

MPX-Fall2020-Group9

2

Generated by Doxygen 1.9.0



<b>1 MPX-Fall2020-Group9</b>	<b>1</b>
<b>2 Class Index</b>	<b>3</b>
2.1 Class List . . . . .	3
<b>3 File Index</b>	<b>5</b>
3.1 File List . . . . .	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 date_time Struct Reference . . . . .	7
4.1.1 Detailed Description . . . . .	7
4.1.2 Member Data Documentation . . . . .	7
4.1.2.1 day_m . . . . .	7
4.1.2.2 day_w . . . . .	8
4.1.2.3 day_y . . . . .	8
4.1.2.4 hour . . . . .	8
4.1.2.5 min . . . . .	8
4.1.2.6 mon . . . . .	8
4.1.2.7 sec . . . . .	8
4.1.2.8 year . . . . .	9
4.2 footer Struct Reference . . . . .	9
4.2.1 Detailed Description . . . . .	9
4.2.2 Member Data Documentation . . . . .	9
4.2.2.1 head . . . . .	9
4.3 gdt_descriptor_struct Struct Reference . . . . .	9
4.3.1 Detailed Description . . . . .	10
4.3.2 Member Data Documentation . . . . .	10
4.3.2.1 base . . . . .	10
4.3.2.2 limit . . . . .	10
4.4 gdt_entry_struct Struct Reference . . . . .	10
4.4.1 Detailed Description . . . . .	10
4.4.2 Member Data Documentation . . . . .	11
4.4.2.1 access . . . . .	11
4.4.2.2 base_high . . . . .	11
4.4.2.3 base_low . . . . .	11
4.4.2.4 base_mid . . . . .	11
4.4.2.5 flags . . . . .	11
4.4.2.6 limit_low . . . . .	12
4.5 header Struct Reference . . . . .	12
4.5.1 Detailed Description . . . . .	12
4.5.2 Member Data Documentation . . . . .	12
4.5.2.1 index_id . . . . .	12
4.5.2.2 size . . . . .	12

4.6 heap Struct Reference . . . . .	13
4.6.1 Detailed Description . . . . .	13
4.6.2 Member Data Documentation . . . . .	13
4.6.2.1 base . . . . .	13
4.6.2.2 index . . . . .	13
4.6.2.3 max_size . . . . .	13
4.6.2.4 min_size . . . . .	14
4.7 idt_entry_struct Struct Reference . . . . .	14
4.7.1 Detailed Description . . . . .	14
4.7.2 Member Data Documentation . . . . .	14
4.7.2.1 base_high . . . . .	14
4.7.2.2 base_low . . . . .	14
4.7.2.3 flags . . . . .	15
4.7.2.4 sselect . . . . .	15
4.7.2.5 zero . . . . .	15
4.8 idt_struct Struct Reference . . . . .	15
4.8.1 Detailed Description . . . . .	15
4.8.2 Member Data Documentation . . . . .	15
4.8.2.1 base . . . . .	16
4.8.2.2 limit . . . . .	16
4.9 index_entry Struct Reference . . . . .	16
4.9.1 Detailed Description . . . . .	16
4.9.2 Member Data Documentation . . . . .	16
4.9.2.1 block . . . . .	16
4.9.2.2 empty . . . . .	17
4.9.2.3 size . . . . .	17
4.10 index_table Struct Reference . . . . .	17
4.10.1 Detailed Description . . . . .	17
4.10.2 Member Data Documentation . . . . .	17
4.10.2.1 id . . . . .	17
4.10.2.2 table . . . . .	18
4.11 page_dir Struct Reference . . . . .	18
4.11.1 Detailed Description . . . . .	18
4.11.2 Member Data Documentation . . . . .	18
4.11.2.1 tables . . . . .	18
4.11.2.2 tables_phys . . . . .	18
4.12 page_entry Struct Reference . . . . .	19
4.12.1 Detailed Description . . . . .	19
4.12.2 Member Data Documentation . . . . .	19
4.12.2.1 accessed . . . . .	19
4.12.2.2 dirty . . . . .	19
4.12.2.3 frameaddr . . . . .	19

4.12.2.4 present . . . . .	20
4.12.2.5 reserved . . . . .	20
4.12.2.6 usermode . . . . .	20
4.12.2.7 writeable . . . . .	20
4.13 page_table Struct Reference . . . . .	20
4.13.1 Detailed Description . . . . .	20
4.13.2 Member Data Documentation . . . . .	21
4.13.2.1 pages . . . . .	21
4.14 param Struct Reference . . . . .	21
4.14.1 Detailed Description . . . . .	21
4.14.2 Member Data Documentation . . . . .	21
4.14.2.1 buffer_ptr . . . . .	21
4.14.2.2 count_ptr . . . . .	22
4.14.2.3 device_id . . . . .	22
4.14.2.4 op_code . . . . .	22
4.15 PCB Struct Reference . . . . .	22
4.15.1 Detailed Description . . . . .	22
4.15.2 Member Data Documentation . . . . .	23
4.15.2.1 nextPCB . . . . .	23
4.15.2.2 prevPCB . . . . .	23
4.15.2.3 priority . . . . .	23
4.15.2.4 processClass . . . . .	23
4.15.2.5 processName . . . . .	23
4.15.2.6 runningStatus . . . . .	24
4.15.2.7 stack . . . . .	24
4.15.2.8 stackBase . . . . .	24
4.15.2.9 stackTop . . . . .	24
4.15.2.10 suspendedStatus . . . . .	24
4.16 queue Struct Reference . . . . .	24
4.16.1 Detailed Description . . . . .	25
4.16.2 Member Data Documentation . . . . .	25
4.16.2.1 count . . . . .	25
4.16.2.2 head . . . . .	25
4.16.2.3 tail . . . . .	25
<b>5 File Documentation</b> . . . . .	<b>27</b>
5.1 include/core/asm.h File Reference . . . . .	27
5.2 include/core/interrupts.h File Reference . . . . .	27
5.2.1 Function Documentation . . . . .	27
5.2.1.1 init_irq() . . . . .	27
5.2.1.2 init_pic() . . . . .	28
5.3 include/core/io.h File Reference . . . . .	28

5.3.1 Macro Definition Documentation	28
5.3.1.1 inb	29
5.3.1.2 outb	29
5.4 include/core/serial.h File Reference	29
5.4.1 Macro Definition Documentation	29
5.4.1.1 COM1	30
5.4.1.2 COM2	30
5.4.1.3 COM3	30
5.4.1.4 COM4	30
5.4.2 Function Documentation	30
5.4.2.1 init_serial()	30
5.4.2.2 polling()	31
5.4.2.3 serial_print()	33
5.4.2.4 serial_println()	33
5.4.2.5 set_serial_in()	33
5.4.2.6 set_serial_out()	34
5.5 include/core/tables.h File Reference	34
5.5.1 Function Documentation	34
5.5.1.1 __attribute__()	35
5.5.1.2 gdt_init_entry()	35
5.5.1.3 idt_set_gate()	35
5.5.1.4 init_gdt()	35
5.5.1.5 init_idt()	36
5.5.2 Variable Documentation	36
5.5.2.1 access	36
5.5.2.2 base	36
5.5.2.3 base_high	36
5.5.2.4 base_low	36
5.5.2.5 base_mid	37
5.5.2.6 flags	37
5.5.2.7 limit	37
5.5.2.8 limit_low	37
5.5.2.9 sselect	37
5.5.2.10 zero	37
5.6 include/mem/heap.h File Reference	38
5.6.1 Macro Definition Documentation	38
5.6.1.1 KHEAP_BASE	38
5.6.1.2 KHEAP_MIN	38
5.6.1.3 KHEAP_SIZE	39
5.6.1.4 TABLE_SIZE	39
5.6.2 Function Documentation	39
5.6.2.1 _kmalloc()	39

5.6.2.2 alloc()	40
5.6.2.3 init_kheap()	40
5.6.2.4 kfree()	40
5.6.2.5 kmalloc()	40
5.6.2.6 make_heap()	40
5.7 include/mem/paging.h File Reference	41
5.7.1 Macro Definition Documentation	41
5.7.1.1 PAGE_SIZE	41
5.7.2 Function Documentation	41
5.7.2.1 clear_bit()	42
5.7.2.2 first_free()	42
5.7.2.3 get_bit()	42
5.7.2.4 get_page()	42
5.7.2.5 init_paging()	43
5.7.2.6 load_page_dir()	43
5.7.2.7 new_frame()	43
5.7.2.8 set_bit()	44
5.8 include/string.h File Reference	44
5.8.1 Function Documentation	44
5.8.1.1 atoi()	45
5.8.1.2 isspace()	45
5.8.1.3 memset()	45
5.8.1.4 strcat()	46
5.8.1.5 strcmp()	46
5.8.1.6 strcpy()	46
5.8.1.7 strlen()	46
5.8.1.8 strtok()	47
5.9 include/system.h File Reference	47
5.9.1 Macro Definition Documentation	48
5.9.1.1 asm	48
5.9.1.2 cli	48
5.9.1.3 GDT_CS_ID	48
5.9.1.4 GDT_DS_ID	49
5.9.1.5 hlt	49
5.9.1.6 iret	49
5.9.1.7 no_warn	49
5.9.1.8 nop	49
5.9.1.9 NULL	49
5.9.1.10 sti	50
5.9.1.11 volatile	50
5.9.2 Typedef Documentation	50
5.9.2.1 size_t	50

---

5.9.2.2 u16int . . . . .	50
5.9.2.3 u32int . . . . .	50
5.9.2.4 u8int . . . . .	50
5.9.3 Function Documentation . . . . .	51
5.9.3.1 irq_on() . . . . .	51
5.9.3.2 klogv() . . . . .	51
5.9.3.3 kpanic() . . . . .	51
5.10 kernel/core/interrupts.c File Reference . . . . .	51
5.10.1 Macro Definition Documentation . . . . .	53
5.10.1.1 ICW1 . . . . .	53
5.10.1.2 ICW4 . . . . .	53
5.10.1.3 io_wait . . . . .	53
5.10.1.4 PIC1 . . . . .	53
5.10.1.5 PIC2 . . . . .	53
5.10.2 Function Documentation . . . . .	53
5.10.2.1 bounds() . . . . .	54
5.10.2.2 breakpoint() . . . . .	54
5.10.2.3 coprocessor() . . . . .	54
5.10.2.4 coprocessor_segment() . . . . .	54
5.10.2.5 debug() . . . . .	54
5.10.2.6 device_not_available() . . . . .	54
5.10.2.7 divide_error() . . . . .	54
5.10.2.8 do_bounds() . . . . .	55
5.10.2.9 do_breakpoint() . . . . .	55
5.10.2.10 do_coprocessor() . . . . .	55
5.10.2.11 do_coprocessor_segment() . . . . .	55
5.10.2.12 do_debug() . . . . .	55
5.10.2.13 do_device_not_available() . . . . .	56
5.10.2.14 do_divide_error() . . . . .	56
5.10.2.15 do_double_fault() . . . . .	56
5.10.2.16 do_general_protection() . . . . .	56
5.10.2.17 do_invalid_op() . . . . .	56
5.10.2.18 do_invalid_tss() . . . . .	57
5.10.2.19 do_isr() . . . . .	57
5.10.2.20 do_nmi() . . . . .	57
5.10.2.21 do_overflow() . . . . .	57
5.10.2.22 do_page_fault() . . . . .	57
5.10.2.23 do_reserved() . . . . .	58
5.10.2.24 do_segment_not_present() . . . . .	58
5.10.2.25 do_stack_segment() . . . . .	58
5.10.2.26 double_fault() . . . . .	58
5.10.2.27 general_protection() . . . . .	58

---



5.10.2.28 init_irq()	59
5.10.2.29 init_pic()	59
5.10.2.30 invalid_op()	60
5.10.2.31 invalid_tss()	60
5.10.2.32 isr0()	60
5.10.2.33 nmi()	60
5.10.2.34 overflow()	60
5.10.2.35 page_fault()	60
5.10.2.36 reserved()	60
5.10.2.37 rtc_isr()	60
5.10.2.38 segment_not_present()	61
5.10.2.39 stack_segment()	61
5.10.3 Variable Documentation	61
5.10.3.1 idt_entries	61
5.11 kernel/core/kmain.c File Reference	61
5.11.1 Function Documentation	61
5.11.1.1 kmain()	62
5.12 kernel/core/serial.c File Reference	62
5.12.1 Macro Definition Documentation	63
5.12.1.1 NO_ERROR	63
5.12.2 Function Documentation	63
5.12.2.1 init_serial()	63
5.12.2.2 polling()	64
5.12.2.3 serial_print()	66
5.12.2.4 serial_println()	66
5.12.2.5 set_serial_in()	66
5.12.2.6 set_serial_out()	67
5.12.3 Variable Documentation	67
5.12.3.1 serial_port_in	67
5.12.3.2 serial_port_out	67
5.13 kernel/core/system.c File Reference	67
5.13.1 Function Documentation	67
5.13.1.1 klogv()	68
5.13.1.2 kpanic()	68
5.14 kernel/core/tables.c File Reference	68
5.14.1 Function Documentation	69
5.14.1.1 gdt_init_entry()	69
5.14.1.2 idt_set_gate()	69
5.14.1.3 init_gdt()	69
5.14.1.4 init_idt()	70
5.14.1.5 write_gdt_ptr()	70
5.14.1.6 write_idt_ptr()	70

5.14.2 Variable Documentation . . . . .	70
5.14.2.1 gdt_entries . . . . .	70
5.14.2.2 gdt_ptr . . . . .	70
5.14.2.3 idt_entries . . . . .	71
5.14.2.4 idt_ptr . . . . .	71
5.15 kernel/mem/heap.c File Reference . . . . .	71
5.15.1 Function Documentation . . . . .	71
5.15.1.1 _kmalloc() . . . . .	72
5.15.1.2 alloc() . . . . .	72
5.15.1.3 kmalloc() . . . . .	72
5.15.1.4 make_heap() . . . . .	73
5.15.2 Variable Documentation . . . . .	73
5.15.2.1 __end . . . . .	73
5.15.2.2 _end . . . . .	73
5.15.2.3 curr_heap . . . . .	73
5.15.2.4 end . . . . .	73
5.15.2.5 kdir . . . . .	74
5.15.2.6 kheap . . . . .	74
5.15.2.7 phys_alloc_addr . . . . .	74
5.16 kernel/mem/paging.c File Reference . . . . .	74
5.16.1 Function Documentation . . . . .	75
5.16.1.1 clear_bit() . . . . .	75
5.16.1.2 find_free() . . . . .	75
5.16.1.3 get_bit() . . . . .	75
5.16.1.4 get_page() . . . . .	76
5.16.1.5 init_paging() . . . . .	76
5.16.1.6 load_page_dir() . . . . .	77
5.16.1.7 new_frame() . . . . .	77
5.16.1.8 set_bit() . . . . .	77
5.16.2 Variable Documentation . . . . .	77
5.16.2.1 cdir . . . . .	77
5.16.2.2 frames . . . . .	78
5.16.2.3 kdir . . . . .	78
5.16.2.4 kheap . . . . .	78
5.16.2.5 mem_size . . . . .	78
5.16.2.6 nframes . . . . .	78
5.16.2.7 page_size . . . . .	78
5.16.2.8 phys_alloc_addr . . . . .	79
5.17 lib/string.c File Reference . . . . .	79
5.17.1 Function Documentation . . . . .	79
5.17.1.1 atoi() . . . . .	79
5.17.1.2 isspace() . . . . .	80

5.17.1.3 <code>memset()</code>	80
5.17.1.4 <code>strcat()</code>	80
5.17.1.5 <code>strcmp()</code>	81
5.17.1.6 <code>strcpy()</code>	81
5.17.1.7 <code>strlen()</code>	81
5.17.1.8 <code>strtok()</code>	82
5.18 modules/mpx_supt.c File Reference	82
5.18.1 Function Documentation	83
5.18.1.1 <code>idle()</code>	83
5.18.1.2 <code>mpx_init()</code>	83
5.18.1.3 <code>sys_alloc_mem()</code>	83
5.18.1.4 <code>sys_free_mem()</code>	84
5.18.1.5 <code>sys_req()</code>	84
5.18.1.6 <code>sys_set_free()</code>	85
5.18.1.7 <code>sys_set_malloc()</code>	85
5.18.2 Variable Documentation	85
5.18.2.1 <code>current_module</code>	85
5.18.2.2 <code>io_module_active</code>	85
5.18.2.3 <code>mem_module_active</code>	85
5.18.2.4 <code>params</code>	86
5.18.2.5 <code>student_free</code>	86
5.18.2.6 <code>student_malloc</code>	86
5.19 modules/mpx_supt.h File Reference	86
5.19.1 Macro Definition Documentation	87
5.19.1.1 <code>COM_PORT</code>	87
5.19.1.2 <code>DEFAULT_DEVICE</code>	87
5.19.1.3 <code>EXIT</code>	87
5.19.1.4 <code>FALSE</code>	87
5.19.1.5 <code>IDLE</code>	88
5.19.1.6 <code>INVALID_BUFFER</code>	88
5.19.1.7 <code>INVALID_COUNT</code>	88
5.19.1.8 <code>INVALID_OPERATION</code>	88
5.19.1.9 <code>IO_MODULE</code>	88
5.19.1.10 <code>MEM_MODULE</code>	88
5.19.1.11 <code>MODULE_F</code>	89
5.19.1.12 <code>MODULE_R1</code>	89
5.19.1.13 <code>MODULE_R2</code>	89
5.19.1.14 <code>MODULE_R3</code>	89
5.19.1.15 <code>MODULE_R4</code>	89
5.19.1.16 <code>MODULE_R5</code>	89
5.19.1.17 <code>READ</code>	90
5.19.1.18 <code>TRUE</code>	90

5.19.1.19 WRITE . . . . .	90
5.19.2 Function Documentation . . . . .	90
5.19.2.1 idle() . . . . .	90
5.19.2.2 mpx_init() . . . . .	91
5.19.2.3 sys_alloc_mem() . . . . .	91
5.19.2.4 sys_free_mem() . . . . .	91
5.19.2.5 sys_req() . . . . .	91
5.19.2.6 sys_set_free() . . . . .	92
5.19.2.7 sys_set_malloc() . . . . .	92
5.20 modules/R1/commhand.c File Reference . . . . .	92
5.20.1 Function Documentation . . . . .	93
5.20.1.1 commhand() . . . . .	93
5.21 modules/R1/commhand.h File Reference . . . . .	96
5.21.1 Function Documentation . . . . .	96
5.21.1.1 commhand() . . . . .	96
5.22 modules/R1/R1commands.c File Reference . . . . .	99
5.22.1 Function Documentation . . . . .	99
5.22.1.1 BCDtoChar() . . . . .	99
5.22.1.2 getDate() . . . . .	100
5.22.1.3 getTime() . . . . .	100
5.22.1.4 help() . . . . .	101
5.22.1.5 intToBCD() . . . . .	103
5.22.1.6 quit() . . . . .	103
5.22.1.7 setDate() . . . . .	103
5.22.1.8 setTime() . . . . .	106
5.22.1.9 version() . . . . .	107
5.23 modules/R1/R1commands.h File Reference . . . . .	108
5.23.1 Function Documentation . . . . .	108
5.23.1.1 BCDtoChar() . . . . .	108
5.23.1.2 change_int_to_binary() . . . . .	108
5.23.1.3 getDate() . . . . .	109
5.23.1.4 getTime() . . . . .	109
5.23.1.5 help() . . . . .	110
5.23.1.6 quit() . . . . .	112
5.23.1.7 setDate() . . . . .	112
5.23.1.8 setTime() . . . . .	115
5.23.1.9 version() . . . . .	116
5.24 modules/R2/R2_Internal_Functions_And_Structures.c File Reference . . . . .	116
5.24.1 Function Documentation . . . . .	117
5.24.1.1 allocatePCB() . . . . .	117
5.24.1.2 allocateQueues() . . . . .	118
5.24.1.3 findPCB() . . . . .	118

5.24.1.4 freePCB()	119
5.24.1.5 getBlocked()	119
5.24.1.6 getReady()	120
5.24.1.7 getSuspendedBlocked()	120
5.24.1.8 getSuspendedReady()	120
5.24.1.9 insertPCB()	120
5.24.1.10 removePCB()	121
5.24.1.11 setupPCB()	123
5.24.2 Variable Documentation	123
5.24.2.1 blocked	124
5.24.2.2 ready	124
5.24.2.3 suspendedBlocked	124
5.24.2.4 suspendedReady	124
5.25 modules/R2/R2_Internal_Functions_And_Structures.h File Reference	124
5.25.1 Typedef Documentation	125
5.25.1.1 PCB	125
5.25.1.2 queue	125
5.25.2 Function Documentation	125
5.25.2.1 allocatePCB()	125
5.25.2.2 allocateQueues()	126
5.25.2.3 findPCB()	126
5.25.2.4 freePCB()	127
5.25.2.5 getBlocked()	127
5.25.2.6 getReady()	128
5.25.2.7 getSuspendedBlocked()	128
5.25.2.8 getSuspendedReady()	128
5.25.2.9 insertPCB()	128
5.25.2.10 removePCB()	129
5.25.2.11 setupPCB()	131
5.26 modules/R2/R2commands.c File Reference	131
5.26.1 Function Documentation	132
5.26.1.1 blockPCB()	132
5.26.1.2 createPCB()	133
5.26.1.3 deletePCB()	133
5.26.1.4 resumePCB()	134
5.26.1.5 setPCBPriority()	135
5.26.1.6 showAll()	135
5.26.1.7 showBlocked()	135
5.26.1.8 showPCB()	136
5.26.1.9 showReady()	138
5.26.1.10 showSuspendedBlocked()	139
5.26.1.11 showSuspendedReady()	140

5.26.1.12 suspendPCB()	140
5.26.1.13 unblockPCB()	141
5.27 modules/R2/R2commands.h File Reference	141
5.27.1 Function Documentation	142
5.27.1.1 blockPCB()	142
5.27.1.2 createPCB()	142
5.27.1.3 deletePCB()	143
5.27.1.4 resumePCB()	144
5.27.1.5 setPCBPRIORITY()	144
5.27.1.6 showAll()	145
5.27.1.7 showBlocked()	145
5.27.1.8 showPCB()	146
5.27.1.9 showReady()	148
5.27.1.10 showSuspendedBlocked()	149
5.27.1.11 showSuspendedReady()	149
5.27.1.12 suspendPCB()	150
5.27.1.13 unblockPCB()	150
5.28 README.md File Reference	151

## **Chapter 1**

# **MPX-Fall2020-Group9**

WVU CS 450 MPX Project files Making operating system// test message





## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">date_time</a>	7
<a href="#">footer</a>	9
<a href="#">gdt_descriptor_struct</a>	9
<a href="#">gdt_entry_struct</a>	10
<a href="#">header</a>	12
<a href="#">heap</a>	13
<a href="#">idt_entry_struct</a>	14
<a href="#">idt_struct</a>	15
<a href="#">index_entry</a>	16
<a href="#">index_table</a>	17
<a href="#">page_dir</a>	18
<a href="#">page_entry</a>	19
<a href="#">page_table</a>	20
<a href="#">param</a>	21
<a href="#">PCB</a>	22
<a href="#">queue</a>	24



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

include/string.h . . . . .	44
include/system.h . . . . .	47
include/core/asm.h . . . . .	27
include/core/interrupts.h . . . . .	27
include/core/io.h . . . . .	28
include/core/serial.h . . . . .	29
include/core/tables.h . . . . .	34
include/mem/heap.h . . . . .	38
include/mem/paging.h . . . . .	41
kernel/core/interrupts.c . . . . .	51
kernel/core/kmain.c . . . . .	61
kernel/core/serial.c . . . . .	62
kernel/core/system.c . . . . .	67
kernel/core/tables.c . . . . .	68
kernel/mem/heap.c . . . . .	71
kernel/mem/paging.c . . . . .	74
lib/string.c . . . . .	79
modules/mpx_supt.c . . . . .	82
modules/mpx_supt.h . . . . .	86
modules/R1/commhand.c . . . . .	92
modules/R1/commhand.h . . . . .	96
modules/R1/R1commands.c . . . . .	99
modules/R1/R1commands.h . . . . .	108
modules/R2/R2_Internal_Functions_And_Structures.c . . . . .	116
modules/R2/R2_Internal_Functions_And_Structures.h . . . . .	124
modules/R2/R2commands.c . . . . .	131
modules/R2/R2commands.h . . . . .	141



# Chapter 4

## Class Documentation

### 4.1 date\_time Struct Reference

```
#include <system.h>
```

#### Public Attributes

- int [sec](#)
- int [min](#)
- int [hour](#)
- int [day\\_w](#)
- int [day\\_m](#)
- int [day\\_y](#)
- int [mon](#)
- int [year](#)

#### 4.1.1 Detailed Description

Definition at line 30 of file system.h.

#### 4.1.2 Member Data Documentation

##### 4.1.2.1 day\_m

```
int date_time::day_m
```

Definition at line 35 of file system.h.

#### 4.1.2.2 day\_w

```
int date_time::day_w
```

Definition at line 34 of file system.h.

#### 4.1.2.3 day\_y

```
int date_time::day_y
```

Definition at line 36 of file system.h.

#### 4.1.2.4 hour

```
int date_time::hour
```

Definition at line 33 of file system.h.

#### 4.1.2.5 min

```
int date_time::min
```

Definition at line 32 of file system.h.

#### 4.1.2.6 mon

```
int date_time::mon
```

Definition at line 37 of file system.h.

#### 4.1.2.7 sec

```
int date_time::sec
```

Definition at line 31 of file system.h.

#### 4.1.2.8 year

```
int date_time::year
```

Definition at line 38 of file system.h.

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

- include/[system.h](#)

## 4.2 footer Struct Reference

```
#include <heap.h>
```

### Public Attributes

- [header head](#)

#### 4.2.1 Detailed Description

Definition at line 16 of file heap.h.

#### 4.2.2 Member Data Documentation

##### 4.2.2.1 head

```
header footer::head
```

Definition at line 17 of file heap.h.

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

- include/mem/[heap.h](#)

## 4.3 gdt\_descriptor\_struct Struct Reference

```
#include <tables.h>
```

### Public Attributes

- [u16int limit](#)
- [u32int base](#)

### 4.3.1 Detailed Description

Definition at line 23 of file tables.h.

### 4.3.2 Member Data Documentation

#### 4.3.2.1 base

```
u32int gdt_descriptor_struct::base
```

Definition at line 26 of file tables.h.

#### 4.3.2.2 limit

```
u16int gdt_descriptor_struct::limit
```

Definition at line 25 of file tables.h.

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

- [include/core/tables.h](#)

## 4.4 gdt\_entry\_struct Struct Reference

```
#include <tables.h>
```

### Public Attributes

- [u16int limit\\_low](#)
- [u16int base\\_low](#)
- [u8int base\\_mid](#)
- [u8int access](#)
- [u8int flags](#)
- [u8int base\\_high](#)

#### 4.4.1 Detailed Description

Definition at line 30 of file tables.h.



## 4.4.2 Member Data Documentation

### 4.4.2.1 access

```
u8int gdt_entry_struct::access
```

Definition at line 35 of file tables.h.

### 4.4.2.2 base\_high

```
u8int gdt_entry_struct::base_high
```

Definition at line 37 of file tables.h.

### 4.4.2.3 base\_low

```
u16int gdt_entry_struct::base_low
```

Definition at line 33 of file tables.h.

### 4.4.2.4 base\_mid

```
u8int gdt_entry_struct::base_mid
```

Definition at line 34 of file tables.h.

### 4.4.2.5 flags

```
u8int gdt_entry_struct::flags
```

Definition at line 36 of file tables.h.

#### 4.4.2.6 limit\_low

```
ul6int gdt_entry_struct::limit_low
```

Definition at line 32 of file tables.h.

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

- include/core/[tables.h](#)

### 4.5 header Struct Reference

```
#include <heap.h>
```

#### Public Attributes

- int [size](#)
- int [index\\_id](#)

#### 4.5.1 Detailed Description

Definition at line 11 of file heap.h.

#### 4.5.2 Member Data Documentation

##### 4.5.2.1 index\_id

```
int header::index_id
```

Definition at line 13 of file heap.h.

##### 4.5.2.2 size

```
int header::size
```

Definition at line 12 of file heap.h.

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

- include/mem/[heap.h](#)

## 4.6 heap Struct Reference

```
#include <heap.h>
```

### Public Attributes

- [index\\_table](#) index
- [u32int](#) base
- [u32int](#) max\_size
- [u32int](#) min\_size

### 4.6.1 Detailed Description

Definition at line 33 of file heap.h.

### 4.6.2 Member Data Documentation

#### 4.6.2.1 base

```
u32int heap::base
```

Definition at line 35 of file heap.h.

#### 4.6.2.2 index

```
index\_table heap::index
```

Definition at line 34 of file heap.h.

#### 4.6.2.3 max\_size

```
u32int heap::max_size
```

Definition at line 36 of file heap.h.

#### 4.6.2.4 min\_size

```
u32int heap::min_size
```

Definition at line 37 of file heap.h.

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

- include/mem/heap.h

## 4.7 idt\_entry\_struct Struct Reference

```
#include <tables.h>
```

### Public Attributes

- u16int base\_low
- u16int sselect
- u8int zero
- u8int flags
- u16int base\_high

#### 4.7.1 Detailed Description

Definition at line 6 of file tables.h.

#### 4.7.2 Member Data Documentation

##### 4.7.2.1 base\_high

```
u16int idt_entry_struct::base_high
```

Definition at line 12 of file tables.h.

##### 4.7.2.2 base\_low

```
u16int idt_entry_struct::base_low
```

Definition at line 8 of file tables.h.

#### 4.7.2.3 flags

```
u8int idt_entry_struct::flags
```

Definition at line 11 of file tables.h.

#### 4.7.2.4 sselect

```
u16int idt_entry_struct::sselect
```

Definition at line 9 of file tables.h.

#### 4.7.2.5 zero

```
u8int idt_entry_struct::zero
```

Definition at line 10 of file tables.h.

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

- [include/core/tables.h](#)

## 4.8 idt\_struct Struct Reference

```
#include <tables.h>
```

### Public Attributes

- [u16int limit](#)
- [u32int base](#)

### 4.8.1 Detailed Description

Definition at line 16 of file tables.h.

### 4.8.2 Member Data Documentation

#### 4.8.2.1 base

```
u32int idt_struct::base
```

Definition at line 19 of file tables.h.

#### 4.8.2.2 limit

```
u16int idt_struct::limit
```

Definition at line 18 of file tables.h.

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

- [include/core/tables.h](#)

### 4.9 index\_entry Struct Reference

```
#include <heap.h>
```

#### Public Attributes

- int [size](#)
- int [empty](#)
- u32int [block](#)

#### 4.9.1 Detailed Description

Definition at line 20 of file heap.h.

#### 4.9.2 Member Data Documentation

##### 4.9.2.1 block

```
u32int index_entry::block
```

Definition at line 23 of file heap.h.

#### 4.9.2.2 `empty`

```
int index_entry::empty
```

Definition at line 22 of file `heap.h`.

