MPX-Fall2020-Group9

3.75

Generated by Doxygen 1.9.0

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:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 alarm Struct Reference

`#include <R4commands.h>`

Public Attributes

- char alarmName [20]
- int alarmTime
- struct alarm * nextAlarm
- struct alarm * prevAlarm

4.1.1 Detailed Description

Definition at line 3 of file R4commands.h.

4.1.2 Member Data Documentation

4.1.2.1 alarmName

`char alarm::alarmName[20]`

Definition at line 5 of file R4commands.h.

4.1.2.2 alarmTime

`int alarm::alarmTime`

Definition at line 6 of file R4commands.h.

4.1.2.3 nextAlarm

```
struct alarm* alarm::nextAlarm
```

Definition at line 7 of file R4commands.h.

4.1.2.4 prevAlarm

```
struct alarm* alarm::prevAlarm
```

Definition at line 8 of file R4commands.h.

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

- modules/R4/R4commands.h

4.2 alarmList Struct Reference

```
#include <R4commands.h>
```

Public Attributes

- int count
- alarm * head
- alarm * tail

4.2.1 Detailed Description

Definition at line 11 of file R4commands.h.

4.2.2 Member Data Documentation

4.2.2.1 count

```
int alarmList::count
```

Definition at line 13 of file R4commands.h.

4.2.2.2 head

```
alarm* alarmList::head
```

Definition at line 14 of file R4commands.h.

4.2.2.3 tail

```
alarm* alarmList::tail
```

Definition at line 15 of file R4commands.h.

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

- modules/R4/R4commands.h

4.3 context Struct Reference

```
#include <R3commands.h>
```

Public Attributes

- u32int gs
- u32int fs
- u32int es
- u32int ds
- u32int edi
- u32int esi
- u32int ebp
- u32int esp
- u32int ebx
- u32int edx
- u32int ecx
- u32int eax
- u32int eip
- u32int cs
- u32int eflags

4.3.1 Detailed Description

Definition at line 3 of file R3commands.h.

4.3.2 Member Data Documentation

4.3.2.1 cs

```
u32int context::cs
```

Definition at line 7 of file R3commands.h.

4.3.2.2 ds

```
u32int context::ds
```

Definition at line 5 of file R3commands.h.

4.3.2.3 eax

```
u32int context::eax
```

Definition at line 6 of file R3commands.h.

4.3.2.4 ebp

```
u32int context::ebp
```

Definition at line 6 of file R3commands.h.

4.3.2.5 ebx

```
u32int context::ebx
```

Definition at line 6 of file R3commands.h.

4.3.2.6 ecx

```
u32int context::ecx
```

Definition at line 6 of file R3commands.h.

4.3.2.7 edi

```
u32int context::edi
```

Definition at line 6 of file R3commands.h.

4.3.2.8 edx

```
u32int context::edx
```

Definition at line 6 of file R3commands.h.

4.3.2.9 eflags

```
u32int context::eflags
```

Definition at line 7 of file R3commands.h.

4.3.2.10 eip

```
u32int context::eip
```

Definition at line 7 of file R3commands.h.

4.3.2.11 es

```
u32int context::es
```

Definition at line 5 of file R3commands.h.

4.3.2.12 esi

```
u32int context::esi
```

Definition at line 6 of file R3commands.h.

4.3.2.13 esp

```
u32int context::esp
```

Definition at line 6 of file R3commands.h.

4.3.2.14 fs

```
u32int context::fs
```

Definition at line 5 of file R3commands.h.

4.3.2.15 gs

```
u32int context::gs
```

Definition at line 5 of file R3commands.h.

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

- modules/R3/R3commands.h

4.4 date_time Struct Reference

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

Public Attributes

- int sec
- int min
- int hour
- int day_w
- int day_m
- int day_y
- int mon
- int year

4.4.1 Detailed Description

Definition at line 30 of file system.h.

4.4.2 Member Data Documentation

4.4.2.1 day_m

```
int date_time::day_m
```

Definition at line 35 of file system.h.

4.4.2.2 day_w

```
int date_time::day_w
```

Definition at line 34 of file system.h.

4.4.2.3 day_y

```
int date_time::day_y
```

Definition at line 36 of file system.h.

4.4.2.4 hour

```
int date_time::hour
```

Definition at line 33 of file system.h.

4.4.2.5 min

```
int date_time::min
```

Definition at line 32 of file system.h.

4.4.2.6 mon

```
int date_time::mon
```

Definition at line 37 of file system.h.

4.4.2.7 sec

```
int date_time::sec
```

Definition at line 31 of file system.h.

4.4.2.8 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

4.5 footer Struct Reference

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

Public Attributes

- header head

4.5.1 Detailed Description

Definition at line 16 of file heap.h.

4.5.2 Member Data Documentation

4.5.2.1 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

4.6 gdt_descriptor_struct Struct Reference

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

Public Attributes

- u16int limit
- u32int base

4.6.1 Detailed Description

Definition at line 23 of file tables.h.

4.6.2 Member Data Documentation

4.6.2.1 base

```
u32int gdt_descriptor_struct::base
```

Definition at line 26 of file tables.h.

4.6.2.2 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

4.7 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

4.7.1 Detailed Description

Definition at line 30 of file tables.h.

4.7.2 Member Data Documentation

4.7.2.1 access

`u8int gdt_entry_struct::access`

Definition at line 35 of file tables.h.

4.7.2.2 base_high

`u8int gdt_entry_struct::base_high`

Definition at line 37 of file tables.h.

4.7.2.3 base_low

`u16int gdt_entry_struct::base_low`

Definition at line 33 of file tables.h.

4.7.2.4 base_mid

```
u8int gdt_entry_struct::base_mid
```

Definition at line 34 of file tables.h.

4.7.2.5 flags

```
u8int gdt_entry_struct::flags
```

Definition at line 36 of file tables.h.

4.7.2.6 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

4.8 header Struct Reference

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

Public Attributes

- int size
- int index_id

4.8.1 Detailed Description

Definition at line 11 of file heap.h.

4.8.2 Member Data Documentation

4.8.2.1 index_id

```
int header::index_id
```

Definition at line 13 of file heap.h.

4.8.2.2 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

4.9 heap Struct Reference

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

Public Attributes

- index_table index
- u32int base
- u32int max_size
- u32int min_size

4.9.1 Detailed Description

Definition at line 33 of file heap.h.

4.9.2 Member Data Documentation

4.9.2.1 base

```
u32int heap::base
```

Definition at line 35 of file heap.h.

4.9.2.2 index

```
index_table heap::index
```

Definition at line 34 of file heap.h.

4.9.2.3 max_size

```
u32int heap::max_size
```

Definition at line 36 of file heap.h.

4.9.2.4 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

4.10 idt_entry_struct Struct Reference

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

Public Attributes

- u16int base_low
- u16int sselect
- u8int zero
- u8int flags
- u16int base_high

4.10.1 Detailed Description

Definition at line 6 of file tables.h.

4.10.2 Member Data Documentation

4.10.2.1 base_high

```
u16int idt_entry_struct::base_high
```

Definition at line 12 of file tables.h.

4.10.2.2 base_low

```
u16int idt_entry_struct::base_low
```

Definition at line 8 of file tables.h.

4.10.2.3 flags

```
u8int idt_entry_struct::flags
```

Definition at line 11 of file tables.h.

4.10.2.4 sselect

```
u16int idt_entry_struct::sselect
```

Definition at line 9 of file tables.h.

4.10.2.5 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

4.11 idt_struct Struct Reference

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

Public Attributes

- u16int limit
- u32int base

4.11.1 Detailed Description

Definition at line 16 of file tables.h.

4.11.2 Member Data Documentation

4.11.2.1 base

```
u32int idt_struct::base
```

Definition at line 19 of file tables.h.

4.11.2.2 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

4.12 index_entry Struct Reference

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

Public Attributes

- int size
- int empty
- u32int block

4.12.1 Detailed Description

Definition at line 20 of file heap.h.

4.12.2 Member Data Documentation

4.12.2.1 block

```
u32int index_entry::block
```

Definition at line 23 of file heap.h.

4.12.2.2 empty

```
int index_entry::empty
```

Definition at line 22 of file heap.h.

4.12.2.3 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

4.13 index_table Struct Reference

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

Public Attributes

- index_entry table [0x1000]
- int id

4.13.1 Detailed Description

Definition at line 27 of file heap.h.

4.13.2 Member Data Documentation

4.13.2.1 id

```
int index_table::id
```

Definition at line 29 of file heap.h.

4.13.2.2 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

4.14 page_dir Struct Reference

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

Public Attributes

- page_table * tables [1024]
- u32int tables_phys [1024]

4.14.1 Detailed Description

Definition at line 34 of file paging.h.

4.14.2 Member Data Documentation

4.14.2.1 tables

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

Definition at line 35 of file paging.h.

4.14.2.2 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

4.15 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

4.15.1 Detailed Description

Definition at line 12 of file paging.h.

4.15.2 Member Data Documentation

4.15.2.1 accessed

```
u32int page_entry::accessed
```

Definition at line 16 of file paging.h.

4.15.2.2 dirty

```
u32int page_entry::dirty
```

Definition at line 17 of file paging.h.

4.15.2.3 frameaddr

```
u32int page_entry::frameaddr
```

Definition at line 19 of file paging.h.

4.15.2.4 present

```
u32int page_entry::present
```

Definition at line 13 of file paging.h.

4.15.2.5 reserved

```
u32int page_entry::reserved
```

Definition at line 18 of file paging.h.

4.15.2.6 usermode

```
u32int page_entry::usermode
```

Definition at line 15 of file paging.h.

4.15.2.7 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

4.16 page_table Struct Reference

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

Public Attributes

- page_entry pages [1024]

4.16.1 Detailed Description

Definition at line 26 of file paging.h.

4.16.2 Member Data Documentation

4.16.2.1 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

4.17 param Struct Reference

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

Public Attributes

- int op_code
- int device_id
- char * buffer_ptr
- int * count_ptr

4.17.1 Detailed Description

Definition at line 31 of file mpx_supt.h.

4.17.2 Member Data Documentation

4.17.2.1 buffer_ptr

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

Definition at line 34 of file mpx_supt.h.

4.17.2.2 count_ptr

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

Definition at line 35 of file mpx_supt.h.

4.17.2.3 device_id

```
int param::device_id
```

Definition at line 33 of file mpx_supt.h.

4.17.2.4 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

4.18 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

4.18.1 Detailed Description

Definition at line 1 of file R2_Internal_Functions_And_Structures.h.

4.18.2 Member Data Documentation

4.18.2.1 nextPCB

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

Definition at line 11 of file R2_Internal_Functions_And_Structures.h.

4.18.2.2 prevPCB

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

Definition at line 12 of file R2_Internal_Functions_And_Structures.h.

4.18.2.3 priority

```
int PCB::priority
```

Definition at line 5 of file R2_Internal_Functions_And_Structures.h.

4.18.2.4 processClass

```
char PCB::processClass
```

Definition at line 4 of file R2_Internal_Functions_And_Structures.h.

4.18.2.5 processName

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

Definition at line 3 of file R2_Internal_Functions_And_Structures.h.

4.18.2.6 runningStatus

```
int PCB::runningStatus
```

Definition at line 6 of file R2_Internal_Functions_And_Structures.h.

4.18.2.7 stack

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

Definition at line 8 of file R2_Internal_Functions_And_Structures.h.

4.18.2.8 stackBase

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

Definition at line 10 of file R2_Internal_Functions_And_Structures.h.

4.18.2.9 stackTop

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

Definition at line 9 of file R2_Internal_Functions_And_Structures.h.

4.18.2.10 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

4.19 queue Struct Reference

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

Public Attributes

- int count
- PCB * head
- PCB * tail

4.19.1 Detailed Description

Definition at line 15 of file R2_Internal_Functions_And_Structures.h.

4.19.2 Member Data Documentation

4.19.2.1 count

```
int queue::count
```

Definition at line 17 of file R2_Internal_Functions_And_Structures.h.

4.19.2.2 head

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

Definition at line 18 of file R2_Internal_Functions_And_Structures.h.

4.19.2.3 tail

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

Definition at line 19 of file R2_Internal_Functions_And_Structures.h.

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

- modules/R2/R2_Internal_Functions_And_Structures.h

Chapter 5

File Documentation

5.1 include/core/asm.h File Reference

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

5.2 include/core/interrupts.h File Reference

Functions

- void init_irq (void)
- void init_pic (void)

5.2.1 Function Documentation

5.2.1.1 init_irq()

```
void init_irq (
            void  )
```

Definition at line 67 of file interrupts.c.

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

```
83        (u32int)invalid_tss,
84        (u32int)segment_not_present,
85        (u32int)stack_segment,
86        (u32int)general_protection,
87        (u32int)page_fault,
88        (u32int)reserved,
89        (u32int)coprocessor};
90
91    // Install handlers; 0x08=sel, 0x8e=flags
92    for (i = 0; i < 32; i++)
93    {
94      if (i < 17)
95        idt_set_gate(i, isrs[i], 0x08, 0x8e);
96      else
97        idt_set_gate(i, (u32int)reserved, 0x08, 0x8e);
98    }
99    // Ignore interrupts from the real time clock
100    idt_set_gate(0x08, (u32int)rtc_isr, 0x08, 0x8e);
101    idt_set_gate(60, (u32int)sys_call_isr, 0x08, 0x8e);
102 }
```

5.2.1.2 init_pic()

```
void init_pic (
              void  )
```

Definition at line 110 of file interrupts.c.

```
111 {
112   outb(PIC1, ICW1); //send initialization code words 1 to PIC1
113   io_wait();
114   outb(PIC2, ICW1); //send icw1 to PIC2
115   io_wait();
116   outb(PIC1 + 1, 0x20); //icw2: remap irq0 to 32
117   io_wait();
118   outb(PIC2 + 1, 0x28); //icw2: remap irq8 to 40
119   io_wait();
120   outb(PIC1 + 1, 4); //icw3
121   io_wait();
122   outb(PIC2 + 1, 2); //icw3
123   io_wait();
124   outb(PIC1 + 1, ICW4); //icw4: 80x86, automatic handling
125   io_wait();
126   outb(PIC2 + 1, ICW4); //icw4: 80x86, automatic handling
127   io_wait();
128   outb(PIC1 + 1, 0xFF); //disable irqs for PIC1
129   io_wait();
130   outb(PIC2 + 1, 0xFF); //disable irqs for PIC2
131 }
```

5.3 include/core/io.h File Reference

Macros

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

5.3.1 Macro Definition Documentation

5.3.1.1 inb

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

Value:

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

Definition at line 15 of file io.h.

