MPX-Fall2020-Group9

2

Generated by Doxygen 1.9.0

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

4.6 heap Struct Reference	13
4.6.1 Detailed Description	13
4.6.2 Member Data Documentation	13
<b>4.6.2.1</b> 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
<b>4.7.2.3</b> 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
<b>4.8.2.1</b> 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
<b>4.9.2.2</b> empty	17
<b>4.9.2.3 size</b>	17
4.10 index_table Struct Reference	17
4.10.1 Detailed Description	17
4.10.2 Member Data Documentation	17
<b>4.10.2.1</b> id	17
<b>4.10.2.2</b> table	18
4.11 page_dir Struct Reference	18
4.11.1 Detailed Description	18
4.11.2 Member Data Documentation	18
<b>4.11.2.1</b> 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
<b>4.12.2.2</b> dirty	19
<b>4.12.2.3</b> 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
File Documentation	27
5.1 include/core/asm.h File Reference	
5.2 include/core/interrupts.h File Reference	
5.2.1 Function Documentation	
5.2.1.1 init_irq()	
5.2.1.2 init_pic()	
5.3 include/core/io.h File Reference	. 28

5

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
5.5.1.1attribute()	. 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
5.15.2.1end	 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 memset()	80
	5.17.1.4 strcat()	80
	5.17.1.5 strcmp()	81
	5.17.1.6 strcpy()	81
	5.17.1.7 strlen()	81
	5.17.1.8 strtok()	82
5.18 modules	s/mpx_supt.c File Reference	82
5.18.1	Function Documentation	83
	5.18.1.1 idle()	83
	5.18.1.2 mpx_init()	83
	5.18.1.3 sys_alloc_mem()	83
	5.18.1.4 sys_free_mem()	84
	5.18.1.5 sys_req()	84
	5.18.1.6 sys_set_free()	85
	5.18.1.7 sys_set_malloc()	85
5.18.2	Variable Documentation	85
	5.18.2.1 current_module	85
	5.18.2.2 io_module_active	85
	5.18.2.3 mem_module_active	85
	5.18.2.4 params	86
	5.18.2.5 student_free	86
	5.18.2.6 student_malloc	86
5.19 modules	s/mpx_supt.h File Reference	86
5.19.1 l	Macro Definition Documentation	87
	5.19.1.1 COM_PORT	87
	5.19.1.2 DEFAULT_DEVICE	87
	5.19.1.3 EXIT	87
	5.19.1.4 FALSE	87
	5.19.1.5 IDLE	88
	5.19.1.6 INVALID_BUFFER	88
	5.19.1.7 INVALID_COUNT	88
	5.19.1.8 INVALID_OPERATION	88
	5.19.1.9 IO_MODULE	88
	5.19.1.10 MEM_MODULE	88
	5.19.1.11 MODULE_F	89
	5.19.1.12 MODULE_R1	89
	5.19.1.13 MODULE_R2	89
	5.19.1.14 MODULE_R3	89
	5.19.1.15 MODULE_R4	89
	5.19.1.16 MODULE_R5	89
	5.19.1.17 READ	90
	5.19.1.18 TRUE	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()	19
5.24.1.5 getBlocked()	19
5.24.1.6 getReady()	20
5.24.1.7 getSuspendedBlocked()	20
5.24.1.8 getSuspendedReady()	20
5.24.1.9 insertPCB()	20
5.24.1.10 removePCB()	21
5.24.1.11 setupPCB()	23
5.24.2 Variable Documentation	23
5.24.2.1 blocked	24
5.24.2.2 ready	24
5.24.2.3 suspendedBlocked	24
5.24.2.4 suspendedReady	24
5.25 modules/R2/R2_Internal_Functions_And_Structures.h File Reference	24
5.25.1 Typedef Documentation	25
5.25.1.1 PCB	25
5.25.1.2 queue	25
5.25.2 Function Documentation	25
5.25.2.1 allocatePCB()	25
5.25.2.2 allocateQueues()	26
5.25.2.3 findPCB()	26
5.25.2.4 freePCB()	27
5.25.2.5 getBlocked()	27
5.25.2.6 getReady()	28
5.25.2.7 getSuspendedBlocked()	28
5.25.2.8 getSuspendedReady()	28
5.25.2.9 insertPCB()	28
5.25.2.10 removePCB()	29
5.25.2.11 setupPCB()	31
5.26 modules/R2/R2commands.c File Reference	31
5.26.1 Function Documentation	32
5.26.1.1 blockPCB()	32
5.26.1.2 createPCB()	33
5.26.1.3 deletePCB()	33
5.26.1.4 resumePCB()	34
5.26.1.5 setPCBPriority()	35
5.26.1.6 showAll()	35
5.26.1.7 showBlocked()	35
5.26.1.8 showPCB()	
5.26.1.9 showReady()	38
5.26.1.10 showSuspendedBlocked()	39
5.26.1.11 showSuspendedReady()	40

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:

date_time	
	9
gdt_descriptor_struct	9
gdt_entry_struct	
header	
heap	
idt_entry_struct	
idt_struct	
index_entry	
index_table	
page_dir	
page_entry	
page_table	
param	
PCB	22
queue	24

4 Class Index

# **Chapter 3**

# File Index

# 3.1 File List

Here is a list of all files with brief descriptions:

include/string.h
include/system.h
include/core/asm.h
include/core/interrupts.h
include/core/io.h
include/core/serial.h
include/core/tables.h
include/mem/heap.h
include/mem/paging.h
kernel/core/interrupts.c
kernel/core/kmain.c
kernel/core/serial.c
kernel/core/system.c
kernel/core/tables.c
kernel/mem/heap.c
kernel/mem/paging.c
lib/string.c
modules/mpx_supt.c
modules/mpx_supt.h
modules/R1/commhand.c
modules/R1/commhand.h
modules/R1/R1commands.c
modules/R1/R1 commands.h
modules/R2/R2_Internal_Functions_And_Structures.c
modules/R2/R2_Internal_Functions_And_Structures.h
modules/R2/R2commands.c
modules/B2/B2commands.h

6 File Index

# **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 monint 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.

4.2 footer Struct Reference 9

#### 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

```
u16int 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 PCB Struct Reference 23

## 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.

24 Class Documentation

#### 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

26 Class Documentation

# **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.

```
76 (u32int) overflow,
(u32int) overflow,
(u32int) debug,
(u32int) bounds,
(u32int) device_not_available,
(u32int) device_not_available,
(u32int) device_not_segment,
(u32int) overcosor_segment,
```

```
82
       (u32int)invalid_tss,
       (u32int) segment_not_present,
84
        (u32int) stack_segment,
8.5
       (u32int)general_protection,
86
       (u32int)page_fault,
       (u32int) reserved,
88
       (u32int) coprocessor
89
90
91
    // Install handlers; 0x08=sel, 0x8e=flags
   for(i=0; i<32; i++){
   if (i<17) idt_set_gate(i, isrs[i], 0x08, 0x8e);</pre>
92
93
       else idt_set_gate(i, (u32int)reserved, 0x08, 0x8e);
96
    // Ignore interrupts from the real time clock
    idt_set_gate(0x08, (u32int)rtc_isr, 0x08, 0x8e);
98 1
```

#### 5.2.1.2 init\_pic()

```
void init_pic (
     void )
```

#### Definition at line 106 of file interrupts.c.

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

Definition at line 15 of file io.h.

#### 5.3.1.2 outb

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()

## Definition at line 22 of file serial.c.

#### 5.4.2.2 polling()

```
int∗ polling (
               char * buffer,
               int * count )
Definition at line 92 of file serial.c.
     // insert your code to gather keyboard input via the technique of polling.
95
96
     char keyboard_character;
97
98
     int cursor = 0;
99
     char log[] = {' \mid 0', ' \mid 0', ' \mid 0', ' \mid 0'};
100
101
102
     int characters_in_buffer = 0;
103
      while (1)
104
105
106
107
        if (inb(COM1 + 5) & 1)
108
                                            // is there input char?
          keyboard_character = inb(COM1); //read the char from COM1
109
110
          if (keyboard_character == '\n' || keyboard_character == '\r')
111
          { // HANDLEING THE CARRIAGE RETURN AND NEW LINE CHARACTERS
112
113
114
            buffer[characters_in_buffer] = ' \0';
115
116
          else if ((keyboard_character == 127 || keyboard_character == 8) && cursor > 0)
117
          { // HANDELING THE BACKSPACE CHARACTER
118
119
120
            //serial_println("Handleing backspace character.");
121
            serial_print("\033[K");
122
            buffer[cursor - 1] = ' \setminus 0';
123
            serial_print("\b \b");
124
            serial_print(buffer + cursor);
125
126
            cursor--;
127
128
            int temp_cursor = cursor;
129
130
            while (buffer[temp_cursor + 1] != ' \0')
131
132
              buffer[temp_cursor] = buffer[temp_cursor + 1];
              buffer[temp_cursor + 1] = ' \0';
133
134
              temp_cursor++;
135
136
137
            characters in buffer --;
138
            cursor = characters_in_buffer;
139
          else if (keyboard_character == '~' && cursor < 99)</pre>
140
          { //HANDLING THE DELETE KEY
141
142
            // \033[3~
143
            serial_print("\033[K");
144
145
            buffer[cursor + 1] = ' \0';
serial_print("\b \b");
146
147
148
            serial_print(buffer + cursor);
149
150
            int temp_cursor = cursor + 1;
151
            while (buffer[temp_cursor + 1] != ' \0')
152
153
              buffer[temp_cursor] = buffer[temp_cursor + 1];
154
              buffer[temp_cursor + 1] = ' \0';
155
156
               temp_cursor++;
157
158
159
            characters_in_buffer--;
160
            cursor = characters_in_buffer;
161
162
          else if (keyboard_character == '\033')
163
          { // HANDLEING FIRST CHARACTER FOR ARROW KEYS
164
165
            log[0] = keyboard_character;
166
167
          else if (keyboard_character == '[' && log[0] == '\033')
168
          { // HANDLEING SECOND CHARACTER FOR ARROW KEYS
169
```

```
170
            log[1] = keyboard_character;
171
          else if (log[0] == '\033' && log[1] == '[')
172
          { // HANDLEING LAST CHARACTER FOR ARROW KEYS
173
174
            log[2] = keyboard_character;
175
176
            if (keyboard_character == 'A')
177
            { //Up arrow
              //Call a history function from the commhand or do nothing
178
179
180
            else if (keyboard_character == 'B')
            { //Down arrow
181
              //Call a history command from the commhand or do nothing
182
183
184
             else if (keyboard_character == 'C' && cursor != 99)
185
            { //Right arrow
186
               serial_print("\033[C");
187
188
              cursor++;
189
190
            else if (keyboard_character == 'D' && cursor != 0)
191
            { //Left arrow
192
              serial_print("\033[D");
193
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;
               serial_print(&keyboard_character);
205
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
              char temp_buffer[strlen(buffer)];
memset(temp_buffer, ' \0', strlen(buffer));
216
217
218
219
               int temp_cursor = 0;
               while (temp_cursor <= characters_in_buffer) //Filling the temp_buffer with all of the
220
       characters from buffer, and inserting the new character.
221
222
                 if (temp_cursor < cursor)</pre>
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
                  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
              {
                buffer[temp_cursor] = temp_buffer[temp_cursor];
241
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 ( {\tt const~char~ * \it 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()

## 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()

#### 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()

## 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\_\_()

#### 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 & 0xFFFFF);
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 & 0xFO;
67    new_entry->access = access;
68 }
```

#### 5.5.1.3 idt set gate()

#### 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->select = 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.

## 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    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

```
ul6int 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

ul6int 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()

Definition at line 24 of file heap.c.

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

## 5.6.2.2 alloc()

## 5.6.2.3 init\_kheap()

```
void init_kheap ( )
```

## 5.6.2.4 kfree()

```
u32int kfree ( )
```

## 5.6.2.5 kmalloc()

#### Definition at line 52 of file heap.c.

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

#### 5.6.2.6 make\_heap()

## 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()

#### 5.7.2.2 first\_free()

```
u32int first_free ( )
```

#### 5.7.2.3 get\_bit()

## Definition at line 56 of file paging.c.

```
5/ {
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()

## Definition at line 85 of file paging.c.

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

#### 5.7.2.5 init\_paging()

void init\_paging ( )

```
Definition at line 111 of file paging.c.
112 {
      //create frame bitmap
113
      nframes = (u32int) (mem_size/page_size);
frames = (u32int*)kmalloc(nframes/32);
114
115
116
      memset(frames, 0, nframes/32);
117
118
      //create kernel directory
      kdir = (page_dir*)_kmalloc(sizeof(page_dir), 1, 0); //page aligned
memset(kdir, 0, sizeof(page_dir));
119
120
121
122
      //get pages for kernel heap
123
      u32int i = 0x0;
      for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=1){</pre>
124
125
        get_page(i,kdir,1);
126
127
128
      //perform identity mapping of used memory
129
      //note: placement_addr gets incremented in get_page,
//so we're mapping the first frames as well
130
      i = 0 \times 0:
131
      while (i < (phys_alloc_addr+0x10000)) {</pre>
132
       new_frame (get_page(i,kdir,1));
i += page_size;
133
134
135
136
      //allocate heap frames now that the placement addr has increased.
137
      //placement addr increases here for heap
for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN);i+=PAGE_SIZE){</pre>
138
139
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.
160
      cdir = new_dir;
161
      asm volatile ("mov %0,%%cr3":: "b"(&cdir->tables_phys[0]));
162
      u32int cr0;
      asm volatile ("mov %%cr0,%0": "=b"(cr0));
163
164
     cr0 |= 0x80000000;
    asm volatile ("mov %0,%%cr0":: "b"(cr0));
5.7.2.7 new frame()
void new_frame (
                page_entry * page )
```

u32int index;

174 {

Definition at line 173 of file paging.c.

