*, `uint8` *character*, `void`(\*)() *function* )

Adds a shortcut to the system.

#### Parameters

*control\_flag* is ctrl the initiator?

*super\_flag* is super the initiator?

*character* shortcut character

(*\*function*)() function to be called

References `shortcut::ch`, `shortcut::control`, `shortcut::func`, `shcut_num`, `SHORTCUTS_ARRAY_SIZE`, and `shortcut::super`.



### 5.58.2.2 void clear\_interrupts ( )

Forbids interrupts.

Referenced by panic(), and pm\_create\_thread().

### 5.58.2.3 void end\_beep ( )

References inb(), IO\_SPEAKER\_PORT, and outb().

Referenced by shell\_cmd\_pong(), and snake().

### 5.58.2.4 sint32 get\_ticks ( )

Returns the timer ticks.

#### Returns

ticks since system boot or last timer wraparound

Referenced by main(), shell\_cmd\_bf(), and snake().

### 5.58.2.5 void halt ( )

Gives the CPU some rest ;-).

### 5.58.2.6 void hd\_init ( )

Initializes the hard disk drive.

References dump\_hd1(), hd1, HDBASE, HDCMD\_IDENTIFY\_DEVICE, HDREG\_CMD, HDREG\_DATA, HDREG\_STAT, HDSTATE\_FLOATINGBUS, HDSTATE\_NOTREADY, inb(), outb(), panic, repinsw(), select\_masterdrive(), and wait\_on\_hd\_interrupt().

Referenced by io\_init().

### 5.58.2.7 void idt\_init ( )

References idt, idt\_fill\_entry(), idt\_flush(), idtp, idt\_pointer::maxsize, and idt\_pointer::start.

Referenced by io\_init().

### 5.58.2.8 uint8 inb ( uint16 port )

Reads a byte from an io-port.

#### Parameters

*port* specifies the port to be read from

#### Returns

the value of the byte that was read from port

Referenced by `end_beep()`, `hd_handler()`, `hd_init()`, `hd_read_sector()`, `hd_write_sector()`, `kb_handler()`, `rtc_init()`, `rtc_update()`, `start_beep()`, and `wait_on_hd_interrupt()`.

#### 5.58.2.9 `void irq_init ( )`

Inits IRQ-support.

References `idt_fill_entry()`, `irq0()`, `irq1()`, `irq10()`, `irq11()`, `irq12()`, `irq13()`, `irq14()`, `irq15()`, `irq2()`, `irq3()`, `irq4()`, `irq5()`, `irq6()`, `irq7()`, `irq8()`, `irq9()`, and `pic_remap()`.

Referenced by `io_init()`.

#### 5.58.2.10 `void isr_init ( )`

Fills the IDT with our interrupt service routines.

References `idt_fill_entry()`, `incoming_syscall()`, `isr0()`, `isr1()`, `isr10()`, `isr11()`, `isr12()`, `isr13()`, `isr14()`, `isr15()`, `isr16()`, `isr17()`, `isr18()`, `isr19()`, `isr2()`, `isr20()`, `isr21()`, `isr22()`, `isr23()`, `isr24()`, `isr25()`, `isr26()`, `isr27()`, `isr28()`, `isr29()`, `isr3()`, `isr30()`, `isr31()`, `isr4()`, `isr5()`, `isr6()`, `isr7()`, `isr8()`, and `isr9()`.

Referenced by `io_init()`.

#### 5.58.2.11 `void keyboard_init ( )`

#### 5.58.2.12 `void make_syscall ( uint16 num, void * syscall_struct )`

#### 5.58.2.13 `void monitor_init ( )`

#### 5.58.2.14 `void monitor_invert ( )`

Inverts the monitor.

Referenced by `monitor_cputc()`.

#### 5.58.2.15 `void mutex_lock ( int * lock )`

Referenced by `p()`, `threadA()`, `threadB()`, and `v()`.

#### 5.58.2.16 `void mutex_unlock ( int * lock )`

Referenced by `p()`, `threadA()`, `threadB()`, and `v()`.

#### 5.58.2.17 `void outb ( uint16 port, uint8 value )`

Writes a byte to an io-port.

#### Parameters

*port* specifies the port to be written to

*value* the value of the byte to be written

**5.58.2.18** void `reactivate_pic` ( )

**5.58.2.19** void `repinsw` ( `uint16 port`, `uint16 * dest`, `uint32 num` )

Reads words from an io-port.

#### Parameters

*port* specifies the port to be read from

*\*dest* the pointer to the memory where the data should be written to

*num* number of words to be read

Referenced by `hd_init()`, and `hd_read_sector()`.

**5.58.2.20** void `repoutsw` ( `uint16 port`, `uint16 * src`, `uint32 num` )

Writes words to an io-port.

#### Parameters

*port* specifies the port to be written to

*\*src* the pointer to the memory where the data should be read from

*num* number of words to be written

Referenced by `hd_write_sector()`.

**5.58.2.21** void `set_disp` ( `uint32 addr` )

Referenced by `update_virt_monitor()`.

**5.58.2.22** void `set_interrupts` ( )

Permits interrupts.

Referenced by `io_init()`, `kb_handler()`, `pm_create_thread()`, `pm_syscall()`, `sleep()`, and `sleep_ticks()`.

**5.58.2.23** void `start_beep` ( `uint32 freq` )

References `inb()`, `IO_SPEAKER_PORT`, `outb()`, `PIT_CONTROL`, `PIT_COUNTER2`, and `PIT_SOUND_CMD`.

Referenced by `shell_cmd_pong()`, and `snake()`.

**5.58.2.24** void `timer_init` ( `int hz` )

Referenced by `io_init()`.

## 5.59 src/kernel/io/io\_harddisk.c File Reference

Harddisk-handler.

```
#include "../include/stdio.h"
#include "../include/types.h"
#include "../include/const.h"
#include "../include/init.h"
#include "../include/util.h"
#include "../io/io.h"
#include "../io/io_harddisk.h"
```

### Functions

- void [dump\\_hd1](#) ()  
*Printing some hard disk informations, taken from the hd1 struct.*
- void [select\\_masterdrive](#) (uint8 head)  
*Selecting a head of the masterdrive.*
- void [wait\\_on\\_hd\\_interrupt](#) (char \*str)  
*Waits on hard disk to set the hd\_interrupt flag or the drive\_ready flag in the status register.*
- struct [address](#) [itoaddr](#) (uint32 iaddr)  
*Converts a linear block number into a valid CHS-address.*
- uint32 [get\\_hdsiz](#)e ()  
*Returns the size of the hard disk.*
- void [hd\\_init](#) ()  
*Initializes the hard disk drive.*
- void [hd\\_write\\_sector](#) (uint32 dest, void \*src)  
*Writes a sector to the hard disk.*
- void [hd\\_read\\_sector](#) (void \*dest, uint32 src)  
*Reads a sector from the hard disk.*
- void [hd\\_handler](#) ()  
*Handles an hard disk interrupt by setting the hd\_interrupt flag.*

### Variables

- struct [hd\\_info](#) [hd1](#)  
*Our single hard disk struct.*

- volatile `bool hd_interrupt` = FALSE  
*Signals a pending hard disk interrupt.*

### 5.59.1 Detailed Description

Harddisk-handler.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.59.2 Function Documentation

#### 5.59.2.1 void dump\_hd1 ( )

Printing some hard disk informations, taken from the `hd1` struct.

References `hd_info::apparent_cyl`, `hd_info::apparent_head`, `hd_info::apparent_sector_per_track`, `hd_info::buffer_size`, `hd_info::buffer_type`, `hd_info::bytes_per_sector`, `get_hdsizes`, `hd1`, `hd_info::lba_dma_flg`, `printf`, and `putchar`.

Referenced by `hd_init()`.

#### 5.59.2.2 uint32 get\_hdsizes ( )

Returns the size of the hard disk.

#### Returns

size of the master hard disk in sectors

References `hd_info::apparent_capacity`, and `hd1`.

#### 5.59.2.3 void hd\_handler ( )

Handles an hard disk interrupt by setting the `hd_interrupt` flag.

References `hd_interrupt`, `HDBASE`, `HDREG_STAT`, `inb()`, and `panic`.

Referenced by `irq_handler()`.

#### 5.59.2.4 void `hd_init` ( )

Initializes the hard disk drive.

References `dump_hd1()`, `hd1`, `HDBASE`, `HDCMD_IDENTIFY_DEVICE`, `HDREG_CMD`, `HDREG_DATA`, `HDREG_STAT`, `HDSTATE_FLOATINGBUS`, `HDSTATE_NOTREADY`, `inb()`, `outb()`, `panic`, `repinsw()`, `select_masterdrive()`, and `wait_on_hd_interrupt()`.

Referenced by `io_init()`.

#### 5.59.2.5 void `hd_read_sector` ( void \* *dest*, uint32 *src* )

Reads a sector from the hard disk.

##### Parameters

- \*dest* the pointer where the read data should go to
- src* the address of the sector to be read from

References `address::cyl`, `get_hdsiz`, `hd_read_sector`, `HDALTBASE`, `HDALTREG_STAT`, `HDBASE`, `HDCMD_READ`, `HDREG_CMD`, `HDREG_COUNT`, `HDREG_CYL_HIGH`, `HDREG_CYL_LOW`, `HDREG_DATA`, `HDREG_SEC`, `HDSTATE_NOTREADY`, `address::head`, `inb()`, `itoaddr()`, `outb()`, `panic`, `repinsw()`, `address::sector`, `select_masterdrive()`, and `wait_on_hd_interrupt()`.

#### 5.59.2.6 void `hd_write_sector` ( uint32 *dest*, void \* *src* )

Writes a sector to the hard disk.

##### Parameters

- dest* the address of the sector to be written to
- \*src* the pointer where data should be written from

The EEEPC needs a cache flush after a write operation.

References `address::cyl`, `get_hdsiz`, `HDALTBASE`, `HDALTREG_STAT`, `HDBASE`, `HDCMD_FLUSH_CACHE`, `HDCMD_WRITE`, `HDREG_CMD`, `HDREG_COUNT`, `HDREG_CYL_HIGH`, `HDREG_CYL_LOW`, `HDREG_DATA`, `HDREG_SEC`, `HDSTATE_NOTREADY`, `address::head`, `inb()`, `itoaddr()`, `outb()`, `panic`, `repoutsw()`, `address::sector`, `select_masterdrive()`, and `wait_on_hd_interrupt()`.

#### 5.59.2.7 struct `address` `itoaddr` ( uint32 *iaddr* ) [read]

Converts a linear block number into a valid CHS-address.

##### Parameters

- iaddr* the logical block number

##### Returns

the CHS-address struct

References `hd_info::apparent_head`, `hd_info::apparent_sector_per_track`, `address::cyl`, `hd1`, `address::head`, and `address::sector`.

Referenced by `hd_read_sector()`, and `hd_write_sector()`.

### 5.59.2.8 void select\_masterdrive ( uint8 head )

Selecting a head of the masterdrive.

#### Parameters

*head* the number of the head to be selected

References HDBASE, HDREG\_DRIVE, MASTERDRIVE, and outb().

Referenced by hd\_init(), hd\_read\_sector(), and hd\_write\_sector().

### 5.59.2.9 void wait\_on\_hd\_interrupt ( char \* str )

Waits on hard disk to set the hd\_interrupt flag or the drive\_ready flag in the status register.

#### Parameters

*str* should be the source of the function call for debugging issues

References halt(), hd\_interrupt, HDALTBASE, HDALTREG\_STAT, HDBASE, HDREG\_ERR, HDREG\_STAT, inb(), and printf.

Referenced by hd\_init(), hd\_read\_sector(), and hd\_write\_sector().

## 5.59.3 Variable Documentation

### 5.59.3.1 struct hd\_info hd1

Our single hard disk struct.

etiOS is handling only one hard disk device.

Referenced by dump\_hd1(), get\_hdsizes(), hd\_init(), and itoaddr().

### 5.59.3.2 volatile bool hd\_interrupt = FALSE

Signals a pending hard disk interrupt.

Referenced by hd\_handler(), and wait\_on\_hd\_interrupt().

## 5.60 src/kernel/io/io\_harddisk.h File Reference

header file for the hard disk driver

```
#include "../include/types.h"
```

### Data Structures

- struct [hd\\_info](#)

*This struct is filled by the IDENTIFY DRIVE command.*

- struct [address](#)

*Address data for the hard disk packed in a struct.*

## Defines

- #define [HDBASE](#) 0x1F0

*The io ports of the EEEPC are mapped at different addresses compared to other architectures.*

- #define [HDALTBASE](#) 0x3F6
- #define [HDREG\\_DATA](#) 0
- #define [HDREG\\_ERR](#) 1
- #define [HDREG\\_COUNT](#) 2
- #define [HDREG\\_SEC](#) 3
- #define [HDREG\\_CYL\\_LOW](#) 4
- #define [HDREG\\_CYL\\_HIGH](#) 5
- #define [HDREG\\_DRIVE](#) 6
- #define [HDREG\\_STAT](#) 7
- #define [HDREG\\_CMD](#) 7
- #define [HDALTREG\\_STAT](#) 0
- #define [HDALTREG\\_ADDR](#) 1
- #define [MASTERDRIVE](#) 0xA0
- #define [SLAVEDRIVE](#) 0xB0
- #define [HDCMD\\_EXEC\\_DRIVE\\_DIAG](#) 0x90
- #define [HDCMD\\_IDENTIFY\\_DEVICE](#) 0xEC
- #define [HDCMD\\_READ](#) 0x20
- #define [HDCMD\\_WRITE](#) 0x30
- #define [HDCMD\\_FLUSH\\_CACHE](#) 0xE7
- #define [HDSTATE\\_FLOATINGBUS](#) 0xFF
- #define [HDSTATE\\_NOTREADY](#) 0x80

## Functions

- struct [hd\\_info](#) [\\_\\_attribute\\_\\_\(\(packed\)\)](#)

*This struct is filled by the IDENTIFY DRIVE command.*

- [uint32](#) [get\\_hdsiz](#)( )

*Returns the size of the hard disk.*

## Variables

- [uint16](#) [config\\_word](#)
- [uint16](#) [num\\_cyl](#)
- [uint16](#) [reserved1](#)
- [uint16](#) [num\\_head](#)
- [uint16](#) [bytes\\_per\\_track](#)
- [uint16](#) [bytes\\_per\\_sector](#)
- [uint16](#) [sector\\_per\\_track](#)



- uint16 manufacturer1 [3]
- uint16 serial [10]
- uint16 buffer\_type
- uint16 buffer\_size
- uint16 num\_ecc\_bytes
- uint16 firmware [4]
- uint16 type [20]
- uint16 rw\_multiple\_flg
- uint16 dw\_io\_flg
- uint16 lba\_dma\_flg
- uint16 reserved2
- uint16 timingmode\_pio
- uint16 timingmode\_dma
- uint16 reserved3
- uint16 apparent\_cyl
- uint16 apparent\_head
- uint16 apparent\_sector\_per\_track
- uint16 apparent\_capacity [2]
- uint16 sectors\_per\_int
- uint16 num\_lba\_sectors [2]
- uint16 mode\_single\_dma
- uint16 mode\_multi\_dma
- uint16 reserved4 [64]
- uint16 manufacturer2 [32]
- uint16 reserved5 [96]
- struct address \_\_attribute\_\_

*Address data for the hard disk packed in a struct.*

### 5.60.1 Detailed Description

header file for the hard disk driver

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.60.2 Define Documentation

#### 5.60.2.1 #define HDAITBASE 0x3F6

Referenced by hd\_read\_sector(), hd\_write\_sector(), and wait\_on\_hd\_interrupt().

**5.60.2.2 #define HDALTREG\_ADDR 1****5.60.2.3 #define HDALTREG\_STAT 0**

Referenced by `hd_read_sector()`, `hd_write_sector()`, and `wait_on_hd_interrupt()`.

**5.60.2.4 #define HDBASE 0x1F0**

The io ports of the EEEPC are mapped at different addresses compared to other architectures.

If you're running etiOS on the EEEPC, you need to compile the kernel with `make EXTRA_CFLAGS=-DEEEPC [target]`

Referenced by `hd_handler()`, `hd_init()`, `hd_read_sector()`, `hd_write_sector()`, `select_masterdrive()`, and `wait_on_hd_interrupt()`.

**5.60.2.5 #define HDCMD\_EXEC\_DRIVE\_DIAG 0x90****5.60.2.6 #define HDCMD\_FLUSH\_CACHE 0xE7**

Referenced by `hd_write_sector()`.

**5.60.2.7 #define HDCMD\_IDENTIFY\_DEVICE 0xEC**

Referenced by `hd_init()`.

**5.60.2.8 #define HDCMD\_READ 0x20**

Referenced by `hd_read_sector()`.

**5.60.2.9 #define HDCMD\_WRITE 0x30**

Referenced by `hd_write_sector()`.

**5.60.2.10 #define HDREG\_CMD 7**

Referenced by `hd_init()`, `hd_read_sector()`, and `hd_write_sector()`.

**5.60.2.11 #define HDREG\_COUNT 2**

Referenced by `hd_read_sector()`, and `hd_write_sector()`.

**5.60.2.12 #define HDREG\_CYL\_HIGH 5**

Referenced by `hd_read_sector()`, and `hd_write_sector()`.

**5.60.2.13 #define HDREG\_CYL\_LOW 4**

Referenced by `hd_read_sector()`, and `hd_write_sector()`.

**5.60.2.14 #define HDREG\_DATA 0**

Referenced by `hd_init()`, `hd_read_sector()`, and `hd_write_sector()`.

**5.60.2.15 #define HDREG\_DRIVE 6**

Referenced by `select_masterdrive()`.

**5.60.2.16 #define HDREG\_ERR 1**

Referenced by `wait_on_hd_interrupt()`.

**5.60.2.17 #define HDREG\_SEC 3**

Referenced by `hd_read_sector()`, and `hd_write_sector()`.

**5.60.2.18 #define HDREG\_STAT 7**

Referenced by `hd_handler()`, `hd_init()`, and `wait_on_hd_interrupt()`.

**5.60.2.19 #define HDSTATE\_FLOATINGBUS 0xFF**

Referenced by `hd_init()`.

**5.60.2.20 #define HDSTATE\_NOTREADY 0x80**

Referenced by `hd_init()`, `hd_read_sector()`, and `hd_write_sector()`.

**5.60.2.21 #define MASTERDRIVE 0xA0**

Referenced by `select_masterdrive()`.

**5.60.2.22 #define SLAVEDRIVE 0xB0****5.60.3 Function Documentation****5.60.3.1 struct hd\_info \_\_attribute\_\_((packed))**

This struct is filled by the IDENTIFY DRIVE command.

It contains hard disk geometry data and more.

**5.60.3.2 uint32 get\_hdsiz ( )**

Returns the size of the hard disk.

**Returns**

size of the master hard disk in sectors

References `hd_info::apparent_capacity`, and `hd1`.

**5.60.4 Variable Documentation****5.60.4.1 struct address \_\_attribute\_\_**

Address data for the hard disk packed in a struct.



- 5.60.4.2 uint16 apparent\_capacity[2]
- 5.60.4.3 uint16 apparent\_cyl
- 5.60.4.4 uint16 apparent\_head
- 5.60.4.5 uint16 apparent\_sector\_per\_track
- 5.60.4.6 uint16 buffer\_size
- 5.60.4.7 uint16 buffer\_type
- 5.60.4.8 uint16 bytes\_per\_sector
- 5.60.4.9 uint16 bytes\_per\_track
- 5.60.4.10 uint16 config\_word
- 5.60.4.11 uint16 dw\_io\_flg
- 5.60.4.12 uint16 firmware[4]
- 5.60.4.13 uint16 lba\_dma\_flg
- 5.60.4.14 uint16 manufacturer1[3]
- 5.60.4.15 uint16 manufacturer2[32]
- 5.60.4.16 uint16 mode\_multi\_dma
- 5.60.4.17 uint16 mode\_single\_dma
- 5.60.4.18 uint16 num\_cyl
- 5.60.4.19 uint16 num\_ecc\_bytes
- 5.60.4.20 uint16 num\_head
- 5.60.4.21 uint16 num\_lba\_sectors[2]
- 5.60.4.22 uint16 reserved1
- 5.60.4.23 uint16 reserved2
- 5.60.4.24 uint16 reserved3
- 5.60.4.25 uint16 reserved4[64]
- 5.60.4.26 uint16 reserved5[96]
- 5.60.4.27 uint16 rw\_multiple\_flg
- 5.60.4.28 uint16 sector\_per\_track
- 5.60.4.29 uint16 sectors\_per\_int
- 5.60.4.30 uint16 serial[10]
- 5.60.4.31 uint16 timingmode\_dma
- 5.60.4.32 uint16 timingmode\_pio

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/stdio.h"
#include "../include/stdlib.h"
#include "../include/assert.h"
#include "../io/io.h"
#include "../io/io_virtual.h"
#include "../io/io_keyboard.h"
#include "../pm/pm_input.h"
```

## Defines

- `#define SHORTCUTS_ARRAY_SIZE 300`

## Functions

- void `add_shortcut` (`bool` control\_flag, `bool` super\_flag, `uint8` character, void(\*function)())  
*Adds a shortcut to the system.*
- void `kb_handler` ()  
*Handles a keyboard interrupt by calling the PM (providing already the right character).*

## Variables

- `bool` `echo` = FALSE  
*Echo-mode flag.*
- `uint8` `shcut_num` = 0  
*Number of registered shortcuts.*
- `shortcut` `shortcuts` [SHORTCUTS\_ARRAY\_SIZE]  
*the keyboard shortcuts struct*
- `bool` `keyboard_state` [256]  
*State of all keys.*

### 5.61.1 Detailed Description

Keyboard-handler.

#### Author

Dmitriy Traytel

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

**5.61.2 Define Documentation****5.61.2.1 #define SHORTCUTS\_ARRAY\_SIZE 300**

Referenced by add\_shortcut().

**5.61.3 Function Documentation****5.61.3.1 void add\_shortcut ( bool *control\_flag*, bool *super\_flag*, uint8 *character*, void(\*)() *function* )**

Adds a shortcut to the system.

**Parameters***control\_flag* is ctrl the initiator?*super\_flag* is super the initiator?*character* shortcut character*(\*function)()* function to be called

References shortcut::ch, shortcut::control, shortcut::func, shcut\_num, SHORTCUTS\_ARRAY\_SIZE, and shortcut::super.

**5.61.3.2 void kb\_handler ( )**

Handles a keyboard interrupt by calling the PM (providing already the right character).

In echo-mode prints the typed character directly to the screen.

References alt, ALT, ctrl, CTRL, CURSOR\_DOWN, CURSOR\_LEFT, CURSOR\_RIGHT, CURSOR\_UP, echo, ESCAPE, shortcut::func, get\_active\_virt\_monitor(), inb(), kb\_alt\_map, kb\_map, KB\_PORT, kb\_shift\_map, KEY\_PRESSED, KEY\_RELEASED, keyboard\_state, LSHIFT, pm\_handle\_input(), RSHIFT, SCROLL\_DOWN, SCROLL\_UP, set\_interrupts(), shcut\_num, shift, super, SUPER, super\_button, virt\_cursor\_move(), virt\_monitor\_invert(), virt\_monitor\_putc(), virt\_monitor\_scrollup(), and virt\_monitor\_scrollup().

Referenced by irq\_handler().



## 5.61.4 Variable Documentation

### 5.61.4.1 bool echo = FALSE

Echo-mode flag.

Referenced by kb\_handler().

### 5.61.4.2 bool keyboard\_state[256]

State of all keys.

TRUE means the key is being held down right now. This is used by the /dev/keyboard device to provide raw keyboard access.

Referenced by dev\_keyboard\_read(), io\_init(), and kb\_handler().

### 5.61.4.3 uint8 shcut\_num = 0

Number of registered shortcuts.

Referenced by add\_shortcut(), and kb\_handler().

### 5.61.4.4 shortcut shortcuts[SHORTCUTS\_ARRAY\_SIZE]

the keyboard shortcuts struct

## 5.62 src/kernel/io/io\_keyboard.h File Reference

Header for the keyboard-handler.

```
#include "../include/types.h"
```

### Data Structures

- struct [shortcut](#)  
*Structure of a shortcut.*

### Defines

- #define [LSHIFT](#) 0x2A
- #define [RSHIFT](#) 0x36
- #define [ALT](#) 0x38
- #define [SCROLL\\_UP](#) 0x49
- #define [SCROLL\\_DOWN](#) 0x51
- #define [CURSOR\\_UP](#) 0x48
- #define [CURSOR\\_DOWN](#) 0x50
- #define [CURSOR\\_LEFT](#) 0x4B
- #define [CURSOR\\_RIGHT](#) 0x4D

- #define `CTRL` 0x1D
- #define `SUPER` 0x5B
- #define `ESCAPE` 0x01
- #define `ENTER` 0x1C
- #define `KB_PORT` 0x60
- #define `KEY_PRESSED` (~0x80)
- #define `KEY_RELEASED` 0x80

## Functions

- void `cursor_move` (uint8 dir)

## Variables

- char `kb_map` []
- char `kb_shift_map` []
- char `kb_alt_map` []
- bool `shift` = 0  
*Shift-pressed flag.*
- bool `alt` = 0  
*Alt-pressed flag.*
- bool `super_button` = 0  
*Super-pressed flag.*
- bool `ctrl` = 0  
*Ctrl-pressed flag.*

### 5.62.1 Detailed Description

Header for the keyboard-handler. Includes the keyboard-map (german).

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

## 5.62.2 Define Documentation

### 5.62.2.1 #define ALT 0x38

Referenced by kb\_handler().

### 5.62.2.2 #define CTRL 0x1D

Referenced by kb\_handler().

### 5.62.2.3 #define CURSOR\_DOWN 0x50

### 5.62.2.4 #define CURSOR\_LEFT 0x4B

### 5.62.2.5 #define CURSOR\_RIGHT 0x4D

### 5.62.2.6 #define CURSOR\_UP 0x48

### 5.62.2.7 #define ENTER 0x1C

### 5.62.2.8 #define ESCAPE 0x01

### 5.62.2.9 #define KB\_PORT 0x60

Referenced by kb\_handler().

### 5.62.2.10 #define KEY\_PRESSED (~0x80)

Referenced by kb\_handler().

### 5.62.2.11 #define KEY\_RELEASED 0x80

Referenced by kb\_handler().

### 5.62.2.12 #define LSHIFT 0x2A

Referenced by kb\_handler().

### 5.62.2.13 #define RSHIFT 0x36

Referenced by kb\_handler().

### 5.62.2.14 #define SCROLL\_DOWN 0x51

Referenced by kb\_handler().

**5.62.2.15 #define SCROLL\_UP 0x49**

Referenced by kb\_handler().

**5.62.2.16 #define SUPER 0x5B**

Referenced by kb\_handler().

**5.62.3 Function Documentation****5.62.3.1 void cursor\_move ( uint8 dir )****5.62.4 Variable Documentation****5.62.4.1 bool alt = 0**

Alt-pressed flag.

Referenced by kb\_handler().

**5.62.4.2 bool ctrl = 0**

Ctrl-pressed flag.

Referenced by kb\_handler().

**5.62.4.3 char kb\_alt\_map[ ]**

Referenced by kb\_handler().

**5.62.4.4 char kb\_map[ ]**

Referenced by kb\_handler().

**5.62.4.5 char kb\_shift\_map[ ]**

Referenced by kb\_handler().

**5.62.4.6 bool shift = 0**

Shift-pressed flag.

Referenced by kb\_handler().

**5.62.4.7 bool super\_button = 0**

Super-pressed flag.

Referenced by kb\_handler().

## 5.63 src/kernel/io/io\_main.c File Reference

IO-init function for main.

```
#include "../io/io.h"
#include "../io/io_rtc.h"
#include "../io/io_virtual.h"
#include "../include/const.h"
#include "../include/debug.h"
#include "../include/string.h"
```

### Functions

- void [io\\_init](#) ()

### Variables

- bool [keyboard\\_state](#) [256]  
*State of all keys.*

### 5.63.1 Detailed Description

IO-init function for main.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.63.2 Function Documentation

#### 5.63.2.1 void [io\\_init](#) ( )

References [dprint\\_separator](#), [dprintf](#), [FALSE](#), [FREQUENCY](#), [hd\\_init\(\)](#), [idt\\_init\(\)](#), [irq\\_init\(\)](#), [isr\\_init\(\)](#), [keyboard\\_state](#), [memset\(\)](#), [set\\_interrupts\(\)](#), and [timer\\_init\(\)](#).

Referenced by [main\(\)](#).

### 5.63.3 Variable Documentation

#### 5.63.3.1 bool keyboard\_state[256]

State of all keys.

TRUE means the key is being held down right now. This is used by the /dev/keyboard device to provide raw keyboard access.

## 5.64 src/kernel/io/io\_monitor.c File Reference

Functions to print things on the monitor.

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/stdio.h"
#include "../include/string.h"
#include "../include/stdlib.h"
#include "../include/util.h"
#include "../include/ringbuffer.h"
#include "../include/assert.h"
#include "../io/io.h"
```

### Functions

- void [set\\_disp](#) (uint32 addr)
- void [monitor\\_cputc](#) (char ch, uint8 fg, uint8 bg)  
*Writes a colored character to the display.*
- void [monitor\\_cputs](#) (char \*str, uint8 fg, uint8 bg)  
*Writes a colored null-terminated string to the display.*
- void [monitor\\_putc](#) (char ch)  
*Writes a character to the display.*
- void [monitor\\_puts](#) (char \*str)  
*Writes a null-terminated string to the display.*
- void [monitor\\_puti](#) (sint32 x)  
*Writes an integer to the display.*
- void [monitor\\_puthex](#) (uint8 ch)  
*Writes a hex-byte to the display.*
- void [monitor\\_invert](#) ()  
*Inverts the monitor.*

## 5.64.1 Detailed Description

Functions to print things on the monitor.

### Author

Dmitriy Traytel

### LastChangedBy:

dtraytel

### Version

### Rev:

12

## 5.64.2 Function Documentation

### 5.64.2.1 void monitor\_cputc ( char *ch*, uint8 *fg*, uint8 *bg* )

Writes a colored character to the display.

#### Parameters

*ch* character to be written

*fg* foreground-color

*bg* background color

References monitor\_cputc(), monitor\_invert(), sleep\_ticks(), and VGA\_DISPLAY.

Referenced by monitor\_cputc(), monitor\_cputs(), monitor\_putc(), monitor\_puthex(), and monitor\_puti().

### 5.64.2.2 void monitor\_cputs ( char \* *str*, uint8 *fg*, uint8 *bg* )

Writes a colored null-terminated string to the display.

#### Parameters

\**str* pointer to the string

*fg* foreground-color

*bg* background color

References monitor\_cputc().

Referenced by draw\_test(), monitor\_puts(), panic(), and update\_virt\_monitor().

### 5.64.2.3 void monitor\_invert ( )

Inverts the monitor.

Referenced by monitor\_cputc().

**5.64.2.4 void monitor\_putc ( char *ch* )**

Writes a character to the display.

**Parameters**

*ch* character to be written

References BLACK, monitor\_cputc(), and WHITE.

Referenced by monitor\_puti().

**5.64.2.5 void monitor\_puthex ( uint8 *ch* )**

Writes a hex-byte to the display.