5.3.1.2 outb

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

Definition at line 8 of file io.h.

5.4 include/core/serial.h File Reference

Macros

- #define COM1 0x3f8
- #define COM2 0x2f8
- #define COM3 0x3e8
- #define COM4 0x2e8

Functions

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

5.4.1 Macro Definition Documentation

5.4.1.1 COM1

```
#define COM1 0x3f8
```

Definition at line 4 of file serial.h.

5.4.1.2 COM2

```
#define COM2 0x2f8
```

Definition at line 5 of file serial.h.

5.4.1.3 COM3

```
#define COM3 0x3e8
```

Definition at line 6 of file serial.h.

5.4.1.4 COM4

```
#define COM4 0x2e8
```

Definition at line 7 of file serial.h.

5.4.2 Function Documentation

5.4.2.1 init_serial()

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

Definition at line 22 of file serial.c.

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

5.4.2.2 polling()

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

Definition at line 92 of file serial.c.

```
93 {
94   // insert your code to gather keyboard input via the technique of polling.
95
96   char keyboard_character;
97
98   int cursor = 0;
99
100   char log[] = {'\0', '\0', '\0', '\0'};
101
102   int characters_in_buffer = 0;
103
104   while (1)
105   {
106
107     if (inb(COM1 + 5) & 1)
108     {                                   // is there input char?
109       keyboard_character = inb(COM1); //read the char from COM1
110
111       if (keyboard_character == '\n' || keyboard_character == '\r')
112       { // HANDLEING THE CARRIAGE RETURN AND NEW LINE CHARACTERS
113
114         buffer[characters_in_buffer] = '\0';
115         break;
116       }
117       else if ((keyboard_character == 127 || keyboard_character == 8) && cursor > 0)
118       { // HANDELING THE BACKSPACE CHARACTER
119
120         //serial_println("Handleing backspace character.");
121         serial_print("\033[K");
122
123         buffer[cursor - 1] = '\0';
124         serial_print("\b \b");
125         serial_print(buffer + cursor);
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];
133           buffer[temp_cursor + 1] = '\0';
134           temp_cursor++;
135         }
136
137         characters_in_buffer--;
138         cursor = characters_in_buffer;
139       }
140       else if (keyboard_character == '~' && cursor < 99)
141       { //HANDLING THE DELETE KEY
142         // \033[3~
143
144         serial_print("\033[K");
145
146         buffer[cursor + 1] = '\0';
147         serial_print("\b \b");
148         serial_print(buffer + cursor);
149
150         int temp_cursor = cursor + 1;
151
152         while (buffer[temp_cursor + 1] != '\0')
153         {
154           buffer[temp_cursor] = buffer[temp_cursor + 1];
155           buffer[temp_cursor + 1] = '\0';
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       }
172       else if (log[0] == '\033' && log[1] == '[')
173       { // HANDLEING LAST CHARACTER FOR ARROW KEYS
174         log[2] = keyboard_character;
175 
176         if (keyboard_character == 'A')
177         { //Up arrow
178           //Call a history function from the commhand or do nothing
179         }
180         else if (keyboard_character == 'B')
181         { //Down arrow
182           //Call a history command from the commhand or do nothing
183         }
184         else if (keyboard_character == 'C' && cursor != 99)
185         { //Right arrow
186 
187           serial_print("\033[C");
188           cursor++;
189         }
190         else if (keyboard_character == 'D' && cursor != 0)
191         { //Left arrow
192 
193           serial_print("\033[D");
194           cursor--;
195         }
196 
197         memset(log, '\0', 4);
198       }
199       else
200       {
201 
202         if (cursor == 0 && buffer[cursor] == '\0') //Adding character at beginning of buffer
203         {
204           buffer[cursor] = keyboard_character;
205           serial_print(buffer + cursor);
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(buffer + cursor);
212           cursor++;
213         }
214         else //Inserting character to the middle of the buffer
215         {
216           char temp_buffer[strlen(buffer)];
217           memset(temp_buffer, '\0', strlen(buffer));
218 
219           int temp_cursor = 0;
220           while (temp_cursor <= characters_in_buffer) //Filling the temp_buffer with all of the
    characters from buffer, and inserting the new character.
221           {
222             if (temp_cursor < cursor)
223             {
224               temp_buffer[temp_cursor] = buffer[temp_cursor];
225             }
226             else if (temp_cursor > cursor)
227             {
228               temp_buffer[temp_cursor] = buffer[temp_cursor - 1];
229             }
230             else
231             { //temp_cursor == cursor
232               temp_buffer[temp_cursor] = keyboard_character;
233             }
234             temp_cursor++;
235           }
236 
237           temp_cursor = 0;
238           int temp_buffer_size = strlen(temp_buffer);
239           while (temp_cursor <= temp_buffer_size) //Setting the contents of the buffer equal to the
    temp_buffer.
240           {
241             buffer[temp_cursor] = temp_buffer[temp_cursor];
242             temp_cursor++;
243           }
244 
245           serial_print("\033[K");
246           serial_print(&keyboard_character);
247           serial_print(buffer + cursor + 1);
248           cursor++;
249         }
250         characters_in_buffer++;
251       }
252     }
253   }
254 
```

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

5.4.2.3 serial_print()

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

Definition at line 56 of file serial.c.

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

5.4.2.4 serial_println()

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

Definition at line 40 of file serial.c.

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

5.4.2.5 set_serial_in()

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

Definition at line 86 of file serial.c.

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

5.4.2.6 set_serial_out()

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

Definition at line 74 of file serial.c.

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

5.5 include/core/tables.h File Reference

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

Classes

- struct idt_entry_struct
- struct idt_struct
- struct gdt_descriptor_struct
- struct gdt_entry_struct

Functions

- struct idt_entry_struct __attribute__ ((packed)) idt_entry
- void idt_set_gate (u8int idx, u32int base, u16int sel, u8int flags)
- void gdt_init_entry (int idx, u32int base, u32int limit, u8int access, u8int flags)
- void init_idt ()
- void init_gdt ()

Variables

- u16int base_low
- u16int sselect
- u8int zero
- u8int flags
- u16int base_high
- u16int limit
- u32int base
- u16int limit_low
- u8int base_mid
- u8int access

5.5.1 Function Documentation

5.5.1.1 __attribute__()

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

5.5.1.2 gdt_init_entry()

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

Definition at line 57 of file tables.c.

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

5.5.1.3 idt_set_gate()

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

Definition at line 27 of file tables.c.

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

5.5.1.4 init_gdt()

```
void init_gdt ( )
```

Definition at line 75 of file tables.c.

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

5.5.1.5 init_idt()

```
void init_idt ( )
```

Definition at line 43 of file tables.c.

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

5.5.2 Variable Documentation

5.5.2.1 access

```
u8int access
```

Definition at line 3 of file tables.h.

5.5.2.2 base

```
u32int base
```

Definition at line 1 of file tables.h.

5.5.2.3 base_high

```
u8int base_high
```

Definition at line 4 of file tables.h.

5.5.2.4 base_low

```
u16int base_low
```

Definition at line 0 of file tables.h.

5.5.2.5 base_mid

```
u8int base_mid
```

Definition at line 2 of file tables.h.

5.5.2.6 flags

```
u8int flags
```

Definition at line 3 of file tables.h.

5.5.2.7 limit

```
u16int limit
```

Definition at line 0 of file tables.h.

5.5.2.8 limit_low

```
u16int limit_low
```

Definition at line 0 of file tables.h.

5.5.2.9 sselect

```
u16int sselect
```

Definition at line 1 of file tables.h.

5.5.2.10 zero

```
u8int zero
```

Definition at line 2 of file tables.h.

5.6 include/mem/heap.h File Reference

Classes

- struct header
- struct footer
- struct index_entry
- struct index_table
- struct heap

Macros

- #define TABLE_SIZE 0x1000
- #define KHEAP_BASE 0xD000000
- #define KHEAP_MIN 0x10000
- #define KHEAP_SIZE 0x1000000

Functions

- u32int _kmalloc (u32int size, int align, u32int *phys_addr)
- u32int kmalloc (u32int size)
- u32int kfree ()
- void init_kheap ()
- u32int alloc (u32int size, heap *hp, int align)
- heap * make_heap (u32int base, u32int max, u32int min)

5.6.1 Macro Definition Documentation

5.6.1.1 KHEAP_BASE

```
#define KHEAP_BASE 0xD000000
```

Definition at line 6 of file heap.h.

5.6.1.2 KHEAP_MIN

```
#define KHEAP_MIN 0x10000
```

Definition at line 7 of file heap.h.

5.6.1.3 KHEAP_SIZE

```
#define KHEAP_SIZE 0x1000000
```

Definition at line 8 of file heap.h.

5.6.1.4 TABLE_SIZE

```
#define TABLE_SIZE 0x1000
```

Definition at line 5 of file heap.h.

5.6.2 Function Documentation

5.6.2.1 _kmalloc()

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

Definition at line 24 of file heap.c.

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

5.6.2.2 alloc()

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

Definition at line 57 of file heap.c.

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

5.6.2.3 init_kheap()

```
void init_kheap ( )
```

5.6.2.4 kfree()

```
u32int kfree ( )
```

5.6.2.5 kmalloc()

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

Definition at line 52 of file heap.c.

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

5.6.2.6 make_heap()

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

Definition at line 71 of file heap.c.

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

5.7 include/mem/paging.h File Reference

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

Classes

- struct page_entry
- struct page_table
- struct page_dir

Macros

- #define PAGE_SIZE 0x1000

Functions

- void set_bit (u32int addr)
- void clear_bit (u32int addr)
- u32int get_bit (u32int addr)
- u32int first_free ()
- void init_paging ()
- void load_page_dir (page_dir *new_page_dir)
- page_entry * get_page (u32int addr, page_dir *dir, int make_table)
- void new_frame (page_entry *page)

5.7.1 Macro Definition Documentation

5.7.1.1 PAGE_SIZE

```
#define PAGE_SIZE 0x1000
```

Definition at line 6 of file paging.h.

5.7.2 Function Documentation

5.7.2.1 clear_bit()

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

Definition at line 44 of file paging.c.

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

5.7.2.2 first_free()

```
u32int first_free ( )
```

5.7.2.3 get_bit()

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

Definition at line 56 of file paging.c.

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

5.7.2.4 get_page()

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

Definition at line 85 of file paging.c.

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

5.7.2.5 init_paging()

```
void init_paging ( )
```

Definition at line 111 of file paging.c.

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

5.7.2.6 load_page_dir()

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

Definition at line 158 of file paging.c.

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

5.7.2.7 new_frame()

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

Definition at line 173 of file paging.c.

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

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

5.7.2.8 set_bit()

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

Definition at line 32 of file paging.c.

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

5.8 include/string.h File Reference

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

Functions

- int isspace (const char *c)
- void * memset (void *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.

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

5.8.1.2 isspace()

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

Definition at line 119 of file string.c.

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

5.8.1.3 memset()

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

Definition at line 137 of file string.c.

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

5.8.1.4 strcat()

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

Definition at line 106 of file string.c.

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

5.8.1.5 strcmp()

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

Definition at line 79 of file string.c.

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

5.8.1.6 strcpy()

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

Definition at line 36 of file string.c.

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

5.8.1.7 strlen()

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

Definition at line 24 of file string.c.

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

5.8.1.8 strtok()

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

Definition at line 151 of file string.c.

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

5.9 include/system.h File Reference

Classes

- struct date_time

Macros

- #define NULL 0
- #define no_warn(p) if (p) while (1) break
- #define asm __asm__
- #define volatile __volatile__
- #define sti() asm volatile ("sti"::)

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

Typedefs

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

Functions

- void klogv (const char *msg)
- void kpanic (const char *msg)

5.9.1 Macro Definition Documentation

5.9.1.1 asm

```
#define asm __asm__
```

Definition at line 11 of file system.h.

5.9.1.2 cli

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

Definition at line 15 of file system.h.

5.9.1.3 GDT_CS_ID

```
#define GDT_CS_ID 0x01
```

Definition at line 20 of file system.h.

5.9.1.4 GDT_DS_ID

```
#define GDT_DS_ID 0x02
```

Definition at line 21 of file system.h.

5.9.1.5 hlt

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

Definition at line 17 of file system.h.

5.9.1.6 iret

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

Definition at line 18 of file system.h.

5.9.1.7 no_warn

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

Definition at line 7 of file system.h.

5.9.1.8 nop

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

Definition at line 16 of file system.h.

5.9.1.9 NULL

```
#define NULL 0
```

Definition at line 4 of file system.h.

5.9.1.10 sti

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

Definition at line 14 of file system.h.

5.9.1.11 volatile

```
#define volatile __volatile__
```

Definition at line 12 of file system.h.

5.9.2 Typedef Documentation

5.9.2.1 size_t

```
typedef unsigned int size_t
```

Definition at line 24 of file system.h.

5.9.2.2 u16int

```
typedef unsigned short u16int
```

Definition at line 26 of file system.h.

5.9.2.3 u32int

```
typedef unsigned long u32int
```

Definition at line 27 of file system.h.

5.9.2.4 u8int

```
typedef unsigned char u8int
```

Definition at line 25 of file system.h.

5.9.3 Function Documentation

5.9.3.1 klogv()

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

Definition at line 11 of file system.c.

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

5.9.3.2 kpanic()

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

Definition at line 24 of file system.c.