```
if (page->frameaddr != 0) return;
if ( (u32int) (-1) == (index=find_free()) ) kpanic("Out of memory");

//mark a frame as in-use
set_bit(index*page_size);
page->present = 1;
page->frameaddr = index;
page->writeable = 1;
page->usermode = 0;
```

## 5.7.2.8 set\_bit()

```
void set_bit (  \mbox{u32int } \mbox{addr })
```

#### Definition at line 32 of file paging.c.

## 5.8 include/string.h File Reference

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

## **Functions**

- int isspace (const char \*c)
- void \*memset (void \*s, 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.
50
     int res=0;
      int charVal=0;
char sign = ' ';
char c = *s;
51
52
53
54
56
      while(isspace(&c)){ ++s; c = *s;} // advance past whitespace
57
58
      if (*s == '-' | | *s == '+' |) sign = *(s++); // save the sign
59
60
61
      while(*s != '\0'){
      charVal = *s - 48;
res = res * 10 + charVal;
63
64
6.5
       s++;
66
68
69
     if ( sign == '-') res=res * -1;
70
71
    return res; // return integer
72
73 }
```

#### 5.8.1.2 isspace()

```
int isspace (  {\tt const\ char\ *c\ )}
```

## Definition at line 119 of file string.c.

## 5.8.1.3 memset()

```
\label{eq:condition} \begin{array}{c} \operatorname{void} * \operatorname{memset} \ ( \\ \operatorname{void} * s, \\ \operatorname{int} \ c, \\ \operatorname{size\_t} \ n \ ) \end{array}
```

#### 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()

#### 5.8.1.5 strcmp()

#### Definition at line 79 of file string.c.

## 5.8.1.6 strcpy()

#### 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()

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

## 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 \ ) \ \ \mbox{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

 ${\tt typedef\ unsigned\ long\ u32int}$ 

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

#### 5.9.3.2 klogv()

```
void klogv ( {\tt const\ char\ \bigstar \it 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 ( {\tt const\ char\ \bigstar \it msg\ )}
```

#### Definition at line 24 of file system.c.

```
cli(); //disable interrupts
clar logmsg[64] = {'\0'}, prefix[] = "Panic: ";
strcat(logmsg, prefix);
strcat(logmsg, msg);
logv(logmsg);
hlt(); //halt
```

## 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.

#### 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.

## 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
70
     // Necessary interrupt handlers for protected mode
71
     u32int isrs[17] = {
72
        (u32int)divide_error,
        (u32int) debug,
73
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,
       (u32int)coprocessor_segment,
81
        (u32int)invalid_tss,
83
        (u32int) segment_not_present,
84
        (u32int) stack_segment,
85
        (u32int)general_protection,
86
        (u32int)page_fault,
        (u32int) reserved,
88
       (u32int)coprocessor
89
    };
90
    // Install handlers; 0x08=sel, 0x8e=flags
91
    for(i=0; i<32; i++){
  if (i<17) idt_set_gate(i, isrs[i], 0x08, 0x8e);</pre>
92
       else idt_set_gate(i, (u32int)reserved, 0x08, 0x8e);
95
    ^{\prime}// Ignore interrupts from the real time clock
96
     idt_set_gate(0x08, (u32int)rtc_isr, 0x08, 0x8e);
97
98 }
```

#### 5.10.2.29 init pic()

```
void init_pic (
     void )
```

# Definition at line 106 of file interrupts.c.

```
107 {
      outb(PIC1,ICW1); //send initialization code words 1 to PIC1
108
109
      io_wait();
      outb(PIC2,ICW1); //send icw1 to PIC2
110
111
      io_wait();
112
      outb(PIC1+1,0x20); //icw2: remap irq0 to 32
     io_wait();
outb(PIC2+1,0x28); //icw2: remap irq8 to 40
113
114
115
     io wait();
     outb(PIC1+1,4); //icw3
116
117
      io_wait();
118
      outb(PIC2+1,2); //icw3
     io_wait();
outb(PIC1+1,ICW4); //icw4: 80x86, automatic handling
119
120
121
      io wait();
      outb(PIC2+1,ICW4); //icw4: 80x86, automatic handling
123
      io_wait();
124
      outb(PIC1+1,0xFF); //disable irqs for PIC1
125
     io_wait();
     outb(PIC2+1,0xFF); //disable irqs for PIC2
126
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 <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.
        // extern uint32 t magic:
29
       // Uncomment if you want to access the multiboot header // extern void *mbd;
30
31
       // char *boot_loader_name = (char*)((long*)mbd)[16];
33
34
       // 0) Initialize Serial I/O
      // functions to initialize serial I/O can be found in serial.c // there are 3 functions to call
35
36
37
       set_serial_in(COM1);
39
40
       set_serial_out(COM1);
41
       klogv("Starting MPX boot sequence...");
klogv("Initialized serial I/O on COM1 device...");
42
43
45
       // 1) Initialize the support software by identifying the current
46
               MPX Module. This will change with each module.
       // you will need to call mpx_init from the mpx_supt.c
47
48
49
       mpx init (MODULE R2);
51
       // 2) Check that the boot was successful and correct when using grub
       // Comment this when booting the kernel directly using QEMU, etc. //if ( magic != 0x2BADB002 ){
52
53
       // kpanic("Boot was not error free. Halting.");
54
55
56
       // 3) Descriptor Tables -- tables.c
       // you will need to initialize the grown // this keeps track of allocated segments and pages
59
60
       klogv("Initializing descriptor tables...");
61
62
       init qdt();
       init_idt();
63
64
65
       init_pic();
66
       sti();
67
       // 4) Interrupt vector table -- tables.c
68
       // this creates and initializes a default interrupt vector table
69
70
       // this function is in tables.c
71
72
       init_irq();
73
74
       klogv("Interrupt vector table initialized!");
75
       // 5) Virtual Memory -- paging.c -- init_paging
       // this function creates the kernel's heap
// from which memory will be allocated when the program calls
77
78
       // sys_alloc_mem UNTIL the memory management module is completed
// this allocates memory using discrete "pages" of physical memory
// NOTE: You will only have about 70000 bytes of dynamic memory
79
80
81
83
       klogv("Initializing virtual memory...");
84
8.5
       init_paging();
86
       // 6) Call YOUR command handler - interface method
87
       klogv("Transferring control to commhand...");
89
       commhand();
90
91
       // 7) System Shutdown on return from your command handler
92
       klogv("Starting system shutdown procedure...");
93
       /* Shutdown Procedure */
95
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()

#### 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
   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
    outb(device + 2, 0xC7);
outb(device + 4, 0x0B);
29
                                           //enable fifo, clear, 14byte threshold
30
                                           //enable interrupts, rts/dsr set
     (void) inb (device);
                                          //read bit to reset port
31
     return NO_ERROR;
```

#### 5.12.2.2 polling()

```
int∗ polling (
                char * buffer,
                int * count )
Definition at line 92 of file serial.c.
     // insert your code to gather keyboard input via the technique of polling.
95
96
     char keyboard_character;
97
98
     int cursor = 0;
99
     char log[] = {' \mid 0', ' \mid 0', ' \mid 0', ' \mid 0'};
100
101
102
      int characters_in_buffer = 0;
103
      while (1)
104
105
106
107
        if (inb(COM1 + 5) & 1)
108
                                              // is there input char?
          keyboard_character = inb(COM1); //read the char from COM1
109
110
           if (keyboard_character == '\n' || keyboard_character == '\r')
111
112
           { // HANDLEING THE CARRIAGE RETURN AND NEW LINE CHARACTERS
113
114
            buffer[characters_in_buffer] = ' \0';
115
116
          else if ((keyboard_character == 127 || keyboard_character == 8) && cursor > 0)
117
           { // HANDELING THE BACKSPACE CHARACTER
118
119
120
             //serial_println("Handleing backspace character.");
121
             serial_print("\033[K");
122
            buffer[cursor - 1] = ' \setminus 0';
123
             serial_print("\b \b");
124
             serial_print(buffer + cursor);
125
126
127
128
            int temp_cursor = cursor;
129
130
             while (buffer[temp_cursor + 1] != ' \0')
131
               buffer[temp_cursor] = buffer[temp_cursor + 1];
buffer[temp_cursor + 1] = ' \ 0';
132
133
134
               temp_cursor++;
135
136
137
             characters_in_buffer--;
138
             cursor = characters_in_buffer;
139
           else if (keyboard_character == '~' && cursor < 99)</pre>
140
           { //HANDLING THE DELETE KEY
141
142
             // \033[3~
143
144
             serial_print("\033[K");
145
            buffer[cursor + 1] = '\0';
serial_print("\b\b");
serial_print(buffer + cursor);
146
147
148
149
150
             int temp_cursor = cursor + 1;
151
             while (buffer[temp_cursor + 1] != ' \ 0')
152
153
              buffer[temp_cursor] = buffer[temp_cursor + 1];
154
               buffer[temp_cursor + 1] = ' \0';
155
156
               temp_cursor++;
157
158
159
             characters_in_buffer--;
160
             cursor = characters_in_buffer;
161
162
           else if (keyboard_character == '\033')
163
           { // HANDLEING FIRST CHARACTER FOR ARROW KEYS
164
165
             log[0] = keyboard_character;
166
167
           else if (keyboard_character == '[' && log[0] == '\033')
168
           { // HANDLEING SECOND CHARACTER FOR ARROW KEYS
169
```

```
170
            log[1] = keyboard_character;
171
          else if (log[0] == '\033' && log[1] == '[')
172
          { // HANDLEING LAST CHARACTER FOR ARROW KEYS
173
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')
            { //Down arrow
181
              //Call a history command from the commhand or do nothing
182
183
184
             else if (keyboard_character == 'C' && cursor != 99)
185
            { //Right arrow
186
              serial print("\033[C");
187
188
              cursor++;
189
190
            else if (keyboard_character == 'D' && cursor != 0)
191
            { //Left arrow
192
              serial_print("\033[D");
193
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;
              serial_print(&keyboard_character);
205
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
              char temp_buffer[strlen(buffer)];
memset(temp_buffer, ' \0', strlen(buffer));
216
217
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)</pre>
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
                  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
                buffer[temp_cursor] = temp_buffer[temp_cursor];
241
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 (  {\tt const~char~ * \it 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()

# 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()

### 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()

# 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()

# 5.13.1.2 kpanic()

# 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 & 0xFO;
67    new_entry->access = access;
68 }
```

### 5.14.1.2 idt\_set\_gate()

#### Definition at line 27 of file tables.c.

```
29 {
30   idt_entry *new_entry = &idt_entries[idx];
31   new_entry->base_low = (base & 0xfffff);
32   new_entry->base_high = (base » 16) & 0xfffff;
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 {
       gdt_ptr.limit = 5 * sizeof(gdt_entry) - 1;
gdt_ptr.base = (u32int) gdt_entries;
77
78
79
      u32int limit = 0xFFFFFFF;
80
     gdt_init_entry(0, 0, 0, 0, 0);
82
       gdt_init_entry(1, 0, limit, 0x9A, 0xCF); //code segment
      gdt_init_entry(2, 0, limit, 0x92, 0xCF); //data segment gdt_init_entry(3, 0, limit, 0xFA, 0xCF); //user mode code segment gdt_init_entry(4, 0, limit, 0xF2, 0xCF); //user mode data segment
8.3
84
85
       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 {
        idt_ptr.limit = 256*sizeof(idt_descriptor) - 1;
        idt_ptr.base = (u32int)idt_entries;
        memset(idt_entries, 0, 256*sizeof(idt_descriptor));
        write_idt_ptr((u32int)&idt_ptr);
        }
```

# 5.14.1.5 write\_gdt\_ptr()

# 5.14.1.6 write\_idt\_ptr()

# 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

```
{\tt 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 <u>end</u>
- void <u>end</u>
- 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.
26
     u32int *addr;
27
     // Allocate on the kernel heap if one has been created
28
     if (kheap != 0) {
29
       addr = (u32int*)alloc(size, kheap, page_align);
30
        if (phys_addr) {
32
          page_entry *page = get_page((u32int)addr, kdir, 0);
33
          *phys_addr = (page->frameaddr*0x1000) + ((u32int)addr & 0xFFF);
34
       return (u32int)addr;
3.5
36
37
     // Else, allocate directly from physical memory
      if (page_align && (phys_alloc_addr & 0xffffff000)) {
   phys_alloc_addr &= 0xfffff000;
   phys_alloc_addr += 0x1000;
39
40
41
42
43
       addr = (u32int*)phys_alloc_addr;
44
       if (phys_addr) {
45
         *phys_addr = phys_alloc_addr;
46
       phys_alloc_addr += size;
47
       return (u32int)addr;
48
49
```

# 5.15.1.2 alloc()

#### 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()

### 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 \*cdir = 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] != 0xFFFFFFFFF) //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 }</pre>
```