**Parameters**

*ch* character to be written

References BLACK, GREEN, and monitor\_cputc().

Referenced by draw\_test().

**5.64.2.6 void monitor\_puti ( sint32 *x* )**

Writes an integer to the display.

**Parameters**

*x* integer to be written

References BLACK, monitor\_cputc(), monitor\_putc(), and RED.

Referenced by draw\_test().

**5.64.2.7 void monitor\_puts ( char \* *str* )**

Writes a null-terminated string to the display.

**Parameters**

*\*str* pointer to the string

References BLACK, monitor\_cputs(), and WHITE.

Referenced by draw\_test(), and grubstruct\_test().

**5.64.2.8 void set\_disp ( uint32 *addr* )**

Referenced by update\_virt\_monitor().



## 5.65 src/kernel/io/io\_rtc.c File Reference

The real-time clock.

```
#include "../io/io_rtc.h"
#include "../io/io.h"
#include "../include/init.h"
#include "../include/stdio.h"
#include "../include/stdlib.h"
#include "../include/string.h"
#include "../include/types.h"
```

### Functions

- `uint8 bcd2bin (uint8 num)`  
*Converts a bcd-value into a binary value.*
- `void calculate_weekday (time_t *ts)`  
*Calculates the weekday with the algorithm from [http://en.wikipedia.org/wiki/Calculating\\_the\\_day\\_of\\_the\\_week](http://en.wikipedia.org/wiki/Calculating_the_day_of_the_week).*
- `void rtc_init ()`  
*Initializes the global data struct.*
- `void rtc_update ()`  
*Updates the date-data in the global time struct.*
- `char * time2str (time_t timestamp, char buf[24])`  
*Converts the data in the global time struct into a readable string.*

### 5.65.1 Detailed Description

The real-time clock.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

## 5.65.2 Function Documentation

### 5.65.2.1 uint8 bcd2bin ( uint8 *num* )

Converts a bcd-value into a binary value.

#### Parameters

*num* the bcd-value to convert

#### Returns

the corresponding binary value

Referenced by calculate\_weekday(), threadA(), and threadB().

### 5.65.2.2 void calculate\_weekday ( time\_t \* *ts* )

Calculates the weekday with the algorithm from [http://en.wikipedia.org/wiki/Calculating\\_the\\_day\\_of\\_the\\_week](http://en.wikipedia.org/wiki/Calculating_the_day_of_the_week).

#### Parameters

*ts* the pointer to the time struct which weekday shall be calculated

References bcd2bin(), time::day, time::month, time::weekday, and time::year.

Referenced by rtc\_init(), and rtc\_update().

### 5.65.2.3 void rtc\_init ( )

Initializes the global data struct.

References calculate\_weekday(), inb(), outb(), RTC\_ADDR, RTC\_CENTURY, RTC\_DATA, RTC\_DAY, RTC\_HOUR, RTC\_MIN, RTC\_MONTH, RTC\_SEC, and RTC\_YEAR.

Referenced by init\_vmonitors().

### 5.65.2.4 void rtc\_update ( )

Updates the date-data in the global time struct.

References calculate\_weekday(), inb(), outb(), RTC\_ADDR, RTC\_CENTURY, RTC\_DATA, RTC\_DAY, RTC\_HOUR, RTC\_MIN, RTC\_MONTH, RTC\_SEC, and RTC\_YEAR.

Referenced by timer\_handler().

### 5.65.2.5 char\* time2str ( time\_t *timestamp*, char *buf*[24] )

Converts the data in the global time struct into a readable string.

#### Parameters

*timestamp* the actual time struct to be converted

*buf* the buffer for the date-string

## Returns

the pointer to the same buffer now filled with the date string

References `time::century`, `time::day`, `time::hour`, `itoa`, `time::min`, `time::month`, `time::sec`, `strcpy()`, `time::weekday`, and `time::year`.

## 5.66 src/kernel/io/io\_rtc.h File Reference

The real-time clock constants and the time struct.

```
#include "../include/types.h"
```

## Data Structures

- struct `time`  
*Global time struct.*

## Defines

- #define `RTC_ADDR` 0x70
- #define `RTC_DATA` 0x71
- #define `RTC_SEC` 0
- #define `RTC_ALERTSEC` 1
- #define `RTC_MIN` 2
- #define `RTC_ALERTMIN` 3
- #define `RTC_HOUR` 4
- #define `RTC_ALERTHOUR` 5
- #define `RTC_WEEKDAY` 6
- #define `RTC_DAY` 7
- #define `RTC_MONTH` 8
- #define `RTC_YEAR` 9
- #define `RTC_STATA` 10
- #define `RTC_STATB` 11
- #define `RTC_STATC` 12
- #define `RTC_STATD` 13
- #define `RTC_CENTURY` 50

## Functions

- `uint8 bcd2bin (uint8 num)`  
*Converts a bcd-value into a binary value.*
- `void rtc_init ()`  
*Initializes the global data struct.*
- `void rtc_update ()`  
*Updates the date-data in the global time struct.*

- char \* `time2str` (`time_t` timestamp, char buf[24])  
*Converts the data in the global time struct into a readable string.*

## Variables

- struct `time` `time`  
*Global time struct.*

### 5.66.1 Detailed Description

The real-time clock constants and the time struct.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.66.2 Define Documentation

#### 5.66.2.1 #define RTC\_ADDR 0x70

Referenced by `rtc_init()`, and `rtc_update()`.

#### 5.66.2.2 #define RTC\_ALERTHOUR 5

#### 5.66.2.3 #define RTC\_ALERTMIN 3

#### 5.66.2.4 #define RTC\_ALERTSEC 1

#### 5.66.2.5 #define RTC\_CENTURY 50

Referenced by `rtc_init()`, and `rtc_update()`.

#### 5.66.2.6 #define RTC\_DATA 0x71

Referenced by `rtc_init()`, and `rtc_update()`.

### 5.66.2.7 #define RTC\_DAY 7

Referenced by rtc\_init(), and rtc\_update().

### 5.66.2.8 #define RTC\_HOUR 4

Referenced by rtc\_init(), and rtc\_update().

### 5.66.2.9 #define RTC\_MIN 2

Referenced by rtc\_init(), and rtc\_update().

### 5.66.2.10 #define RTC\_MONTH 8

Referenced by rtc\_init(), and rtc\_update().

### 5.66.2.11 #define RTC\_SEC 0

Referenced by rtc\_init(), and rtc\_update().

### 5.66.2.12 #define RTC\_STATA 10

### 5.66.2.13 #define RTC\_STATB 11

### 5.66.2.14 #define RTC\_STATC 12

### 5.66.2.15 #define RTC\_STATD 13

### 5.66.2.16 #define RTC\_WEEKDAY 6

### 5.66.2.17 #define RTC\_YEAR 9

Referenced by rtc\_init(), and rtc\_update().

## 5.66.3 Function Documentation

### 5.66.3.1 uint8 bcd2bin ( uint8 *num* )

Converts a bcd-value into a binary value.

#### Parameters

*num* the bcd-value to convert

#### Returns

the corresponding binary value

Referenced by calculate\_weekday(), threadA(), and threadB().

### 5.66.3.2 void rtc\_init ( )

Initializes the global data struct.

References calculate\_weekday(), inb(), outb(), RTC\_ADDR, RTC\_CENTURY, RTC\_DATA, RTC\_DAY, RTC\_HOUR, RTC\_MIN, RTC\_MONTH, RTC\_SEC, and RTC\_YEAR.

Referenced by init\_vmonitors().

### 5.66.3.3 void rtc\_update ( )

Updates the date-data in the global time struct.

References calculate\_weekday(), inb(), outb(), RTC\_ADDR, RTC\_CENTURY, RTC\_DATA, RTC\_DAY, RTC\_HOUR, RTC\_MIN, RTC\_MONTH, RTC\_SEC, and RTC\_YEAR.

Referenced by timer\_handler().

### 5.66.3.4 char\* time2str ( time\_t timestamp, char buf[24] )

Converts the data in the global time struct into a readable string.

#### Parameters

*timestamp* the actual time struct to be converted

*buf* the buffer for the date-string

#### Returns

the pointer to the same buffer now filled with the date string

References time::century, time::day, time::hour, itoa, time::min, time::month, time::sec, strcpy(), time::weekday, and time::year.

## 5.66.4 Variable Documentation

### 5.66.4.1 struct time time

Global time struct.

## 5.67 src/kernel/io/io\_sound.c File Reference

Sound driver.

```
#include "../include/types.h"
#include "io_sound.h"
#include "io_timer.h"
#include "io.h"
```

## Functions

- void [start\\_beep](#) (uint32 freq)
- void [end\\_beep](#) ()

## Variables

- const int [notes](#) [12][8]
- const char \* [note\\_names](#) [12]

### 5.67.1 Detailed Description

Sound driver.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.67.2 Function Documentation

#### 5.67.2.1 void [end\\_beep](#) ( )

References [inb\(\)](#), [IO\\_SPEAKER\\_PORT](#), and [outb\(\)](#).

Referenced by [shell\\_cmd\\_pong\(\)](#), and [snake\(\)](#).

#### 5.67.2.2 void [start\\_beep](#) ( uint32 *freq* )

References [inb\(\)](#), [IO\\_SPEAKER\\_PORT](#), [outb\(\)](#), [PIT\\_CONTROL](#), [PIT\\_COUNTER2](#), and [PIT\\_SOUND\\_CMD](#).

Referenced by [shell\\_cmd\\_pong\(\)](#), and [snake\(\)](#).

### 5.67.3 Variable Documentation

#### 5.67.3.1 const char\* [note\\_names](#)[12]

#### Initial value:

```
{
    "C",
    "CSH",
    "D",
    "DSH",
    "E",
    "F",
    "FSH",
    "G",
    "GSH",
    "A",
    "ASH",
    "B"
}
```

### 5.67.3.2 const int notes[12][8]

**Initial value:**

```
{
    {16, 33, 65, 131, 262, 523, 1046, 2093},
    {17, 35, 69, 139, 277, 554, 1109, 2217},
    {18, 37, 73, 147, 294, 587, 1175, 2349},
    {19, 39, 78, 155, 311, 622, 1244, 2489},
    {21, 41, 82, 165, 330, 659, 1328, 2637},
    {22, 44, 87, 175, 349, 698, 1397, 2794},
    {23, 46, 92, 185, 370, 740, 1480, 2960},
    {24, 49, 98, 196, 392, 784, 1568, 3136},
    {26, 52, 104, 208, 415, 831, 1661, 3322},
    {27, 55, 110, 220, 440, 880, 1760, 3520},
    {29, 58, 116, 233, 466, 932, 1865, 3729},
    {31, 62, 123, 245, 494, 988, 1975, 3951},
}
```

## 5.68 src/kernel/io/io\_sound.h File Reference

Sound driver header.

### Defines

- #define [IO\\_SPEAKER\\_PORT](#) 0x61
- #define [NOTE\\_C](#) 0
- #define [NOTE\\_CSH](#) 1
- #define [NOTE\\_D](#) 2
- #define [NOTE\\_DSH](#) 3
- #define [NOTE\\_E](#) 4
- #define [NOTE\\_F](#) 5
- #define [NOTE\\_FSH](#) 6
- #define [NOTE\\_G](#) 7
- #define [NOTE\\_GSH](#) 8
- #define [NOTE\\_A](#) 9
- #define [NOTE\\_ASH](#) 10
- #define [NOTE\\_B](#) 11
- #define [NOTE\\_C0](#) notes[0][0]
- #define [NOTE\\_C1](#) notes[0][1]



- #define NOTE\_C2 notes[0][2]
- #define NOTE\_C3 notes[0][3]
- #define NOTE\_C4 notes[0][4]
- #define NOTE\_C5 notes[0][5]
- #define NOTE\_C6 notes[0][6]
- #define NOTE\_C7 notes[0][7]
- #define NOTE\_CSH0 notes[1][0]
- #define NOTE\_CSH1 notes[1][1]
- #define NOTE\_CSH2 notes[1][2]
- #define NOTE\_CSH3 notes[1][3]
- #define NOTE\_CSH4 notes[1][4]
- #define NOTE\_CSH5 notes[1][5]
- #define NOTE\_CSH6 notes[1][6]
- #define NOTE\_CSH7 notes[1][7]
- #define NOTE\_D0 notes[2][0]
- #define NOTE\_D1 notes[2][1]
- #define NOTE\_D2 notes[2][2]
- #define NOTE\_D3 notes[2][3]
- #define NOTE\_D4 notes[2][4]
- #define NOTE\_D5 notes[2][5]
- #define NOTE\_D6 notes[2][6]
- #define NOTE\_D7 notes[2][7]
- #define NOTE\_DSH0 notes[3][0]
- #define NOTE\_DSH1 notes[3][1]
- #define NOTE\_DSH2 notes[3][2]
- #define NOTE\_DSH3 notes[3][3]
- #define NOTE\_DSH4 notes[3][4]
- #define NOTE\_DSH5 notes[3][5]
- #define NOTE\_DSH6 notes[3][6]
- #define NOTE\_DSH7 notes[3][7]
- #define NOTE\_E0 notes[4][0]
- #define NOTE\_E1 notes[4][1]
- #define NOTE\_E2 notes[4][2]
- #define NOTE\_E3 notes[4][3]
- #define NOTE\_E4 notes[4][4]
- #define NOTE\_E5 notes[4][5]
- #define NOTE\_E6 notes[4][6]
- #define NOTE\_E7 notes[4][7]
- #define NOTE\_F0 notes[5][0]
- #define NOTE\_F1 notes[5][1]
- #define NOTE\_F2 notes[5][2]
- #define NOTE\_F3 notes[5][3]
- #define NOTE\_F4 notes[5][4]
- #define NOTE\_F5 notes[5][5]
- #define NOTE\_F6 notes[5][6]
- #define NOTE\_F7 notes[5][7]
- #define NOTE\_FSH0 notes[6][0]
- #define NOTE\_FSH1 notes[6][1]
- #define NOTE\_FSH2 notes[6][2]
- #define NOTE\_FSH3 notes[6][3]

- #define `NOTE_FSH4` `notes`[6][4]
- #define `NOTE_FSH5` `notes`[6][5]
- #define `NOTE_FSH6` `notes`[6][6]
- #define `NOTE_FSH7` `notes`[6][7]
- #define `NOTE_G0` `notes`[7][0]
- #define `NOTE_G1` `notes`[7][1]
- #define `NOTE_G2` `notes`[7][2]
- #define `NOTE_G3` `notes`[7][3]
- #define `NOTE_G4` `notes`[7][4]
- #define `NOTE_G5` `notes`[7][5]
- #define `NOTE_G6` `notes`[7][6]
- #define `NOTE_G7` `notes`[7][7]
- #define `NOTE_GSH0` `notes`[8][0]
- #define `NOTE_GSH1` `notes`[8][1]
- #define `NOTE_GSH2` `notes`[8][2]
- #define `NOTE_GSH3` `notes`[8][3]
- #define `NOTE_GSH4` `notes`[8][4]
- #define `NOTE_GSH5` `notes`[8][5]
- #define `NOTE_GSH6` `notes`[8][6]
- #define `NOTE_GSH7` `notes`[8][7]
- #define `NOTE_A0` `notes`[9][0]
- #define `NOTE_A1` `notes`[9][1]
- #define `NOTE_A2` `notes`[9][2]
- #define `NOTE_A3` `notes`[9][3]
- #define `NOTE_A4` `notes`[9][4]
- #define `NOTE_A5` `notes`[9][5]
- #define `NOTE_A6` `notes`[9][6]
- #define `NOTE_A7` `notes`[9][7]
- #define `NOTE_ASH0` `notes`[10][0]
- #define `NOTE_ASH1` `notes`[10][1]
- #define `NOTE_ASH2` `notes`[10][2]
- #define `NOTE_ASH3` `notes`[10][3]
- #define `NOTE_ASH4` `notes`[10][4]
- #define `NOTE_ASH5` `notes`[10][5]
- #define `NOTE_ASH6` `notes`[10][6]
- #define `NOTE_ASH7` `notes`[10][7]
- #define `NOTE_B0` `notes`[11][0]
- #define `NOTE_B1` `notes`[11][1]
- #define `NOTE_B2` `notes`[11][2]
- #define `NOTE_B3` `notes`[11][3]
- #define `NOTE_B4` `notes`[11][4]
- #define `NOTE_B5` `notes`[11][5]
- #define `NOTE_B6` `notes`[11][6]
- #define `NOTE_B7` `notes`[11][7]

## Variables

- const int `notes` [12][8]
- const char \* `note_names` [12]

## 5.68.1 Detailed Description

Sound driver header.

### Author

Dmitriy Traytel

### LastChangedBy:

dtraytel

### Version

### Rev:

12

## 5.68.2 Define Documentation

### 5.68.2.1 #define IO\_SPEAKER\_PORT 0x61

Referenced by end\_beep(), and start\_beep().

### 5.68.2.2 #define NOTE\_A 9

Referenced by synth().

### 5.68.2.3 #define NOTE\_A0 notes[9][0]

### 5.68.2.4 #define NOTE\_A1 notes[9][1]

### 5.68.2.5 #define NOTE\_A2 notes[9][2]

### 5.68.2.6 #define NOTE\_A3 notes[9][3]

### 5.68.2.7 #define NOTE\_A4 notes[9][4]

### 5.68.2.8 #define NOTE\_A5 notes[9][5]

### 5.68.2.9 #define NOTE\_A6 notes[9][6]

### 5.68.2.10 #define NOTE\_A7 notes[9][7]

### 5.68.2.11 #define NOTE\_ASH 10

Referenced by synth().

5.68.2.12 **#define NOTE\_ASH0 notes[10][0]**

5.68.2.13 **#define NOTE\_ASH1 notes[10][1]**

5.68.2.14 **#define NOTE\_ASH2 notes[10][2]**

5.68.2.15 **#define NOTE\_ASH3 notes[10][3]**

5.68.2.16 **#define NOTE\_ASH4 notes[10][4]**

5.68.2.17 **#define NOTE\_ASH5 notes[10][5]**

5.68.2.18 **#define NOTE\_ASH6 notes[10][6]**

5.68.2.19 **#define NOTE\_ASH7 notes[10][7]**

5.68.2.20 **#define NOTE\_B 11**

Referenced by synth().

5.68.2.21 **#define NOTE\_B0 notes[11][0]**

5.68.2.22 **#define NOTE\_B1 notes[11][1]**

5.68.2.23 **#define NOTE\_B2 notes[11][2]**

5.68.2.24 **#define NOTE\_B3 notes[11][3]**

5.68.2.25 **#define NOTE\_B4 notes[11][4]**

5.68.2.26 **#define NOTE\_B5 notes[11][5]**

5.68.2.27 **#define NOTE\_B6 notes[11][6]**

5.68.2.28 **#define NOTE\_B7 notes[11][7]**

5.68.2.29 **#define NOTE\_C 0**

Referenced by synth().

5.68.2.30 `#define NOTE_C0 notes[0][0]`

5.68.2.31 `#define NOTE_C1 notes[0][1]`

5.68.2.32 `#define NOTE_C2 notes[0][2]`

5.68.2.33 `#define NOTE_C3 notes[0][3]`

5.68.2.34 `#define NOTE_C4 notes[0][4]`

5.68.2.35 `#define NOTE_C5 notes[0][5]`

5.68.2.36 `#define NOTE_C6 notes[0][6]`

5.68.2.37 `#define NOTE_C7 notes[0][7]`

5.68.2.38 `#define NOTE_CSH 1`

Referenced by synth().

5.68.2.39 `#define NOTE_CSH0 notes[1][0]`

5.68.2.40 `#define NOTE_CSH1 notes[1][1]`

5.68.2.41 `#define NOTE_CSH2 notes[1][2]`

5.68.2.42 `#define NOTE_CSH3 notes[1][3]`

5.68.2.43 `#define NOTE_CSH4 notes[1][4]`

5.68.2.44 `#define NOTE_CSH5 notes[1][5]`

5.68.2.45 `#define NOTE_CSH6 notes[1][6]`

5.68.2.46 `#define NOTE_CSH7 notes[1][7]`

5.68.2.47 `#define NOTE_D 2`

Referenced by synth().

5.68.2.48 `#define NOTE_D0 notes[2][0]`

5.68.2.49 `#define NOTE_D1 notes[2][1]`

5.68.2.50 `#define NOTE_D2 notes[2][2]`

5.68.2.51 `#define NOTE_D3 notes[2][3]`

5.68.2.52 `#define NOTE_D4 notes[2][4]`

5.68.2.53 `#define NOTE_D5 notes[2][5]`

5.68.2.54 `#define NOTE_D6 notes[2][6]`

5.68.2.55 `#define NOTE_D7 notes[2][7]`

5.68.2.56 `#define NOTE_DSH 3`

Referenced by synth().

5.68.2.57 `#define NOTE_DSH0 notes[3][0]`

5.68.2.58 `#define NOTE_DSH1 notes[3][1]`

5.68.2.59 `#define NOTE_DSH2 notes[3][2]`

5.68.2.60 `#define NOTE_DSH3 notes[3][3]`

5.68.2.61 `#define NOTE_DSH4 notes[3][4]`

5.68.2.62 `#define NOTE_DSH5 notes[3][5]`

5.68.2.63 `#define NOTE_DSH6 notes[3][6]`

5.68.2.64 `#define NOTE_DSH7 notes[3][7]`

5.68.2.65 `#define NOTE_E 4`

Referenced by synth().

5.68.2.66 `#define NOTE_E0 notes[4][0]`

5.68.2.67 `#define NOTE_E1 notes[4][1]`

5.68.2.68 `#define NOTE_E2 notes[4][2]`

5.68.2.69 `#define NOTE_E3 notes[4][3]`

5.68.2.70 `#define NOTE_E4 notes[4][4]`

5.68.2.71 `#define NOTE_E5 notes[4][5]`

5.68.2.72 `#define NOTE_E6 notes[4][6]`

5.68.2.73 `#define NOTE_E7 notes[4][7]`

5.68.2.74 `#define NOTE_F 5`

Referenced by synth().

5.68.2.75 `#define NOTE_F0 notes[5][0]`

5.68.2.76 `#define NOTE_F1 notes[5][1]`

5.68.2.77 `#define NOTE_F2 notes[5][2]`

5.68.2.78 `#define NOTE_F3 notes[5][3]`

5.68.2.79 `#define NOTE_F4 notes[5][4]`

5.68.2.80 `#define NOTE_F5 notes[5][5]`

5.68.2.81 `#define NOTE_F6 notes[5][6]`

5.68.2.82 `#define NOTE_F7 notes[5][7]`

5.68.2.83 `#define NOTE_FSH 6`

Referenced by synth().

5.68.2.84 #define NOTE\_FSH0 notes[6][0]

5.68.2.85 #define NOTE\_FSH1 notes[6][1]

5.68.2.86 #define NOTE\_FSH2 notes[6][2]

5.68.2.87 #define NOTE\_FSH3 notes[6][3]

5.68.2.88 #define NOTE\_FSH4 notes[6][4]

5.68.2.89 #define NOTE\_FSH5 notes[6][5]

5.68.2.90 #define NOTE\_FSH6 notes[6][6]

5.68.2.91 #define NOTE\_FSH7 notes[6][7]

5.68.2.92 #define NOTE\_G 7

Referenced by synth().

5.68.2.93 #define NOTE\_G0 notes[7][0]

5.68.2.94 #define NOTE\_G1 notes[7][1]

5.68.2.95 #define NOTE\_G2 notes[7][2]

5.68.2.96 #define NOTE\_G3 notes[7][3]

5.68.2.97 #define NOTE\_G4 notes[7][4]

5.68.2.98 #define NOTE\_G5 notes[7][5]

5.68.2.99 #define NOTE\_G6 notes[7][6]

5.68.2.100 #define NOTE\_G7 notes[7][7]

5.68.2.101 #define NOTE\_GSH 8

Referenced by synth().



5.68.2.102 `#define NOTE_GSH0 notes[8][0]`

5.68.2.103 `#define NOTE_GSH1 notes[8][1]`

5.68.2.104 `#define NOTE_GSH2 notes[8][2]`

5.68.2.105 `#define NOTE_GSH3 notes[8][3]`

5.68.2.106 `#define NOTE_GSH4 notes[8][4]`

5.68.2.107 `#define NOTE_GSH5 notes[8][5]`

5.68.2.108 `#define NOTE_GSH6 notes[8][6]`

5.68.2.109 `#define NOTE_GSH7 notes[8][7]`

### 5.68.3 Variable Documentation

5.68.3.1 `const char* note_names[12]`

5.68.3.2 `const int notes[12][8]`

## 5.69 src/kernel/io/io\_timer.c File Reference

Timer-handler.

```
#include "../include/const.h"
#include "../include/limits.h"
#include "../include/types.h"
#include "../include/stdio.h"
#include "../io/io.h"
#include "../io/io_timer.h"
#include "../io/io_rtc.h"
#include "../io/io_virtual.h"
#include "../pm/pm_main.h"
```

### Functions

- [uint32 timer\\_handler](#) ([uint32](#) context)  
*Handles a timer-interrupt by incrementing the ticks-counter and calling the PM.*
- [sint32 get\\_ticks](#) ()  
*Returns the timer ticks.*
- void [sleep](#) ([sint32 num](#))  
*Sleeps num seconds.*
- void [sleep\\_ticks](#) ([sint32 num](#))

*Sleeps num ticks.*

- void `timer_init` (`sint32` freq)

*Initializes the timer with the given frequency freq.*

### 5.69.1 Detailed Description

Timer-handler.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.69.2 Function Documentation

#### 5.69.2.1 `sint32 get_ticks ( )`

Returns the timer ticks.

#### Returns

ticks since system boot or last timer wraparound

Referenced by `main()`, `shell_cmd_bf()`, and `snake()`.

#### 5.69.2.2 `void sleep ( sint32 num )`

Sleeps num seconds.

References `FREQUENCY`, `halt()`, `reactivate_pic()`, `set_interrupts()`, `SINT32_MAX`, and `sleep()`.

Referenced by `sleep()`, `sleep_test()`, and `sleep_ticks()`.

#### 5.69.2.3 `void sleep_ticks ( sint32 num )`

Sleeps num ticks.

References `halt()`, `reactivate_pic()`, `set_interrupts()`, `SINT32_MAX`, and `sleep()`.

Referenced by `monitor_cputc()`.

#### 5.69.2.4 uint32 timer\_handler ( uint32 context )

Handles a timer-interrupt by incrementing the ticks-counter and calling the PM.

References `get_active_virt_monitor()`, `pm_schedule()`, `rtc_update()`, `SINT32_MAX`, and `update_virt_monitor()`.

Referenced by `irq_handler()`.

#### 5.69.2.5 void timer\_init ( sint32 freq )

Initializes the timer with the given frequency `freq`.

References `outb()`, `PIT_CONTROL`, `PIT_COUNTER0`, and `PIT_INIT_CMD`.

## 5.70 src/kernel/io/io\_timer.h File Reference

header file for the timer-handler

### Defines

- #define `PIT_COUNTER0` 0x40
- #define `PIT_COUNTER1` 0x41
- #define `PIT_COUNTER2` 0x42
- #define `PIT_CONTROL` 0x43
- #define `PIT_INIT_CMD` 0x36
- #define `PIT_SOUND_CMD` 0xB6

### 5.70.1 Detailed Description

header file for the timer-handler

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.70.2 Define Documentation

#### 5.70.2.1 #define PIT\_CONTROL 0x43

Referenced by `start_beep()`, and `timer_init()`.

### 5.70.2.2 `#define PIT_COUNTER0 0x40`

Referenced by `timer_init()`.

### 5.70.2.3 `#define PIT_COUNTER1 0x41`

### 5.70.2.4 `#define PIT_COUNTER2 0x42`

Referenced by `start_beep()`.

### 5.70.2.5 `#define PIT_INIT_CMD 0x36`

Referenced by `timer_init()`.

### 5.70.2.6 `#define PIT_SOUND_CMD 0xB6`

Referenced by `start_beep()`.

## 5.71 `src/kernel/io/io_virtual.c` File Reference

Function used by virtual outputs.

```
#include "../include/stdlib.h"
#include "../include/stdio.h"
#include "../include/string.h"
#include "../include/util.h"
#include "../include/debug.h"
#include "../include/assert.h"
#include "../include/const.h"
#include "../pm/pm_main.h"
#include "io.h"
#include "io_rtc.h"
#include "io_virtual.h"
```

### Functions

- void `new_virt_monitor` (`virt_monitor` \*vm, `uint32` pid)  
*Creates a new virtual output.*
- void `free_virt_monitor` (`virt_monitor` \*vm)  
*Deletes a virtual output.*
- void `virt_monitor_cputc` (`virt_monitor` \*vm, char ch, `uint8` fg, `uint8` bg)  
*Writes a colored character to the virtual monitor.*

- `uint8 get_color_tag` (char \*str)  
*Decodes the color tag on the beginning of the given string as a color.*
- `int virt_monitor_cputs` (virt\_monitor \*vm, char \*str, uint8 fg, uint8 bg)  
*Writes a colored null-terminated string to the virtual monitor.*
- `void virt_monitor_putc` (virt\_monitor \*vm, char ch)  
*Writes a character to the virtual monitor.*
- `int virt_monitor_puts` (virt\_monitor \*vm, char \*str)  
*Writes a null-terminated string to the virtual monitor.*
- `void virt_cursor_move` (virt\_monitor \*vm, uint8 dir)  
*Moves the cursor in the given direction.*
- `void update_virt_monitor` (virt\_monitor \*vm)  
*Updates the virtual monitor.*
- `void virt_monitor_scrollup` (virt\_monitor \*vm)  
*Scrolls a virtual monitor up on request.*
- `void virt_monitor_scrolldown` (virt\_monitor \*vm)  
*Scrolls a virtual monitor down on request.*
- `void virt_monitor_invert` (virt\_monitor \*vm)  
*Inverts a virtual monitor.*

### 5.71.1 Detailed Description

Function used by virtual outputs.

**Author**

Dmitriy Traytel

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

## 5.71.2 Function Documentation

### 5.71.2.1 void free\_virt\_monitor ( virt\_monitor \* *vm* )

Deletes a virtual output.

#### Parameters

*\*vm* virtual monitor to be deleted

References active\_monitor, virt\_monitor::begin, free, maxvmonitor, memset(), virt\_monitor::name, virt\_monitor::pid, pm\_get\_proc(), pm\_set\_focus\_proc(), process\_t::vmonitor, and vmonitors.

Referenced by pm\_destroy\_thread().

### 5.71.2.2 uint8 get\_color\_tag ( char \* *str* )

Decodes the color tag on the beginning of the given string as a color.

#### Parameters

*str* string starting with a tag

#### Returns

the decoded

References memcpy, and strcmp.

Referenced by virt\_monitor\_cputs().

### 5.71.2.3 void new\_virt\_monitor ( virt\_monitor \* *vm*, uint32 *pid* )

Creates a new virtual output.

#### Parameters

*\*vm* pointer to the virtual monitor struct, where the data should be written to

*pid* PID of the appropriate process

References ASSERT, virt\_monitor::begin, virt\_monitor::disable\_refresh, mallocn, virt\_monitor::name, name, virt\_monitor::offset, virt\_monitor::pid, virt\_monitor::scrolldown\_limit, virt\_monitor::scrollup\_limit, virt\_monitor::size, VIRTUAL\_MONITOR\_SIZE, virt\_monitor::vis\_begin, and VMONITOR\_HEIGHT.