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

5.10 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 sys_call_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 153 of file interrupts.c.

```
154 {
155   kpanic("Bounds error");
156 }
```

5.10.2.9 do_breakpoint()

```
void do_breakpoint ( )
```

Definition at line 145 of file interrupts.c.

```
146 {
147   kpanic("Breakpoint");
148 }
```

5.10.2.10 do_coprocessor()

```
void do_coprocessor ( )
```

Definition at line 197 of file interrupts.c.

```
198 {
199   kpanic("Coprocessor error");
200 }
```

5.10.2.11 do_coprocessor_segment()

```
void do_coprocessor_segment ( )
```

Definition at line 169 of file interrupts.c.

```
170 {
171   kpanic("Coprocessor segment error");
172 }
```

5.10.2.12 do_debug()

```
void do_debug ( )
```

Definition at line 137 of file interrupts.c.

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

5.10.2.13 do_device_not_available()

```
void do_device_not_available ( )
```

Definition at line 161 of file interrupts.c.

```
162 {
163   kpanic("Device not available");
164 }
```

5.10.2.14 do_divide_error()

```
void do_divide_error ( )
```

Definition at line 133 of file interrupts.c.

```
134 {
135   kpanic("Division-by-zero");
136 }
```

5.10.2.15 do_double_fault()

```
void do_double_fault ( )
```

Definition at line 165 of file interrupts.c.

```
166 {
167   kpanic("Double fault");
168 }
```

5.10.2.16 do_general_protection()

```
void do_general_protection ( )
```

Definition at line 185 of file interrupts.c.

```
186 {
187   kpanic("General protection fault");
188 }
```

5.10.2.17 do_invalid_op()

```
void do_invalid_op ( )
```

Definition at line 157 of file interrupts.c.

```
158 {
159   kpanic("Invalid operation");
160 }
```

5.10.2.18 do_invalid_tss()

```
void do_invalid_tss ( )
```

Definition at line 173 of file interrupts.c.

```
174 {
175   kpanic("Invalid TSS");
176 }
```

5.10.2.19 do_isr()

```
void do_isr ( )
```

Definition at line 54 of file interrupts.c.

```
55 {
56   char in = inb(COM2);
57   serial_print(&in);
58   serial_println("here");
59   outb(0x20, 0x20); //EOI
60 }
```

5.10.2.20 do_nmi()

```
void do_nmi ( )
```

Definition at line 141 of file interrupts.c.

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

5.10.2.21 do_overflow()

```
void do_overflow ( )
```

Definition at line 149 of file interrupts.c.

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

5.10.2.22 do_page_fault()

```
void do_page_fault ( )
```

Definition at line 189 of file interrupts.c.

```
190 {
191   kpanic("Page Fault");
192 }
```

5.10.2.23 do_reserved()

```
void do_reserved ( )
```

Definition at line 193 of file interrupts.c.

```
194 {
195   serial_println("die: reserved");
196 }
```

5.10.2.24 do_segment_not_present()

```
void do_segment_not_present ( )
```

Definition at line 177 of file interrupts.c.

```
178 {
179   kpanic("Segment not present");
180 }
```

5.10.2.25 do_stack_segment()

```
void do_stack_segment ( )
```

Definition at line 181 of file interrupts.c.

```
182 {
183   kpanic("Stack segment error");
184 }
```

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 67 of file interrupts.c.

```
68 {
69   int i;
70
71   // Necessary interrupt handlers for protected mode
72   u32int isrs[17] = {
73       (u32int)divide_error,
74       (u32int)debug,
75       (u32int)nmi,
76       (u32int)breakpoint,
77       (u32int)overflow,
78       (u32int)bounds,
79       (u32int)invalid_op,
80       (u32int)device_not_available,
81       (u32int)double_fault,
82       (u32int)coprocessor_segment,
83       (u32int)invalid_tss,
84       (u32int)segment_not_present,
85       (u32int)stack_segment,
86       (u32int)general_protection,
87       (u32int)page_fault,
```

```
88        (u32int)reserved,
89        (u32int)coprocessor};
90
91    // Install handlers; 0x08=sel, 0x8e=flags
92    for (i = 0; i < 32; i++)
93    {
94      if (i < 17)
95        idt_set_gate(i, isrs[i], 0x08, 0x8e);
96      else
97        idt_set_gate(i, (u32int)reserved, 0x08, 0x8e);
98    }
99    // Ignore interrupts from the real time clock
100    idt_set_gate(0x08, (u32int)rtc_isr, 0x08, 0x8e);
101    idt_set_gate(60, (u32int)sys_call_isr, 0x08, 0x8e);
102 }
```

5.10.2.29 init_pic()

```
void init_pic (
              void  )
```

Definition at line 110 of file interrupts.c.

```
111 {
112    outb(PIC1, ICW1); //send initialization code words 1 to PIC1
113    io_wait();
114    outb(PIC2, ICW1); //send icw1 to PIC2
115    io_wait();
116    outb(PIC1 + 1, 0x20); //icw2: remap irq0 to 32
117    io_wait();
118    outb(PIC2 + 1, 0x28); //icw2: remap irq8 to 40
119    io_wait();
120    outb(PIC1 + 1, 4); //icw3
121    io_wait();
122    outb(PIC2 + 1, 2); //icw3
123    io_wait();
124    outb(PIC1 + 1, ICW4); //icw4: 80x86, automatic handling
125    io_wait();
126    outb(PIC2 + 1, ICW4); //icw4: 80x86, automatic handling
127    io_wait();
128    outb(PIC1 + 1, 0xFF); //disable irqs for PIC1
129    io_wait();
130    outb(PIC2 + 1, 0xFF); //disable irqs for PIC2
131 }
```

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 ( )
```

5.10.2.40 sys_call_isr()

```
void sys_call_isr ( )
```

5.10.3 Variable Documentation

5.10.3.1 idt_entries

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

Definition at line 17 of file tables.c.

5.11 kernel/core/kmain.c File Reference

```
#include <stdint.h>
#include <string.h>
#include <system.h>
#include <core/io.h>
#include <core/serial.h>
#include <core/tables.h>
#include <core/interrupts.h>
#include <mem/heap.h>
#include <mem/paging.h>
#include "modules/mpx_supt.h"
#include "modules/R1/commhand.h"
#include "modules/R2/R2commands.h"
#include "modules/R2/R2_Internal_Functions_And_Structures.h"
#include "modules/R3/R3commands.h"
#include "modules/R4/R4commands.h"
```

Functions

- void kmain (void)

5.11.1 Function Documentation

5.11.1.1 kmain()

```
void kmain (
            void  )
```

Definition at line 31 of file kmain.c.

```
32 {
33     // extern uint32_t magic;
34     // Uncomment if you want to access the multiboot header
35     // extern void *mbd;
36     // char *boot_loader_name = (char*)((long*)mbd)[16];
37 
38     // 0) Initialize Serial I/O
39     // functions to initialize serial I/O can be found in serial.c
```

```
40    // there are 3 functions to call
41
42    init_serial(COM1);
43    set_serial_in(COM1);
44    set_serial_out(COM1);
45
46    klogv("Starting MPX boot sequence...");
47    klogv("Initialized serial I/O on COM1 device...");
48
49    // 1) Initialize the support software by identifying the current
50    //     MPX Module.  This will change with each module.
51    // you will need to call mpx_init from the mpx_supt.c
52
53    mpx_init(MODULE_R4);
54
55    // 2) Check that the boot was successful and correct when using grub
56    // Comment this when booting the kernel directly using QEMU, etc.
57    //if ( magic != 0x2BADB002 ){
58    //  kpanic("Boot was not error free. Halting.");
59    //}
60
61    // 3) Descriptor Tables -- tables.c
62    //  you will need to initialize the global
63    // this keeps track of allocated segments and pages
64    klogv("Initializing descriptor tables...");
65
66    init_gdt();
67    init_idt();
68
69    init_pic();
70    sti();
71
72    // 4)  Interrupt vector table --  tables.c
73    // this creates and initializes a default interrupt vector table
74    // this function is in tables.c
75
76    init_irq();
77
78    klogv("Interrupt vector table initialized!");
79
80    // 5) Virtual Memory -- paging.c  -- init_paging
81    //  this function creates the kernel's heap
82    //  from which memory will be allocated when the program calls
83    // sys_alloc_mem UNTIL the memory management module  is completed
84    // this allocates memory using discrete "pages" of physical memory
85    // NOTE:  You will only have about 70000 bytes of dynamic memory
86    //
87    klogv("Initializing virtual memory...");
88
89    init_paging();
90
91    // 6) Call YOUR command handler -  interface method
92    klogv("Transferring control to commhand...");
93    //commhand(); //Removed for R4
94
95    allocateQueues();
96    //allocateAlarms();
97
98    createPCB("Commhand", 's', 9);
99    PCB *new_pcb = findPCB("Commhand");
100    context *cp = (context *)(new_pcb->stackTop);
101    memset(cp, 0, sizeof(context));
102    cp->fs = 0x10;
103    cp->gs = 0x10;
104    cp->ds = 0x10;
105    cp->es = 0x10;
106    cp->cs = 0x8;
107    cp->ebp = (u32int)(new_pcb->stack);
108    cp->esp = (u32int)(new_pcb->stackTop);
109    cp->eip = (u32int)commhand; // The function correlating to the process, ie. Proc1
110    cp->eflags = 0x202;
111
112    // createPCB("Alarm", 'a', 1);
113    // PCB *AlarmPCB = findPCB("Alarm");
114    // context *cpAlarm = (context *)(AlarmPCB->stackTop);
115    // memset(cpAlarm, 0, sizeof(context));
116    // cpAlarm->fs = 0x10;
117    // cpAlarm->gs = 0x10;
118    // cpAlarm->ds = 0x10;
119    // cpAlarm->es = 0x10;
120    // cpAlarm->cs = 0x8;
121    // cpAlarm->ebp = (u32int)(AlarmPCB->stack);
122    // cpAlarm->esp = (u32int)(AlarmPCB->stackTop);
123    // cpAlarm->eip = (u32int)alarmPCB; // The function correlating to the process, ie. Proc1
124    // cpAlarm->eflags = 0x202;
125
126    createPCB("Idle", 's', 0);
```

```
127     PCB *idlePCB = findPCB("Idle");
128     context *cpIDLE = (context *)(idlePCB->stackTop);
129     memset(cpIDLE, 0, sizeof(context));
130     cpIDLE->fs = 0x10;
131     cpIDLE->gs = 0x10;
132     cpIDLE->ds = 0x10;
133     cpIDLE->es = 0x10;
134     cpIDLE->cs = 0x8;
135     cpIDLE->ebp = (u32int)(idlePCB->stack);
136     cpIDLE->esp = (u32int)(idlePCB->stackTop);
137     cpIDLE->eip = (u32int)idle; // The function correlating to the process, ie. Proc1
138     cpIDLE->eflags = 0x202;
139 
140     asm volatile("int $60");
141 
142     // 7) System Shutdown on return from your command handler
143 
144     klogv("Starting system shutdown procedure...");
145 
146     /* Shutdown Procedure */
147     klogv("Shutdown complete. You may now turn off the machine. (QEMU: C-a x)");
148     hlt();
149 }
```

5.12 kernel/core/serial.c File Reference

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

Macros

- #define NO_ERROR 0

Functions

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

Variables

- int serial_port_out = 0
- int serial_port_in = 0

5.12.1 Macro Definition Documentation

5.12.1.1 NO_ERROR

```
#define NO_ERROR 0
```

Definition at line 12 of file serial.c.

5.12.2 Function Documentation

5.12.2.1 init_serial()

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

Definition at line 22 of file serial.c.

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

5.12.2.2 polling()

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

Definition at line 92 of file serial.c.

```
93 {
94   // insert your code to gather keyboard input via the technique of polling.
95
96   char keyboard_character;
97
98   int cursor = 0;
99
100   char log[] = {'\0', '\0', '\0', '\0'};
101
102   int characters_in_buffer = 0;
103
104   while (1)
105   {
106
107     if (inb(COM1 + 5) & 1)
108     {                                  // is there input char?
109       keyboard_character = inb(COM1); //read the char from COM1
110
111       if (keyboard_character == '\n' || keyboard_character == '\r')
112       { // HANDLEING THE CARRIAGE RETURN AND NEW LINE CHARACTERS
113
114         buffer[characters_in_buffer] = '\0';
115         break;
116       }
117       else if ((keyboard_character == 127 || keyboard_character == 8) && cursor > 0)
118       { // HANDELING THE BACKSPACE CHARACTER
119
120         //serial_println("Handleing backspace character.");
121         serial_print("\033[K");
```