# 5.16.1.3 get\_bit()

# 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.
87
      u32int phys_addr;
      u32int index = addr / page_size / 1024;
u32int offset = addr / page_size % 1024;
88
89
90
      //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) {
       dir->tables[index] = (page_table*)_kmalloc(sizeof(page_table), 1, &phys_addr);
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
      nframes = (u32int) (mem_size/page_size);
frames = (u32int*)kmalloc(nframes/32);
114
115
116
      memset(frames, 0, nframes/32);
117
118
      //create kernel directory
119
      kdir = (page_dir*)_kmalloc(sizeof(page_dir), 1, 0); //page aligned
      memset(kdir, 0, sizeof(page_dir));
120
121
122
      //get pages for kernel heap
      u32int i = 0x0;
123
124
      for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=1){</pre>
125
        get_page(i,kdir,1);
126
127
      // {\tt perform\ identity\ mapping\ of\ used\ memory}
128
      //note: placement_addr gets incremented in get_page, //so we're mapping the first frames as well
129
130
131
       i = 0x0;
132
      while (i < (phys_alloc_addr+0x10000)) {</pre>
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){</pre>
140
        new_frame(get_page(i,kdir,1));
141
142
       //load the kernel page directory; enable paging
143
144
      load_page_dir(kdir);
145
      //setup the kernel heap
kheap = make_heap(KHEAP_BASE, KHEAP_SIZE, KHEAP_BASE+KHEAP_MIN);
146
147
148 }
```

#### 5.16.1.6 load\_page\_dir()

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

# 5.17.1.2 isspace()

```
int isspace (  {\tt const\ char\ *\it c}\ )
```

Definition at line 119 of file string.c.

# 5.17.1.3 memset()

```
\begin{tabular}{ll} \begin{tabular}{ll} void * s, \\ int $c$, \\ size\_t $n$ ) \end{tabular}
```

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()

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

# 5.17.1.6 strcpy()

### 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 ( {\rm 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.
      static char *tok_tmp = NULL;
153
154
      const char *p = s2;
155
156
      //new string
      if (s1!=NULL) {
157
158
       tok\_tmp = s1;
159
      //old string cont'd
160
161
      else {
      if (tok_tmp==NULL) {
162
163
          return NULL;
164
        s1 = tok\_tmp;
165
166
167
      //skip leading s2 characters
168
      while ( *p && *s1 ) {
  if (*s1==*p) {
169
170
        ++s1;
p = s2;
continue;
171
172
173
174
175
        ++p;
176
177
      //no more to parse
178
      if (!*s1){
179
       return (tok_tmp = NULL);
180
181
182
183
      //skip non-s2 characters
184
      tok\_tmp = s1;
      while (*tok_tmp) {
  p = s2;
185
186
        while (*p){
187
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;
```

# 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(\frac{\pmu}{u32int}))
- void sys\_set\_free (int(\(\frac{1}{3}\)func)(void \(\frac{1}{3}\))
- 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 {
         char msg[30];
175
176
        int count=0;
177
          memset( msg, '\0', sizeof(msg));
strcpy(msg, "IDLE PROCESS EXECUTING.\n");
count = strlen(msg);
178
179
180
181
182
         sys_req( WRITE, DEFAULT_DEVICE, msg, &count);
sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
183
184
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()

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 }
```

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

#### 159 {

### 5.18.1.5 sys\_req()

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

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

### 5.18.1.6 sys\_set\_free()

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

### 5.18.1.7 sys\_set\_malloc()

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(\frac{\pmu}{u32int}))
- void sys\_set\_free (int(\(\frac{\pm}{\text{func}}\))(void \(\frac{\pm}{\pm}\))
- 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;
```

# 5.19.2.3 sys\_alloc\_mem()

# 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.

#### 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
        // triger interrupt 60h to invoke
        params.op_code = op_code;
asm volatile ("int $60");
60
61
62
    }// idle or exit
63
     else if (op_code == READ || op_code == WRITE) {
        // validate buffer pointer and count pointer
        if (buffer_ptr == NULL)
  return_code = INVALID_BUFFER;
67
        else if (count_ptr == NULL | | *count_ptr <= 0)
return_code = INVALID_COUNT;</pre>
68
69
70
71
        // if parameters are valid store in the params structure
        if ( return_code == 0) {
  params.op_code = op_code;
72
73
          params.device_id = device_id;
params.buffer_ptr = buffer_ptr;
params.count_ptr = count_ptr;
74
75
76
78
          if (!io_module_active) {
79
            // if default device
            if (op_code == READ)
  return_code = *(polling(buffer_ptr, count_ptr));
80
81
82
83
            else //must be WRITE
84
               return_code = serial_print(buffer_ptr);
85
          } else {// I/O module is implemented
asm volatile ("int $60");
86
87
          } // NOT IO_MODULE
88
89
    } else return_code = INVALID_OPERATION;
91
      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;
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;
```

# 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
        {\tt char welcomeMSG[] = "\nWelcome to our CS 450 Project!\nType help to see what you can do!\n\n";}
13
        int welcomeLength = strlen(welcomeMSG);
sys_req(WRITE, DEFAULT_DEVICE, welcomeMSG, &welcomeLength);
14
15
16
17
        char cmdBuffer[100];
18
        int bufferSize;
19
        allocateQueues();
20
        int quitFlag = 0;
21
22
23
        while (!quitFlag)
24
25
             //get a command: cal polling fx
26
            memset (cmdBuffer, ' \0', 100);
2.7
28
            bufferSize = 99; // reset size before each call to read
29
31
             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
32
             char newLine[] = "\n";
33
             int newLineCount = 1;
34
             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
35
36
37
             if (strcmp(cmdBuffer, "help") == 0)
38
39
                 help();
40
             else if (strcmp(cmdBuffer, "version") == 0)
41
43
44
45
            else if (strcmp(cmdBuffer, "getDate") == 0)
46
                 getDate();
49
             else if (strcmp(cmdBuffer, "setDate") == 0)
50
51
                 setDate();
52
53
            else if (strcmp(cmdBuffer, "getTime") == 0)
55
                 getTime();
56
             else if (strcmp(cmdBuffer, "setTime") == 0)
57
58
                 setTime();
59
             else if (strcmp(cmdBuffer, "createPCB") == 0)
63
                 char processName[20];
64
                 char processClass;
65
                 int processPriority;
66
                 {\tt char}\ {\tt nameMsg[]} = "Please enter a name for the PCB you wish to create. (The name can be no
67
        more than 20 characters) n";
68
                 int nameMsgLen = strlen(nameMsg);
                 sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
69
70
71
                 strcpy(processName, cmdBuffer);
```

```
73
                memset (cmdBuffer, ' \0', 100);
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);
                sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
77
                sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
78
79
                sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
80
                if (strcmp(cmdBuffer, "a") == 0)
81
                    processClass = 'a';
82
83
                else if (strcmp(cmdBuffer, "s") == 0)
84
85
86
                     processClass = 's';
87
88
                else
89
                {
90
                    processClass = '\0';
                memset (cmdBuffer, '\0', 100);
92
93
94
                \hbox{char priorityMsg[] = "Please enter a priority for the PCB you wish to create. (The priorities of the priorities of the priorities of the priorities of the priority for the PCB you wish to create.)}
       range from 0 to 9) n";
95
                int priorityMsqLen = strlen(priorityMsq);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
96
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
             else if (strcmp(cmdBuffer, "deletePCB") == 0)
103
104
105
                 char processName[20];
106
                 \verb|char| nameMsg[]| = \verb|mplease| enter the name for the PCB you wish to delete. (The name can be no
107
       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);
                 sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
111
112
                 strcpy(processName, cmdBuffer);
113
114
                 deletePCB (processName);
116
             else if (strcmp(cmdBuffer, "blockPCB") == 0)
117
                 char processName[20];
118
119
                 char nameMsg[] = "Please enter the name for the PCB you wish to block. (The name can be no
120
       more than 20 characters) \n";
121
                 int nameMsgLen = strlen(nameMsg);
                 sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
122
123
                 sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
                 sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
124
125
                 strcpy(processName, cmdBuffer);
126
127
                 blockPCB(processName);
128
             else if (strcmp(cmdBuffer, "unblockPCB") == 0)
129
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";
                 int nameMsgLen = strlen(nameMsg);
134
                 sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen); sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
135
136
137
                 sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
138
                 strcpy(processName, cmdBuffer);
139
140
                 unblockPCB (processName);
141
             else if (strcmp(cmdBuffer, "suspendPCB") == 0)
142
143
144
                 char processName[20];
145
146
                 \verb|char| nameMsg[]| = \verb|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);
                 sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
149
150
                 sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
151
                 strcpy(processName, cmdBuffer);
152
                 suspendPCB(processName);
153
```

```
154
155
                        else if (strcmp(cmdBuffer, "resumePCB") == 0)
156
157
                                char processName[20];
158
                                char nameMsq[] = "Please enter the name for the PCB you wish to resume. (The name can be no
159
              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);
                                sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
163
164
                                strcpy(processName, cmdBuffer);
165
166
                                resumePCB (processName);
167
168
                        else if (strcmp(cmdBuffer, "setPCBPriority") == 0)
169
170
                                char processName[20];
171
                                int newProcessPriority;
                                {\tt char} \ {\tt nameMsg[]} = {\tt "Please} \ {\tt enter} \ {\tt the} \ {\tt name} \ {\tt for} \ {\tt the} \ {\tt PCB} \ {\tt you} \ {\tt wish} \ {\tt to} \ {\tt change} \ {\tt priorities} \ {\tt for.} \ ({\tt The} \ {\tt change} \ {\tt priorities} \ {\tt for.} \ ({\tt The} \ {\tt change} \ {\tt priorities} \ {\tt for.} \ ({\tt The} \ {\tt change} \ {\tt priorities} \ {\tt for.} \ ({\tt The} \ {\tt change} \ {\tt priorities} \ {\tt for.} \ ({\tt The} \ {\tt change} \ {\tt priorities} \ {\tt for.} \ ({\tt The} \ {\tt change} \ {\tt priorities} \ {\tt for.} \ ({\tt The} \ {\tt change} \ {\tt priorities} \ {\tt change} \ {\tt chang
173
              name can be no more than 20 characters) \n";
174
                                int nameMsgLen = strlen(nameMsg);
sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
175
                                sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
176
177
                                strcpy(processName, cmdBuffer);
178
179
180
                                \verb|char| priorityMsg[] = \verb|mPlease| enter a priority for the PCB you wish to change priorities for.
              (The priorities range from 0 to 9)\n";
int priorityMsgLen = strlen(priorityMsg);
181
182
                                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                                sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
183
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
                                char nameMsg[] = "Please enter the name for the PCB you wish to see. (The name can be no
193
              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
                                showReadv();
205
                        else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
206
207
                        {
208
                                showSuspendedReady();
209
                        else if (strcmp(cmdBuffer, "showSuspendedBlocked") == 0)
210
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
                                char message[] = "Unrecognized Command n";
230
231
232
                                int tempBuffer = strlen(message);
233
234
                                sys_req(WRITE, DEFAULT_DEVICE, (char *)message, &tempBuffer);
235
236
```

# 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
         char \ welcomeMSG[] = "\nWelcome to our CS 450 \ Project!\nType help to see what you can do!\n\n"; int welcomeLength = strlen(welcomeMSG); 
14
        sys_req(WRITE, DEFAULT_DEVICE, welcomeMSG, &welcomeLength);
1.5
16
        char cmdBuffer[100];
17
18
        int bufferSize;
19
        allocateQueues();
20
2.1
        int quitFlag = 0;
22
23
        while (!quitFlaq)
24
25
            //get a command: cal polling fx
26
            memset (cmdBuffer, ' \0', 100);
27
28
            bufferSize = 99; // reset size before each call to read
29
30
31
            sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
            char newLine[] = "\n";
int newLineCount = 1;
33
34
            sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
35
36
            if (strcmp(cmdBuffer, "help") == 0)
38
39
                 help();
40
            else if (strcmp(cmdBuffer, "version") == 0)
41
42
44
45
            else if (strcmp(cmdBuffer, "getDate") == 0)
46
47
                 getDate();
48
            else if (strcmp(cmdBuffer, "setDate") == 0)
52
            else if (strcmp(cmdBuffer, "getTime") == 0)
5.3
54
55
                 getTime();
```

```
else if (strcmp(cmdBuffer, "setTime") == 0)
59
                setTime();
60
            else if (strcmp(cmdBuffer, "createPCB") == 0)
61
62
63
                char processName[20];
64
                char processClass;
                int processPriority;
65
66
                \verb|char| nameMsg[] = \verb|memsg[]| = \verb|memsg[]| |
67
       more than 20 characters)\n";
                int nameMsgLen = strlen(nameMsg);
68
                sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
69
70
                sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
71
                sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
                strcpy(processName, cmdBuffer);
memset(cmdBuffer, '\0', 100);
72
73
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);
                sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
77
78
79
                if (strcmp(cmdBuffer, "a") == 0)
80
81
82
                    processClass = 'a';
83
                else if (strcmp(cmdBuffer, "s") == 0)
84
85
                    processClass = 's';
86
87
88
                else
89
                    processClass = '\0';
90
91
92
                memset (cmdBuffer, ' \0', 100);
                {\tt char \ priorityMsg[] = "Please \ enter \ a \ priority \ for \ the \ PCB \ you \ wish \ to \ create. \ (The \ priorities \ to \ create)}
94
       range from 0 to 9) \n";
95
                int priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
96
                sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
                sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
98
99
                processPriority = atoi(cmdBuffer);
100
101
                 createPCB(processName, processClass, processPriority);
102
             else if (strcmp(cmdBuffer, "deletePCB") == 0)
103
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);
sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
109
110
                 sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
111
                 sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
112
                 strcpy(processName, cmdBuffer);
113
                 deletePCB (processName);
114
115
             else if (strcmp(cmdBuffer, "blockPCB") == 0)
116
117
118
                 char processName[20];
119
                 char nameMsg[] = "Please enter the name for the PCB you wish to block. (The name can be no
120
       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);
                 sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
136
                 sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
137
```