Referenced by start\_vmonitor().

### 5.71.2.4 void update\_virt\_monitor ( virt\_monitor \* *vm* )

Updates the virtual monitor.

#### Parameters

*\*vm* virtual monitor to be updated

References `virt_monitor::begin`, `BLACK`, `CURSOR_OFFSET`, `virt_monitor::disable_refresh`, `get_active_virt_monitor_name()`, `GREEN`, `memcpy`, `monitor_cputs()`, `MONITOR_START`, `virt_monitor::offset`, `outb()`, `set_disp()`, `virt_monitor::size`, `VGA_DISPLAY`, `virt_monitor::vis_begin`, and `VMONITOR_HEIGHT`.

Referenced by `timer_handler()`.

#### 5.71.2.5 void virt\_cursor\_move ( virt\_monitor \* vm, uint8 dir )

Moves the cursor in the given direction.

##### Parameters

*vm* virtual monitor, on which the cursor shall be moved  
*dir* 0=UP, 1=DOWN, 2=LEFT, 3=RIGHT

References `CURSOR_OFFSET`, `virt_monitor::disable_refresh`, `LINE_WIDTH`, `virt_monitor::offset`, `outb()`, and `VMONITOR_HEIGHT`.

Referenced by `kb_handler()`.

#### 5.71.2.6 void virt\_monitor\_cputc ( virt\_monitor \* vm, char ch, uint8 fg, uint8 bg )

Writes a colored character to the virtual monitor.

##### Parameters

*vm* virtual monitor, on which the character shall be written  
*ch* character to be written  
*fg* foreground-color  
*bg* background color

References `virt_monitor::begin`, `LINE_WIDTH`, `virt_monitor::offset`, `virt_monitor::scrollup_limit`, `virt_monitor::size`, `virt_monitor_cputc()`, `virt_monitor::vis_begin`, and `VMONITOR_HEIGHT`.

Referenced by `cputchar()`, `virt_monitor_cputc()`, `virt_monitor_cputs()`, and `virt_monitor_puts()`.

#### 5.71.2.7 int virt\_monitor\_cputs ( virt\_monitor \* vm, char \* str, uint8 fg, uint8 bg )

Writes a colored null-terminated string to the virtual monitor.

##### Parameters

*vm* virtual monitor, on which the string shall be written  
*str* pointer to the string  
*fg* foreground-color  
*bg* background color

References `get_color_tag()`, `virt_monitor_cputc()`, and `WHITE`.

Referenced by `cputs()`, `puts()`, and `virt_monitor_puts()`.

### 5.71.2.8 void virt\_monitor\_invert ( virt\_monitor \* *vm* )

Inverts a virtual monitor.

#### Parameters

*\*vm* virtual monitor to invert

References virt\_monitor::vis\_begin, and VMONITOR\_HEIGHT.

Referenced by kb\_handler().

### 5.71.2.9 void virt\_monitor\_putc ( virt\_monitor \* *vm*, char *ch* )

Writes a character to the virtual monitor.

#### Parameters

*\*vm* virtual monitor, on which the character shall be written

*ch* character to be written

References BLACK, virt\_monitor\_cputc(), and WHITE.

Referenced by kb\_handler(), and putchar().

### 5.71.2.10 int virt\_monitor\_puts ( virt\_monitor \* *vm*, char \* *str* )

Writes a null-terminated string to the virtual monitor.

#### Parameters

*\*vm* virtual monitor, on which the string shall be written

*\*str* pointer to the string

References BLACK, virt\_monitor\_cputs(), and WHITE.

Referenced by aprintf(), and dev\_stdout\_write().

### 5.71.2.11 void virt\_monitor\_scrolldown ( virt\_monitor \* *vm* )

Scrolls a virtual monitor down on request.

#### Parameters

*\*vm* virtual monitor to scroll

References virt\_monitor::begin, virt\_monitor::scrolldown\_limit, virt\_monitor::scrollup\_limit, virt\_monitor::size, and virt\_monitor::vis\_begin.

Referenced by kb\_handler().



### 5.71.2.12 void virt\_monitor\_scrollup ( virt\_monitor \* vm )

Scrolls a virtual monitor up on request.

#### Parameters

\**vm* virtual monitor to scroll

References virt\_monitor::begin, LINE\_WIDTH, virt\_monitor::scrolldown\_limit, virt\_monitor::scrollup\_limit, virt\_monitor::size, and virt\_monitor::vis\_begin.

Referenced by kb\_handler().

## 5.72 src/kernel/io/io\_virtual.h File Reference

Header for the virtual monitor structure.

```
#include "../include/types.h"
```

### Data Structures

- struct [virt\\_monitor](#)  
*Structure that represents a virtual monitor.*

### Defines

- #define [VIRTUAL\\_MONITOR\\_SIZE](#) 160000
- #define [LINE\\_WIDTH](#) 80
- #define [VMONITOR\\_HEIGHT](#) 0xF00
- #define [MONITOR\\_START](#) 0xB80A0
- #define [CURSOR\\_OFFSET](#) 80

### Functions

- [virt\\_monitor \\* get\\_active\\_virt\\_monitor \(\)](#)  
*Returns the pointer to the active virtual monitor.*
- [char \\* get\\_active\\_virt\\_monitor\\_name \(\)](#)  
*Returns the name of the active virtual monitor.*
- [void new\\_virt\\_monitor \(virt\\_monitor \\*vm, uint32 pid\)](#)  
*Creates a new virtual output.*
- [void free\\_virt\\_monitor \(virt\\_monitor \\*vm\)](#)  
*Deletes a virtual output.*
- [void update\\_virt\\_monitor \(\)](#)
- [void virt\\_cursor\\_move \(virt\\_monitor \\*vm, uint8 dir\)](#)

*Moves the cursor in the given direction.*

- void `virt_monitor_scrolldown` (`virt_monitor *vm`)  
*Scrolls a virtual monitor down on request.*
- void `virt_monitor_scrollup` (`virt_monitor *vm`)  
*Scrolls a virtual monitor up on request.*
- void `virt_monitor_invert` (`virt_monitor *vm`)  
*Inverts a virtual monitor.*
- void `virt_printf` (`virt_monitor vm`, `char *fmt`,...)
- void `virt_monitor_cputc` (`virt_monitor *vm`, `char ch`, `uint8 fg`, `uint8 bg`)  
*Writes a colored character to the virtual monitor.*
- int `virt_monitor_cputs` (`virt_monitor *vm`, `char *str`, `uint8 fg`, `uint8 bg`)  
*Writes a colored null-terminated string to the virtual monitor.*
- void `virt_monitor_putc` (`virt_monitor *vm`, `char ch`)  
*Writes a character to the virtual monitor.*
- int `virt_monitor_puts` (`virt_monitor *vm`, `char *str`)  
*Writes a null-terminated string to the virtual monitor.*
- void `init_vmonitors` ()  
*Initializes the virtual monitors.*
- `virt_monitor * start_vmonitor` (`char *name`, `uint32 pid`)  
*Initializes and registers a virtual monitor.*
- void `switch_monitor_down` ()  
*Switches to the previous virtual monitor.*
- void `switch_monitor_up` ()  
*Switches to the next virtual monitor.*

## Variables

- `uint16 maxvmonitor`
- `uint16 active_monitor`
- `virt_monitor * vmonitors`

### 5.72.1 Detailed Description

Header for the virtual monitor structure.

#### Author

Dmitriy Traytel

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

## 5.72.2 Define Documentation

### 5.72.2.1 #define CURSOR\_OFFSET 80

Referenced by update\_virt\_monitor(), and virt\_cursor\_move().

### 5.72.2.2 #define LINE\_WIDTH 80

Referenced by virt\_cursor\_move(), virt\_monitor\_cputc(), and virt\_monitor\_scrollup().

### 5.72.2.3 #define MONITOR\_START 0xB80A0

Referenced by update\_virt\_monitor().

### 5.72.2.4 #define VIRTUAL\_MONITOR\_SIZE 160000

Referenced by new\_virt\_monitor().

### 5.72.2.5 #define VMONITOR\_HEIGHT 0xF00

Referenced by new\_virt\_monitor(), update\_virt\_monitor(), virt\_cursor\_move(), virt\_monitor\_cputc(), and virt\_monitor\_invert().

## 5.72.3 Function Documentation

### 5.72.3.1 void free\_virt\_monitor ( virt\_monitor \* *vm* )

Deletes a virtual output.

**Parameters**

*\*vm* virtual monitor to be deleted

References active\_monitor, virt\_monitor::begin, free, maxvmonitor, memset(), virt\_monitor::name, virt\_monitor::pid, pm\_get\_proc(), pm\_set\_focus\_proc(), process\_t::vmonitor, and vmonitors.

Referenced by pm\_destroy\_thread().

### 5.72.3.2 `virt_monitor* get_active_virt_monitor ( )`

Returns the pointer to the active virtual monitor.

#### Returns

pointer to the active virtual monitor

References `active_monitor`.

Referenced by `dev_framebuffer_close()`, `dev_framebuffer_open()`, `kb_handler()`, `pm_init()`, `shell_cmd_snapshot()`, and `timer_handler()`.

### 5.72.3.3 `char* get_active_virt_monitor_name ( )`

Returns the name of the active virtual monitor.

#### Returns

pointer to the name string

References `active_monitor`, `memcpy`, `virt_monitor::name`, `name`, and `time2str`.

Referenced by `update_virt_monitor()`.

### 5.72.3.4 `void init_vmonitors ( )`

Initializes the virtual monitors.

References `ASSERT`, `callocn()`, `num_vmonitor_limit`, `rtc_init()`, and `start_vmonitor()`.

Referenced by `main()`.

### 5.72.3.5 `void new_virt_monitor ( virt_monitor * vm, uint32 pid )`

Creates a new virtual output.

#### Parameters

***\*vm*** pointer to the virtual monitor struct, where the data should be written to

***pid*** PID of the appropriate process

References `ASSERT`, `virt_monitor::begin`, `virt_monitor::disable_refresh`, `mallocn`, `virt_monitor::name`, `name`, `virt_monitor::offset`, `virt_monitor::pid`, `virt_monitor::scrolldown_limit`, `virt_monitor::scrollup_limit`, `virt_monitor::size`, `VIRTUAL_MONITOR_SIZE`, `virt_monitor::vis_begin`, and `VMONITOR_HEIGHT`.

Referenced by `start_vmonitor()`.

### 5.72.3.6 `virt_monitor* start_vmonitor ( char * name, uint32 pid )`

Initializes and registers a virtual monitor.

**Parameters**

*\*name* name of the new virtual monitor  
*pid* PID of the appropriate process

**Returns**

pointer to the new virtual monitor

References active\_monitor, maxvmonitor, memcpy, memset(), virt\_monitor::name, new\_virt\_monitor(), and strlen.

Referenced by init\_vmonitors(), and pm\_create\_thread().

**5.72.3.7 void switch\_monitor\_down ( )**

Switches to the previous virtual monitor.

References active\_monitor, maxvmonitor, and pm\_set\_focus\_proc().

Referenced by do\_tests().

**5.72.3.8 void switch\_monitor\_up ( )**

Switches to the next virtual monitor.

References active\_monitor, maxvmonitor, and pm\_set\_focus\_proc().

Referenced by do\_tests().

**5.72.3.9 void update\_virt\_monitor ( )****5.72.3.10 void virt\_cursor\_move ( virt\_monitor \* *vm*, uint8 *dir* )**

Moves the cursor in the given direction.

**Parameters**

*\*vm* virtual monitor, on which the cursor shall be moved  
*dir* 0=UP, 1=DOWN, 2=LEFT, 3=RIGHT

References CURSOR\_OFFSET, virt\_monitor::disable\_refresh, LINE\_WIDTH, virt\_monitor::offset, outb(), and VMONITOR\_HEIGHT.

Referenced by kb\_handler().

**5.72.3.11 void virt\_monitor\_cputc ( virt\_monitor \* *vm*, char *ch*, uint8 *fg*, uint8 *bg* )**

Writes a colored character to the virtual monitor.

**Parameters**

*\*vm* virtual monitor, on which the character shall be written  
*ch* character to be written

*fg* foreground-color  
*bg* background color

References `virt_monitor::begin`, `LINE_WIDTH`, `virt_monitor::offset`, `virt_monitor::scrollup_limit`, `virt_monitor::size`, `virt_monitor_cputc()`, `virt_monitor::vis_begin`, and `VMONITOR_HEIGHT`.

Referenced by `cputchar()`, `virt_monitor_cputc()`, `virt_monitor_cputs()`, and `virt_monitor_putc()`.

#### 5.72.3.12 `int virt_monitor_cputs ( virt_monitor * vm, char * str, uint8 fg, uint8 bg )`

Writes a colored null-terminated string to the virtual monitor.

##### Parameters

*\*vm* virtual monitor, on which the string shall be written  
*\*str* pointer to the string  
*fg* foreground-color  
*bg* background color

References `get_color_tag()`, `virt_monitor_cputc()`, and `WHITE`.

Referenced by `cputs()`, `puts()`, and `virt_monitor_puts()`.

#### 5.72.3.13 `void virt_monitor_invert ( virt_monitor * vm )`

Inverts a virtual monitor.

##### Parameters

*\*vm* virtual monitor to invert

References `virt_monitor::vis_begin`, and `VMONITOR_HEIGHT`.

Referenced by `kb_handler()`.

#### 5.72.3.14 `void virt_monitor_putc ( virt_monitor * vm, char ch )`

Writes a character to the virtual monitor.

##### Parameters

*\*vm* virtual monitor, on which the character shall be written  
*ch* character to be written

References `BLACK`, `virt_monitor_cputc()`, and `WHITE`.

Referenced by `kb_handler()`, and `putchar()`.

#### 5.72.3.15 `int virt_monitor_puts ( virt_monitor * vm, char * str )`

Writes a null-terminated string to the virtual monitor.

### Parameters

- \**vm* virtual monitor, on which the string shall be written
- \**str* pointer to the string

References BLACK, virt\_monitor\_cputs(), and WHITE.

Referenced by aprintf(), and dev\_stdout\_write().

#### 5.72.3.16 void virt\_monitor\_scrolldown ( virt\_monitor \* *vm* )

Scrolls a virtual monitor down on request.

### Parameters

- \**vm* virtual monitor to scroll

References virt\_monitor::begin, virt\_monitor::scrolldown\_limit, virt\_monitor::scrollup\_limit, virt\_monitor::size, and virt\_monitor::vis\_begin.

Referenced by kb\_handler().

#### 5.72.3.17 void virt\_monitor\_scrollup ( virt\_monitor \* *vm* )

Scrolls a virtual monitor up on request.

### Parameters

- \**vm* virtual monitor to scroll

References virt\_monitor::begin, LINE\_WIDTH, virt\_monitor::scrolldown\_limit, virt\_monitor::scrollup\_limit, virt\_monitor::size, and virt\_monitor::vis\_begin.

Referenced by kb\_handler().

#### 5.72.3.18 void virt\_printf ( virt\_monitor *vm*, char \* *fmt*, ... )

## 5.72.4 Variable Documentation

### 5.72.4.1 uint16 active\_monitor

Referenced by free\_virt\_monitor(), get\_active\_virt\_monitor(), get\_active\_virt\_monitor\_name(), start\_vmonitor(), switch\_monitor\_down(), and switch\_monitor\_up().

### 5.72.4.2 uint16 maxvmonitor

Referenced by free\_virt\_monitor(), start\_vmonitor(), switch\_monitor\_down(), and switch\_monitor\_up().

### 5.72.4.3 virt\_monitor\* vmonitors

Referenced by free\_virt\_monitor().

## 5.73 src/kernel/io/io\_virtual\_monitors.c File Reference

Shell & monitor interface.

```
#include "../include/stdio.h"
#include "../include/stdlib.h"
#include "../include/string.h"
#include "../include/util.h"
#include "../include/assert.h"
#include "../pm/pm_main.h"
#include "io_rtc.h"
#include "io_virtual.h"
```

### Functions

- void `switch_monitor_up()`  
*Switches to the next virtual monitor.*
- void `switch_monitor_down()`  
*Switches to the previous virtual monitor.*
- void `init_vmonitors()`  
*Initializes the virtual monitors.*
- `virt_monitor *` `start_vmonitor(char *name, uint32 pid)`  
*Initializes and registers a virtual monitor.*
- `virt_monitor *` `get_active_virt_monitor()`  
*Returns the pointer to the active virtual monitor.*
- `char *` `get_active_virt_monitor_name()`  
*Returns the name of the active virtual monitor.*

### Variables

- `uint16` `active_monitor` = -1
- `uint16` `maxvmonitor` = -1
- `virt_monitor *` `vmonitors`
- `uint16` `num_vmonitor_limit` = 1000

### 5.73.1 Detailed Description

Shell & monitor interface.

#### Author

Dmitriy Traytel



**LastChangedBy:**

dtraytel

**Version****Rev:**

12

## 5.73.2 Function Documentation

### 5.73.2.1 virt\_monitor\* get\_active\_virt\_monitor ( )

Returns the pointer to the active virtual monitor.

**Returns**

pointer to the active virtual monitor

References active\_monitor.

Referenced by dev\_framebuffer\_close(), dev\_framebuffer\_open(), kb\_handler(), pm\_init(), shell\_cmd\_snapshot(), and timer\_handler().

### 5.73.2.2 char\* get\_active\_virt\_monitor\_name ( )

Returns the name of the active virtual monitor.

**Returns**

pointer to the name string

References active\_monitor, memcpy, virt\_monitor::name, name, and time2str.

Referenced by update\_virt\_monitor().

### 5.73.2.3 void init\_vmonitors ( )

Initializes the virtual monitors.

References ASSERT, callocn(), num\_vmonitor\_limit, rtc\_init(), and start\_vmonitor().

Referenced by main().

### 5.73.2.4 virt\_monitor\* start\_vmonitor ( char \* name, uint32 pid )

Initializes and registers a virtual monitor.

**Parameters**

*\*name* name of the new virtual monitor

*pid* PID of the appropriate process

**Returns**

pointer to the new virtual monitor

References `active_monitor`, `maxvmonitor`, `memcpy`, `memset`(), `virt_monitor::name`, `new_virt_monitor()`, and `strlen`.

Referenced by `init_vmonitors()`, and `pm_create_thread()`.

**5.73.2.5 void switch\_monitor\_down ( )**

Switches to the previous virtual monitor.

References `active_monitor`, `maxvmonitor`, and `pm_set_focus_proc()`.

Referenced by `do_tests()`.

**5.73.2.6 void switch\_monitor\_up ( )**

Switches to the next virtual monitor.

References `active_monitor`, `maxvmonitor`, and `pm_set_focus_proc()`.

Referenced by `do_tests()`.

**5.73.3 Variable Documentation****5.73.3.1 uint16 active\_monitor = -1**

Referenced by `free_virt_monitor()`, `get_active_virt_monitor()`, `get_active_virt_monitor_name()`, `start_vmonitor()`, `switch_monitor_down()`, and `switch_monitor_up()`.

**5.73.3.2 uint16 maxvmonitor = -1**

Referenced by `free_virt_monitor()`, `start_vmonitor()`, `switch_monitor_down()`, and `switch_monitor_up()`.

**5.73.3.3 uint16 num\_vmonitor\_limit = 1000**

Referenced by `init_vmonitors()`.

**5.73.3.4 virt\_monitor\* vmonitors**

Referenced by `free_virt_monitor()`.

**5.74 src/kernel/lib/ringbuffer.c File Reference**

Ring buffer implementation.

```
#include "../include/ringbuffer.h"
```

```
#include "../include/types.h"
```

```
#include "../include/const.h"
#include "../include/assert.h"
#include "../include/stdio.h"
#include "../include/string.h"
#include "../include/stdlib.h"
```

## Functions

- `ring_fifo * rf_alloc (uint32 size)`  
*Allocates and initializes a new ring\_buffer.*
- `ring_fifo * rf_copy (ring_fifo *fifo)`  
*Copies the given ring\_buffer into a new allocated copy.*
- `void rf_free (ring_fifo *fifo)`  
*Destroys a previously allocated ring\_buffer.*
- `void rf_clear (ring_fifo *fifo)`  
*Clears a ring buffer.*
- `uint32 rf_getlength (ring_fifo *fifo)`  
*Returns the buffer's number of used bytes.*
- `bool rf_isfull (ring_fifo *fifo)`  
*Checks whether the given ring\_buffer is full.*
- `bool rf_isempty (ring_fifo *fifo)`  
*Checks whether the given ring\_buffer is empty.*
- `void rf_dump (ring_fifo *fifo)`  
*Prints information about the buffer to stdout.*
- `sint32 rf_write (ring_fifo *fifo, uint8 *buf, uint32 count)`  
*Writes data into a ring\_buffer.*
- `sint32 rf_read (ring_fifo *fifo, uint8 *buf, uint32 count)`  
*Reads data from a ring\_buffer.*

### 5.74.1 Detailed Description

Ring buffer implementation. Mainly used by the process management stdin queue.

#### Author

dbader

#### LastChangedBy:

dtraytel

**Version****Rev:**

12

**5.74.2 Function Documentation****5.74.2.1 ring\_fifo\* rf\_alloc ( uint32 size )**

Allocates and initializes a new ring\_buffer.

**Parameters**

*size* max size of the buffer

**Returns**

pointer to new ring\_buffer or NULL on error

References ring\_fifo::data, mallocn, NULL, rf\_clear(), and ring\_fifo::size.

Referenced by pm\_create\_thread(), pm\_init(), rf\_copy(), and snake().

**5.74.2.2 void rf\_clear ( ring\_fifo \* fifo )**

Clears a ring buffer.

**Parameters**

*fifo* the buffer

References ASSERT, ring\_fifo::end, ring\_fifo::len, NULL, and ring\_fifo::start.

Referenced by rf\_alloc().

**5.74.2.3 ring\_fifo\* rf\_copy ( ring\_fifo \* fifo )**

Copies the given ring\_buffer into a new allocated copy.

**Parameters**

*fifo* the ring\_buffer to copy from

**Returns**

pointer to the new copy of the given ring\_buffer

References ring\_fifo::data, ring\_fifo::end, ring\_fifo::len, memcpy, rf\_alloc(), ring\_fifo::size, and ring\_fifo::start.

Referenced by body\_collision(), and draw\_snake().

#### 5.74.2.4 void rf\_dump ( ring\_fifo \* fifo )

Prints information about the buffer to stdout.

##### Parameters

*fifo* the buffer

References ASSERT, ring\_fifo::data, ring\_fifo::end, ring\_fifo::len, NULL, printf, ring\_fifo::size, and ring\_fifo::start.

#### 5.74.2.5 void rf\_free ( ring\_fifo \* fifo )

Destroys a previously allocated ring\_buffer.

##### Parameters

*fifo* the ring buffer to free

References ASSERT, ring\_fifo::data, free, and NULL.

Referenced by body\_collision(), draw\_snake(), pm\_destroy\_thread(), and snake().

#### 5.74.2.6 uint32 rf\_getlength ( ring\_fifo \* fifo )

Returns the buffer's number of used bytes.

##### Parameters

*fifo* the buffer

##### Returns

Number of used bytes

References ASSERT, ring\_fifo::len, and NULL.

Referenced by body\_collision(), dev\_stdin\_read(), draw\_snake(), and snake().

#### 5.74.2.7 bool rf\_isempty ( ring\_fifo \* fifo )

Checks whether the given ring\_buffer is empty.

##### Parameters

*fifo* the buffer to check

##### Returns

TRUE if the buffer contains at least one byte of data, false if not

References ASSERT, ring\_fifo::len, and NULL.

### 5.74.2.8 `bool rf_isfull ( ring_fifo * fifo )`

Checks whether the given ring\_buffer is full.

#### Parameters

*fifo* the buffer to check

#### Returns

TRUE if at least one byte of storage is available, FALSE if not.

References ASSERT, ring\_fifo::len, NULL, and ring\_fifo::size.

### 5.74.2.9 `sint32 rf_read ( ring_fifo * fifo, uint8 * buf, uint32 count )`

Reads data from a ring\_buffer.

Less than count bytes might be read if the buffer becomes empty.

#### Parameters

*fifo* ring\_buffer to read from

*buf* buffer to write to

*count* number of bytes to write

#### Returns

Number of bytes read, -1 on error

References ring\_fifo::data, ring\_fifo::len, NULL, ring\_fifo::size, and ring\_fifo::start.

Referenced by body\_collision(), dev\_stdin\_read(), draw\_snake(), and snake().

### 5.74.2.10 `sint32 rf_write ( ring_fifo * fifo, uint8 * buf, uint32 count )`

Writes data into a ring\_buffer.

If the buffer gets filled up less than count bytes might be written.

#### Parameters

*fifo* ring\_buffer to write to

*buf* buffer to read from

*count* number of bytes to write

#### Returns

Number of bytes written, -1 on error.

References ring\_fifo::data, ring\_fifo::end, ring\_fifo::len, NULL, printf, and ring\_fifo::size.

Referenced by dev\_stdin\_write(), pm\_handle\_input(), and snake().

## 5.75 src/kernel/lib/stdio.c File Reference

Standard I/O functions.

```
#include "../include/types.h"
#include "../include/const.h"
#include "../include/string.h"
#include "../include/stdio.h"
#include "../include/stdarg.h"
#include "../io/io_virtual.h"
#include "../pm/pm_main.h"
```

### Defines

- #define `SELECT_VMONITOR()` ((`kernel_proc` == NULL) ? `get_active_virt_monitor()` : `kernel_proc->vmonitor`)  
*FIXME: This is used to select a vmonitor for printing.*

### Functions

- int `putchar` (char c)  
*Writes a character to stdout.*
- int `cputchar` (char c, `uint8` fg, `uint8` bg)
- int `puts` (char \*s)  
*Writes a string to stdout.*
- int `cputs` (char \*s, `uint8` fg, `uint8` bg)
- int `vsnprintf` (char \*s, int n, char \*format, `va_list` ap)
- int `snprintf` (char \*buf, int size, char \*fmt,...)
- void `printf` (char \*fmt,...)  
*Prints formatted output.*

### 5.75.1 Detailed Description

Standard I/O functions.

#### Author

dbader

#### LastChangedBy:

dbader

#### Version

Rev:

16

## 5.75.2 Define Documentation

**5.75.2.1** `#define SELECT_VMONITOR( ) ((kernel_proc == NULL) ? get_active_virt_monitor() : kernel_proc->vmonitor)`

FIXME: This is used to select a vmonitor for printing.

Simply using the active monitor does not work, as kernel messages get printed to the vmonitors of other processes. Always using the kernel\_proc vmonitor also fails because kernel\_proc is initialized very late during boot which causes messages to be lost.

I feel this is quite hackish and should be changed as soon as anybody comes up with a proper solution.

Referenced by `cputchar()`, `cputs()`, `putchar()`, and `puts()`.

## 5.75.3 Function Documentation

**5.75.3.1** `int cputchar ( char c, uint8 fg, uint8 bg )`

References `SELECT_VMONITOR`, and `virt_monitor_cputc()`.

**5.75.3.2** `int cputs ( char * s, uint8 fg, uint8 bg )`

References `SELECT_VMONITOR`, and `virt_monitor_cputs()`.

**5.75.3.3** `void printf ( char * fmt, ... )`

Prints formatted output.

The following format specifiers are supported: `%%` - prints the `%` character. `i` - prints a signed integer. `d` - prints a signed integer. `u` - prints an unsigned integer. `b` - prints an unsigned integer in binary format (base 2). `o` - prints an unsigned integer in octal format (base 8). `x` - prints an unsigned integer in hexadecimal format (base 16). `c` - prints a single character. `s` - prints a string. `"(null)"` if argument is `NULL`. `p` - prints a pointer (base 16). `%{` - prints the string until `}` colored. All other format specifiers are ignored.

### Parameters

*fmt* format string

... variable number of arguments

References `puts`, `va_end`, `va_start`, and `vsnprintf`.

**5.75.3.4** `int putchar ( char c )`

Writes a character to stdout.

### Parameters

*c* character to write



**Returns**

the character written

References SELECT\_VMONITOR, and virt\_monitor\_putc().

**5.75.3.5 int puts ( char \* s )**

Writes a string to stdout.

**Bug**

Does not write a terminating newline character. A change in behavior breaks other things (namely [printf\(\)](#)). Please leave it like that for now.

**Parameters**

*s* the string

**Returns**

the number of characters written

References BLACK, SELECT\_VMONITOR, virt\_monitor\_cputs(), and WHITE.

**5.75.3.6 int snprintf ( char \* buf, int size, char \* fmt, ... )**

References va\_end, va\_start, and vsnprintf.

**5.75.3.7 int vsnprintf ( char \* s, int n, char \* format, va\_list ap )**

References itoa, NULL, and va\_arg.

## 5.76 src/kernel/lib/stdlib.c File Reference

Functions to allocate and free memory.

```
#include "../include/types.h"
#include "../include/const.h"
#include "../include/stdio.h"
#include "../include/string.h"
#include "../include/debug.h"
#include "../mm/mm.h"
#include "../mm/mm_paging.h"
```

**Functions**

- void [srand](#) (unsigned int seed)

- `int rand ()`
- `void * malloc (size_t size, char *name)`  
*Allocates 'size' bytes and additionally saves a name in the header of the block.*
- `void * malloc (size_t size)`  
*Allocates 'size' bytes.*
- `void * calloc (size_t n, size_t size, char *name)`  
*Allocates space for n elements of the same size and additionally saves a name in the header of the block.*
- `void * calloc (size_t n, size_t size)`  
*Allocates space for n elements of the same size.*
- `void free (void *start)`  
*Frees a memory block.*
- `void * realloc (void *pointer, size_t size)`  
*Reallocates a memory block to 'size' bytes.*
- `void mem_dump ()`
- `uint32 free_memory ()`  
*Function to return the free memory space.*

### 5.76.1 Detailed Description

Functions to allocate and free memory.

#### Author

Johannes Schamburger

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.76.2 Function Documentation

#### 5.76.2.1 void\* calloc ( size\_t n, size\_t size )

Allocates space for n elements of the same size.

#### Parameters

*n* number of elements

*size* size of each element

References callocn().

#### 5.76.2.2 void\* callocn ( size\_t *n*, size\_t *size*, char \* *name* )

Allocates space for *n* elements of the same size and additionally saves a name in the header of the block.

##### Parameters

*n* number of elements

*size* size of each element

*name* name of the memory block (mainly for debugging purposes)

References bzero, mallocn, and NULL.

Referenced by calloc(), init\_vmonitors(), and malloc\_test().

#### 5.76.2.3 void free ( void \* *start* )

Frees a memory block.

##### Parameters

*start* pointer to the start of the block that shall be freed

References heap\_free(), and kernel\_heap.

#### 5.76.2.4 uint32 free\_memory ( )

Function to return the free memory space.

##### Returns

free memory space in bytes

#### 5.76.2.5 void\* malloc ( size\_t *size* )

Allocates 'size' bytes.

ATTENTION: if malloc fails (i.e. there is not enough free space), the return value is (void\*) NULL. So, this should always be tested!

##### Parameters

*size* how much space shall be allocated

##### Returns

pointer to the allocated space

References mallocn.