```
122
123         buffer[cursor - 1] = '\0';
124         serial_print("\b \b");
125         serial_print(buffer + cursor);
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];
133           buffer[temp_cursor + 1] = '\0';
134           temp_cursor++;
135         }
136
137         characters_in_buffer--;
138         cursor = characters_in_buffer;
139       }
140       else if (keyboard_character == '~' && cursor < 99)
141       { //HANDLING THE DELETE KEY
142         // \033[3~
143
144         serial_print("\033[K");
145
146         buffer[cursor + 1] = '\0';
147         serial_print("\b \b");
148         serial_print(buffer + cursor);
149
150         int temp_cursor = cursor + 1;
151
152         while (buffer[temp_cursor + 1] != '\0')
153         {
154           buffer[temp_cursor] = buffer[temp_cursor + 1];
155           buffer[temp_cursor + 1] = '\0';
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       }
172       else if (log[0] == '\033' && log[1] == '[')
173       { // HANDLEING LAST CHARACTER FOR ARROW KEYS
174         log[2] = keyboard_character;
175
176         if (keyboard_character == 'A')
177         { //Up arrow
178           //Call a history function from the commhand or do nothing
179         }
180         else if (keyboard_character == 'B')
181         { //Down arrow
182           //Call a history command from the commhand or do nothing
183         }
184         else if (keyboard_character == 'C' && cursor != 99)
185         { //Right arrow
186
187           serial_print("\033[C");
188           cursor++;
189         }
190         else if (keyboard_character == 'D' && cursor != 0)
191         { //Left arrow
192
193           serial_print("\033[D");
194           cursor--;
195         }
196
197         memset(log, '\0', 4);
198       }
199       else
200       {
201
202         if (cursor == 0 && buffer[cursor] == '\0') //Adding character at beginning of buffer
203         {
204           buffer[cursor] = keyboard_character;
205           serial_print(buffer + cursor);
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(buffer + cursor);
212           cursor++;
213         }
214         else //Inserting character to the middle of the buffer
215         {
216           char temp_buffer[strlen(buffer)];
217           memset(temp_buffer, '\0', strlen(buffer));
218 
219           int temp_cursor = 0;
220           while (temp_cursor <= characters_in_buffer) //Filling the temp_buffer with all of the
      characters from buffer, and inserting the new character.
221           {
222             if (temp_cursor < cursor)
223             {
224               temp_buffer[temp_cursor] = buffer[temp_cursor];
225             }
226             else if (temp_cursor > cursor)
227             {
228               temp_buffer[temp_cursor] = buffer[temp_cursor - 1];
229             }
230             else
231             { //temp_cursor == cursor
232               temp_buffer[temp_cursor] = keyboard_character;
233             }
234             temp_cursor++;
235           }
236 
237           temp_cursor = 0;
238           int temp_buffer_size = strlen(temp_buffer);
239           while (temp_cursor <= temp_buffer_size) //Setting the contents of the buffer equal to the
      temp_buffer.
240           {
241             buffer[temp_cursor] = temp_buffer[temp_cursor];
242             temp_cursor++;
243           }
244 
245           serial_print("\033[K");
246           serial_print(&keyboard_character);
247           serial_print(buffer + cursor + 1);
248           cursor++;
249         }
250         characters_in_buffer++;
251       }
252     }
253   }
254 
255   *count = characters_in_buffer; // buffer count
256 
257   return count;
258 }
```

5.12.2.3 serial_print()

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

Definition at line 56 of file serial.c.

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

5.12.2.4 serial_println()

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

Definition at line 40 of file serial.c.

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

5.12.2.5 set_serial_in()

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

Definition at line 86 of file serial.c.

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

5.12.2.6 set_serial_out()

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

Definition at line 74 of file serial.c.

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

5.12.3 Variable Documentation

5.12.3.1 serial_port_in

```
int serial_port_in = 0
```

Definition at line 16 of file serial.c.

5.12.3.2 serial_port_out

```
int serial_port_out = 0
```

Definition at line 15 of file serial.c.

5.13 kernel/core/system.c File Reference

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

Functions

- void klogv (const char *msg)
- void kpanic (const char *msg)

5.13.1 Function Documentation

5.13.1.1 klogv()

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

Definition at line 11 of file system.c.

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

5.13.1.2 kpanic()

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

Definition at line 24 of file system.c.

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

5.14 kernel/core/tables.c File Reference

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

Functions

- void write_gdt_ptr (u32int, size_t)
- void write_idt_ptr (u32int)
- void idt_set_gate (u8int idx, u32int base, u16int sel, u8int flags)
- void init_idt ()
- void gdt_init_entry (int idx, u32int base, u32int limit, u8int access, u8int flags)
- void init_gdt ()

Variables

- gdt_descriptor gdt_ptr
- gdt_entry gdt_entries [5]
- idt_descriptor idt_ptr
- idt_entry idt_entries [256]

5.14.1 Function Documentation

5.14.1.1 gdt_init_entry()

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

Definition at line 57 of file tables.c.

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

5.14.1.2 idt_set_gate()

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

Definition at line 27 of file tables.c.

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

5.14.1.3 init_gdt()

```
void init_gdt ( )
```

Definition at line 75 of file tables.c.

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

5.14.1.4 init_idt()

```
void init_idt ( )
```

Definition at line 43 of file tables.c.

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

5.14.1.5 write_gdt_ptr()

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

5.14.1.6 write_idt_ptr()

```
void write_idt_ptr (
             u32int  )
```

5.14.2 Variable Documentation

5.14.2.1 gdt_entries

```
gdt_entry gdt_entries[5]
```

Definition at line 13 of file tables.c.

5.14.2.2 gdt_ptr

```
gdt_descriptor gdt_ptr
```

Definition at line 12 of file tables.c.

5.14.2.3 idt_entries

```
idt_entry idt_entries[256]
```

Definition at line 17 of file tables.c.

5.14.2.4 idt_ptr

```
idt_descriptor idt_ptr
```

Definition at line 16 of file tables.c.

5.15 kernel/mem/heap.c File Reference

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

Functions

- u32int _kmalloc (u32int size, int page_align, u32int *phys_addr)
- u32int kmalloc (u32int size)
- u32int alloc (u32int size, heap *h, int align)
- heap * make_heap (u32int base, u32int max, u32int min)

Variables

- heap * kheap = 0
- heap * curr_heap = 0
- page_dir * kdir
- void * end
- void _end
- void __end
- u32int phys_alloc_addr = (u32int)&end

5.15.1 Function Documentation

5.15.1.1 _kmalloc()

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

Definition at line 24 of file heap.c.

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

5.15.1.2 alloc()

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

Definition at line 57 of file heap.c.

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

5.15.1.3 kmalloc()

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

Definition at line 52 of file heap.c.

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

5.15.1.4 make_heap()

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

Definition at line 71 of file heap.c.

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

5.15.2 Variable Documentation

5.15.2.1 __end

```
void __end
```

Definition at line 18 of file heap.c.

5.15.2.2 _end

```
void _end
```

Definition at line 18 of file heap.c.

5.15.2.3 curr_heap

```
heap* curr_heap = 0
```

Definition at line 15 of file heap.c.

5.15.2.4 end

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

5.15.2.5 kdir

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

Definition at line 21 of file paging.c.

5.15.2.6 kheap

```
heap* kheap = 0
```

Definition at line 14 of file heap.c.

5.15.2.7 phys_alloc_addr

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

Definition at line 22 of file heap.c.

5.16 kernel/mem/paging.c File Reference

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

Functions

- void set_bit (u32int addr)
- void clear_bit (u32int addr)
- u32int get_bit (u32int addr)
- u32int find_free ()
- page_entry * get_page (u32int addr, page_dir *dir, int make_table)
- void init_paging ()
- void load_page_dir (page_dir *new_dir)
- void new_frame (page_entry *page)

Variables

- u32int mem_size = 0x4000000
- u32int page_size = 0x1000
- u32int nframes
- u32int * frames
- page_dir * kdir = 0
- page_dir * 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] != 0xFFFFFFFF) //if frame not full
73       for (j=0; j<32; j++) //find first free bit
74     if (!(frames[i] & (1 « j)))
75       return i*32+j;
76
77   return -1; //no free frames
78 }
```

5.16.1.3 get_bit()

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

Definition at line 56 of file paging.c.

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

5.16.1.4 get_page()

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

Definition at line 85 of file paging.c.

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

5.16.1.5 init_paging()

```
void init_paging ( )
```

Definition at line 111 of file paging.c.

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

5.16.1.6 load_page_dir()

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

Definition at line 158 of file paging.c.

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

5.16.1.7 new_frame()

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

Definition at line 173 of file paging.c.

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

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

5.16.1.8 set_bit()

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

Definition at line 32 of file paging.c.

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

5.16.2 Variable Documentation

5.16.2.1 cdir

```
page_dir* cdir = 0
```

Definition at line 22 of file paging.c.

5.16.2.2 frames

```
u32int* frames
```

Definition at line 19 of file paging.c.

5.16.2.3 kdir

```
page_dir* kdir = 0
```

Definition at line 21 of file paging.c.

5.16.2.4 kheap

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

Definition at line 14 of file heap.c.

5.16.2.5 mem_size

```
u32int mem_size = 0x4000000
```

Definition at line 15 of file paging.c.

5.16.2.6 nframes

```
u32int nframes
```

Definition at line 18 of file paging.c.

5.16.2.7 page_size

```
u32int page_size = 0x1000
```

Definition at line 16 of file paging.c.

5.16.2.8 phys_alloc_addr

```
u32int phys_alloc_addr  [extern]
```

Definition at line 22 of file heap.c.

5.17 lib/string.c File Reference

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

Functions

- int strlen (const char *s)
- char * strcpy (char *s1, const char *s2)
- int atoi (const char *s)
- int strcmp (const char *s1, const char *s2)
- char * strcat (char *s1, const char *s2)
- int isspace (const char *c)
- void * memset (void *s, int c, size_t n)
- char * strtok (char *s1, const char *s2)

5.17.1 Function Documentation

5.17.1.1 atoi()

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

Definition at line 48 of file string.c.

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

5.17.1.2 isspace()

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

Definition at line 119 of file string.c.

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

5.17.1.3 memset()

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

Definition at line 137 of file string.c.

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

5.17.1.4 strcat()

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

Definition at line 106 of file string.c.

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

5.17.1.5 strcmp()

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

Definition at line 79 of file string.c.

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

5.17.1.6 strcpy()

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

Definition at line 36 of file string.c.

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

5.17.1.7 strlen()

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

Definition at line 24 of file string.c.

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

5.17.1.8 strtok()

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

Definition at line 151 of file string.c.

```
152 {
153   static char *tok_tmp = NULL;
154   const char *p = s2;
155
156   //new string
157   if (s1!=NULL){
158     tok_tmp = s1;
159   }
160   //old string cont'd
161   else {
162     if (tok_tmp==NULL){
163       return NULL;
164     }
165     s1 = tok_tmp;
166   }
167
168   //skip leading s2 characters
169   while ( *p && *s1 ){
170     if (*s1==*p){
171       ++s1;
172       p = s2;
173       continue;
174     }
175     ++p;
176   }
177
178   //no more to parse
179   if (!*s1){
180     return (tok_tmp = NULL);
181   }
182
183   //skip non-s2 characters
184   tok_tmp = s1;
```

```
185   while (*tok_tmp){
186     p = s2;
187     while (*p){
188       if (*tok_tmp==*p++){
189     *tok_tmp++ = '\0';
190     return s1;
191       }
192     }
193     ++tok_tmp;
194   }
195
196   //end of string
197   tok_tmp = NULL;
198   return s1;
199 }
```

5.18 modules/mpx_supt.c File Reference

```
#include "mpx_supt.h"
#include <mem/heap.h>
#include <string.h>
#include <core/serial.h>
#include "R2/R2commands.h"
#include "R2/R2_Internal_Functions_And_Structures.h"
#include "R3/R3commands.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(*func)(u32int))
- void sys_set_free (int(*func)(void *))
- void * sys_alloc_mem (u32int size)
- int sys_free_mem (void *ptr)
- void idle ()
- u32int * sys_call (context *registers)

Variables

- param params
- int current_module = -1
- u32int(* student_malloc)(u32int)
- int(* student_free)(void *)
- PCB * COP
- context * callerContext

5.18.1 Function Documentation

5.18.1.1 idle()

```
void idle ( )
```

Definition at line 178 of file mpx_supt.c.

```
179 {
180   char msg[30];
181   int count = 0;
182
183   memset(msg, '\0', sizeof(msg));
184   strcpy(msg, "IDLE PROCESS EXECUTING.\n");
185   count = strlen(msg);
186
187   while (1)
188   {
189     sys_req(WRITE, DEFAULT_DEVICE, msg, &count);
190     sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
191   }
192 }
```

5.18.1.2 mpx_init()

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

Definition at line 114 of file mpx_supt.c.

```
115 {
116
117   current_module = cur_mod;
118   if (cur_mod == MEM_MODULE)
119     mem_module_active = TRUE;
120
121   if (cur_mod == IO_MODULE)
122     io_module_active = TRUE;
123 }
```

5.18.1.3 sys_alloc_mem()

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

Definition at line 150 of file mpx_supt.c.

```
151 {
152   if (!mem_module_active)
153     return (void *)kmalloc(size);
154   else
155     return (void *)(*student_malloc)(size);
156 }
```

5.18.1.4 sys_call()

```
u32int* sys_call (
              context * registers )
```

Definition at line 196 of file mpx_supt.c.

```
197 { // Benjamin and Anastase programmed this function
198
199   PCB *tempOOP = NULL;
200   if (COP == NULL)
201   { // sys_call has not been called yet.
202
203     callerContext = registers;
204   }
205   else
206   {
207     if (params.op_code == IDLE)
208     { // Save the context (reassign COP's stack top).
209       COP->runningStatus = 0;
210       COP->stackTop = (unsigned char *)registers;
211       tempOOP = COP;
212     }
213     else if (params.op_code == EXIT)
214     { // free COP.
215       sys_free_mem(COP);
216     }
217   }
218
219   queue *ready = getReady();
220
221   if (ready->head != NULL)
222   {
223     COP = ready->head;
224     removePCB(COP);
225     COP->runningStatus = 1;
226
227     if (tempOOP != NULL)
228     {
229       insertPCB(tempOOP);
230     }
231
232     return (u32int *)COP->stackTop;
233   }
234   return (u32int *)callerContext;
235 }
```

5.18.1.5 sys_free_mem()

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

Definition at line 163 of file mpx_supt.c.

```
164 {
165   if (mem_module_active)
166     return (*student_free)(ptr);
167   // otherwise we don't free anything
168   return -1;
169 }
```

5.18.1.6 sys_req()

```
int sys_req (
              int op_code,
              int device_id,
```

```
              char * buffer_ptr,
              int * count_ptr )
```

Definition at line 50 of file mpx_supt.c.