```
138
                   strcpy(processName, cmdBuffer);
139
140
                   unblockPCB (processName);
141
              else if (strcmp(cmdBuffer, "suspendPCB") == 0)
142
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";
                   int nameMsgLen = strlen(nameMsg);
sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
147
148
149
150
151
                   strcpy(processName, cmdBuffer);
152
                   suspendPCB(processName);
153
154
155
              else if (strcmp(cmdBuffer, "resumePCB") == 0)
156
157
                   char processName[20];
158
                   \verb|char| nameMsg[] = "Please enter the name for the PCB you wish to resume."| (The name can be no
159
        more than 20 characters)\n";
    int nameMsgLen = strlen(nameMsg);
160
                   sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
161
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);
                   sys_req(WRITE, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
176
177
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";
  int priorityMsgLen = strlen(priorityMsg);
  sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
181
182
183
184
                   sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
185
                   newProcessPriority = atoi(cmdBuffer);
186
187
                   setPCBPriority(processName, newProcessPriority);
188
              else if (strcmp(cmdBuffer, "showPCB") == 0)
189
190
191
                   char processName[20];
192
193
                   \verb|char| nameMsg[]| = \verb|rP| lease enter the name for the PCB you wish to see. (The name can be no
        more than 20 characters)\n";
                   int nameMsgLen = strlen(nameMsg);
194
195
                   sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
                   sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
196
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
              else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
206
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
              else if (strcmp(cmdBuffer, "showAll") == 0)
218
219
```

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

603 }

## 5.22.1.2 getDate()

```
void getDate ( )
Definition at line 343 of file R1commands.c.
344 {
345
           char buffer [4] = " | 0 | 0 | 0 | 0 ";
346
347
           int count = 4;
           char divider = '/';
348
349
           char newLine[1] = "\n";
           int newLineCount = 1;
350
351
           outb(0x70, 0x07); // getting Day of month value
352
           BCDtoChar(inb(0x71), buffer);
353
354
           buffer[2] = divider;
           sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, ' \ 0', count);
355
356
357
358
           outb(0x70, 0x08); // getting Month value
359
           BCDtoChar(inb(0x71), buffer);
360
           buffer[2] = divider;
           sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, '\0', count);
361
362
363
           outb(0x70, 0x32); // getting Year value second byte BCDtoChar(inb(0x71), buffer);
364
365
366
           buffer[2] = ' \0';
           suner[2] - 10
sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, '\0', count);
367
368
369
           outb(0x70, 0x09); // getting Year value first byte BCDtoChar(inb(0x71), buffer);
370
371
           sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, '\0', count);
372
373
374
           \label{eq:sys_reg} \begin{split} & \text{sys\_reg(WRITE, DEFAULT\_DEVICE, newLine, &newLineCount);} \\ & \text{memset(newLine, '\0', newLineCount);} \end{split}
375
376
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";
         int count = 4;

char divider = ':';

char newLine[1] = "\n";

int newLineCount = 1;
194
195
196
197
198
          outb(0x70, 0x04); // getting Hour value
199
200
          BCDtoChar(inb(0x71), buffer);
201
          buffer[2] = divider;
          sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, ' \ 0', count);
202
203
204
205
          outb(0x70, 0x02); // getting Minute value
206
          BCDtoChar(inb(0x71), buffer);
207
          buffer[2] = divider;
          bullet[2] = divide;,
sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, ' \ 0', count);
208
209
210
211
          outb(0x70, 0x00); // getting Second value
212
          BCDtoChar(inb(0x71), buffer);
213
          buffer[2] = ' \setminus 0';
          sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, '\0', count);
214
215
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
        // Help Description section
14
       char helpDesc[] = "help: Returns basic command information.\n";
15
16
17
       int tempBuffer = strlen(helpDesc);
18
       sys reg(WRITE, DEFAULT DEVICE, (char *)helpDesc, &tempBuffer);
19
       memset(helpDesc, '\0', tempBuffer);
20
21
       // Version Description section
23
       char versionDesc[] = "version: Returns the current version of the software.\n";
24
25
       tempBuffer = strlen(versionDesc);
2.6
       sys_req(WRITE, DEFAULT_DEVICE, (char *)versionDesc, &tempBuffer);
27
28
       memset(versionDesc, '\0', tempBuffer);
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
       char setTimeDesc[] = "setTime: Allows the user to change the set time.\n";
39
40
41
       tempBuffer = strlen(setTimeDesc);
42
43
       sys_req(WRITE, DEFAULT_DEVICE, (char *)setTimeDesc, &tempBuffer);
44
       memset(setTimeDesc, '\0', tempBuffer);
4.5
46
       // getDate Description section
       char getDateDesc[] = "getDate: Returns the current set date.\n";
47
48
49
       tempBuffer = strlen(getDateDesc);
50
       sys_req(WRITE, DEFAULT_DEVICE, (char *)getDateDesc, &tempBuffer);
51
       memset(getDateDesc, '\0', tempBuffer);
52
53
        // setDate Description section
54
5.5
       char setDateDesc[] = "setDate: Allows the user to change the set date.\ln";
56
57
       tempBuffer = strlen(setDateDesc);
58
       sys_req(WRITE, DEFAULT_DEVICE, (char *)setDateDesc, &tempBuffer);
59
       memset(setDateDesc, '\0', tempBuffer);
60
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
       sys_req(WRITE, DEFAULT_DEVICE, (char *)createPCBDesc, &tempBuffer);
68
       memset(createPCBDesc, '\0', tempBuffer);
69
       // deletePCB Description section
70
       char deletePCBDesc[] = "deletePCB: Will delete a specific PCB from what ever queue it is in. \n";
71
72
73
       tempBuffer = strlen(deletePCBDesc);
74
       sys_req(WRITE, DEFAULT_DEVICE, (char *)deletePCBDesc, &tempBuffer);
memset(deletePCBDesc, '\0', tempBuffer);
7.5
76
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
       sys_req(WRITE, DEFAULT_DEVICE, (char *)blockPCBDesc, &tempBuffer);
83
       memset(blockPCBDesc, '\0', tempBuffer);
84
85
        // unblockPCB Description section
86
87
       {\tt char unblockPCBDesc[] = "unblockPCB: Will change a specific PCB's state to ready. \\ \\ {\tt ln";}
88
89
       tempBuffer = strlen(unblockPCBDesc);
90
       sys_req(WRITE, DEFAULT_DEVICE, (char *)unblockPCBDesc, &tempBuffer);
```

```
92
        memset(unblockPCBDesc, '\0', tempBuffer);
93
94
        // suspendPCB Description section
        char suspendPCBDesc[] = "suspendPCB: Will suspend a specific PCB. \n";
9.5
96
        tempBuffer = strlen(suspendPCBDesc);
98
99
        sys_req(WRITE, DEFAULT_DEVICE, (char *)suspendPCBDesc, &tempBuffer);
100
         memset(suspendPCBDesc, '\0', tempBuffer);
101
         // resumePCB Description section char resumePCBDesc[] = "resumePCB: Will unsuspend a specific PCB. \n";
102
103
104
105
         tempBuffer = strlen(resumePCBDesc);
106
         sys_req(WRITE, DEFAULT_DEVICE, (char *)resumePCBDesc, &tempBuffer);
memset(resumePCBDesc, '\0', tempBuffer);
107
108
109
110
         // setPCBPriority Description section
111
         char setPCBPriorityDesc[] = "setPCBPriority: Will change the priority of a specific PCB. \n";
112
113
         tempBuffer = strlen(setPCBPriorityDesc);
114
         sys_req(WRITE, DEFAULT_DEVICE, (char *)setPCBPriorityDesc, &tempBuffer);
memset(setPCBPriorityDesc, '\0', tempBuffer);
115
116
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
         // showReady Description section
char showReadyDesc[] = "showReady: Will display the name, class, state, suspended status, and
126
127
        priority of every PCB in the ready queue. \n";
128
129
         tempBuffer = strlen(showReadyDesc);
130
         sys_req(WRITE, DEFAULT_DEVICE, (char *)showReadyDesc, &tempBuffer);
memset(showReadyDesc, '\0', tempBuffer);
131
132
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(showSuspendedReadvDesc);
138
         sys_req(WRITE, DEFAULT_DEVICE, (char *) showSuspendedReadyDesc, &tempBuffer);
memset(showSuspendedReadyDesc, '\0', tempBuffer);
139
140
141
         // showSuspendedBlocked Description section
char showSuspendedBlockedDesc[] = "showSuspendedBlocked: Will display the name, class, state,
142
143
        suspended status, and priority of every PCB in the suspended blocked queue.\ensuremath{\mid} n";
144
145
         tempBuffer = strlen(showSuspendedBlockedDesc);
146
147
         sys_req(WRITE, DEFAULT_DEVICE, (char *)showSuspendedBlockedDesc, &tempBuffer);
         memset (showSuspendedBlockedDesc, '\0', tempBuffer);
148
149
150
         // showBlocked Description section
         char showBlockedDesc[] = "showBlocked: Will display the name, class, state, suspended status, and
151
        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
         // \ {\tt showAll \ Description \ section}
        char showAllDesc[] = "showReady: Will display the name, class, state, suspended status, and priority of every PCB in all 4 queues.|n|;
159
160
161
         tempBuffer = strlen(showAllDesc);
162
         sys_req(WRITE, DEFAULT_DEVICE, (char *)showAllDesc, &tempBuffer);
memset(showAllDesc, '\0', tempBuffer);
163
164
165
166
         // quit Description section
         char quitDesc[] = "quit: Allows the user to shut the system down.\n";
167
168
169
         tempBuffer = strlen(quitDesc);
170
         sys_req(WRITE, DEFAULT_DEVICE, (char *)quitDesc, &tempBuffer);
171
         memset(quitDesc, '\0', tempBuffer);
172
```

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

# 5.22.1.5 intToBCD()

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";
        int quitMsgLength = strlen(quitMsg);
610
611
        sys_req(WRITE, DEFAULT_DEVICE, quitMsg, &quitMsgLength);
612
        char quitAns[] = "\0\0";
613
        int quitAnsLength = 1;
sys_req(READ, DEFAULT_DEVICE, quitAns, &quitAnsLength);
char answer = quitAns[0];
614
615
616
617
618
        if (answer == 'y' || answer == 'Y')
619
62.0
             flag = 1;
621
        else if (answer == 'n' || answer == 'N')
622
623
        {
624
             flag = 0;
625
626
        else
627
             char error[] = "Invalid input!\n";
628
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
392
393
394
         char year[5] = "\0\0\0\0\0"; // year buffer
395
        int flag = 0; // thrown if input is invalid
396
397
398
        do
399
         {
400
             sys_req(READ, DEFAULT_DEVICE, year, &count);
401
             if (atoi(year) > 0)
402
403
                  sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
404
                  flag = 0;
405
406
                  char yearUpper[3] = "\0\0\0";
char yearLower[3] = "\0\0\0";
407
408
409
                 yearUpper[0] = year[0];
yearUpper[1] = year[1];
yearLower[0] = year[2];
410
411
412
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
                  char invalid[] = "Invalid year.\n";
427
                  int lengthInval = strlen(invalid);
428
                  sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
429
430
                  memset(invalid, '\0', lengthInval);
431
432
                  flag = 1;
433
        } while (flag == 1);
434
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
        char month[4] = "\0\0\n\0,
443
444
        count = 4; // used to print month
445
446
447
             sys_req(READ, DEFAULT_DEVICE, month, &count);
if (atoi(month) < 13 && atoi(month) > 0)
448
449
450
451
452
                  sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
453
                 flag = 0;
454
                  cli();
455
456
                  outb(0x70, 0x08); // Setting month value
457
458
                  outb(0x71, intToBCD(atoi(month)));
459
460
                  sti();
461
             else
462
463
464
                  char invalid[] = "Invalid month.\n";
465
                  int lengthInval = strlen(invalid);
                  sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
466
467
                  memset(invalid, '\0', lengthInval);
468
                  flag = 1;
469
470
471
         } while (flag == 1);
472
        char instruction3[] = "Please type the desired day of month. I.E.: dd. \ln";
474
475
```

```
476
        length = strlen(instruction3);
477
        sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
478
        memset(instruction3, '\0', length);
479
        char day[4] = "\0\0\n\0";
480
        count = 4; // used to print day
481
482
483
484
             sys_req(READ, DEFAULT_DEVICE, day, &count);
485
             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
if ((atoi(year) % 4 == 0 && atoi(year) % 100 != 0) || atoi(year) % 400 == 0)
486
487
488
             { // checking for leap year
489
490
                 char leapYear[] = "This is a leap year. February has 29 days.\n";
491
                 length = strlen(leapYear);
492
                 sys_req(WRITE, DEFAULT_DEVICE, leapYear, &length);
493
                 memset(leapYear, '\0', length);
494
495
                 if ((atoi(month) == 1 || atoi(month) == 3 || atoi(month) == 5 || atoi(month) == 7 ||
496
       atoi(month) == 8 | atoi(month) == 10 | atoi(month) == 12) && atoi(day) > 31)
497
                 {
498
                     flag = 1;
                      char invalid[] = "Invalid day.\n";
499
                      length = strlen(invalid);
500
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);
                      sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
509
                     memset (invalid, '\0', length);
510
511
512
                 else if ((atoi(month) == 2) && atoi(day) > 29)
513
514
                     flag = 1;
                      char invalid[] = "Invalid day.\n";
515
                     length = strlen(invalid);
sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
516
517
                     memset(invalid, '\0', length);
518
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();
                 }
531
532
             else if (atoi(year) % 4 != 0 || atoi(year) % 400 != 0)
533
             \{\ //\ {\it checking for leap year}
534
                 char noLeap[] = "This is not a leap year.\n";
535
536
                 length = strlen(noLeap);
                 sys_req(WRITE, DEFAULT_DEVICE, noLeap, &length);
537
538
                 memset(noLeap, '\0', length);
539
                 if ((atoi(month) == 1 \mid atoi(month) == 3 \mid atoi(month) == 5 \mid atoi(month) == 7 \mid
540
       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);
                     sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
memset(invalid, '\0', length);
545
546
547
                 else if ((atoi(month) == 4 | atoi(month) == 6 | atoi(month) == 9 | atoi(month) == 11) &&
       atoi(day) > 30)
549
550
                     flag = 1;
                      char invalid[] = "Invalid day.\n";
551
                     length = strlen(invalid);
sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
552
553
                     memset(invalid, '\0', length);
554
555
556
                 else if ((atoi(month) == 2) && atoi(day) > 28)
557
                     flaq = 1;
558
```