### 5.76.2.6 void\* mallocn ( size\_t size, char \* name )

Allocates 'size' bytes and additionally saves a name in the header of the block.

ATTENTION: if malloc fails (i.e. there is not enough free space), the return value is (void\*) NULL. So, this should always be tested!

#### Parameters

*size* how much space shall be allocated

*name* name of the memory block (mainly for debugging purposes)

#### Returns

pointer to the allocated space

References heap\_mallocn(), and kernel\_heap.

### 5.76.2.7 void mem\_dump ( )

References heap\_mem\_dump().

### 5.76.2.8 int rand ( )

Referenced by snake().

### 5.76.2.9 void\* realloc ( void \* pointer, size\_t size )

Reallocates a memory block to 'size' bytes.

ATTENTION: if realloc fails (i.e. there is not enough free space), the return value is (void\*) NULL. So this should always be tested! Especially [realloc\(\)](#) shouldn't be used like this:

```
int* p = malloc(50);
p = realloc(100);
```

In this case, if realloc fails, p is overwritten by (void\*) NULL. So, the memory allocated for p is no longer accessible, which means that the data stored in p is lost and the memory allocated for p can't be used any more.

#### Parameters

*pointer* pointer to the old allocated space

*size* the new size

#### Returns

pointer to the reallocated space

References free, mallocn, memmove(), mm\_header::name, mm\_header::next, NULL, and mm\_header::size.

Referenced by malloc\_test().

### 5.76.2.10 void srand ( unsigned int seed )

## 5.77 src/kernel/lib/string.c File Reference

Basic functions for string manipulation.

```
#include "../include/const.h"
#include "../include/types.h"
#include "../include/stdio.h"
#include "../include/stdlib.h"
```

### Functions

- **uint32 strlen** (char \*str)  
*Returns the length of a null-terminated string.*
- char \* **strcpy** (char \*dest, char \*src)  
*Copies string src to string dest (including terminating \0 character).*
- char \* **strncpy** (char \*dest, char \*src, **size\_t** n)  
*Copies at most n characters from src to dest.*
- char \* **strchr** (char \*str, char ch)  
*Locates the first occurrence of ch in the string pointed to by str.*
- char \* **strdup** (char \*str)  
*Duplicates a string.*
- char \* **strcat** (char \*s1, char \*s2)  
*Concatenates two strings by appending a copy of s2 to the end of s1.*
- char \* **strncat** (char \*s1, char \*s2, **size\_t** n)  
*Concatenates two strings by appending a copy of s2 to the end of s1.*
- char \* **strsep** (char \*\*str\_ptr, char \*delims)  
*Tokenizes a string.*
- **sint32 strcmp** (char \*s1, char \*s2)  
*Compares two strings.*
- void \* **memset** (void \*dest, **uint8** value, **size\_t** count)  
*Writes count bytes of value value to the memory referenced by dest.*
- void **bzero** (void \*dest, **size\_t** count)  
*Writes count bytes of value zero to dest.*
- void \* **memcpy** (void \*dest, void \*src, **size\_t** count)  
*Copies count bytes from src to dest.*

- void \* [memmove](#) (void \*dest, void \*src, [size\\_t](#) count)  
*Copies count bytes form src to dest.*
- char \* [strreverse](#) (char \*str)  
*Reverses a string in place.*
- int [isspace](#) (char c)  
*Test if a character represents whitespace.*
- char \* [itoa](#) (int n, char \*str, unsigned int [base](#))  
*Converts an integer into a string using an arbitrary base.*
- int [strtol](#) (char \*nptr, char \*\*endptr, int [base](#))  
*Converts an string into an integer.*
- int [atoi](#) (char \*str)  
*Converts an string into an integer (base 10 is assumed).*

### 5.77.1 Detailed Description

Basic functions for string manipulation.

#### Author

Dmitriy Traytel  
dbader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.77.2 Function Documentation

#### 5.77.2.1 int atoi ( char \* str )

Converts an string into an integer (base 10 is assumed).

This function will skip whitespace and read a (possibly) signed number until it reaches a char that can't be part of the number.

#### Parameters

*str* the source string buffer

**Returns**

The conversion result

References NULL, and strtol().

Referenced by atoi\_test(), shell\_cmd\_kill(), and shell\_cmd\_nice().

**5.77.2.2 void bzero ( void \* *dest*, size\_t *count* )**

Writes count bytes of value zero to dest.

**Parameters**

*dest* Destination memory

*count* Number of bytes to write

References memset().

**5.77.2.3 int isspace ( char *c* )**

Test if a character represents whitespace.

**Parameters**

*c* the character to be tested

**Returns**

whether the character is to be considered whitespace

For the purposes of this function space, horizontal tab, newline, carriage return, form feed and vertical tab are considered whitespace.

Referenced by strtol().

**5.77.2.4 char\* itoa ( int *n*, char \* *str*, unsigned int *base* )**

Converts an integer into a string using an arbitrary base.

Make sure the buffer for the converted string is large enough. The smaller the base the more space is required, eg for converting a 32 bit integer you can expect 35 bytes to be sufficient (base 2: 32 bytes for digits + 1 byte for the terminator + some space just to feel safe and take into account future format changes). In other, GAD inspired words: bufsize  $\geq$  ceil(log2(MAX\_INT)) which is equal to: bufsize  $\geq$  sizeof(int) \* 8

This is not a part of the C standard library.

**Parameters**

*n* the number to convert

*str* the target string buffer

*base* the base to use for the conversion

**Returns**

The original value of `str`

References `num`, and `strreverse()`.

**5.77.2.5 void\* memcpy ( void \* *dest*, void \* *src*, size\_t *count* )**

Copies `count` bytes from `src` to `dest`.

Undefined behaviour when `src` and `dest` are overlapping (use [memmove\(\)](#) instead).

**See also**

[memmove](#)

**Parameters**

*dest* Destination memory

*src* Source memory

*count* Number of bytes to copy

**Returns**

`dest`, the destination memory

**5.77.2.6 void\* memmove ( void \* *dest*, void \* *src*, size\_t *count* )**

Copies `count` bytes from `src` to `dest`.

Unlike [memcpy\(\)](#), [memmove\(\)](#) supports copy operations where the blocks referenced by `src` and `dest` are overlapping. Should you not need such functionality, use [memcpy\(\)](#) for performance reasons.

**Bug**

There has to be a better way, ie one that does not use dynamic memory.

**See also**

[memcpy](#)

**Parameters**

*dest* Destination memory

*src* Source memory

*count* Number of bytes to copy

**Returns**

`dest`, the destination memory

References `free`, `malloc`, and `memcpy`.

Referenced by `realloc()`.



**5.77.2.7 void\* memset ( void \* *dest*, uint8 *value*, size\_t *count* )**

Writes count bytes of value value to the memory referenced by dest.

**Parameters**

*dest* Destination memory

*value* Value dest is filled with

*count* Number of bytes to write

**Returns**

dest, the destination memory

Referenced by bzero(), dev\_null\_read(), free\_virt\_monitor(), get\_page(), io\_init(), memview\_main(), pm\_init(), shell\_autocomplete(), shell\_cmd\_clear(), shell\_cmd\_pong(), shell\_main(), snake(), snapshot(), start\_vmonitor(), syscall\_test\_thread(), and update\_view().

**5.77.2.8 char\* strcat ( char \* *s1*, char \* *s2* )**

Concatenates two strings by appending a copy of s2 to the end of s1.

**Parameters**

*s1* the "head"

*s2* the "tail"

**Returns**

the concatenated string

References strchr(), and strcpy().

**5.77.2.9 char\* strchr ( char \* *str*, char *ch* )**

Locates the first occurrence of ch in the string pointed to by str.

The terminating \0 character is considered to be part of the string.

**Parameters**

*str* the string to search

*ch* the character to look for

**Returns**

the first occurrence of ch in str or NULL if not found

Referenced by strcat(), strncat(), and strsep().

### 5.77.2.10 `sint32 strcmp ( char * s1, char * s2 )`

Compares two strings.

#### Parameters

*s1* String

*s2* String

#### Returns

0 if both strings are equal, >0 if s1 greater than s2, <0 if s1 is less than s2

### 5.77.2.11 `char* strcpy ( char * dest, char * src )`

Copies string src to string dest (including terminating \0 character).

For stability and security reasons, try to use [strncpy\(\)](#) whenever possible.

#### See also

[strncpy](#)

#### Parameters

*dest* Destination string

*src* Source string

#### Returns

dest, the destination string

Referenced by [new\\_shell\(\)](#), [potatoes\\_time2str\(\)](#), [shell\\_cmd\\_cd\(\)](#), [shell\\_cmd\\_snake\(\)](#), [shell\\_main\(\)](#), [shell\\_-makepath\(\)](#), [snapshot\(\)](#), [strcat\(\)](#), [strdup\(\)](#), and [time2str\(\)](#).

### 5.77.2.12 `char* strdup ( char * str )`

Duplicates a string.

[strdup\(\)](#) allocates sufficient memory for a copy of the string str, copies it and returns a pointer to the copied string. Strings returned by [strdup\(\)](#) must be released by calling [free\(\)](#).

#### Parameters

*str* the string to duplicate

#### Returns

The pointer to the duplicated string or NULL on error

References [mallocn](#), [NULL](#), [strcpy\(\)](#), and [strlen](#).

**5.77.2.13 uint32 strlen ( char \* *str* )**

Returns the length of a null-terminated string.

**Parameters**

*str* String to check

**Returns**

Number of characters preceding the terminating null character.

**5.77.2.14 char\* strncat ( char \* *s1*, char \* *s2*, size\_t *n* )**

Concatenates two strings by appending a copy of *s2* to the end of *s1*.

Not more than *n* characters are copied from *s2*.

**Parameters**

*s1* the "head"

*s2* the "tail"

*n* max number of characters to copy

**Returns**

the concatenated string

References strchr(), and strncpy.

Referenced by new\_shell(), potatoes\_strncat(), and strings\_test().

**5.77.2.15 char\* strncpy ( char \* *dest*, char \* *src*, size\_t *n* )**

Copies at most *n* characters from *src* to *dest*.

If *src* is less than *n* characters long, the remainder of *dest* is filled with `\0` characters. Otherwise, *dest* is not terminated.

**Parameters**

*dest* Destination string

*src* Source string

*n* Number of bytes to copy at most

**Returns**

*dest*, the destination string

**5.77.2.16 char\* strreverse ( char \* *str* )**

Reverses a string in place.

This is not a part of the C standard library.

**Parameters**

*str* the string to reverse

**Returns**

the reversed string

References end, start, and strlen.

Referenced by itoa().

**5.77.2.17 char\* strsep ( char \*\* str\_ptr, char \* delims )**

Tokenizes a string.

Take note that [strsep\(\)](#) will manipulate both the string pointer *\*\*str\_ptr* points at as well as the contents of the respective string.

Example:

```
char path[] = "/usr/share/bin/editor";
char delim[] = "/";
char *tok;
char *copy = strdup(path);
char *work_copy = copy;

do {
    printf("strsep(\"%s\") ", work_copy);
    tok = strsep(&work_copy, delim);
    printf("-> \"%s\"\n", tok);
} while (tok != NULL);

printf("\ncopy = %p\n", copy);
printf("work_copy = %p\n", work_copy);
puts("done.");

free(copy);
```

**Bug**

The current implementation does not handle multiple delimiters (as specified in the libc manual). Only the first character in *\*delims* is used for tokenizing the input string.

**Parameters**

*str\_ptr* Pointer to string to tokenize

*delims* String containing all delimiter characters

**Returns**

The next token or NULL if the end of the input string was reached

References NULL, and strchr().

**5.77.2.18 int strtol ( char \* nptr, char \*\* endptr, int base )**

Converts an string into an integer.

Bases in the range from 2 to 36 are handled, with a-z and A-Z being treated as the digits with values 10 to 35.

If *base* is 0, try to guess the base, if the string starts with "0x" it will be treated as base 16, if it starts with "0" it will be treated as base 8, otherwise base 10 is assumed.

No range check is performed, so if the value is greater than or equal to  $2^{31}-1$  or less than  $-2^{31}$  the results will be undefined.

### Parameters

*str* the source string buffer

*endptr* if this not equal to NULL, the address of the first character that was not parsed is written to this address

*base* the base to be used for conversion

### Returns

the result of the conversion

References isspace(), and NULL.

Referenced by atoi(), and atoi\_test().

## 5.78 src/kernel/mm/mm.h File Reference

Definitions for the functions and variables used in the memory management.

```
#include "../include/types.h"
```

### Data Structures

- struct [mm\\_header](#)

*the structure of a header of an occupied memory block*

- struct [heap\\_t](#)
- struct [gdt\\_entry](#)

*The structure of one GDT entry.*

- struct [gdt\\_pointer](#)

*The structure of the GDT pointer which tells the processor where to find our GDT.*

### Defines

- #define [KHEAP\\_START](#) 0xC0000000
- #define [KHEAP\\_INITIAL\\_SIZE](#) 0x500000
- #define [HEAP\\_MIN\\_SIZE](#) 0x70000
- #define [HEAP\\_EXPAND\\_STEP\\_SIZE](#) 1048576

*the heap is expanded only by steps of a certain size.*

## Typedefs

- typedef struct [mm\\_header](#) [mm\\_header](#)  
*the structure of a header of an occupied memory block*

## Functions

- void [mm\\_init](#) ()
- void [mm\\_init\\_output](#) ()  
*Second part of the memory management initialization.*
- [heap\\_t](#) \* [create\\_heap](#) ([uint32](#) start\_addr, [uint32](#) end\_addr, [uint32](#) max\_addr, [uint8](#) supervisor, [uint8](#) readonly, [uint32](#) heap\_addr)  
*Creates a heap.*
- void \* [heap\\_mallocn](#) ([size\\_t](#) size, char \*name, [uint8](#) page\_aligned, [heap\\_t](#) \*heap)  
*Allocates 'size' bytes and additionally saves a name in the header of the block.*
- void [heap\\_free](#) (void \*ptr, [heap\\_t](#) \*heap)  
*Frees a memory block.*
- void [heap\\_mem\\_dump](#) ()  
*Dumps the heap.*
- struct [gdt\\_entry](#) [\\_\\_attribute\\_\\_\(\(packed\)\)](#)  
*The structure of one GDT entry.*
- void [gdt\\_init](#) ()  
*initializes the GDT*
- void [mm\\_move\\_block](#) ([uint32](#) dest, [mm\\_header](#) \*src)

## Variables

- [heap\\_t](#) \* [kernel\\_heap](#)
- [uint16](#) [limit\\_low](#)
- [uint16](#) [base\\_low](#)
- [uint8](#) [base\\_middle](#)
- [uint8](#) [access](#)
- [uint8](#) [granularity](#)
- [uint8](#) [base\\_high](#)
- [uint16](#) [limit](#)
- [uint32](#) [base](#)
- struct [gdt\\_entry](#) [gdt](#) [3]
- struct [gdt\\_pointer](#) [gp](#)

## 5.78.1 Detailed Description

Definitions for the functions and variables used in the memory management.

### Author

Johannes Schamburger

### LastChangedBy:

dtraytel

### Version

### Rev:

20

## 5.78.2 Define Documentation

### 5.78.2.1 `#define HEAP_EXPAND_STEP_SIZE 1048576`

the heap is expanded only by steps of a certain size.

Referenced by `heap_mallocn()`.

### 5.78.2.2 `#define HEAP_MIN_SIZE 0x70000`

Referenced by `heap_contract()`.

### 5.78.2.3 `#define KHEAP_INITIAL_SIZE 0x500000`

### 5.78.2.4 `#define KHEAP_START 0xC0000000`

Referenced by `mm_init()`.

## 5.78.3 Typedef Documentation

### 5.78.3.1 `typedef struct mm_header mm_header`

the structure of a header of an occupied memory block

## 5.78.4 Function Documentation

### 5.78.4.1 `struct gdt_entry __attribute__((packed))`

The structure of one GDT entry.

The structure of the GDT pointer which tells the processor where to find our GDT.

access byte: 7 | 6 | 5 | 4 | 3 | 0 | present | ring (0-3) | descriptor type | segment type (1010:code;0010:data) | granularity byte: 7 | 6 | 5 | 4 | 3 | 0 | granularity | operand size | always 0 | available (always 0) | segment length (bits 19 - 16) | granularity bit: 0 -> 1 byte; 1 -> 4 kbyte operand size: 0 -> 16bit; 1 -> 32bit

#### 5.78.4.2 **heap\_t\*** `create_heap ( uint32 start_addr, uint32 end_addr, uint32 max_addr, uint8 supervisor, uint8 readonly, uint32 heap_addr )`

Creates a heap.

##### Parameters

*start\_addr* virtual start address of the heap  
*end\_addr* virtual end address of the heap  
*max\_addr* virtual maximum address of the heap (needed for expand)  
*supervisor* 1: supervisor mode; 0: user mode  
*readonly* 1: read-only; 0: readable and writeable  
*heap\_addr* physical address of the heap structure

##### Returns

the created heap

References `heap_t::end`, `kernel_heap`, `heap_t::max_addr`, `mm_header::name`, `mm_header::next`, `mm_header::prev`, `heap_t::readonly`, `mm_header::size`, `heap_t::start`, `strncpy`, and `heap_t::supervisor`.

Referenced by `mm_init()`.

#### 5.78.4.3 **void** `gdt_init ( )`

initializes the GDT

References `gdt_pointer::base`, `gdt`, `gdt_add_entry()`, `gdt_flush()`, `gp`, and `gdt_pointer::limit`.

Referenced by `mm_init_output()`.

#### 5.78.4.4 **void** `heap_free ( void * start, heap_t * heap )`

Frees a memory block.

##### Parameters

*start* pointer to the start of the block that shall be freed

References `dprintf`.

Referenced by `free()`.

#### 5.78.4.5 **void\*** `heap_mallocn ( size_t size, char * name, uint8 page_aligned, heap_t * heap )`

Allocates 'size' bytes and additionally saves a name in the header of the block.

##### Parameters

*size* how much space shall be allocated



*name* name of the memory block (mainly for debugging purposes)

*page\_aligned* 1: the block is allocated at a page-aligned address

*heap* the heap in which the block shall be allocated

### Returns

pointer to the allocated space

References heap\_t::end, heap\_expand(), HEAP\_EXPAND\_STEP\_SIZE, heap\_get\_size(), heap\_mallocn(), heap\_setup\_block(), kernel\_heap, mm\_header::next, NULL, mm\_header::prev, and heap\_t::start.

Referenced by heap\_mallocn(), and mallocn().

#### 5.78.4.6 void heap\_mem\_dump ( )

Dumps the heap.

References \_printf(), heap\_t::end, kernel\_heap, mm\_header::name, mm\_header::next, mm\_header::size, and heap\_t::start.

Referenced by malloc\_test(), and mem\_dump().

#### 5.78.4.7 void mm\_init ( )

#### 5.78.4.8 void mm\_init\_output ( )

Second part of the memory management initialization.

(must be called after [init\\_vmonitors\(\)](#) in [main\(\)](#); the GDT initialization must be here, too, because it uses [dprintf\(\)](#) )

References dprint\_separator, dprintf, heap\_t::end, gdt\_init(), kernel\_heap, printf, and heap\_t::start.

Referenced by main().

#### 5.78.4.9 void mm\_move\_block ( uint32 dest, mm\_header \* src )

### 5.78.5 Variable Documentation

#### 5.78.5.1 uint8 access

#### 5.78.5.2 uint32 base

#### 5.78.5.3 uint8 base\_high

#### 5.78.5.4 uint16 base\_low

#### 5.78.5.5 uint8 base\_middle

#### 5.78.5.6 struct gdt\_entry gdt[3]

Referenced by gdt\_add\_entry(), and gdt\_init().

### 5.78.5.7 struct gdt\_pointer gp

Referenced by gdt\_init().

### 5.78.5.8 uint8 granularity

### 5.78.5.9 heap\_t\* kernel\_heap

Referenced by create\_heap(), free(), heap\_mallocn(), heap\_mem\_dump(), malloc\_test(), mallocn(), mark\_visual\_block(), memview\_main(), mm\_init(), mm\_init\_output(), mv\_show\_stats(), and update\_view().

### 5.78.5.10 uint16 limit

### 5.78.5.11 uint16 limit\_low

## 5.79 src/kernel/mm/mm\_bitmap.c File Reference

all functions concerning the allocation and deallocation of frames

```
#include "../include/types.h"
#include "../include/stdio.h"
#include "../include/string.h"
#include "../include/init.h"
#include "mm_paging.h"
#include "mm_bitmap.h"
```

### Functions

- void [set\\_frame](#) (uint32 frame\_addr)  
*Marks a frame as occupied.*
- void [clear\\_frame](#) (uint32 frame\_addr)  
*Marks a frame as free.*
- bool [test\\_frame](#) (uint32 frame\_addr)  
*Tests if a frame is free.*
- uint32 [first\\_free\\_frame](#) ()  
*Finds the first free frame in the bitset.*
- void [alloc\\_frame](#) (page\_t \*page, int kernel, int writeable)  
*Allocates a frame.*
- void [free\\_frame](#) (page\_t \*page)  
*Deallocates a frame.*

- void `bitset_test` ()  
*just some tests to see if the bitset is working properly*

### 5.79.1 Detailed Description

all functions concerning the allocation and deallocation of frames

#### Author

Johannes Schamburger

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.79.2 Function Documentation

#### 5.79.2.1 void `alloc_frame` ( `page_t` \* `page`, `int` `kernel`, `int` `writable` )

Allocates a frame.

#### Parameters

*page* the page

*kernel* 1: frame is only accessible from the kernel

*writable* 1: frame is writable; 0: frame is read-only

References `first_free_frame()`, `page::frame`, `panic`, `page::present`, `page::rw`, `set_frame()`, and `page::user`.

Referenced by `heap_expand()`, and `mm_init()`.

#### 5.79.2.2 void `bitset_test` ( )

*just some tests to see if the bitset is working properly*

References `first_free_frame()`, `nframes`, `printf`, `set_frame()`, and `test_frame()`.

#### 5.79.2.3 void `clear_frame` ( `uint32` `frame_addr` )

Marks a frame as free.

#### Parameters

*frame\_addr* address of the frame

References `frames`.

Referenced by `free_frame()`.

#### 5.79.2.4 `uint32 first_free_frame ( )`

Finds the first free frame in the bitset.

##### Returns

number of the first free frame

References frames, and nframes.

Referenced by alloc\_frame(), and bitset\_test().

#### 5.79.2.5 `void free_frame ( page_t * page )`

Deallocates a frame.

##### Parameters

*page* the page

References clear\_frame(), and page::frame.

Referenced by heap\_contract().

#### 5.79.2.6 `void set_frame ( uint32 frame_addr )`

Marks a frame as occupied.

##### Parameters

*frame\_addr* address of the frame

References frames.

Referenced by alloc\_frame(), and bitset\_test().

#### 5.79.2.7 `bool test_frame ( uint32 frame_addr )`

Tests if a frame is free.

##### Parameters

*frame\_addr* address of the frame

##### Returns

0: frame is occupied; any other value: frame is free

References frames.

Referenced by bitset\_test().

## 5.80 src/kernel/mm/mm\_bitmap.h File Reference

Declarations of the functions needed for frame allocation.

```
#include "../include/types.h"
#include "../include/stdio.h"
#include "mm_paging.h"
```

### Functions

- void `set_frame` (uint32 frame\_addr)  
*Marks a frame as occupied.*
- void `clear_frame` (uint32 frame\_addr)  
*Marks a frame as free.*
- bool `test_frame` (uint32 frame\_addr)  
*Tests if a frame is free.*
- uint32 `first_free_frame` ()  
*Finds the first free frame in the bitset.*
- void `alloc_frame` (page\_t \*page, int kernel, int writeable)  
*Allocates a frame.*
- void `free_frame` (page\_t \*page)  
*Deallocates a frame.*
- void `bitset_test` ()  
*just some tests to see if the bitset is working properly*

### 5.80.1 Detailed Description

Declarations of the functions needed for frame allocation.

#### Author

Johannes Schamburger  
\$LastChangedBy \$

#### Version

\$Rev \$

### 5.80.2 Function Documentation

#### 5.80.2.1 void `alloc_frame` ( page\_t \* page, int kernel, int writeable )

Allocates a frame.

**Parameters**

*page* the page

*kernel* 1: frame is only accessible from the kernel

*writable* 1: frame is writable; 0: frame is read-only

References `first_free_frame()`, `page::frame`, `panic`, `page::present`, `page::rw`, `set_frame()`, and `page::user`.

Referenced by `heap_expand()`, and `mm_init()`.

**5.80.2.2 void bitset\_test ( )**

just some tests to see if the bitset is working properly

References `first_free_frame()`, `nframes`, `printf`, `set_frame()`, and `test_frame()`.

**5.80.2.3 void clear\_frame ( uint32 frame\_addr )**

Marks a frame as free.

**Parameters**

*frame\_addr* address of the frame

References `frames`.

Referenced by `free_frame()`.

**5.80.2.4 uint32 first\_free\_frame ( )**

Finds the first free frame in the bitset.

**Returns**

number of the first free frame

References `frames`, and `nframes`.

Referenced by `alloc_frame()`, and `bitset_test()`.

**5.80.2.5 void free\_frame ( page\_t \* page )**

Deallocates a frame.

**Parameters**

*page* the page

References `clear_frame()`, and `page::frame`.

Referenced by `heap_contract()`.

### 5.80.2.6 void set\_frame ( uint32 frame\_addr )

Marks a frame as occupied.

#### Parameters

*frame\_addr* address of the frame

References frames.

Referenced by alloc\_frame(), and bitset\_test().

### 5.80.2.7 bool test\_frame ( uint32 frame\_addr )

Tests if a frame is free.

#### Parameters

*frame\_addr* address of the frame

#### Returns

0: frame is occupied; any other value: frame is free

References frames.

Referenced by bitset\_test().

## 5.81 src/kernel/mm/mm\_gdt.c File Reference

Creates and initializes the global descriptor table.

```
#include "../include/types.h"
#include "../include/debug.h"
#include "mm.h"
```

### Functions

- void [gdt\\_flush](#) ()
- void [gdt\\_add\\_entry](#) (sint32 num, uint32 base, uint32 limit, uint8 access, uint8 gran)  
*defines one GDT entry*
- void [gdt\\_init](#) ()  
*initializes the GDT*

### 5.81.1 Detailed Description

Creates and initializes the global descriptor table. based on JamesM's kernel development tutorials

**Author**

Johannes Schamburger

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

## 5.81.2 Function Documentation

### 5.81.2.1 void gdt\_add\_entry ( sint32 num, uint32 base, uint32 limit, uint8 access, uint8 gran )

defines one GDT entry

**Parameters**

*num* number of the entry that shall be added

*base* base address

*limit* end address

*access* access byte (structure

**See also**

[mm.h](#))

**Parameters**

*gran* granularity byte (structure

**See also**

[mm.h](#))

References `gdt_entry::access`, `gdt_entry::base_high`, `gdt_entry::base_low`, `gdt_entry::base_middle`, `dprintf`, `gdt`, `gdt_entry::granularity`, and `gdt_entry::limit_low`.

Referenced by `gdt_init()`.

### 5.81.2.2 void gdt\_flush ( )

Referenced by `gdt_init()`.

### 5.81.2.3 void gdt\_init ( )

initializes the GDT

References `gdt_pointer::base`, `gdt`, `gdt_add_entry()`, `gdt_flush()`, `gp`, and `gdt_pointer::limit`.

Referenced by `mm_init_output()`.



## 5.82 src/kernel/mm/mm\_heap.c File Reference

Declarations for the functions and variables needed for paging.

```
#include "../include/types.h"
#include "../include/string.h"
#include "../include/stdio.h"
#include "../include/const.h"
#include "../include/debug.h"
#include "../include/assert.h"
#include "../include/stdlib.h"
#include "mm.h"
#include "mm_bitmap.h"
#include "mm_paging.h"
```

### Functions

- `heap_t * create_heap (uint32 start_addr, uint32 end_addr, uint32 max_addr, uint8 supervisor, uint8 readonly, uint32 heap_addr)`  
*Creates a heap.*
- `uint32 heap_get_size (heap_t *heap)`  
*Returns the total size of a heap.*
- `void heap_expand (uint32 new_size, heap_t *heap)`  
*Expands 'heap' to 'new\_size'.*
- `uint32 heap_contract (uint32 new_size, heap_t *heap)`  
*Contracts 'heap' to 'new\_size'.*
- `mm_header * heap_setup_block (mm_header *next_block, uint32 position, size_t size, char *name)`  
*Sets up a new block in the heap.*
- `void * heap_mallocn (size_t size, char *name, uint8 page_aligned, heap_t *heap)`  
*Allocates 'size' bytes and additionally saves a name in the header of the block.*
- `void heap_free (void *start, heap_t *heap)`  
*Frees a memory block.*
- `void _printf (char *fmt,...)`  
*Prints formatted output to STDOUT.*
- `void heap_mem_dump ()`  
*Dumps the heap.*

## 5.82.1 Detailed Description

Declarations for the functions and variables needed for paging.

### Author

Johannes Schamburger  
 \$LastChangedBy \$

### Version

\$Rev \$

## 5.82.2 Function Documentation

### 5.82.2.1 void \_printf ( char \* *fmt*, ... )

Prints formatted output to STDOUT.

#### See also

[printf](#) This exists as a stub to ease the separation of the shell from the kernel code (as of now, the shell could simply call the kernel [printf](#)).

### 5.82.2.2 heap\_t\* create\_heap ( uint32 *start\_addr*, uint32 *end\_addr*, uint32 *max\_addr*, uint8 *supervisor*, uint8 *readonly*, uint32 *heap\_addr* )

Creates a heap.

#### Parameters

*start\_addr* virtual start address of the heap  
*end\_addr* virtual end address of the heap  
*max\_addr* virtual maximum address of the heap (needed for expand)  
*supervisor* 1: supervisor mode; 0: user mode  
*readonly* 1: read-only; 0: readable and writeable  
*heap\_addr* physical address of the heap structure

#### Returns

the created heap

References heap\_t::end, kernel\_heap, heap\_t::max\_addr, mm\_header::name, mm\_header::next, mm\_header::prev, heap\_t::readonly, mm\_header::size, heap\_t::start, strncpy, and heap\_t::supervisor.

Referenced by mm\_init().

### 5.82.2.3 uint32 heap\_contract ( uint32 *new\_size*, heap\_t \* *heap* )

Contracts 'heap' to 'new\_size'.

**Parameters**

*new\_size* new size of the heap  
*heap* the heap that shall be contracted

**Returns**

new size of the heap

References ASSERT, heap\_t::end, free\_frame(), get\_page(), heap\_get\_size(), HEAP\_MIN\_SIZE, kernel\_dir, mm\_header::name, mm\_header::next, mm\_header::prev, mm\_header::size, heap\_t::start, and strncpy.

**5.82.2.4 void heap\_expand ( uint32 new\_size, heap\_t \* heap )**

Expands 'heap' to 'new\_size'.

**Parameters**

*new\_size* new size of the heap  
*heap* the heap that shall be expanded

References alloc\_frame(), ASSERT, heap\_t::end, get\_page(), heap\_get\_size(), kernel\_dir, mm\_header::name, mm\_header::next, mm\_header::prev, heap\_t::readonly, mm\_header::size, heap\_t::start, strncpy, and heap\_t::supervisor.