```
55 {
56   int return_code = 0;
57
58   if (op_code == IDLE || op_code == EXIT)
59   {
60     // store the process's operation request
61     // triger interrupt 60h to invoke
62     params.op_code = op_code;
63     asm volatile("int $60");
64   } // idle or exit
65
66   else if (op_code == READ || op_code == WRITE)
67   {
68     // validate buffer pointer and count pointer
69     if (buffer_ptr == NULL)
70       return_code = INVALID_BUFFER;
71     else if (count_ptr == NULL || *count_ptr <= 0)
72       return_code = INVALID_COUNT;
73
74     // if parameters are valid store in the params structure
75     if (return_code == 0)
76     {
77       params.op_code = op_code;
78       params.device_id = device_id;
79       params.buffer_ptr = buffer_ptr;
80       params.count_ptr = count_ptr;
81
82       if (!io_module_active)
83       {
84         // if default device
85         if (op_code == READ)
86           return_code = *(polling(buffer_ptr, count_ptr));
87
88         else //must be WRITE
89           return_code = serial_print(buffer_ptr);
90       }
91       else
92       { // I/O module is implemented
93         asm volatile("int $60");
94       } // NOT IO_MODULE
95     }
96   }
97   else
98     return_code = INVALID_OPERATION;
99
100   return return_code;
101 } // end of sys_req
```

5.18.1.7 sys_set_free()

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

Definition at line 140 of file mpx_supt.c.

```
141 {
142   student_free = func;
143 }
```

5.18.1.8 sys_set_malloc()

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

Definition at line 130 of file mpx_supt.c.

```
131 {
132   student_malloc = func;
133 }
```

5.18.2 Variable Documentation

5.18.2.1 callerContext

```
context* callerContext
```

Definition at line 195 of file mpx_supt.c.

5.18.2.2 COP

```
PCB* COP
```

Definition at line 194 of file mpx_supt.c.

5.18.2.3 current_module

```
int current_module = -1
```

Definition at line 21 of file mpx_supt.c.

5.18.2.4 params

```
param params
```

Definition at line 18 of file mpx_supt.c.

5.18.2.5 student_free

```
int(* student_free) (void *)
```

Definition at line 31 of file mpx_supt.c.

5.18.2.6 student_malloc

```
u32int(* student_malloc) (u32int)
```

Definition at line 27 of file mpx_supt.c.

5.19 modules/mpx_supt.h File Reference

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

Classes

- struct param

Macros

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

Functions

- int sys_req (int op_code, int device_id, char *buffer_ptr, int *count_ptr)
- void mpx_init (int cur_mod)
- void sys_set_malloc (u32int(*func)(u32int))
- void sys_set_free (int(*func)(void *))
- void * sys_alloc_mem (u32int size)
- int sys_free_mem (void *ptr)
- void idle ()

5.19.1 Macro Definition Documentation

5.19.1.1 COM_PORT

```
#define COM_PORT 222
```

Definition at line 29 of file mpx_supt.h.

5.19.1.2 DEFAULT_DEVICE

```
#define DEFAULT_DEVICE 111
```

Definition at line 28 of file mpx_supt.h.

5.19.1.3 EXIT

```
#define EXIT 0
```

Definition at line 6 of file mpx_supt.h.

5.19.1.4 FALSE

```
#define FALSE 0
```

Definition at line 13 of file mpx_supt.h.

5.19.1.5 IDLE

```
#define IDLE 1
```

Definition at line 7 of file mpx_supt.h.

5.19.1.6 INVALID_BUFFER

```
#define INVALID_BUFFER 1000
```

Definition at line 25 of file mpx_supt.h.

5.19.1.7 INVALID_COUNT

```
#define INVALID_COUNT 2000
```

Definition at line 26 of file mpx_supt.h.

5.19.1.8 INVALID_OPERATION

```
#define INVALID_OPERATION 4
```

Definition at line 10 of file mpx_supt.h.

5.19.1.9 IO_MODULE

```
#define IO_MODULE 10
```

Definition at line 21 of file mpx_supt.h.

5.19.1.10 MEM_MODULE

```
#define MEM_MODULE 11
```

Definition at line 22 of file mpx_supt.h.

5.19.1.11 MODULE_F

```
#define MODULE_F 9
```

Definition at line 20 of file mpx_supt.h.

5.19.1.12 MODULE_R1

```
#define MODULE_R1 0
```

Definition at line 15 of file mpx_supt.h.

5.19.1.13 MODULE_R2

```
#define MODULE_R2 1
```

Definition at line 16 of file mpx_supt.h.

5.19.1.14 MODULE_R3

```
#define MODULE_R3 2
```

Definition at line 17 of file mpx_supt.h.

5.19.1.15 MODULE_R4

```
#define MODULE_R4 4
```

Definition at line 18 of file mpx_supt.h.

5.19.1.16 MODULE_R5

```
#define MODULE_R5 8
```

Definition at line 19 of file mpx_supt.h.

5.19.1.17 READ

```
#define READ 2
```

Definition at line 8 of file mpx_supt.h.

5.19.1.18 TRUE

```
#define TRUE 1
```

Definition at line 12 of file mpx_supt.h.

5.19.1.19 WRITE

```
#define WRITE 3
```

Definition at line 9 of file mpx_supt.h.

5.19.2 Function Documentation

5.19.2.1 idle()

```
void idle ( )
```

Definition at line 178 of file mpx_supt.c.

```
179 {
180   char msg[30];
181   int count = 0;
182 
183   memset(msg, '\0', sizeof(msg));
184   strcpy(msg, "IDLE PROCESS EXECUTING.\n");
185   count = strlen(msg);
186 
187   while (1)
188   {
189     sys_req(WRITE, DEFAULT_DEVICE, msg, &count);
190     sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
191   }
192 }
```

5.19.2.2 mpx_init()

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

Definition at line 114 of file mpx_supt.c.

```
115 {
116 
117   current_module = cur_mod;
118   if (cur_mod == MEM_MODULE)
119     mem_module_active = TRUE;
120 
121   if (cur_mod == IO_MODULE)
122     io_module_active = TRUE;
123 }
```

5.19.2.3 sys_alloc_mem()

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

Definition at line 150 of file mpx_supt.c.

```
151 {
152   if (!mem_module_active)
153     return (void *)kmalloc(size);
154   else
155     return (void *)(*student_malloc)(size);
156 }
```

5.19.2.4 sys_free_mem()

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

Definition at line 163 of file mpx_supt.c.

```
164 {
165   if (mem_module_active)
166     return (*student_free)(ptr);
167   // otherwise we don't free anything
168   return -1;
169 }
```

5.19.2.5 sys_req()

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

Definition at line 50 of file mpx_supt.c.

```
55 {
56   int return_code = 0;
57
58   if (op_code == IDLE || op_code == EXIT)
59   {
60     // store the process's operation request
61     // triger interrupt 60h to invoke
62     params.op_code = op_code;
63     asm volatile("int $60");
64   } // idle or exit
65
66   else if (op_code == READ || op_code == WRITE)
67   {
68     // validate buffer pointer and count pointer
69     if (buffer_ptr == NULL)
70       return_code = INVALID_BUFFER;
71     else if (count_ptr == NULL || *count_ptr <= 0)
72       return_code = INVALID_COUNT;
73
74     // if parameters are valid store in the params structure
75     if (return_code == 0)
76     {
77       params.op_code = op_code;
78       params.device_id = device_id;
79       params.buffer_ptr = buffer_ptr;
80       params.count_ptr = count_ptr;
81
82       if (!io_module_active)
83       {
84         // if default device
85         if (op_code == READ)
86           return_code = *(polling(buffer_ptr, count_ptr));
87
88         else //must be WRITE
89           return_code = serial_print(buffer_ptr);
90       }
91       else
92       { // I/O module is implemented
93         asm volatile("int $60");
94       } // NOT IO_MODULE
95     }
96   }
97   else
98     return_code = INVALID_OPERATION;
99
100   return return_code;
101 } // end of sys_req
```

5.19.2.6 sys_set_free()

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

Definition at line 140 of file mpx_supt.c.

```
141 {
142   student_free = func;
143 }
```

5.19.2.7 sys_set_malloc()

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

Definition at line 130 of file mpx_supt.c.

```
131 {
132   student_malloc = func;
133 }
```

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"
#include "../R3/R3commands.h"
#include "../R4/R4commands.h"
```

Functions

- void commhand ()

5.20.1 Function Documentation

5.20.1.1 commhand()

```
void commhand ( )
```

Definition at line 12 of file commhand.c.

```
13 {
14
15     printMessage("\nWelcome to our CS 450 Project!\nType help to see what you can do!\n\n");
16
17     char cmdBuffer[100];
18     int bufferSize;
19     char processName[20];
20     int processPriority;
21
22     int quitFlag = 0;
23
24     while (!quitFlag)
25     {
26         //get a command: cal polling fx
27
28         memset(cmdBuffer, '\0', 100);
29
30         bufferSize = 99; // reset size before each call to read
31
32         sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
33
34         printMessage("\n");
35
36         if (strcmp(cmdBuffer, "help") == 0)
37         {
38             help();
39         }
40         else if (strcmp(cmdBuffer, "version") == 0)
41         {
42             version();
43         }
44         else if (strcmp(cmdBuffer, "getDate") == 0)
45         {
46             getDate();
47         }
48         else if (strcmp(cmdBuffer, "setDate") == 0)
49         {
50             setDate();
51         }
52         else if (strcmp(cmdBuffer, "getTime") == 0)
53         {
54             getTime();
55         }
56         else if (strcmp(cmdBuffer, "setTime") == 0)
57         {
58             setTime();
59         }
60         // else if (strcmp(cmdBuffer, "createPCB") == 0)
61         // {
62         //  printMessage("Please enter a name for the PCB you wish to create. (The name can be no more
       than 20 characters)\n");
63         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
64         //  printMessage("\n");
65         //  strcpy(processName, cmdBuffer);
66         //  memset(cmdBuffer, '\0', 100);
67
68         //  printMessage("Please enter a class for the PCB you wish to create. ('a' for application or
       's' for system)\n");
69         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
70         //  printMessage("\n");
71         //  if (strcmp(cmdBuffer, "a") == 0)
72         //  {
73         //      processClass = 'a';
74         //  }
75         //  else if (strcmp(cmdBuffer, "s") == 0)
76         //  {
77         //      processClass = 's';
78         //  }
79         //  else
80         //  {
81         //      processClass = '\0';
82         //  }
83         //  memset(cmdBuffer, '\0', 100);
84
85         //  printMessage("Please enter a priority for the PCB you wish to create. (The priorities range
       from 0 to 9)\n");
86         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
87         //  printMessage("\n");
88         //  processPriority = atoi(cmdBuffer);
89
```

```
90          //  createPCB(processName, processClass, processPriority);
91          // }
92          else if (strcmp(cmdBuffer, "deletePCB") == 0)
93          {
94              printMessage("Please enter the name for the PCB you wish to delete. (The name can be no more
    than 20 characters)\n");
95              sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
96              printMessage("\n");
97              strcpy(processName, cmdBuffer);
98
99              deletePCB(processName);
100         }
101         // else if (strcmp(cmdBuffer, "blockPCB") == 0)
102         // {
103         //  printMessage("Please enter the name for the PCB you wish to block. (The name can be no more
    than 20 characters)\n");
104         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
105         //  printMessage("\n");
106         //  strcpy(processName, cmdBuffer);
107
108         //  blockPCB(processName);
109         // }
110         // else if (strcmp(cmdBuffer, "unblockPCB") == 0)
111         // {
112         //  printMessage("Please enter the name for the PCB you wish to unblock. (The name can be no
    more than 20 characters)\n");
113         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
114         //  printMessage("\n");
115         //  strcpy(processName, cmdBuffer);
116
117         //  unblockPCB(processName);
118         // }
119         else if (strcmp(cmdBuffer, "suspendPCB") == 0)
120         {
121             printMessage("Please enter the name for the PCB you wish to suspend. (The name can be no
    more than 20 characters)\n");
122             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
123             printMessage("\n");
124             strcpy(processName, cmdBuffer);
125
126             suspendPCB(processName);
127         }
128         else if (strcmp(cmdBuffer, "resumePCB") == 0)
129         {
130             printMessage("Please enter the name for the PCB you wish to resume. (The name can be no more
    than 20 characters)\n");
131             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
132             printMessage("\n");
133             strcpy(processName, cmdBuffer);
134
135             resumePCB(processName);
136         }
137         else if (strcmp(cmdBuffer, "setPCBPriority") == 0)
138         {
139             printMessage("Please enter the name for the PCB you wish to change priorities for. (The name
    can be no more than 20 characters)\n");
140             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
141             printMessage("\n");
142             strcpy(processName, cmdBuffer);
143
144             printMessage("Please enter a priority for the PCB you wish to change priorities for. (The
    priorities range from 0 to 9)\n");
145             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
146             printMessage("\n");
147             processPriority = atoi(cmdBuffer);
148
149             setPCBPriority(processName, processPriority);
150         }
151         else if (strcmp(cmdBuffer, "showPCB") == 0)
152         {
153             printMessage("Please enter the name for the PCB you wish to see. (The name can be no more
    than 20 characters)\n");
154             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
155             printMessage("\n");
156             strcpy(processName, cmdBuffer);
157
158             showPCB(processName);
159         }
160         else if (strcmp(cmdBuffer, "showReady") == 0)
161         {
162             showReady();
163         }
164         else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
165         {
166             showSuspendedReady();
167         }
168         else if (strcmp(cmdBuffer, "showSuspendedBlocked") == 0)
```

```
169         {
170             showSuspendedBlocked();
171         }
172         else if (strcmp(cmdBuffer, "showBlocked") == 0)
173         {
174             showBlocked();
175         }
176         else if (strcmp(cmdBuffer, "showAll") == 0)
177         {
178             showAll();
179         }
180         // else if (strcmp(cmdBuffer, "yield") == 0)
181         // {
182         //  yield();
183         // }
184         else if (strcmp(cmdBuffer, "loadr3") == 0)
185         {
186             loadr3();
187         }
188         else if (strcmp(cmdBuffer, "infinitePCB") == 0)
189         {
190             infinitePCB();
191         }
192         // else if (strcmp(cmdBuffer, "addAlarm") == 0)
193         // {
194         //  addAlarm();
195         // }
196         else if (strcmp(cmdBuffer, "quit") == 0)
197         {
198             quitFlag = quit();
199 
200             if (quitFlag == 1)
201             {
202                 sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
203             }
204 
205             printMessage("\n");
206         }
207         else
208         {
209             printMessage("Unrecognized Command\n");
210         }
211 
212         sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
213 
214         // process the command: take array buffer chars and make a string. Decide what the cmd wants to
    do
215         // see if quit was entered: if string == quit = 1
216     }
217 }
```