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

#### 5.22.1.8 setTime()

```
int setTime ( )
```

#### Definition at line 221 of file R1commands.c.

```
222 {
224
         int count = 4; // counter for printing
225
         char spacer[1] = "\n"; // used to space out terminal outputs
226
         int spaceCount = 1;
227
228
230
         char instruction1[] = "Please type the desired hours. I.E.: hh.\n";
231
232
         int length = strlen(instruction1);
233
234
         sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
235
         memset(instruction1, '\0', length);
236
237
         char hour[4] = "\0\0\n\0";
238
239
         int flag = 0;
240
241
242
243
              sys_req(READ, DEFAULT_DEVICE, hour, &count);
244
              if (atoi(hour) < 24 && atoi(hour) >= 0)
245
246
                  sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
247
248
                  flag = 0;
249
250
              else
2.51
                  char invalid[] = "Invalid hours.\n";
int lengthInval = strlen(invalid);
sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
252
253
254
255
256
                  memset(invalid, '\0', lengthInval);
257
                  flag = 1;
258
259
         } while (flag == 1);
260
262
         char instruction2[] = "Please type the desired minutes. I.E.: mm.\n";
263
264
         length = strlen(instruction2);
265
         sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
266
267
         memset(instruction2, '\0', length);
268
269
         char minute[4] = "\0\0\n\0";
```

```
270
271
272
              sys_req(READ, DEFAULT_DEVICE, minute, &count);
273
2.74
              if (atoi(minute) < 60 && atoi(minute) >= 0)
275
276
277
                  sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
278
                  flag = 0;
279
280
             else
281
                  char invalid[] = "Invalid minutes.\n";
282
                  int lengthInval = strlen(invalid);
283
284
                  sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
285
                  sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
                  memset(invalid, '\0', lengthInval);
286
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
         sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
296
297
         memset(instruction3, '\0', length);
298
         char second[4] = "\0\0\n\0";
299
300
301
         do
302
         {
303
              sys_req(READ, DEFAULT_DEVICE, second, &count);
304
              if (atoi(second) < 60 && atoi(second) >= 0)
305
306
307
                  sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
                  flag = 0;
308
309
310
              else
311
                  char invalid[] = "Invalid seconds.\n";
312
                  int lengthInval = strlen(invalid);
sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
313
314
315
316
                  memset(invalid, '\0', lengthInval);
317
                  flag = 1;
318
         } while (flag == 1);
319
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);
         sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
memset(exitMessage, '\0', exitLength);
memset(spacer, '\0', spaceCount);
336
337
338
339
340
         return 0;
341 }
```

#### 5.22.1.9 version()

```
int version ( )
```

Definition at line 177 of file R1commands.c.

```
178 {
179
```

# 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()

#### 5.23.1.3 getDate()

```
void getDate ( )
Definition at line 343 of file R1commands.c.
344 {
345
           char buffer [4] = " | 0 | 0 | 0 | 0 ";
346
347
           int count = 4;
           char divider = '/';
348
349
           char newLine[1] = "\n";
350
           int newLineCount = 1;
351
           outb(0x70, 0x07); // getting Day of month value
352
           BCDtoChar(inb(0x71), buffer);
353
354
           buffer[2] = divider;
           sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, ' \ 0', count);
355
356
357
           outb(0x70, 0x08); // getting Month value
358
359
           BCDtoChar(inb(0x71), buffer);
360
           buffer[2] = divider;
           sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, '\0', count);
361
362
363
           outb(0x70, 0x32); // getting Year value second byte BCDtoChar(inb(0x71), buffer);
364
365
366
           buffer[2] = ' \0';
           suner[2] - 10
sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, '\0', count);
367
368
369
           outb(0x70, 0x09); // getting Year value first byte BCDtoChar(inb(0x71), buffer);
370
371
           sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, '\0', count);
372
373
374
           \label{eq:sys_reg} \begin{split} & \text{sys\_reg} \, (\text{WRITE, DEFAULT\_DEVICE, newLine, \&newLineCount}) \, ; \\ & \text{memset} \, (\text{newLine, '\0', newLineCount}) \, ; \end{split}
375
376
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";
          int count = 4;

char divider = ':';

char newLine[1] = "\n";

int newLineCount = 1;
194
195
196
197
198
199
          outb(0x70, 0x04); // getting Hour value
200
           BCDtoChar(inb(0x71), buffer);
201
          buffer[2] = divider;
          \label{eq:sys_req} \begin{tabular}{ll} sys\_req (WRITE, DEFAULT\_DEVICE, buffer, \&count); \\ memset (buffer, ' \ 0', count); \\ \end{tabular}
202
203
204
205
          outb(0x70, 0x02); // getting Minute value
206
           BCDtoChar(inb(0x71), buffer);
207
          buffer[2] = divider;
          sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, ' \0', count);
208
209
210
211
          outb(0x70, 0x00); // getting Second value
          BCDtoChar(inb(0x71), buffer);
212
213
          buffer[2] = ' \setminus 0';
          sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
memset(buffer, '\0', count);
214
215
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 // Help Description section 14 char helpDesc[] = "help: Returns basic command information.\n"; 15 16 17 int tempBuffer = strlen(helpDesc); 18 sys reg(WRITE, DEFAULT DEVICE, (char \*)helpDesc, &tempBuffer); 19 memset(helpDesc, '\0', tempBuffer); 20 21 // Version Description section 23 char versionDesc[] = "version: Returns the current version of the software.\n"; 24 25 tempBuffer = strlen(versionDesc); 2.6 27 sys\_req(WRITE, DEFAULT\_DEVICE, (char \*)versionDesc, &tempBuffer); 28 memset(versionDesc, '\0', tempBuffer); 30 // getTime Description section 31 char getTimeDesc[] = "getTime: Returns the current set time.\n"; 32 tempBuffer = strlen(getTimeDesc); 33 34 35 sys\_req(WRITE, DEFAULT\_DEVICE, (char \*)getTimeDesc, &tempBuffer); 36 memset(getTimeDesc, '\0', tempBuffer); 37 38 // setTime Description section char setTimeDesc[] = "setTime: Allows the user to change the set time.\n"; 39 40 41 tempBuffer = strlen(setTimeDesc); 42 43 sys\_req(WRITE, DEFAULT\_DEVICE, (char \*)setTimeDesc, &tempBuffer); 44 memset(setTimeDesc, '\0', tempBuffer); 4.5 46 // getDate Description section char getDateDesc[] = "getDate: Returns the current set date.\n"; 47 48 49 tempBuffer = strlen(getDateDesc); 50 sys\_req(WRITE, DEFAULT\_DEVICE, (char \*)getDateDesc, &tempBuffer); 51 memset(getDateDesc, '\0', tempBuffer); 52 53 // setDate Description section 54 5.5 char setDateDesc[] = "setDate: Allows the user to change the set date. $\ln$ "; 56 57 tempBuffer = strlen(setDateDesc); 58 sys\_req(WRITE, DEFAULT\_DEVICE, (char \*)setDateDesc, &tempBuffer); 59 memset(setDateDesc, '\0', tempBuffer); 60 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 sys\_req(WRITE, DEFAULT\_DEVICE, (char \*)createPCBDesc, &tempBuffer); 68 memset(createPCBDesc, '\0', tempBuffer); 69 // deletePCB Description section 70 char deletePCBDesc[] = "deletePCB: Will delete a specific PCB from what ever queue it is in. \n"; 71 72 73 tempBuffer = strlen(deletePCBDesc); 74 sys\_req(WRITE, DEFAULT\_DEVICE, (char \*)deletePCBDesc, &tempBuffer);
memset(deletePCBDesc, '\0', tempBuffer); 7.5 76 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 sys\_req(WRITE, DEFAULT\_DEVICE, (char \*)blockPCBDesc, &tempBuffer); 83 memset(blockPCBDesc, '\0', tempBuffer); 84 85 // unblockPCB Description section 86 87  ${\tt char unblockPCBDesc[] = "unblockPCB: Will change a specific PCB's state to ready. \\ \\ {\tt ln";}$ 88 tempBuffer = strlen(unblockPCBDesc); 89 90 91 sys\_req(WRITE, DEFAULT\_DEVICE, (char \*)unblockPCBDesc, &tempBuffer);

```
92
        memset(unblockPCBDesc, '\0', tempBuffer);
93
94
        // suspendPCB Description section
        char suspendPCBDesc[] = "suspendPCB: Will suspend a specific PCB. \n";
9.5
96
97
        tempBuffer = strlen(suspendPCBDesc);
98
99
        sys_req(WRITE, DEFAULT_DEVICE, (char *)suspendPCBDesc, &tempBuffer);
100
         memset(suspendPCBDesc, '\0', tempBuffer);
101
         // resumePCB Description section char resumePCBDesc[] = "resumePCB: Will unsuspend a specific PCB. \n";
102
103
104
105
         tempBuffer = strlen(resumePCBDesc);
106
         sys_req(WRITE, DEFAULT_DEVICE, (char *)resumePCBDesc, &tempBuffer);
memset(resumePCBDesc, '\0', tempBuffer);
107
108
109
110
         // setPCBPriority Description section
111
         char setPCBPriorityDesc[] = "setPCBPriority: Will change the priority of a specific PCB. \n";
112
113
         tempBuffer = strlen(setPCBPriorityDesc);
114
         sys_req(WRITE, DEFAULT_DEVICE, (char *)setPCBPriorityDesc, &tempBuffer);
memset(setPCBPriorityDesc, '\0', tempBuffer);
115
116
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
         // showReady Description section
char showReadyDesc[] = "showReady: Will display the name, class, state, suspended status, and
126
127
        priority of every PCB in the ready queue. \n";
128
129
         tempBuffer = strlen(showReadyDesc);
130
         sys_req(WRITE, DEFAULT_DEVICE, (char *)showReadyDesc, &tempBuffer);
memset(showReadyDesc, '\0', tempBuffer);
131
132
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(showSuspendedReadvDesc);
138
         sys_req(WRITE, DEFAULT_DEVICE, (char *) showSuspendedReadyDesc, &tempBuffer);
memset(showSuspendedReadyDesc, '\0', tempBuffer);
139
140
141
         // showSuspendedBlocked Description section
char showSuspendedBlockedDesc[] = "showSuspendedBlocked: Will display the name, class, state,
142
143
        suspended status, and priority of every PCB in the suspended blocked queue.\ensuremath{\mid} n";
144
145
         tempBuffer = strlen(showSuspendedBlockedDesc);
146
147
         sys_req(WRITE, DEFAULT_DEVICE, (char *)showSuspendedBlockedDesc, &tempBuffer);
         memset(showSuspendedBlockedDesc, '\0', tempBuffer);
148
149
150
         // showBlocked Description section
         char showBlockedDesc[] = "showBlocked: Will display the name, class, state, suspended status, and
151
        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
         // \ {\tt showAll \ Description \ section}
        char showAllDesc[] = "showReady: Will display the name, class, state, suspended status, and priority of every PCB in all 4 queues.|n|;
159
160
161
         tempBuffer = strlen(showAllDesc);
162
         sys_req(WRITE, DEFAULT_DEVICE, (char *)showAllDesc, &tempBuffer);
memset(showAllDesc, '\0', tempBuffer);
163
164
165
         // quit Description section
166
         char quitDesc[] = "quit: Allows the user to shut the system down.\n";
167
168
169
         tempBuffer = strlen(quitDesc);
170
         sys_req(WRITE, DEFAULT_DEVICE, (char *)quitDesc, &tempBuffer);
171
         memset(quitDesc, '\0', tempBuffer);
172
```

```
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);
        sys_req(WRITE, DEFAULT_DEVICE, quitMsg, &quitMsgLength);
611
612
613
        char quitAns[] = " \setminus 0 \setminus 0";
        int quitAnsLength = 1;
614
        sys_req(READ, DEFAULT_DEVICE, quitAns, &quitAnsLength);
char answer = quitAns[0];
615
616
617
         if (answer == 'y' || answer == 'Y')
618
619
620
             flaq = 1:
621
622
        else if (answer == 'n' || answer == 'N')
623
624
             flag = 0;
625
626
        else
627
        {
             char error[] = "Invalid input!\n";
628
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.