#### 4.9.2.3 `size`

```
int index_entry::size
```

Definition at line 21 of file `heap.h`.

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

- `include/mem/heap.h`

## 4.10 `index_table` Struct Reference

```
#include <heap.h>
```

### Public Attributes

- `index_entry table` [0x1000]
- `int id`

#### 4.10.1 Detailed Description

Definition at line 27 of file `heap.h`.

#### 4.10.2 Member Data Documentation

##### 4.10.2.1 `id`

```
int index_table::id
```

Definition at line 29 of file `heap.h`.

#### 4.10.2.2 table

```
index_entry index_table::table[0x1000]
```

Definition at line 28 of file heap.h.

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

- include/mem/heap.h

### 4.11 page\_dir Struct Reference

```
#include <paging.h>
```

#### Public Attributes

- [page\\_table](#) \*tables [1024]
- [u32int](#) tables\_phys [1024]

#### 4.11.1 Detailed Description

Definition at line 34 of file paging.h.

#### 4.11.2 Member Data Documentation

##### 4.11.2.1 tables

```
page_table* page_dir::tables[1024]
```

Definition at line 35 of file paging.h.

##### 4.11.2.2 tables\_phys

```
u32int page_dir::tables_phys[1024]
```

Definition at line 36 of file paging.h.

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

- include/mem/paging.h



## 4.12 page\_entry Struct Reference

```
#include <paging.h>
```

### Public Attributes

- `u32int present`: 1
- `u32int writeable`: 1
- `u32int usermode`: 1
- `u32int accessed`: 1
- `u32int dirty`: 1
- `u32int reserved`: 7
- `u32int frameaddr`: 20

### 4.12.1 Detailed Description

Definition at line 12 of file `paging.h`.

### 4.12.2 Member Data Documentation

#### 4.12.2.1 accessed

```
u32int page_entry::accessed
```

Definition at line 16 of file `paging.h`.

#### 4.12.2.2 dirty

```
u32int page_entry::dirty
```

Definition at line 17 of file `paging.h`.

#### 4.12.2.3 frameaddr

```
u32int page_entry::frameaddr
```

Definition at line 19 of file `paging.h`.

#### 4.12.2.4 present

```
u32int page_entry::present
```

Definition at line 13 of file paging.h.

#### 4.12.2.5 reserved

```
u32int page_entry::reserved
```

Definition at line 18 of file paging.h.

#### 4.12.2.6 usermode

```
u32int page_entry::usermode
```

Definition at line 15 of file paging.h.

#### 4.12.2.7 writeable

```
u32int page_entry::writeable
```

Definition at line 14 of file paging.h.

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

- [include/mem/paging.h](#)

### 4.13 page\_table Struct Reference

```
#include <paging.h>
```

#### Public Attributes

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

#### 4.13.1 Detailed Description

Definition at line 26 of file paging.h.

## 4.13.2 Member Data Documentation

### 4.13.2.1 pages

```
page_entry page_table::pages[1024]
```

Definition at line 27 of file paging.h.

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

- include/mem/paging.h

## 4.14 param Struct Reference

```
#include <mpx_supt.h>
```

### Public Attributes

- int [op\\_code](#)
- int [device\\_id](#)
- char \*[buffer\\_ptr](#)
- int \*[count\\_ptr](#)

### 4.14.1 Detailed Description

Definition at line 31 of file mpx\_supt.h.

## 4.14.2 Member Data Documentation

### 4.14.2.1 buffer\_ptr

```
char* param::buffer_ptr
```

Definition at line 34 of file mpx\_supt.h.

#### 4.14.2.2 count\_ptr

```
int* param::count_ptr
```

Definition at line 35 of file mpx\_supt.h.

#### 4.14.2.3 device\_id

```
int param::device_id
```

Definition at line 33 of file mpx\_supt.h.

#### 4.14.2.4 op\_code

```
int param::op_code
```

Definition at line 32 of file mpx\_supt.h.

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

- [modules/mpx\\_supt.h](#)

## 4.15 PCB Struct Reference

```
#include <R2_Internal_Functions_And_Structures.h>
```

### Public Attributes

- char [processName](#) [20]
- char [processClass](#)
- int [priority](#)
- int [runningStatus](#)
- int [suspendedStatus](#)
- unsigned char [stack](#) [1024]
- unsigned char [\\*stackTop](#)
- unsigned char [\\*stackBase](#)
- struct [PCB](#) [\\*nextPCB](#)
- struct [PCB](#) [\\*prevPCB](#)

#### 4.15.1 Detailed Description

Definition at line 1 of file R2\_Internal\_Functions\_And\_Structures.h.

## 4.15.2 Member Data Documentation

### 4.15.2.1 nextPCB

```
struct PCB* PCB::nextPCB
```

Definition at line 11 of file R2\_Internal\_Functions\_And\_Structures.h.

### 4.15.2.2 prevPCB

```
struct PCB* PCB::prevPCB
```

Definition at line 12 of file R2\_Internal\_Functions\_And\_Structures.h.

### 4.15.2.3 priority

```
int PCB::priority
```

Definition at line 5 of file R2\_Internal\_Functions\_And\_Structures.h.

### 4.15.2.4 processClass

```
char PCB::processClass
```

Definition at line 4 of file R2\_Internal\_Functions\_And\_Structures.h.

### 4.15.2.5 processName

```
char PCB::processName[20]
```

Definition at line 3 of file R2\_Internal\_Functions\_And\_Structures.h.

#### 4.15.2.6 runningStatus

```
int PCB::runningStatus
```

Definition at line 6 of file R2\_Internal\_Functions\_And\_Structures.h.

#### 4.15.2.7 stack

```
unsigned char PCB::stack[1024]
```

Definition at line 8 of file R2\_Internal\_Functions\_And\_Structures.h.

#### 4.15.2.8 stackBase

```
unsigned char* PCB::stackBase
```

Definition at line 10 of file R2\_Internal\_Functions\_And\_Structures.h.

#### 4.15.2.9 stackTop

```
unsigned char* PCB::stackTop
```

Definition at line 9 of file R2\_Internal\_Functions\_And\_Structures.h.

#### 4.15.2.10 suspendedStatus

```
int PCB::suspendedStatus
```

Definition at line 7 of file R2\_Internal\_Functions\_And\_Structures.h.

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

- [modules/R2/R2\\_Internal\\_Functions\\_And\\_Structures.h](#)

## 4.16 queue Struct Reference

```
#include <R2_Internal_Functions_And_Structures.h>
```

## Public Attributes

- int [count](#)
- [PCB](#) \*[head](#)
- [PCB](#) \*[tail](#)

### 4.16.1 Detailed Description

Definition at line 15 of file [R2\\_Internal\\_Functions\\_And\\_Structures.h](#).

### 4.16.2 Member Data Documentation

#### 4.16.2.1 count

```
int queue::count
```

Definition at line 17 of file [R2\\_Internal\\_Functions\\_And\\_Structures.h](#).

#### 4.16.2.2 head

```
PCB* queue::head
```

Definition at line 18 of file [R2\\_Internal\\_Functions\\_And\\_Structures.h](#).

#### 4.16.2.3 tail

```
PCB* queue::tail
```

Definition at line 19 of file [R2\\_Internal\\_Functions\\_And\\_Structures.h](#).

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

- [modules/R2/R2\\_Internal\\_Functions\\_And\\_Structures.h](#)





# Chapter 5

## File Documentation

### 5.1 include/core/asm.h File Reference

```
#include <system.h>
#include <tables.h>
```

### 5.2 include/core/interrupts.h File Reference

#### Functions

- void [init\\_irq](#) (void)
- void [init\\_pic](#) (void)

#### 5.2.1 Function Documentation

##### 5.2.1.1 [init\\_irq\(\)](#)

```
void init_irq (
    void )
```

Definition at line 66 of file interrupts.c.

```
67 {
68     int i;
69
70     // Necessary interrupt handlers for protected mode
71     u32int isrs[17] = {
72         (u32int)divide_error,
73         (u32int)debug,
74         (u32int)nmi,
75         (u32int)breakpoint,
76         (u32int)overflow,
77         (u32int)bounds,
78         (u32int)invalid_op,
79         (u32int)device_not_available,
80         (u32int)double_fault,
81         (u32int)coprocessor_segment,
```

```

82     (u32int)invalid_tss,
83     (u32int)segment_not_present,
84     (u32int)stack_segment,
85     (u32int)general_protection,
86     (u32int)page_fault,
87     (u32int)reserved,
88     (u32int)coprocessor
89 };
90
91 // Install handlers; 0x08=sel, 0x8e=flags
92 for(i=0; i<32; i++){
93     if (i<17) idt_set_gate(i, isrs[i], 0x08, 0x8e);
94     else idt_set_gate(i, (u32int)reserved, 0x08, 0x8e);
95 }
96 // Ignore interrupts from the real time clock
97 idt_set_gate(0x08, (u32int)rtc_isr, 0x08, 0x8e);
98 }

```

### 5.2.1.2 init\_pic()

```

void init_pic (
    void )

```

Definition at line 106 of file interrupts.c.

```

107 {
108     outb(PIC1,ICW1);    //send initialization code words 1 to PIC1
109     io_wait();
110     outb(PIC2,ICW1);    //send icw1 to PIC2
111     io_wait();
112     outb(PIC1+1,0x20);  //icw2: remap irq0 to 32
113     io_wait();
114     outb(PIC2+1,0x28);  //icw2: remap irq8 to 40
115     io_wait();
116     outb(PIC1+1,4);     //icw3
117     io_wait();
118     outb(PIC2+1,2);     //icw3
119     io_wait();
120     outb(PIC1+1,ICW4);  //icw4: 80x86, automatic handling
121     io_wait();
122     outb(PIC2+1,ICW4);  //icw4: 80x86, automatic handling
123     io_wait();
124     outb(PIC1+1,0xFF);  //disable irqs for PIC1
125     io_wait();
126     outb(PIC2+1,0xFF);  //disable irqs for PIC2
127 }

```

## 5.3 include/core/io.h File Reference

### Macros

- #define `outb`(port, data) `asm volatile ("outb %%al,%%dx" : : "a" (data), "d" (port))`
- #define `inb`(port)

### 5.3.1 Macro Definition Documentation

### 5.3.1.1 inb

```
#define inb(  
    port )
```

#### Value:

```
((  
    unsigned char r;  
    asm volatile ("inb %%dx, %%al": "=a" (r): "d" (port));  
    r;  
))
```

Definition at line 15 of file io.h.

### 5.3.1.2 outb

```
#define outb(  
    port,  
    data )  asm volatile ("outb %%al, %%dx" : : "a" (data), "d" (port))
```

Definition at line 8 of file io.h.

## 5.4 include/core/serial.h File Reference

### Macros

- #define [COM1](#) 0x3f8
- #define [COM2](#) 0x2f8
- #define [COM3](#) 0x3e8
- #define [COM4](#) 0x2e8

### Functions

- int [init\\_serial](#) (int device)
- int [serial\\_println](#) (const char \*msg)
- int [serial\\_print](#) (const char \*msg)
- int [set\\_serial\\_out](#) (int device)
- int [set\\_serial\\_in](#) (int device)
- int \*[polling](#) (char \*buffer, int \*count)

### 5.4.1 Macro Definition Documentation

#### 5.4.1.1 COM1

```
#define COM1 0x3f8
```

Definition at line 4 of file serial.h.

#### 5.4.1.2 COM2

```
#define COM2 0x2f8
```

Definition at line 5 of file serial.h.

#### 5.4.1.3 COM3

```
#define COM3 0x3e8
```

Definition at line 6 of file serial.h.

#### 5.4.1.4 COM4

```
#define COM4 0x2e8
```

Definition at line 7 of file serial.h.

### 5.4.2 Function Documentation

#### 5.4.2.1 init\_serial()

```
int init_serial (  
    int device )
```

Definition at line 22 of file serial.c.

```
23 {  
24     outb(device + 1, 0x00);           //disable interrupts  
25     outb(device + 3, 0x80);           //set line control register  
26     outb(device + 0, 115200 / 9600); //set bsd least sig bit  
27     outb(device + 1, 0x00);           //brd most significant bit  
28     outb(device + 3, 0x03);           //lock divisor; 8bits, no parity, one stop  
29     outb(device + 2, 0xC7);           //enable fifo, clear, 14byte threshold  
30     outb(device + 4, 0x0B);           //enable interrupts, rts/dsr set  
31     (void)inb(device);                //read bit to reset port  
32     return NO_ERROR;  
33 }
```

## 5.4.2.2 polling()

```
int* polling (
    char * buffer,
    int * count )
```

Definition at line 92 of file serial.c.

```
93 {
94     // insert your code to gather keyboard input via the technique of polling.
95
96     char keyboard_character;
97
98     int cursor = 0;
99
100    char log[] = {'\0', '\0', '\0', '\0'};
101
102    int characters_in_buffer = 0;
103
104    while (1)
105    {
106
107        if (inb(COM1 + 5) & 1)
108        {
109            // is there input char?
110            keyboard_character = inb(COM1); //read the char from COM1
111
112            if (keyboard_character == '\n' || keyboard_character == '\r')
113            { // HANDLING THE CARRIAGE RETURN AND NEW LINE CHARACTERS
114
115                buffer[characters_in_buffer] = ' \0';
116                break;
117            }
118            else if ((keyboard_character == 127 || keyboard_character == 8) && cursor > 0)
119            { // HANDLING THE BACKSPACE CHARACTER
120
121                //serial_println("Handleing backspace character.");
122                serial_print("\033[K");
123
124                buffer[cursor - 1] = ' \0';
125                serial_print("\b\b");
126                serial_print(buffer + cursor);
127                cursor--;
128
129                int temp_cursor = cursor;
130
131                while (buffer[temp_cursor + 1] != ' \0')
132                {
133                    buffer[temp_cursor] = buffer[temp_cursor + 1];
134                    buffer[temp_cursor + 1] = ' \0';
135                    temp_cursor++;
136                }
137
138                characters_in_buffer--;
139                cursor = characters_in_buffer;
140            }
141            else if (keyboard_character == '~' && cursor < 99)
142            { //HANDLING THE DELETE KEY
143                // \033[3~
144
145                serial_print("\033[K");
146
147                buffer[cursor + 1] = ' \0';
148                serial_print("\b\b");
149                serial_print(buffer + cursor);
150
151                int temp_cursor = cursor + 1;
152
153                while (buffer[temp_cursor + 1] != ' \0')
154                {
155                    buffer[temp_cursor] = buffer[temp_cursor + 1];
156                    buffer[temp_cursor + 1] = ' \0';
157                    temp_cursor++;
158                }
159
160                characters_in_buffer--;
161                cursor = characters_in_buffer;
162            }
163            else if (keyboard_character == '\033')
164            { // HANDLING FIRST CHARACTER FOR ARROW KEYS
165
166                log[0] = keyboard_character;
167            }
168            else if (keyboard_character == '[' && log[0] == '\033')
169            { // HANDLING SECOND CHARACTER FOR ARROW KEYS
```

```

170     log[1] = keyboard_character;
171 }
172 else if (log[0] == '\033' && log[1] == '[')
173 { // HANDLING LAST CHARACTER FOR ARROW KEYS
174     log[2] = keyboard_character;
175
176     if (keyboard_character == 'A')
177     { //Up arrow
178         //Call a history function from the commhand or do nothing
179     }
180     else if (keyboard_character == 'B')
181     { //Down arrow
182         //Call a history command from the commhand or do nothing
183     }
184     else if (keyboard_character == 'C' && cursor != 99)
185     { //Right arrow
186
187         serial_print("\033[C");
188         cursor++;
189     }
190     else if (keyboard_character == 'D' && cursor != 0)
191     { //Left arrow
192
193         serial_print("\033[D");
194         cursor--;
195     }
196
197     memset(log, '\0', 4);
198 }
199 else
200 {
201
202     if (cursor == 0 && buffer[cursor] == '\0') //Adding character at beginning of buffer
203     {
204         buffer[cursor] = keyboard_character;
205         serial_print(&keyboard_character);
206         cursor++;
207     }
208     else if (buffer[cursor] == '\0') //Adding character at the end of the buffer
209     {
210         buffer[cursor] = keyboard_character;
211         serial_print(&keyboard_character);
212         cursor++;
213     }
214     else //Inserting character to the middle of the buffer
215     {
216         char temp_buffer[strlen(buffer)];
217         memset(temp_buffer, '\0', strlen(buffer));
218
219         int temp_cursor = 0;
220         while (temp_cursor <= characters_in_buffer) //Filling the temp_buffer with all of the
characters from buffer, and inserting the new character.
221         {
222             if (temp_cursor < cursor)
223             {
224                 temp_buffer[temp_cursor] = buffer[temp_cursor];
225             }
226             else if (temp_cursor > cursor)
227             {
228                 temp_buffer[temp_cursor] = buffer[temp_cursor - 1];
229             }
230             else
231             { //temp_cursor == cursor
232                 temp_buffer[temp_cursor] = keyboard_character;
233             }
234             temp_cursor++;
235         }
236
237         temp_cursor = 0;
238         int temp_buffer_size = strlen(temp_buffer);
239         while (temp_cursor <= temp_buffer_size) //Setting the contents of the buffer equal to the
temp_buffer.
240         {
241             buffer[temp_cursor] = temp_buffer[temp_cursor];
242             temp_cursor++;
243         }
244
245         serial_print("\033[K");
246         serial_print(&keyboard_character);
247         serial_print(buffer + cursor + 1);
248         cursor++;
249     }
250     characters_in_buffer++;
251 }
252 }
253 }
254

```

```
255  *count = characters_in_buffer; // buffer count
256
257  return count;
258 }
```

#### 5.4.2.3 serial\_print()

```
int serial_print (
    const char *msg )
```

Definition at line 56 of file serial.c.

```
57 {
58     int i;
59     for (i = 0; *(i + msg) != '\0'; i++)
60     {
61         outb(serial_port_out, *(i + msg));
62     }
63     if (*msg == '\r')
64         outb(serial_port_out, '\n');
65     return NO_ERROR;
66 }
```

#### 5.4.2.4 serial\_println()

```
int serial_println (
    const char *msg )
```

Definition at line 40 of file serial.c.

```
41 {
42     int i;
43     for (i = 0; *(i + msg) != '\0'; i++)
44     {
45         outb(serial_port_out, *(i + msg));
46     }
47     outb(serial_port_out, '\r');
48     outb(serial_port_out, '\n');
49     return NO_ERROR;
50 }
```

#### 5.4.2.5 set\_serial\_in()

```
int set_serial_in (
    int device )
```

Definition at line 86 of file serial.c.

```
87 {
88     serial_port_in = device;
89     return NO_ERROR;
90 }
```

#### 5.4.2.6 set\_serial\_out()

```
int set_serial_out (
    int device )
```

Definition at line 74 of file serial.c.

```
75 {
76     serial_port_out = device;
77     return NO_ERROR;
78 }
```

## 5.5 include/core/tables.h File Reference

```
#include "system.h"
```

### Classes

- struct [idt\\_entry\\_struct](#)
- struct [idt\\_struct](#)
- struct [gdt\\_descriptor\\_struct](#)
- struct [gdt\\_entry\\_struct](#)

### Functions

- struct [idt\\_entry\\_struct](#) [\\_\\_attribute\\_\\_\(\(packed\)\) idt\\_entry](#)
- void [idt\\_set\\_gate](#) (u8int idx, u32int base, u16int sel, u8int flags)
- void [gdt\\_init\\_entry](#) (int idx, u32int base, u32int limit, u8int access, u8int flags)
- void [init\\_idt](#) ()
- void [init\\_gdt](#) ()

### Variables

- u16int [base\\_low](#)
- u16int [sselect](#)
- u8int [zero](#)
- u8int [flags](#)
- u16int [base\\_high](#)
- u16int [limit](#)
- u32int [base](#)
- u16int [limit\\_low](#)
- u8int [base\\_mid](#)
- u8int [access](#)

#### 5.5.1 Function Documentation



### 5.5.1.1 \_\_attribute\_\_()

```
struct gdt_entry_struct __attribute__ (
    (packed) )
```

### 5.5.1.2 gdt\_init\_entry()

```
void gdt_init_entry (
    int idx,
    u32int base,
    u32int limit,
    u8int access,
    u8int flags )
```

Definition at line 57 of file tables.c.

```
59 {
60     gdt_entry *new_entry = &gdt_entries[idx];
61     new_entry->base_low = (base & 0xFFFF);
62     new_entry->base_mid = (base » 16) & 0xFF;
63     new_entry->base_high = (base » 24) & 0xFF;
64     new_entry->limit_low = (limit & 0xFFFF);
65     new_entry->flags = (limit » 16) & 0xFF;
66     new_entry->flags |= flags & 0xF0;
67     new_entry->access = access;
68 }
```

### 5.5.1.3 idt\_set\_gate()

```
void idt_set_gate (
    u8int idx,
    u32int base,
    u16int sel,
    u8int flags )
```

Definition at line 27 of file tables.c.

```
29 {
30     idt_entry *new_entry = &idt_entries[idx];
31     new_entry->base_low = (base & 0xFFFF);
32     new_entry->base_high = (base » 16) & 0xFFFF;
33     new_entry->sselect = sel;
34     new_entry->zero = 0;
35     new_entry->flags = flags;
36 }
```

### 5.5.1.4 init\_gdt()

```
void init_gdt ( )
```

Definition at line 75 of file tables.c.

```
76 {
77     gdt_ptr.limit = 5 * sizeof(gdt_entry) - 1;
78     gdt_ptr.base = (u32int) gdt_entries;
79
80     u32int limit = 0xFFFFFFFF;
81     gdt_init_entry(0, 0, 0, 0, 0); //required null segment
82     gdt_init_entry(1, 0, limit, 0x9A, 0xCF); //code segment
83     gdt_init_entry(2, 0, limit, 0x92, 0xCF); //data segment
84     gdt_init_entry(3, 0, limit, 0xFA, 0xCF); //user mode code segment
85     gdt_init_entry(4, 0, limit, 0xF2, 0xCF); //user mode data segment
86
87     write_gdt_ptr((u32int) &gdt_ptr, sizeof(gdt_ptr));
88 }
```

#### 5.5.1.5 init\_idt()

```
void init_idt ( )
```

Definition at line 43 of file tables.c.

```
44 {  
45     idt_ptr.limit = 256*sizeof(idt_descriptor) - 1;  
46     idt_ptr.base  = (u32int)idt_entries;  
47     memset(idt_entries, 0, 256*sizeof(idt_descriptor));  
48  
49     write_idt_ptr((u32int)&idt_ptr);  
50 }
```

### 5.5.2 Variable Documentation

#### 5.5.2.1 access

```
u8int access
```

Definition at line 3 of file tables.h.

#### 5.5.2.2 base

```
u32int base
```

Definition at line 1 of file tables.h.

#### 5.5.2.3 base\_high

```
u8int base_high
```

Definition at line 4 of file tables.h.

#### 5.5.2.4 base\_low

```
u16int base_low
```

Definition at line 0 of file tables.h.

#### 5.5.2.5 base\_mid

```
u8int base_mid
```

Definition at line 2 of file tables.h.

#### 5.5.2.6 flags

```
u8int flags
```

Definition at line 3 of file tables.h.

#### 5.5.2.7 limit

```
u16int limit
```

Definition at line 0 of file tables.h.

#### 5.5.2.8 limit\_low

```
u16int limit_low
```

Definition at line 0 of file tables.h.

#### 5.5.2.9 sselect

```
u16int sselect
```

Definition at line 1 of file tables.h.

#### 5.5.2.10 zero

```
u8int zero
```

Definition at line 2 of file tables.h.

## 5.6 include/mem/heap.h File Reference

### Classes

- struct [header](#)
- struct [footer](#)
- struct [index\\_entry](#)
- struct [index\\_table](#)
- struct [heap](#)

### Macros

- #define [TABLE\\_SIZE](#) 0x1000
- #define [KHEAP\\_BASE](#) 0xD000000
- #define [KHEAP\\_MIN](#) 0x10000
- #define [KHEAP\\_SIZE](#) 0x1000000

### Functions

- [u32int \\_kmalloc](#) ([u32int](#) size, int align, [u32int](#)\*phys\_addr)
- [u32int kmalloc](#) ([u32int](#) size)
- [u32int kfree](#) ()
- void [init\\_kheap](#) ()
- [u32int alloc](#) ([u32int](#) size, [heap](#)\*hp, int align)
- [heap](#)\*[make\\_heap](#) ([u32int](#) base, [u32int](#) max, [u32int](#) min)

### 5.6.1 Macro Definition Documentation

#### 5.6.1.1 KHEAP\_BASE

```
#define KHEAP_BASE 0xD000000
```

Definition at line 6 of file heap.h.

#### 5.6.1.2 KHEAP\_MIN

```
#define KHEAP_MIN 0x10000
```

Definition at line 7 of file heap.h.

### 5.6.1.3 KHEAP\_SIZE

```
#define KHEAP_SIZE 0x1000000
```

Definition at line 8 of file heap.h.

### 5.6.1.4 TABLE\_SIZE

```
#define TABLE_SIZE 0x1000
```

Definition at line 5 of file heap.h.

## 5.6.2 Function Documentation

### 5.6.2.1 \_kmalloc()

```
u32int _kmalloc (
    u32int size,
    int align,
    u32int *phys_addr )
```

Definition at line 24 of file heap.c.

```
25 {
26     u32int *addr;
27
28     // Allocate on the kernel heap if one has been created
29     if (kheap != 0){
30         addr = (u32int*)alloc(size, kheap, page_align);
31         if (phys_addr){
32             page_entry *page = get_page((u32int)addr, kdir, 0);
33             *phys_addr = (page->frameaddr*0x1000) + ((u32int)addr & 0xFFF);
34         }
35         return (u32int)addr;
36     }
37     // Else, allocate directly from physical memory
38     else {
39         if (page_align && (phys_alloc_addr & 0xFFFFF000)){
40             phys_alloc_addr &= 0xFFFFF000;
41             phys_alloc_addr += 0x1000;
42         }
43         addr = (u32int*)phys_alloc_addr;
44         if (phys_addr){
45             *phys_addr = phys_alloc_addr;
46         }
47         phys_alloc_addr += size;
48         return (u32int)addr;
49     }
50 }
```

### 5.6.2.2 alloc()

```
u32int alloc (
    u32int size,
    heap * hp,
    int align )
```

Definition at line 57 of file heap.c.

```
58 {
59     no_warn(size || align || h);
60     static u32int heap_addr = KHEAP_BASE;
61
62     u32int base = heap_addr;
63     heap_addr += size;
64
65     if (heap_addr > KHEAP_BASE + KHEAP_MIN)
66         serial_println("Heap is full!");
67
68     return base;
69 }
```

### 5.6.2.3 init\_kheap()

```
void init_kheap ( )
```

### 5.6.2.4 kfree()

```
u32int kfree ( )
```

### 5.6.2.5 kmalloc()

```
u32int kmalloc (
    u32int size )
```

Definition at line 52 of file heap.c.

```
53 {
54     return _kmalloc(size, 0, 0);
55 }
```

### 5.6.2.6 make\_heap()

```
heap* make_heap (
    u32int base,
    u32int max,
    u32int min )
```

Definition at line 71 of file heap.c.

```
72 {
73     no_warn(base || max || min);
74     return (heap*) kmalloc(sizeof(heap));
75 }
```

## 5.7 include/mem/paging.h File Reference

```
#include <system.h>
```

### Classes

- struct [page\\_entry](#)
- struct [page\\_table](#)
- struct [page\\_dir](#)

### Macros

- #define [PAGE\\_SIZE](#) 0x1000

### Functions

- void [set\\_bit](#) (u32int addr)
- void [clear\\_bit](#) (u32int addr)
- u32int [get\\_bit](#) (u32int addr)
- u32int [first\\_free](#) ()
- void [init\\_paging](#) ()
- void [load\\_page\\_dir](#) ([page\\_dir](#) \*new\_page\_dir)
- [page\\_entry](#) \*[get\\_page](#) (u32int addr, [page\\_dir](#) \*dir, int make\_table)
- void [new\\_frame](#) ([page\\_entry](#) \*page)

### 5.7.1 Macro Definition Documentation

#### 5.7.1.1 PAGE\_SIZE

```
#define PAGE_SIZE 0x1000
```

Definition at line 6 of file paging.h.