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 12 of file commhand.c.

```
13 {
14 
15     printMessage("\nWelcome to our CS 450 Project!\nType help to see what you can do!\n\n");
```

```
16
17      char cmdBuffer[100];
18      int bufferSize;
19      char processName[20];
20      int processPriority;
21
22      int quitFlag = 0;
23
24      while (!quitFlag)
25      {
26          //get a command: cal polling fx
27
28          memset(cmdBuffer, '\0', 100);
29
30          bufferSize = 99; // reset size before each call to read
31
32          sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
33
34          printMessage("\n");
35
36          if (strcmp(cmdBuffer, "help") == 0)
37          {
38              help();
39          }
40          else if (strcmp(cmdBuffer, "version") == 0)
41          {
42              version();
43          }
44          else if (strcmp(cmdBuffer, "getDate") == 0)
45          {
46              getDate();
47          }
48          else if (strcmp(cmdBuffer, "setDate") == 0)
49          {
50              setDate();
51          }
52          else if (strcmp(cmdBuffer, "getTime") == 0)
53          {
54              getTime();
55          }
56          else if (strcmp(cmdBuffer, "setTime") == 0)
57          {
58              setTime();
59          }
60          // else if (strcmp(cmdBuffer, "createPCB") == 0)
61          // {
62          //  printMessage("Please enter a name for the PCB you wish to create. (The name can be no more
        than 20 characters)\n");
63          //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
64          //  printMessage("\n");
65          //  strcpy(processName, cmdBuffer);
66          //  memset(cmdBuffer, '\0', 100);
67
68          //  printMessage("Please enter a class for the PCB you wish to create. ('a' for application or
        's' for system)\n");
69          //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
70          //  printMessage("\n");
71          //  if (strcmp(cmdBuffer, "a") == 0)
72          //  {
73          //      processClass = 'a';
74          //  }
75          //  else if (strcmp(cmdBuffer, "s") == 0)
76          //  {
77          //      processClass = 's';
78          //  }
79          //  else
80          //  {
81          //      processClass = '\0';
82          //  }
83          //  memset(cmdBuffer, '\0', 100);
84
85          //  printMessage("Please enter a priority for the PCB you wish to create. (The priorities range
        from 0 to 9)\n");
86          //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
87          //  printMessage("\n");
88          //  processPriority = atoi(cmdBuffer);
89
90          //  createPCB(processName, processClass, processPriority);
91          // }
92          else if (strcmp(cmdBuffer, "deletePCB") == 0)
93          {
94              printMessage("Please enter the name for the PCB you wish to delete. (The name can be no more
        than 20 characters)\n");
95              sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
96              printMessage("\n");
97              strcpy(processName, cmdBuffer);
98
```

```
99              deletePCB(processName);
100         }
101         // else if (strcmp(cmdBuffer, "blockPCB") == 0)
102         // {
103         //  printMessage("Please enter the name for the PCB you wish to block. (The name can be no more
        than 20 characters)\n");
104         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
105         //  printMessage("\n");
106         //  strcpy(processName, cmdBuffer);
107 
108         //  blockPCB(processName);
109         // }
110         // else if (strcmp(cmdBuffer, "unblockPCB") == 0)
111         // {
112         //  printMessage("Please enter the name for the PCB you wish to unblock. (The name can be no
        more than 20 characters)\n");
113         //  sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
114         //  printMessage("\n");
115         //  strcpy(processName, cmdBuffer);
116 
117         //  unblockPCB(processName);
118         // }
119         else if (strcmp(cmdBuffer, "suspendPCB") == 0)
120         {
121             printMessage("Please enter the name for the PCB you wish to suspend. (The name can be no
        more than 20 characters)\n");
122             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
123             printMessage("\n");
124             strcpy(processName, cmdBuffer);
125 
126             suspendPCB(processName);
127         }
128         else if (strcmp(cmdBuffer, "resumePCB") == 0)
129         {
130             printMessage("Please enter the name for the PCB you wish to resume. (The name can be no more
        than 20 characters)\n");
131             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
132             printMessage("\n");
133             strcpy(processName, cmdBuffer);
134 
135             resumePCB(processName);
136         }
137         else if (strcmp(cmdBuffer, "setPCBPriority") == 0)
138         {
139             printMessage("Please enter the name for the PCB you wish to change priorities for. (The name
        can be no more than 20 characters)\n");
140             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
141             printMessage("\n");
142             strcpy(processName, cmdBuffer);
143 
144             printMessage("Please enter a priority for the PCB you wish to change priorities for. (The
        priorities range from 0 to 9)\n");
145             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
146             printMessage("\n");
147             processPriority = atoi(cmdBuffer);
148 
149             setPCBPriority(processName, processPriority);
150         }
151         else if (strcmp(cmdBuffer, "showPCB") == 0)
152         {
153             printMessage("Please enter the name for the PCB you wish to see. (The name can be no more
        than 20 characters)\n");
154             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
155             printMessage("\n");
156             strcpy(processName, cmdBuffer);
157 
158             showPCB(processName);
159         }
160         else if (strcmp(cmdBuffer, "showReady") == 0)
161         {
162             showReady();
163         }
164         else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
165         {
166             showSuspendedReady();
167         }
168         else if (strcmp(cmdBuffer, "showSuspendedBlocked") == 0)
169         {
170             showSuspendedBlocked();
171         }
172         else if (strcmp(cmdBuffer, "showBlocked") == 0)
173         {
174             showBlocked();
175         }
176         else if (strcmp(cmdBuffer, "showAll") == 0)
177         {
178             showAll();
```

```
179         }
180         // else if (strcmp(cmdBuffer, "yield") == 0)
181         // {
182         //  yield();
183         // }
184         else if (strcmp(cmdBuffer, "loadr3") == 0)
185         {
186             loadr3();
187         }
188         else if (strcmp(cmdBuffer, "infinitePCB") == 0)
189         {
190             infinitePCB();
191         }
192         // else if (strcmp(cmdBuffer, "addAlarm") == 0)
193         // {
194         //  addAlarm();
195         // }
196         else if (strcmp(cmdBuffer, "quit") == 0)
197         {
198             quitFlag = quit();
199 
200             if (quitFlag == 1)
201             {
202                 sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
203             }
204 
205             printMessage("\n");
206         }
207         else
208         {
209             printMessage("Unrecognized Command\n");
210         }
211 
212         sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
213 
214         // process the command: take array buffer chars and make a string. Decide what the cmd wants to
    do
215         // see if quit was entered: if string == quit = 1
216     }
217 }
```

5.22 modules/R1/R1commands.c File Reference

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

Functions

- int BCDtoChar (unsigned char test, char *buffer)
- unsigned char intToBCD (int test)
- void printMessage (char *str)
- void help ()
- int version ()
- void getTime ()
- int setTime ()
- void getDate ()
- int setDate ()
- void deleteQueue (queue *queue)
- void removeAll ()
- int quit ()

5.22.1 Function Documentation

5.22.1.1 BCDtoChar()

```
int BCDtoChar (
            unsigned char test,
            char * buffer )
```

Definition at line 376 of file R1commands.c.

```
377 {
378
379     int val1 = (test / 16);
380     int val2 = (test % 16);
381
382     buffer[0] = val1 + '0';
383     buffer[1] = val2 + '0';
384
385     return 0;
386 }
```

5.22.1.2 deleteQueue()

```
void deleteQueue (
            queue * queue )
```

Definition at line 388 of file R1commands.c.

```
389 {
390     PCB *tempPtr;
391     int loop;
392     for (loop = 0; loop < queue->count; loop++)
393     {
394         tempPtr = queue->head;
395         removePCB(tempPtr);
396     }
397 }
```

5.22.1.3 getDate()

```
void getDate ( )
```

Definition at line 179 of file R1commands.c.

```
180 {
181
182     char buffer[4] = "\0\0\0\0";
183     int count = 4;
184     char divider = '/';
185     char newLine[1] = "\n";
186     int newLineCount = 1;
187
188     outb(0x70, 0x07); // getting Day of month value
189     BCDtoChar(inb(0x71), buffer);
190     buffer[2] = divider;
191     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
192     memset(buffer, '\0', count);
193
194     outb(0x70, 0x08); // getting Month value
195     BCDtoChar(inb(0x71), buffer);
196     buffer[2] = divider;
197     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
```

```
198     memset(buffer, '\0', count);
199
200     outb(0x70, 0x32); // getting Year value second byte
201     BCDtoChar(inb(0x71), buffer);
202     buffer[2] = '\0';
203     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
204     memset(buffer, '\0', count);
205
206     outb(0x70, 0x09); // getting Year value first byte
207     BCDtoChar(inb(0x71), buffer);
208     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
209     memset(buffer, '\0', count);
210
211     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
212     memset(newLine, '\0', newLineCount);
213 }
```

5.22.1.4 getTime()

```
void getTime ( )
```

Definition at line 61 of file R1commands.c.

```
62 {
63
64     char buffer[4] = "\0\0\0";
65     int count = 4;
66     char divider = ':';
67     char newLine[1] = "\n";
68     int newLineCount = 1;
69
70     outb(0x70, 0x04); // getting Hour value
71     BCDtoChar(inb(0x71), buffer);
72     buffer[2] = divider;
73     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
74     memset(buffer, '\0', count);
75
76     outb(0x70, 0x02); // getting Minute value
77     BCDtoChar(inb(0x71), buffer);
78     buffer[2] = divider;
79     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
80     memset(buffer, '\0', count);
81
82     outb(0x70, 0x00); // getting Second value
83     BCDtoChar(inb(0x71), buffer);
84     buffer[2] = '\0';
85     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
86     memset(buffer, '\0', count);
87
88     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
89     memset(newLine, '\0', newLineCount);
90 }
```

5.22.1.5 help()

```
void help ( )
```

Definition at line 28 of file R1commands.c.

```
29 {
30     printMessage("help: Returns basic command information.\n");
31     printMessage("version: Returns the current version of the software.\n");
32     printMessage("getTime: Returns the current set time.\n");
33     printMessage("setTime: Allows the user to change the set time.\n");
34     printMessage("getDate: Returns the current set date.\n");
35     printMessage("setDate: Allows the user to change the set date.\n");
36     // printMessage("createPCB: Will create a PCB and put it into the ready queue by default.\n");
37     printMessage("deletePCB: Will delete a specific PCB from what ever queue it is in.\n");
38     // printMessage("blockPCB: Will change a specific PCB's state to blocked.\n");
39     // printMessage("unblockPCB: Will change a specific PCB's state to ready.\n");
40     printMessage("suspendPCB: Will suspend a specific PCB.\n");
41     printMessage("resumePCB: Will unsuspend a specific PCB.\n");
```

```
42     printMessage("setPCBPriority: Will change the priority of a specific PCB.\n");
43     printMessage("showPCB: Will display the name, class, state, suspended status, and priority of a
       specific PCB.\n");
44     printMessage("showReady: Will display the name, class, state, suspended status, and priority of every
       PCB in the ready queue.\n");
45     printMessage("showSuspendedReady: Will display the name, class, state, suspended status, and priority
       of every PCB in the suspended ready queue.\n");
46     printMessage("showSuspendedBlocked: Will display the name, class, state, suspended status, and
       priority of every PCB in the suspended blocked queue.\n");
47     printMessage("showBlocked: Will display the name, class, state, suspended status, and priority of
       every PCB in the blocked queue.\n");
48     printMessage("showReady: Will display the name, class, state, suspended status, and priority of every
       PCB in all 4 queues.\n");
49     printMessage("yield: Will cause commhand to voluntarily allow other processes to use the CPU.\n
       (removed for R4)");
50     printMessage("loadr3: Will load all processes for R3. \n");
51     printMessage("quit: Allows the user to shut the system down.\n");
52 }
```

5.22.1.6 intToBCD()

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

Definition at line 370 of file R1commands.c.

```
371 {
372
373     return (((test / 10) « 4) | (test % 10));
374 }
```

5.22.1.7 printMessage()

```
void printMessage (
              char * str )
```

Definition at line 13 of file R1commands.c.

```
14 {
15     char Desc[137];
16
17     size_t length = strlen(str);
18     if (length > (sizeof(Desc) - 2))
19     {
20         length = sizeof(Desc) - 2;
21         Desc[sizeof(Desc) - 1] = '\0';
22     }
23     strcpy(Desc, str);
24     int tempBuffer = strlen(Desc);
25     sys_req(WRITE, DEFAULT_DEVICE, (char *)Desc, &tempBuffer);
26 }
```

5.22.1.8 quit()

```
int quit ( )
```

Definition at line 422 of file R1commands.c.

```
423 {
424     int flag = 0;
425 
426     printMessage("Are you sure you want to shutdown? y/n\n");
427 
428     char quitAns[] = "\0\0";
429     int quitAnsLength = 1;
430     sys_req(READ, DEFAULT_DEVICE, quitAns, &quitAnsLength);
431     char answer = quitAns[0];
432 
433     if (answer == 'y' || answer == 'Y')
434     {
435         flag = 1;
436         //removeAll processes.
437         removeAll();
438         printMessage("\n");
439     }
440     else if (answer == 'n' || answer == 'N')
441     {
442         flag = 0;
443         printMessage("\n");
444     }
445     else
446     {
447         printMessage("Invalid input!\n");
448     }
449 
450     return flag;
451 }
```

5.22.1.9 removeAll()

```
void removeAll ( )
```

Definition at line 399 of file R1commands.c.

```
400 {
401     if (getReady()->head != NULL)
402     {
403         deleteQueue(getReady());
404     }
405 
406     if (getBlocked()->head != NULL)
407     {
408         deleteQueue(getBlocked());
409     }
410 
411     if (getSuspendedBlocked()->head != NULL)
412     {
413         deleteQueue(getSuspendedBlocked());
414     }
415 
416     if (getSuspendedReady()->head != NULL)
417     {
418         deleteQueue(getSuspendedReady());
419     }
420 }
```