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

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

```
if ((atoi(month) == 1 || atoi(month) == 3 || atoi(month) == 5 || atoi(month) == 7 ||
496
        atoi(month) == 8 || atoi(month) == 10 || atoi(month) == 12) && atoi(day) > 31)
497
                 {
498
                      flag = 1;
                      char invalid[] = "Invalid day.\n";
499
                      length = strlen(invalid);
500
                      sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
501
502
                      memset(invalid, '\0', length);
503
                  else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
504
       atoi(day) > 30)
505
                 {
506
                      flag = 1;
507
                      char invalid[] = "Invalid day.\n";
508
                      length = strlen(invalid);
                      sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
memset(invalid, '\0', length);
509
510
511
512
                  else if ((atoi(month) == 2) && atoi(day) > 29)
513
                  {
514
                      flag = 1;
                      char invalid[] = "Invalid day.\n";
515
                      length = strlen(invalid);
sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
memset(invalid, '\0', length);
516
517
518
519
520
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
                  char noLeap[] = "This is not a leap year.\n";
535
                  length = strlen(noLeap);
sys_req(WRITE, DEFAULT_DEVICE, noLeap, &length);
memset(noLeap, '\0', length);
536
537
538
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);
                      memset(invalid, '\0', length);
546
547
                  else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
548
       atoi(day) > 30)
549
                 {
550
                      flag = 1;
                      char invalid[] = "Invalid day.\n";
551
                      length = strlen(invalid);
552
                      rength = strenthwaltd,,
sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
memset(invalid, '\0', length);
553
554
556
                  else if ((atoi(month) == 2) && atoi(day) > 28)
557
558
                      flag = 1;
                      char invalid[] = "Invalid day.\n";
559
                      length = strlen(invalid);
560
                      sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
561
                      memset(invalid, '\0', length);
562
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";
```

```
int exitLength = strlen(exitMessage);
sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
memset(exitMessage, '\0', exitLength);
memset(spacer, '\0', spaceCount);
sas
return 0;
sss }
```

#### 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
        char spacer[1] = "\n"; // used to space out terminal outputs int spaceCount = 1;
226
227
228
        char instruction1[] = "Please type the desired hours. I.E.: hh.\n";
230
231
232
        int length = strlen(instruction1);
233
        234
235
236
237
        char hour[4] = "\ 0\ 0\ n\ 0";
238
239
        int flag = 0;
240
241
242
243
            sys_req(READ, DEFAULT_DEVICE, hour, &count);
244
            if (atoi(hour) < 24 && atoi(hour) >= 0)
245
246
                sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
2.47
248
                flag = 0;
249
            }
250
            else
251
252
                char invalid[] = "Invalid hours.\n";
                int lengthInval = strlen(invalid);
253
                sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
254
255
                memset(invalid, '\0', lengthInval);
256
                flag = 1;
257
258
        } while (flag == 1);
259
260
        char instruction2[] = "Please type the desired minutes. I.E.: mm.\n";
262
263
264
        length = strlen(instruction2);
265
        266
2.67
268
269
        char minute[4] = "\0\0\n\0;
270
271
2.72
            sys_req(READ, DEFAULT_DEVICE, minute, &count);
if (atoi(minute) < 60 && atoi(minute) >= 0)
273
274
275
276
277
                 sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
278
                flag = 0;
279
280
            else
281
282
                char invalid[] = "Invalid minutes.\n";
                int lengthInval = strlen(invalid);
283
                 sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
284
285
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
                memset(invalid, '\0', lengthInval);
286
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
302
              sys_req(READ, DEFAULT_DEVICE, second, &count);
303
304
              if (atoi(second) < 60 && atoi(second) >= 0)
305
306
                   sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
307
308
                  flag = 0;
309
              }
310
              else
311
312
                   char invalid[] = "Invalid seconds.\n";
                  int lengthInval = strlen(invalid);
sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
313
314
315
                   memset(invalid, '\0', lengthInval);
316
317
318
         } while (flag == 1);
319
320
321
         cli();
322
323
         outb(0x70, 0x04); // Hour
324
         outb(0x71, intToBCD(atoi(hour)));
325
         outb(0x70, 0x02); // Minute
326
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);
         sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
memset(exitMessage, '\0', exitLength);
memset(spacer, '\0', spaceCount);
336
337
338
339
340
         return 0;
341 }
```

## 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
        sys_req(WRITE, DEFAULT_DEVICE, (char *)version, &tempBuffer);
184
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 {
      //COLTON WILL PROGRAM THIS FUNCTION
18
      stack, and perform any reasonable initialization.
19
      PCB *newPCB = (PCB *)sys_alloc_mem(sizeof(PCB));
20
      char name[20] = "newPCB";
21
22
      strcpy(newPCB->processName, name);
23
      newPCB->suspendedStatus = 1;
newPCB->runningStatus = -1;
24
25
      newPCB->stackTop = (newPCB->stackTop + 1024);
newPCB->stackBase = newPCB->stackBase;
26
27
28
      newPCB->priority = 0;
29
      // Setting the PCBs prev and next PCB
30
      newPCB->nextPCB = NULL:
31
      newPCB->prevPCB = NULL;
32
33
      newPCB->processClass = NULL;
35
36
      return newPCB;
37 }
```

## 5.24.1.2 allocateQueues()

```
void allocateQueues ( )
```

```
Definition at line 377 of file R2_Internal_Functions_And_Structures.c.
```

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

# 5.24.1.3 findPCB()

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

```
79
       // ANASTASE WILL PROGRAM THIS FUNCTION
80
       // {\it findPCB} \, () \  \, {\it will search all queues for a process with a given name.} \, \,
81
82
83
       if (strlen(processName) > 20)
85
            char error_message[30] = "Invalid process name.\n";
86
87
            int error_size = strlen(error_message);
            sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
88
            return NULL;
89
           //return cz we have to stop if the process name is too long
90
92
93
94
            PCB *tempPCB = ready->head;
95
           int value = 0;
96
            while (value <= ready->count)
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
118
                     tempPCB = tempPCB->nextPCB;
119
120
                     value++;
121
122
             }
```

```
123
124
            tempPCB = suspendedBlocked->head;
125
            value = 0;
            while (value <= suspendedBlocked->count)
126
127
128
                if (strcmp(tempPCB->processName, processName) == 0)
129
130
                    return tempPCB;
131
132
133
               {
134
                    tempPCB = tempPCB->nextPCB;
135
                    value++;
136
137
           }
138
           tempPCB = suspendedReady->head;
139
            value = 0;
140
           while (value <= suspendedReady->count)
141
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()

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

# 5.24.1.5 getBlocked()

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()

#### 5.24.1.8 getSuspendedReady()

# 5.24.1.9 insertPCB()

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

```
160
        //BENJAMIN WILL PROGRAM THIS FUNCTION
161
        //insertPCB() will insert a PCB into the appropriate queue.
162
        //Note: The ready queue is a priority queue and the blocked queue is a FIFO queue.
163
164
165
        if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 1)
166
        { // Insert into ready queue
            PCB *tempPtr = ready->head;
167
168
            if (tempPtr != NULL)
169
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
                        PCB_to_insert->nextPCB = tempPtr;
180
                        PCB_to_insert->prevPCB = tempPtr->prevPCB;
181
182
                        tempPtr->prevPCB = PCB_to_insert;
183
```

```
184
                     else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
185
186
                         tempPtr->nextPCB = PCB_to_insert;
                         PCB_to_insert->prevPCB = tempPtr;
ready->tail = PCB_to_insert;
187
188
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
        else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
201
        { // Insert into suspended ready queue
202
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
2.31
            {
232
                suspendedReadv->count++;
                suspendedReady->head = PCB_to_insert;
233
                suspendedReady->tail = PCB_to_insert;
234
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;
2.44
245
        else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
246
        { // Insert into suspended blocked queue
            PCB *tempPtr = suspendedBlocked->tail;
247
248
249
            tempPtr->nextPCB = PCB_to_insert;
250
            PCB_to_insert->prevPCB = tempPtr;
            suspendedBlocked->tail = PCB_to_insert;
251
252
        }
253 }
```

## 5.24.1.10 removePCB()

Definition at line 255 of file R2\_Internal\_Functions\_And\_Structures.c.  $_{256\ \ \{}$ 

```
//BENJAMIN WILL PROGRAM THIS FUNCTION
257
258
259
        //removePCB() will remove a PCB from the queue in which it is currently stored.
260
        PCB *removedPCB = findPCB(PCB_to_remove->processName);
2.61
        if (removedPCB == NULL)
262
263
        {
264
265
        else if (removedPCB == ready->head)
266
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;
            removedNext->prevPCB = NULL;
280
281
            removedPCB->nextPCB = NULL;
            blocked->count--;
282
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
            suspendedBlocked->head = removedNext:
299
300
            removedNext->prevPCB = NULL;
            removedPCB->nextPCB = NULL;
301
302
            suspendedBlocked->count--;
303
            return 0;
304
        else if (removedPCB == readv->tail)
305
306
307
            PCB *removedPrev = removedPCB->prevPCB;
308
309
            ready->tail = removedPrev;
310
            removedPrev->nextPCB = NULL:
            removedPCB->prevPCB = NULL;
311
312
            ready->count--;
313
            return 0;
314
315
        else if (removedPCB == blocked->tail)
316
317
            PCB *removedPrev = removedPCB->prevPCB;
318
319
            blocked->tail = removedPrev;
            removedPrev->nextPCB = NULL;
320
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;
            removedPrev->nextPCB = NULL;
330
            removedPCB->prevPCB = NULL;
331
332
            suspendedReady->count--;
333
            return 0;
334
335
        else if (removedPCB == suspendedBlocked->tail)
336
337
            PCB *removedPrev = removedPCB->prevPCB;
338
            suspendedBlocked->tail = removedPrev;
339
340
            removedPrev->nextPCB = NULL;
341
            removedPCB->prevPCB = NULL;
342
            suspendedBlocked->count--;
343
            return 0:
```

```
344
        }
345
        else
346
             PCB *tempPrev = removedPCB->prevPCB;
PCB *tempNext = removedPCB->nextPCB;
347
348
349
             tempPrev->nextPCB = tempNext;
350
351
             tempNext->prevPCB = tempPrev;
352
             removedPCB->nextPCB = NULL;
removedPCB->prevPCB = NULL;
353
354
355
356
             if (removedPCB->runningStatus == 0 && removedPCB->suspendedStatus == 1)
357
358
                  ready->count--;
359
             else if (removedPCB->runningStatus == -1 && removedPCB->suspendedStatus == 1)
360
361
362
                  blocked->count--;
363
364
             else if (removedPCB->runningStatus == 0 && removedPCB->suspendedStatus == 0)
365
366
                  suspendedReady->count--;
367
             else if (removedPCB->runningStatus == -1 && removedPCB->suspendedStatus == 0)
368
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
52
        //setupPcb() will call allocatePCB() to create an empty PCB, initializes the PCB information, sets
       the PCB state to ready, not suspended.
53
       PCB *returnedPCB = allocatePCB();
54
55
56
       if (findPCB(processName) ->processName == processName)
57
           char message[] = "There is already a PCB with this name.\n"; int messLength = strlen(message);
58
59
60
           sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
61
62
           returnedPCB = NULL;
63
64
       else
65
66
           strcpy(returnedPCB->processName, processName);
67
           returnedPCB->processClass = processClass;
68
            returnedPCB->priority = processPriority;
70
            returnedPCB->runningStatus = 0;
71
            returnedPCB->suspendedStatus = 1;
72
       }
73
       return returnedPCB;
74
```

#### 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 ()
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.

```
//COLTON WILL PROGRAM THIS FUNCTION
17
18
        // \verb|allocatePCB()| will use sys_alloc_mem()| to allocate memory for a new PCB, possible including the
        stack, and perform any reasonable initialization.
PCB *newPCB = (PCB *)sys_alloc_mem(sizeof(PCB));
19
21
        char name[20] = "newPCB";
22
        strcpy(newPCB->processName, name);
2.3
        newPCB->suspendedStatus = 1;
24
        newPCB->runningStatus = -1;
25
        newPCB->stackTop = (newPCB->stackTop + 1024);
newPCB->stackBase = newPCB->stackBase;
26
27
        newPCB->priority = 0;
28
29
30
        // Setting the PCBs prev and next PCB
        newPCB->nextPCB = NULL;
newPCB->prevPCB = NULL;
31
        newPCB->processClass = NULL;
35
        return newPCB;
36
```

## 5.25.2.2 allocateQueues()

```
void allocateQueues ( )
```

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

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

# 5.25.2.3 findPCB()

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

```
79
       // ANASTASE WILL PROGRAM THIS FUNCTION
80
       // {\it findPCB} \, () \  \, {\it will search all queues for a process with a given name.} \, \,
81
82
83
       if (strlen(processName) > 20)
84
85
            char error_message[30] = "Invalid process name.\n";
86
87
            int error_size = strlen(error_message);
            sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
88
            return NULL;
89
           //return cz we have to stop if the process name is too long
90
92
93
94
            PCB *tempPCB = ready->head;
95
           int value = 0;
96
            while (value <= ready->count)
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
118
                     tempPCB = tempPCB->nextPCB;
119
120
                     value++;
121
122
             }
```

```
123
124
            tempPCB = suspendedBlocked->head;
125
            value = 0;
            while (value <= suspendedBlocked->count)
126
127
128
                if (strcmp(tempPCB->processName, processName) == 0)
129
130
                    return tempPCB;
131
132
133
               {
134
                    tempPCB = tempPCB->nextPCB;
135
                    value++;
136
137
           }
138
           tempPCB = suspendedReady->head;
139
            value = 0;
140
           while (value <= suspendedReady->count)
141
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()