Referenced by heap\_mallocn().

**5.82.2.5 void heap\_free ( void \* start, heap\_t \* heap )**

Frees a memory block.

**Parameters**

*start* pointer to the start of the block that shall be freed

References dprintf.

Referenced by free().

**5.82.2.6 uint32 heap\_get\_size ( heap\_t \* heap )**

Returns the total size of a heap.

**Parameters**

*heap* the heap

**Returns**

size of the heap

References heap\_t::end, and heap\_t::start.

Referenced by heap\_contract(), heap\_expand(), heap\_mallocn(), and update\_view().

**5.82.2.7 void\* heap\_mallocn ( size\_t size, char \* name, uint8 page\_aligned, heap\_t \* heap )**

Allocates 'size' bytes and additionally saves a name in the header of the block.

**Parameters**

*size* how much space shall be allocated  
*name* name of the memory block (mainly for debugging purposes)  
*page\_aligned* 1: the block is allocated at a page-aligned address  
*heap* the heap in which the block shall be allocated

**Returns**

pointer to the allocated space

References heap\_t::end, heap\_expand(), HEAP\_EXPAND\_STEP\_SIZE, heap\_get\_size(), heap\_mallocn(), heap\_setup\_block(), kernel\_heap, mm\_header::next, NULL, mm\_header::prev, and heap\_t::start.

Referenced by heap\_mallocn(), and mallocn().

**5.82.2.8 void heap\_mem\_dump ( )**

Dumps the heap.

References \_printf(), heap\_t::end, kernel\_heap, mm\_header::name, mm\_header::next, mm\_header::size, and heap\_t::start.

Referenced by malloc\_test(), and mem\_dump().

**5.82.2.9 mm\_header\* heap\_setup\_block ( mm\_header \* next\_block, uint32 position, size\_t size, char \* name )**

Sets up a new block in the heap.

**Parameters**

*next\_block* pointer to the block after the one to be inserted  
*position* end of the block before the one to be inserted  
*size* size of the new block  
*name* name of the new block

**Returns**

the header of the new block

References mm\_header::name, mm\_header::next, mm\_header::prev, mm\_header::size, and strncpy.

Referenced by heap\_mallocn().

**5.83 src/kernel/mm/mm\_main.c File Reference**

Initializes memory management.

```
#include "../include/types.h"
#include "../include/stdio.h"
#include "../include/stdlib.h"
#include "../include/debug.h"
#include "../include/string.h"
#include "mm.h"
#include "mm_paging.h"
#include "mm_bitmap.h"
#include "../io/io_virtual.h"
```

## Functions

- void `mm_init` (uint32 start, uint32 end)  
*initializes memory management (including the GDT)*
- void `mm_init_output` ()  
*Second part of the memory management initialization.*
- void `switch_page_dir` (page\_directory\_t \*new)  
*Loads a page-directory to the CR3 register.*
- page\_t \* `get_page` (uint32 address, int make, page\_directory\_t \*dir)  
*Returns a pointer to the required page.*
- uint32 `kmalloc` (uint32 size, int page\_aligned, uint32 \*phys)  
*Allocates a fixed-size block of memory.*

### 5.83.1 Detailed Description

Initializes memory management.

#### Author

Johannes Schamburger

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

## 5.83.2 Function Documentation

### 5.83.2.1 `page_t* get_page ( uint32 address, int make, page_directory_t * dir )`

Returns a pointer to the required page.

#### Parameters

*address*

*make* if `make == 1` and the page doesn't already exist, it is created

*dir* the page-directory

References `kmalloc()`, `memset()`, `page_table::pages`, `page_directory::tables`, and `page_directory::tablesPhysical`.

Referenced by `heap_contract()`, `heap_expand()`, and `mm_init()`.

### 5.83.2.2 `uint32 kmalloc ( uint32 size, int page_aligned, uint32 * phys )`

Allocates a fixed-size block of memory.

#### Parameters

*size* the size of the block that shall be allocated

*aligned* 1: the block shall be page-aligned

#### Returns

address of the allocated block

References `placement_addr`.

Referenced by `get_page()`, and `mm_init()`.

### 5.83.2.3 `void mm_init ( uint32 start, uint32 end )`

initializes memory management (including the GDT)

#### Parameters

*start* start address of the free memory (= end of kernel)

*end* end address of the available memory

References `alloc_frame()`, `bzero`, `create_heap()`, `current_dir`, `frames`, `get_page()`, `kernel_dir`, `kernel_heap`, `KHEAP_START`, `kmalloc()`, `nframes`, `placement_addr`, and `switch_page_dir()`.

Referenced by `main()`.

### 5.83.2.4 `void mm_init_output ( )`

Second part of the memory management initialization.

(must be called after `init_ymonitors()` in `main()`; the GDT initialization must be here, too, because it uses `dprintf()`)

References `dprint_separator`, `dprintf`, `heap_t::end`, `gdt_init()`, `kernel_heap`, `printf`, and `heap_t::start`.  
Referenced by `main()`.

### 5.83.2.5 void switch\_page\_dir ( page\_directory\_t \* new )

Loads a page-directory to the CR3 register.

#### Parameters

*new* the page-directory that shall be loaded to CR3

References `current_dir`, `read_from_cr0()`, `write_to_cr0()`, and `write_to_cr3()`.

Referenced by `mm_init()`.

## 5.84 src/kernel/mm/mm\_paging.h File Reference

Declarations for the functions and variables needed for paging.

```
#include "../include/types.h"  
#include "../pm/pm_main.h"
```

### Data Structures

- struct [page](#)
- struct [page\\_table](#)
- struct [page\\_directory](#)

### Typedefs

- typedef struct [page](#) [page\\_t](#)
- typedef struct [page\\_table](#) [page\\_table\\_t](#)
- typedef struct [page\\_directory](#) [page\\_directory\\_t](#)

### Functions

- void [switch\\_page\\_dir](#) ([page\\_directory\\_t](#) \*new)  
*Loads a page-directory to the CR3 register.*
- [page\\_t](#) \* [get\\_page](#) ([uint32](#) address, int make, [page\\_directory\\_t](#) \*dir)  
*Returns a pointer to the required page.*
- void [page\\_fault](#) ([cpu\\_state\\_t](#) state)
- [uint32](#) [kmalloc](#) ([uint32](#) size, int page\_aligned, [uint32](#) \*phys)  
*Allocates a fixed-size block of memory.*
- void [write\\_to\\_cr0](#) ([uint32](#) cr0)
- [uint32](#) [read\\_from\\_cr0](#) ()
- void [write\\_to\\_cr3](#) ([uint32](#) cr3)
- [uint32](#) [read\\_from\\_cr3](#) ()

## Variables

- [uint32 placement\\_addr](#)
- [page\\_directory\\_t \\* kernel\\_dir](#)
- [page\\_directory\\_t \\* current\\_dir](#)
- [uint32 \\* frames](#)
- [uint32 nframes](#)

### 5.84.1 Detailed Description

Declarations for the functions and variables needed for paging.

#### Author

Johannes Schamburger  
\$LastChangedBy \$

#### Version

\$Rev \$

### 5.84.2 Typedef Documentation

**5.84.2.1** `typedef struct page_directory page_directory_t`

**5.84.2.2** `typedef struct page page_t`

**5.84.2.3** `typedef struct page_table page_table_t`

### 5.84.3 Function Documentation

**5.84.3.1** `page_t* get_page ( uint32 address, int make, page_directory_t * dir )`

Returns a pointer to the required page.

#### Parameters

*address*

*make* if `make == 1` and the page doesn't already exist, it is created

*dir* the page-directory

References `kmalloc()`, `memset()`, `page_table::pages`, `page_directory::tables`, and `page_directory::tablesPhysical`.

Referenced by `heap_contract()`, `heap_expand()`, and `mm_init()`.

**5.84.3.2** `uint32 kmalloc ( uint32 size, int page_aligned, uint32 * phys )`

Allocates a fixed-size block of memory.

#### Parameters

*size* the size of the block that shall be allocated



*aligned* 1: the block shall be page-aligned

### Returns

adress of the allocated block

References placement\_addr.

Referenced by get\_page(), and mm\_init().

#### 5.84.3.3 void page\_fault ( cpu\_state\_t state )

#### 5.84.3.4 uint32 read\_from\_cr0 ( )

Referenced by switch\_page\_dir().

#### 5.84.3.5 uint32 read\_from\_cr3 ( )

#### 5.84.3.6 void switch\_page\_dir ( page\_directory\_t \* new )

Loads a page-directory to the CR3 register.

### Parameters

*new* the page-directory that shall be loaded to CR3

References current\_dir, read\_from\_cr0(), write\_to\_cr0(), and write\_to\_cr3().

Referenced by mm\_init().

#### 5.84.3.7 void write\_to\_cr0 ( uint32 cr0 )

Referenced by switch\_page\_dir().

#### 5.84.3.8 void write\_to\_cr3 ( uint32 cr3 )

Referenced by switch\_page\_dir().

## 5.84.4 Variable Documentation

#### 5.84.4.1 page\_directory\_t\* current\_dir

Referenced by mm\_init(), and switch\_page\_dir().

#### 5.84.4.2 uint32\* frames

Referenced by clear\_frame(), first\_free\_frame(), mm\_init(), set\_frame(), and test\_frame().

#### 5.84.4.3 page\_directory\_t\* kernel\_dir

Referenced by heap\_contract(), heap\_expand(), and mm\_init().

#### 5.84.4.4 uint32 nframes

Referenced by `bitset_test()`, `first_free_frame()`, and `mm_init()`.

#### 5.84.4.5 uint32 placement\_addr

Referenced by `kmalloc()`, and `mm_init()`.

## 5.85 src/kernel/pm/dev\_brainfuck.c File Reference

The brainfuck device.

```
#include "../include/types.h"
#include "../include/const.h"
#include "../include/string.h"
#include "../include/debug.h"
#include "pm_devices.h"
#include "../../../apps/brainfuck_interpreter.h"
```

### Functions

- `int dev_brainfuck_open` (void \*dev, char \*path, int oflag, int mode)
- `int dev_brainfuck_close` (void \*dev, int fd)
- `int dev_brainfuck_read` (void \*dev, int fd, void \*buf, int size)
- `int dev_brainfuck_write` (void \*dev, int fd, void \*buf, int size)
- `int dev_brainfuck_seek` (void \*dev, int fd, int offset, int whence)

### Variables

- `device_t dev_brainfuck`

### 5.85.1 Detailed Description

The brainfuck device.

#### Author

Dmitriy Traytel

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

## 5.85.2 Function Documentation

**5.85.2.1** `int dev_brainfuck_close ( void * dev, int fd )`

**5.85.2.2** `int dev_brainfuck_open ( void * dev, char * path, int oflag, int mode )`

**5.85.2.3** `int dev_brainfuck_read ( void * dev, int fd, void * buf, int size )`

References `interpret_bf()`.

**5.85.2.4** `int dev_brainfuck_seek ( void * dev, int fd, int offset, int whence )`

**5.85.2.5** `int dev_brainfuck_write ( void * dev, int fd, void * buf, int size )`

References `interpret_bf()`.

## 5.85.3 Variable Documentation

### 5.85.3.1 `device_t dev_brainfuck`

**Initial value:**

```
{
    "/dev/brainfuck",
    BRAINFUCK,
    NULL,
    dev_brainfuck_open,
    dev_brainfuck_close,
    dev_brainfuck_read,
    dev_brainfuck_write,
    dev_brainfuck_seek
}
```

## 5.86 src/kernel/pm/dev\_clock.c File Reference

The clock device.

```
#include "../include/types.h"
#include "../include/const.h"
#include "../include/string.h"
#include "../include/stdlib.h"
#include "../include/debug.h"
#include "pm_devices.h"
#include "../io/io_rtc.h"
```

### Functions

- `int dev_clock_open` (void \*dev, char \*path, int oflag, int mode)
- `int dev_clock_close` (void \*dev, int fd)

- int `dev_clock_read` (void \*dev, int fd, void \*buf, int size)
- int `dev_clock_write` (void \*dev, int fd, void \*buf, int size)
- int `dev_clock_seek` (void \*dev, int fd, int offset, int whence)

## Variables

- `device_t dev_clock`

### 5.86.1 Detailed Description

The clock device. This provides access to the current time and date in string form.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.86.2 Function Documentation

**5.86.2.1** int `dev_clock_close` ( void \* *dev*, int *fd* )

**5.86.2.2** int `dev_clock_open` ( void \* *dev*, char \* *path*, int *oflag*, int *mode* )

**5.86.2.3** int `dev_clock_read` ( void \* *dev*, int *fd*, void \* *buf*, int *size* )

References free, mallocn, strlen, strncpy, and time2str.

**5.86.2.4** int `dev_clock_seek` ( void \* *dev*, int *fd*, int *offset*, int *whence* )

**5.86.2.5** int `dev_clock_write` ( void \* *dev*, int *fd*, void \* *buf*, int *size* )

### 5.86.3 Variable Documentation

**5.86.3.1** `device_t dev_clock`

#### Initial value:

```
{
    "/dev/clock",
    6,
    NULL,
    dev_clock_open,
```

```
    dev_clock_close,  
    dev_clock_read,  
    dev_clock_write,  
    dev_clock_seek  
}
```

## 5.87 src/kernel/pm/dev\_framebuffer.c File Reference

The framebuffer device.

```
#include "../include/types.h"  
#include "../include/const.h"  
#include "../include/string.h"  
#include "../include/debug.h"  
#include "../io/io_virtual.h"  
#include "../io/io.h"  
#include "pm_devices.h"  
#include "pm_main.h"
```

### Functions

- [int dev\\_framebuffer\\_open](#) (void \*dev, char \*path, int oflag, int mode)
- [int dev\\_framebuffer\\_close](#) (void \*dev, int fd)
- [int dev\\_framebuffer\\_read](#) (void \*dev, int fd, void \*buf, int size)
- [int dev\\_framebuffer\\_write](#) (void \*dev, int fd, void \*buf, int size)
- [int dev\\_framebuffer\\_seek](#) (void \*dev, int fd, int offset, int whence)

### Variables

- [int dev\\_framebuffer\\_count](#) = 0
- [device\\_t dev\\_framebuffer](#)

### 5.87.1 Detailed Description

The framebuffer device. This provides a 80x25 pixels, 16 color video screen. Video output through the framebuffer device is much faster as it writes directly to the video memory and does not use the conversion functions of the vmonitors. Used primarily for the various demo games.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

Rev:

12

## 5.87.2 Function Documentation

### 5.87.2.1 `int dev_framebuffer_close ( void * dev, int fd )`

References `dev_framebuffer_count`, `virt_monitor::disable_refresh`, and `get_active_virt_monitor()`.

### 5.87.2.2 `int dev_framebuffer_open ( void * dev, char * path, int oflag, int mode )`

References `dev_framebuffer_count`, `virt_monitor::disable_refresh`, `dprintf`, and `get_active_virt_monitor()`.

### 5.87.2.3 `int dev_framebuffer_read ( void * dev, int fd, void * buf, int size )`

### 5.87.2.4 `int dev_framebuffer_seek ( void * dev, int fd, int offset, int whence )`

### 5.87.2.5 `int dev_framebuffer_write ( void * dev, int fd, void * buf, int size )`

References `active_proc`, `focus_proc`, and `VGA_DISPLAY`.

## 5.87.3 Variable Documentation

### 5.87.3.1 `device_t dev_framebuffer`

Initial value:

```
{
    "/dev/framebuffer",
    3,
    &dev_framebuffer_count,
    dev_framebuffer_open,
    dev_framebuffer_close,
    dev_framebuffer_read,
    dev_framebuffer_write,
    dev_framebuffer_seek
}
```

### 5.87.3.2 `int dev_framebuffer_count = 0`

Referenced by `dev_framebuffer_close()`, and `dev_framebuffer_open()`.

## 5.88 `src/kernel/pm/dev_keyboard.c` File Reference

The keyboard device.

```
#include "../include/types.h"
#include "../include/const.h"
#include "../include/string.h"
```

```
#include "../include/debug.h"
#include "../io/io_virtual.h"
#include "pm_devices.h"
```

## Functions

- int [dev\\_keyboard\\_open](#) (void \*dev, char \*path, int oflag, int mode)
- int [dev\\_keyboard\\_close](#) (void \*dev, int fd)
- int [dev\\_keyboard\\_read](#) (void \*dev, int fd, void \*buf, int size)
- int [dev\\_keyboard\\_write](#) (void \*dev, int fd, void \*buf, int size)
- int [dev\\_keyboard\\_seek](#) (void \*dev, int fd, int offset, int whence)

## Variables

- [bool keyboard\\_state](#) [256]  
*State of all keys.*
- [device\\_t dev\\_keyboard](#)

### 5.88.1 Detailed Description

The keyboard device. Provides direct access to the state of all keys. Used to retrieve the state of keys which do not produce ASCII characters.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.88.2 Function Documentation

**5.88.2.1** int [dev\\_keyboard\\_close](#) ( void \* *dev*, int *fd* )

**5.88.2.2** int [dev\\_keyboard\\_open](#) ( void \* *dev*, char \* *path*, int *oflag*, int *mode* )

**5.88.2.3** int [dev\\_keyboard\\_read](#) ( void \* *dev*, int *fd*, void \* *buf*, int *size* )

References [keyboard\\_state](#), and [memcpy](#).

**5.88.2.4** `int dev_keyboard_seek ( void * dev, int fd, int offset, int whence )`

**5.88.2.5** `int dev_keyboard_write ( void * dev, int fd, void * buf, int size )`

### 5.88.3 Variable Documentation

#### 5.88.3.1 `device_t dev_keyboard`

##### Initial value:

```
{
    "/dev/keyboard",
    4,
    NULL,
    dev_keyboard_open,
    dev_keyboard_close,
    dev_keyboard_read,
    dev_keyboard_write,
    dev_keyboard_seek
}
```

#### 5.88.3.2 `bool keyboard_state[256]`

State of all keys.

TRUE means the key is being held down right now. This is used by the `/dev/keyboard` device to provide raw keyboard access.

Referenced by `dev_keyboard_read()`, `io_init()`, and `kb_handler()`.

## 5.89 `src/kernel/pm/dev_null.c` File Reference

The null device.

```
#include "../include/types.h"
#include "../include/const.h"
#include "../include/string.h"
#include "../include/debug.h"
#include "pm_devices.h"
```

### Functions

- `int dev_null_open` (void \*dev, char \*path, int oflag, int mode)
- `int dev_null_close` (void \*dev, int fd)
- `int dev_null_read` (void \*dev, int fd, void \*buf, int size)
- `int dev_null_write` (void \*dev, int fd, void \*buf, int size)
- `int dev_null_seek` (void \*dev, int fd, int offset, int whence)

### Variables

- `device_t dev_null`



## 5.89.1 Detailed Description

The null device. This is just a dummy device. Use its source code as an outline for devices you want to implement.

### Author

Daniel Bader

### LastChangedBy:

dtraytel

### Version

### Rev:

12

## 5.89.2 Function Documentation

**5.89.2.1** `int dev_null_close ( void * dev, int fd )`

**5.89.2.2** `int dev_null_open ( void * dev, char * path, int oflag, int mode )`

**5.89.2.3** `int dev_null_read ( void * dev, int fd, void * buf, int size )`

References `dprintf`, and `memset()`.

**5.89.2.4** `int dev_null_seek ( void * dev, int fd, int offset, int whence )`

**5.89.2.5** `int dev_null_write ( void * dev, int fd, void * buf, int size )`

References `dprintf`.

## 5.89.3 Variable Documentation

### 5.89.3.1 `device_t dev_null`

#### Initial value:

```
{
    "/dev/null",
    0,
    NULL,
    dev_null_open,
    dev_null_close,
    dev_null_read,
    dev_null_write,
    dev_null_seek
}
```

## 5.90 src/kernel/pm/dev\_stdin.c File Reference

The STDIN device.

```
#include "../include/types.h"
#include "../include/const.h"
#include "../include/string.h"
#include "../include/debug.h"
#include "pm_devices.h"
#include "pm_main.h"
#include "../include/ringbuffer.h"
```

### Functions

- [int dev\\_stdin\\_open](#) (void \*dev, char \*path, int oflag, int mode)
- [int dev\\_stdin\\_close](#) (void \*dev, int fd)
- [int dev\\_stdin\\_read](#) (void \*dev, int fd, void \*buf, int size)
- [int dev\\_stdin\\_write](#) (void \*dev, int fd, void \*buf, int size)
- [int dev\\_stdin\\_seek](#) (void \*dev, int fd, int offset, int whence)

### Variables

- [device\\_t dev\\_stdin](#)

### 5.90.1 Detailed Description

The STDIN device. Provides a process with access to its character input queue.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

## 5.90.2 Function Documentation

**5.90.2.1** `int dev_stdin_close ( void * dev, int fd )`

**5.90.2.2** `int dev_stdin_open ( void * dev, char * path, int oflag, int mode )`

**5.90.2.3** `int dev_stdin_read ( void * dev, int fd, void * buf, int size )`

References `active_proc`, `focus_proc`, `halt()`, `rf_getlength()`, `rf_read()`, `process_t::state`, and `process_t::stdin`.

**5.90.2.4** `int dev_stdin_seek ( void * dev, int fd, int offset, int whence )`

**5.90.2.5** `int dev_stdin_write ( void * dev, int fd, void * buf, int size )`

References `active_proc`, `rf_write()`, and `process_t::stdin`.

## 5.90.3 Variable Documentation

**5.90.3.1** `device_t dev_stdin`

**Initial value:**

```
{
    "/dev/stdin",
    2,
    NULL,
    dev_stdin_open,
    dev_stdin_close,
    dev_stdin_read,
    dev_stdin_write,
    dev_stdin_seek
}
```

## 5.91 src/kernel/pm/dev\_stdout.c File Reference

The STDOUT device.

```
#include "../include/types.h"
#include "../include/const.h"
#include "../include/string.h"
#include "../include/debug.h"
#include "../io/io_virtual.h"
#include "syscalls_cli.h"
#include "pm_devices.h"
#include "pm_main.h"
```

### Functions

- `int dev_stdout_open (void *dev, char *path, int oflag, int mode)`

- int `dev_stdout_close` (void \*dev, int fd)
- int `dev_stdout_read` (void \*dev, int fd, void \*buf, int size)
- int `dev_stdout_write` (void \*dev, int fd, void \*buf, int size)
- int `dev_stdout_seek` (void \*dev, int fd, int offset, int whence)

## Variables

- `device_t dev_stdout`

### 5.91.1 Detailed Description

The STDOUT device. Allows a process to write text to the vmonitor assigned to it.

#### Author

Daniel Bader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.91.2 Function Documentation

**5.91.2.1** int `dev_stdout_close` ( void \* *dev*, int *fd* )

**5.91.2.2** int `dev_stdout_open` ( void \* *dev*, char \* *path*, int *oflag*, int *mode* )

**5.91.2.3** int `dev_stdout_read` ( void \* *dev*, int *fd*, void \* *buf*, int *size* )

**5.91.2.4** int `dev_stdout_seek` ( void \* *dev*, int *fd*, int *offset*, int *whence* )

**5.91.2.5** int `dev_stdout_write` ( void \* *dev*, int *fd*, void \* *buf*, int *size* )

References `_free()`, `_malloc()`, `active_proc`, `memcpy`, `virt_monitor_puts()`, and `process_t::vmonitor`.

### 5.91.3 Variable Documentation

#### 5.91.3.1 `device_t dev_stdout`

##### Initial value:

```
{
    "/dev/stdout",
    1,
    NULL,
}
```

```
    dev_stdout_open,  
    dev_stdout_close,  
    dev_stdout_read,  
    dev_stdout_write,  
    dev_stdout_seek  
}
```

## 5.92 src/kernel/pm/pm\_devices.c File Reference

Implementation of the device subsystem.

```
#include "../include/assert.h"  
#include "../include/const.h"  
#include "../include/debug.h"  
#include "../include/init.h"  
#include "../include/string.h"  
#include "../fs/fs_const.h"  
#include "../fs/fs_types.h"  
#include "../fs/fs_file_table.h"  
#include "../fs/fs_main.h"  
#include "pm_devices.h"
```

### Functions

- `device_t * pm_name2device (char *name)`  
*Resolves a device name string to its corresponding `device_t` pointer.*
- `device_t * pm_fd2device (int fd)`  
*Resolves a device handle to its corresponding `device_t` pointer.*
- `void pm_register_device (device_t *dev)`  
*Registers a new device.*

### Variables

- `device_t * devices_head = NULL`  
*Devices list.*

### 5.92.1 Detailed Description

Implementation of the device subsystem.

#### Author

dbader

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

## 5.92.2 Function Documentation

### 5.92.2.1 `device_t* pm_fd2device ( int fd )`

Resolves a device handle to its corresponding [device\\_t](#) pointer.

**Parameters**

*fd* the device handle to resolve

**Returns**

the corresponding device structure or NULL if none could be found

References `device_t::fd`, `device_t::next`, and NULL.

Referenced by `pm_register_device()`, `sys_close()`, `sys_read()`, `sys_seek()`, and `sys_write()`.

### 5.92.2.2 `device_t* pm_name2device ( char * name )`

Resolves a device name string to its corresponding [device\\_t](#) pointer.

**Parameters**

*name* the device name to resolve

**Returns**

the corresponding device structure or NULL if none could be found

References `device_t::name`, `device_t::next`, NULL, and `strcmp`.

Referenced by `pm_register_device()`, and `sys_open()`.

### 5.92.2.3 `void pm_register_device ( device_t * dev )`

Registers a new device.

**Parameters**

*dev* the device struct

References `do_create()`, `do_file_exists()`, `dprintf`, `device_t::fd`, `device_t::name`, `device_t::next`, NULL, `pm_fd2device()`, `pm_name2device()`, and `printf`.

Referenced by `pm_init()`.

### 5.92.3 Variable Documentation

#### 5.92.3.1 `device_t* devices_head = NULL`

Devices list.

Holds all devices currently registered.

## 5.93 src/kernel/pm/pm\_devices.h File Reference

Definitions for the device subsystem.

### Data Structures

- struct `device_t`  
*Describes a single device.*

### Defines

- #define `MAX_DEVICES` 100  
*Maximum number of registered devices.*

### Typedefs

- typedef int(\* `dev_open_func`)(void \*dev, char \*path, int oflag, int mode)
- typedef int(\* `dev_close_func`)(void \*dev, int fd)
- typedef int(\* `dev_read_func`)(void \*dev, int fd, void \*buf, int size)
- typedef int(\* `dev_write_func`)(void \*dev, int fd, void \*buf, int size)
- typedef int(\* `dev_seek_func`)(void \*dev, int fd, int offset, int whence)
- typedef struct `device_t` `device_t`  
*Describes a single device.*

### Functions

- `device_t * pm_name2device` (char \*name)  
*Resolves a device name string to its corresponding `device_t` pointer.*
- `device_t * pm_fd2device` (int fd)  
*Resolves a device handle to its corresponding `device_t` pointer.*
- void `pm_register_device` (`device_t *dev`)  
*Registers a new device.*

### 5.93.1 Detailed Description

Definitions for the device subsystem.

**Author**

dbader

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

### 5.93.2 Define Documentation

#### 5.93.2.1 #define MAX\_DEVICES 100

Maximum number of registered devices.

All file descriptors below this number are assumed to be device handles.

Referenced by `sys_close()`, `sys_read()`, `sys_seek()`, and `sys_write()`.

### 5.93.3 Typedef Documentation

#### 5.93.3.1 typedef int(\* dev\_close\_func)(void \*dev, int fd)

#### 5.93.3.2 typedef int(\* dev\_open\_func)(void \*dev, char \*path, int oflag, int mode)

#### 5.93.3.3 typedef int(\* dev\_read\_func)(void \*dev, int fd, void \*buf, int size)

#### 5.93.3.4 typedef int(\* dev\_seek\_func)(void \*dev, int fd, int offset, int whence)

#### 5.93.3.5 typedef int(\* dev\_write\_func)(void \*dev, int fd, void \*buf, int size)

#### 5.93.3.6 typedef struct device\_t device\_t

Describes a single device.

### 5.93.4 Function Documentation

#### 5.93.4.1 device\_t\* pm\_fd2device ( int *fd* )

Resolves a device handle to its corresponding [device\\_t](#) pointer.

**Parameters**

*fd* the device handle to resolve



**Returns**

the corresponding device structure or NULL if none could be found

References `device_t::fd`, `device_t::next`, and NULL.

Referenced by `pm_register_device()`, `sys_close()`, `sys_read()`, `sys_seek()`, and `sys_write()`.

**5.93.4.2 device\_t\* pm\_name2device ( char \* name )**

Resolves a device name string to its corresponding `device_t` pointer.

**Parameters**

*name* the device name to resolve

**Returns**

the corresponding device structure or NULL if none could be found

References `device_t::name`, `device_t::next`, NULL, and `strcmp`.

Referenced by `pm_register_device()`, and `sys_open()`.

**5.93.4.3 void pm\_register\_device ( device\_t \* dev )**

Registers a new device.

**Parameters**

*dev* the device struct

References `do_create()`, `do_file_exists()`, `dprintf`, `device_t::fd`, `device_t::name`, `device_t::next`, NULL, `pm_fd2device()`, `pm_name2device()`, and `printf`.

Referenced by `pm_init()`.

## 5.94 src/kernel/pm/pm\_input.c File Reference

Process management code that handles keyboard input.

```
#include "../include/const.h"
#include "../include/debug.h"
#include "../io/io.h"
#include "../include/ringbuffer.h"
#include "pm_main.h"
```

**Functions**

- void `pm_handle_input` (char c)  
*Handles keyboard input.*

### 5.94.1 Detailed Description

Process management code that handles keyboard input. After a keypress an interrupt is generated which gets handled by the I/O code. In [io\\_keyboard.c](#) the keyboard scancode gets converted into an ascii character which is then given to [pm\\_handle\\_input\(\)](#).

[pm\\_handle\\_input\(\)](#) then writes the new character to the focussed process' stdin queue. The respective process can then choose to read its stdin queue at any time via a call to `read(STDIN, &buf, len)`.

#### Author

dbader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.94.2 Function Documentation

#### 5.94.2.1 void pm\_handle\_input ( char c )

Handles keyboard input.

This gets called from the I/O code after a keypress was received and translated. [pm\\_handle\\_input\(\)](#) then distributes incoming character to the focussed process' input queue.

#### Parameters

*c* input character

References `dprintf`, `focus_proc`, `NULL`, `PSTATE_STDINSLEEP`, `rf_write()`, `process_t::state`, and `process_t::stdin`.

Referenced by `kb_handler()`.

## 5.95 src/kernel/pm/pm\_input.h File Reference

Input management header.

### Functions

- void [pm\\_handle\\_input](#) (char c)  
*Handles keyboard input.*

## 5.95.1 Detailed Description

Input management header.

### Author

dbader

### LastChangedBy:

dtraytel

### Version

### Rev:

12

## 5.95.2 Function Documentation

### 5.95.2.1 void pm\_handle\_input ( char c )

Handles keyboard input.

This gets called from the I/O code after a keypress was received and translated. [pm\\_handle\\_input\(\)](#) then distributes incoming character to the focussed process' input queue.

### Parameters

*c* input character

References `dprintf`, `focus_proc`, `NULL`, `PSTATE_STDINSLEEP`, `rf_write()`, `process_t::state`, and `process_t::stdin`.