### 5.7.2 Function Documentation

### 5.7.2.1 clear\_bit()

```
void clear_bit (
    u32int addr )
```

Definition at line 44 of file paging.c.

```
45 {
46     u32int frame = addr/page_size;
47     u32int index = frame/32;
48     u32int offset = frame%32;
49     frames[index] &= ~(1 « offset);
50 }
```

### 5.7.2.2 first\_free()

```
u32int first_free ( )
```

### 5.7.2.3 get\_bit()

```
u32int get_bit (
    u32int addr )
```

Definition at line 56 of file paging.c.

```
57 {
58     u32int frame = addr/page_size;
59     u32int index = frame/32;
60     u32int offset = frame%32;
61     return (frames[index] & (1 « offset));
62 }
```

### 5.7.2.4 get\_page()

```
page_entry* get_page (
    u32int addr,
    page_dir * dir,
    int make_table )
```

Definition at line 85 of file paging.c.

```
86 {
87     u32int phys_addr;
88     u32int index = addr / page_size / 1024;
89     u32int offset = addr / page_size % 1024;
90
91     //return it if it exists
92     if (dir->tables[index])
93         return &dir->tables[index]->pages[offset];
94
95     //create it
96     else if (make_table){
97         dir->tables[index] = (page_table*)_kmallocc(sizeof(page_table), 1, &phys_addr);
98         dir->tables_phys[index] = phys_addr | 0x7; //enable present, writable
99         return &dir->tables[index]->pages[offset];
100     }
101     else return 0;
102 }
```



### 5.7.2.5 init\_paging()

```
void init_paging ( )
```

Definition at line 111 of file paging.c.

```
112 {
113     //create frame bitmap
114     nframes = (u32int) (mem_size/page_size);
115     frames = (u32int*) kmalloc(nframes/32);
116     memset(frames, 0, nframes/32);
117
118     //create kernel directory
119     kdir = (page_dir*) _kmalloc(sizeof(page_dir), 1, 0); //page aligned
120     memset(kdir, 0, sizeof(page_dir));
121
122     //get pages for kernel heap
123     u32int i = 0x0;
124     for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=1){
125         get_page(i, kdir, 1);
126     }
127
128     //perform identity mapping of used memory
129     //note: placement_addr gets incremented in get_page,
130     //so we're mapping the first frames as well
131     i = 0x0;
132     while (i < (phys_alloc_addr+0x10000)){
133         new_frame(get_page(i, kdir, 1));
134         i += page_size;
135     }
136
137     //allocate heap frames now that the placement addr has increased.
138     //placement addr increases here for heap
139     for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=PAGE_SIZE){
140         new_frame(get_page(i, kdir, 1));
141     }
142
143     //load the kernel page directory; enable paging
144     load_page_dir(kdir);
145
146     //setup the kernel heap
147     kheap = make_heap(KHEAP_BASE, KHEAP_SIZE, KHEAP_BASE+KHEAP_MIN);
148 }
```

### 5.7.2.6 load\_page\_dir()

```
void load_page_dir (
    page_dir * new_page_dir )
```

Definition at line 158 of file paging.c.

```
159 {
160     cdir = new_dir;
161     asm volatile ("mov %0,%%cr3:: "b"(&cdir->tables_phys[0]));
162     u32int cr0;
163     asm volatile ("mov %%cr0,%0: "=b"(cr0));
164     cr0 |= 0x80000000;
165     asm volatile ("mov %0,%%cr0:: "b"(cr0));
166 }
```

### 5.7.2.7 new\_frame()

```
void new_frame (
    page_entry * page )
```

Definition at line 173 of file paging.c.

```
174 {
175     u32int index;
```

```

176  if (page->frameaddr != 0) return;
177  if ( (u32int)(-1) == (index=find_free()) ) kpanic("Out of memory");
178
179  //mark a frame as in-use
180  set_bit(index*page_size);
181  page->present = 1;
182  page->frameaddr = index;
183  page->writeable = 1;
184  page->usermode = 0;
185 }

```

### 5.7.2.8 set\_bit()

```

void set_bit (
    u32int addr )

```

Definition at line 32 of file paging.c.

```

33 {
34     u32int frame = addr/page_size;
35     u32int index = frame/32;
36     u32int offset = frame%32;
37     frames[index] |= (1 << offset);
38 }

```

## 5.8 include/string.h File Reference

```
#include <system.h>
```

### Functions

- int [isspace](#) (const char \*c)
- void \*[memset](#) (void \*, int c, [size\\_t](#) n)
- char \*[strcpy](#) (char \*s1, const char \*s2)
- char \*[strcat](#) (char \*s1, const char \*s2)
- int [strlen](#) (const char \*s)
- int [strcmp](#) (const char \*s1, const char \*s2)
- char \*[strtok](#) (char \*s1, const char \*s2)
- int [atoi](#) (const char \*s)

### 5.8.1 Function Documentation

### 5.8.1.1 atoi()

```
int atoi (
    const char * s )
```

Definition at line 48 of file string.c.

```
49 {
50     int res=0;
51     int charVal=0;
52     char sign = ' ';
53     char c = *s;
54
55
56     while(isspace(&c)){ ++s; c = *s;} // advance past whitespace
57
58
59     if (*s == '-' || *s == '+') sign = *(s++); // save the sign
60
61
62     while(*s != '\0'){
63         charVal = *s - 48;
64         res = res * 10 + charVal;
65         s++;
66     }
67
68
69     if ( sign == '-') res=res * -1;
70
71
72     return res; // return integer
73 }
```

### 5.8.1.2 isspace()

```
int isspace (
    const char * c )
```

Definition at line 119 of file string.c.

```
120 {
121     if (*c == ' ' ||
122         *c == '\n' ||
123         *c == '\r' ||
124         *c == '\f' ||
125         *c == '\t' ||
126         *c == '\v'){
127         return 1;
128     }
129     return 0;
130 }
```

### 5.8.1.3 memset()

```
void* memset (
    void * s,
    int c,
    size_t n )
```

Definition at line 137 of file string.c.

```
138 {
139     unsigned char *p = (unsigned char *) s;
140     while(n--){
141         *p++ = (unsigned char) c;
142     }
143     return s;
144 }
```

#### 5.8.1.4 strcat()

```
char* strcat (
    char * s1,
    const char * s2 )
```

Definition at line 106 of file string.c.

```
107 {
108     char *rc = s1;
109     if (*s1) while(++s1);
110     while( (*s1++ = *s2++) );
111     return rc;
112 }
```

#### 5.8.1.5 strcmp()

```
int strcmp (
    const char * s1,
    const char * s2 )
```

Definition at line 79 of file string.c.

```
80 {
81
82     // Remarks:
83     // 1) If we made it to the end of both strings (i. e. our pointer points to a
84     // '\0' character), the function will return 0
85     // 2) If we didn't make it to the end of both strings, the function will
86     // return the difference of the characters at the first index of
87     // indifference.
88     while ( (*s1) && (*s1==*s2) ){
89         ++s1;
90         ++s2;
91     }
92     return ( *(unsigned char *)s1 - *(unsigned char *)s2 );
93 }
```

#### 5.8.1.6 strcpy()

```
char* strcpy (
    char * s1,
    const char * s2 )
```

Definition at line 36 of file string.c.

```
37 {
38     char *rc = s1;
39     while( (*s1++ = *s2++) );
40     return rc; // return pointer to destination string
41 }
```

#### 5.8.1.7 strlen()

```
int strlen (
    const char * s )
```

Definition at line 24 of file string.c.

```
25 {
26     int r1 = 0;
27     if (*s) while(*s++) r1++;
28     return r1; //return length of string
29 }
```

### 5.8.1.8 strtok()

```
char* strtok (
    char * s1,
    const char * s2 )
```

Definition at line 151 of file string.c.

```
152 {
153     static char *tok_tmp = NULL;
154     const char *p = s2;
155
156     //new string
157     if (s1!=NULL){
158         tok_tmp = s1;
159     }
160     //old string cont'd
161     else {
162         if (tok_tmp==NULL){
163             return NULL;
164         }
165         s1 = tok_tmp;
166     }
167
168     //skip leading s2 characters
169     while ( *p && *s1 ){
170         if (*s1==*p){
171             ++s1;
172             p = s2;
173             continue;
174         }
175         ++p;
176     }
177
178     //no more to parse
179     if (!*s1){
180         return (tok_tmp = NULL);
181     }
182
183     //skip non-s2 characters
184     tok_tmp = s1;
185     while (*tok_tmp){
186         p = s2;
187         while (*p){
188             if (*tok_tmp==*p++){
189                 *tok_tmp++ = '\0';
190                 return s1;
191             }
192         }
193         ++tok_tmp;
194     }
195
196     //end of string
197     tok_tmp = NULL;
198     return s1;
199 }
```

## 5.9 include/system.h File Reference

### Classes

- struct [date\\_time](#)

### Macros

- #define [NULL](#) 0
- #define [no\\_warn](#)(p) if (p) while (1) break
- #define [asm](#) \_\_asm\_\_
- #define [volatile](#) \_\_volatile\_\_
- #define [sti](#)() [asm volatile](#) ("sti::")

- `#define cli() asm volatile ("cli::")`
- `#define nop() asm volatile ("nop::")`
- `#define hlt() asm volatile ("hlt::")`
- `#define iret() asm volatile ("iret::")`
- `#define GDT_CS_ID 0x01`
- `#define GDT_DS_ID 0x02`

## Typedefs

- `typedef unsigned int size_t`
- `typedef unsigned char u8int`
- `typedef unsigned short u16int`
- `typedef unsigned long u32int`

## Functions

- `static int irq_on ()`
- `void klogv (const char *msg)`
- `void kpanic (const char *msg)`

## 5.9.1 Macro Definition Documentation

### 5.9.1.1 asm

```
#define asm __asm__
```

Definition at line 11 of file system.h.

### 5.9.1.2 cli

```
#define cli( ) asm volatile ("cli::")
```

Definition at line 15 of file system.h.

### 5.9.1.3 GDT\_CS\_ID

```
#define GDT_CS_ID 0x01
```

Definition at line 20 of file system.h.

#### 5.9.1.4 GDT\_DS\_ID

```
#define GDT_DS_ID 0x02
```

Definition at line 21 of file system.h.

#### 5.9.1.5 hlt

```
#define hlt( ) asm volatile ("hlt:::")
```

Definition at line 17 of file system.h.

#### 5.9.1.6 iret

```
#define iret( ) asm volatile ("iret:::")
```

Definition at line 18 of file system.h.

#### 5.9.1.7 no\_warn

```
#define no_warn(  
    p ) if (p) while (1) break
```

Definition at line 7 of file system.h.

#### 5.9.1.8 nop

```
#define nop( ) asm volatile ("nop:::")
```

Definition at line 16 of file system.h.

#### 5.9.1.9 NULL

```
#define NULL 0
```

Definition at line 4 of file system.h.

#### 5.9.1.10 sti

```
#define sti( ) asm volatile ("sti::")
```

Definition at line 14 of file system.h.

#### 5.9.1.11 volatile

```
#define volatile __volatile__
```

Definition at line 12 of file system.h.

### 5.9.2 Typedef Documentation

#### 5.9.2.1 size\_t

```
typedef unsigned int size_t
```

Definition at line 24 of file system.h.

#### 5.9.2.2 u16int

```
typedef unsigned short u16int
```

Definition at line 26 of file system.h.

#### 5.9.2.3 u32int

```
typedef unsigned long u32int
```

Definition at line 27 of file system.h.

#### 5.9.2.4 u8int

```
typedef unsigned char u8int
```

Definition at line 25 of file system.h.



### 5.9.3 Function Documentation

#### 5.9.3.1 irq\_on()

```
static int irq_on ( ) [inline], [static]
```

Definition at line 42 of file system.h.

```
43 {
44     int f;
45     asm volatile ("pushf\n\t"
46                 "popl %0"
47                 : "=g" (f));
48     return f & (1 << 9);
49 }
```

#### 5.9.3.2 klogv()

```
void klogv (
            const char *msg )
```

Definition at line 11 of file system.c.

```
12 {
13     char logmsg[64] = {'\0'}, prefix[] = "klogv: ";
14     strcat(logmsg, prefix);
15     strcat(logmsg, msg);
16     serial_println(logmsg);
17 }
```

#### 5.9.3.3 kpanic()

```
void kpanic (
            const char *msg )
```

Definition at line 24 of file system.c.

```
25 {
26     cli(); //disable interrupts
27     char logmsg[64] = {'\0'}, prefix[] = "Panic: ";
28     strcat(logmsg, prefix);
29     strcat(logmsg, msg);
30     klogv(logmsg);
31     hlt(); //halt
32 }
```

## 5.10 kernel/core/interrupts.c File Reference

```
#include <system.h>
#include <core/io.h>
#include <core/serial.h>
#include <core/tables.h>
#include <core/interrupts.h>
```

## Macros

- `#define PIC1 0x20`
- `#define PIC2 0xA0`
- `#define ICW1 0x11`
- `#define ICW4 0x01`
- `#define io_wait() asm volatile ("outb $0x80")`

## Functions

- void `divide_error` ()
- void `debug` ()
- void `nmi` ()
- void `breakpoint` ()
- void `overflow` ()
- void `bounds` ()
- void `invalid_op` ()
- void `device_not_available` ()
- void `double_fault` ()
- void `coprocessor_segment` ()
- void `invalid_tss` ()
- void `segment_not_present` ()
- void `stack_segment` ()
- void `general_protection` ()
- void `page_fault` ()
- void `reserved` ()
- void `coprocessor` ()
- void `rtc_isr` ()
- void `isr0` ()
- void `do_isr` ()
- void `init_irq` (void)
- void `init_pic` (void)
- void `do_divide_error` ()
- void `do_debug` ()
- void `do_nmi` ()
- void `do_breakpoint` ()
- void `do_overflow` ()
- void `do_bounds` ()
- void `do_invalid_op` ()
- void `do_device_not_available` ()
- void `do_double_fault` ()
- void `do_coprocessor_segment` ()
- void `do_invalid_tss` ()
- void `do_segment_not_present` ()
- void `do_stack_segment` ()
- void `do_general_protection` ()
- void `do_page_fault` ()
- void `do_reserved` ()
- void `do_coprocessor` ()

## Variables

- idt\_entry `idt_entries` [256]

## 5.10.1 Macro Definition Documentation

### 5.10.1.1 ICW1

```
#define ICW1 0x11
```

Definition at line 20 of file interrupts.c.

### 5.10.1.2 ICW4

```
#define ICW4 0x01
```

Definition at line 21 of file interrupts.c.

### 5.10.1.3 io\_wait

```
#define io_wait( ) asm volatile ("outb $0x80")
```

Definition at line 28 of file interrupts.c.

### 5.10.1.4 PIC1

```
#define PIC1 0x20
```

Definition at line 16 of file interrupts.c.

### 5.10.1.5 PIC2

```
#define PIC2 0xA0
```

Definition at line 17 of file interrupts.c.

## 5.10.2 Function Documentation

**5.10.2.1 bounds()**

```
void bounds ( )
```

**5.10.2.2 breakpoint()**

```
void breakpoint ( )
```

**5.10.2.3 coprocessor()**

```
void coprocessor ( )
```

**5.10.2.4 coprocessor\_segment()**

```
void coprocessor_segment ( )
```

**5.10.2.5 debug()**

```
void debug ( )
```

**5.10.2.6 device\_not\_available()**

```
void device_not_available ( )
```

**5.10.2.7 divide\_error()**

```
void divide_error ( )
```

### 5.10.2.8 do\_bounds()

```
void do_bounds ( )
```

Definition at line 149 of file interrupts.c.

```
150 {  
151     kpanic("Bounds error");  
152 }
```

### 5.10.2.9 do\_breakpoint()

```
void do_breakpoint ( )
```

Definition at line 141 of file interrupts.c.

```
142 {  
143     kpanic("Breakpoint");  
144 }
```

### 5.10.2.10 do\_coprocessor()

```
void do_coprocessor ( )
```

Definition at line 193 of file interrupts.c.

```
194 {  
195     kpanic("Coprocessor error");  
196 }
```

### 5.10.2.11 do\_coprocessor\_segment()

```
void do_coprocessor_segment ( )
```

Definition at line 165 of file interrupts.c.

```
166 {  
167     kpanic("Coprocessor segment error");  
168 }
```

### 5.10.2.12 do\_debug()

```
void do_debug ( )
```

Definition at line 133 of file interrupts.c.

```
134 {  
135     kpanic("Debug");  
136 }
```

#### 5.10.2.13 do\_device\_not\_available()

```
void do_device_not_available ( )
```

Definition at line 157 of file interrupts.c.

```
158 {  
159     kpanic("Device not available");  
160 }
```

#### 5.10.2.14 do\_divide\_error()

```
void do_divide_error ( )
```

Definition at line 129 of file interrupts.c.

```
130 {  
131     kpanic("Division-by-zero");  
132 }
```

#### 5.10.2.15 do\_double\_fault()

```
void do_double_fault ( )
```

Definition at line 161 of file interrupts.c.

```
162 {  
163     kpanic("Double fault");  
164 }
```

#### 5.10.2.16 do\_general\_protection()

```
void do_general_protection ( )
```

Definition at line 181 of file interrupts.c.

```
182 {  
183     kpanic("General protection fault");  
184 }
```

#### 5.10.2.17 do\_invalid\_op()

```
void do_invalid_op ( )
```

Definition at line 153 of file interrupts.c.

```
154 {  
155     kpanic("Invalid operation");  
156 }
```

#### 5.10.2.18 do\_invalid\_tss()

```
void do_invalid_tss ( )
```

Definition at line 169 of file interrupts.c.

```
170 {  
171     kpanic("Invalid TSS");  
172 }
```

#### 5.10.2.19 do\_isr()

```
void do_isr ( )
```

Definition at line 53 of file interrupts.c.

```
54 {  
55     char in = inb(COM2);  
56     serial_print(&in);  
57     serial_println("here");  
58     outb(0x20,0x20); //EOI  
59 }
```

#### 5.10.2.20 do\_nmi()

```
void do_nmi ( )
```

Definition at line 137 of file interrupts.c.

```
138 {  
139     kpanic("NMI");  
140 }
```

#### 5.10.2.21 do\_overflow()

```
void do_overflow ( )
```

Definition at line 145 of file interrupts.c.

```
146 {  
147     kpanic("Overflow error");  
148 }
```

#### 5.10.2.22 do\_page\_fault()

```
void do_page_fault ( )
```

Definition at line 185 of file interrupts.c.

```
186 {  
187     kpanic("Page Fault");  
188 }
```

#### 5.10.2.23 do\_reserved()

```
void do_reserved ( )
```

Definition at line 189 of file interrupts.c.

```
190 {  
191     serial_println("die: reserved");  
192 }
```

#### 5.10.2.24 do\_segment\_not\_present()

```
void do_segment_not_present ( )
```

Definition at line 173 of file interrupts.c.

```
174 {  
175     kpanic("Segment not present");  
176 }
```

#### 5.10.2.25 do\_stack\_segment()

```
void do_stack_segment ( )
```

Definition at line 177 of file interrupts.c.

```
178 {  
179     kpanic("Stack segment error");  
180 }
```

#### 5.10.2.26 double\_fault()

```
void double_fault ( )
```

#### 5.10.2.27 general\_protection()

```
void general_protection ( )
```



### 5.10.2.28 init\_irq()

```
void init_irq (
    void )
```

Definition at line 66 of file interrupts.c.

```
67 {
68     int i;
69
70     // Necessary interrupt handlers for protected mode
71     u32int isrs[17] = {
72         (u32int)divide_error,
73         (u32int)debug,
74         (u32int)nmi,
75         (u32int)breakpoint,
76         (u32int)overflow,
77         (u32int)bounds,
78         (u32int)invalid_op,
79         (u32int)device_not_available,
80         (u32int)double_fault,
81         (u32int)coprocessor_segment,
82         (u32int)invalid_tss,
83         (u32int)segment_not_present,
84         (u32int)stack_segment,
85         (u32int)general_protection,
86         (u32int)page_fault,
87         (u32int)reserved,
88         (u32int)coprocessor
89     };
90
91     // Install handlers; 0x08=sel, 0x8e=flags
92     for(i=0; i<32; i++){
93         if (i<17) idt_set_gate(i, isrs[i], 0x08, 0x8e);
94         else idt_set_gate(i, (u32int)reserved, 0x08, 0x8e);
95     }
96     // Ignore interrupts from the real time clock
97     idt_set_gate(0x08, (u32int)rtc_isr, 0x08, 0x8e);
98 }
```

### 5.10.2.29 init\_pic()

```
void init_pic (
    void )
```

Definition at line 106 of file interrupts.c.

```
107 {
108     outb(PIC1,ICW1);    //send initialization code words 1 to PIC1
109     io_wait();
110     outb(PIC2,ICW1);    //send icw1 to PIC2
111     io_wait();
112     outb(PIC1+1,0x20); //icw2: remap irq0 to 32
113     io_wait();
114     outb(PIC2+1,0x28); //icw2: remap irq8 to 40
115     io_wait();
116     outb(PIC1+1,4);     //icw3
117     io_wait();
118     outb(PIC2+1,2);     //icw3
119     io_wait();
120     outb(PIC1+1,ICW4); //icw4: 80x86, automatic handling
121     io_wait();
122     outb(PIC2+1,ICW4); //icw4: 80x86, automatic handling
123     io_wait();
124     outb(PIC1+1,0xFF); //disable irqs for PIC1
125     io_wait();
126     outb(PIC2+1,0xFF); //disable irqs for PIC2
127 }
```

**5.10.2.30 invalid\_op()**

```
void invalid_op ( )
```

**5.10.2.31 invalid\_tss()**

```
void invalid_tss ( )
```

**5.10.2.32 isr0()**

```
void isr0 ( )
```

**5.10.2.33 nmi()**

```
void nmi ( )
```

**5.10.2.34 overflow()**

```
void overflow ( )
```

**5.10.2.35 page\_fault()**

```
void page_fault ( )
```

**5.10.2.36 reserved()**

```
void reserved ( )
```

**5.10.2.37 rtc\_isr()**

```
void rtc_isr ( )
```

### 5.10.2.38 segment\_not\_present()

```
void segment_not_present ( )
```

### 5.10.2.39 stack\_segment()

```
void stack_segment ( )
```

## 5.10.3 Variable Documentation

### 5.10.3.1 idt\_entries

```
idt_entry idt_entries[256] [extern]
```

Definition at line 17 of file tables.c.

## 5.11 kernel/core/kmain.c File Reference

```
#include <stdint.h>
#include <string.h>
#include <system.h>
#include <core/io.h>
#include <core/serial.h>
#include <core/tables.h>
#include <core/interrupts.h>
#include <mem/heap.h>
#include <mem/paging.h>
#include "modules/mpx_supt.h"
#include "modules/R1/commhand.h"
```

## Functions

- void [kmain](#) (void)

### 5.11.1 Function Documentation

### 5.11.1.1 kmain()

```
void kmain (
    void )
```

Definition at line 27 of file kmain.c.

```
28 {
29     // extern uint32_t magic;
30     // Uncomment if you want to access the multiboot header
31     // extern void *mbd;
32     // char *boot_loader_name = (char*)((long*)mbd)[16];
33
34     // 0) Initialize Serial I/O
35     // functions to initialize serial I/O can be found in serial.c
36     // there are 3 functions to call
37
38     init_serial(COM1);
39     set_serial_in(COM1);
40     set_serial_out(COM1);
41
42     klogv("Starting MPX boot sequence...");
43     klogv("Initialized serial I/O on COM1 device...");
44
45     // 1) Initialize the support software by identifying the current
46     //     MPX Module. This will change with each module.
47     // you will need to call mpx_init from the mpx_supt.c
48
49     mpx_init(MODULE_R2);
50
51     // 2) Check that the boot was successful and correct when using grub
52     // Comment this when booting the kernel directly using QEMU, etc.
53     //if ( magic != 0x2BADB002 ){
54     //     kpanic("Boot was not error free. Halting.");
55     //}
56
57     // 3) Descriptor Tables -- tables.c
58     // you will need to initialize the global
59     // this keeps track of allocated segments and pages
60     klogv("Initializing descriptor tables...");
61
62     init_gdt();
63     init_idt();
64
65     init_pic();
66     sti();
67
68     // 4) Interrupt vector table -- tables.c
69     // this creates and initializes a default interrupt vector table
70     // this function is in tables.c
71
72     init_irq();
73
74     klogv("Interrupt vector table initialized!");
75
76     // 5) Virtual Memory -- paging.c -- init_paging
77     // this function creates the kernel's heap
78     // from which memory will be allocated when the program calls
79     // sys_alloc_mem UNTIL the memory management module is completed
80     // this allocates memory using discrete "pages" of physical memory
81     // NOTE: You will only have about 70000 bytes of dynamic memory
82     //
83     klogv("Initializing virtual memory...");
84
85     init_paging();
86
87     // 6) Call YOUR command handler - interface method
88     klogv("Transferring control to commhand...");
89     commhand();
90
91     // 7) System Shutdown on return from your command handler
92
93     klogv("Starting system shutdown procedure...");
94
95     /* Shutdown Procedure */
96     klogv("Shutdown complete. You may now turn off the machine. (QEMU: C-a x)");
97     hlt();
98 }
```

## 5.12 kernel/core/serial.c File Reference

```
#include <stdint.h>
```

```
#include <string.h>
#include <core/io.h>
#include <core/serial.h>
```

## Macros

- #define `NO_ERROR` 0

## Functions

- int `init_serial` (int device)
- int `serial_println` (const char \*msg)
- int `serial_print` (const char \*msg)
- int `set_serial_out` (int device)
- int `set_serial_in` (int device)
- int \*`polling` (char \*buffer, int \*count)

## Variables

- int `serial_port_out` = 0
- int `serial_port_in` = 0

## 5.12.1 Macro Definition Documentation

### 5.12.1.1 NO\_ERROR

```
#define NO_ERROR 0
```

Definition at line 12 of file serial.c.

## 5.12.2 Function Documentation

### 5.12.2.1 init\_serial()

```
int init_serial (
    int device )
```

Definition at line 22 of file serial.c.

```
23 {
24     outb(device + 1, 0x00);           //disable interrupts
25     outb(device + 3, 0x80);           //set line control register
26     outb(device + 0, 115200 / 9600); //set bsd least sig bit
27     outb(device + 1, 0x00);           //brd most significant bit
28     outb(device + 3, 0x03);           //lock divisor; 8bits, no parity, one stop
29     outb(device + 2, 0xC7);           //enable fifo, clear, 14byte threshold
30     outb(device + 4, 0x0B);           //enable interrupts, rts/dsr set
31     (void)inb(device);                //read bit to reset port
32     return NO_ERROR;
33 }
```

### 5.12.2.2 polling()

```
int* polling (
    char * buffer,
    int * count )
```

Definition at line 92 of file serial.c.

```
93 {
94     // insert your code to gather keyboard input via the technique of polling.
95
96     char keyboard_character;
97
98     int cursor = 0;
99
100    char log[] = {'\0', '\0', '\0', '\0'};
101
102    int characters_in_buffer = 0;
103
104    while (1)
105    {
106
107        if (inb(COM1 + 5) & 1)
108        {
109            // is there input char?
110            keyboard_character = inb(COM1); //read the char from COM1
111
112            if (keyboard_character == '\n' || keyboard_character == '\r')
113            { // HANDLING THE CARRIAGE RETURN AND NEW LINE CHARACTERS
114
115                buffer[characters_in_buffer] = ' \0';
116                break;
117            }
118            else if ((keyboard_character == 127 || keyboard_character == 8) && cursor > 0)
119            { // HANDLING THE BACKSPACE CHARACTER
120
121                //serial_println("Handleing backspace character.");
122                serial_print("\033[K");
123
124                buffer[cursor - 1] = ' \0';
125                serial_print("\b\b");
126                serial_print(buffer + cursor);
127                cursor--;
128
129                int temp_cursor = cursor;
130
131                while (buffer[temp_cursor + 1] != ' \0')
132                {
133                    buffer[temp_cursor] = buffer[temp_cursor + 1];
134                    buffer[temp_cursor + 1] = ' \0';
135                    temp_cursor++;
136                }
137
138                characters_in_buffer--;
139                cursor = characters_in_buffer;
140            }
141            else if (keyboard_character == '~' && cursor < 99)
142            { //HANDLING THE DELETE KEY
143                // \033[3~
144
145                serial_print("\033[K");
146
147                buffer[cursor + 1] = ' \0';
148                serial_print("\b\b");
149                serial_print(buffer + cursor);
150
151                int temp_cursor = cursor + 1;
152
153                while (buffer[temp_cursor + 1] != ' \0')
154                {
155                    buffer[temp_cursor] = buffer[temp_cursor + 1];
156                    buffer[temp_cursor + 1] = ' \0';
157                    temp_cursor++;
158                }
159
160                characters_in_buffer--;
161                cursor = characters_in_buffer;
162            }
163            else if (keyboard_character == '\033')
164            { // HANDLING FIRST CHARACTER FOR ARROW KEYS
165
166                log[0] = keyboard_character;
167            }
168            else if (keyboard_character == '[' && log[0] == '\033')
169            { // HANDLING SECOND CHARACTER FOR ARROW KEYS
```

```

170     log[1] = keyboard_character;
171 }
172 else if (log[0] == '\033' && log[1] == '[')
173 { // HANDLING LAST CHARACTER FOR ARROW KEYS
174     log[2] = keyboard_character;
175
176     if (keyboard_character == 'A')
177     { //Up arrow
178         //Call a history function from the commhand or do nothing
179     }
180     else if (keyboard_character == 'B')
181     { //Down arrow
182         //Call a history command from the commhand or do nothing
183     }
184     else if (keyboard_character == 'C' && cursor != 99)
185     { //Right arrow
186
187         serial_print("\033[C");
188         cursor++;
189     }
190     else if (keyboard_character == 'D' && cursor != 0)
191     { //Left arrow
192
193         serial_print("\033[D");
194         cursor--;
195     }
196
197     memset(log, '\0', 4);
198 }
199 else
200 {
201
202     if (cursor == 0 && buffer[cursor] == '\0') //Adding character at beginning of buffer
203     {
204         buffer[cursor] = keyboard_character;
205         serial_print(&keyboard_character);
206         cursor++;
207     }
208     else if (buffer[cursor] == '\0') //Adding character at the end of the buffer
209     {
210         buffer[cursor] = keyboard_character;
211         serial_print(&keyboard_character);
212         cursor++;
213     }
214     else //Inserting character to the middle of the buffer
215     {
216         char temp_buffer[strlen(buffer)];
217         memset(temp_buffer, '\0', strlen(buffer));
218
219         int temp_cursor = 0;
220         while (temp_cursor <= characters_in_buffer) //Filling the temp_buffer with all of the
characters from buffer, and inserting the new character.
221         {
222             if (temp_cursor < cursor)
223             {
224                 temp_buffer[temp_cursor] = buffer[temp_cursor];
225             }
226             else if (temp_cursor > cursor)
227             {
228                 temp_buffer[temp_cursor] = buffer[temp_cursor - 1];
229             }
230             else
231             { //temp_cursor == cursor
232                 temp_buffer[temp_cursor] = keyboard_character;
233             }
234             temp_cursor++;
235         }
236
237         temp_cursor = 0;
238         int temp_buffer_size = strlen(temp_buffer);
239         while (temp_cursor <= temp_buffer_size) //Setting the contents of the buffer equal to the
temp_buffer.
240         {
241             buffer[temp_cursor] = temp_buffer[temp_cursor];
242             temp_cursor++;
243         }
244
245         serial_print("\033[K");
246         serial_print(&keyboard_character);
247         serial_print(buffer + cursor + 1);
248         cursor++;
249     }
250     characters_in_buffer++;
251 }
252 }
253 }
254

```

```
255  *count = characters_in_buffer; // buffer count
256
257  return count;
258 }
```

### 5.12.2.3 serial\_print()

```
int serial_print (
    const char *msg )
```

Definition at line 56 of file serial.c.

```
57 {
58     int i;
59     for (i = 0; *(i + msg) != '\0'; i++)
60     {
61         outb(serial_port_out, *(i + msg));
62     }
63     if (*msg == '\r')
64         outb(serial_port_out, '\n');
65     return NO_ERROR;
66 }
```

### 5.12.2.4 serial\_println()

```
int serial_println (
    const char *msg )
```

Definition at line 40 of file serial.c.

```
41 {
42     int i;
43     for (i = 0; *(i + msg) != '\0'; i++)
44     {
45         outb(serial_port_out, *(i + msg));
46     }
47     outb(serial_port_out, '\r');
48     outb(serial_port_out, '\n');
49     return NO_ERROR;
50 }
```

### 5.12.2.5 set\_serial\_in()

```
int set_serial_in (
    int device )
```

Definition at line 86 of file serial.c.

```
87 {
88     serial_port_in = device;
89     return NO_ERROR;
90 }
```



#### 5.12.2.6 set\_serial\_out()

```
int set_serial_out (
    int device )
```

Definition at line 74 of file serial.c.

```
75 {
76     serial_port_out = device;
77     return NO_ERROR;
78 }
```

### 5.12.3 Variable Documentation

#### 5.12.3.1 serial\_port\_in

```
int serial_port_in = 0
```

Definition at line 16 of file serial.c.

#### 5.12.3.2 serial\_port\_out

```
int serial_port_out = 0
```

Definition at line 15 of file serial.c.