5.22.1.10 setDate()

```
int setDate ( )
```

Definition at line 215 of file R1commands.c.

```
216 {
217
218     int count = 4; // used to print year
219
221     printMessage("Please type the desired year. I.E.: yyyy.\n");
222
223     char year[5] = "\0\0\0\0\0"; // year buffer
224
225     int flag = 0; // thrown if input is invalid
226
227     do
228     {
229         sys_req(READ, DEFAULT_DEVICE, year, &count);
230         if (atoi(year) > 0)
231         {
232
233             printMessage("\n");
234             flag = 0;
235
236             char yearUpper[3] = "\0\0\0";
237             char yearLower[3] = "\0\0\0";
238
239             yearUpper[0] = year[0];
240             yearUpper[1] = year[1];
241             yearLower[0] = year[2];
242             yearLower[1] = year[3];
243
244             cli();
245
246             outb(0x70, 0x32); // Setting first byte year value
247             outb(0x71, intToBCD(atoi(yearUpper)));
248
249             outb(0x70, 0x09); // Setting second byte year value
250             outb(0x71, intToBCD(atoi(yearLower)));
251
252             sti();
253         }
254         else
255         {
256             printMessage("\nInvalid year.\n");
257             flag = 1;
258         }
259     } while (flag == 1);
260
262     printMessage("Please type the desired month. I.E.: mm.\n");
263
264     char month[4] = "\0\0\n\0";
265     count = 4; // used to print month
266
267     do
268     {
269         sys_req(READ, DEFAULT_DEVICE, month, &count);
270         if (atoi(month) < 13 && atoi(month) > 0)
271         {
272
273             printMessage("\n");
274             flag = 0;
275
276             cli();
277
278             outb(0x70, 0x08); // Setting month value
279             outb(0x71, intToBCD(atoi(month)));
280
281             sti();
282         }
283         else
284         {
285             printMessage("\nInvalid month.\n");
286             flag = 1;
287         }
288     } while (flag == 1);
289
291     printMessage("Please type the desired day of month. I.E.: dd.\n");
292
293     char day[4] = "\0\0\n\0";
294     count = 4; // used to print day
295
296     do
297     {
298         sys_req(READ, DEFAULT_DEVICE, day, &count);
```

```
299         printMessage("\n");
300         if ((atoi(year) % 4 == 0 && atoi(year) % 100 != 0) || atoi(year) % 400 == 0)
301         { // checking for leap year
302 
303             printMessage("This is a leap year. February has 29 days.\n");
304 
305             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)
306             {
307                 flag = 1;
308                 printMessage("Invalid day.\n");
309             }
310             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
    atoi(day) > 30)
311             {
312                 flag = 1;
313                 printMessage("Invalid day.\n");
314             }
315             else if ((atoi(month) == 2) && atoi(day) > 29)
316             {
317                 flag = 1;
318                 printMessage("Invalid day.\n");
319             }
320             else
321             {
322 
323                 flag = 0;
324                 cli();
325 
326                 outb(0x70, 0x07); // Setting day of month value
327                 outb(0x71, intToBCD(atoi(day)));
328 
329                 sti();
330             }
331         }
332         else if (atoi(year) % 4 != 0 || atoi(year) % 400 != 0)
333         { // checking for leap year
334 
335             printMessage("This is not a leap year.\n");
336 
337             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)
338             {
339                 flag = 1;
340                 printMessage("Invalid day.\n");
341             }
342             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
    atoi(day) > 30)
343             {
344                 flag = 1;
345                 printMessage("Invalid day.\n");
346             }
347             else if ((atoi(month) == 2) && atoi(day) > 28)
348             {
349                 flag = 1;
350                 printMessage("Invalid day.\n");
351             }
352             else
353             {
354 
355                 cli();
356 
357                 outb(0x70, 0x07); // Setting day of month value
358                 outb(0x71, intToBCD(atoi(day)));
359 
360                 sti();
361             }
362         }
363 
364     } while (flag == 1);
365 
366     printMessage("The date has been set.\n");
367     return 0;
368 }
```

5.22.1.11 setTime()

```
int setTime ( )
```

Definition at line 92 of file R1commands.c.

```
93 {
94
95     int count = 4; // counter for printing
96
98     printMessage("Please type the desired hours. I.E.: hh.\n");
99
100     char hour[4] = "\0\0\n\0";
101
102     int flag = 0;
103
104     do
105     {
106         sys_req(READ, DEFAULT_DEVICE, hour, &count);
107         if (atoi(hour) < 24 && atoi(hour) >= 0)
108         {
109
110             printMessage("\n");
111             flag = 0;
112         }
113         else
114         {
115             printMessage("\nInvalid hours.\n");
116             flag = 1;
117         }
118     } while (flag == 1);
119
121     printMessage("Please type the desired minutes. I.E.: mm.\n");
122
123     char minute[4] = "\0\0\n\0";
124
125     do
126     {
127         sys_req(READ, DEFAULT_DEVICE, minute, &count);
128         if (atoi(minute) < 60 && atoi(minute) >= 0)
129         {
130
131             printMessage("\n");
132             flag = 0;
133         }
134         else
135         {
136             printMessage("\nInvalid minutes.\n");
137             flag = 1;
138         }
139     } while (flag == 1);
140
142     printMessage("Please type the desired seconds. I.E.: ss.\n");
143     char second[4] = "\0\0\n\0";
144
145     do
146     {
147         sys_req(READ, DEFAULT_DEVICE, second, &count);
148         if (atoi(second) < 60 && atoi(second) >= 0)
149         {
150
151             printMessage("\n");
152             flag = 0;
153         }
154         else
155         {
156             printMessage("Invalid seconds.\n");
157             flag = 1;
158         }
159     } while (flag == 1);
160
161     cli();
162
163     outb(0x70, 0x04); // Hour
164     outb(0x71, intToBCD(atoi(hour)));
165
166     outb(0x70, 0x02); // Minute
167     outb(0x71, intToBCD(atoi(minute)));
168
169     outb(0x70, 0x00); // Second
170     outb(0x71, intToBCD(atoi(second)));
171
172     sti();
173
174     printMessage("The time has been set.\n");
175
176     return 0;
177 }
```

5.22.1.12 version()

```
int version ( )
```

Definition at line 54 of file R1commands.c.

```
55 {
56     printMessage("Version 3.75\n");
57 
58     return 0;
59 }
```

5.23 modules/R1/R1commands.h File Reference

Functions

- void printMessage (char *str)
- 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 376 of file R1commands.c.

```
377 {
378 
379     int val1 = (test / 16);
380     int val2 = (test % 16);
381 
382     buffer[0] = val1 + '0';
383     buffer[1] = val2 + '0';
384 
385     return 0;
386 }
```

5.23.1.2 change_int_to_binary()

```
unsigned int change_int_to_binary (
            int test )
```

5.23.1.3 getDate()

```
void getDate ( )
```

Definition at line 179 of file R1commands.c.

```
180 {
181
182     char buffer[4] = "\0\0\0\0";
183     int count = 4;
184     char divider = '/';
185     char newLine[1] = "\n";
186     int newLineCount = 1;
187
188     outb(0x70, 0x07); // getting Day of month value
189     BCDtoChar(inb(0x71), buffer);
190     buffer[2] = divider;
191     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
192     memset(buffer, '\0', count);
193
194     outb(0x70, 0x08); // getting Month value
195     BCDtoChar(inb(0x71), buffer);
196     buffer[2] = divider;
197     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
198     memset(buffer, '\0', count);
199
200     outb(0x70, 0x32); // getting Year value second byte
201     BCDtoChar(inb(0x71), buffer);
202     buffer[2] = '\0';
203     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
204     memset(buffer, '\0', count);
205
206     outb(0x70, 0x09); // getting Year value first byte
207     BCDtoChar(inb(0x71), buffer);
208     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
209     memset(buffer, '\0', count);
210
211     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
212     memset(newLine, '\0', newLineCount);
213 }
```

5.23.1.4 getTime()

```
void getTime ( )
```

Definition at line 61 of file R1commands.c.

```
62 {
63
64     char buffer[4] = "\0\0\0";
65     int count = 4;
66     char divider = ':';
67     char newLine[1] = "\n";
68     int newLineCount = 1;
69
70     outb(0x70, 0x04); // getting Hour value
71     BCDtoChar(inb(0x71), buffer);
72     buffer[2] = divider;
73     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
74     memset(buffer, '\0', count);
75
76     outb(0x70, 0x02); // getting Minute value
77     BCDtoChar(inb(0x71), buffer);
78     buffer[2] = divider;
79     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
80     memset(buffer, '\0', count);
81
82     outb(0x70, 0x00); // getting Second value
83     BCDtoChar(inb(0x71), buffer);
84     buffer[2] = '\0';
85     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
86     memset(buffer, '\0', count);
87
88     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
89     memset(newLine, '\0', newLineCount);
90 }
```

5.23.1.5 help()

```
void help ( )
```

Definition at line 28 of file R1commands.c.

```
29 {
30     printMessage("help: Returns basic command information.\n");
31     printMessage("version: Returns the current version of the software.\n");
32     printMessage("getTime: Returns the current set time.\n");
33     printMessage("setTime: Allows the user to change the set time.\n");
34     printMessage("getDate: Returns the current set date.\n");
35     printMessage("setDate: Allows the user to change the set date.\n");
36     // printMessage("createPCB: Will create a PCB and put it into the ready queue by default.\n");
37     printMessage("deletePCB: Will delete a specific PCB from what ever queue it is in.\n");
38     // printMessage("blockPCB: Will change a specific PCB's state to blocked.\n");
39     // printMessage("unblockPCB: Will change a specific PCB's state to ready.\n");
40     printMessage("suspendPCB: Will suspend a specific PCB.\n");
41     printMessage("resumePCB: Will unsuspend a specific PCB.\n");
42     printMessage("setPCBPriority: Will change the priority of a specific PCB.\n");
43     printMessage("showPCB: Will display the name, class, state, suspended status, and priority of a
       specific PCB.\n");
44     printMessage("showReady: Will display the name, class, state, suspended status, and priority of every
       PCB in the ready queue.\n");
45     printMessage("showSuspendedReady: Will display the name, class, state, suspended status, and priority
       of every PCB in the suspended ready queue.\n");
46     printMessage("showSuspendedBlocked: Will display the name, class, state, suspended status, and
       priority of every PCB in the suspended blocked queue.\n");
47     printMessage("showBlocked: Will display the name, class, state, suspended status, and priority of
       every PCB in the blocked queue.\n");
48     printMessage("showReady: Will display the name, class, state, suspended status, and priority of every
       PCB in all 4 queues.\n");
49     printMessage("yield: Will cause commhand to voluntarily allow other processes to use the CPU.\n
       (removed for R4)");
50     printMessage("loadr3: Will load all processes for R3. \n");
51     printMessage("quit: Allows the user to shut the system down.\n");
52 }
```

5.23.1.6 printMessage()

```
void printMessage (
            char * str )
```

Definition at line 13 of file R1commands.c.

```
14 {
15     char Desc[137];
16
17     size_t length = strlen(str);
18     if (length > (sizeof(Desc) - 2))
19     {
20         length = sizeof(Desc) - 2;
21         Desc[sizeof(Desc) - 1] = '\0';
22     }
23     strcpy(Desc, str);
24     int tempBuffer = strlen(Desc);
25     sys_req(WRITE, DEFAULT_DEVICE, (char *)Desc, &tempBuffer);
26 }
```

5.23.1.7 quit()

```
int quit ( )
```

Definition at line 422 of file R1commands.c.

```
423 {
424     int flag = 0;
425
426     printMessage("Are you sure you want to shutdown? y/n\n");
427
```

```
428     char quitAns[] = "\0\0";
429     int quitAnsLength = 1;
430     sys_req(READ, DEFAULT_DEVICE, quitAns, &quitAnsLength);
431     char answer = quitAns[0];
432 
433     if (answer == 'y' || answer == 'Y')
434     {
435         flag = 1;
436         //removeAll processes.
437         removeAll();
438         printMessage("\n");
439     }
440     else if (answer == 'n' || answer == 'N')
441     {
442         flag = 0;
443         printMessage("\n");
444     }
445     else
446     {
447         printMessage("Invalid input!\n");
448     }
449 
450     return flag;
451 }
```