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

# 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()

#### 5.25.2.8 getSuspendedReady()

# 5.25.2.9 insertPCB()

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

```
160
        //BENJAMIN WILL PROGRAM THIS FUNCTION
161
        //insertPCB() will insert a PCB into the appropriate queue.
162
        //Note: The ready queue is a priority queue and the blocked queue is a FIFO queue.
163
164
165
        if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 1)
166
        { // Insert into ready queue
            PCB *tempPtr = ready->head;
167
168
            if (tempPtr != NULL)
169
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
                        PCB_to_insert->nextPCB = tempPtr;
180
                        PCB_to_insert->prevPCB = tempPtr->prevPCB;
181
182
                        tempPtr->prevPCB = PCB_to_insert;
183
```

```
184
                     else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
185
186
                         tempPtr->nextPCB = PCB_to_insert;
                         PCB_to_insert->prevPCB = tempPtr;
ready->tail = PCB_to_insert;
187
188
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
        else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
201
        { // Insert into suspended ready queue
202
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
2.31
            {
232
                suspendedReadv->count++;
                suspendedReady->head = PCB_to_insert;
233
                suspendedReady->tail = PCB_to_insert;
234
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;
2.44
245
        else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
246
        { // Insert into suspended blocked queue
            PCB *tempPtr = suspendedBlocked->tail;
247
248
249
            tempPtr->nextPCB = PCB_to_insert;
250
            PCB_to_insert->prevPCB = tempPtr;
            suspendedBlocked->tail = PCB_to_insert;
251
252
        }
253 }
```

## 5.25.2.10 removePCB()

Definition at line 255 of file R2\_Internal\_Functions\_And\_Structures.c.  $_{256\ \ \{}$ 

```
//BENJAMIN WILL PROGRAM THIS FUNCTION
257
258
259
        //removePCB() will remove a PCB from the queue in which it is currently stored.
260
        PCB *removedPCB = findPCB(PCB_to_remove->processName);
2.61
        if (removedPCB == NULL)
262
263
        {
264
265
        else if (removedPCB == ready->head)
266
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;
            removedNext->prevPCB = NULL;
280
281
            removedPCB->nextPCB = NULL;
            blocked->count--;
282
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
            suspendedBlocked->head = removedNext:
299
300
            removedNext->prevPCB = NULL;
            removedPCB->nextPCB = NULL;
301
302
            suspendedBlocked->count--;
303
            return 0;
304
        else if (removedPCB == readv->tail)
305
306
307
            PCB *removedPrev = removedPCB->prevPCB;
308
309
            ready->tail = removedPrev;
310
            removedPrev->nextPCB = NULL:
            removedPCB->prevPCB = NULL;
311
312
            ready->count--;
313
            return 0;
314
315
        else if (removedPCB == blocked->tail)
316
317
            PCB *removedPrev = removedPCB->prevPCB;
318
319
            blocked->tail = removedPrev;
            removedPrev->nextPCB = NULL;
320
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;
            removedPrev->nextPCB = NULL;
330
            removedPCB->prevPCB = NULL;
331
332
            suspendedReady->count--;
333
            return 0;
334
335
        else if (removedPCB == suspendedBlocked->tail)
336
337
            PCB *removedPrev = removedPCB->prevPCB;
338
            suspendedBlocked->tail = removedPrev;
339
340
            removedPrev->nextPCB = NULL;
341
            removedPCB->prevPCB = NULL;
342
            suspendedBlocked->count--;
            return 0:
343
```

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

### 5.25.2.11 setupPCB()

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
       if (findPCB(processName) ->processName == processName)
56
57
       {
58
           char message[] = "There is already a PCB with this name.\n";
59
           int messLength = strlen(message);
           sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
60
61
           returnedPCB = NULL;
62
64
65
66
           strcpy(returnedPCB->processName, processName);
67
68
           returnedPCB->processClass = processClass;
           returnedPCB->priority = processPriority;
69
70
           returnedPCB->runningStatus = 0;
71
           returnedPCB->suspendedStatus = 1;
72
7.3
74
       return returnedPCB:
```

# 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 <a href="mailto:suspendPCB">suspendPCB</a> (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()

### Definition at line 113 of file R2commands.c.

```
114 { // ANASTASE WILL PROGRAM THIS FUNCTION
115
        \ensuremath{//} find pcb and validate process name
116
        PCB *pcb_to_block = findPCB(processName);
117
118
119
        if (pcb_to_block != NULL)
120
121
            pcb_to_block->runningStatus = -1; // blocked
122
             removePCB(pcb_to_block);
            insertPCB(pcb_to_block);
123
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
       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.
       Priority must be valid.
21
2.2
       if (findPCB(processName) != NULL | | strlen(processName) > 20)
23
24
       { // Check if the process has a unique name, and if it has a valid name.
25
           char errMsg[125];
            strcpy(errMsg, "The PCB could not be created as it either does not have a unique name or the name
26
       is longer than 20 characters!\n");
27
            int errLen = strlen(errMsg);
            sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
28
29
       else if (processClass != 'a' && processClass != 's')
30
       { // Check if the process has a valid class.
            char errMsg[100];
32
           strcpy(errMsg, "The PCB could not be created as it does not have a valid class!\n");
int errLen = strlen(errMsg);
sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
33
34
35
36
       else if (processPriority < 0 || processPriority > 9)
37
38
       { // Check if the process has a valid priority.
39
            char errMsg[100];
            strcpy(errMsg, "The PCB could not be created as it does not have a valid priority!\n"); int errLen = strlen(errMsg);
40
41
           sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
42
43
44
45
       { // Make the PCB
46
           PCB *createdPCB = setupPCB(processName, processClass, processPriority);
47
           char msg[] = "The PCB was created!\n";
int msgLen = strlen(msg);
48
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
59
       The deletePCB command will remove a PCB from the appropriate queue and then free all associated
60
       This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
61
       */
       /*
62
63
       Error Checking:
       Name must be valid.
65
66
67
       if (strlen(processName) > 20)
68
       { // Check if the process has a valid name.
          char errMsg[100];
69
           strcpy(errMsg, "The PCB could not be deleted as the name is longer than 20 characters!\n");
```

```
int errLen = strlen(errMsg);
            sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
72
73
74
        PCB *PCB_to_delete = findPCB(processName);
7.5
76
77
        if (PCB_to_delete == NULL)
78
79
            char errMsg[42] = "The PCB you want to remove does not exist\n";
            int errMsgLen = 42;
sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
80
81
82
83
        else
84
85
            int removed = removePCB(PCB_to_delete);
86
            if (removed == 1)
87
                 char errMsg[] = "The PCB could not be unlinked. \n";
88
                 int errMsgLen = strlen(errMsg);
89
                 sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
91
92
            else
9.3
                 int result = sys_free_mem(PCB_to_delete);
if (result == -1)
94
95
                 {
                     char errMsg[50];
97
                      strcpy(errMsg, "The PCB could not be successfully deleted n");
98
                     int errLen = strlen(errMsg);
sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
99
100
101
                  }
102
                  else
103
104
                       char msg[50];
                      strcpy(msg, "The desired PCB was deleted n");
int msgLen = strlen(msg);
105
106
                      sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
107
108
109
             }
110
         }
111 }
```

# 5.26.1.4 resumePCB()

## 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
196
197
        PCB *PCBtoResume = findPCB(processName);
198
199
        if (PCBtoResume == NULL | strlen(processName) > 20)
200
        {
             char nameError[] = "This is not a valid name.\n";
201
202
             int printCount = strlen(nameError);
203
             sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
204
205
        else
206
207
             removePCB(PCBtoResume);
208
            PCBtoResume->suspendedStatus = 1;
209
            insertPCB(PCBtoResume);
210
            char msg[] = "The PCB was successfully resumed!\n";
211
            int msgLen = strlen(msg);
sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
212
213
215 }
```

#### 5.26.1.5 setPCBPriority()

```
void setPCBPriority (
              char * processName,
              int newProcessPriority)
Definition at line 217 of file R2commands.c.
218 { // ANASTASE WILL PROGRAM THIS FUNCTION
220
        // Sets a PCB's priority and reinserts the process into the correct place in the correct queue
221
2.2.2
223
        Error Checking:
224
        Name must be valid.
225
        newPriority
226
227
        // find the process and validate the name
228
229
        PCB *tempPCB = findPCB(processName);
230
231
        if ((tempPCB != NULL) && (newProcessPriority >= 0) && (newProcessPriority < 10))
232
            tempPCB->priority = newProcessPriority;
233
            removePCB(tempPCB);
234
            insertPCB(tempPCB);
235
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
617
            Class
618
            State
            Suspended Status
619
            Priority
620
621
622
623
        Error Checking:
624
        None
62.5
        */
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";
        int message_size = strlen(print_message);
sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
578
579
580
581
         // printPCBs(blocked);
        queue *tempQueue = getBlocked();
582
583
         PCB *tempPtr = tempQueue->head; //PCB_container->head;
584
         int count = tempQueue->count;
585
        if (count == 0)
586
587
588
             // the queue is empty
             char error_message[30] = "The queue is empty.\n";
int error_size = strlen(error_message);
589
590
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);
         //sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
600
601
602
         while (value < count)</pre>
603
         { // testing for <== or <
             // Print out the process
604
605
             showPCB(tempPtr->processName);
606
             // increment pcb*tempPtr, the loop variable.
             tempPtr = tempPtr->nextPCB;
607
             value++;
608
609
610 }
```

# 5.26.1.8 showPCB()

### 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
2.51
             Priority
252
253
254
255
        Error Checking:
256
        Name must be valid.
257
258
259
        if (strlen(processName) > 20)
        { // Check if the process has a valid name.
260
261
             char errMsg[100];
             strcpy(errMsg, "The PCB could not be shown as the name is longer than 20 characters!\n");
int errLen = strlen(errMsg);
262
263
             sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
2.64
265
        }
266
        else
267
        {
```

```
268
              PCB *PCB_to_show = findPCB(processName);
270
271
              if (PCB_to_show == NULL)
2.72
              { // Check to see if the PCB exists.
273
                   char errMsq[100];
                   strcpy(errMsg, "The PCB could not be shown, as it does not exist!\n");
274
275
                   int errLen = strlen(errMsg);
276
                   sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
277
278
              else
279
                   // Print out the PCB name.
280
                   char nameMsg[50];
281
                   strcpy(nameMsg, "The process name is: ");
int nameMsgLen = strlen(nameMsg);
282
283
284
                   sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
                   char name[20];
285
286
                   strcpy(name, PCB_to_show->processName);
                   int nameLen = strlen(name);
287
288
                   sys_req(WRITE, DEFAULT_DEVICE, name, &nameLen);
                   char newLine[1];
strcpy(newLine, "\n");
int newLineLen = 1;
289
290
291
                   sys_reg(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);
292
293
294
                   // Print out PCB class
295
                   char classMsg[50];
                   strcpy(classMsg, "The process class is: ");
int classMsgLen = strlen(classMsg);
296
297
298
                   sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
299
300
                   if (PCB_to_show->processClass == 'a')
301
                   {
                        char appMsg[50];
strcpy(appMsg, "application");
int appMsgLen = strlen(appMsg);
302
303
304
                        sys_req(WRITE, DEFAULT_DEVICE, appMsg, &appMsgLen);
305
306
                   }
307
                   else
308
309
                        char sysMsg[50];
                        strcpy(sysMsg, "system");
int sysMsgLen = strlen(sysMsg);
310
311
                        sys_req(WRITE, DEFAULT_DEVICE, sysMsg, &sysMsgLen);
312
314
                   sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);
315
                   // Print out the PCB state
316
317
318
                   if (PCB_to_show->runningStatus == 0)
319
                   { // The process is ready.
                        char stateMsg[50];
strcpy(stateMsg, "The process is ready!\n");
int stateMsgLen = strlen(stateMsg);
320
321
322
                        sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
323
325
                   else if (PCB_to_show->runningStatus == -1)
326
                   { // The process is blocked.
                        char stateMsg[50];
strcpy(stateMsg, "The process is blocked!\n");
int stateMsgLen = strlen(stateMsg);
327
328
329
330
                        sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
331
332
                   else if (PCB_to_show->runningStatus == 1)
333
                   \{\ //\ {\it The process is running.}
                        char stateMsg[50];
strcpy(stateMsg, "The process is running!\n");
int stateMsgLen = strlen(stateMsg);
334
335
336
                        sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
337
338
339
340
                   // Print out the PCB suspended status
341
                   if (PCB to show->suspendedStatus == 0)
342
343
                   { // The process is suspended
344
                        char susMsg[50];
                        strcpy(susMsg, "The process is suspended!\n");
int susMsgLen = strlen(susMsg);
345
346
                        sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
347
348
349
                   else if (PCB_to_show->suspendedStatus == 1)
                   { // The process is not suspended
350
351
                        char susMsg[50];
                        strcpy(susMsg, "The process is not suspended!\n");
int susMsgLen = strlen(susMsg);
352
353
                        sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
354
```