## 5.13 kernel/core/system.c File Reference

```
#include <string.h>
#include <system.h>
#include <core/serial.h>
```

### Functions

- void [klogv](#) (const char \*msg)
- void [kpanic](#) (const char \*msg)

#### 5.13.1 Function Documentation

### 5.13.1.1 klogv()

```
void klogv (
    const char *msg )
```

Definition at line 11 of file system.c.

```
12 {
13     char logmsg[64] = {'\0'}, prefix[] = "klogv: ";
14     strcat(logmsg, prefix);
15     strcat(logmsg, msg);
16     serial_println(logmsg);
17 }
```

### 5.13.1.2 kpanic()

```
void kpanic (
    const char *msg )
```

Definition at line 24 of file system.c.

```
25 {
26     cli(); //disable interrupts
27     char logmsg[64] = {'\0'}, prefix[] = "Panic: ";
28     strcat(logmsg, prefix);
29     strcat(logmsg, msg);
30     klogv(logmsg);
31     hlt(); //halt
32 }
```

## 5.14 kernel/core/tables.c File Reference

```
#include <string.h>
#include <core/tables.h>
```

### Functions

- void [write\\_gdt\\_ptr](#) (u32int, size\_t)
- void [write\\_idt\\_ptr](#) (u32int)
- void [idt\\_set\\_gate](#) (u8int idx, u32int base, u16int sel, u8int flags)
- void [init\\_idt](#) ()
- void [gdt\\_init\\_entry](#) (int idx, u32int base, u32int limit, u8int access, u8int flags)
- void [init\\_gdt](#) ()

### Variables

- gdt\_descriptor [gdt\\_ptr](#)
- gdt\_entry [gdt\\_entries](#) [5]
- idt\_descriptor [idt\\_ptr](#)
- idt\_entry [idt\\_entries](#) [256]

## 5.14.1 Function Documentation

### 5.14.1.1 gdt\_init\_entry()

```
void gdt_init_entry (
    int idx,
    u32int base,
    u32int limit,
    u8int access,
    u8int flags )
```

Definition at line 57 of file tables.c.

```
59 {
60     gdt_entry *new_entry = &gdt_entries[idx];
61     new_entry->base_low = (base & 0xFFFF);
62     new_entry->base_mid = (base >> 16) & 0xFF;
63     new_entry->base_high = (base >> 24) & 0xFF;
64     new_entry->limit_low = (limit & 0xFFFF);
65     new_entry->flags = (limit >> 16) & 0xFF;
66     new_entry->flags |= flags & 0xF0;
67     new_entry->access = access;
68 }
```

### 5.14.1.2 idt\_set\_gate()

```
void idt_set_gate (
    u8int idx,
    u32int base,
    u16int sel,
    u8int flags )
```

Definition at line 27 of file tables.c.

```
29 {
30     idt_entry *new_entry = &idt_entries[idx];
31     new_entry->base_low = (base & 0xFFFF);
32     new_entry->base_high = (base >> 16) & 0xFFFF;
33     new_entry->sselect = sel;
34     new_entry->zero = 0;
35     new_entry->flags = flags;
36 }
```

### 5.14.1.3 init\_gdt()

```
void init_gdt ( )
```

Definition at line 75 of file tables.c.

```
76 {
77     gdt_ptr.limit = 5 * sizeof(gdt_entry) - 1;
78     gdt_ptr.base = (u32int) gdt_entries;
79
80     u32int limit = 0xFFFFFFFF;
81     gdt_init_entry(0, 0, 0, 0, 0); //required null segment
82     gdt_init_entry(1, 0, limit, 0x9A, 0xCF); //code segment
83     gdt_init_entry(2, 0, limit, 0x92, 0xCF); //data segment
84     gdt_init_entry(3, 0, limit, 0xFA, 0xCF); //user mode code segment
85     gdt_init_entry(4, 0, limit, 0xF2, 0xCF); //user mode data segment
86
87     write_gdt_ptr((u32int) &gdt_ptr, sizeof(gdt_ptr));
88 }
```

#### 5.14.1.4 init\_idt()

```
void init_idt ( )
```

Definition at line 43 of file tables.c.

```
44 {  
45     idt_ptr.limit = 256*sizeof(idt_descriptor) - 1;  
46     idt_ptr.base  = (u32int)idt_entries;  
47     memset(idt_entries, 0, 256*sizeof(idt_descriptor));  
48  
49     write_idt_ptr((u32int)&idt_ptr);  
50 }
```

#### 5.14.1.5 write\_gdt\_ptr()

```
void write_gdt_ptr (  
    u32int ,  
    size_t )
```

#### 5.14.1.6 write\_idt\_ptr()

```
void write_idt_ptr (  
    u32int )
```

### 5.14.2 Variable Documentation

#### 5.14.2.1 gdt\_entries

```
gdt_entry gdt_entries[5]
```

Definition at line 13 of file tables.c.

#### 5.14.2.2 gdt\_ptr

```
gdt_descriptor gdt_ptr
```

Definition at line 12 of file tables.c.

### 5.14.2.3 idt\_entries

```
idt_entry idt_entries[256]
```

Definition at line 17 of file tables.c.

### 5.14.2.4 idt\_ptr

```
idt_descriptor idt_ptr
```

Definition at line 16 of file tables.c.

## 5.15 kernel/mem/heap.c File Reference

```
#include <system.h>
#include <string.h>
#include <core/serial.h>
#include <mem/heap.h>
#include <mem/paging.h>
```

### Functions

- [u32int \\_kmalloc](#) ([u32int](#) size, int page\_align, [u32int](#)\*phys\_addr)
- [u32int kmalloc](#) ([u32int](#) size)
- [u32int alloc](#) ([u32int](#) size, [heap](#)\*h, int align)
- [heap](#)\*make\_heap ([u32int](#) base, [u32int](#) max, [u32int](#) min)

### Variables

- [heap](#)\*kheap = 0
- [heap](#)\*curr\_heap = 0
- [page\\_dir](#)\*kdir
- void\*end
- void \_end
- void \_\_end
- [u32int](#) phys\_alloc\_addr = ([u32int](#))&end

### 5.15.1 Function Documentation

### 5.15.1.1 \_kmalloc()

```
u32int _kmalloc (
    u32int size,
    int page_align,
    u32int *phys_addr )
```

Definition at line 24 of file heap.c.

```
25 {
26     u32int *addr;
27
28     // Allocate on the kernel heap if one has been created
29     if (kheap != 0){
30         addr = (u32int*)alloc(size, kheap, page_align);
31         if (phys_addr){
32             page_entry *page = get_page((u32int)addr, kdir, 0);
33             *phys_addr = (page->frameaddr*0x1000) + ((u32int)addr & 0xFFF);
34         }
35         return (u32int)addr;
36     }
37     // Else, allocate directly from physical memory
38     else {
39         if (page_align && (phys_alloc_addr & 0xFFFFF000)){
40             phys_alloc_addr &= 0xFFFFF000;
41             phys_alloc_addr += 0x1000;
42         }
43         addr = (u32int*)phys_alloc_addr;
44         if (phys_addr){
45             *phys_addr = phys_alloc_addr;
46         }
47         phys_alloc_addr += size;
48         return (u32int)addr;
49     }
50 }
```

### 5.15.1.2 alloc()

```
u32int alloc (
    u32int size,
    heap *h,
    int align )
```

Definition at line 57 of file heap.c.

```
58 {
59     no_warn(size||align||h);
60     static u32int heap_addr = KHEAP_BASE;
61
62     u32int base = heap_addr;
63     heap_addr += size;
64
65     if (heap_addr > KHEAP_BASE + KHEAP_MIN)
66         serial_println("Heap is full!");
67
68     return base;
69 }
```

### 5.15.1.3 kmalloc()

```
u32int kmalloc (
    u32int size )
```

Definition at line 52 of file heap.c.

```
53 {
54     return _kmalloc(size,0,0);
55 }
```

#### 5.15.1.4 make\_heap()

```
heap* make_heap (
    u32int base,
    u32int max,
    u32int min )
```

Definition at line 71 of file heap.c.

```
72 {
73     no_warn (base | max | min);
74     return (heap*) kmalloc (sizeof (heap));
75 }
```

### 5.15.2 Variable Documentation

#### 5.15.2.1 \_\_end

```
void __end
```

Definition at line 18 of file heap.c.

#### 5.15.2.2 \_end

```
void _end
```

Definition at line 18 of file heap.c.

#### 5.15.2.3 curr\_heap

```
heap* curr_heap = 0
```

Definition at line 15 of file heap.c.

#### 5.15.2.4 end

```
void* end [extern]
```

#### 5.15.2.5 kdir

```
page_dir* kdir [extern]
```

Definition at line 21 of file paging.c.

#### 5.15.2.6 kheap

```
heap* kheap = 0
```

Definition at line 14 of file heap.c.

#### 5.15.2.7 phys\_alloc\_addr

```
u32int phys_alloc_addr = (u32int)&end
```

Definition at line 22 of file heap.c.

## 5.16 kernel/mem/paging.c File Reference

```
#include <system.h>
#include <string.h>
#include "mem/heap.h"
#include "mem/paging.h"
```

### Functions

- void [set\\_bit](#) (u32int addr)
- void [clear\\_bit](#) (u32int addr)
- u32int [get\\_bit](#) (u32int addr)
- u32int [find\\_free](#) ()
- page\_entry \*[get\\_page](#) (u32int addr, page\_dir \*dir, int make\_table)
- void [init\\_paging](#) ()
- void [load\\_page\\_dir](#) (page\_dir \*new\_dir)
- void [new\\_frame](#) (page\_entry \*page)

### Variables

- u32int [mem\\_size](#) = 0x4000000
- u32int [page\\_size](#) = 0x1000
- u32int [nframes](#)
- u32int \*[frames](#)
- page\_dir \*[kdir](#) = 0
- page\_dir \*[kdir](#) = 0
- u32int [phys\\_alloc\\_addr](#)
- heap \*[kheap](#)



## 5.16.1 Function Documentation

### 5.16.1.1 clear\_bit()

```
void clear_bit (
    u32int addr )
```

Definition at line 44 of file paging.c.

```
45 {
46     u32int frame = addr/page_size;
47     u32int index = frame/32;
48     u32int offset = frame%32;
49     frames[index] &= ~(1 << offset);
50 }
```

### 5.16.1.2 find\_free()

```
u32int find_free ( )
```

Definition at line 68 of file paging.c.

```
69 {
70     u32int i, j;
71     for (i=0; i<nframes/32; i++)
72         if (frames[i] != 0xFFFFFFFF) //if frame not full
73             for (j=0; j<32; j++) //find first free bit
74                 if (!(frames[i] & (1 << j)))
75                     return i*32+j;
76
77     return -1; //no free frames
78 }
```

### 5.16.1.3 get\_bit()

```
u32int get_bit (
    u32int addr )
```

Definition at line 56 of file paging.c.

```
57 {
58     u32int frame = addr/page_size;
59     u32int index = frame/32;
60     u32int offset = frame%32;
61     return (frames[index] & (1 << offset));
62 }
```

### 5.16.1.4 get\_page()

```
page_entry* get_page (
    u32int addr,
    page_dir * dir,
    int make_table )
```

Definition at line 85 of file paging.c.

```
86 {
87     u32int phys_addr;
88     u32int index = addr / page_size / 1024;
89     u32int offset = addr / page_size % 1024;
90
91     //return it if it exists
92     if (dir->tables[index])
93         return &dir->tables[index]->pages[offset];
94
95     //create it
96     else if (make_table){
97         dir->tables[index] = (page_table*)_kmalloc(sizeof(page_table), 1, &phys_addr);
98         dir->tables_phys[index] = phys_addr | 0x7; //enable present, writable
99         return &dir->tables[index]->pages[offset];
100     }
101     else return 0;
102 }
```

### 5.16.1.5 init\_paging()

```
void init_paging ( )
```

Definition at line 111 of file paging.c.

```
112 {
113     //create frame bitmap
114     nframes = (u32int)(mem_size/page_size);
115     frames = (u32int*)kmalloc(nframes/32);
116     memset(frames, 0, nframes/32);
117
118     //create kernel directory
119     kdir = (page_dir*)_kmalloc(sizeof(page_dir), 1, 0); //page aligned
120     memset(kdir, 0, sizeof(page_dir));
121
122     //get pages for kernel heap
123     u32int i = 0x0;
124     for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=1){
125         get_page(i, kdir, 1);
126     }
127
128     //perform identity mapping of used memory
129     //note: placement_addr gets incremented in get_page,
130     //so we're mapping the first frames as well
131     i = 0x0;
132     while (i < (phys_alloc_addr+0x10000)){
133         new_frame(get_page(i, kdir, 1));
134         i += page_size;
135     }
136
137     //allocate heap frames now that the placement addr has increased.
138     //placement addr increases here for heap
139     for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=PAGE_SIZE){
140         new_frame(get_page(i, kdir, 1));
141     }
142
143     //load the kernel page directory; enable paging
144     load_page_dir(kdir);
145
146     //setup the kernel heap
147     kheap = make_heap(KHEAP_BASE, KHEAP_SIZE, KHEAP_BASE+KHEAP_MIN);
148 }
```

### 5.16.1.6 load\_page\_dir()

```
void load_page_dir (
    page_dir * new_dir )
```

Definition at line 158 of file paging.c.

```
159 {
160     cdir = new_dir;
161     asm volatile ("mov %0,%cr3:: "b"(&cdir->tables_phys[0]));
162     u32int cr0;
163     asm volatile ("mov %%cr0,%0": "=b"(cr0));
164     cr0 |= 0x80000000;
165     asm volatile ("mov %0,%%cr0:: "b"(cr0));
166 }
```

### 5.16.1.7 new\_frame()

```
void new_frame (
    page_entry * page )
```

Definition at line 173 of file paging.c.

```
174 {
175     u32int index;
176     if (page->frameaddr != 0) return;
177     if ( (u32int)(-1) == (index=find_free()) ) kpanic("Out of memory");
178
179     //mark a frame as in-use
180     set_bit(index*page_size);
181     page->present = 1;
182     page->frameaddr = index;
183     page->writeable = 1;
184     page->usermode = 0;
185 }
```

### 5.16.1.8 set\_bit()

```
void set_bit (
    u32int addr )
```

Definition at line 32 of file paging.c.

```
33 {
34     u32int frame = addr/page_size;
35     u32int index = frame/32;
36     u32int offset = frame%32;
37     frames[index] |= (1 << offset);
38 }
```

## 5.16.2 Variable Documentation

### 5.16.2.1 cdir

```
page_dir* cdir = 0
```

Definition at line 22 of file paging.c.

#### 5.16.2.2 frames

```
u32int* frames
```

Definition at line 19 of file paging.c.

#### 5.16.2.3 kdir

```
page_dir* kdir = 0
```

Definition at line 21 of file paging.c.

#### 5.16.2.4 kheap

```
heap* kheap [extern]
```

Definition at line 14 of file heap.c.

#### 5.16.2.5 mem\_size

```
u32int mem_size = 0x4000000
```

Definition at line 15 of file paging.c.

#### 5.16.2.6 nframes

```
u32int nframes
```

Definition at line 18 of file paging.c.

#### 5.16.2.7 page\_size

```
u32int page_size = 0x1000
```

Definition at line 16 of file paging.c.

### 5.16.2.8 phys\_alloc\_addr

```
u32int phys_alloc_addr [extern]
```

Definition at line 22 of file heap.c.

## 5.17 lib/string.c File Reference

```
#include <system.h>
#include <string.h>
```

### Functions

- int [strlen](#) (const char \*s)
- char \*[strcpy](#) (char \*s1, const char \*s2)
- int [atoi](#) (const char \*s)
- int [strcmp](#) (const char \*s1, const char \*s2)
- char \*[strcat](#) (char \*s1, const char \*s2)
- int [isspace](#) (const char c)
- void \*[memset](#) (void \*s, int c, [size\\_t](#) n)
- char \*[strtok](#) (char \*s1, const char \*s2)

### 5.17.1 Function Documentation

#### 5.17.1.1 atoi()

```
int atoi (
    const char *s )
```

Definition at line 48 of file string.c.

```
49 {
50     int res=0;
51     int charVal=0;
52     char sign = ' ';
53     char c = *s;
54
55
56     while(isspace(&c)){ ++s; c = *s;} // advance past whitespace
57
58     if (*s == '-' || *s == '+') sign = *(s++); // save the sign
59
60     while(*s != '\0'){
61         charVal = *s - 48;
62         res = res * 10 + charVal;
63         s++;
64     }
65
66     if ( sign == '-' ) res=res * -1;
67
68     return res; // return integer
69 }
70
71
72
73 }
```

### 5.17.1.2 isspace()

```
int isspace (
    const char * c )
```

Definition at line 119 of file string.c.

```
120 {
121     if (*c == ' ' ||
122         *c == '\n' ||
123         *c == '\r' ||
124         *c == '\f' ||
125         *c == '\t' ||
126         *c == '\v') {
127         return 1;
128     }
129     return 0;
130 }
```

### 5.17.1.3 memset()

```
void* memset (
    void * s,
    int c,
    size_t n )
```

Definition at line 137 of file string.c.

```
138 {
139     unsigned char *p = (unsigned char *) s;
140     while(n--){
141         *p++ = (unsigned char) c;
142     }
143     return s;
144 }
```

### 5.17.1.4 strcat()

```
char* strcat (
    char * s1,
    const char * s2 )
```

Definition at line 106 of file string.c.

```
107 {
108     char *rc = s1;
109     if (*s1) while(++s1);
110     while( (*s1++ = *s2++) );
111     return rc;
112 }
```

### 5.17.1.5 strcmp()

```
int strcmp (
    const char *s1,
    const char *s2 )
```

Definition at line 79 of file string.c.

```
80 {
81
82     // Remarks:
83     // 1) If we made it to the end of both strings (i. e. our pointer points to a
84     //     '\0' character), the function will return 0
85     // 2) If we didn't make it to the end of both strings, the function will
86     //     return the difference of the characters at the first index of
87     //     indifference.
88     while ( (*s1) && (*s1==*s2) ){
89         ++s1;
90         ++s2;
91     }
92     return ( *(unsigned char *)s1 - *(unsigned char *)s2 );
93 }
```

### 5.17.1.6 strcpy()

```
char* strcpy (
    char *s1,
    const char *s2 )
```

Definition at line 36 of file string.c.

```
37 {
38     char *rc = s1;
39     while( (*s1++ = *s2++) );
40     return rc; // return pointer to destination string
41 }
```

### 5.17.1.7 strlen()

```
int strlen (
    const char *s )
```

Definition at line 24 of file string.c.

```
25 {
26     int r1 = 0;
27     if (*s) while(*s++) r1++;
28     return r1; //return length of string
29 }
```

### 5.17.1.8 strtok()

```
char* strtok (
    char * s1,
    const char * s2 )
```

Definition at line 151 of file string.c.

```
152 {
153     static char *tok_tmp = NULL;
154     const char *p = s2;
155
156     //new string
157     if (s1!=NULL){
158         tok_tmp = s1;
159     }
160     //old string cont'd
161     else {
162         if (tok_tmp==NULL){
163             return NULL;
164         }
165         s1 = tok_tmp;
166     }
167
168     //skip leading s2 characters
169     while ( *p && *s1 ){
170         if (*s1==*p){
171             ++s1;
172             p = s2;
173             continue;
174         }
175         ++p;
176     }
177
178     //no more to parse
179     if (!*s1){
180         return (tok_tmp = NULL);
181     }
182
183     //skip non-s2 characters
184     tok_tmp = s1;
185     while (*tok_tmp){
186         p = s2;
187         while (*p){
188             if (*tok_tmp==*p++){
189                 *tok_tmp++ = '\0';
190                 return s1;
191             }
192         }
193         ++tok_tmp;
194     }
195
196     //end of string
197     tok_tmp = NULL;
198     return s1;
199 }
```

## 5.18 modules/mpx\_supt.c File Reference

```
#include "mpx_supt.h"
#include <mem/heap.h>
#include <string.h>
#include <core/serial.h>
```

### Functions

- int [sys\\_req](#) (int op\_code, int device\_id, char\*buffer\_ptr, int\*count\_ptr)
- void [mpx\\_init](#) (int cur\_mod)
- void [sys\\_set\\_malloc](#) (u32int)(\*unc)(u32int))
- void [sys\\_set\\_free](#) (int)(\*unc)(void \*)
- void \*[sys\\_alloc\\_mem](#) (u32int size)
- int [sys\\_free\\_mem](#) (void \*ptr)
- void [idle](#) ()



## Variables

- [param params](#)
- `int current_module = -1`
- `static int io_module_active = 0`
- `static int mem_module_active = 0`
- `u32int(*student_malloc)(u32int)`
- `int(*student_free)(void *)`

## 5.18.1 Function Documentation

### 5.18.1.1 idle()

```
void idle ( )
```

Definition at line 173 of file mpx\_supt.c.

```
174 {
175     char msg[30];
176     int count=0;
177
178     memset( msg, '\0', sizeof(msg));
179     strcpy(msg, "IDLE PROCESS EXECUTING.\n");
180     count = strlen(msg);
181
182     while(1){
183         sys_req( WRITE, DEFAULT_DEVICE, msg, &count);
184         sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
185     }
186 }
```

### 5.18.1.2 mpx\_init()

```
void mpx_init (
    int cur_mod )
```

Definition at line 106 of file mpx\_supt.c.

```
107 {
108
109     current_module = cur_mod;
110     if (cur_mod == MEM_MODULE)
111         mem_module_active = TRUE;
112
113     if (cur_mod == IO_MODULE)
114         io_module_active = TRUE;
115 }
```

### 5.18.1.3 sys\_alloc\_mem()

```
void* sys_alloc_mem (
    u32int size )
```

Definition at line 144 of file mpx\_supt.c.

```
145 {
146     if (!mem_module_active)
147         return (void *) kcalloc(size);
148     else
149         return (void *) (*student_malloc)(size);
150 }
```

#### 5.18.1.4 sys\_free\_mem()

```
int sys_free_mem (
    void *ptr )
```

Definition at line 158 of file mpx\_supt.c.

```
159 {
160     if (mem_module_active)
161         return (*student_free)(ptr);
162     // otherwise we don't free anything
163     return -1;
164 }
```

#### 5.18.1.5 sys\_req()

```
int sys_req (
    int op_code,
    int device_id,
    char *buffer_ptr,
    int *count_ptr )
```

Definition at line 49 of file mpx\_supt.c.

```
54 {
55     int return_code = 0;
56
57     if (op_code == IDLE || op_code == EXIT){
58         // store the process's operation request
59         // trigger interrupt 60h to invoke
60         params.op_code = op_code;
61         asm volatile ("int $60");
62     } // idle or exit
63
64     else if (op_code == READ || op_code == WRITE) {
65         // validate buffer pointer and count pointer
66         if (buffer_ptr == NULL)
67             return_code = INVALID_BUFFER;
68         else if (count_ptr == NULL || *count_ptr <= 0)
69             return_code = INVALID_COUNT;
70
71         // if parameters are valid store in the params structure
72         if ( return_code == 0){
73             params.op_code = op_code;
74             params.device_id = device_id;
75             params.buffer_ptr = buffer_ptr;
76             params.count_ptr = count_ptr;
77
78             if (!io_module_active){
79                 // if default device
80                 if (op_code == READ)
81                     return_code = *(polling(buffer_ptr, count_ptr));
82
83                 else //must be WRITE
84                     return_code = serial_print(buffer_ptr);
85
86             } else { // I/O module is implemented
87                 asm volatile ("int $60");
88             } // NOT IO_MODULE
89         }
90     } else return_code = INVALID_OPERATION;
91
92     return return_code;
93 } // end of sys_req
```

### 5.18.1.6 sys\_set\_free()

```
void sys_set_free (
    int (*)(void *) func )
```

Definition at line 134 of file mpx\_supt.c.

```
135 {
136     student_free = func;
137 }
```

### 5.18.1.7 sys\_set\_malloc()

```
void sys_set_malloc (
    u32int (*)(u32int) func )
```

Definition at line 124 of file mpx\_supt.c.

```
125 {
126     student_malloc = func;
127 }
```

## 5.18.2 Variable Documentation

### 5.18.2.1 current\_module

```
int current_module = -1
```

Definition at line 18 of file mpx\_supt.c.

### 5.18.2.2 io\_module\_active

```
int io_module_active = 0 [static]
```

Definition at line 19 of file mpx\_supt.c.

### 5.18.2.3 mem\_module\_active

```
int mem_module_active = 0 [static]
```

Definition at line 20 of file mpx\_supt.c.

#### 5.18.2.4 params

`param` params

Definition at line 15 of file mpx\_supt.c.

#### 5.18.2.5 student\_free

`int (* student_free) (void *)`

Definition at line 28 of file mpx\_supt.c.

#### 5.18.2.6 student\_malloc

`u32int (* student_malloc) (u32int)`

Definition at line 24 of file mpx\_supt.c.

## 5.19 modules/mpx\_supt.h File Reference

```
#include <system.h>
```

### Classes

- struct `param`

### Macros

- `#define EXIT` 0
- `#define IDLE` 1
- `#define READ` 2
- `#define WRITE` 3
- `#define INVALID_OPERATION` 4
- `#define TRUE` 1
- `#define FALSE` 0
- `#define MODULE_R1` 0
- `#define MODULE_R2` 1
- `#define MODULE_R3` 2
- `#define MODULE_R4` 4
- `#define MODULE_R5` 8
- `#define MODULE_F` 9
- `#define IO_MODULE` 10
- `#define MEM_MODULE` 11
- `#define INVALID_BUFFER` 1000
- `#define INVALID_COUNT` 2000
- `#define DEFAULT_DEVICE` 111
- `#define COM_PORT` 222

## Functions

- int [sys\\_req](#) (int op\_code, int device\_id, char\*buffer\_ptr, int\*count\_ptr)
- void [mpx\\_init](#) (int cur\_mod)
- void [sys\\_set\\_malloc](#) (u32int)(\*func)(u32int))
- void [sys\\_set\\_free](#) (int)(\*func)(void \*)
- void \*[sys\\_alloc\\_mem](#) (u32int size)
- int [sys\\_free\\_mem](#) (void \*ptr)
- void [idle](#) ()

## 5.19.1 Macro Definition Documentation

### 5.19.1.1 COM\_PORT

```
#define COM_PORT 222
```

Definition at line 29 of file mpx\_supt.h.

### 5.19.1.2 DEFAULT\_DEVICE

```
#define DEFAULT_DEVICE 111
```

Definition at line 28 of file mpx\_supt.h.

### 5.19.1.3 EXIT

```
#define EXIT 0
```

Definition at line 6 of file mpx\_supt.h.

### 5.19.1.4 FALSE

```
#define FALSE 0
```

Definition at line 13 of file mpx\_supt.h.

#### 5.19.1.5 IDLE

```
#define IDLE 1
```

Definition at line 7 of file mpx\_supt.h.

#### 5.19.1.6 INVALID\_BUFFER

```
#define INVALID_BUFFER 1000
```

Definition at line 25 of file mpx\_supt.h.

#### 5.19.1.7 INVALID\_COUNT

```
#define INVALID_COUNT 2000
```

Definition at line 26 of file mpx\_supt.h.

#### 5.19.1.8 INVALID\_OPERATION

```
#define INVALID_OPERATION 4
```

Definition at line 10 of file mpx\_supt.h.

#### 5.19.1.9 IO\_MODULE

```
#define IO_MODULE 10
```

Definition at line 21 of file mpx\_supt.h.

#### 5.19.1.10 MEM\_MODULE

```
#define MEM_MODULE 11
```

Definition at line 22 of file mpx\_supt.h.

#### 5.19.1.11 MODULE\_F

```
#define MODULE_F 9
```

Definition at line 20 of file mpx\_supt.h.

#### 5.19.1.12 MODULE\_R1

```
#define MODULE_R1 0
```

Definition at line 15 of file mpx\_supt.h.

#### 5.19.1.13 MODULE\_R2

```
#define MODULE_R2 1
```

Definition at line 16 of file mpx\_supt.h.

#### 5.19.1.14 MODULE\_R3

```
#define MODULE_R3 2
```

Definition at line 17 of file mpx\_supt.h.

#### 5.19.1.15 MODULE\_R4

```
#define MODULE_R4 4
```

Definition at line 18 of file mpx\_supt.h.

#### 5.19.1.16 MODULE\_R5

```
#define MODULE_R5 8
```

Definition at line 19 of file mpx\_supt.h.

#### 5.19.1.17 READ

```
#define READ 2
```

Definition at line 8 of file mpx\_supt.h.

#### 5.19.1.18 TRUE

```
#define TRUE 1
```

Definition at line 12 of file mpx\_supt.h.

#### 5.19.1.19 WRITE

```
#define WRITE 3
```

Definition at line 9 of file mpx\_supt.h.