Referenced by `kb_handler()`.

## 5.96 src/kernel/pm/pm\_main.c File Reference

Process management main source file.

```
#include "../include/assert.h"
#include "../include/const.h"
#include "../include/stdio.h"
#include "../include/stdlib.h"
#include "../include/string.h"
#include "../include/debug.h"
#include "../include/ringbuffer.h"
#include "../include/init.h"
#include "../io/io.h"
```

```
#include "../io/io_virtual.h"
#include "../fs/fs_const.h"
#include "../fs/fs_types.h"
#include "../fs/fs_file_table.h"
#include "../fs/fs_main.h"
#include "pm_main.h"
#include "pm_syscalls.h"
#include "syscalls_shared.h"
#include "pm_devices.h"
#include "../../apps/brainfuck_interpreter.h"
```

## Functions

- [uint32 getpid \(\)](#)  
*Returns the PID of the active process.*
- [void pm\\_init \(\)](#)  
*Performs process management initializations.*
- [uint32 pm\\_schedule \(uint32 context\)](#)  
*Switch to the next process in a round robin fashion.*
- [uint32 pm\\_create\\_thread \(char \\*name, void\(\\*entry\)\(\), uint32 stacksize\)](#)  
*Creates a new thread.*
- [void pm\\_destroy\\_thread \(process\\_t \\*proc\)](#)  
*Releases a thread's resources.*
- [void pm\\_set\\_thread\\_priority \(uint32 pid, uint32 prio\)](#)  
*Modifies a thread's priority.*
- [void pm\\_set\\_focus\\_proc \(uint32 pid\)](#)  
*Gives a process the input focus.*
- [process\\_t \\* pm\\_get\\_proc \(uint32 pid\)](#)  
*Returns the process that belongs to the given pid.*
- [void pm\\_kill\\_proc \(uint32 pid\)](#)  
*Kills the process with the given pid.*
- [void aprintf \(char \\*fmt,...\)](#)  
*Prints text to the active process' vmonitor.*
- [void pm\\_dump \(\)](#)  
*Prints some status information about all processes.*

## Variables

- `process_t * procs_head` = NULL  
*Process list.*
- `process_t * active_proc` = NULL  
*Pointer to the active process, ie the process which is executing.*
- `process_t * focus_proc` = NULL  
*Pointer to the process which has the input focus.*
- `process_t * kernel_proc` = NULL  
*Pointer to the kernel process.*
- `uint32 next_pid` = 0  
*Process ID of the next process that gets created.*
- `device_t dev_null`
- `device_t dev_stdout`
- `device_t dev_stdin`
- `device_t dev_brainfuck`
- `device_t dev_framebuffer`
- `device_t dev_keyboard`
- `device_t dev_clock`

### 5.96.1 Detailed Description

Process management main source file.

#### Author

dbader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

21

### 5.96.2 Function Documentation

#### 5.96.2.1 void `aprintf`( char \* *fmt*, ... )

Prints text to the active process' vmonitor.

#### Parameters

*fmt* format string, `sprintf` formatting rules apply.

... any remaining arguments

References `va_end`, `va_start`, `virt_monitor_puts()`, `process_t::vmonitor`, and `vsnprintf`.

Referenced by `pm_dump()`.

### 5.96.2.2 uint32 getpid ( )

Returns the PID of the active process.

#### Returns

the PID

References `ASSERT`, `NULL`, and `process_t::pid`.

Referenced by `sys_exit()`, and `sys_getpid()`.

### 5.96.2.3 uint32 pm\_create\_thread ( char \* name, void(\*)() entry, uint32 stacksize )

Creates a new thread.

#### Parameters

*name* the process's name

*entry* the entry point

*stacksize* the stack size

#### Returns

the PID of the new thread

References `ASSERT`, `clear_interrupts()`, `process_t::context`, `dprintf`, `init_proc_file_table()`, `mallocn`, `process_t::name`, `process_t::next`, `next_pid`, `NULL`, `panic`, `process_t::pft`, `process_t::pid`, `process_t::priority`, `process_t::remaining_timeslices`, `rf_alloc()`, `set_interrupts()`, `process_t::stack_start`, `start_vmonitor()`, `process_t::state`, `process_t::stdin`, `STDIN_QUEUE_SIZE`, `strdup`, and `process_t::vmonitor`.

Referenced by `make_snapshot()`, `new_shell()`, `paralleleModellierung()`, `shell_cmd_memview()`, `shell_cmd_snake()`, `shell_cmd_speed()`, `shell_cmd_synth()`, `syscall_test()`, `threadA_test()`, `threadB_test()`, `threadC_test()`, `threadConsumer_test()`, `threadD_test()`, and `threadProducer_test()`.

### 5.96.2.4 void pm\_destroy\_thread ( process\_t \* proc )

Releases a thread's resources.

The only place this should be called from is `pm_schedule()` - to mark a process for destruction, set its "dead" flag.

#### See also

[pm\\_schedule](#)

#### Parameters

*proc* the process to destroy

References `dprintf`, `free`, `free_virt_monitor()`, `process_t::name`, `NULL`, `process_t::pid`, `rf_free()`, `process_t::stack_start`, `process_t::stdin`, and `process_t::vmonitor`.

Referenced by `pm_schedule()`.

#### 5.96.2.5 void pm\_dump ( )

Prints some status information about all processes.

Can be used as a crude form of unix's "ps" command.

References `aprintf()`, `process_t::context`, `process_t::name`, `process_t::next`, `p()`, `process_t::pid`, and `process_t::priority`.

Referenced by `shell_cmd_ps()`.

#### 5.96.2.6 process\_t\* pm\_get\_proc ( uint32 pid )

Returns the process that belongs to the given pid.

##### Parameters

*pid* the pid

##### Returns

process that belongs to the given pid, `NULL` if not found.

References `process_t::next`, `p()`, and `process_t::pid`.

Referenced by `free_virt_monitor()`, `pm_kill_proc()`, `pm_set_focus_proc()`, and `pm_set_thread_priority()`.

#### 5.96.2.7 void pm\_init ( )

Performs process management initializations.

References `do_file_exists()`, `do_mkdir()`, `dprint_separator`, `dprintf`, `get_active_virt_monitor()`, `init_bf()`, `init_proc_file_table()`, `mallocn`, `MAX_SYSCALL`, `memset()`, `process_t::name`, `process_t::next`, `next_pid`, `process_t::pft`, `process_t::pid`, `pm_register_device()`, `process_t::priority`, `process_t::remaining_timeslices`, `rf_alloc()`, `process_t::state`, `process_t::stdin`, `STDIN_QUEUE_SIZE`, and `process_t::vmonitor`.

Referenced by `main()`.

#### 5.96.2.8 void pm\_kill\_proc ( uint32 pid )

Kills the process with the given pid.

##### Parameters

*pid* the process id

References `p()`, `pm_get_proc()`, and `process_t::state`.

Referenced by `sys_kill()`.

### 5.96.2.9 uint32 pm\_schedule ( uint32 context )

Switch to the next process in a round robin fashion.

#### Parameters

*context* the tasks context on the stack

#### Returns

the context of the process which becomes active

References process\_t::context, process\_t::next, NULL, pm\_destroy\_thread(), process\_t::priority, PSTATE\_ALIVE, PSTATE\_DEAD, PSTATE\_STDINSLEEP, process\_t::remaining\_timeslices, and process\_t::state.

Referenced by timer\_handler().

### 5.96.2.10 void pm\_set\_focus\_proc ( uint32 pid )

Gives a process the input focus.

#### Parameters

*pid* the pid of the process receiving the focus

References pm\_get\_proc().

Referenced by free\_virt\_monitor(), switch\_monitor\_down(), and switch\_monitor\_up().

### 5.96.2.11 void pm\_set\_thread\_priority ( uint32 pid, uint32 prio )

Modifies a thread's priority.

Note that this will not affect the remaining timeslices of a thread that is currently active in order to keep other threads from starving.

#### Parameters

*pid* the process id

*prio* the new priority

References dprintf, p(), pm\_get\_proc(), and process\_t::priority.

Referenced by shell\_cmd\_nice(), and threadD\_test().

## 5.96.3 Variable Documentation

### 5.96.3.1 process\_t\* active\_proc = NULL

Pointer to the active process, ie the process which is executing.

Referenced by dev\_framebuffer\_write(), dev\_stdin\_read(), dev\_stdin\_write(), dev\_stdout\_write(), snake(), speed(), sys\_close(), sys\_exit(), sys\_malloc(), sys\_open(), sys\_read(), sys\_seek(), sys\_stat(), and sys\_write().



**5.96.3.2 device\_t dev\_brainfuck****5.96.3.3 device\_t dev\_clock****5.96.3.4 device\_t dev\_framebuffer****5.96.3.5 device\_t dev\_keyboard****5.96.3.6 device\_t dev\_null****5.96.3.7 device\_t dev\_stdin****5.96.3.8 device\_t dev\_stdout****5.96.3.9 process\_t\* focus\_proc = NULL**

Pointer to the process which has the input focus.

Referenced by dev\_framebuffer\_write(), dev\_stdin\_read(), and pm\_handle\_input().

**5.96.3.10 process\_t\* kernel\_proc = NULL**

Pointer to the kernel process.

**5.96.3.11 uint32 next\_pid = 0**

Process ID of the next process that gets created.

Referenced by pm\_create\_thread(), and pm\_init().

**5.96.3.12 process\_t\* procs\_head = NULL**

Process list.

## 5.97 src/kernel/pm/pm\_main.h File Reference

Process management main header file.

```
#include "../include/ringbuffer.h"
#include "../fs/fs_types.h"
#include "../fs/fs_const.h"
#include "../io/io_virtual.h"
#include "../include/ringbuffer.h"
#include "../io/io_virtual.h"
```

### Data Structures

- struct [cpu\\_state\\_t](#)

*Represents the state of all cpu registers at a given time.*

- struct [process\\_t](#)  
*Structure describing a single process.*

## Defines

- #define [PSTATE\\_ALIVE](#) 0  
*The process is alive, ie it will get executed in one of the next [pm\\_schedule\(\)](#) calls.*
- #define [PSTATE\\_DEAD](#) 1  
*The process is dead, ie it will get purged in one of the next [pm\\_schedule\(\)](#) calls.*
- #define [PSTATE\\_STDINSLEEP](#) 2  
*Process is sleeping, due to stdin requesting input, but none available.*
- #define [STDIN\\_QUEUE\\_SIZE](#) 512  
*Maximum number of input bytes the STDIN data structure can queue up.*

## Typedefs

- typedef struct [process\\_t](#) [process\\_t](#)  
*Structure describing a single process.*

## Functions

- [uint32](#) [getpid](#) ()  
*Returns the PID of the active process.*
- void [pm\\_init](#) ()  
*Performs process management initializations.*
- [uint32](#) [pm\\_schedule](#) ([uint32](#) context)  
*Switch to the next process in a round robin fashion.*
- [uint32](#) [pm\\_create\\_thread](#) (char \*name, void(\*entry)(), [uint32](#) stacksize)  
*Creates a new thread.*
- void [pm\\_destroy\\_thread](#) ([process\\_t](#) \*proc)  
*Releases a thread's resources.*
- [process\\_t](#) \* [pm\\_get\\_proc](#) ([uint32](#) pid)  
*Returns the process that belongs to the given pid.*
- void [pm\\_set\\_thread\\_priority](#) ([uint32](#) pid, [uint32](#) prio)  
*Modifies a thread's priority.*

- void `pm_set_focus_proc` (uint32 pid)  
*Gives a process the input focus.*
- void `pm_kill_proc` (uint32 pid)  
*Kills the process with the given pid.*
- void `_syscall` (uint32 id, void \*data)

## Variables

- `process_t * procs_head`  
*Process list.*
- `process_t * active_proc`  
*Pointer to the active process, ie the process which is executing.*
- `process_t * focus_proc`  
*Pointer to the process which has the input focus.*
- `process_t * kernel_proc`  
*Pointer to the kernel process.*

### 5.97.1 Detailed Description

Process management main header file.

#### Author

dbader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

21

### 5.97.2 Define Documentation

#### 5.97.2.1 #define PSTATE\_ALIVE 0

The process is alive, ie it will get executed in one of the next `pm_schedule()` calls.

Referenced by `pm_schedule()`.

### 5.97.2.2 #define PSTATE\_DEAD 1

The process is dead, ie it will get purged in one of the next `pm_schedule()` calls.

Referenced by `pm_schedule()`.

### 5.97.2.3 #define PSTATE\_STDINSLEEP 2

Process is sleeping, due to stdin requesting input, but none available.

Referenced by `pm_handle_input()`, and `pm_schedule()`.

### 5.97.2.4 #define STDIN\_QUEUE\_SIZE 512

Maximum number of input bytes the STDIN data structure can queue up.

Referenced by `pm_create_thread()`, and `pm_init()`.

## 5.97.3 Typedef Documentation

### 5.97.3.1 typedef struct process\_t process\_t

Structure describing a single process.

TODO: Maybe move this into `pm_process.h`

## 5.97.4 Function Documentation

### 5.97.4.1 void \_syscall ( uint32 id, void \* data )

Referenced by `_close()`, `_exit()`, `_free()`, `_getpid()`, `_kill()`, `_log()`, `_malloc()`, `_open()`, `_read()`, `_seek()`, `_stat()`, `_unlink()`, and `_write()`.

### 5.97.4.2 uint32 getpid ( )

Returns the PID of the active process.

#### Returns

the PID

References `ASSERT`, `NULL`, and `process_t::pid`.

Referenced by `sys_exit()`, and `sys_getpid()`.

### 5.97.4.3 uint32 pm\_create\_thread ( char \* name, void(\*)() entry, uint32 stacksize )

Creates a new thread.

#### Parameters

*name* the process's name

*entry* the entry point

*stacksize* the stack size

### Returns

the PID of the new thread

References ASSERT, clear\_interrupts(), process\_t::context, dprintf, init\_proc\_file\_table(), mallocn, process\_t::name, process\_t::next, next\_pid, NULL, panic, process\_t::pft, process\_t::pid, process\_t::priority, process\_t::remaining\_timeslices, rf\_alloc(), set\_interrupts(), process\_t::stack\_start, start\_vmonitor(), process\_t::state, process\_t::stdin, STDIN\_QUEUE\_SIZE, strdup, and process\_t::vmonitor.

Referenced by make\_snapshot(), new\_shell(), paralleleModellierung(), shell\_cmd\_memview(), shell\_cmd\_snake(), shell\_cmd\_speed(), shell\_cmd\_synth(), syscall\_test(), threadA\_test(), threadB\_test(), threadC\_test(), threadConsumer\_test(), threadD\_test(), and threadProducer\_test().

#### 5.97.4.4 void pm\_destroy\_thread ( process\_t \* *proc* )

Releases a thread's resources.

The only place this should be called from is [pm\\_schedule\(\)](#) - to mark a process for destruction, set its "dead" flag.

### See also

[pm\\_schedule](#)

### Parameters

*proc* the process to destroy

References dprintf, free, free\_virt\_monitor(), process\_t::name, NULL, process\_t::pid, rf\_free(), process\_t::stack\_start, process\_t::stdin, and process\_t::vmonitor.

Referenced by pm\_schedule().

#### 5.97.4.5 process\_t\* pm\_get\_proc ( uint32 *pid* )

Returns the process that belongs to the given pid.

### Parameters

*pid* the pid

### Returns

process that belongs to the given pid, NULL if not found.

References process\_t::next, p(), and process\_t::pid.

Referenced by free\_virt\_monitor(), pm\_kill\_proc(), pm\_set\_focus\_proc(), and pm\_set\_thread\_priority().

#### 5.97.4.6 void pm\_init ( )

Performs process management initializations.

References do\_file\_exists(), do\_mkdir(), dprint\_separator, dprintf, get\_active\_virt\_monitor(), init\_bf(), init\_proc\_file\_table(), mallocn, MAX\_SYSCALL, memset(), process\_t::name, process\_t::next, next\_pid, process\_t::pft, process\_t::pid, pm\_register\_device(), process\_t::priority, process\_t::remaining\_timeslices, rf\_alloc(), process\_t::state, process\_t::stdin, STDIN\_QUEUE\_SIZE, and process\_t::vmonitor.

#### 5.97.4.7 void pm\_kill\_proc ( uint32 pid )

Kills the process with the given pid.

##### Parameters

*pid* the process id

References p(), pm\_get\_proc(), and process\_t::state.

Referenced by sys\_kill().

#### 5.97.4.8 uint32 pm\_schedule ( uint32 context )

Switch to the next process in a round robin fashion.

##### Parameters

*context* the tasks context on the stack

##### Returns

the context of the process which becomes active

References process\_t::context, process\_t::next, NULL, pm\_destroy\_thread(), process\_t::priority, PSTATE\_ALIVE, PSTATE\_DEAD, PSTATE\_STDINSLEEP, process\_t::remaining\_timeslices, and process\_t::state.

Referenced by timer\_handler().

#### 5.97.4.9 void pm\_set\_focus\_proc ( uint32 pid )

Gives a process the input focus.

##### Parameters

*pid* the pid of the process receiving the focus

References pm\_get\_proc().

Referenced by free\_virt\_monitor(), switch\_monitor\_down(), and switch\_monitor\_up().

#### 5.97.4.10 void pm\_set\_thread\_priority ( uint32 pid, uint32 prio )

Modifies a thread's priority.

Note that this will not affect the remaining timeslices of a thread that is currently active in order to keep other threads from starving.

##### Parameters

*pid* the process id

*prio* the new priority

References `dprintf`, `p()`, `pm_get_proc()`, and `process_t::priority`.

Referenced by `shell_cmd_nice()`, and `threadD_test()`.

### 5.97.5 Variable Documentation

#### 5.97.5.1 process\_t\* active\_proc

Pointer to the active process, ie the process which is executing.

Referenced by `dev_framebuffer_write()`, `dev_stdin_read()`, `dev_stdin_write()`, `dev_stdout_write()`, `snake()`, `speed()`, `sys_close()`, `sys_exit()`, `sys_malloc()`, `sys_open()`, `sys_read()`, `sys_seek()`, `sys_stat()`, and `sys_write()`.

#### 5.97.5.2 process\_t\* focus\_proc

Pointer to the process which has the input focus.

Referenced by `dev_framebuffer_write()`, `dev_stdin_read()`, and `pm_handle_input()`.

#### 5.97.5.3 process\_t\* kernel\_proc

Pointer to the kernel process.

#### 5.97.5.4 process\_t\* procs\_head

Process list.

## 5.98 src/kernel/pm/pm\_syscalls.c File Reference

This is the kernel side implementation of the syscalls.

```
#include "../include/assert.h"
#include "../include/const.h"
#include "../include/stdio.h"
#include "../include/stdlib.h"
#include "../include/string.h"
```

```
#include "../include/debug.h"
#include "../include/ringbuffer.h"
#include "pm_main.h"
#include "syscalls_shared.h"
#include "../fs/fs_file_table.h"
#include "../fs/fs_main.h"
#include "../fs/fs_io_functions.h"
#include "../include/init.h"
#include "pm_syscalls.h"
#include "pm_devices.h"
```

## Defines

- #define [SYSCALL\\_TRACE](#)  
*Syscall trace macro.*

## Functions

- void [pm\\_syscall](#) (uint32 id, void \*data)  
*The syscall dispatch function.*
- void [sys\\_log](#) (void \*data)  
*void log(char \*msg) Prints debug logging output to the console.*
- void [sys\\_exit](#) (void \*data)  
*void \_exit(int status) Performs normal program termination.*
- void [sys\\_getpid](#) (void \*data)  
*int \_getpid();*
- void [sys\\_open](#) (void \*data)  
*int \_open(char \*path, int oflag, ...);*
- void [sys\\_close](#) (void \*data)  
*int \_close(int fd);*
- void [sys\\_read](#) (void \*data)  
*int \_read(int fd, void \*buf, int size);*
- void [sys\\_write](#) (void \*data)  
*int \_write(int fd, void \*buf, int size);*
- void [sys\\_seek](#) (void \*data)  
*int \_seek(int fd, int offset, int whence);*



- void `sys_malloc` (void \*data)  
`void* _malloc(size_t size);`
- void `sys_free` (void \*data)  
`void _free(void *block);`
- void `sys_unlink` (void \*data)  
`int _unlink(char *path)`
- void `sys_stat` (void \*data)  
`int _stat(char *path, stat *buf)`
- void `sys_kill` (void \*data)  
`void _kill(int pid)`

## Variables

- `syscall_handler syscall_table []`  
*The system call table.*

### 5.98.1 Detailed Description

This is the kernel side implementation of the syscalls.

#### Author

dbader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.98.2 Define Documentation

#### 5.98.2.1 #define SYSCALL\_TRACE

Syscall trace macro.

Uncomment to print a message everytime a syscall gets executed.

Referenced by `sys_close()`, `sys_exit()`, `sys_free()`, `sys_getpid()`, `sys_kill()`, `sys_log()`, `sys_malloc()`, `sys_open()`, `sys_read()`, `sys_seek()`, `sys_stat()`, `sys_unlink()`, and `sys_write()`.

### 5.98.3 Function Documentation

#### 5.98.3.1 void pm\_syscall ( uint32 *id*, void \* *data* )

The syscall dispatch function.

This gets called whenever a thread request a syscall by raising the syscall interrupt (int 0x42). It checks the syscall id for validity and calls the appropriate syscall handler.

#### See also

[syscall\\_table](#)  
[syscalls\\_shared.h](#)

#### Parameters

*id* the syscall id number  
*data* pointer to the syscall argument structure

References MAX\_SYSCALL, panic, set\_interrupts(), and syscall\_table.

#### 5.98.3.2 void sys\_close ( void \* *data* )

[int \\_close\(int fd\)](#);

References active\_proc, device\_t::close, do\_close\_pf(), EOF, sc\_close\_args\_t::fd, MAX\_DEVICES, process\_t::pft, pm\_fd2device(), sc\_close\_args\_t::success, and SYSCALL\_TRACE.

#### 5.98.3.3 void sys\_exit ( void \* *data* )

[void \\_exit\(int status\)](#) Performs normal program termination.

exit() marks the current process as terminated ("dead"), informs any waiting parents but does not yet destroy the process. Freeing process resources is done by [pm\\_schedule\(\)](#).

#### Parameters

*data* arguments

References active\_proc, ASSERT, dprintf, getpid(), NULL, panic, process\_t::state, and SYSCALL\_TRACE.

#### 5.98.3.4 void sys\_free ( void \* *data* )

[void \\_free\(void \\*block\)](#);

References free, and SYSCALL\_TRACE.

#### 5.98.3.5 void sys\_getpid ( void \* *data* )

[int \\_getpid\(\)](#);

References getpid(), NULL, and SYSCALL\_TRACE.

**5.98.3.6 void sys\_kill ( void \* data )**

void [\\_kill\(int pid\)](#)

References [pm\\_kill\\_proc\(\)](#), and [SYSCALL\\_TRACE](#).

**5.98.3.7 void sys\_log ( void \* data )**

void [log\(char \\*msg\)](#) Prints debug logging output to the console.

**Parameters**

*data* arguments

References [puts](#), and [SYSCALL\\_TRACE](#).

**5.98.3.8 void sys\_malloc ( void \* data )**

void\* [\\_malloc\(size\\_t size\)](#);

References [active\\_proc](#), [mallocn](#), [sc\\_malloc\\_args\\_t::mem](#), [process\\_t::name](#), [sc\\_malloc\\_args\\_t::size](#), and [SYSCALL\\_TRACE](#).

**5.98.3.9 void sys\_open ( void \* data )**

int [\\_open\(char \\*path, int oflag, ...\)](#);

References [active\\_proc](#), [do\\_mkdir\(\)](#), [do\\_mkfile\(\)](#), [do\\_open\(\)](#), [sc\\_open\\_args\\_t::fd](#), [free](#), [insert\\_proc\\_file\(\)](#), [sc\\_open\\_args\\_t::mode](#), [NOT\\_POSSIBLE](#), [NULL](#), [O\\_CREAT](#), [sc\\_open\\_args\\_t::oflag](#), [device\\_t::open](#), [sc\\_open\\_args\\_t::path](#), [process\\_t::pft](#), [pm\\_name2device\(\)](#), [strdup](#), [strlen](#), and [SYSCALL\\_TRACE](#).

**5.98.3.10 void sys\_read ( void \* data )**

int [\\_read\(int fd, void \\*buf, int size\)](#);

References [active\\_proc](#), [sc\\_read\\_write\\_args\\_t::buf](#), [do\\_read\(\)](#), [sc\\_read\\_write\\_args\\_t::fd](#), [get\\_proc\\_file\(\)](#), [MAX\\_DEVICES](#), [proc\\_file::pf\\_f\\_desc](#), [proc\\_file::pf\\_pos](#), [process\\_t::pft](#), [pm\\_fd2device\(\)](#), [device\\_t::read](#), [sc\\_read\\_write\\_args\\_t::rw\\_count](#), [sc\\_read\\_write\\_args\\_t::size](#), and [SYSCALL\\_TRACE](#).

**5.98.3.11 void sys\_seek ( void \* data )**

int [\\_seek\(int fd, int offset, int whence\)](#);

References [active\\_proc](#), [file::f\\_inode](#), [sc\\_seek\\_args\\_t::fd](#), [get\\_file\(\)](#), [get\\_proc\\_file\(\)](#), [m\\_inode::i\\_size](#), [MAX\\_DEVICES](#), [sc\\_seek\\_args\\_t::offset](#), [proc\\_file::pf\\_f\\_desc](#), [proc\\_file::pf\\_pos](#), [process\\_t::pft](#), [pm\\_fd2device\(\)](#), [sc\\_seek\\_args\\_t::pos](#), [device\\_t::seek](#), [SEEK\\_CUR](#), [SEEK\\_END](#), [SEEK\\_SET](#), [SYSCALL\\_TRACE](#), and [sc\\_seek\\_args\\_t::whence](#).

**5.98.3.12 void sys\_stat ( void \* data )**

int [\\_stat\(char \\*path, stat \\*buf\)](#)

References `active_proc`, `sc_stat_args_t::buf`, `bzero`, `do_close_pf()`, `do_open()`, `dprintf`, `get_file_info()`, `insert_proc_file()`, `memcpy`, `NOT_FOUND`, `sc_stat_args_t::path`, `process_t::pft`, `RED`, `sc_stat_args_t::success`, and `SYSCALL_TRACE`.

### 5.98.3.13 `void sys_unlink ( void * data )`

`int _unlink(char *path)`

References `DIRECTORY`, `do_close()`, `do_open()`, `do_remove()`, `dprintf`, `get_file()`, `get_file_info()`, `file_info::mode`, `NOT_FOUND`, `sc_unlink_args_t::path`, `RED`, `file_info::size`, `sc_unlink_args_t::success`, and `SYSCALL_TRACE`.

### 5.98.3.14 `void sys_write ( void * data )`

`int _write(int fd, void *buf, int size);`

References `active_proc`, `sc_read_write_args_t::buf`, `DIRECTORY`, `do_write()`, `sc_read_write_args_t::fd`, `get_file_info()`, `get_proc_file()`, `MAX_DEVICES`, `file_info::mode`, `proc_file::pf_f_desc`, `proc_file::pf_pos`, `process_t::pft`, `pm_fd2device()`, `sc_read_write_args_t::rw_count`, `sc_read_write_args_t::size`, `SYSCALL_TRACE`, and `device_t::write`.

## 5.98.4 Variable Documentation

### 5.98.4.1 `syscall_handler syscall_table[]`

**Initial value:**

```
{
    sys_log,
    sys_exit,
    sys_getpid,
    sys_open,
    sys_close,
    sys_read,
    sys_write,
    sys_seek,
    sys_malloc,
    sys_free,
    sys_unlink,
    sys_stat,
    sys_kill
}
```

The system call table.

Make sure this corresponds to the constants defined in `syscall_shared.h` (otherwise mayhem ensues).

Referenced by `pm_syscall()`.

## 5.99 `src/kernel/pm/pm_syscalls.h` File Reference

Header file for the kernel side syscall implementations.

## Typedefs

- typedef void(\* [syscall\\_handler](#) )(void \*data)  
*Pointer to void sys\_XXX(void \*data) function.*

## Functions

- void [sys\\_log](#) (void \*data)  
*void log(char \*msg) Prints debug logging output to the console.*
- void [sys\\_exit](#) (void \*data)  
*void \_exit(int status) Performs normal program termination.*
- void [sys\\_getpid](#) (void \*data)  
*int \_getpid();*
- void [sys\\_open](#) (void \*data)  
*int \_open(char \*path, int oflag, ...);*
- void [sys\\_close](#) (void \*data)  
*int \_close(int fd);*
- void [sys\\_read](#) (void \*data)  
*int \_read(int fd, void \*buf, int size);*
- void [sys\\_write](#) (void \*data)  
*int \_write(int fd, void \*buf, int size);*
- void [sys\\_seek](#) (void \*data)  
*int \_seek(int fd, int offset, int whence);*
- void [sys\\_malloc](#) (void \*data)  
*void\* \_malloc(size\_t size);*
- void [sys\\_free](#) (void \*data)  
*void \_free(void \*block);*
- void [sys\\_unlink](#) (void \*data)  
*int \_unlink(char \*path)*
- void [sys\\_stat](#) (void \*data)  
*int \_stat(char \*path, stat \*buf)*
- void [sys\\_kill](#) (void \*data)  
*void \_kill(int pid)*

## Variables

- [syscall\\_handler syscall\\_table](#) []  
*The system call table.*

### 5.99.1 Detailed Description

Header file for the kernel side syscall implementations.

#### Author

dbader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.99.2 Typedef Documentation

#### 5.99.2.1 typedef void(\* syscall\_handler)(void \*data)

Pointer to void sys\_XXX(void \*data) function.

### 5.99.3 Function Documentation

#### 5.99.3.1 void sys\_close ( void \* data )

[int \\_close\(int fd\)](#);

References [active\\_proc](#), [device\\_t::close](#), [do\\_close\\_pf\(\)](#), [EOF](#), [sc\\_close\\_args\\_t::fd](#), [MAX\\_DEVICES](#), [process\\_t::pft](#), [pm\\_fd2device\(\)](#), [sc\\_close\\_args\\_t::success](#), and [SYSCALL\\_TRACE](#).

#### 5.99.3.2 void sys\_exit ( void \* data )

[void \\_exit\(int status\)](#) Performs normal program termination.

[exit\(\)](#) marks the current process as terminated ("dead"), informs any waiting parents but does not yet destroy the process. Freeing process resources is done by [pm\\_schedule\(\)](#).

#### Parameters

*data* arguments

References [active\\_proc](#), [ASSERT](#), [dprintf](#), [getpid\(\)](#), [NULL](#), [panic](#), [process\\_t::state](#), and [SYSCALL\\_TRACE](#).

**5.99.3.3 void sys\_free ( void \* data )**

void [\\_free](#)(void \*block);

References [free](#), and [SYSCALL\\_TRACE](#).

**5.99.3.4 void sys\_getpid ( void \* data )**

int [\\_getpid](#)();

References [getpid](#)(), [NULL](#), and [SYSCALL\\_TRACE](#).

**5.99.3.5 void sys\_kill ( void \* data )**

void [\\_kill](#)(int pid)

References [pm\\_kill\\_proc](#)(), and [SYSCALL\\_TRACE](#).