```
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:
                      strcpy(priorityMsg, "The process priority is 0!\n");
priorityMsgLen = strlen(priorityMsg);
sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
364
365
366
367
                      break;
368
369
                  case 1:
                      370
371
                       sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
372
373
374
375
376
                      strcpy(priorityMsg, "The process priority is 2!\n");
                      priorityMsgLen = strlen(priorityMsg);
sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
377
378
379
                      break;
380
381
382
                      strcpy(priorityMsg, "The process priority is 3!\n");
                      priorityMsgLen = strlen(priorityMsg);
sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
383
384
385
                      break;
386
387
388
                       strcpy(priorityMsg, "The process priority is 4!\n");
                      priorityMsglen = strlen(priorityMsg);
sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
389
390
391
                      break;
392
393
394
                      strcpy(priorityMsg, "The process priority is 5!\n");
395
                      priorityMsgLen = strlen(priorityMsg);
                       sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
396
397
                      break:
398
399
400
                      strcpy(priorityMsg, "The process priority is 6!\n");
401
                      priorityMsgLen = strlen(priorityMsg);
402
                       sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
403
404
405
                  case 7:
406
                      strcpy(priorityMsg, "The process priority is 7!\n");
                      priorityMsgLen = strlen(priorityMsg);
sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
407
408
409
                      break:
410
                  case 8:
412
                      strcpy(priorityMsg, "The process priority is 8!\n");
                       priorityMsgLen = strlen(priorityMsg);
413
414
                       sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
415
                      break:
416
417
                  case 9:
                      strcpy(priorityMsg, "The process priority is 9!\n");
418
419
                      priorityMsgLen = strlen(priorityMsg);
                       sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
420
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
433
        Displays the following information for each PCB in the ready queue:
434
             Process Name
435
             Class
436
             State
             Suspended Status
437
438
            Priority
439
440
        /*
        Error Checking:
441
442
        None
443
444
        char message[] = "Printing the ready queue:\n"; int messLength = strlen(message);
445
446
        sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
447
448
449
        queue *tempQueue = getReady();
450
        PCB *tempPCB = tempQueue->head;
451
452
        int loop = 0;
        int count = tempQueue->count;
453
454
455
        if (count == 0)
456
457
             // the queue is empty
             char error_message[30] = "The queue is empty.\n";
458
459
             int error_size = strlen(error_message);
             sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
460
461
            return;
462
        }
463
4\,6\,4
        while (loop < count)</pre>
465
             showPCB(tempPCB->processName);
466
             PCB *tempNext = tempPCB->nextPCB;
467
            loop++;
468
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
         char message[] = "Printing the suspended blocked queue:\n";
int messLength = strlen(message);
sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
531
532
533
534
535
         queue *tempQueue = getSuspendedBlocked();
536
         PCB *tempPCB = tempQueue->head;
537
538
         int loop = 0;
         int count = tempQueue->count;
539
540
541
         if (count == 0)
542
         {
543
              // the queue is empty
              char error_message[30] = "The queue is empty.\n"; int error_size = strlen(error_message);
544
545
              sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
546
547
              return;
548
```

```
549
550     while (loop < count)
551     {
          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
         Displays the following information for each PCB in the suspended ready queue:
476
477
             Process Name
478
              Class
479
              State
480
              Suspended Status
481
             Priority
482
483
         Error Checking:
484
485
         None
486
487
         char message[] = "Printing the suspended ready queue:\n"; int messLength = strlen(message);
488
489
         sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
490
491
492
         queue *tempQueue = getSuspendedReady();
493
         PCB *tempPCB = tempQueue->head;
494
         int loop = 0;
int count = tempQueue->count;
495
496
497
         if (count == 0)
498
499
500
              // the queue is empty
              char error_message[30] = "The queue is empty.\n";
501
              int error_size = strlen(error_message);
502
              sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
503
504
             return;
505
         }
506
507
         while (loop < count)</pre>
508
             showPCB(tempPCB->processName);
PCB *tempNext = tempPCB->nextPCB;
509
510
511
              loop++;
512
             tempPCB = tempNext;
513
514 }
```

# 5.26.1.12 suspendPCB()

168

```
169
        if (PCBtoSuspend == NULL | | strlen(processName) > 20)
170
            char nameError[] = "This is not a valid name.\n";
171
172
            int printCount = strlen(nameError);
            sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
173
174
175
       else
176
            removePCB(PCBtoSuspend);
177
178
            PCBtoSuspend->suspendedStatus = 0;
           insertPCB(PCBtoSuspend);
179
180
           char msg[] = "The PCB was successfully suspended!\n";
int msgLen = strlen(msg);
181
182
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
        Places a PCB in the unblocked state and reinserts it into the appropriate queue.
135
136
137
        Error Checking:
138
139
        Name must be valid.
140
141
142
        PCB *pcb_to_unblock = findPCB(processName);
144
        if (pcb_to_unblock != NULL)
145
             pcb_to_unblock->runningStatus = 0; // ready
146
                                                   // is this the right place to put that function?
147
             removePCB(pcb to unblock);
148
             insertPCB(pcb_to_unblock);
149
150
             char msg[] = "The PCB was successfully unblocked!\n";
             int msglen = wine PCB was successfully unbloc
int msgLen = strlen(msg);
sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
151
152
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
        PCB *pcb_to_block = findPCB(processName);
117
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()

#### 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:
       Name must be unique and valid.
18
19
       Class must be valid.
       Priority must be valid.
22
       if (findPCB(processName) != NULL || strlen(processName) > 20)
2.3
       { // Check if the process has a unique name, and if it has a valid name.
24
           char errMsg[125];
25
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");
2.7
            int errLen = strlen(errMsg);
            sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
2.8
29
       else if (processClass != 'a' && processClass != 's')
30
31
       { // Check if the process has a valid class.
            char errMsg[100];
32
33
            strcpy(errMsg, "The PCB could not be created as it does not have a valid class!\n");
34
            int errLen = strlen(errMsg);
            sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
35
36
       else if (processPriority < 0 || processPriority > 9)
38
       { // Check if the process has a valid priority.
39
            char errMsg[100];
40
            \textbf{strcpy} \, (\texttt{errMsg}, \,\, \texttt{"The PCB could not be created as it does not have a valid priority!} \, | \, n \texttt{"}) \, ;
           int errLen = strlen(errMsg);
sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
41
42
43
       else
```

#### 5.27.1.3 deletePCB()

#### Definition at line 56 of file R2commands.c.

```
57 { // BENJAMIN WILL PROGRAM THIS FUNCTION
59
       The deletePCB command will remove a PCB from the appropriate queue and then free all associated
       memory.
       This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
60
61
62
       Error Checking:
       Name must be valid.
65
66
       if (strlen(processName) > 20)
67
68
       { // Check if the process has a valid name.
           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
            char errMsg[42] = "The PCB you want to remove does not exist\n";
79
            int errMsgLen = 42;
sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
80
82
83
84
            int removed = removePCB(PCB_to_delete);
if (removed == 1)
8.5
86
                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
                int result = sys_free_mem(PCB_to_delete);
95
                if (result == -1)
96
                     char errMsg[50]; strcpy(errMsg, "The PCB could not be successfully deleted n"); int errLen = strlen(errMsg);
97
98
99
100
                      sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
101
102
                 else
103
                      char msg[50];
104
                      strcpy(msg, "The desired PCB was deleted n");
int msgLen = strlen(msg);
105
106
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
196
197
        PCB *PCBtoResume = findPCB(processName);
198
199
        if (PCBtoResume == NULL | | strlen(processName) > 20)
200
             char nameError[] = "This is not a valid name.\n";
201
             int printCount = strlen(nameError);
sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
202
203
204
205
        else
206
207
             removePCB(PCBtoResume);
208
            PCBtoResume->suspendedStatus = 1;
            insertPCB(PCBtoResume);
209
211
            char msg[] = "The PCB was successfully resumed!\n";
212
             int msgLen = strlen(msg);
             sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
213
214
215 }
```

### 5.27.1.5 setPCBPriority()

#### 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
        if ((tempPCB != NULL) && (newProcessPriority >= 0) && (newProcessPriority < 10))</pre>
231
232
             tempPCB->priority = newProcessPriority;
233
234
             removePCB(tempPCB);
235
            insertPCB(tempPCB);
236
            char msg[] = "The PCB's priority was successfully changed!\n"; int msgLen = strlen(msg);
237
238
            sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
239
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:
            Process Name
616
617
            Class
618
            State
            Suspended Status
619
            Priority
620
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
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
        char print_message[30] = "The blocked queue:\n";
577
        int message_size = strlen(print_message);
578
        sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
579
580
581
        // printPCBs(blocked);
582
        queue *tempQueue = getBlocked();
        PCB *tempPtr = tempQueue->head; //PCB_container->head;
int count = tempQueue->count;
583
584
585
586
        if (count == 0)
587
588
             // the queue is empty
             char error_message[30] = "The queue is empty.\n";
589
             int error_size = strlen(error_message);
590
591
             sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
592
593
594
        // The queue is not empty
595
596
        int value = 0:
597
        // Testing purpose
        //char print_message[38]="The blocke queue testing:\n";
598
599
        //int message_size=strlen(print_message);
600
        //sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
601
602
        while (value < count)</pre>
        { // testing for <== or < // Print out the process
603
604
605
             showPCB(tempPtr->processName);
```

#### 5.27.1.8 showPCB()

```
void showPCB (
                  char * processName )
Definition at line 243 of file R2commands.c.
244 { // BENJAMIN WILL PROGRAM THIS FUNCTION
245
          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];
               strcpy(errMsg, "The PCB could not be shown as the name is longer than 20 characters!\n");
int errLen = strlen(errMsg);
sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
262
263
264
265
266
          else
267
2.68
               PCB *PCB to show = findPCB(processName);
269
270
               if (PCB_to_show == NULL)
271
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(errMsq);
                    sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
276
277
278
               else
279
                    // Print out the PCB name.
280
                    char nameMsg[50];
strcpy(nameMsg, "The process name is: ");
int nameMsgLen = strlen(nameMsg);
281
282
283
284
                    sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
                    char name[20];
285
                    strcpy(name, PCB_to_show->processName);
int nameLen = strlen(name);
sys_req(WRITE, DEFAULT_DEVICE, name, &nameLen);
286
287
288
289
                    char newLine[1];
                    strcpy(newLine, "\n");
int newLineLen = 1;
290
291
292
                    sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);
293
                    // Print out PCB class
294
                    char classMsg[50];
strcpy(classMsg, "The process class is: ");
int classMsgLen = strlen(classMsg);
295
296
297
298
                    sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
299
300
                    if (PCB_to_show->processClass == 'a')
301
302
                         char appMsg[50];
                         strcpy(appMsg, "application");
int appMsgLen = strlen(appMsg);
sys_req(WRITE, DEFAULT_DEVICE, appMsg, &appMsgLen);
303
304
305
306
307
                    else
308
309
                         char sysMsg[50];
```

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

```
397
                          break;
398
399
                     case 6:
                          strcpy(priorityMsg, "The process priority is 6!\ n");
400
                          priorityMsgLen = strlen(priorityMsg);
sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
401
402
403
404
405
                     case 7:
                          strcpy(priorityMsg, "The process priority is 7!\n");
priorityMsgLen = strlen(priorityMsg);
sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
406
407
408
409
                          break;
410
411
                     case 8:
                          412
413
                          sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
414
415
                          break;
416
417
                     case 9:
418
                          strcpy(priorityMsg, "The process priority is 9!\ n");
                          strcpy(prioritymsg, "The process priority is 5:11 ),
priorityMsgLen = strlen(priorityMsg);
sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
419
420
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
436
437
             Suspended Status
438
             Priority
439
440
441
        Error Checking:
442
        None
443
444
        char message[] = "Printing the ready queue:\n";
445
        int messLength = strlen(message);
sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
446
447
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
             // the queue is empty
457
             char error_message[30] = "The queue is empty.\n";
int error_size = strlen(error_message);
458
459
             sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
460
461
462
         }
463
464
        while (loop < count)</pre>
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
519
         Displays the following information for each PCB in the suspended blocked queue:
520
             Process Name
521
              Class
522
              State
              Suspended Status
523
524
              Priority
525
526
527
         Error Checking:
528
         None
529
530
         char message[] = "Printing the suspended blocked queue:\n"; int messLength = strlen(message);
531
532
533
         sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
534
         queue *tempQueue = getSuspendedBlocked();
535
536
         PCB *tempPCB = tempQueue->head;
537
538
         int loop = 0;
539
         int count = tempQueue->count;
540
541
         if (count == 0)
542
543
              // the gueue is empty
              int equation in the quality in the quality in the error_message[30] = "The quality in the error_size = strlen(error_message);
544
545
546
              sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
547
              return;
548
         }
549
         while (loop < count)</pre>
551
552
              showPCB(tempPCB->processName);
553
              PCB *tempNext = tempPCB->nextPCB;
554
              loop++;
              tempPCB = tempNext;
555
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
480
            Suspended Status
481
            Priority
482
483
484
        Error Checking:
485
        None
486
487
        char message[] = "Printing the suspended ready queue:\n";
488
        int messLength = strlen(message);
489
        sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
490
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
              char error_message[30] = "The queue is empty.\n";
501
              int error_size = strlen(error_message);
sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
502
503
504
505
506
507
         while (loop < count)</pre>
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
166
167
         PCB *PCBtoSuspend = findPCB(processName);
168
         if (PCBtoSuspend == NULL | | strlen(processName) > 20)
169
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
             removePCB(PCBtoSuspend);
177
178
             PCBtoSuspend->suspendedStatus = 0;
179
             insertPCB(PCBtoSuspend);
180
             char msg[] = "The PCB was successfully suspended!\n";
int msgLen = strlen(msg);
sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
181
182
```

#### 5.27.1.13 unblockPCB()

145

```
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)
```

```
pcb_to_unblock->runningStatus = 0; // ready
removePCB(pcb_to_unblock); // is this the right place to put that function?
insertPCB(pcb_to_unblock);

char msg[] = "The PCB was successfully unblocked!\n";
int msgLen = strlen(msg);
sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
}

sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
}
```

# 5.28 README.md File Reference