### 5.19.2 Function Documentation

#### 5.19.2.1 idle()

```
void idle ( )
```

Definition at line 173 of file mpx\_supt.c.

```
174 {  
175     char msg[30];  
176     int count=0;  
177  
178     memset( msg, '\0', sizeof(msg));  
179     strcpy(msg, "IDLE PROCESS EXECUTING.\n");  
180     count = strlen(msg);  
181  
182     while(1){  
183         sys_req( WRITE, DEFAULT_DEVICE, msg, &count);  
184         sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);  
185     }  
186 }
```



### 5.19.2.2 mpx\_init()

```
void mpx_init (
    int cur_mod )
```

Definition at line 106 of file mpx\_supt.c.

```
107 {
108
109     current_module = cur_mod;
110     if (cur_mod == MEM_MODULE)
111         mem_module_active = TRUE;
112
113     if (cur_mod == IO_MODULE)
114         io_module_active = TRUE;
115 }
```

### 5.19.2.3 sys\_alloc\_mem()

```
void* sys_alloc_mem (
    u32int size )
```

Definition at line 144 of file mpx\_supt.c.

```
145 {
146     if (!mem_module_active)
147         return (void *) kmalloc(size);
148     else
149         return (void *) (*student_malloc)(size);
150 }
```

### 5.19.2.4 sys\_free\_mem()

```
int sys_free_mem (
    void *ptr )
```

Definition at line 158 of file mpx\_supt.c.

```
159 {
160     if (mem_module_active)
161         return (*student_free)(ptr);
162     // otherwise we don't free anything
163     return -1;
164 }
```

### 5.19.2.5 sys\_req()

```
int sys_req (
    int op_code,
    int device_id,
    char *buffer_ptr,
    int *count_ptr )
```

Definition at line 49 of file mpx\_supt.c.

```
54 {
55     int return_code = 0;
56
57     if (op_code == IDLE || op_code == EXIT){
```

```

58     // store the process's operation request
59     // trigger interrupt 60h to invoke
60     params.op_code = op_code;
61     asm volatile ("int $60");
62 } // idle or exit
63
64 else if (op_code == READ || op_code == WRITE) {
65     // validate buffer pointer and count pointer
66     if (buffer_ptr == NULL)
67         return_code = INVALID_BUFFER;
68     else if (count_ptr == NULL || *count_ptr <= 0)
69         return_code = INVALID_COUNT;
70
71     // if parameters are valid store in the params structure
72     if ( return_code == 0){
73         params.op_code = op_code;
74         params.device_id = device_id;
75         params.buffer_ptr = buffer_ptr;
76         params.count_ptr = count_ptr;
77
78         if (!io_module_active){
79             // if default device
80             if (op_code == READ)
81                 return_code = *(polling(buffer_ptr, count_ptr));
82
83             else //must be WRITE
84                 return_code = serial_print(buffer_ptr);
85
86         } else { // I/O module is implemented
87             asm volatile ("int $60");
88         } // NOT IO_MODULE
89     }
90 } else return_code = INVALID_OPERATION;
91
92 return return_code;
93 } // end of sys_req

```

### 5.19.2.6 sys\_set\_free()

```

void sys_set_free (
    int(*) (void *) func )

```

Definition at line 134 of file mpx\_supt.c.

```

135 {
136     student_free = func;
137 }

```

### 5.19.2.7 sys\_set\_malloc()

```

void sys_set_malloc (
    u32int(*) (u32int) func )

```

Definition at line 124 of file mpx\_supt.c.

```

125 {
126     student_malloc = func;
127 }

```

## 5.20 modules/R1/commhand.c File Reference

```

#include <core/serial.h>
#include <string.h>
#include "../mpx_supt.h"
#include "R1commands.h"
#include "../R2/R2commands.h"
#include "../R2/R2_Internal_Functions_And_Structures.h"

```

## Functions

- int [commhand](#) ()

### 5.20.1 Function Documentation

#### 5.20.1.1 commhand()

int commhand ( )

Definition at line 10 of file commhand.c.

```

11 {
12
13     char welcomeMSG[] = "\nWelcome to our CS 450 Project!\nType help to see what you can do!\n\n";
14     int welcomeLength = strlen(welcomeMSG);
15     sys_req(WRITE, DEFAULT_DEVICE, welcomeMSG, &welcomeLength);
16
17     char cmdBuffer[100];
18     int bufferSize;
19     allocateQueues();
20
21     int quitFlag = 0;
22
23     while (!quitFlag)
24     {
25         //get a command: cal polling fx
26
27         memset(cmdBuffer, ' ', 100);
28
29         bufferSize = 99; // reset size before each call to read
30
31         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
32
33         char newLine[] = "\n";
34         int newLineCount = 1;
35         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
36
37         if (strcmp(cmdBuffer, "help") == 0)
38         {
39             help();
40         }
41         else if (strcmp(cmdBuffer, "version") == 0)
42         {
43             version();
44         }
45         else if (strcmp(cmdBuffer, "getDate") == 0)
46         {
47             getDate();
48         }
49         else if (strcmp(cmdBuffer, "setDate") == 0)
50         {
51             setDate();
52         }
53         else if (strcmp(cmdBuffer, "getTime") == 0)
54         {
55             getTime();
56         }
57         else if (strcmp(cmdBuffer, "setTime") == 0)
58         {
59             setTime();
60         }
61         else if (strcmp(cmdBuffer, "createPCB") == 0)
62         {
63             char processName[20];
64             char processClass;
65             int processPriority;
66
67             char nameMsg[] = "Please enter a name for the PCB you wish to create. (The name can be no
more than 20 characters)\n";
68             int nameMsgLen = strlen(nameMsg);
69             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
70             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
71             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
72             strcpy(processName, cmdBuffer);

```

```

73         memset(cmdBuffer, '\0', 100);
74
75         char classMsg[] = "Please enter a class for the PCB you wish to create. ('a' for application
or 's' for system)\n";
76         int classMsgLen = strlen(classMsg);
77         sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
78         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
79         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
80         if (strcmp(cmdBuffer, "a") == 0)
81         {
82             processClass = 'a';
83         }
84         else if (strcmp(cmdBuffer, "s") == 0)
85         {
86             processClass = 's';
87         }
88         else
89         {
90             processClass = '\0';
91         }
92         memset(cmdBuffer, '\0', 100);
93
94         char priorityMsg[] = "Please enter a priority for the PCB you wish to create. (The priorities
range from 0 to 9)\n";
95         int priorityMsgLen = strlen(priorityMsg);
96         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
97         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
98         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
99         processPriority = atoi(cmdBuffer);
100
101         createPCB(processName, processClass, processPriority);
102     }
103     else if (strcmp(cmdBuffer, "deletePCB") == 0)
104     {
105         char processName[20];
106
107         char nameMsg[] = "Please enter the name for the PCB you wish to delete. (The name can be no
more than 20 characters)\n";
108         int nameMsgLen = strlen(nameMsg);
109         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
110         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
111         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
112         strcpy(processName, cmdBuffer);
113
114         deletePCB(processName);
115     }
116     else if (strcmp(cmdBuffer, "blockPCB") == 0)
117     {
118         char processName[20];
119
120         char nameMsg[] = "Please enter the name for the PCB you wish to block. (The name can be no
more than 20 characters)\n";
121         int nameMsgLen = strlen(nameMsg);
122         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
123         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
124         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
125         strcpy(processName, cmdBuffer);
126
127         blockPCB(processName);
128     }
129     else if (strcmp(cmdBuffer, "unblockPCB") == 0)
130     {
131         char processName[20];
132
133         char nameMsg[] = "Please enter the name for the PCB you wish to unblock. (The name can be no
more than 20 characters)\n";
134         int nameMsgLen = strlen(nameMsg);
135         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
136         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
137         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
138         strcpy(processName, cmdBuffer);
139
140         unblockPCB(processName);
141     }
142     else if (strcmp(cmdBuffer, "suspendPCB") == 0)
143     {
144         char processName[20];
145
146         char nameMsg[] = "Please enter the name for the PCB you wish to suspend. (The name can be no
more than 20 characters)\n";
147         int nameMsgLen = strlen(nameMsg);
148         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
149         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
150         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
151         strcpy(processName, cmdBuffer);
152
153         suspendPCB(processName);

```

```

154     }
155     else if (strcmp(cmdBuffer, "resumePCB") == 0)
156     {
157         char processName[20];
158
159         char nameMsg[] = "Please enter the name for the PCB you wish to resume. (The name can be no
more than 20 characters)\n";
160         int nameMsgLen = strlen(nameMsg);
161         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
162         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
163         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
164         strcpy(processName, cmdBuffer);
165
166         resumePCB(processName);
167     }
168     else if (strcmp(cmdBuffer, "setPCBPRIORITY") == 0)
169     {
170         char processName[20];
171         int newProcessPriority;
172
173         char nameMsg[] = "Please enter the name for the PCB you wish to change priorities for. (The
name can be no more than 20 characters)\n";
174         int nameMsgLen = strlen(nameMsg);
175         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
176         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
177         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
178         strcpy(processName, cmdBuffer);
179
180         char priorityMsg[] = "Please enter a priority for the PCB you wish to change priorities for.
(The priorities range from 0 to 9)\n";
181         int priorityMsgLen = strlen(priorityMsg);
182         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
183         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
184         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
185         newProcessPriority = atoi(cmdBuffer);
186
187         setPCBPRIORITY(processName, newProcessPriority);
188     }
189     else if (strcmp(cmdBuffer, "showPCB") == 0)
190     {
191         char processName[20];
192
193         char nameMsg[] = "Please enter the name for the PCB you wish to see. (The name can be no
more than 20 characters)\n";
194         int nameMsgLen = strlen(nameMsg);
195         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
196         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
197         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
198         strcpy(processName, cmdBuffer);
199
200         showPCB(processName);
201     }
202     else if (strcmp(cmdBuffer, "showReady") == 0)
203     {
204         showReady();
205     }
206     else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
207     {
208         showSuspendedReady();
209     }
210     else if (strcmp(cmdBuffer, "showSuspendedBlocked") == 0)
211     {
212         showSuspendedBlocked();
213     }
214     else if (strcmp(cmdBuffer, "showBlocked") == 0)
215     {
216         showBlocked();
217     }
218     else if (strcmp(cmdBuffer, "showAll") == 0)
219     {
220         showAll();
221     }
222     else if (strcmp(cmdBuffer, "quit") == 0)
223     {
224         quitFlag = quit();
225
226         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
227     }
228     else
229     {
230         char message[] = "Unrecognized Command\n";
231
232         int tempBuffer = strlen(message);
233
234         sys_req(WRITE, DEFAULT_DEVICE, (char *)message, &tempBuffer);
235     }
236

```

```

237         // process the command: take array buffer chars and make a string. Decide what the cmd wants to
238     do
239     // see if quit was entered: if string == quit = 1
240     }
241     return 0;
242 }

```

## 5.21 modules/R1/commhand.h File Reference

### Functions

- int [commhand](#) ()

#### 5.21.1 Function Documentation

##### 5.21.1.1 commhand()

```
int commhand ( )
```

Definition at line 10 of file commhand.c.

```

11 {
12
13     char welcomeMSG[] = "\nWelcome to our CS 450 Project!\nType help to see what you can do!\n\n";
14     int welcomeLength = strlen(welcomeMSG);
15     sys_req(WRITE, DEFAULT_DEVICE, welcomeMSG, &welcomeLength);
16
17     char cmdBuffer[100];
18     int bufferSize;
19     allocateQueues();
20
21     int quitFlag = 0;
22
23     while (!quitFlag)
24     {
25         //get a command: cal polling fx
26
27         memset(cmdBuffer, ' ', 100);
28
29         bufferSize = 99; // reset size before each call to read
30
31         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
32
33         char newLine[] = "\n";
34         int newLineCount = 1;
35         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
36
37         if (strcmp(cmdBuffer, "help") == 0)
38         {
39             help();
40         }
41         else if (strcmp(cmdBuffer, "version") == 0)
42         {
43             version();
44         }
45         else if (strcmp(cmdBuffer, "getDate") == 0)
46         {
47             getDate();
48         }
49         else if (strcmp(cmdBuffer, "setDate") == 0)
50         {
51             setDate();
52         }
53         else if (strcmp(cmdBuffer, "getTime") == 0)
54         {
55             getTime();
56         }
57     }
58 }

```

```

57         else if (strcmp(cmdBuffer, "setTime") == 0)
58         {
59             setTime();
60         }
61         else if (strcmp(cmdBuffer, "createPCB") == 0)
62         {
63             char processName[20];
64             char processClass;
65             int processPriority;
66
67             char nameMsg[] = "Please enter a name for the PCB you wish to create. (The name can be no
more than 20 characters)\n";
68             int nameMsgLen = strlen(nameMsg);
69             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
70             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
71             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
72             strcpy(processName, cmdBuffer);
73             memset(cmdBuffer, ' ', 100);
74
75             char classMsg[] = "Please enter a class for the PCB you wish to create. ('a' for application
or 's' for system)\n";
76             int classMsgLen = strlen(classMsg);
77             sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
78             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
79             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
80             if (strcmp(cmdBuffer, "a") == 0)
81             {
82                 processClass = 'a';
83             }
84             else if (strcmp(cmdBuffer, "s") == 0)
85             {
86                 processClass = 's';
87             }
88             else
89             {
90                 processClass = ' ';
91             }
92             memset(cmdBuffer, ' ', 100);
93
94             char priorityMsg[] = "Please enter a priority for the PCB you wish to create. (The priorities
range from 0 to 9)\n";
95             int priorityMsgLen = strlen(priorityMsg);
96             sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
97             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
98             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
99             processPriority = atoi(cmdBuffer);
100
101             createPCB(processName, processClass, processPriority);
102         }
103         else if (strcmp(cmdBuffer, "deletePCB") == 0)
104         {
105             char processName[20];
106
107             char nameMsg[] = "Please enter the name for the PCB you wish to delete. (The name can be no
more than 20 characters)\n";
108             int nameMsgLen = strlen(nameMsg);
109             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
110             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
111             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
112             strcpy(processName, cmdBuffer);
113
114             deletePCB(processName);
115         }
116         else if (strcmp(cmdBuffer, "blockPCB") == 0)
117         {
118             char processName[20];
119
120             char nameMsg[] = "Please enter the name for the PCB you wish to block. (The name can be no
more than 20 characters)\n";
121             int nameMsgLen = strlen(nameMsg);
122             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
123             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
124             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
125             strcpy(processName, cmdBuffer);
126
127             blockPCB(processName);
128         }
129         else if (strcmp(cmdBuffer, "unblockPCB") == 0)
130         {
131             char processName[20];
132
133             char nameMsg[] = "Please enter the name for the PCB you wish to unblock. (The name can be no
more than 20 characters)\n";
134             int nameMsgLen = strlen(nameMsg);
135             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
136             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
137             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);

```

```

138         strcpy(processName, cmdBuffer);
139
140         unblockPCB(processName);
141     }
142     else if (strcmp(cmdBuffer, "suspendPCB") == 0)
143     {
144         char processName[20];
145
146         char nameMsg[] = "Please enter the name for the PCB you wish to suspend. (The name can be no
more than 20 characters)\n";
147         int nameMsgLen = strlen(nameMsg);
148         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
149         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
150         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
151         strcpy(processName, cmdBuffer);
152
153         suspendPCB(processName);
154     }
155     else if (strcmp(cmdBuffer, "resumePCB") == 0)
156     {
157         char processName[20];
158
159         char nameMsg[] = "Please enter the name for the PCB you wish to resume. (The name can be no
more than 20 characters)\n";
160         int nameMsgLen = strlen(nameMsg);
161         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
162         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
163         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
164         strcpy(processName, cmdBuffer);
165
166         resumePCB(processName);
167     }
168     else if (strcmp(cmdBuffer, "setPCBPRIORITY") == 0)
169     {
170         char processName[20];
171         int newProcessPriority;
172
173         char nameMsg[] = "Please enter the name for the PCB you wish to change priorities for. (The
name can be no more than 20 characters)\n";
174         int nameMsgLen = strlen(nameMsg);
175         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
176         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
177         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
178         strcpy(processName, cmdBuffer);
179
180         char priorityMsg[] = "Please enter a priority for the PCB you wish to change priorities for.
(The priorities range from 0 to 9)\n";
181         int priorityMsgLen = strlen(priorityMsg);
182         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
183         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
184         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
185         newProcessPriority = atoi(cmdBuffer);
186
187         setPCBPRIORITY(processName, newProcessPriority);
188     }
189     else if (strcmp(cmdBuffer, "showPCB") == 0)
190     {
191         char processName[20];
192
193         char nameMsg[] = "Please enter the name for the PCB you wish to see. (The name can be no
more than 20 characters)\n";
194         int nameMsgLen = strlen(nameMsg);
195         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
196         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
197         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
198         strcpy(processName, cmdBuffer);
199
200         showPCB(processName);
201     }
202     else if (strcmp(cmdBuffer, "showReady") == 0)
203     {
204         showReady();
205     }
206     else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
207     {
208         showSuspendedReady();
209     }
210     else if (strcmp(cmdBuffer, "showSuspendedBlocked") == 0)
211     {
212         showSuspendedBlocked();
213     }
214     else if (strcmp(cmdBuffer, "showBlocked") == 0)
215     {
216         showBlocked();
217     }
218     else if (strcmp(cmdBuffer, "showAll") == 0)
219     {

```



```

220         showAll();
221     }
222     else if (strcmp(cmdBuffer, "quit") == 0)
223     {
224         quitFlag = quit();
225
226         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
227     }
228     else
229     {
230         char message[] = "Unrecognized Command\n";
231
232         int tempBuffer = strlen(message);
233
234         sys_req(WRITE, DEFAULT_DEVICE, (char *)message, &tempBuffer);
235     }
236
237     // process the command: take array buffer chars and make a string. Decide what the cmd wants to
do
238     // see if quit was entered: if string == quit = 1
239 }
240
241     return 0;
242 }

```

## 5.22 modules/R1/R1commands.c File Reference

```

#include <core/serial.h>
#include <string.h>
#include "../mpx_supt.h"
#include <core/io.h>

```

### Functions

- int [BCDtoChar](#) (unsigned char test, char \*buffer)
- unsigned char [intToBCD](#) (int test)
- int [help](#) ()
- int [version](#) ()
- void [getTime](#) ()
- int [setTime](#) ()
- void [getDate](#) ()
- int [setDate](#) ()
- int [quit](#) ()

### 5.22.1 Function Documentation

#### 5.22.1.1 BCDtoChar()

```

int BCDtoChar (
    unsigned char test,
    char * buffer )

```

Definition at line 593 of file R1commands.c.

```

594 {
595
596     int val1 = (test / 16);
597     int val2 = (test % 16);
598
599     buffer[0] = val1 + '0';
600     buffer[1] = val2 + '0';
601
602     return 0;
603 }

```

### 5.22.1.2 getDate()

```
void getDate ( )
```

Definition at line 343 of file R1commands.c.

```

344 {
345
346     char buffer[4] = " \0\0\0\0";
347     int count = 4;
348     char divider = '/';
349     char newLine[1] = "\n";
350     int newLineCount = 1;
351
352     outb(0x70, 0x07); // getting Day of month value
353     BCDtoChar(inb(0x71), buffer);
354     buffer[2] = divider;
355     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
356     memset(buffer, ' \0', count);
357
358     outb(0x70, 0x08); // getting Month value
359     BCDtoChar(inb(0x71), buffer);
360     buffer[2] = divider;
361     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
362     memset(buffer, ' \0', count);
363
364     outb(0x70, 0x32); // getting Year value second byte
365     BCDtoChar(inb(0x71), buffer);
366     buffer[2] = ' \0';
367     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
368     memset(buffer, ' \0', count);
369
370     outb(0x70, 0x09); // getting Year value first byte
371     BCDtoChar(inb(0x71), buffer);
372     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
373     memset(buffer, ' \0', count);
374
375     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
376     memset(newLine, ' \0', newLineCount);
377 }
```

### 5.22.1.3 getTime()

```
void getTime ( )
```

Definition at line 190 of file R1commands.c.

```

191 {
192
193     char buffer[4] = " \0\0\0\0";
194     int count = 4;
195     char divider = ':';
196     char newLine[1] = "\n";
197     int newLineCount = 1;
198
199     outb(0x70, 0x04); // getting Hour value
200     BCDtoChar(inb(0x71), buffer);
201     buffer[2] = divider;
202     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
203     memset(buffer, ' \0', count);
204
205     outb(0x70, 0x02); // getting Minute value
206     BCDtoChar(inb(0x71), buffer);
207     buffer[2] = divider;
208     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
209     memset(buffer, ' \0', count);
210
211     outb(0x70, 0x00); // getting Second value
212     BCDtoChar(inb(0x71), buffer);
213     buffer[2] = ' \0';
214     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
215     memset(buffer, ' \0', count);
216
217     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
218     memset(newLine, ' \0', newLineCount);
219 }
```

## 5.22.1.4 help()

```
int help ( )
```

Definition at line 11 of file R1commands.c.

```
12 {
13
14     // Help Description section
15     char helpDesc[] = "help: Returns basic command information.\n";
16
17     int tempBuffer = strlen(helpDesc);
18
19     sys_req(WRITE, DEFAULT_DEVICE, (char *)helpDesc, &tempBuffer);
20     memset(helpDesc, '\0', tempBuffer);
21
22     // Version Description section
23     char versionDesc[] = "version: Returns the current version of the software.\n";
24
25     tempBuffer = strlen(versionDesc);
26
27     sys_req(WRITE, DEFAULT_DEVICE, (char *)versionDesc, &tempBuffer);
28     memset(versionDesc, '\0', tempBuffer);
29
30     // getTime Description section
31     char getTimeDesc[] = "getTime: Returns the current set time.\n";
32
33     tempBuffer = strlen(getTimeDesc);
34
35     sys_req(WRITE, DEFAULT_DEVICE, (char *)getTimeDesc, &tempBuffer);
36     memset(getTimeDesc, '\0', tempBuffer);
37
38     // setTime Description section
39     char setTimeDesc[] = "setTime: Allows the user to change the set time.\n";
40
41     tempBuffer = strlen(setTimeDesc);
42
43     sys_req(WRITE, DEFAULT_DEVICE, (char *)setTimeDesc, &tempBuffer);
44     memset(setTimeDesc, '\0', tempBuffer);
45
46     // getDate Description section
47     char getDateDesc[] = "getDate: Returns the current set date.\n";
48
49     tempBuffer = strlen(getDateDesc);
50
51     sys_req(WRITE, DEFAULT_DEVICE, (char *)getDateDesc, &tempBuffer);
52     memset(getDateDesc, '\0', tempBuffer);
53
54     // setDate Description section
55     char setDateDesc[] = "setDate: Allows the user to change the set date.\n";
56
57     tempBuffer = strlen(setDateDesc);
58
59     sys_req(WRITE, DEFAULT_DEVICE, (char *)setDateDesc, &tempBuffer);
60     memset(setDateDesc, '\0', tempBuffer);
61
62     // createPCB Description section
63     char createPCBDesc[] = "createPCB: Will create a PCB and put it into the ready queue by default.\n";
64
65     tempBuffer = strlen(createPCBDesc);
66
67     sys_req(WRITE, DEFAULT_DEVICE, (char *)createPCBDesc, &tempBuffer);
68     memset(createPCBDesc, '\0', tempBuffer);
69
70     // deletePCB Description section
71     char deletePCBDesc[] = "deletePCB: Will delete a specific PCB from what ever queue it is in. \n";
72
73     tempBuffer = strlen(deletePCBDesc);
74
75     sys_req(WRITE, DEFAULT_DEVICE, (char *)deletePCBDesc, &tempBuffer);
76     memset(deletePCBDesc, '\0', tempBuffer);
77
78     // blockPCB Description section
79     char blockPCBDesc[] = "blockPCB: Will change a specific PCB's state to blocked. \n";
80
81     tempBuffer = strlen(blockPCBDesc);
82
83     sys_req(WRITE, DEFAULT_DEVICE, (char *)blockPCBDesc, &tempBuffer);
84     memset(blockPCBDesc, '\0', tempBuffer);
85
86     // unblockPCB Description section
87     char unblockPCBDesc[] = "unblockPCB: Will change a specific PCB's state to ready. \n";
88
89     tempBuffer = strlen(unblockPCBDesc);
90
91     sys_req(WRITE, DEFAULT_DEVICE, (char *)unblockPCBDesc, &tempBuffer);
```

```

92     memset(unblockPCBDesc, '\0', tempBuffer);
93
94     // suspendPCB Description section
95     char suspendPCBDesc[] = "suspendPCB: Will suspend a specific PCB. \n";
96
97     tempBuffer = strlen(suspendPCBDesc);
98
99     sys_req(WRITE, DEFAULT_DEVICE, (char *)suspendPCBDesc, &tempBuffer);
100    memset(suspendPCBDesc, '\0', tempBuffer);
101
102    // resumePCB Description section
103    char resumePCBDesc[] = "resumePCB: Will unsuspend a specific PCB. \n";
104
105    tempBuffer = strlen(resumePCBDesc);
106
107    sys_req(WRITE, DEFAULT_DEVICE, (char *)resumePCBDesc, &tempBuffer);
108    memset(resumePCBDesc, '\0', tempBuffer);
109
110    // setPCBPRIORITY Description section
111    char setPCBPRIORITYDesc[] = "setPCBPRIORITY: Will change the priority of a specific PCB. \n";
112
113    tempBuffer = strlen(setPCBPRIORITYDesc);
114
115    sys_req(WRITE, DEFAULT_DEVICE, (char *)setPCBPRIORITYDesc, &tempBuffer);
116    memset(setPCBPRIORITYDesc, '\0', tempBuffer);
117
118    // showPCB Description section
119    char showPCBDesc[] = "showPCB: Will display the name, class, state, suspended status, and priority
of a specific PCB. \n";
120
121    tempBuffer = strlen(showPCBDesc);
122
123    sys_req(WRITE, DEFAULT_DEVICE, (char *)showPCBDesc, &tempBuffer);
124    memset(showPCBDesc, '\0', tempBuffer);
125
126    // showReady Description section
127    char showReadyDesc[] = "showReady: Will display the name, class, state, suspended status, and
priority of every PCB in the ready queue.\n";
128
129    tempBuffer = strlen(showReadyDesc);
130
131    sys_req(WRITE, DEFAULT_DEVICE, (char *)showReadyDesc, &tempBuffer);
132    memset(showReadyDesc, '\0', tempBuffer);
133
134    // showSuspendedReady Description section
135    char showSuspendedReadyDesc[] = "showSuspendedReady: Will display the name, class, state, suspended
status, and priority of every PCB in the suspended ready queue.\n";
136
137    tempBuffer = strlen(showSuspendedReadyDesc);
138
139    sys_req(WRITE, DEFAULT_DEVICE, (char *)showSuspendedReadyDesc, &tempBuffer);
140    memset(showSuspendedReadyDesc, '\0', tempBuffer);
141
142    // showSuspendedBlocked Description section
143    char showSuspendedBlockedDesc[] = "showSuspendedBlocked: Will display the name, class, state,
suspended status, and priority of every PCB in the suspended blocked queue.\n";
144
145    tempBuffer = strlen(showSuspendedBlockedDesc);
146
147    sys_req(WRITE, DEFAULT_DEVICE, (char *)showSuspendedBlockedDesc, &tempBuffer);
148    memset(showSuspendedBlockedDesc, '\0', tempBuffer);
149
150    // showBlocked Description section
151    char showBlockedDesc[] = "showBlocked: Will display the name, class, state, suspended status, and
priority of every PCB in the blocked queue.\n";
152
153    tempBuffer = strlen(showBlockedDesc);
154
155    sys_req(WRITE, DEFAULT_DEVICE, (char *)showBlockedDesc, &tempBuffer);
156    memset(showBlockedDesc, '\0', tempBuffer);
157
158    // showAll Description section
159    char showAllDesc[] = "showReady: Will display the name, class, state, suspended status, and priority
of every PCB in all 4 queues.\n";
160
161    tempBuffer = strlen(showAllDesc);
162
163    sys_req(WRITE, DEFAULT_DEVICE, (char *)showAllDesc, &tempBuffer);
164    memset(showAllDesc, '\0', tempBuffer);
165
166    // quit Description section
167    char quitDesc[] = "quit: Allows the user to shut the system down.\n";
168
169    tempBuffer = strlen(quitDesc);
170
171    sys_req(WRITE, DEFAULT_DEVICE, (char *)quitDesc, &tempBuffer);
172    memset(quitDesc, '\0', tempBuffer);

```

```
173
174     return 0;
175 }
```

### 5.22.1.5 intToBCD()

```
unsigned char intToBCD (
    int test )
```

Definition at line 587 of file R1commands.c.

```
588 {
589
590     return (((test / 10) << 4) | (test % 10));
591 }
```

### 5.22.1.6 quit()

```
int quit ( )
```

Definition at line 605 of file R1commands.c.

```
606 {
607     int flag = 0;
608
609     char quitMsg[] = "Are you sure you want to shutdown? y/n\n";
610     int quitMsgLength = strlen(quitMsg);
611     sys_req(WRITE, DEFAULT_DEVICE, quitMsg, &quitMsgLength);
612
613     char quitAns[] = "\0\0";
614     int quitAnsLength = 1;
615     sys_req(READ, DEFAULT_DEVICE, quitAns, &quitAnsLength);
616     char answer = quitAns[0];
617
618     if (answer == 'y' || answer == 'Y')
619     {
620         flag = 1;
621     }
622     else if (answer == 'n' || answer == 'N')
623     {
624         flag = 0;
625     }
626     else
627     {
628         char error[] = "Invalid input!\n";
629         int errorLength = strlen(error);
630         sys_req(WRITE, DEFAULT_DEVICE, error, &errorLength);
631     }
632
633     return flag;
634 }
```

### 5.22.1.7 setDate()

```
int setDate ( )
```

Definition at line 379 of file R1commands.c.

```
380 {
381
382     int count = 4; // used to print year
383
384     char spacer[1] = "\n"; // used to space out terminal outputs
385     int spaceCount = 1;
```

```

386
388     char instruction1[] = "Please type the desired year. I.E.: yyyy.\n";
389     int length = strlen(instruction1);
390
391     sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
392     memset(instruction1, '\0', length);
393
394     char year[5] = "\0\0\0\0\0"; // year buffer
395
396     int flag = 0; // thrown if input is invalid
397
398     do
399     {
400         sys_req(READ, DEFAULT_DEVICE, year, &count);
401         if (atoi(year) > 0)
402         {
403
404             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
405             flag = 0;
406
407             char yearUpper[3] = "\0\0\0";
408             char yearLower[3] = "\0\0\0";
409
410             yearUpper[0] = year[0];
411             yearUpper[1] = year[1];
412             yearLower[0] = year[2];
413             yearLower[1] = year[3];
414
415             cli();
416
417             outb(0x70, 0x32); // Setting first byte year value
418             outb(0x71, intToBCD(atoi(yearUpper)));
419
420             outb(0x70, 0x09); // Setting second byte year value
421             outb(0x71, intToBCD(atoi(yearLower)));
422
423             sti();
424         }
425     else
426     {
427         char invalid[] = "Invalid year.\n";
428         int lengthInval = strlen(invalid);
429         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
430         sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
431         memset(invalid, '\0', lengthInval);
432         flag = 1;
433     }
434 } while (flag == 1);
435
437     char instruction2[] = "Please type the desired month. I.E.: mm.\n";
438     length = strlen(instruction2);
439
440     sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
441     memset(instruction2, '\0', length);
442
443     char month[4] = "\0\0\0\0";
444     count = 4; // used to print month
445
446     do
447     {
448         sys_req(READ, DEFAULT_DEVICE, month, &count);
449         if (atoi(month) < 13 && atoi(month) > 0)
450         {
451
452             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
453             flag = 0;
454
455             cli();
456
457             outb(0x70, 0x08); // Setting month value
458             outb(0x71, intToBCD(atoi(month)));
459
460             sti();
461         }
462     else
463     {
464         char invalid[] = "Invalid month.\n";
465         int lengthInval = strlen(invalid);
466         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
467         sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
468         memset(invalid, '\0', lengthInval);
469         flag = 1;
470     }
471 } while (flag == 1);
472
474     char instruction3[] = "Please type the desired day of month. I.E.: dd.\n";
475

```

```

476     length = strlen(instruction3);
477     sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
478     memset(instruction3, '\0', length);
479
480     char day[4] = "\0\0\0\0";
481     count = 4; // used to print day
482
483     do
484     {
485         sys_req(READ, DEFAULT_DEVICE, day, &count);
486         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
487         if ((atoi(year) % 4 == 0 && atoi(year) % 100 != 0) || atoi(year) % 400 == 0)
488         { // checking for leap year
489
490             char leapYear[] = "This is a leap year. February has 29 days.\n";
491             length = strlen(leapYear);
492
493             sys_req(WRITE, DEFAULT_DEVICE, leapYear, &length);
494             memset(leapYear, '\0', length);
495
496             if ((atoi(month) == 1 || atoi(month) == 3 || atoi(month) == 5 || atoi(month) == 7 ||
atoi(month) == 8 || atoi(month) == 10 || atoi(month) == 12) && atoi(day) > 31)
497             {
498                 flag = 1;
499                 char invalid[] = "Invalid day.\n";
500                 length = strlen(invalid);
501                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
502                 memset(invalid, '\0', length);
503             }
504             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
atoi(day) > 30)
505             {
506                 flag = 1;
507                 char invalid[] = "Invalid day.\n";
508                 length = strlen(invalid);
509                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
510                 memset(invalid, '\0', length);
511             }
512             else if ((atoi(month) == 2) && atoi(day) > 29)
513             {
514                 flag = 1;
515                 char invalid[] = "Invalid day.\n";
516                 length = strlen(invalid);
517                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
518                 memset(invalid, '\0', length);
519             }
520             else
521             {
522
523                 flag = 0;
524                 cli();
525
526                 outb(0x70, 0x07); // Setting day of month value
527                 outb(0x71, intToBCD(atoi(day)));
528
529                 sti();
530             }
531         }
532         else if (atoi(year) % 4 != 0 || atoi(year) % 400 != 0)
533         { // checking for leap year
534
535             char noLeap[] = "This is not a leap year.\n";
536             length = strlen(noLeap);
537             sys_req(WRITE, DEFAULT_DEVICE, noLeap, &length);
538             memset(noLeap, '\0', length);
539
540             if ((atoi(month) == 1 || atoi(month) == 3 || atoi(month) == 5 || atoi(month) == 7 ||
atoi(month) == 8 || atoi(month) == 10 || atoi(month) == 12) && atoi(day) > 31)
541             {
542                 flag = 1;
543                 char invalid[] = "Invalid day.\n";
544                 length = strlen(invalid);
545                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
546                 memset(invalid, '\0', length);
547             }
548             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
atoi(day) > 30)
549             {
550                 flag = 1;
551                 char invalid[] = "Invalid day.\n";
552                 length = strlen(invalid);
553                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
554                 memset(invalid, '\0', length);
555             }
556             else if ((atoi(month) == 2) && atoi(day) > 28)
557             {
558                 flag = 1;