**5.99.3.6 void sys\_log ( void \* data )**

void [log](#)(char \*msg) Prints debug logging output to the console.

**Parameters**

*data* arguments

References [puts](#), and [SYSCALL\\_TRACE](#).

**5.99.3.7 void sys\_malloc ( void \* data )**

void\* [\\_malloc](#)(size\_t size);

References [active\\_proc](#), [mallocn](#), [sc\\_malloc\\_args\\_t::mem](#), [process\\_t::name](#), [sc\\_malloc\\_args\\_t::size](#), and [SYSCALL\\_TRACE](#).

**5.99.3.8 void sys\_open ( void \* data )**

int [\\_open](#)(char \*path, int oflag, ...);

References [active\\_proc](#), [do\\_mkdir](#)(), [do\\_mkfile](#)(), [do\\_open](#)(), [sc\\_open\\_args\\_t::fd](#), [free](#), [insert\\_proc\\_file](#)(), [sc\\_open\\_args\\_t::mode](#), [NOT\\_POSSIBLE](#), [NULL](#), [O\\_CREAT](#), [sc\\_open\\_args\\_t::oflag](#), [device\\_t::open](#), [sc\\_open\\_args\\_t::path](#), [process\\_t::pft](#), [pm\\_name2device](#)(), [strdup](#), [strlen](#), and [SYSCALL\\_TRACE](#).

**5.99.3.9 void sys\_read ( void \* data )**

int [\\_read](#)(int fd, void \*buf, int size);

References [active\\_proc](#), [sc\\_read\\_write\\_args\\_t::buf](#), [do\\_read](#)(), [sc\\_read\\_write\\_args\\_t::fd](#), [get\\_proc\\_file](#)(), [MAX\\_DEVICES](#), [proc\\_file::pf\\_f\\_desc](#), [proc\\_file::pf\\_pos](#), [process\\_t::pft](#), [pm\\_fd2device](#)(), [device\\_t::read](#), [sc\\_read\\_write\\_args\\_t::rw\\_count](#), [sc\\_read\\_write\\_args\\_t::size](#), and [SYSCALL\\_TRACE](#).

**5.99.3.10 void sys\_seek ( void \* data )**

```
int _seek(int fd, int offset, int whence);
```

References active\_proc, file::f\_inode, sc\_seek\_args\_t::fd, get\_file(), get\_proc\_file(), m\_inode::i\_size, MAX\_DEVICES, sc\_seek\_args\_t::offset, proc\_file::pf\_f\_desc, proc\_file::pf\_pos, process\_t::pft, pm\_fd2device(), sc\_seek\_args\_t::pos, device\_t::seek, SEEK\_CUR, SEEK\_END, SEEK\_SET, SYSCALL\_TRACE, and sc\_seek\_args\_t::whence.

**5.99.3.11 void sys\_stat ( void \* data )**

```
int _stat(char *path, stat *buf)
```

References active\_proc, sc\_stat\_args\_t::buf, bzero, do\_close\_pf(), do\_open(), dprintf, get\_file\_info(), insert\_proc\_file(), memcpy, NOT\_FOUND, sc\_stat\_args\_t::path, process\_t::pft, RED, sc\_stat\_args\_t::success, and SYSCALL\_TRACE.

**5.99.3.12 void sys\_unlink ( void \* data )**

```
int _unlink(char *path)
```

References DIRECTORY, do\_close(), do\_open(), do\_remove(), dprintf, get\_file(), get\_file\_info(), file\_info::mode, NOT\_FOUND, sc\_unlink\_args\_t::path, RED, file\_info::size, sc\_unlink\_args\_t::success, and SYSCALL\_TRACE.

**5.99.3.13 void sys\_write ( void \* data )**

```
int _write(int fd, void *buf, int size);
```

References active\_proc, sc\_read\_write\_args\_t::buf, DIRECTORY, do\_write(), sc\_read\_write\_args\_t::fd, get\_file\_info(), get\_proc\_file(), MAX\_DEVICES, file\_info::mode, proc\_file::pf\_f\_desc, proc\_file::pf\_pos, process\_t::pft, pm\_fd2device(), sc\_read\_write\_args\_t::rw\_count, sc\_read\_write\_args\_t::size, SYSCALL\_TRACE, and device\_t::write.

**5.99.4 Variable Documentation****5.99.4.1 syscall\_handler syscall\_table[ ]**

The system call table.

Make sure this corresponds to the constants defined in syscall\_shared.h (otherwise mayhem ensues).

Referenced by pm\_syscall().

**5.100 src/kernel/pm/syscalls\_cli.c File Reference**

This contains all the syscall definitions for the "client" side.

```
#include "pm_main.h"
#include "syscalls_shared.h"
```



## Functions

- void `_log` (char \*msg)  
*Writes a string to the kernel debug monitor.*
- void `_exit` (int status)  
*Exits the calling process and releases its resources.*
- int `_getpid` ()  
*Returns the PID (process id) of the calling process.*
- int `_open` (char \*path, int oflag, int mode)  
*Opens a file or a device.*
- int `_close` (int fd)  
*Closes a open file handle.*
- int `_read` (int fd, void \*buf, int size)  
*Reads data from a file or device.*
- int `_write` (int fd, void \*buf, int size)  
*Writes data into a file or device.*
- int `_seek` (int fd, int offset, int whence)  
*Moves the file pointer in a file or device.*
- void \* `_malloc` (size\_t size)  
*Allocates new memory.*
- void `_free` (void \*block)  
*Frees memory previously allocated through `malloc()`.*
- int `_unlink` (char \*path)  
*Deletes a filename and possible the file it refers to.*
- int `_stat` (char \*path, stat \*buf)  
*Returns extended file information for a single filename.*
- void `_kill` (int pid)  
*Kills the process with the given pid.*

### 5.100.1 Detailed Description

This contains all the syscall definitions for the "client" side. Every etiOS userspace program needs this file to make system calls.

Most of the code here consists of simple stubs which convert between the function's arguments and the argument format expected by the kernel.

See [syscalls\\_shared.h](#) for the definitions of the argument data structures (sc\_XXX\_args\_t).

**Author**

dbader

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

**5.100.2 Function Documentation****5.100.2.1 int \_close ( int *fd* )**

Closes a open file handle.

**Parameters**

*fd* the file descriptor of the file to close

**Returns**

0 on success, -1 on error

References `_syscall()`, `sc_close_args_t::fd`, `sc_close_args_t::success`, and `SYS_CLOSE`.

Referenced by `init_bf()`, `memview_main()`, `mv_switch_to_textmode()`, `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cd()`, `shell_cmd_cp()`, `shell_cmd_date()`, `shell_cmd_exec()`, `shell_cmd_ls()`, `shell_cmd_mkdir()`, `shell_cmd_pong()`, `shell_cmd_snapshot()`, `shell_cmd_speed()`, `shell_cmd_touch()`, `shell_cmd_write()`, `shell_main()`, `snake()`, `snapshot()`, `speed()`, `synth()`, `syscall_test_thread()`, and `test_batch_files()`.

**5.100.2.2 void \_exit ( int *status* )**

Exits the calling process and releases its resources.

**Parameters**

*status* the return status

References `_syscall()`, `SYS_EXIT`, and `WAIT_FOR_INTERRUPT`.

Referenced by `memview_main()`, `shell_cmd_exit()`, `shell_main()`, `snake()`, `snapshot()`, `speed()`, `synth()`, and `syscall_test_thread()`.

**5.100.2.3 void \_free ( void \* *block* )**

Frees memory previously allocated through `malloc()`.

**See also**

[\\_malloc](#)

**Parameters**

*block* pointer to a memory block allocated through malloc

References `_syscall()`, and `SYS_FREE`.

Referenced by `dev_stdout_write()`, `free_all_dummy_blocks()`, `free_dummy_block()`, `interpret_bf()`, `new_-shell()`, `reset_bf()`, `resize_buf()`, `shell_cmd_clear()`, `shell_cmd_exec()`, `shell_cmd_exit()`, `shell_cmd_-snake()`, `shell_handle_command()`, `snapshot()`, `speed()`, `syscall_test_thread()`, `threadA()`, and `threadB()`.

**5.100.2.4 int \_getpid ( )**

Returns the PID (process id) of the calling process.

**Returns**

the process id

References `_syscall()`, `process_t::pid`, and `SYS_GETPID`.

Referenced by `syscall_test_thread()`.

**5.100.2.5 void kill ( int pid )**

Kills the process with the given pid.

**Parameters**

*pid* process id

References `_syscall()`, and `SYS_KILL`.

Referenced by `shell_cmd_kill()`.

**5.100.2.6 void \_log ( char \* msg )**

Writes a string to the kernel debug monitor.

Useful to dump strings that should not be displayed in the process's own vmonitor.

**Parameters**

*msg* the text to print

References `_syscall()`, and `SYS_LOG`.

Referenced by `syscall_test_thread()`, `threadA()`, `threadB()`, `threadC()`, and `threadD()`.

**5.100.2.7 void\* \_malloc ( size\_t size )**

Allocates new memory.

**See also**

[\\_free](#)

**Parameters**

*size* the number of bytes to allocate

**Returns**

a pointer to the new memory area or NULL if allocation failed

References `_syscall()`, `sc_malloc_args_t::mem`, `sc_malloc_args_t::size`, and `SYS_MALLOC`.

Referenced by `allocate_dummy_block()`, `dev_stdout_write()`, `do_tests()`, `init_bf()`, `interpret_bf()`, `new_-shell()`, `paralleleModellierung()`, `reset_bf()`, `resize_buf()`, `shell_cmd_clear()`, `shell_cmd_exec()`, `shell_cmd_snake()`, `snapshot()`, `speed()`, `syscall_test_thread()`, `threadA()`, and `threadB()`.

**5.100.2.8 int \_open ( char \* path, int oflag, int mode )**

Opens a file or a device.

**Parameters**

*path* the path of the file to open

*oflag* the open flag.

**See also**

[O\\_OPEN](#)

[O\\_CREAT](#)

**Parameters**

*mode* not used as of now. Set to 0.

**Returns**

a valid handle on success or -1 if failed

References `_syscall()`, `sc_open_args_t::fd`, `sc_open_args_t::mode`, `sc_open_args_t::oflag`, `sc_open_args_t::path`, and `SYS_OPEN`.

Referenced by `init_bf()`, `memview_main()`, `mv_show_stats()`, `mv_switch_to_graphicsmode()`, `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cd()`, `shell_cmd_cp()`, `shell_cmd_date()`, `shell_cmd_exec()`, `shell_cmd_ls()`, `shell_cmd_mkdir()`, `shell_cmd_pong()`, `shell_cmd_snapshot()`, `shell_cmd_speed()`, `shell_cmd_synth()`, `shell_cmd_touch()`, `shell_cmd_write()`, `shell_main()`, `snake()`, `snapshot()`, `speed()`, `synth()`, `syscall_test_thread()`, and `test_batch_files()`.

**5.100.2.9 int \_read ( int fd, void \* buf, int size )**

Reads data from a file or device.

**Parameters**

*fd* the file descriptor

*buf* the buffer to receive the data read

*size* the number of bytes to read

**Returns**

the number of bytes read or -1 on error

References `_syscall()`, `sc_read_write_args_t::buf`, `sc_read_write_args_t::fd`, `sc_read_write_args_t::rw_count`, `sc_read_write_args_t::size`, and `SYS_READ`.

Referenced by `_fgetch()`, `fgetch()`, `interpret_bf()`, `keydown()`, `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cp()`, `shell_cmd_date()`, `shell_cmd_exec()`, `shell_cmd_ls()`, `shell_cmd_pong()`, `shell_cmd_snapshot()`, `shell_cmd_synth()`, `snake()`, `speed()`, and `syscall_test_thread()`.

**5.100.2.10 `int _seek ( int fd, int offset, int whence )`**

Moves the file pointer in a file or device.

The file pointer determines the current read and write position inside a file.

**Parameters**

*fd* the file descriptor

*offset* the offset from the position specified by the "whence" argument

*whence* the starting position of the seek operation.

**See also**

[SEEK\\_SET](#)  
[SEEK\\_CUR](#)  
[SEEK\\_END](#)

**Returns**

the new position of the file pointer

References `_syscall()`, `sc_seek_args_t::fd`, `sc_seek_args_t::offset`, `sc_seek_args_t::pos`, `SYS_SEEK`, and `sc_seek_args_t::whence`.

Referenced by `shell_cmd_synth()`, `speed()`, and `syscall_test_thread()`.

**5.100.2.11 `int _stat ( char * path, stat * buf )`**

Returns extended file information for a single filename.

**Parameters**

*path* the filename

*buf* the stat structure to receive the results

**Returns**

zero on success, -1 on error

References `_syscall()`, `sc_stat_args_t::buf`, `sc_stat_args_t::path`, `sc_stat_args_t::success`, and `SYS_STAT`.

Referenced by `shell_cmd_ls()`.

**5.100.2.12** `int_unlink ( char * path )`

Deletes a filename and possible the file it refers to.

**Parameters**

*path* the filename to remove

**Returns**

zero on success, -1 on error

References `_syscall()`, `sc_unlink_args_t::path`, `sc_unlink_args_t::success`, and `SYS_UNLINK`.

Referenced by `shell_cmd_rm()`, and `speed()`.

**5.100.2.13** `int_write ( int fd, void * buf, int size )`

Writes data into a file or device.

**Parameters**

*fd* the file descriptor

*buf* the buffer containing the data to write

*size* the number of bytes to write

**Returns**

the number of bytes written or -1 on error

References `_syscall()`, `sc_read_write_args_t::buf`, `sc_read_write_args_t::fd`, `sc_read_write_args_t::rw_count`, `sc_read_write_args_t::size`, and `SYS_WRITE`.

Referenced by `_fputc()`, `_fputs()`, `_printf()`, `init_bf()`, `interpret_bf()`, `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cp()`, `shell_cmd_pong()`, `shell_cmd_write()`, `snake()`, `snapshot()`, `speed()`, `syscall_test_thread()`, `test_batch_files()`, and `update_view()`.

**5.101** `src/kernel/pm/syscalls_cli.h` File Reference

Header file for the "client" side syscall magic.

```
#include "../include/types.h"
#include "../fs/fs_types.h"
#include "syscalls_shared.h"
#include "../fs/fs_types.h"
```

**Functions**

- `void _log (char *msg)`

*Writes a string to the kernel debug monitor.*

- void `_exit` (int status)  
*Exits the calling process and releases its resources.*
- int `_getpid` ()  
*Returns the PID (process id) of the calling process.*
- int `_open` (char \*path, int oflag, int mode)  
*Opens a file or a device.*
- int `_close` (int fd)  
*Closes a open file handle.*
- int `_read` (int fd, void \*buf, int size)  
*Reads data from a file or device.*
- int `_write` (int fd, void \*buf, int size)  
*Writes data into a file or device.*
- int `_seek` (int fd, int offset, int whence)  
*Moves the file pointer in a file or device.*
- void \* `_malloc` (size\_t size)  
*Allocates new memory.*
- void `_free` (void \*block)  
*Frees memory previously allocated through `malloc()`.*
- int `_unlink` (char \*path)  
*Deletes a filename and possible the file it refers to.*
- int `_stat` (char \*path, `stat` \*buf)  
*Returns extended file information for a single filename.*
- void `_kill` (int pid)  
*Kills the process with the given pid.*
- void `_printf` (char \*fmt,...)  
*Prints formatted output to `STDOUT`.*
- char \* `_fgets` (char \*s, int n, int fd)  
*Reads a string from a file descriptor into a buffer.*
- int `_fputs` (char \*s, int fd)  
*Writes a string into a given file.*
- int `_fgetch` (int fd)  
*Waits until a character from the given file could be read and returns it.*
- int `_fputch` (char ch, int fd)  
*Writes a character to the given file.*

### 5.101.1 Detailed Description

Header file for the "client" side syscall magic.

**Author**

dbader

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

### 5.101.2 Function Documentation

#### 5.101.2.1 `int _close ( int fd )`

Closes a open file handle.

**Parameters**

*fd* the file descriptor of the file to close

**Returns**

0 on success, -1 on error

References `_syscall()`, `sc_close_args_t::fd`, `sc_close_args_t::success`, and `SYS_CLOSE`.

Referenced by `init_bf()`, `memview_main()`, `mv_switch_to_textmode()`, `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cd()`, `shell_cmd_cp()`, `shell_cmd_date()`, `shell_cmd_exec()`, `shell_cmd_ls()`, `shell_cmd_mkdir()`, `shell_cmd_pong()`, `shell_cmd_snapshot()`, `shell_cmd_speed()`, `shell_cmd_touch()`, `shell_cmd_write()`, `shell_main()`, `snake()`, `snapshot()`, `speed()`, `synth()`, `syscall_test_thread()`, and `test_batch_files()`.

#### 5.101.2.2 `void _exit ( int status )`

Exits the calling process and releases its resources.

**Parameters**

*status* the return status

References `_syscall()`, `SYS_EXIT`, and `WAIT_FOR_INTERRUPT`.

Referenced by `memview_main()`, `shell_cmd_exit()`, `shell_main()`, `snake()`, `snapshot()`, `speed()`, `synth()`, and `syscall_test_thread()`.



### 5.101.2.3 int \_fgetch ( int *fd* )

Waits until a character from the given file could be read and returns it.

#### Parameters

*fd* the file descriptor

#### Returns

the character that was read

References `_read()`, and `halt()`.

Referenced by `_fgets()`, `memview_main()`, `mv_show_stats()`, `shell_cmd_pong()`, `snake()`, and `speed()`.

### 5.101.2.4 char\* \_fgets ( char \* *s*, int *n*, int *fd* )

Reads a string from a file descriptor into a buffer.

#### Parameters

*s* the string buffer

*n* the maximum number of bytes to read (ie the buffer size)

*fd* the file descriptor

#### Returns

the string buffer

References `_fgetch()`, `_fputch()`, `start`, and `STDOUT`.

Referenced by `shell_main()`, and `snapshot()`.

### 5.101.2.5 int \_fputch ( char *ch*, int *fd* )

Writes a character to the given file.

#### Parameters

*ch* the character to write

*fd* the file descriptor

#### Returns

the number of bytes written, -1 on error

References `_write()`.

Referenced by `_fgets()`, and `shell_cmd_cat()`.

### 5.101.2.6 `int fputs ( char * s, int fd )`

Writes a string into a given file.

#### Parameters

*s* the string to write  
*fd* the file descriptor

#### Returns

the number of bytes written, -1 on error

References `_write()`, and `strlen`.

Referenced by `shell_autocomplete()`, and `shell_cmd_clear()`.

### 5.101.2.7 `void free ( void * block )`

Frees memory previously allocated through `malloc()`.

#### See also

[\\_malloc](#)

#### Parameters

*block* pointer to a memory block allocated through `malloc`

References `_syscall()`, and `SYS_FREE`.

Referenced by `dev_stdout_write()`, `free_all_dummy_blocks()`, `free_dummy_block()`, `interpret_bf()`, `new_shell()`, `reset_bf()`, `resize_buf()`, `shell_cmd_clear()`, `shell_cmd_exec()`, `shell_cmd_exit()`, `shell_cmd_snake()`, `shell_handle_command()`, `snapshot()`, `speed()`, `syscall_test_thread()`, `threadA()`, and `threadB()`.

### 5.101.2.8 `int getpid ( )`

Returns the PID (process id) of the calling process.

#### Returns

the process id

References `_syscall()`, `process_t::pid`, and `SYS_GETPID`.

Referenced by `syscall_test_thread()`.

### 5.101.2.9 `void kill ( int pid )`

Kills the process with the given pid.

#### Parameters

*pid* process id

References `_syscall()`, and `SYS_KILL`.

Referenced by `shell_cmd_kill()`.

#### 5.101.2.10 void \_log ( char \* *msg* )

Writes a string to the kernel debug monitor.

Useful to dump strings that should not be displayed in the process's own vmonitor.

##### Parameters

*msg* the text to print

References `_syscall()`, and `SYS_LOG`.

Referenced by `syscall_test_thread()`, `threadA()`, `threadB()`, `threadC()`, and `threadD()`.

#### 5.101.2.11 void\* \_malloc ( size\_t *size* )

Allocates new memory.

##### See also

[\\_free](#)

##### Parameters

*size* the number of bytes to allocate

##### Returns

a pointer to the new memory area or NULL if allocation failed

References `_syscall()`, `sc_malloc_args_t::mem`, `sc_malloc_args_t::size`, and `SYS_MALLOC`.

Referenced by `allocate_dummy_block()`, `dev_stdout_write()`, `do_tests()`, `init_bf()`, `interpret_bf()`, `new_shell()`, `paralleleModellierung()`, `reset_bf()`, `resize_buf()`, `shell_cmd_clear()`, `shell_cmd_exec()`, `shell_cmd_snake()`, `snapshot()`, `speed()`, `syscall_test_thread()`, `threadA()`, and `threadB()`.

#### 5.101.2.12 int \_open ( char \* *path*, int *oflag*, int *mode* )

Opens a file or a device.

##### Parameters

*path* the path of the file to open

*oflag* the open flag.

##### See also

[O\\_OPEN](#)  
[O\\_CREAT](#)

##### Parameters

*mode* not used as of now. Set to 0.

##### Returns

a valid handle on success or -1 if failed

References `_syscall()`, `sc_open_args_t::fd`, `sc_open_args_t::mode`, `sc_open_args_t::oflag`, `sc_open_args_t::path`, and `SYS_OPEN`.

Referenced by `init_bf()`, `memview_main()`, `mv_show_stats()`, `mv_switch_to_graphicsmode()`, `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cd()`, `shell_cmd_cp()`, `shell_cmd_date()`, `shell_cmd_exec()`, `shell_cmd_ls()`, `shell_cmd_mkdir()`, `shell_cmd_pong()`, `shell_cmd_snapshot()`, `shell_cmd_speed()`, `shell_cmd_synth()`, `shell_cmd_touch()`, `shell_cmd_write()`, `shell_main()`, `snake()`, `snapshot()`, `speed()`, `synth()`, `syscall_test_thread()`, and `test_batch_files()`.

### 5.101.2.13 `void _printf ( char * fmt, ... )`

Prints formatted output to STDOUT.

#### See also

[printf](#) This exists as a stub to ease the separation of the shell from the kernel code (as of now, the shell could simply call the kernel [printf](#)).

References `_write()`, `STDOUT`, `va_end`, `va_start`, and `vsnprintf`.

Referenced by `heap_mem_dump()`, `memview_main()`, `mv_do_benchmark()`, `mv_show_stats()`, `mv_switch_to_textmode()`, `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cd()`, `shell_cmd_cmdlist()`, `shell_cmd_cp()`, `shell_cmd_date()`, `shell_cmd_echo()`, `shell_cmd_exec()`, `shell_cmd_exit()`, `shell_cmd_kill()`, `shell_cmd_ls()`, `shell_cmd_mkdir()`, `shell_cmd_nice()`, `shell_cmd_pong()`, `shell_cmd_pwd()`, `shell_cmd_rm()`, `shell_cmd_snapshot()`, `shell_cmd_speed()`, `shell_cmd_synth()`, `shell_cmd_test()`, `shell_cmd_touch()`, `shell_cmd_write()`, `shell_handle_command()`, `shell_main()`, `snake()`, `snapshot()`, `speed()`, `threadA()`, `threadB()`, `threadConsumer()`, `threadFabrik()`, `threadGeschaeft1()`, `threadGeschaeft2()`, `threadLastwagen1()`, `threadLastwagen2()`, `threadMitarbeiter()`, and `threadProducer()`.

### 5.101.2.14 `int _read ( int fd, void * buf, int size )`

Reads data from a file or device.

#### Parameters

- fd* the file descriptor
- buf* the buffer to receive the data read
- size* the number of bytes to read

#### Returns

the number of bytes read or -1 on error

References `_syscall()`, `sc_read_write_args_t::buf`, `sc_read_write_args_t::fd`, `sc_read_write_args_t::rw_count`, `sc_read_write_args_t::size`, and `SYS_READ`.

Referenced by `_fgetch()`, `fgetch()`, `interpret_bf()`, `keydown()`, `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cp()`, `shell_cmd_date()`, `shell_cmd_exec()`, `shell_cmd_ls()`, `shell_cmd_pong()`, `shell_cmd_snapshot()`, `shell_cmd_synth()`, `snake()`, `speed()`, and `syscall_test_thread()`.

### 5.101.2.15 `int _seek ( int fd, int offset, int whence )`

Moves the file pointer in a file or device.

The file pointer determines the current read and write position inside a file.

**Parameters**

*fd* the file descriptor  
*offset* the offset from the position specified by the "whence" argument  
*whence* the starting position of the seek operation.

**See also**

[SEEK\\_SET](#)  
[SEEK\\_CUR](#)  
[SEEK\\_END](#)

**Returns**

the new position of the file pointer

References `_syscall()`, `sc_seek_args_t::fd`, `sc_seek_args_t::offset`, `sc_seek_args_t::pos`, `SYS_SEEK`, and `sc_seek_args_t::whence`.

Referenced by `shell_cmd_synth()`, `speed()`, and `syscall_test_thread()`.

**5.101.2.16 int \_stat ( char \* path, stat \* buf )**

Returns extended file information for a single filename.

**Parameters**

*path* the filename  
*buf* the stat structure to receive the results

**Returns**

zero on success, -1 on error

References `_syscall()`, `sc_stat_args_t::buf`, `sc_stat_args_t::path`, `sc_stat_args_t::success`, and `SYS_STAT`.

Referenced by `shell_cmd_ls()`.

**5.101.2.17 int \_unlink ( char \* path )**

Deletes a filename and possible the file it refers to.

**Parameters**

*path* the filename to remove

**Returns**

zero on success, -1 on error

References `_syscall()`, `sc_unlink_args_t::path`, `sc_unlink_args_t::success`, and `SYS_UNLINK`.

Referenced by `shell_cmd_rm()`, and `speed()`.

### 5.101.2.18 `int _write ( int fd, void * buf, int size )`

Writes data into a file or device.

#### Parameters

- fd* the file descriptor
- buf* the buffer containing the data to write
- size* the number of bytes to write

#### Returns

the number of bytes written or -1 on error

References `_syscall()`, `sc_read_write_args_t::buf`, `sc_read_write_args_t::fd`, `sc_read_write_args_t::rw_count`, `sc_read_write_args_t::size`, and `SYS_WRITE`.

Referenced by `_fputc()`, `_fputs()`, `_printf()`, `init_bf()`, `interpret_bf()`, `shell_cmd_bf()`, `shell_cmd_cat()`, `shell_cmd_cp()`, `shell_cmd_pong()`, `shell_cmd_write()`, `snake()`, `snapshot()`, `speed()`, `syscall_test_thread()`, `test_batch_files()`, and `update_view()`.

## 5.102 `src/kernel/pm/syscalls_shared.h` File Reference

This holds definitions and things which need to be shared between the kernel side of the syscall subsystem and the client side.

```
#include "../fs/fs_types.h"
#include "../io/io_rtc.h"
```

### Data Structures

- struct `sc_open_args_t`  
*Arguments for the OPEN syscall.*
- struct `sc_close_args_t`  
*Arguments for the CLOSE syscall.*
- struct `sc_read_write_args_t`  
*Arguments for the READ and WRITE syscalls.*
- struct `sc_seek_args_t`  
*Arguments for the SEEK syscall.*
- struct `sc_malloc_args_t`  
*Arguments for the MALLOC syscall.*
- struct `sc_unlink_args_t`  
*Arguments for the UNLINK syscall.*
- struct `sc_stat_args_t`  
*Arguments for the STAT syscall.*

## Defines

- #define [WAIT\\_FOR\\_INTERRUPT\(\)](#) `__asm__("hlt");`  
*Stops execution until the next interrupt occurs.*
  
- #define [SYS\\_LOG](#) 0
- #define [SYS\\_EXIT](#) 1
- #define [SYS\\_GETPID](#) 2
- #define [SYS\\_OPEN](#) 3
- #define [SYS\\_CLOSE](#) 4
- #define [SYS\\_READ](#) 5
- #define [SYS\\_WRITE](#) 6
- #define [SYS\\_SEEK](#) 7
- #define [SYS\\_MALLOC](#) 8
- #define [SYS\\_FREE](#) 9
- #define [SYS\\_UNLINK](#) 10
- #define [SYS\\_STAT](#) 11
- #define [SYS\\_KILL](#) 12
- #define [MAX\\_SYSCALL](#) 12  
*The highest syscall id that is still valid.*
  
- #define [O\\_OPEN](#) 0  
*Attempts to open a file.*
  
- #define [O\\_CREAT](#) 1  
*Attempts to open a file.*
  
- #define [SEEK\\_SET](#) 0  
*Seek to an absolute position.*
  
- #define [SEEK\\_CUR](#) 1  
*Seek relative from the current position of the file pointer.*
  
- #define [SEEK\\_END](#) 2  
*Seek relative to the end of the file.*

## Typedefs

- typedef struct [sc\\_open\\_args\\_t](#) [sc\\_open\\_args\\_t](#)  
*Arguments for the OPEN syscall.*
  
- typedef struct [sc\\_close\\_args\\_t](#) [sc\\_close\\_args\\_t](#)  
*Arguments for the CLOSE syscall.*
  
- typedef struct [sc\\_read\\_write\\_args\\_t](#) [sc\\_read\\_write\\_args\\_t](#)  
*Arguments for the READ and WRITE syscalls.*
  
- typedef struct [sc\\_seek\\_args\\_t](#) [sc\\_seek\\_args\\_t](#)  
*Arguments for the SEEK syscall.*

- typedef struct [sc\\_malloc\\_args\\_t](#) [sc\\_malloc\\_args\\_t](#)  
*Arguments for the MALLOC syscall.*
- typedef struct [sc\\_unlink\\_args\\_t](#) [sc\\_unlink\\_args\\_t](#)  
*Arguments for the UNLINK syscall.*
- typedef [file\\_info\\_t](#) [stat](#)  
*Stat structure returned by the STAT syscall.*
- typedef struct [sc\\_stat\\_args\\_t](#) [sc\\_stat\\_args\\_t](#)  
*Arguments for the STAT syscall.*

### 5.102.1 Detailed Description

This holds definitions and things which need to be shared between the kernel side of the syscall subsystem and the client side.

**Author**

dbader

**LastChangedBy:**

dtraytel

**Version****Rev:**

12

### 5.102.2 Define Documentation

#### 5.102.2.1 #define MAX\_SYSCALL 12

The highest syscall id that is still valid.

Be sure to update this!

Referenced by `pm_init()`, and `pm_syscall()`.

#### 5.102.2.2 #define O\_CREAT 1

Attempts to open a file.

If the file does not exist yet `open()` creates the file and opens it.

Referenced by `init_bf()`, `shell_cmd_cp()`, `shell_cmd_mkdir()`, `shell_cmd_speed()`, `shell_cmd_touch()`, `snapshot()`, `speed()`, `sys_open()`, and `test_batch_files()`.



**5.102.2.3 #define O\_OPEN 0**

Attempts to open a file.

If the file does not exist, `open()` aborts and returns -1.

**5.102.2.4 #define SEEK\_CUR 1**

Seek relative from the current position of the file pointer.

Referenced by `sys_seek()`, and `syscall_test_thread()`.

**5.102.2.5 #define SEEK\_END 2**

Seek relative to the end of the file.