```

```

559         char invalid[] = "Invalid day.\n";
560         length = strlen(invalid);
561         sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
562         memset(invalid, '\0', length);
563     }
564     else
565     {
566
567         cli();
568
569         outb(0x70, 0x07); // Setting day of month value
570         outb(0x71, intToBCD(atoi(day)));
571
572         sti();
573     }
574 }
575
576 } while (flag == 1);
577
578 char exitMessage[] = "The date has been set.\n";
579 int exitLength = strlen(exitMessage);
580 sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
581 memset(exitMessage, '\0', exitLength);
582 memset(spacer, '\0', spaceCount);
583
584 return 0;
585 }

```

### 5.22.1.8 setTime()

```
int setTime ( )
```

Definition at line 221 of file R1commands.c.

```

222 {
223
224     int count = 4; // counter for printing
225
226     char spacer[1] = "\n"; // used to space out terminal outputs
227     int spaceCount = 1;
228
229     char instruction1[] = "Please type the desired hours. I.E.: hh.\n";
230
231     int length = strlen(instruction1);
232
233     sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
234     memset(instruction1, '\0', length);
235
236     char hour[4] = "\0\0\0\0";
237
238     int flag = 0;
239
240     do
241     {
242         sys_req(READ, DEFAULT_DEVICE, hour, &count);
243         if (atoi(hour) < 24 && atoi(hour) >= 0)
244         {
245             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
246             flag = 0;
247         }
248         else
249         {
250             char invalid[] = "Invalid hours.\n";
251             int lengthInval = strlen(invalid);
252             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
253             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
254             memset(invalid, '\0', lengthInval);
255             flag = 1;
256         }
257     } while (flag == 1);
258
259     char instruction2[] = "Please type the desired minutes. I.E.: mm.\n";
260
261     length = strlen(instruction2);
262
263     sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
264     memset(instruction2, '\0', length);
265
266     char minute[4] = "\0\0\0\0";

```



```

270
271     do
272     {
273         sys_req(READ, DEFAULT_DEVICE, minute, &count);
274         if (atoi(minute) < 60 && atoi(minute) >= 0)
275         {
276             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
277             flag = 0;
278         }
279     }
280     else
281     {
282         char invalid[] = "Invalid minutes.\n";
283         int lengthInval = strlen(invalid);
284         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
285         sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
286         memset(invalid, '\\0', lengthInval);
287         flag = 1;
288     }
289 } while (flag == 1);
290
292 char instruction3[] = "Please type the desired seconds. I.E.: ss.\n";
293
294 length = strlen(instruction3);
295
296 sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
297 memset(instruction3, '\\0', length);
298
299 char second[4] = "\\0\\0\\n\\0";
300
301 do
302 {
303     sys_req(READ, DEFAULT_DEVICE, second, &count);
304     if (atoi(second) < 60 && atoi(second) >= 0)
305     {
306         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
307         flag = 0;
308     }
309 }
310 else
311 {
312     char invalid[] = "Invalid seconds.\n";
313     int lengthInval = strlen(invalid);
314     sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
315     sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
316     memset(invalid, '\\0', lengthInval);
317     flag = 1;
318 }
319 } while (flag == 1);
320
321 cli();
322
323 outb(0x70, 0x04); // Hour
324 outb(0x71, intToBCD(atoi(hour)));
325
326 outb(0x70, 0x02); // Minute
327 outb(0x71, intToBCD(atoi(minute)));
328
329 outb(0x70, 0x00); // Second
330 outb(0x71, intToBCD(atoi(second)));
331
332 sti();
333
334 char exitMessage[] = "The time has been set.\n";
335 int exitLength = strlen(exitMessage);
336 sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
337 memset(exitMessage, '\\0', exitLength);
338 memset(spacer, '\\0', spaceCount);
339
340 return 0;
341 }

```

### 5.22.1.9 version()

```
int version ( )
```

Definition at line 177 of file R1commands.c.

```

178 {
179

```

```

180     char version[] = "Version 2.0\n";
181
182     int tempBuffer = strlen(version);
183
184     sys_req(WRITE, DEFAULT_DEVICE, (char *)version, &tempBuffer);
185     memset(version, '\0', tempBuffer);
186
187     return 0;
188 }

```

## 5.23 modules/R1/R1commands.h File Reference

### Functions

- void [help](#) ()
- void [version](#) ()
- void [getTime](#) ()
- void [setTime](#) ()
- void [getDate](#) ()
- void [setDate](#) ()
- unsigned int [change\\_int\\_to\\_binary](#) (int test)
- int [BCDtoChar](#) (unsigned char test, char \*buffer)
- int [quit](#) ()

### 5.23.1 Function Documentation

#### 5.23.1.1 BCDtoChar()

```

int BCDtoChar (
    unsigned char test,
    char * buffer )

```

Definition at line 593 of file R1commands.c.

```

594 {
595
596     int val1 = (test / 16);
597     int val2 = (test % 16);
598
599     buffer[0] = val1 + '\0';
600     buffer[1] = val2 + '\0';
601
602     return 0;
603 }

```

#### 5.23.1.2 change\_int\_to\_binary()

```

unsigned int change_int_to_binary (
    int test )

```

### 5.23.1.3 getDate()

```
void getDate ( )
```

Definition at line 343 of file R1commands.c.

```
344 {
345
346     char buffer[4] = " \0\0\0\0";
347     int count = 4;
348     char divider = '/';
349     char newLine[1] = "\n";
350     int newLineCount = 1;
351
352     outb(0x70, 0x07); // getting Day of month value
353     BCDtoChar(inb(0x71), buffer);
354     buffer[2] = divider;
355     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
356     memset(buffer, ' \0', count);
357
358     outb(0x70, 0x08); // getting Month value
359     BCDtoChar(inb(0x71), buffer);
360     buffer[2] = divider;
361     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
362     memset(buffer, ' \0', count);
363
364     outb(0x70, 0x32); // getting Year value second byte
365     BCDtoChar(inb(0x71), buffer);
366     buffer[2] = ' \0';
367     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
368     memset(buffer, ' \0', count);
369
370     outb(0x70, 0x09); // getting Year value first byte
371     BCDtoChar(inb(0x71), buffer);
372     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
373     memset(buffer, ' \0', count);
374
375     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
376     memset(newLine, ' \0', newLineCount);
377 }
```

### 5.23.1.4 getTime()

```
void getTime ( )
```

Definition at line 190 of file R1commands.c.

```
191 {
192
193     char buffer[4] = " \0\0\0\0";
194     int count = 4;
195     char divider = ':';
196     char newLine[1] = "\n";
197     int newLineCount = 1;
198
199     outb(0x70, 0x04); // getting Hour value
200     BCDtoChar(inb(0x71), buffer);
201     buffer[2] = divider;
202     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
203     memset(buffer, ' \0', count);
204
205     outb(0x70, 0x02); // getting Minute value
206     BCDtoChar(inb(0x71), buffer);
207     buffer[2] = divider;
208     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
209     memset(buffer, ' \0', count);
210
211     outb(0x70, 0x00); // getting Second value
212     BCDtoChar(inb(0x71), buffer);
213     buffer[2] = ' \0';
214     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
215     memset(buffer, ' \0', count);
216
217     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
218     memset(newLine, ' \0', newLineCount);
219 }
```

### 5.23.1.5 help()

```
void help ( )
```

Definition at line 11 of file R1commands.c.

```
12 {
13
14     // Help Description section
15     char helpDesc[] = "help: Returns basic command information.\n";
16
17     int tempBuffer = strlen(helpDesc);
18
19     sys_req(WRITE, DEFAULT_DEVICE, (char *)helpDesc, &tempBuffer);
20     memset(helpDesc, '\0', tempBuffer);
21
22     // Version Description section
23     char versionDesc[] = "version: Returns the current version of the software.\n";
24
25     tempBuffer = strlen(versionDesc);
26
27     sys_req(WRITE, DEFAULT_DEVICE, (char *)versionDesc, &tempBuffer);
28     memset(versionDesc, '\0', tempBuffer);
29
30     // getTime Description section
31     char getTimeDesc[] = "getTime: Returns the current set time.\n";
32
33     tempBuffer = strlen(getTimeDesc);
34
35     sys_req(WRITE, DEFAULT_DEVICE, (char *)getTimeDesc, &tempBuffer);
36     memset(getTimeDesc, '\0', tempBuffer);
37
38     // setTime Description section
39     char setTimeDesc[] = "setTime: Allows the user to change the set time.\n";
40
41     tempBuffer = strlen(setTimeDesc);
42
43     sys_req(WRITE, DEFAULT_DEVICE, (char *)setTimeDesc, &tempBuffer);
44     memset(setTimeDesc, '\0', tempBuffer);
45
46     // getDate Description section
47     char getDateDesc[] = "getDate: Returns the current set date.\n";
48
49     tempBuffer = strlen(getDateDesc);
50
51     sys_req(WRITE, DEFAULT_DEVICE, (char *)getDateDesc, &tempBuffer);
52     memset(getDateDesc, '\0', tempBuffer);
53
54     // setDate Description section
55     char setDateDesc[] = "setDate: Allows the user to change the set date.\n";
56
57     tempBuffer = strlen(setDateDesc);
58
59     sys_req(WRITE, DEFAULT_DEVICE, (char *)setDateDesc, &tempBuffer);
60     memset(setDateDesc, '\0', tempBuffer);
61
62     // createPCB Description section
63     char createPCBDesc[] = "createPCB: Will create a PCB and put it into the ready queue by default.\n";
64
65     tempBuffer = strlen(createPCBDesc);
66
67     sys_req(WRITE, DEFAULT_DEVICE, (char *)createPCBDesc, &tempBuffer);
68     memset(createPCBDesc, '\0', tempBuffer);
69
70     // deletePCB Description section
71     char deletePCBDesc[] = "deletePCB: Will delete a specific PCB from what ever queue it is in. \n";
72
73     tempBuffer = strlen(deletePCBDesc);
74
75     sys_req(WRITE, DEFAULT_DEVICE, (char *)deletePCBDesc, &tempBuffer);
76     memset(deletePCBDesc, '\0', tempBuffer);
77
78     // blockPCB Description section
79     char blockPCBDesc[] = "blockPCB: Will change a specific PCB's state to blocked. \n";
80
81     tempBuffer = strlen(blockPCBDesc);
82
83     sys_req(WRITE, DEFAULT_DEVICE, (char *)blockPCBDesc, &tempBuffer);
84     memset(blockPCBDesc, '\0', tempBuffer);
85
86     // unblockPCB Description section
87     char unblockPCBDesc[] = "unblockPCB: Will change a specific PCB's state to ready. \n";
88
89     tempBuffer = strlen(unblockPCBDesc);
90
91     sys_req(WRITE, DEFAULT_DEVICE, (char *)unblockPCBDesc, &tempBuffer);
```

```

92     memset(unblockPCBDesc, '\0', tempBuffer);
93
94     // suspendPCB Description section
95     char suspendPCBDesc[] = "suspendPCB: Will suspend a specific PCB. \n";
96
97     tempBuffer = strlen(suspendPCBDesc);
98
99     sys_req(WRITE, DEFAULT_DEVICE, (char *)suspendPCBDesc, &tempBuffer);
100    memset(suspendPCBDesc, '\0', tempBuffer);
101
102    // resumePCB Description section
103    char resumePCBDesc[] = "resumePCB: Will unsuspend a specific PCB. \n";
104
105    tempBuffer = strlen(resumePCBDesc);
106
107    sys_req(WRITE, DEFAULT_DEVICE, (char *)resumePCBDesc, &tempBuffer);
108    memset(resumePCBDesc, '\0', tempBuffer);
109
110    // setPCBPRIORITY Description section
111    char setPCBPRIORITYDesc[] = "setPCBPRIORITY: Will change the priority of a specific PCB. \n";
112
113    tempBuffer = strlen(setPCBPRIORITYDesc);
114
115    sys_req(WRITE, DEFAULT_DEVICE, (char *)setPCBPRIORITYDesc, &tempBuffer);
116    memset(setPCBPRIORITYDesc, '\0', tempBuffer);
117
118    // showPCB Description section
119    char showPCBDesc[] = "showPCB: Will display the name, class, state, suspended status, and priority
of a specific PCB. \n";
120
121    tempBuffer = strlen(showPCBDesc);
122
123    sys_req(WRITE, DEFAULT_DEVICE, (char *)showPCBDesc, &tempBuffer);
124    memset(showPCBDesc, '\0', tempBuffer);
125
126    // showReady Description section
127    char showReadyDesc[] = "showReady: Will display the name, class, state, suspended status, and
priority of every PCB in the ready queue.\n";
128
129    tempBuffer = strlen(showReadyDesc);
130
131    sys_req(WRITE, DEFAULT_DEVICE, (char *)showReadyDesc, &tempBuffer);
132    memset(showReadyDesc, '\0', tempBuffer);
133
134    // showSuspendedReady Description section
135    char showSuspendedReadyDesc[] = "showSuspendedReady: Will display the name, class, state, suspended
status, and priority of every PCB in the suspended ready queue.\n";
136
137    tempBuffer = strlen(showSuspendedReadyDesc);
138
139    sys_req(WRITE, DEFAULT_DEVICE, (char *)showSuspendedReadyDesc, &tempBuffer);
140    memset(showSuspendedReadyDesc, '\0', tempBuffer);
141
142    // showSuspendedBlocked Description section
143    char showSuspendedBlockedDesc[] = "showSuspendedBlocked: Will display the name, class, state,
suspended status, and priority of every PCB in the suspended blocked queue.\n";
144
145    tempBuffer = strlen(showSuspendedBlockedDesc);
146
147    sys_req(WRITE, DEFAULT_DEVICE, (char *)showSuspendedBlockedDesc, &tempBuffer);
148    memset(showSuspendedBlockedDesc, '\0', tempBuffer);
149
150    // showBlocked Description section
151    char showBlockedDesc[] = "showBlocked: Will display the name, class, state, suspended status, and
priority of every PCB in the blocked queue.\n";
152
153    tempBuffer = strlen(showBlockedDesc);
154
155    sys_req(WRITE, DEFAULT_DEVICE, (char *)showBlockedDesc, &tempBuffer);
156    memset(showBlockedDesc, '\0', tempBuffer);
157
158    // showAll Description section
159    char showAllDesc[] = "showReady: Will display the name, class, state, suspended status, and priority
of every PCB in all 4 queues.\n";
160
161    tempBuffer = strlen(showAllDesc);
162
163    sys_req(WRITE, DEFAULT_DEVICE, (char *)showAllDesc, &tempBuffer);
164    memset(showAllDesc, '\0', tempBuffer);
165
166    // quit Description section
167    char quitDesc[] = "quit: Allows the user to shut the system down.\n";
168
169    tempBuffer = strlen(quitDesc);
170
171    sys_req(WRITE, DEFAULT_DEVICE, (char *)quitDesc, &tempBuffer);
172    memset(quitDesc, '\0', tempBuffer);

```

```

173
174     return 0;
175 }

```

### 5.23.1.6 quit()

```
int quit ( )
```

Definition at line 605 of file R1commands.c.

```

606 {
607     int flag = 0;
608
609     char quitMsg[] = "Are you sure you want to shutdown? y/n\n";
610     int quitMsgLength = strlen(quitMsg);
611     sys_req(WRITE, DEFAULT_DEVICE, quitMsg, &quitMsgLength);
612
613     char quitAns[] = "\0\0";
614     int quitAnsLength = 1;
615     sys_req(READ, DEFAULT_DEVICE, quitAns, &quitAnsLength);
616     char answer = quitAns[0];
617
618     if (answer == 'y' || answer == 'Y')
619     {
620         flag = 1;
621     }
622     else if (answer == 'n' || answer == 'N')
623     {
624         flag = 0;
625     }
626     else
627     {
628         char error[] = "Invalid input!\n";
629         int errorLength = strlen(error);
630         sys_req(WRITE, DEFAULT_DEVICE, error, &errorLength);
631     }
632
633     return flag;
634 }

```

### 5.23.1.7 setDate()

```
void setDate ( )
```

Definition at line 379 of file R1commands.c.

```

380 {
381
382     int count = 4; // used to print year
383
384     char spacer[1] = "\n"; // used to space out terminal outputs
385     int spaceCount = 1;
386
387     char instruction1[] = "Please type the desired year. I.E.: yyyy.\n";
388     int length = strlen(instruction1);
389
390     sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
391     memset(instruction1, '\0', length);
392
393     char year[5] = "\0\0\0\0\0"; // year buffer
394
395     int flag = 0; // thrown if input is invalid
396
397     do
398     {
399         sys_req(READ, DEFAULT_DEVICE, year, &count);
400         if (atoi(year) > 0)
401         {
402             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
403             flag = 0;
404         }
405     }
406 }

```

```

407         char yearUpper[3] = "\0\0\0";
408         char yearLower[3] = "\0\0\0";
409
410         yearUpper[0] = year[0];
411         yearUpper[1] = year[1];
412         yearLower[0] = year[2];
413         yearLower[1] = year[3];
414
415         cli();
416
417         outb(0x70, 0x32); // Setting first byte year value
418         outb(0x71, intToBCD(atoi(yearUpper)));
419
420         outb(0x70, 0x09); // Setting second byte year value
421         outb(0x71, intToBCD(atoi(yearLower)));
422
423         sti();
424     }
425     else
426     {
427         char invalid[] = "Invalid year.\n";
428         int lengthInval = strlen(invalid);
429         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
430         sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
431         memset(invalid, '\0', lengthInval);
432         flag = 1;
433     }
434 } while (flag == 1);
435
436 char instruction2[] = "Please type the desired month. I.E.: mm.\n";
437 length = strlen(instruction2);
438
439 sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
440 memset(instruction2, '\0', length);
441
442 char month[4] = "\0\0\n\0";
443 count = 4; // used to print month
444
445 do
446 {
447     sys_req(READ, DEFAULT_DEVICE, month, &count);
448     if (atoi(month) < 13 && atoi(month) > 0)
449     {
450
451         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
452         flag = 0;
453
454         cli();
455
456         outb(0x70, 0x08); // Setting month value
457         outb(0x71, intToBCD(atoi(month)));
458
459         sti();
460     }
461     else
462     {
463         char invalid[] = "Invalid month.\n";
464         int lengthInval = strlen(invalid);
465         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
466         sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
467         memset(invalid, '\0', lengthInval);
468         flag = 1;
469     }
470 } while (flag == 1);
471
472 char instruction3[] = "Please type the desired day of month. I.E.: dd.\n";
473
474 length = strlen(instruction3);
475 sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
476 memset(instruction3, '\0', length);
477
478 char day[4] = "\0\0\n\0";
479 count = 4; // used to print day
480
481 do
482 {
483     sys_req(READ, DEFAULT_DEVICE, day, &count);
484     sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
485     if ((atoi(year) % 4 == 0 && atoi(year) % 100 != 0) || atoi(year) % 400 == 0)
486     { // checking for leap year
487
488         char leapYear[] = "This is a leap year. February has 29 days.\n";
489         length = strlen(leapYear);
490
491         sys_req(WRITE, DEFAULT_DEVICE, leapYear, &length);
492         memset(leapYear, '\0', length);
493     }
494 }
495

```

```

496         if ((atoi(month) == 1 || atoi(month) == 3 || atoi(month) == 5 || atoi(month) == 7 ||
atoi(month) == 8 || atoi(month) == 10 || atoi(month) == 12) && atoi(day) > 31)
497         {
498             flag = 1;
499             char invalid[] = "Invalid day.\n";
500             length = strlen(invalid);
501             sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
502             memset(invalid, '\0', length);
503         }
504         else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
atoi(day) > 30)
505         {
506             flag = 1;
507             char invalid[] = "Invalid day.\n";
508             length = strlen(invalid);
509             sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
510             memset(invalid, '\0', length);
511         }
512         else if ((atoi(month) == 2) && atoi(day) > 29)
513         {
514             flag = 1;
515             char invalid[] = "Invalid day.\n";
516             length = strlen(invalid);
517             sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
518             memset(invalid, '\0', length);
519         }
520         else
521         {
522             flag = 0;
523             cli();
524
525             outb(0x70, 0x07); // Setting day of month value
526             outb(0x71, intToBCD(atoi(day)));
527
528             sti();
529         }
530     }
531 }
532 else if (atoi(year) % 4 != 0 || atoi(year) % 400 != 0)
533 { // checking for leap year
534
535     char noLeap[] = "This is not a leap year.\n";
536     length = strlen(noLeap);
537     sys_req(WRITE, DEFAULT_DEVICE, noLeap, &length);
538     memset(noLeap, '\0', length);
539
540     if ((atoi(month) == 1 || atoi(month) == 3 || atoi(month) == 5 || atoi(month) == 7 ||
atoi(month) == 8 || atoi(month) == 10 || atoi(month) == 12) && atoi(day) > 31)
541     {
542         flag = 1;
543         char invalid[] = "Invalid day.\n";
544         length = strlen(invalid);
545         sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
546         memset(invalid, '\0', length);
547     }
548     else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
atoi(day) > 30)
549     {
550         flag = 1;
551         char invalid[] = "Invalid day.\n";
552         length = strlen(invalid);
553         sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
554         memset(invalid, '\0', length);
555     }
556     else if ((atoi(month) == 2) && atoi(day) > 28)
557     {
558         flag = 1;
559         char invalid[] = "Invalid day.\n";
560         length = strlen(invalid);
561         sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
562         memset(invalid, '\0', length);
563     }
564     else
565     {
566         cli();
567
568         outb(0x70, 0x07); // Setting day of month value
569         outb(0x71, intToBCD(atoi(day)));
570
571         sti();
572     }
573 }
574 }
575 } while (flag == 1);
576
577 char exitMessage[] = "The date has been set.\n";

```



```

579     int exitLength = strlen(exitMessage);
580     sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
581     memset(exitMessage, '\\0', exitLength);
582     memset(spacer, '\\0', spaceCount);
583
584     return 0;
585 }

```

### 5.23.1.8 setTime()

```
void setTime ( )
```

Definition at line 221 of file R1commands.c.

```

222 {
223
224     int count = 4; // counter for printing
225
226     char spacer[1] = "\\n"; // used to space out terminal outputs
227     int spaceCount = 1;
228
229     char instruction1[] = "Please type the desired hours. I.E.: hh.\\n";
230
231     int length = strlen(instruction1);
232
233     sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
234     memset(instruction1, '\\0', length);
235
236     char hour[4] = "\\0\\0\\n\\0";
237
238     int flag = 0;
239
240     do
241     {
242         sys_req(READ, DEFAULT_DEVICE, hour, &count);
243         if (atoi(hour) < 24 && atoi(hour) >= 0)
244         {
245             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
246             flag = 0;
247         }
248         else
249         {
250             char invalid[] = "Invalid hours.\\n";
251             int lengthInval = strlen(invalid);
252             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
253             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
254             memset(invalid, '\\0', lengthInval);
255             flag = 1;
256         }
257     } while (flag == 1);
258
259     char instruction2[] = "Please type the desired minutes. I.E.: mm.\\n";
260
261     length = strlen(instruction2);
262
263     sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
264     memset(instruction2, '\\0', length);
265
266     char minute[4] = "\\0\\0\\n\\0";
267
268     do
269     {
270         sys_req(READ, DEFAULT_DEVICE, minute, &count);
271         if (atoi(minute) < 60 && atoi(minute) >= 0)
272         {
273             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
274             flag = 0;
275         }
276         else
277         {
278             char invalid[] = "Invalid minutes.\\n";
279             int lengthInval = strlen(invalid);
280             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
281             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
282             memset(invalid, '\\0', lengthInval);
283             flag = 1;
284         }
285     } while (flag == 1);
286 }

```

```

290
291 char instruction3[] = "Please type the desired seconds. I.E.: ss.\n";
292
293 length = strlen(instruction3);
294
295 sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
296 memset(instruction3, '\0', length);
297
298 char second[4] = "\0\0\0\0";
299
300 do
301 {
302     sys_req(READ, DEFAULT_DEVICE, second, &count);
303     if (atoi(second) < 60 && atoi(second) >= 0)
304     {
305         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
306         flag = 0;
307     }
308     else
309     {
310         char invalid[] = "Invalid seconds.\n";
311         int lengthInval = strlen(invalid);
312         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
313         sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
314         memset(invalid, '\0', lengthInval);
315         flag = 1;
316     }
317 } while (flag == 1);
318
319 cli();
320
321 outb(0x70, 0x04); // Hour
322 outb(0x71, intToBCD(atoi(hour)));
323
324 outb(0x70, 0x02); // Minute
325 outb(0x71, intToBCD(atoi(minute)));
326
327 outb(0x70, 0x00); // Second
328 outb(0x71, intToBCD(atoi(second)));
329
330 sti();
331
332 char exitMessage[] = "The time has been set.\n";
333 int exitLength = strlen(exitMessage);
334 sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
335 memset(exitMessage, '\0', exitLength);
336 memset(spacer, '\0', spaceCount);
337
338 return 0;
339
340 }

```

### 5.23.1.9 version()

```
void version ( )
```

Definition at line 177 of file R1commands.c.

```

178 {
179
180     char version[] = "Version 2.0\n";
181
182     int tempBuffer = strlen(version);
183
184     sys_req(WRITE, DEFAULT_DEVICE, (char *)version, &tempBuffer);
185     memset(version, '\0', tempBuffer);
186
187     return 0;
188 }

```

## 5.24 modules/R2/R2\_Internal\_Functions\_And\_Structures.c File Reference

```

#include <string.h>
#include "../mpx_supt.h"

```

```
#include "R2_Internal_Functions_And_Structures.h"
```

## Functions

- [PCB \\*allocatePCB \(\)](#)
- [int freePCB \(PCB \\*PCB\\_to\\_free\)](#)
- [PCB \\*setupPCB \(char \\*processName, unsigned char processClass, int processPriority\)](#)
- [PCB \\*findPCB \(char \\*processName\)](#)
- [void insertPCB \(PCB \\*PCB\\_to\\_insert\)](#)
- [int removePCB \(PCB \\*PCB\\_to\\_remove\)](#)
- [void allocateQueues \(\)](#)
- [queue \\*getReady \(\)](#)
- [queue \\*getBlocked \(\)](#)
- [queue \\*getSuspendedReady \(\)](#)
- [queue \\*getSuspendedBlocked \(\)](#)

## Variables

- [queue \\*ready](#)
- [queue \\*blocked](#)
- [queue \\*suspendedReady](#)
- [queue \\*suspendedBlocked](#)

### 5.24.1 Function Documentation

#### 5.24.1.1 allocatePCB()

```
PCB* allocatePCB ( )
```

Definition at line 14 of file R2\_Internal\_Functions\_And\_Structures.c.

```
15 {
16     //COLTON WILL PROGRAM THIS FUNCTION
17
18     //allocatePCB() will use sys_alloc_mem() to allocate memory for a new PCB, possible including the
19     //stack, and perform any reasonable initialization.
20     PCB *newPCB = (PCB *)sys_alloc_mem(sizeof(PCB));
21
22     char name[20] = "newPCB";
23     strcpy(newPCB->processName, name);
24
25     newPCB->suspendedStatus = 1;
26     newPCB->runningStatus = -1;
27     newPCB->stackTop = (newPCB->stackTop + 1024);
28     newPCB->stackBase = newPCB->stackBase;
29     newPCB->priority = 0;
30
31     // Setting the PCBs prev and next PCB
32     newPCB->nextPCB = NULL;
33     newPCB->prevPCB = NULL;
34
35     newPCB->processClass = NULL;
36
37     return newPCB;
38 }
```

### 5.24.1.2 allocateQueues()

```
void allocateQueues ( )
```

Definition at line 377 of file R2\_Internal\_Functions\_And\_Structures.c.

```
378 {
379     ready = sys_alloc_mem(sizeof(queue));
380     ready->count = 0;
381     ready->head = NULL;
382     ready->tail = NULL;
383     blocked = sys_alloc_mem(sizeof(queue));
384     blocked->count = 0;
385     blocked->head = NULL;
386     blocked->tail = NULL;
387     suspendedReady = sys_alloc_mem(sizeof(queue));
388     suspendedReady->count = 0;
389     suspendedReady->head = NULL;
390     suspendedReady->tail = NULL;
391     suspendedBlocked = sys_alloc_mem(sizeof(queue));
392     suspendedBlocked->count = 0;
393     suspendedBlocked->head = NULL;
394     suspendedBlocked->tail = NULL;
395 }
```

### 5.24.1.3 findPCB()

```
PCB* findPCB (
    char *processName )
```

Definition at line 77 of file R2\_Internal\_Functions\_And\_Structures.c.

```
78 {
79     // ANASTASE WILL PROGRAM THIS FUNCTION
80
81     //findPCB() will search all queues for a process with a given name.
82
83     if (strlen(processName) > 20)
84     {
85
86         char error_message[30] = "Invalid process name.\n";
87         int error_size = strlen(error_message);
88         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
89         return NULL;
90         //return cz we have to stop if the process name is too long
91     }
92     else
93     {
94         PCB *tempPCB = ready->head;
95         int value = 0;
96         while (value <= ready->count)
97         {
98             if (strcmp(tempPCB->processName, processName) == 0)
99             {
100                 return tempPCB;
101             }
102             else
103             {
104                 tempPCB = tempPCB->nextPCB;
105                 value++;
106             }
107         }
108
109         tempPCB = blocked->head;
110         value = 0;
111         while (value <= blocked->count)
112         {
113             if (strcmp(tempPCB->processName, processName) == 0)
114             {
115                 return tempPCB;
116             }
117             else
118             {
119                 tempPCB = tempPCB->nextPCB;
120                 value++;
121             }
122         }
123     }
```

```

123
124     tempPCB = suspendedBlocked->head;
125     value = 0;
126     while (value <= suspendedBlocked->count)
127     {
128         if (strcmp(tempPCB->processName, processName) == 0)
129         {
130             return tempPCB;
131         }
132         else
133         {
134             tempPCB = tempPCB->nextPCB;
135             value++;
136         }
137     }
138
139     tempPCB = suspendedReady->head;
140     value = 0;
141     while (value <= suspendedReady->count)
142     {
143         if (strcmp(tempPCB->processName, processName) == 0)
144         {
145             return tempPCB;
146         }
147         else
148         {
149             tempPCB = tempPCB->nextPCB;
150             value++;
151         }
152     }
153
154     return NULL;
155 }
156 }

```

#### 5.24.1.4 freePCB()

```

int freePCB (
    PCB *PCB_to_free )

```

Definition at line 39 of file R2\_Internal\_Functions\_And\_Structures.c.

```

40 {
41     // ANASTASE WILL PROGRAM THIS FUNCTION
42
43     //freePCB() will use sys_free_mem() to free all memory associated with a given PCB (the stack, the
44     PCB itself, etc.)
45     return sys_free_mem(PCB_to_free);
46 }

```

#### 5.24.1.5 getBlocked()

```

queue* getBlocked ( )

```

Definition at line 402 of file R2\_Internal\_Functions\_And\_Structures.c.

```

403 {
404     return blocked;
405 }

```

### 5.24.1.6 getReady()

```
queue* getReady ( )
```

Definition at line 397 of file R2\_Internal\_Functions\_And\_Structures.c.

```
398 {
399     return ready;
400 }
```

### 5.24.1.7 getSuspendedBlocked()

```
queue* getSuspendedBlocked ( )
```

Definition at line 412 of file R2\_Internal\_Functions\_And\_Structures.c.

```
413 {
414     return suspendedBlocked;
415 }
```

### 5.24.1.8 getSuspendedReady()

```
queue* getSuspendedReady ( )
```

Definition at line 407 of file R2\_Internal\_Functions\_And\_Structures.c.

```
408 {
409     return suspendedReady;
410 }
```

### 5.24.1.9 insertPCB()

```
void insertPCB (
    PCB *PCB_to_insert )
```

Definition at line 158 of file R2\_Internal\_Functions\_And\_Structures.c.

```
159 {
160     //BENJAMIN WILL PROGRAM THIS FUNCTION
161
162     //insertPCB() will insert a PCB into the appropriate queue.
163     //Note: The ready queue is a priority queue and the blocked queue is a FIFO queue.
164
165     if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 1)
166     { // Insert into ready queue
167         PCB *tempPtr = ready->head;
168
169         if (tempPtr != NULL)
170         {
171             int temp = 0;
172             while (temp <= ready->count)
173             {
174                 if (PCB_to_insert->priority < tempPtr->priority)
175                 {
176                     tempPtr = tempPtr->nextPCB;
177                 }
178                 else if (PCB_to_insert->priority >= tempPtr->priority)
179                 {
180                     PCB_to_insert->nextPCB = tempPtr;
181                     PCB_to_insert->prevPCB = tempPtr->prevPCB;
182                     tempPtr->prevPCB = PCB_to_insert;
183                 }
184             }
185         }
186     }
187 }
```

```

184         else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
185         {
186             tempPtr->nextPCB = PCB_to_insert;
187             PCB_to_insert->prevPCB = tempPtr;
188             ready->tail = PCB_to_insert;
189         }
190         temp++;
191     }
192     ready->count++;
193 }
194 else
195 {
196     ready->count++;
197     ready->head = PCB_to_insert;
198     ready->tail = PCB_to_insert;
199 }
200 }
201 else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
202 { // Insert into suspended ready queue
203     PCB *tempPtr = suspendedReady->head;
204
205     if (tempPtr != NULL)
206     {
207         int temp = 0;
208         while (temp <= suspendedReady->count)
209         {
210             if (PCB_to_insert->priority < tempPtr->priority)
211             {
212                 tempPtr = tempPtr->nextPCB;
213             }
214             else if (PCB_to_insert->priority >= tempPtr->priority)
215             {
216                 PCB_to_insert->nextPCB = tempPtr;
217                 PCB_to_insert->prevPCB = tempPtr->prevPCB;
218                 tempPtr->prevPCB = PCB_to_insert;
219             }
220             else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
221             {
222                 tempPtr->nextPCB = PCB_to_insert;
223                 PCB_to_insert->prevPCB = tempPtr;
224                 suspendedReady->tail = PCB_to_insert;
225             }
226             temp++;
227         }
228         suspendedReady->count++;
229     }
230     else
231     {
232         suspendedReady->count++;
233         suspendedReady->head = PCB_to_insert;
234         suspendedReady->tail = PCB_to_insert;
235     }
236 }
237 else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 1)
238 { // Insert into blocked queue
239     PCB *tempPtr = blocked->tail;
240
241     tempPtr->nextPCB = PCB_to_insert;
242     PCB_to_insert->prevPCB = tempPtr;
243     blocked->tail = PCB_to_insert;
244 }
245 else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
246 { // Insert into suspended blocked queue
247     PCB *tempPtr = suspendedBlocked->tail;
248
249     tempPtr->nextPCB = PCB_to_insert;
250     PCB_to_insert->prevPCB = tempPtr;
251     suspendedBlocked->tail = PCB_to_insert;
252 }
253 }

```

#### 5.24.1.10 removePCB()

```

int removePCB (
    PCB *PCB_to_remove )

```

Definition at line 255 of file R2\_Internal\_Functions\_And\_Structures.c.

```

256 {

```

```

257 //BENJAMIN WILL PROGRAM THIS FUNCTION
258
259 //removePCB() will remove a PCB from the queue in which it is currently stored.
260
261 PCB *removedPCB = findPCB(PCB_to_remove->processName);
262 if (removedPCB == NULL)
263 {
264     return 1;
265 }
266 else if (removedPCB == ready->head)
267 {
268     PCB *removedNext = removedPCB->nextPCB;
269
270     ready->head = removedNext;
271     removedNext->prevPCB = NULL;
272     removedPCB->nextPCB = NULL;
273     ready->count--;
274     return 0;
275 }
276 else if (removedPCB == blocked->head)
277 {
278     PCB *removedNext = removedPCB->nextPCB;
279     blocked->head = removedNext;
280     removedNext->prevPCB = NULL;
281     removedPCB->nextPCB = NULL;
282     blocked->count--;
283     return 0;
284 }
285 else if (removedPCB == suspendedReady->head)
286 {
287     PCB *removedNext = removedPCB->nextPCB;
288
289     suspendedReady->head = removedNext;
290     removedNext->prevPCB = NULL;
291     removedPCB->nextPCB = NULL;
292     suspendedReady->count--;
293     return 0;
294 }
295 else if (removedPCB == suspendedBlocked->head)
296 {
297     PCB *removedNext = removedPCB->nextPCB;
298
299     suspendedBlocked->head = removedNext;
300     removedNext->prevPCB = NULL;
301     removedPCB->nextPCB = NULL;
302     suspendedBlocked->count--;
303     return 0;
304 }
305 else if (removedPCB == ready->tail)
306 {
307     PCB *removedPrev = removedPCB->prevPCB;
308
309     ready->tail = removedPrev;
310     removedPrev->nextPCB = NULL;
311     removedPCB->prevPCB = NULL;
312     ready->count--;
313     return 0;
314 }
315 else if (removedPCB == blocked->tail)
316 {
317     PCB *removedPrev = removedPCB->prevPCB;
318
319     blocked->tail = removedPrev;
320     removedPrev->nextPCB = NULL;
321     removedPCB->prevPCB = NULL;
322     blocked->count--;
323     return 0;
324 }
325 else if (removedPCB == suspendedReady->tail)
326 {
327     PCB *removedPrev = removedPCB->prevPCB;
328
329     suspendedReady->tail = removedPrev;
330     removedPrev->nextPCB = NULL;
331     removedPCB->prevPCB = NULL;
332     suspendedReady->count--;
333     return 0;
334 }
335 else if (removedPCB == suspendedBlocked->tail)
336 {
337     PCB *removedPrev = removedPCB->prevPCB;
338
339     suspendedBlocked->tail = removedPrev;
340     removedPrev->nextPCB = NULL;
341     removedPCB->prevPCB = NULL;
342     suspendedBlocked->count--;
343     return 0;

```



```

344     }
345     else
346     {
347         PCB *tempPrev = removedPCB->prevPCB;
348         PCB *tempNext = removedPCB->nextPCB;
349
350         tempPrev->nextPCB = tempNext;
351         tempNext->prevPCB = tempPrev;
352
353         removedPCB->nextPCB = NULL;
354         removedPCB->prevPCB = NULL;
355
356         if (removedPCB->runningStatus == 0 && removedPCB->suspendedStatus == 1)
357         {
358             ready->count--;
359         }
360         else if (removedPCB->runningStatus == -1 && removedPCB->suspendedStatus == 1)
361         {
362             blocked->count--;
363         }
364         else if (removedPCB->runningStatus == 0 && removedPCB->suspendedStatus == 0)
365         {
366             suspendedReady->count--;
367         }
368         else if (removedPCB->runningStatus == -1 && removedPCB->suspendedStatus == 0)
369         {
370             suspendedBlocked->count--;
371         }
372
373         return 0;
374     }
375 }

```

#### 5.24.1.11 setupPCB()

```

PCB* setupPCB (
    char *processName,
    unsigned char processClass,
    int processPriority )

```

Definition at line 48 of file R2\_Internal\_Functions\_And\_Structures.c.

```

49 {
50     //COLTON WILL PROGRAM THIS FUNCTION
51
52     //setupPcb() will call allocatePCB() to create an empty PCB, initializes the PCB information, sets
    the PCB state to ready, not suspended.
53
54     PCB *returnedPCB = allocatePCB();
55
56     if (findPCB(processName)->processName == processName)
57     {
58         char message[] = "There is already a PCB with this name.\n";
59         int messLength = strlen(message);
60         sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
61
62         returnedPCB = NULL;
63     }
64     else
65     {
66
67         strcpy(returnedPCB->processName, processName);
68         returnedPCB->processClass = processClass;
69         returnedPCB->priority = processPriority;
70         returnedPCB->runningStatus = 0;
71         returnedPCB->suspendedStatus = 1;
72     }
73
74     return returnedPCB;
75 }

```

## 5.24.2 Variable Documentation

#### 5.24.2.1 blocked

`queue*` blocked

Definition at line 8 of file R2\_Internal\_Functions\_And\_Structures.c.

#### 5.24.2.2 ready

`queue*` ready

Definition at line 7 of file R2\_Internal\_Functions\_And\_Structures.c.

#### 5.24.2.3 suspendedBlocked

`queue*` suspendedBlocked

Definition at line 10 of file R2\_Internal\_Functions\_And\_Structures.c.

#### 5.24.2.4 suspendedReady

`queue*` suspendedReady

Definition at line 9 of file R2\_Internal\_Functions\_And\_Structures.c.

## 5.25 modules/R2/R2\_Internal\_Functions\_And\_Structures.h File Reference

### Classes

- struct `PCB`
- struct `queue`

### Typedefs

- typedef struct `PCB PCB`
- typedef struct `queue queue`

## Functions

- `PCB *allocatePCB ()`
- `int freePCB (PCB *PCB_to_free)`
- `PCB *setupPCB (char *processName, unsigned char processClass, int processPriority)`
- `PCB *findPCB (char *processName)`
- `void insertPCB (PCB *PCB_to_insert)`
- `int removePCB (PCB *PCB_to_remove)`
- `void allocateQueues ()`
- `queue *getReady ()`
- `queue *getBlocked ()`
- `queue *getSuspendedReady ()`
- `queue *getSuspendedBlocked ()`

### 5.25.1 Typedef Documentation

#### 5.25.1.1 PCB

```
typedef struct PCB PCB
```

#### 5.25.1.2 queue

```
typedef struct queue queue
```

### 5.25.2 Function Documentation

#### 5.25.2.1 allocatePCB()

```
PCB* allocatePCB ( )
```

Definition at line 14 of file R2\_Internal\_Functions\_And\_Structures.c.

```

15 {
16     //COLTON WILL PROGRAM THIS FUNCTION
17
18     //allocatePCB() will use sys_alloc_mem() to allocate memory for a new PCB, possible including the
19     //stack, and perform any reasonable initialization.
20     PCB *newPCB = (PCB *)sys_alloc_mem(sizeof(PCB));
21
22     char name[20] = "newPCB";
23     strcpy(newPCB->processName, name);
24
25     newPCB->suspendedStatus = 1;
26     newPCB->runningStatus = -1;
27     newPCB->stackTop = (newPCB->stackTop + 1024);
28     newPCB->stackBase = newPCB->stackBase;
29     newPCB->priority = 0;
30
31     // Setting the PCBs prev and next PCB
32     newPCB->nextPCB = NULL;
33     newPCB->prevPCB = NULL;
34
35     newPCB->processClass = NULL;
36
37     return newPCB;
38 }
```

### 5.25.2.2 allocateQueues()

```
void allocateQueues ( )
```

Definition at line 377 of file R2\_Internal\_Functions\_And\_Structures.c.

```

378 {
379     ready = sys_alloc_mem(sizeof(queue));
380     ready->count = 0;
381     ready->head = NULL;
382     ready->tail = NULL;
383     blocked = sys_alloc_mem(sizeof(queue));
384     blocked->count = 0;
385     blocked->head = NULL;
386     blocked->tail = NULL;
387     suspendedReady = sys_alloc_mem(sizeof(queue));
388     suspendedReady->count = 0;
389     suspendedReady->head = NULL;
390     suspendedReady->tail = NULL;
391     suspendedBlocked = sys_alloc_mem(sizeof(queue));
392     suspendedBlocked->count = 0;
393     suspendedBlocked->head = NULL;
394     suspendedBlocked->tail = NULL;
395 }
```

### 5.25.2.3 findPCB()

```
PCB* findPCB (
    char *processName )
```

Definition at line 77 of file R2\_Internal\_Functions\_And\_Structures.c.

```

78 {
79     // ANASTASE WILL PROGRAM THIS FUNCTION
80
81     //findPCB() will search all queues for a process with a given name.
82
83     if (strlen(processName) > 20)
84     {
85
86         char error_message[30] = "Invalid process name.\n";
87         int error_size = strlen(error_message);
88         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
89         return NULL;
90         //return cz we have to stop if the process name is too long
91     }
92     else
93     {
94         PCB *tempPCB = ready->head;
95         int value = 0;
96         while (value <= ready->count)
97         {
98             if (strcmp(tempPCB->processName, processName) == 0)
99             {
100                 return tempPCB;
101             }
102             else
103             {
104                 tempPCB = tempPCB->nextPCB;
105                 value++;
106             }
107         }
108
109         tempPCB = blocked->head;
110         value = 0;
111         while (value <= blocked->count)
112         {
113             if (strcmp(tempPCB->processName, processName) == 0)
114             {
115                 return tempPCB;
116             }
117             else
118             {
119                 tempPCB = tempPCB->nextPCB;
120                 value++;
121             }
122         }

```

```

123
124     tempPCB = suspendedBlocked->head;
125     value = 0;
126     while (value <= suspendedBlocked->count)
127     {
128         if (strcmp(tempPCB->processName, processName) == 0)
129         {
130             return tempPCB;
131         }
132         else
133         {
134             tempPCB = tempPCB->nextPCB;
135             value++;
136         }
137     }
138
139     tempPCB = suspendedReady->head;
140     value = 0;
141     while (value <= suspendedReady->count)
142     {
143         if (strcmp(tempPCB->processName, processName) == 0)
144         {
145             return tempPCB;
146         }
147         else
148         {
149             tempPCB = tempPCB->nextPCB;
150             value++;
151         }
152     }
153
154     return NULL;
155 }
156 }

```

#### 5.25.2.4 freePCB()

```

int freePCB (
    PCB * PCB_to_free )

```

Definition at line 39 of file R2\_Internal\_Functions\_And\_Structures.c.

```

40 {
41     // ANASTASE WILL PROGRAM THIS FUNCTION
42
43     //freePCB() will use sys_free_mem() to free all memory associated with a given PCB (the stack, the
44     PCB itself, etc.)
45     return sys_free_mem(PCB_to_free);
46 }

```

#### 5.25.2.5 getBlocked()

```

queue* getBlocked ( )

```

Definition at line 402 of file R2\_Internal\_Functions\_And\_Structures.c.

```

403 {
404     return blocked;
405 }

```

### 5.25.2.6 getReady()

```
queue* getReady ( )
```

Definition at line 397 of file R2\_Internal\_Functions\_And\_Structures.c.

```
398 {
399     return ready;
400 }
```

### 5.25.2.7 getSuspendedBlocked()

```
queue* getSuspendedBlocked ( )
```

Definition at line 412 of file R2\_Internal\_Functions\_And\_Structures.c.

```
413 {
414     return suspendedBlocked;
415 }
```

### 5.25.2.8 getSuspendedReady()

```
queue* getSuspendedReady ( )
```

Definition at line 407 of file R2\_Internal\_Functions\_And\_Structures.c.

```
408 {
409     return suspendedReady;
410 }
```

### 5.25.2.9 insertPCB()

```
void insertPCB (
    PCB *PCB_to_insert )
```

Definition at line 158 of file R2\_Internal\_Functions\_And\_Structures.c.

```
159 {
160     //BENJAMIN WILL PROGRAM THIS FUNCTION
161
162     //insertPCB() will insert a PCB into the appropriate queue.
163     //Note: The ready queue is a priority queue and the blocked queue is a FIFO queue.
164
165     if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 1)
166     { // Insert into ready queue
167         PCB *tempPtr = ready->head;
168
169         if (tempPtr != NULL)
170         {
171             int temp = 0;
172             while (temp <= ready->count)
173             {
174                 if (PCB_to_insert->priority < tempPtr->priority)
175                 {
176                     tempPtr = tempPtr->nextPCB;
177                 }
178                 else if (PCB_to_insert->priority >= tempPtr->priority)
179                 {
180                     PCB_to_insert->nextPCB = tempPtr;
181                     PCB_to_insert->prevPCB = tempPtr->prevPCB;
182                     tempPtr->prevPCB = PCB_to_insert;
183                 }
184             }
185         }
186     }
187 }
```

```

184         else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
185         {
186             tempPtr->nextPCB = PCB_to_insert;
187             PCB_to_insert->prevPCB = tempPtr;
188             ready->tail = PCB_to_insert;
189         }
190         temp++;
191     }
192     ready->count++;
193 }
194 else
195 {
196     ready->count++;
197     ready->head = PCB_to_insert;
198     ready->tail = PCB_to_insert;
199 }
200 }
201 else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
202 { // Insert into suspended ready queue
203     PCB *tempPtr = suspendedReady->head;
204
205     if (tempPtr != NULL)
206     {
207         int temp = 0;
208         while (temp <= suspendedReady->count)
209         {
210             if (PCB_to_insert->priority < tempPtr->priority)
211             {
212                 tempPtr = tempPtr->nextPCB;
213             }
214             else if (PCB_to_insert->priority >= tempPtr->priority)
215             {
216                 PCB_to_insert->nextPCB = tempPtr;
217                 PCB_to_insert->prevPCB = tempPtr->prevPCB;
218                 tempPtr->prevPCB = PCB_to_insert;
219             }
220             else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
221             {
222                 tempPtr->nextPCB = PCB_to_insert;
223                 PCB_to_insert->prevPCB = tempPtr;
224                 suspendedReady->tail = PCB_to_insert;
225             }
226             temp++;
227         }
228         suspendedReady->count++;
229     }
230     else
231     {
232         suspendedReady->count++;
233         suspendedReady->head = PCB_to_insert;
234         suspendedReady->tail = PCB_to_insert;
235     }
236 }
237 else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 1)
238 { // Insert into blocked queue
239     PCB *tempPtr = blocked->tail;
240
241     tempPtr->nextPCB = PCB_to_insert;
242     PCB_to_insert->prevPCB = tempPtr;
243     blocked->tail = PCB_to_insert;
244 }
245 else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
246 { // Insert into suspended blocked queue
247     PCB *tempPtr = suspendedBlocked->tail;
248
249     tempPtr->nextPCB = PCB_to_insert;
250     PCB_to_insert->prevPCB = tempPtr;
251     suspendedBlocked->tail = PCB_to_insert;
252 }
253 }

```

### 5.25.2.10 removePCB()

```

int removePCB (
    PCB *PCB_to_remove )

```

Definition at line 255 of file R2\_Internal\_Functions\_And\_Structures.c.

```

256 {

```

```

257 //BENJAMIN WILL PROGRAM THIS FUNCTION
258
259 //removePCB() will remove a PCB from the queue in which it is currently stored.
260
261 PCB *removedPCB = findPCB(PCB_to_remove->processName);
262 if (removedPCB == NULL)
263 {
264     return 1;
265 }
266 else if (removedPCB == ready->head)
267 {
268     PCB *removedNext = removedPCB->nextPCB;
269
270     ready->head = removedNext;
271     removedNext->prevPCB = NULL;
272     removedPCB->nextPCB = NULL;
273     ready->count--;
274     return 0;
275 }
276 else if (removedPCB == blocked->head)
277 {
278     PCB *removedNext = removedPCB->nextPCB;
279     blocked->head = removedNext;
280     removedNext->prevPCB = NULL;
281     removedPCB->nextPCB = NULL;
282     blocked->count--;
283     return 0;
284 }
285 else if (removedPCB == suspendedReady->head)
286 {
287     PCB *removedNext = removedPCB->nextPCB;
288
289     suspendedReady->head = removedNext;
290     removedNext->prevPCB = NULL;
291     removedPCB->nextPCB = NULL;
292     suspendedReady->count--;
293     return 0;
294 }
295 else if (removedPCB == suspendedBlocked->head)
296 {
297     PCB *removedNext = removedPCB->nextPCB;
298
299     suspendedBlocked->head = removedNext;
300     removedNext->prevPCB = NULL;
301     removedPCB->nextPCB = NULL;
302     suspendedBlocked->count--;
303     return 0;
304 }
305 else if (removedPCB == ready->tail)
306 {
307     PCB *removedPrev = removedPCB->prevPCB;
308
309     ready->tail = removedPrev;
310     removedPrev->nextPCB = NULL;
311     removedPCB->prevPCB = NULL;
312     ready->count--;
313     return 0;
314 }
315 else if (removedPCB == blocked->tail)
316 {
317     PCB *removedPrev = removedPCB->prevPCB;
318
319     blocked->tail = removedPrev;
320     removedPrev->nextPCB = NULL;
321     removedPCB->prevPCB = NULL;
322     blocked->count--;
323     return 0;
324 }
325 else if (removedPCB == suspendedReady->tail)
326 {
327     PCB *removedPrev = removedPCB->prevPCB;
328
329     suspendedReady->tail = removedPrev;
330     removedPrev->nextPCB = NULL;
331     removedPCB->prevPCB = NULL;
332     suspendedReady->count--;
333     return 0;
334 }
335 else if (removedPCB == suspendedBlocked->tail)
336 {
337     PCB *removedPrev = removedPCB->prevPCB;
338
339     suspendedBlocked->tail = removedPrev;
340     removedPrev->nextPCB = NULL;
341     removedPCB->prevPCB = NULL;
342     suspendedBlocked->count--;
343     return 0;

```



```

344     }
345     else
346     {
347         PCB *tempPrev = removedPCB->prevPCB;
348         PCB *tempNext = removedPCB->nextPCB;
349
350         tempPrev->nextPCB = tempNext;
351         tempNext->prevPCB = tempPrev;
352
353         removedPCB->nextPCB = NULL;
354         removedPCB->prevPCB = NULL;
355
356         if (removedPCB->runningStatus == 0 && removedPCB->suspendedStatus == 1)
357         {
358             ready->count--;
359         }
360         else if (removedPCB->runningStatus == -1 && removedPCB->suspendedStatus == 1)
361         {
362             blocked->count--;
363         }
364         else if (removedPCB->runningStatus == 0 && removedPCB->suspendedStatus == 0)
365         {
366             suspendedReady->count--;
367         }
368         else if (removedPCB->runningStatus == -1 && removedPCB->suspendedStatus == 0)
369         {
370             suspendedBlocked->count--;
371         }
372
373         return 0;
374     }
375 }

```

### 5.25.2.11 setupPCB()

```

PCB* setupPCB (
    char *processName,
    unsigned char processClass,
    int processPriority )

```

Definition at line 48 of file R2\_Internal\_Functions\_And\_Structures.c.

```

49 {
50     //COLTON WILL PROGRAM THIS FUNCTION
51
52     //setupPcb() will call allocatePCB() to create an empty PCB, initializes the PCB information, sets
53     //the PCB state to ready, not suspended.
54
55     PCB *returnedPCB = allocatePCB();
56
57     if (findPCB(processName)->processName == processName)
58     {
59         char message[] = "There is already a PCB with this name.\n";
60         int messLength = strlen(message);
61         sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
62
63         returnedPCB = NULL;
64     }
65     else
66     {
67         strcpy(returnedPCB->processName, processName);
68         returnedPCB->processClass = processClass;
69         returnedPCB->priority = processPriority;
70         returnedPCB->runningStatus = 0;
71         returnedPCB->suspendedStatus = 1;
72     }
73
74     return returnedPCB;
75 }

```

## 5.26 modules/R2/R2commands.c File Reference

```

#include <string.h>
#include "../mpx_supt.h"

```

```
#include "R2_Internal_Functions_And_Structures.h"
#include "R2commands.h"
#include <core/serial.h>
```

## Functions

- void [createPCB](#) (char \*processName, char processClass, int processPriority)
- void [deletePCB](#) (char \*processName)
- void [blockPCB](#) (char \*processName)
- void [unblockPCB](#) (char \*processName)
- void [suspendPCB](#) (char \*processName)
- void [resumePCB](#) (char \*processName)
- void [setPCBPRIORITY](#) (char \*processName, int newProcessPriority)
- void [showPCB](#) (char \*processName)
- void [showReady](#) ()
- void [showSuspendedReady](#) ()
- void [showSuspendedBlocked](#) ()
- void [showBlocked](#) ()
- void [showAll](#) ()

### 5.26.1 Function Documentation

#### 5.26.1.1 blockPCB()

```
void blockPCB (
    char *processName )
```

Definition at line 113 of file R2commands.c.

```
114 { // ANASTASE WILL PROGRAM THIS FUNCTION
115
116     // find pcb and validate process name
117     PCB *pcb_to_block = findPCB(processName);
118
119     if (pcb_to_block != NULL)
120     {
121         pcb_to_block->runningStatus = -1; // blocked
122         removePCB(pcb_to_block);
123         insertPCB(pcb_to_block);
124
125         char msg[] = "The PCB was successfully blocked!\n";
126         int msgLen = strlen(msg);
127         sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
128     }
129 }
```

## 5.26.1.2 createPCB()

```
void createPCB (
    char *processName,
    char processClass,
    int processPriority )
```

Definition at line 11 of file R2commands.c.

```
12 { // BENJAMIN WILL PROGRAM THIS FUNCTION
13     /*
14     The createPCB command will call setupPCB() and insert the PCB in the appropriate queue
15     */
16     /*
17     Error Checking:
18     Name must be unique and valid.
19     Class must be valid.
20     Priority must be valid.
21     */
22
23     if (findPCB(processName) != NULL || strlen(processName) > 20)
24     { // Check if the process has a unique name, and if it has a valid name.
25         char errMsg[125];
26         strcpy(errMsg, "The PCB could not be created as it either does not have a unique name or the name
is longer than 20 characters!\n");
27         int errLen = strlen(errMsg);
28         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
29     }
30     else if (processClass != 'a' && processClass != 's')
31     { // Check if the process has a valid class.
32         char errMsg[100];
33         strcpy(errMsg, "The PCB could not be created as it does not have a valid class!\n");
34         int errLen = strlen(errMsg);
35         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
36     }
37     else if (processPriority < 0 || processPriority > 9)
38     { // Check if the process has a valid priority.
39         char errMsg[100];
40         strcpy(errMsg, "The PCB could not be created as it does not have a valid priority!\n");
41         int errLen = strlen(errMsg);
42         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
43     }
44     else
45     { // Make the PCB
46         PCB *createdPCB = setupPCB(processName, processClass, processPriority);
47
48         char msg[] = "The PCB was created!\n";
49         int msgLen = strlen(msg);
50         sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
51
52         insertPCB(createdPCB);
53     }
54 }
```

## 5.26.1.3 deletePCB()

```
void deletePCB (
    char *processName )
```

Definition at line 56 of file R2commands.c.

```
57 { // BENJAMIN WILL PROGRAM THIS FUNCTION
58     /*
59     The deletePCB command will remove a PCB from the appropriate queue and then free all associated
memory.
60     This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
61     */
62     /*
63     Error Checking:
64     Name must be valid.
65     */
66
67     if (strlen(processName) > 20)
68     { // Check if the process has a valid name.
69         char errMsg[100];
70         strcpy(errMsg, "The PCB could not be deleted as the name is longer than 20 characters!\n");
```

```

71     int errLen = strlen(errMsg);
72     sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
73 }
74
75 PCB *PCB_to_delete = findPCB(processName);
76
77 if (PCB_to_delete == NULL)
78 {
79     char errMsg[42] = "The PCB you want to remove does not exist\n";
80     int errMsgLen = 42;
81     sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
82 }
83 else
84 {
85     int removed = removePCB(PCB_to_delete);
86     if (removed == 1)
87     {
88         char errMsg[] = "The PCB could not be unlinked.\n";
89         int errMsgLen = strlen(errMsg);
90         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
91     }
92     else
93     {
94         int result = sys_free_mem(PCB_to_delete);
95         if (result == -1)
96         {
97             char errMsg[50];
98             strcpy(errMsg, "The PCB could not be successfully deleted\n");
99             int errLen = strlen(errMsg);
100             sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
101         }
102         else
103         {
104             char msg[50];
105             strcpy(msg, "The desired PCB was deleted\n");
106             int msgLen = strlen(msg);
107             sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
108         }
109     }
110 }
111 }

```

#### 5.26.1.4 resumePCB()

```

void resumePCB (
    char *processName )

```

Definition at line 187 of file R2commands.c.

```

188 { // COLTON WILL PROGRAM THIS FUNCTION
189     /*
190      Places a PCB in the not suspended state and reinserts it into the appropriate queue
191      */
192
193     PCB *PCBtoResume = findPCB(processName);
194
195     if (PCBtoResume == NULL || strlen(processName) > 20)
196     {
197         char nameError[] = "This is not a valid name.\n";
198         int printCount = strlen(nameError);
199         sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
200     }
201     else
202     {
203         removePCB(PCBtoResume);
204         PCBtoResume->suspendedStatus = 1;
205         insertPCB(PCBtoResume);
206
207         char msg[] = "The PCB was successfully resumed!\n";
208         int msgLen = strlen(msg);
209         sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
210     }
211 }
212 }

```

### 5.26.1.5 setPCBPriorty()

```
void setPCBPriorty (
    char *processName,
    int newProcessPriority )
```

Definition at line 217 of file R2commands.c.

```
218 { // ANASTASE WILL PROGRAM THIS FUNCTION
219
220     // Sets a PCB's priority and reinserts the process into the correct place in the correct queue
221
222     /*
223     Error Checking:
224     Name must be valid.
225     newPriority
226     */
227
228     // find the process and validate the name
229     PCB *tempPCB = findPCB(processName);
230
231     if ((tempPCB != NULL) && (newProcessPriority >= 0) && (newProcessPriority < 10))
232     {
233         tempPCB->priority = newProcessPriority;
234         removePCB(tempPCB);
235         insertPCB(tempPCB);
236
237         char msg[] = "The PCB's priority was successfully changed!\n";
238         int msgLen = strlen(msg);
239         sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
240     }
241 }
```