Referenced by `main()`, `shell_cmd_synth()`, `speed()`, and `sys_seek()`.

**5.102.2.6 #define SEEK\_SET 0**

Seek to an absolute position.

Referenced by `main()`, `shell_cmd_synth()`, `speed()`, and `sys_seek()`.

**5.102.2.7 #define SYS\_CLOSE 4**

Referenced by `_close()`.

**5.102.2.8 #define SYS\_EXIT 1**

Referenced by `_exit()`.

**5.102.2.9 #define SYS\_FREE 9**

Referenced by `_free()`.

**5.102.2.10 #define SYS\_GETPID 2**

Referenced by `_getpid()`.

**5.102.2.11 #define SYS\_KILL 12**

Referenced by `_kill()`.

**5.102.2.12 #define SYS\_LOG 0**

Referenced by `_log()`.

**5.102.2.13 #define SYS\_MALLOC 8**

Referenced by `_malloc()`.

**5.102.2.14 #define SYS\_OPEN 3**

Referenced by `_open()`.

**5.102.2.15 #define SYS\_READ 5**

Referenced by `_read()`.

**5.102.2.16 #define SYS\_SEEK 7**

Referenced by `_seek()`.

**5.102.2.17 #define SYS\_STAT 11**

Referenced by `_stat()`.

**5.102.2.18 #define SYS\_UNLINK 10**

Referenced by `_unlink()`.

**5.102.2.19 #define SYS\_WRITE 6**

Referenced by `_write()`.

**5.102.2.20 #define WAIT\_FOR\_INTERRUPT( ) \_\_asm\_\_("hlt");**

Stops execution until the next interrupt occurs.

Referenced by `_exit()`.

**5.102.3 Typedef Documentation****5.102.3.1 typedef struct sc\_close\_args\_t sc\_close\_args\_t**

Arguments for the CLOSE syscall.

**5.102.3.2 typedef struct sc\_malloc\_args\_t sc\_malloc\_args\_t**

Arguments for the MALLOC syscall.

**5.102.3.3 typedef struct sc\_open\_args\_t sc\_open\_args\_t**

Arguments for the OPEN syscall.

#### 5.102.3.4 typedef struct sc\_read\_write\_args\_t sc\_read\_write\_args\_t

Arguments for the READ and WRITE syscalls.

#### 5.102.3.5 typedef struct sc\_seek\_args\_t sc\_seek\_args\_t

Arguments for the SEEK syscall.

#### 5.102.3.6 typedef struct sc\_stat\_args\_t sc\_stat\_args\_t

Arguments for the STAT syscall.

#### 5.102.3.7 typedef struct sc\_unlink\_args\_t sc\_unlink\_args\_t

Arguments for the UNLINK syscall.

#### 5.102.3.8 typedef file\_info\_t stat

Stat structure returned by the STAT syscall.

See also

[file\\_info\\_t](#)

## 5.103 src/tools/bin2c/bin2c.c File Reference

Reads a file and dumps its contents into a C byte array.

```
#include <stdio.h>
#include <stdlib.h>
```

### Defines

- #define [VARHDR](#) `"/* %s, %d bytes */\nunsigned char data[ ] = {\n"`
- #define [VARFMT](#) `"0x%.2x"`
- #define [VAREND](#) `"\n}\n"`
- #define [VARTAB](#) `" "`
- #define [ITEMSPERLINE](#) `15`

### Functions

- [int main](#) (int argc, char \*\*argv)

### 5.103.1 Detailed Description

Reads a file and dumps its contents into a C byte array. Example:

- contents of input.bin: "Hello, World. This is a test."
- then run "bin2c input.bin > test.c"
- you get:

```

        unsigned char data[] = {
        0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x2c, 0x20, 0x57, 0x
6f,
        0x72, 0x6c, 0x64, 0x21, 0x20, 0x54, 0x68, 0x69, 0x
73,
        0x20, 0x69, 0x73, 0x20, 0x61, 0x20, 0x74, 0x65, 0x
73,
        0x74, 0x2e, 0x0a
        }

```

Build via "gcc bin2c.c -o bin2c" (the makefile also has a target called bin2c)

#### Author

dbader

#### LastChangedBy:

dtraytel

#### Version

#### Rev:

12

### 5.103.2 Define Documentation

#### 5.103.2.1 #define ITEMSPERLINE 15

Referenced by main().

#### 5.103.2.2 #define VAREND "\n}\n"

Referenced by main().

#### 5.103.2.3 #define VARFMT "0x%.2x"

Referenced by main().

#### 5.103.2.4 #define VARHDR "/\* %s, %d bytes \*/\nunsigned char data[] = {\n"

Referenced by main().

### 5.103.2.5 #define VARTAB " "

Referenced by main().

## 5.103.3 Function Documentation

### 5.103.3.1 int main ( int argc, char \*\* argv )

References free, ITEMSPERLINE, malloc(), printf, SEEK\_END, SEEK\_SET, size, VAREND, VARFMT, VARHDR, and VARTAB.

## 5.104 src/tools/chips/chips.c File Reference

```
#include "chips.h"
#include <errno.h>
#include <fcntl.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
```

### Data Structures

- struct [command\\_t](#)

### Functions

- int [chips\\_usage](#) (char const \*progrname)
- int [main](#) (int argc, char \*\*argv)

## 5.104.1 Function Documentation

### 5.104.1.1 int chips\_usage ( char const \* progrname )

References name.

Referenced by main().

### 5.104.1.2 int main ( int argc, char \*\* argv )

References chips\_usage(), command\_t::exec, fs\_init(), name, NULL, potatoes\_disc\_create(), potatoes\_disc\_destroy(), potatoes\_set\_current\_disk(), and strcmp.

## 5.105 src/tools/chips/chips.h File Reference

### Data Structures

- struct [potatoes\\_time](#)
- struct [potatoes\\_dir\\_entry](#)
- struct [potatoes\\_file\\_info](#)
- struct [PotatoesDisk](#)

### Defines

- #define [POTATOES\\_DATA\\_FILE](#) 1
- #define [POTATOES\\_DIRECTORY](#) 2
- #define [POTATOES\\_NAME\\_SIZE](#) 28
- #define [POTATOES\\_BLOCK\\_SIZE](#) 512
- #define [POTATOES\\_DIR\\_ENTRY\\_SIZE](#) sizeof (struct [potatoes\\_dir\\_entry](#))
- #define [POTATOES\\_DIR\\_ENTRIES\\_PER\\_BLOCK](#) ((POTATOES\_BLOCK\_SIZE)/(POTATOES\_DIR\_ENTRY\_SIZE))

### Typedefs

- typedef unsigned char [potatoes\\_uint8](#)
- typedef signed char [potatoes\\_sint8](#)
- typedef unsigned short [potatoes\\_uint16](#)
- typedef signed short [potatoes\\_sint16](#)
- typedef unsigned int [potatoes\\_uint32](#)
- typedef signed int [potatoes\\_sint32](#)
- typedef float [potatoes\\_float32](#)
- typedef double [potatoes\\_float64](#)
- typedef [potatoes\\_uint8](#) [potatoes\\_bool](#)
- typedef [potatoes\\_uint32](#) [potatoes\\_size\\_t](#)
- typedef [potatoes\\_uint32](#) [potatoes\\_block\\_nr](#)
- typedef [potatoes\\_sint16](#) [potatoes\\_inode\\_nr](#)
- typedef [potatoes\\_sint16](#) [potatoes\\_file\\_nr](#)
- typedef struct [potatoes\\_time](#) [potatoes\\_time\\_t](#)
- typedef struct [potatoes\\_file\\_info](#) [potatoes\\_file\\_info\\_t](#)

### Functions

- struct [potatoes\\_dir\\_entry](#) [\\_\\_attribute\\_\\_\(\(packed\)\) potatoes\\_dir\\_entry](#)
- [PotatoesDisk](#) \* [potatoes\\_disc\\_create](#) (char const \*file)
- void [potatoes\\_disc\\_destroy](#) ([PotatoesDisk](#) \*disk)
- void [potatoes\\_set\\_current\\_disk](#) ([PotatoesDisk](#) \*disk)
- void [dump\\_consts](#) ()

*Prints out all important constants concerning the file system.*

- void [fs\\_init](#) ()

*Initializes the file system.*

- void `fs_shutdown` ()

*Shuts the file system down and writes all important information to HD.*

- `potatoes_file_nr do_open` (char \*abs\_path)
- int `do_close` (potatoes\_file\_nr fd)
- `potatoes_size_t do_read` (potatoes\_file\_nr fd, void \*buf, potatoes\_size\_t count, potatoes\_uint32 pos)
- `potatoes_size_t do_write` (potatoes\_file\_nr fd, void \*buf, potatoes\_size\_t count, potatoes\_uint32 pos)
- `potatoes_file_info_t * get_file_info` (potatoes\_file\_nr fd, potatoes\_file\_info\_t \*info)
- `potatoes_bool fs_truncate` (char \*abs\_path, potatoes\_uint32 size)
- `potatoes_bool fs_create` (char \*abs\_path, int data\_type)

*Creates a file from absolute path by inserting the name into the containing directory, creating a new inode and writing it to HD.*

- `potatoes_bool fs_delete` (char \*abs\_path)

*Deletes a file by removing it from the containing directory.*

## Variables

- `potatoes_block_nr inode`
- char `name` [POTATOES\_NAME\_SIZE]

### 5.105.1 Define Documentation

#### 5.105.1.1 #define POTATOES\_BLOCK\_SIZE 512

#### 5.105.1.2 #define POTATOES\_DATA\_FILE 1

Referenced by `chips_create()`, and `chips_unlink()`.

#### 5.105.1.3 #define POTATOES\_DIR\_ENTRIES\_PER\_BLOCK ((POTATOES\_BLOCK\_SIZE)/(POTATOES\_DIR\_ENTRY\_SIZE))

#### 5.105.1.4 #define POTATOES\_DIR\_ENTRY\_SIZE sizeof (struct potatoes\_dir\_entry)

#### 5.105.1.5 #define POTATOES\_DIRECTORY 2

Referenced by `chips_mkdir()`.

5.105.1.6 `#define POTATOES_NAME_SIZE 28`

## 5.105.2 Typedef Documentation

5.105.2.1 `typedef potatoes_uint32 potatoes_block_nr`

5.105.2.2 `typedef potatoes_uint8 potatoes_bool`

5.105.2.3 `typedef struct potatoes_file_info potatoes_file_info_t`

5.105.2.4 `typedef potatoes_sint16 potatoes_file_nr`

5.105.2.5 `typedef potatoes_sint16 potatoes_inode_nr`

5.105.2.6 `typedef signed short potatoes_sint16`

5.105.2.7 `typedef signed int potatoes_sint32`

5.105.2.8 `typedef signed char potatoes_sint8`

5.105.2.9 `typedef potatoes_uint32 potatoes_size_t`

5.105.2.10 `typedef struct potatoes_time potatoes_time_t`

5.105.2.11 `typedef unsigned short potatoes_uint16`

5.105.2.12 `typedef unsigned int potatoes_uint32`

5.105.2.13 `typedef unsigned char potatoes_uint8`

5.105.2.14 `typedef float potatpes_float32`

5.105.2.15 `typedef double potatpes_float64`

## 5.105.3 Function Documentation

5.105.3.1 `struct potatoes_dir_entry __attribute__((packed))`

5.105.3.2 `int do_close ( potatoes_file_nr fd )`

5.105.3.3 `potatoes_file_nr do_open ( char * abs_path )`

References `fs_open()`.

Referenced by `__ls()`, `chips_unlink()`, `sys_open()`, `sys_stat()`, `sys_unlink()`, `test_close()`, `test_ls()`, and `test_-PM()`.



**5.105.3.4** potatoes\_size\_t do\_read ( potatoes\_file\_nr *fd*, void \* *buf*, potatoes\_size\_t *count*, potatoes\_uint32 *pos* )

**5.105.3.5** potatoes\_size\_t do\_write ( potatoes\_file\_nr *fd*, void \* *buf*, potatoes\_size\_t *count*, potatoes\_uint32 *pos* )

**5.105.3.6** void dump\_consts ( )

Prints out all important constants concerning the file system.

For debug purposes only.

References ADDR\_PER\_BLOCK, BYTES\_DIRECT, BYTES\_DOUBLE\_INDIRECT, BYTES\_SINGLE\_INDIRECT, DIR\_ENTRIES\_PER\_BLOCK, DISK\_INODE\_SIZE, INODES\_PER\_BLOCK, MEM\_INODE\_SIZE, NUM\_FILES, NUM\_INODES, NUM\_PROC\_FILES, printf, ROOT\_INODE\_BLOCK, and SUPER\_SIZE.

**5.105.3.7** potatoes\_bool fs\_create ( char \* *abs\_path*, int *data\_type* )

Creates a file from absolute path by inserting the name into the containing directory, creating a new inode and writing it to HD.

#### Parameters

*abs\_path* absolute file path

*data\_type* DATA\_FILE | DIRECTORY (

#### See also

[fs\\_const.h](#))

#### Returns

result status of the create operation

References CREATE, and fs\_create\_delete().

Referenced by chips\_create(), chips\_mkdir(), do\_create(), do\_mkdir(), test\_create(), test\_delete(), test\_open\_close(), test\_rw\_qualitative(), and test\_rw\_quantitative().

**5.105.3.8** potatoes\_bool fs\_delete ( char \* *abs\_path* )

Deletes a file by removing it from the containing directory.

#### Parameters

*abs\_path* absolute file path

#### Returns

result status of the delete operation

References DELETE, fs\_create\_delete(), and NULL.

Referenced by chips\_unlink(), do\_remove(), and test\_delete().

### 5.105.3.9 void fs\_init ( )

Initializes the file system.

References bmap, create\_fs(), dprint\_separator, dprintf, free, load\_fs(), panic, and printf.

### 5.105.3.10 void fs\_shutdown ( )

Shuts the file system down and writes all important information to HD.

References bmap, free, fs\_close(), gft, printf, write\_bmap(), write\_root(), and write\_super\_block().

### 5.105.3.11 potatoes\_bool fs\_truncate ( char \* *abs\_path*, potatoes\_uint32 *size* )

### 5.105.3.12 potatoes\_file\_info\_t\* get\_file\_info ( potatoes\_file\_nr *fd*, potatoes\_file\_info\_t \* *info* )

### 5.105.3.13 PotatoesDisk\* potatoes\_disc\_create ( char const \* *file* )

References PotatoesDisk::data, free, malloc(), NULL, and PotatoesDisk::size.

Referenced by main().

### 5.105.3.14 void potatoes\_disc\_destroy ( PotatoesDisk \* *disk* )

References PotatoesDisk::data, free, NULL, and PotatoesDisk::size.

Referenced by main().

### 5.105.3.15 void potatoes\_set\_current\_disk ( PotatoesDisk \* *disk* )

Referenced by main().

## 5.105.4 Variable Documentation

### 5.105.4.1 potatoes\_block\_nr inode

### 5.105.4.2 char name[POTATOES\_NAME\_SIZE]

## 5.106 src/tools/chips/chipsfs.c File Reference

```
#include "chips.h"
#include <fuse.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdarg.h>
```

```
#include <sys/types.h>
#include <sys/stat.h>
```

## Defines

- #define [FUSE\\_USE\\_VERSION](#) 26

## Functions

- int [chips\\_create](#) (const char \*path, mode\_t mode, struct fuse\_file\_info \*fi)
- int [chips\\_mkdir](#) (const char \*path, mode\_t mode)
- int [chips\\_unlink](#) (const char \*path)
- int [main](#) (int argc, char \*\*argv)

### 5.106.1 Define Documentation

#### 5.106.1.1 #define FUSE\_USE\_VERSION 26

### 5.106.2 Function Documentation

#### 5.106.2.1 int chips\_create ( const char \* *path*, mode\_t *mode*, struct fuse\_file\_info \* *fi* )

References [fs\\_create\(\)](#), and [POTATOES\\_DATA\\_FILE](#).

#### 5.106.2.2 int chips\_mkdir ( const char \* *path*, mode\_t *mode* )

References [fs\\_create\(\)](#), and [POTATOES\\_DIRECTORY](#).

#### 5.106.2.3 int chips\_unlink ( const char \* *path* )

References [do\\_close\(\)](#), [do\\_open\(\)](#), [fs\\_delete\(\)](#), [get\\_file\\_info\(\)](#), [potatoes\\_file\\_info::mode](#), and [POTATOES\\_DATA\\_FILE](#).

#### 5.106.2.4 int main ( int *argc*, char \*\* *argv* )

References [fs\\_init\(\)](#), [NULL](#), [potatoes\\_disc\\_create\(\)](#), and [potatoes\\_set\\_current\\_disk\(\)](#).

## 5.107 src/tools/chips/fs\_wrappers.c File Reference

```
#include "chips.h"
#include <errno.h>
#include <fcntl.h>
#include <stdarg.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
```

## Functions

- void [potatoes\\_bzero](#) (void \*dest, [potatoes\\_uint32](#) count)
- void \* [potatoes\\_malloc](#) ([potatoes\\_uint32](#) size)
- void \* [potatoes\\_mallocn](#) ([potatoes\\_uint32](#) size, char \*name)
- void [potatoes\\_free](#) (void \*start)
- void [potatoes\\_mem\\_dump](#) ()
- char \* [potatoes\\_strncpy](#) (char \*dest, char \*src, [size\\_t](#) n)
- [potatoes\\_uint32](#) [potatoes\\_strlen](#) (char \*str)
- char \* [potatoes\\_strdup](#) (char \*str)
- char \* [potatoes\\_strsep](#) (char \*\*str\_ptr, char \*delims)
- [potatoes\\_sint32](#) [potatoes\\_strcmp](#) (char \*s1, char \*s2)
- void \* [potatoes\\_memcpy](#) (void \*dest, void \*src, [potatoes\\_size\\_t](#) count)
- void [potatoes\\_printf](#) (char \*fmt,...)
- char \* [potatoes\\_strncat](#) (char \*s1, char \*s2, [potatoes\\_size\\_t](#) n)
- char \* [potatoes\\_strcat](#) (char \*s1, char \*s2)
- void [potatoes\\_panic](#) (char \*msg)
- char \* [potatoes\\_itoa](#) (int n, char \*str, unsigned int base)
- char \* [potatoes\\_time2str](#) ([potatoes\\_time\\_t](#) timestamp, char buf[24])
- [potatoes\\_uint32](#) [potatoes\\_get\\_hdsiz](#) ()
- void [potatoes\\_hd\\_write\\_sector](#) ([potatoes\\_uint32](#) dest, void \*src)
- void [potatoes\\_hd\\_read\\_sector](#) (void \*dest, [potatoes\\_uint32](#) src)
- [PotatoesDisk](#) \* [potatoes\\_disc\\_create](#) (char const \*file)
- void [potatoes\\_disc\\_destroy](#) ([PotatoesDisk](#) \*disk)
- void [potatoes\\_set\\_current\\_disk](#) ([PotatoesDisk](#) \*disk)

## Variables

- [PotatoesDisk](#) \* [potatoes\\_current\\_disc](#) = NULL

### 5.107.1 Function Documentation

#### 5.107.1.1 void potatoes\_bzero ( void \* dest, potatoes\_uint32 count )

References [bzero](#).

**5.107.1.2 PotatoesDisk\* potatoes\_disc\_create ( char const \* *file* )**

References PotatoesDisk::data, free, malloc(), NULL, and PotatoesDisk::size.

Referenced by main().

**5.107.1.3 void potatoes\_disc\_destroy ( PotatoesDisk \* *disk* )**

References PotatoesDisk::data, free, NULL, and PotatoesDisk::size.

Referenced by main().

**5.107.1.4 void potatoes\_free ( void \* *start* )**

References free.

**5.107.1.5 potatoes\_uint32 potatoes\_get\_hdsz ( )**

References NULL, and PotatoesDisk::size.

**5.107.1.6 void potatoes\_hd\_read\_sector ( void \* *dest*, potatoes\_uint32 *src* )**

References PotatoesDisk::data, memcpy, NULL, and PotatoesDisk::size.

**5.107.1.7 void potatoes\_hd\_write\_sector ( potatoes\_uint32 *dest*, void \* *src* )**

References PotatoesDisk::data, memcpy, NULL, and PotatoesDisk::size.

**5.107.1.8 char\* potatoes\_itoa ( int *n*, char \* *str*, unsigned int *base* )****5.107.1.9 void\* potatoes\_malloc ( potatoes\_uint32 *size* )**

References malloc().

**5.107.1.10 void\* potatoes\_mallocn ( potatoes\_uint32 *size*, char \* *name* )**

References malloc().

**5.107.1.11 void potatoes\_mem\_dump ( )****5.107.1.12 void\* potatoes\_memcpy ( void \* *dest*, void \* *src*, potatoes\_size\_t *count* )**

References memcpy.

**5.107.1.13 void potatoes\_panic ( char \* *msg* )****5.107.1.14 void potatoes\_printf ( char \* *fmt*, ... )**

References va\_end, and va\_start.

**5.107.1.15 void potatoes\_set\_current\_disk ( PotatoesDisk \* *disk* )**

Referenced by main().

**5.107.1.16 char\* potatoes\_strcat ( char \* *s1*, char \* *s2* )**

References strcat.

**5.107.1.17 potatoes\_uint32 potatoes\_strcmp ( char \* *s1*, char \* *s2* )**

References strcmp.

**5.107.1.18 char\* potatoes\_strdup ( char \* *str* )**

References strdup.

**5.107.1.19 potatoes\_uint32 potatoes\_strlen ( char \* *str* )**

References strlen.

**5.107.1.20 char\* potatoes\_strncat ( char \* *s1*, char \* *s2*, potatoes\_size\_t *n* )**

References strncat().

**5.107.1.21 char\* potatoes\_strncpy ( char \* *dest*, char \* *src*, size\_t *n* )**

References strncpy.

**5.107.1.22 char\* potatoes\_strsep ( char \*\* *str\_ptr*, char \* *delims* )**

References strsep.

**5.107.1.23 char\* potatoes\_time2str ( potatoes\_time\_t *timestamp*, char *buf*[24] )**

References strcpy().

**5.107.2 Variable Documentation****5.107.2.1 PotatoesDisk\* potatoes\_current\_disc = NULL****5.108 src/tools/chips/fs\_wrappers.h File Reference****Defines**

- #define [bzero](#) potatoes\_bzero
- #define [cputs](#) potatoes\_cputs

- #define `free` potatoes\_free
- #define `get_hdsiz` potatoes\_get\_hdsiz
- #define `hd_read_sector` potatoes\_hd\_read\_sector
- #define `hd_write_sector` potatoes\_hd\_write\_sector
- #define `itoa` potatoes\_itoa
- #define `mallocn` potatoes\_mallocn
- #define `mem_dump` potatoes\_mem\_dump
- #define `memcpy` potatoes\_memcpy
- #define `panic` potatoes\_panic
- #define `printf` potatoes\_printf
- #define `printf` potatoes\_printf
- #define `putchar` potatoes\_putchar
- #define `puts` potatoes\_puts
- #define `snprintf` potatoes\_snprintf
- #define `strcat` potatoes\_strcat
- #define `strcmp` potatoes\_strcmp
- #define `strdup` potatoes\_strdup
- #define `strlen` potatoes\_strlen
- #define `strncpy` potatoes\_strncpy
- #define `strsep` potatoes\_strsep
- #define `time2str` potatoes\_time2str
- #define `vsprintf` potatoes\_vsprintf

## 5.108.1 Define Documentation

### 5.108.1.1 #define `bzero` potatoes\_bzero

Referenced by `__ls()`, `callocn()`, `clear_block()`, `clear_buffer()`, `clear_cache()`, `delete_entry()`, `get_filename()`, `get_path()`, `malloc_bmap()`, `mm_init()`, `new_minode()`, `potatoes_bzero()`, `read_minode()`, `resize_buf()`, `shell_cmd_exec()`, `shell_cmd_ls()`, `speed()`, `strings_test()`, `sys_stat()`, `test_create()`, `test_ls()`, `test_PM()`, `test_rw_qualitative()`, and `test_rw_quantitative()`.

### 5.108.1.2 #define `cputs` potatoes\_cputs

### 5.108.1.3 #define `free` potatoes\_free

Referenced by `delete_file_from_dir()`, `dev_clock_read()`, `free_file()`, `free_virt_monitor()`, `fs_create_delete()`, `fs_init()`, `fs_shutdown()`, `fs_write()`, `hd_stressread_test()`, `hd_stresswrite_test()`, `hd_write_test()`, `insert_file()`, `insert_file_into_dir()`, `main()`, `malloc_test()`, `memmove()`, `pm_destroy_thread()`, `potatoes_disc_create()`, `potatoes_disc_destroy()`, `potatoes_free()`, `print_time()`, `realloc()`, `rf_free()`, `search_file()`, `shell_cmd_synth()`, `strsep_test()`, `sys_free()`, and `sys_open()`.

### 5.108.1.4 #define `get_hdsiz` potatoes\_get\_hdsiz

Referenced by `dump_bmap()`, `dump_hd1()`, `get_free_block()`, `hd_read_sector()`, `hd_stressread_test()`, `hd_stresswrite_test()`, `hd_write_sector()`, `init_super_block()`, and `malloc_bmap()`.

### 5.108.1.5 #define `hd_read_sector` potatoes\_hd\_read\_sector

Referenced by `cache_block()`, `hd_read_sector()`, `hd_stressread_test()`, and `hd_test()`.

**5.108.1.6 #define hd\_write\_sector potatoes\_hd\_write\_sector**

Referenced by `hd_stresswrite_test()`, `hd_test()`, `hd_write_test()`, and `wrt_block()`.

**5.108.1.7 #define itoa potatoes\_itoa**

Referenced by `new_shell()`, `test_create()`, `time2str()`, and `vsnprintf()`.

**5.108.1.8 #define mallocn potatoes\_mallocn**

Referenced by `callocn()`, `delete_file_from_dir()`, `dev_clock_read()`, `fs_write()`, `get_filename()`, `get_path()`, `hd_stressread_test()`, `hd_stresswrite_test()`, `hd_write_test()`, `insert_file_into_dir()`, `malloc()`, `malloc_bmap()`, `malloc_test()`, `memmove()`, `new_minode()`, `new_virt_monitor()`, `pm_create_thread()`, `pm_init()`, `print_time()`, `realloc()`, `rf_alloc()`, `strdup()`, `sys_malloc()`, `test_open_close()`, and `write_root()`.

**5.108.1.9 #define mem\_dump potatoes\_mem\_dump**

Referenced by `malloc_test()`, `shell_cmd_memdump()`, and `test_rw_quantitative()`.

**5.108.1.10 #define memcpy potatoes\_memcpy**

Referenced by `create_entry()`, `create_root()`, `dev_keyboard_read()`, `dev_stdout_write()`, `fs_read()`, `fs_write()`, `get_active_virt_monitor_name()`, `get_color_tag()`, `get_filename()`, `get_path()`, `interpret_bf()`, `make_snapshot()`, `memmove()`, `potatoes_hd_read_sector()`, `potatoes_hd_write_sector()`, `potatoes_memcpy()`, `rd_block()`, `resize_buf()`, `rf_copy()`, `start_vmonitor()`, `sys_stat()`, `update_virt_monitor()`, and `wrt_block()`.

**5.108.1.11 #define panic potatoes\_panic**

Referenced by `alloc_frame()`, `fs_init()`, `hd_handler()`, `hd_init()`, `hd_read_sector()`, `hd_write_sector()`, `isr_handler()`, `pm_create_thread()`, `pm_syscall()`, and `sys_exit()`.

**5.108.1.12 #define printf potatoes\_printf**

Referenced by `bitset_test()`, `create_fs()`, `delete_file_from_dir()`, `do_tests()`, `dump_consts()`, `dump_hd1()`, `fs_create_delete()`, `fs_init()`, `fs_shutdown()`, `grubstruct_test()`, `hd_stressread_test()`, `hd_stresswrite_test()`, `hd_write_test()`, `insert_file_into_dir()`, `isr_test()`, `load_fs()`, `main()`, `malloc_test()`, `mm_init_output()`, `pm_register_device()`, `print_time()`, `printf_test()`, `ralph_wiggum()`, `rf_dump()`, `rf_write()`, `sleep_test()`, `strings_test()`, `strsep_test()`, `syscall_test_thread()`, and `wait_on_hd_interrupt()`.

**5.108.1.13 #define printf potatoes\_printf****5.108.1.14 #define putchar potatoes\_putchar**

Referenced by `dump_hd1()`, and `nullptr_test()`.



**5.108.1.15 #define puts potatoes\_puts**

Referenced by `hd_test()`, `printf()`, and `sys_log()`.

**5.108.1.16 #define snprintf potatoes\_snprintf****5.108.1.17 #define strcat potatoes\_strcat**

Referenced by `potatoes_strcat()`, `shell_autocomplete()`, `shell_cmd_ls()`, `shell_cmd_mkdir()`, `shell_makepath()`, `strings_test()`, `test_create()`, and `test_rw_qualitative()`.

**5.108.1.18 #define strcmp potatoes\_strcmp**

Referenced by `create_entry()`, `delete_entry()`, `find_filename()`, `fs_create_delete()`, `fs_open()`, `get_color_tag()`, `main()`, `name2desc()`, `pm_name2device()`, `potatoes_strcmp()`, `search_file()`, `shell_autocomplete()`, `shell_cmd_bf()`, `shell_cmd_pong()`, `shell_cmd_snake()`, and `shell_handle_command()`.

**5.108.1.19 #define strdup potatoes\_strdup**

Referenced by `insert_file()`, `pm_create_thread()`, `potatoes_strdup()`, `search_file()`, `shell_handle_command()`, `strsep_test()`, and `sys_open()`.

**5.108.1.20 #define strlen potatoes\_strlen**

Referenced by `_fputs()`, `dev_clock_read()`, `draw_test()`, `get_filename()`, `get_path()`, `init_bf()`, `potatoes_strlen()`, `shell_autocomplete()`, `shell_cmd_bf()`, `shell_cmd_exec()`, `shell_cmd_ls()`, `shell_cmd_mkdir()`, `shell_cmd_write()`, `shell_handle_command()`, `shell_main()`, `shell_makepath()`, `snake()`, `snapshot()`, `speed()`, `start_vmonitor()`, `strdup()`, `strreverse()`, `sys_open()`, `syscall_test_thread()`, and `test_batch_files()`.

**5.108.1.21 #define strncpy potatoes\_strncpy**

Referenced by `create_heap()`, `dev_clock_read()`, `get_file_info()`, `heap_contract()`, `heap_expand()`, `heap_setup_block()`, `potatoes_strncpy()`, `shell_autocomplete()`, `shell_cmd_ls()`, and `strncat()`.

**5.108.1.22 #define strsep potatoes\_strsep**

Referenced by `potatoes_strsep()`, `rfsearch()`, `search_file()`, `shell_handle_command()`, and `strsep_test()`.

**5.108.1.23 #define time2str potatoes\_time2str**

Referenced by `dev_clock_read()`, `fs_write()`, `get_active_virt_monitor_name()`, `print_time()`, `shell_cmd_ls()`, `threadA()`, and `threadB()`.

**5.108.1.24 #define vsnprintf potatoes\_vsnprintf**

Referenced by `_printf()`, `aprntf()`, `printf()`, and `snprintf()`.