### 5.26.1.6 showAll()

```
void showAll ( )
```

Definition at line 612 of file R2commands.c.

```
613 { // COLTON WILL PROGRAM THIS FUNCTION
614     /*
615     Displays the following information for each PCB in the ready and blocked queues:
616     Process Name
617     Class
618     State
619     Suspended Status
620     Priority
621     */
622     /*
623     Error Checking:
624     None
625     */
626
627     showReady();
628     showSuspendedReady();
629     showBlocked();
630     showSuspendedBlocked();
631 }
```

### 5.26.1.7 showBlocked()

```
void showBlocked ( )
```

Definition at line 559 of file R2commands.c.

```
560 { // ANASTASE WILL PROGRAM THIS FUNCTION
561     /*
562     Displays the following information for each PCB in the blocked queue:
563     Process Name
```

```

564         Class
565         State
566         Suspended Status
567         Priority
568         HEAD
569     */
570     /*
571     Error Checking:
572     None
573     */
574
575     // check
576
577     char print_message[30] = "The blocked queue:\n";
578     int message_size = strlen(print_message);
579     sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
580
581     // printPCBs(blocked);
582     queue *tempQueue = getBlocked();
583     PCB *tempPtr = tempQueue->head; //PCB_container->head;
584     int count = tempQueue->count;
585
586     if (count == 0)
587     {
588         // the queue is empty
589         char error_message[30] = "The queue is empty.\n";
590         int error_size = strlen(error_message);
591         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
592         return;
593     }
594     // The queue is not empty
595
596     int value = 0;
597     // Testing purpose
598     //char print_message[38]="The blocke queue testing:\n";
599     //int message_size=strlen(print_message);
600     //sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
601
602     while (value < count)
603     { // testing for <= or <
604         // Print out the process
605         showPCB(tempPtr->processName);
606         // increment pcb*tempPtr, the loop variable.
607         tempPtr = tempPtr->nextPCB;
608         value++;
609     }
610 }

```

### 5.26.1.8 showPCB()

```

void showPCB (
    char *processName )

```

Definition at line 243 of file R2commands.c.

```

244 { // BENJAMIN WILL PROGRAM THIS FUNCTION
245     /*
246     Displays the following information for a PCB:
247         Process Name
248         Class
249         State
250         Suspended Status
251         Priority
252     */
253
254     /*
255     Error Checking:
256     Name must be valid.
257     */
258
259     if (strlen(processName) > 20)
260     { // Check if the process has a valid name.
261         char errMsg[100];
262         strcpy(errMsg, "The PCB could not be shown as the name is longer than 20 characters!\n");
263         int errLen = strlen(errMsg);
264         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
265     }
266     else
267     {

```

```

268
269     PCB *PCB_to_show = findPCB(processName);
270
271     if (PCB_to_show == NULL)
272     { // Check to see if the PCB exists.
273         char errMsg[100];
274         strcpy(errMsg, "The PCB could not be shown, as it does not exist!\n");
275         int errLen = strlen(errMsg);
276         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
277     }
278     else
279     {
280         // Print out the PCB name.
281         char nameMsg[50];
282         strcpy(nameMsg, "The process name is: ");
283         int nameMsgLen = strlen(nameMsg);
284         sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
285         char name[20];
286         strcpy(name, PCB_to_show->processName);
287         int nameLen = strlen(name);
288         sys_req(WRITE, DEFAULT_DEVICE, name, &nameLen);
289         char newLine[1];
290         strcpy(newLine, "\n");
291         int newLineLen = 1;
292         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);
293
294         // Print out PCB class
295         char classMsg[50];
296         strcpy(classMsg, "The process class is: ");
297         int classMsgLen = strlen(classMsg);
298         sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
299
300         if (PCB_to_show->processClass == 'a')
301         {
302             char appMsg[50];
303             strcpy(appMsg, "application");
304             int appMsgLen = strlen(appMsg);
305             sys_req(WRITE, DEFAULT_DEVICE, appMsg, &appMsgLen);
306         }
307         else
308         {
309             char sysMsg[50];
310             strcpy(sysMsg, "system");
311             int sysMsgLen = strlen(sysMsg);
312             sys_req(WRITE, DEFAULT_DEVICE, sysMsg, &sysMsgLen);
313         }
314         sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);
315
316         // Print out the PCB state
317
318         if (PCB_to_show->runningStatus == 0)
319         { // The process is ready.
320             char stateMsg[50];
321             strcpy(stateMsg, "The process is ready!\n");
322             int stateMsgLen = strlen(stateMsg);
323             sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
324         }
325         else if (PCB_to_show->runningStatus == -1)
326         { // The process is blocked.
327             char stateMsg[50];
328             strcpy(stateMsg, "The process is blocked!\n");
329             int stateMsgLen = strlen(stateMsg);
330             sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
331         }
332         else if (PCB_to_show->runningStatus == 1)
333         { // The process is running.
334             char stateMsg[50];
335             strcpy(stateMsg, "The process is running!\n");
336             int stateMsgLen = strlen(stateMsg);
337             sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
338         }
339
340         // Print out the PCB suspended status
341
342         if (PCB_to_show->suspendedStatus == 0)
343         { // The process is suspended
344             char susMsg[50];
345             strcpy(susMsg, "The process is suspended!\n");
346             int susMsgLen = strlen(susMsg);
347             sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
348         }
349         else if (PCB_to_show->suspendedStatus == 1)
350         { // The process is not suspended
351             char susMsg[50];
352             strcpy(susMsg, "The process is not suspended!\n");
353             int susMsgLen = strlen(susMsg);
354             sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);

```

```

355     }
356
357     // Print out the PCB priority
358     char priorityMsg[50];
359     int priorityMsgLen = 0;
360
361     switch (PCB_to_show->priority)
362     {
363     case 0:
364         strcpy(priorityMsg, "The process priority is 0!\n");
365         priorityMsgLen = strlen(priorityMsg);
366         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
367         break;
368
369     case 1:
370         strcpy(priorityMsg, "The process priority is 1!\n");
371         priorityMsgLen = strlen(priorityMsg);
372         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
373         break;
374
375     case 2:
376         strcpy(priorityMsg, "The process priority is 2!\n");
377         priorityMsgLen = strlen(priorityMsg);
378         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
379         break;
380
381     case 3:
382         strcpy(priorityMsg, "The process priority is 3!\n");
383         priorityMsgLen = strlen(priorityMsg);
384         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
385         break;
386
387     case 4:
388         strcpy(priorityMsg, "The process priority is 4!\n");
389         priorityMsgLen = strlen(priorityMsg);
390         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
391         break;
392
393     case 5:
394         strcpy(priorityMsg, "The process priority is 5!\n");
395         priorityMsgLen = strlen(priorityMsg);
396         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
397         break;
398
399     case 6:
400         strcpy(priorityMsg, "The process priority is 6!\n");
401         priorityMsgLen = strlen(priorityMsg);
402         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
403         break;
404
405     case 7:
406         strcpy(priorityMsg, "The process priority is 7!\n");
407         priorityMsgLen = strlen(priorityMsg);
408         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
409         break;
410
411     case 8:
412         strcpy(priorityMsg, "The process priority is 8!\n");
413         priorityMsgLen = strlen(priorityMsg);
414         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
415         break;
416
417     case 9:
418         strcpy(priorityMsg, "The process priority is 9!\n");
419         priorityMsgLen = strlen(priorityMsg);
420         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
421         break;
422
423     default:
424         break;
425     }
426 }
427 }
428 }

```

### 5.26.1.9 showReady()

```
void showReady ( )
```

Definition at line 430 of file R2commands.c.



```

431 { // COLTON WILL PROGRAM THIS FUNCTION
432     /*
433      * Displays the following information for each PCB in the ready queue:
434      *   Process Name
435      *   Class
436      *   State
437      *   Suspended Status
438      *   Priority
439      */
440     /*
441     * Error Checking:
442     *   None
443     */
444
445     char message[] = "Printing the ready queue:\n";
446     int messLength = strlen(message);
447     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
448
449     queue *tempQueue = getReady();
450     PCB *tempPCB = tempQueue->head;
451
452     int loop = 0;
453     int count = tempQueue->count;
454
455     if (count == 0)
456     {
457         // the queue is empty
458         char error_message[30] = "The queue is empty.\n";
459         int error_size = strlen(error_message);
460         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
461         return;
462     }
463
464     while (loop < count)
465     {
466         showPCB(tempPCB->processName);
467         PCB *tempNext = tempPCB->nextPCB;
468         loop++;
469         tempPCB = tempNext;
470     }
471 }

```

### 5.26.1.10 showSuspendedBlocked()

void showSuspendedBlocked ( )

Definition at line 516 of file R2commands.c.

```

517 { // COLTON WILL PROGRAM THIS FUNCTION
518     /*
519      * Displays the following information for each PCB in the suspended blocked queue:
520      *   Process Name
521      *   Class
522      *   State
523      *   Suspended Status
524      *   Priority
525      */
526     /*
527     * Error Checking:
528     *   None
529     */
530
531     char message[] = "Printing the suspended blocked queue:\n";
532     int messLength = strlen(message);
533     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
534
535     queue *tempQueue = getSuspendedBlocked();
536     PCB *tempPCB = tempQueue->head;
537
538     int loop = 0;
539     int count = tempQueue->count;
540
541     if (count == 0)
542     {
543         // the queue is empty
544         char error_message[30] = "The queue is empty.\n";
545         int error_size = strlen(error_message);
546         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
547         return;
548     }

```

```

549
550     while (loop < count)
551     {
552         showPCB(tempPCB->processName);
553         PCB *tempNext = tempPCB->nextPCB;
554         loop++;
555         tempPCB = tempNext;
556     }
557 }

```

### 5.26.1.11 showSuspendedReady()

```
void showSuspendedReady ( )
```

Definition at line 473 of file R2commands.c.

```

474 { // COLTON WILL PROGRAM THIS FUNCTION
475     /*
476     Displays the following information for each PCB in the suspended ready queue:
477         Process Name
478         Class
479         State
480         Suspended Status
481         Priority
482     */
483     /*
484     Error Checking:
485     None
486     */
487
488     char message[] = "Printing the suspended ready queue:\n";
489     int messLength = strlen(message);
490     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
491
492     queue *tempQueue = getSuspendedReady();
493     PCB *tempPCB = tempQueue->head;
494
495     int loop = 0;
496     int count = tempQueue->count;
497
498     if (count == 0)
499     {
500         // the queue is empty
501         char error_message[30] = "The queue is empty.\n";
502         int error_size = strlen(error_message);
503         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
504         return;
505     }
506
507     while (loop < count)
508     {
509         showPCB(tempPCB->processName);
510         PCB *tempNext = tempPCB->nextPCB;
511         loop++;
512         tempPCB = tempNext;
513     }
514 }

```

### 5.26.1.12 suspendPCB()

```
void suspendPCB (
    char *processName )
```

Definition at line 157 of file R2commands.c.

```

158 { // COLTON WILL PROGRAM THIS FUNCTION
159     /*
160     Places a PCB in the suspended state and reinserts it into the appropriate queue
161     */
162
163     PCB *PCBtoSuspend = findPCB(processName);
164 }

```

```

169     if (PCBtoSuspend == NULL || strlen(processName) > 20)
170     {
171         char nameError[] = "This is not a valid name.\n";
172         int printCount = strlen(nameError);
173         sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
174     }
175     else
176     {
177         removePCB(PCBtoSuspend);
178         PCBtoSuspend->suspendedStatus = 0;
179         insertPCB(PCBtoSuspend);
180
181         char msg[] = "The PCB was successfully suspended!\n";
182         int msgLen = strlen(msg);
183         sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
184     }
185 }

```

### 5.26.1.13 unblockPCB()

```

void unblockPCB (
    char *processName )

```

Definition at line 131 of file R2commands.c.

```

132 { // ANASTASE WILL PROGRAM THIS FUNCTION
133
134     /*
135     Places a PCB in the unblocked state and reinserts it into the appropriate queue.
136     */
137     /*
138     Error Checking:
139     Name must be valid.
140
141     */
142
143     PCB *pcb_to_unblock = findPCB(processName);
144     if (pcb_to_unblock != NULL)
145     {
146         pcb_to_unblock->runningStatus = 0; // ready
147         removePCB(pcb_to_unblock);          // is this the right place to put that function?
148         insertPCB(pcb_to_unblock);
149
150         char msg[] = "The PCB was successfully unblocked!\n";
151         int msgLen = strlen(msg);
152         sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
153     }
154 }

```

## 5.27 modules/R2/R2commands.h File Reference

### Functions

- void [createPCB](#) (char \*processName, char processClass, int processPriority)
- void [deletePCB](#) (char \*processName)
- void [blockPCB](#) (char \*processName)
- void [unblockPCB](#) (char \*processName)
- void [suspendPCB](#) (char \*processName)
- void [resumePCB](#) (char \*processName)
- void [setPCBPRIORITY](#) (char \*processName, int newProcessPriority)
- void [showPCB](#) (char \*processName)
- void [showReady](#) ()
- void [showSuspendedBlocked](#) ()
- void [showSuspendedReady](#) ()
- void [showBlocked](#) ()
- void [showAll](#) ()

## 5.27.1 Function Documentation

### 5.27.1.1 blockPCB()

```
void blockPCB (
    char *processName )
```

Definition at line 113 of file R2commands.c.

```
114 { // ANASTASE WILL PROGRAM THIS FUNCTION
115
116     // find pcb and validate process name
117     PCB *pcb_to_block = findPCB(processName);
118
119     if (pcb_to_block != NULL)
120     {
121         pcb_to_block->runningStatus = -1; // blocked
122         removePCB(pcb_to_block);
123         insertPCB(pcb_to_block);
124
125         char msg[] = "The PCB was successfully blocked!\n";
126         int msgLen = strlen(msg);
127         sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
128     }
129 }
```

### 5.27.1.2 createPCB()

```
void createPCB (
    char *processName,
    char processClass,
    int processPriority )
```

Definition at line 11 of file R2commands.c.

```
12 { // BENJAMIN WILL PROGRAM THIS FUNCTION
13     /*
14     The createPCB command will call setupPCB() and insert the PCB in the appropriate queue
15     */
16     /*
17     Error Checking:
18     Name must be unique and valid.
19     Class must be valid.
20     Priority must be valid.
21     */
22
23     if (findPCB(processName) != NULL || strlen(processName) > 20)
24     { // Check if the process has a unique name, and if it has a valid name.
25         char errMsg[125];
26         strcpy(errMsg, "The PCB could not be created as it either does not have a unique name or the name
is longer than 20 characters!\n");
27         int errLen = strlen(errMsg);
28         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
29     }
30     else if (processClass != 'a' && processClass != 's')
31     { // Check if the process has a valid class.
32         char errMsg[100];
33         strcpy(errMsg, "The PCB could not be created as it does not have a valid class!\n");
34         int errLen = strlen(errMsg);
35         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
36     }
37     else if (processPriority < 0 || processPriority > 9)
38     { // Check if the process has a valid priority.
39         char errMsg[100];
40         strcpy(errMsg, "The PCB could not be created as it does not have a valid priority!\n");
41         int errLen = strlen(errMsg);
42         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
43     }
44     else
```

```

45     { // Make the PCB
46         PCB *createdPCB = setupPCB(processName, processClass, processPriority);
47
48         char msg[] = "The PCB was created!\n";
49         int msgLen = strlen(msg);
50         sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
51
52         insertPCB(createdPCB);
53     }
54 }

```

### 5.27.1.3 deletePCB()

```

void deletePCB (
    char *processName )

```

Definition at line 56 of file R2commands.c.

```

57 { // BENJAMIN WILL PROGRAM THIS FUNCTION
58     /*
59     The deletePCB command will remove a PCB from the appropriate queue and then free all associated
60     memory.
61     This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
62     */
63     /* Error Checking:
64     Name must be valid.
65     */
66
67     if (strlen(processName) > 20)
68     { // Check if the process has a valid name.
69         char errMsg[100];
70         strcpy(errMsg, "The PCB could not be deleted as the name is longer than 20 characters!\n");
71         int errLen = strlen(errMsg);
72         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
73     }
74
75     PCB *PCB_to_delete = findPCB(processName);
76
77     if (PCB_to_delete == NULL)
78     {
79         char errMsg[42] = "The PCB you want to remove does not exist\n";
80         int errMsgLen = 42;
81         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
82     }
83     else
84     {
85         int removed = removePCB(PCB_to_delete);
86         if (removed == 1)
87         {
88             char errMsg[] = "The PCB could not be unlinked.\n";
89             int errMsgLen = strlen(errMsg);
90             sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
91         }
92         else
93         {
94             int result = sys_free_mem(PCB_to_delete);
95             if (result == -1)
96             {
97                 char errMsg[50];
98                 strcpy(errMsg, "The PCB could not be successfully deleted\n");
99                 int errLen = strlen(errMsg);
100                 sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
101             }
102             else
103             {
104                 char msg[50];
105                 strcpy(msg, "The desired PCB was deleted\n");
106                 int msgLen = strlen(msg);
107                 sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
108             }
109         }
110     }
111 }

```

### 5.27.1.4 resumePCB()

```
void resumePCB (
    char *processName )
```

Definition at line 187 of file R2commands.c.

```
188 { // COLTON WILL PROGRAM THIS FUNCTION
189     /*
190      Places a PCB in the not suspended state and reinserts it into the appropriate queue
191      */
192
193     PCB *PCBtoResume = findPCB(processName);
194
195     if (PCBtoResume == NULL || strlen(processName) > 20)
196     {
197         char nameError[] = "This is not a valid name.\n";
198         int printCount = strlen(nameError);
199         sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
200     }
201     else
202     {
203         removePCB(PCBtoResume);
204         PCBtoResume->suspendedStatus = 1;
205         insertPCB(PCBtoResume);
206
207         char msg[] = "The PCB was successfully resumed!\n";
208         int msgLen = strlen(msg);
209         sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
210     }
211 }
212 }
```

### 5.27.1.5 setPCBPriorty()

```
void setPCBPriorty (
    char *processName,
    int newProcessPriority )
```

Definition at line 217 of file R2commands.c.

```
218 { // ANASTASE WILL PROGRAM THIS FUNCTION
219
220     // Sets a PCB's priority and reinserts the process into the correct place in the correct queue
221
222     /*
223     Error Checking:
224     Name must be valid.
225     newPriority
226     */
227
228     // find the process and validate the name
229     PCB *tempPCB = findPCB(processName);
230
231     if ((tempPCB != NULL) && (newProcessPriority >= 0) && (newProcessPriority < 10))
232     {
233         tempPCB->priority = newProcessPriority;
234         removePCB(tempPCB);
235         insertPCB(tempPCB);
236
237         char msg[] = "The PCB's priority was successfully changed!\n";
238         int msgLen = strlen(msg);
239         sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
240     }
241 }
```

### 5.27.1.6 showAll()

```
void showAll ( )
```

Definition at line 612 of file R2commands.c.

```
613 { // COLTON WILL PROGRAM THIS FUNCTION
614     /*
615      * Displays the following information for each PCB in the ready and blocked queues:
616      *   Process Name
617      *   Class
618      *   State
619      *   Suspended Status
620      *   Priority
621      */
622     /*
623     * Error Checking:
624     *   None
625     */
626
627     showReady();
628     showSuspendedReady();
629     showBlocked();
630     showSuspendedBlocked();
631 }
```

### 5.27.1.7 showBlocked()

```
void showBlocked ( )
```

Definition at line 559 of file R2commands.c.

```
560 { // ANASTASE WILL PROGRAM THIS FUNCTION
561     /*
562      * Displays the following information for each PCB in the blocked queue:
563      *   Process Name
564      *   Class
565      *   State
566      *   Suspended Status
567      *   Priority
568      *   HEAD
569      */
570     /*
571     * Error Checking:
572     *   None
573     */
574
575     // check
576
577     char print_message[30] = "The blocked queue:\n";
578     int message_size = strlen(print_message);
579     sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
580
581     // printPCBs(blocked);
582     queue *tempQueue = getBlocked();
583     PCB *tempPtr = tempQueue->head; //PCB_container->head;
584     int count = tempQueue->count;
585
586     if (count == 0)
587     {
588         // the queue is empty
589         char error_message[30] = "The queue is empty.\n";
590         int error_size = strlen(error_message);
591         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
592         return;
593     }
594     // The queue is not empty
595
596     int value = 0;
597     // Testing purpose
598     //char print_message[38]="The blocke queue testing:\n";
599     //int message_size=strlen(print_message);
600     //sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
601
602     while (value < count)
603     { // testing for <== or <
604         // Print out the process
605         showPCB(tempPtr->processName);
```

```

606         // increment pcb*tempPtr, the loop variable.
607         tempPtr = tempPtr->nextPCB;
608         value++;
609     }
610 }

```

### 5.27.1.8 showPCB()

```

void showPCB (
    char *processName )

```

Definition at line 243 of file R2commands.c.

```

244 { // BENJAMIN WILL PROGRAM THIS FUNCTION
245     /*
246     Displays the following information for a PCB:
247         Process Name
248         Class
249         State
250         Suspended Status
251         Priority
252     */
253
254     /*
255     Error Checking:
256     Name must be valid.
257     */
258
259     if (strlen(processName) > 20)
260     { // Check if the process has a valid name.
261         char errMsg[100];
262         strcpy(errMsg, "The PCB could not be shown as the name is longer than 20 characters!\n");
263         int errLen = strlen(errMsg);
264         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
265     }
266     else
267     {
268
269         PCB *PCB_to_show = findPCB(processName);
270
271         if (PCB_to_show == NULL)
272         { // Check to see if the PCB exists.
273             char errMsg[100];
274             strcpy(errMsg, "The PCB could not be shown, as it does not exist!\n");
275             int errLen = strlen(errMsg);
276             sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
277         }
278         else
279         {
280             // Print out the PCB name.
281             char nameMsg[50];
282             strcpy(nameMsg, "The process name is: ");
283             int nameMsgLen = strlen(nameMsg);
284             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
285             char name[20];
286             strcpy(name, PCB_to_show->processName);
287             int nameLen = strlen(name);
288             sys_req(WRITE, DEFAULT_DEVICE, name, &nameLen);
289             char newLine[1];
290             strcpy(newLine, "\n");
291             int newLineLen = 1;
292             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);
293
294             // Print out PCB class
295             char classMsg[50];
296             strcpy(classMsg, "The process class is: ");
297             int classMsgLen = strlen(classMsg);
298             sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
299
300             if (PCB_to_show->processClass == 'a')
301             {
302                 char appMsg[50];
303                 strcpy(appMsg, "application");
304                 int appMsgLen = strlen(appMsg);
305                 sys_req(WRITE, DEFAULT_DEVICE, appMsg, &appMsgLen);
306             }
307             else
308             {
309                 char sysMsg[50];

```



```

310         strcpy(sysMsg, "system");
311         int sysMsgLen = strlen(sysMsg);
312         sys_req(WRITE, DEFAULT_DEVICE, sysMsg, &sysMsgLen);
313     }
314     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);
315
316     // Print out the PCB state
317
318     if (PCB_to_show->runningStatus == 0)
319     { // The process is ready.
320         char stateMsg[50];
321         strcpy(stateMsg, "The process is ready!\n");
322         int stateMsgLen = strlen(stateMsg);
323         sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
324     }
325     else if (PCB_to_show->runningStatus == -1)
326     { // The process is blocked.
327         char stateMsg[50];
328         strcpy(stateMsg, "The process is blocked!\n");
329         int stateMsgLen = strlen(stateMsg);
330         sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
331     }
332     else if (PCB_to_show->runningStatus == 1)
333     { // The process is running.
334         char stateMsg[50];
335         strcpy(stateMsg, "The process is running!\n");
336         int stateMsgLen = strlen(stateMsg);
337         sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
338     }
339
340     // Print out the PCB suspended status
341
342     if (PCB_to_show->suspendedStatus == 0)
343     { // The process is suspended
344         char susMsg[50];
345         strcpy(susMsg, "The process is suspended!\n");
346         int susMsgLen = strlen(susMsg);
347         sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
348     }
349     else if (PCB_to_show->suspendedStatus == 1)
350     { // The process is not suspended
351         char susMsg[50];
352         strcpy(susMsg, "The process is not suspended!\n");
353         int susMsgLen = strlen(susMsg);
354         sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
355     }
356
357     // Print out the PCB priority
358     char priorityMsg[50];
359     int priorityMsgLen = 0;
360
361     switch (PCB_to_show->priority)
362     {
363     case 0:
364         strcpy(priorityMsg, "The process priority is 0!\n");
365         priorityMsgLen = strlen(priorityMsg);
366         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
367         break;
368
369     case 1:
370         strcpy(priorityMsg, "The process priority is 1!\n");
371         priorityMsgLen = strlen(priorityMsg);
372         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
373         break;
374
375     case 2:
376         strcpy(priorityMsg, "The process priority is 2!\n");
377         priorityMsgLen = strlen(priorityMsg);
378         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
379         break;
380
381     case 3:
382         strcpy(priorityMsg, "The process priority is 3!\n");
383         priorityMsgLen = strlen(priorityMsg);
384         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
385         break;
386
387     case 4:
388         strcpy(priorityMsg, "The process priority is 4!\n");
389         priorityMsgLen = strlen(priorityMsg);
390         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
391         break;
392
393     case 5:
394         strcpy(priorityMsg, "The process priority is 5!\n");
395         priorityMsgLen = strlen(priorityMsg);
396         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);

```

```

397         break;
398
399     case 6:
400         strcpy(priorityMsg, "The process priority is 6!\n");
401         priorityMsgLen = strlen(priorityMsg);
402         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
403         break;
404
405     case 7:
406         strcpy(priorityMsg, "The process priority is 7!\n");
407         priorityMsgLen = strlen(priorityMsg);
408         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
409         break;
410
411     case 8:
412         strcpy(priorityMsg, "The process priority is 8!\n");
413         priorityMsgLen = strlen(priorityMsg);
414         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
415         break;
416
417     case 9:
418         strcpy(priorityMsg, "The process priority is 9!\n");
419         priorityMsgLen = strlen(priorityMsg);
420         sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
421         break;
422
423     default:
424         break;
425     }
426 }
427 }
428 }

```

### 5.27.1.9 showReady()

```
void showReady ( )
```

Definition at line 430 of file R2commands.c.

```

431 { // COLTON WILL PROGRAM THIS FUNCTION
432     /*
433     Displays the following information for each PCB in the ready queue:
434         Process Name
435         Class
436         State
437         Suspended Status
438         Priority
439     */
440     /*
441     Error Checking:
442     None
443     */
444
445     char message[] = "Printing the ready queue:\n";
446     int messLength = strlen(message);
447     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
448
449     queue *tempQueue = getReady();
450     PCB *tempPCB = tempQueue->head;
451
452     int loop = 0;
453     int count = tempQueue->count;
454
455     if (count == 0)
456     {
457         // the queue is empty
458         char error_message[30] = "The queue is empty.\n";
459         int error_size = strlen(error_message);
460         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
461         return;
462     }
463
464     while (loop < count)
465     {
466         showPCB(tempPCB->processName);
467         PCB *tempNext = tempPCB->nextPCB;
468         loop++;
469         tempPCB = tempNext;
470     }
471 }

```

**5.27.1.10 showSuspendedBlocked()**

```
void showSuspendedBlocked ( )
```

Definition at line 516 of file R2commands.c.

```
517 { // COLTON WILL PROGRAM THIS FUNCTION
518     /*
519     Displays the following information for each PCB in the suspended blocked queue:
520         Process Name
521         Class
522         State
523         Suspended Status
524         Priority
525     */
526     /*
527     Error Checking:
528     None
529     */
530
531     char message[] = "Printing the suspended blocked queue:\n";
532     int messLength = strlen(message);
533     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
534
535     queue *tempQueue = getSuspendedBlocked();
536     PCB *tempPCB = tempQueue->head;
537
538     int loop = 0;
539     int count = tempQueue->count;
540
541     if (count == 0)
542     {
543         // the queue is empty
544         char error_message[30] = "The queue is empty.\n";
545         int error_size = strlen(error_message);
546         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
547         return;
548     }
549
550     while (loop < count)
551     {
552         showPCB(tempPCB->processName);
553         PCB *tempNext = tempPCB->nextPCB;
554         loop++;
555         tempPCB = tempNext;
556     }
557 }
```

**5.27.1.11 showSuspendedReady()**

```
void showSuspendedReady ( )
```

Definition at line 473 of file R2commands.c.

```
474 { // COLTON WILL PROGRAM THIS FUNCTION
475     /*
476     Displays the following information for each PCB in the suspended ready queue:
477         Process Name
478         Class
479         State
480         Suspended Status
481         Priority
482     */
483     /*
484     Error Checking:
485     None
486     */
487
488     char message[] = "Printing the suspended ready queue:\n";
489     int messLength = strlen(message);
490     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
491
492     queue *tempQueue = getSuspendedReady();
493     PCB *tempPCB = tempQueue->head;
494
495     int loop = 0;
496     int count = tempQueue->count;
497 }
```

```

498     if (count == 0)
499     {
500         // the queue is empty
501         char error_message[30] = "The queue is empty.\n";
502         int error_size = strlen(error_message);
503         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
504         return;
505     }
506
507     while (loop < count)
508     {
509         showPCB(tempPCB->processName);
510         PCB *tempNext = tempPCB->nextPCB;
511         loop++;
512         tempPCB = tempNext;
513     }
514 }

```

### 5.27.1.12 suspendPCB()

```

void suspendPCB (
    char *processName )

```

Definition at line 157 of file R2commands.c.

```

158 { // COLTON WILL PROGRAM THIS FUNCTION
159     /*
160      Places a PCB in the suspended state and reinserts it into the appropriate queue
161      */
162
163     PCB *PCBtoSuspend = findPCB(processName);
164
165     if (PCBtoSuspend == NULL || strlen(processName) > 20)
166     {
167         char nameError[] = "This is not a valid name.\n";
168         int printCount = strlen(nameError);
169         sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
170     }
171     else
172     {
173         removePCB(PCBtoSuspend);
174         PCBtoSuspend->suspendedStatus = 0;
175         insertPCB(PCBtoSuspend);
176
177         char msg[] = "The PCB was successfully suspended!\n";
178         int msgLen = strlen(msg);
179         sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
180     }
181 }
182 }

```

### 5.27.1.13 unblockPCB()

```

void unblockPCB (
    char *processName )

```

Definition at line 131 of file R2commands.c.

```

132 { // ANASTASE WILL PROGRAM THIS FUNCTION
133     /*
134      Places a PCB in the unblocked state and reinserts it into the appropriate queue.
135      */
136     /*
137     Error Checking:
138     Name must be valid.
139     */
140
141     PCB *pcb_to_unblock = findPCB(processName);
142     if (pcb_to_unblock != NULL)
143     {

```

```
146     pcb_to_unblock->runningStatus = 0; // ready
147     removePCB(pcb_to_unblock);          // is this the right place to put that function?
148     insertPCB(pcb_to_unblock);
149
150     char msg[] = "The PCB was successfully unblocked!\n";
151     int msgLen = strlen(msg);
152     sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
153 }
154 }
```

## 5.28 README.md File Reference