5.23.1.8 setDate()

```
void setDate ( )
```

Definition at line 215 of file R1commands.c.

```
216 {
217 
218     int count = 4; // used to print year
219 
221     printMessage("Please type the desired year. I.E.: yyyy.\n");
222 
223     char year[5] = "\0\0\0\0\0"; // year buffer
224 
225     int flag = 0; // thrown if input is invalid
226 
227     do
228     {
229         sys_req(READ, DEFAULT_DEVICE, year, &count);
230         if (atoi(year) > 0)
231         {
232 
233             printMessage("\n");
234             flag = 0;
235 
236             char yearUpper[3] = "\0\0\0";
237             char yearLower[3] = "\0\0\0";
238 
239             yearUpper[0] = year[0];
240             yearUpper[1] = year[1];
241             yearLower[0] = year[2];
242             yearLower[1] = year[3];
243 
244             cli();
245 
246             outb(0x70, 0x32); // Setting first byte year value
247             outb(0x71, intToBCD(atoi(yearUpper)));
248 
249             outb(0x70, 0x09); // Setting second byte year value
250             outb(0x71, intToBCD(atoi(yearLower)));
251 
252             sti();
253         }
254         else
255         {
256             printMessage("\nInvalid year.\n");
257             flag = 1;
258         }
259     } while (flag == 1);
260 
262     printMessage("Please type the desired month. I.E.: mm.\n");
263 
264     char month[4] = "\0\0\n\0";
265     count = 4; // used to print month
266 
```

```
267     do
268     {
269         sys_req(READ, DEFAULT_DEVICE, month, &count);
270         if (atoi(month) < 13 && atoi(month) > 0)
271         {
272 
273             printMessage("\n");
274             flag = 0;
275 
276             cli();
277 
278             outb(0x70, 0x08); // Setting month value
279             outb(0x71, intToBCD(atoi(month)));
280 
281             sti();
282         }
283         else
284         {
285             printMessage("\nInvalid month.\n");
286             flag = 1;
287         }
288     } while (flag == 1);
289 
291     printMessage("Please type the desired day of month. I.E.: dd.\n");
292 
293     char day[4] = "\0\0\n\0";
294     count = 4; // used to print day
295 
296     do
297     {
298         sys_req(READ, DEFAULT_DEVICE, day, &count);
299         printMessage("\n");
300         if ((atoi(year) % 4 == 0 && atoi(year) % 100 != 0) || atoi(year) % 400 == 0)
301         { // checking for leap year
302 
303             printMessage("This is a leap year. February has 29 days.\n");
304 
305             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)
306             {
307                 flag = 1;
308                 printMessage("Invalid day.\n");
309             }
310             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
    atoi(day) > 30)
311             {
312                 flag = 1;
313                 printMessage("Invalid day.\n");
314             }
315             else if ((atoi(month) == 2) && atoi(day) > 29)
316             {
317                 flag = 1;
318                 printMessage("Invalid day.\n");
319             }
320             else
321             {
322 
323                 flag = 0;
324                 cli();
325 
326                 outb(0x70, 0x07); // Setting day of month value
327                 outb(0x71, intToBCD(atoi(day)));
328 
329                 sti();
330             }
331         }
332         else if (atoi(year) % 4 != 0 || atoi(year) % 400 != 0)
333         { // checking for leap year
334 
335             printMessage("This is not a leap year.\n");
336 
337             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)
338             {
339                 flag = 1;
340                 printMessage("Invalid day.\n");
341             }
342             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
    atoi(day) > 30)
343             {
344                 flag = 1;
345                 printMessage("Invalid day.\n");
346             }
347             else if ((atoi(month) == 2) && atoi(day) > 28)
348             {
349                 flag = 1;
350                 printMessage("Invalid day.\n");
```

```
351             }
352             else
353             {
354 
355                 cli();
356 
357                 outb(0x70, 0x07); // Setting day of month value
358                 outb(0x71, intToBCD(atoi(day)));
359 
360                 sti();
361             }
362         }
363 
364     } while (flag == 1);
365 
366     printMessage("The date has been set.\n");
367     return 0;
368 }
```

5.23.1.9 setTime()

```
void setTime ( )
```

Definition at line 92 of file R1commands.c.

```
93 {
94 
95     int count = 4; // counter for printing
96 
98     printMessage("Please type the desired hours. I.E.: hh.\n");
99 
100     char hour[4] = "\0\0\n\0";
101 
102     int flag = 0;
103 
104     do
105     {
106         sys_req(READ, DEFAULT_DEVICE, hour, &count);
107         if (atoi(hour) < 24 && atoi(hour) >= 0)
108         {
109 
110             printMessage("\n");
111             flag = 0;
112         }
113         else
114         {
115             printMessage("\nInvalid hours.\n");
116             flag = 1;
117         }
118     } while (flag == 1);
119 
121     printMessage("Please type the desired minutes. I.E.: mm.\n");
122 
123     char minute[4] = "\0\0\n\0";
124 
125     do
126     {
127         sys_req(READ, DEFAULT_DEVICE, minute, &count);
128         if (atoi(minute) < 60 && atoi(minute) >= 0)
129         {
130 
131             printMessage("\n");
132             flag = 0;
133         }
134         else
135         {
136             printMessage("\nInvalid minutes.\n");
137             flag = 1;
138         }
139     } while (flag == 1);
140 
142     printMessage("Please type the desired seconds. I.E.: ss.\n");
143     char second[4] = "\0\0\n\0";
144 
145     do
146     {
147         sys_req(READ, DEFAULT_DEVICE, second, &count);
148         if (atoi(second) < 60 && atoi(second) >= 0)
149         {
150 
```

```
151             printMessage("\n");
152             flag = 0;
153         }
154         else
155         {
156             printMessage("Invalid seconds.\n");
157             flag = 1;
158         }
159     } while (flag == 1);
160 
161     cli();
162 
163     outb(0x70, 0x04); // Hour
164     outb(0x71, intToBCD(atoi(hour)));
165 
166     outb(0x70, 0x02); // Minute
167     outb(0x71, intToBCD(atoi(minute)));
168 
169     outb(0x70, 0x00); // Second
170     outb(0x71, intToBCD(atoi(second)));
171 
172     sti();
173 
174     printMessage("The time has been set.\n");
175 
176     return 0;
177 }
```

5.23.1.10 version()

```
void version ( )
```

Definition at line 54 of file R1commands.c.

```
55 {
56     printMessage("Version 3.75\n");
57 
58     return 0;
59 }
```

5.24 modules/R2/R2_Internal_Functions_And_Structures.c File Reference

```
#include <string.h>
#include <core/serial.h>
#include "../mpx_supt.h"
#include "../R1/R1commands.h"
#include "R2_Internal_Functions_And_Structures.h"
#include "../R3/R3commands.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 17 of file R2_Internal_Functions_And_Structures.c.

```
18 {
19     //COLTON WILL PROGRAM THIS FUNCTION
20
21     //allocatePCB() will use sys_alloc_mem() to allocate memory for a new PCB, possible including the
       stack, and perform any reasonable initialization.
22     PCB *newPCB = (PCB *)sys_alloc_mem(sizeof(PCB));
23
24     char name[20] = "newPCB";
25     strcpy(newPCB->processName, name);
26
27     newPCB->suspendedStatus = 1;
28     newPCB->runningStatus = -1;
29     newPCB->stackTop = (newPCB->stack + 1024) - sizeof(context);
30     newPCB->stackBase = newPCB->stack;
31     newPCB->priority = 0;
32
33     // Setting the PCBs prev and next PCB
34     newPCB->nextPCB = NULL;
35     newPCB->prevPCB = NULL;
36
37     newPCB->processClass = NULL;
38
39     return newPCB;
40 }
```

5.24.1.2 allocateQueues()

```
void allocateQueues ( )
```

Definition at line 430 of file R2_Internal_Functions_And_Structures.c.

```
431 {
432     ready = sys_alloc_mem(sizeof(queue));
433     ready->count = 0;
434     ready->head = NULL;
435     ready->tail = NULL;
436
437     blocked = sys_alloc_mem(sizeof(queue));
438     blocked->count = 0;
439     blocked->head = NULL;
440     blocked->tail = NULL;
441
442     suspendedReady = sys_alloc_mem(sizeof(queue));
443     suspendedReady->count = 0;
444     suspendedReady->head = NULL;
445     suspendedReady->tail = NULL;
446
447     suspendedBlocked = sys_alloc_mem(sizeof(queue));
448     suspendedBlocked->count = 0;
449     suspendedBlocked->head = NULL;
450     suspendedBlocked->tail = NULL;
451 }
```

5.24.1.3 findPCB()

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

Definition at line 82 of file R2_Internal_Functions_And_Structures.c.

```
83 {
84     // ANASTASE WILL PROGRAM THIS FUNCTION
85 
86     //findPCB() will search all queues for a process with a given name.
87 
88     if (strlen(processName) > 20)
89     {
90 
91         printMessage("Invalid process name.\n");
92         return NULL;
93         //return cz we have to stop if the process name is too long
94     }
95     else
96     {
97         PCB *tempPCB = ready->head;
98         int value = 0;
99         while (value < ready->count)
100         {
101             if (strcmp(tempPCB->processName, processName) == 0)
102             {
103                 return tempPCB;
104             }
105             else
106             {
107                 tempPCB = tempPCB->nextPCB;
108                 value++;
109             }
110         }
111 
112         tempPCB = blocked->head;
113         value = 0;
114         while (value < blocked->count)
115         {
116             if (strcmp(tempPCB->processName, processName) == 0)
117             {
118                 return tempPCB;
119             }
120             else
121             {
122                 tempPCB = tempPCB->nextPCB;
123                 value++;
124             }
125         }
126 
127         tempPCB = suspendedBlocked->head;
128         value = 0;
129         while (value < suspendedBlocked->count)
130         {
131             if (strcmp(tempPCB->processName, processName) == 0)
132             {
133                 return tempPCB;
134             }
135             else
136             {
137                 tempPCB = tempPCB->nextPCB;
138                 value++;
139             }
140         }
141 
142         tempPCB = suspendedReady->head;
143         value = 0;
144         while (value < suspendedReady->count)
145         {
146             if (strcmp(tempPCB->processName, processName) == 0)
147             {
148                 return tempPCB;
149             }
150             else
151             {
152                 tempPCB = tempPCB->nextPCB;
153                 value++;
154             }
155         }
156 
157         return NULL;
158     }
159 }
```

5.24.1.4 freePCB()

```
int freePCB (
             PCB * PCB_to_free )
```

Definition at line 42 of file R2_Internal_Functions_And_Structures.c.

```
43 {
44     // ANASTASE WILL PROGRAM THIS FUNCTION
45
46     //freePCB() will use sys_free_mem() to free all memory associated with a given PCB (the stack, the
       PCB itself, etc.)
47
48     return sys_free_mem(PCB_to_free);
49 }
```

5.24.1.5 getBlocked()

```
queue* getBlocked ( )
```

Definition at line 458 of file R2_Internal_Functions_And_Structures.c.

```
459 {
460     return blocked;
461 }
```

5.24.1.6 getReady()

```
queue* getReady ( )
```

Definition at line 453 of file R2_Internal_Functions_And_Structures.c.

```
454 {
455     return ready;
456 }
```

5.24.1.7 getSuspendedBlocked()

```
queue* getSuspendedBlocked ( )
```

Definition at line 468 of file R2_Internal_Functions_And_Structures.c.

```
469 {
470     return suspendedBlocked;
471 }
```

5.24.1.8 getSuspendedReady()

```
queue* getSuspendedReady ( )
```

Definition at line 463 of file R2_Internal_Functions_And_Structures.c.

```
464 {
465     return suspendedReady;
466 }
```

5.24.1.9 insertPCB()

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

Definition at line 161 of file R2_Internal_Functions_And_Structures.c.

```
162 {
163     //BENJAMIN WILL PROGRAM THIS FUNCTION
164
165     //insertPCB() will insert a PCB into the appropriate queue.
166     //Note: The ready queue is a priority queue and the blocked queue is a FIFO queue.
167
168     if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 1)
169     { // Insert into ready queue
170         PCB *tempPtr = ready->head;
171
172         if (tempPtr != NULL)
173         {
174             int temp = 0;
175             while (temp < ready->count)
176             {
177                 if (PCB_to_insert->priority > ready->head->priority)
178                 { // insert at head
179                     PCB_to_insert->nextPCB = tempPtr;
180                     tempPtr->prevPCB = PCB_to_insert;
181                     ready->head = PCB_to_insert;
182                     ready->count++;
183                     break;
184                 }
185                 else if (PCB_to_insert->priority <= ready->tail->priority)
186                 { // insert at tail
187                     ready->tail->nextPCB = PCB_to_insert;
188                     PCB_to_insert->prevPCB = ready->tail;
189                     ready->tail = PCB_to_insert;
190                     ready->count++;
191                     break;
192                 }
193                 else if (PCB_to_insert->priority > tempPtr->priority)
194                 { // insert at middle
195                     PCB *prevPtr = tempPtr->prevPCB;
196
197                     prevPtr->nextPCB = PCB_to_insert;
198
199                     PCB_to_insert->prevPCB = prevPtr;
200                     PCB_to_insert->nextPCB = tempPtr;
201
202                     tempPtr->prevPCB = PCB_to_insert;
203
204                     ready->count++;
205                     break;
206                 }
207                 else
208                 { // move tempPtr through the queue
209                     tempPtr = tempPtr->nextPCB;
210                 }
211                 temp++;
212             }
213         }
214         else
215         {
216             ready->head = PCB_to_insert;
217             ready->tail = PCB_to_insert;
218             ready->count++;
219         }
220     }
221     else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
222     { // Insert into suspended ready queue
223         PCB *tempPtr = suspendedReady->head;
224
225         if (tempPtr != NULL)
226         {
227             int temp = 0;
228             while (temp < suspendedReady->count)
229             {
230                 if (PCB_to_insert->priority > suspendedReady->head->priority)
231                 { // insert at head
232                     PCB_to_insert->nextPCB = tempPtr;
233                     tempPtr->prevPCB = PCB_to_insert;
234                     suspendedReady->head = PCB_to_insert;
235                     suspendedReady->count++;
236                     break;
237                 }
238                 else if (PCB_to_insert->priority <= suspendedReady->tail->priority)
239                 { // insert at tail
```

```
240
241                     suspendedReady->tail->nextPCB = PCB_to_insert;
242                     PCB_to_insert->prevPCB = suspendedReady->tail;
243                     suspendedReady->tail = PCB_to_insert;
244                     suspendedReady->count++;
245                     break;
246                 }
247                 else if (PCB_to_insert->priority > tempPtr->priority)
248                 { // insert at middle
249                     PCB *prevPtr = tempPtr->prevPCB;
250
251                     prevPtr->nextPCB = PCB_to_insert;
252
253                     PCB_to_insert->prevPCB = prevPtr;
254                     PCB_to_insert->nextPCB = tempPtr;
255
256                     tempPtr->prevPCB = PCB_to_insert;
257
258                     ready->count++;
259                     break;
260                 }
261                 else
262                 { // move tempPtr through the queue
263                     tempPtr = tempPtr->nextPCB;
264                 }
265                 temp++;
266             }
267         }
268         else
269         {
270             suspendedReady->count++;
271             suspendedReady->head = PCB_to_insert;
272             suspendedReady->tail = PCB_to_insert;
273         }
274     }
275     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 1)
276     { // Insert into blocked queue
277         if (blocked->head != NULL)
278         {
279             blocked->tail->nextPCB = PCB_to_insert;
280             PCB_to_insert->prevPCB = blocked->tail;
281             blocked->tail = PCB_to_insert;
282             blocked->count++;
283         }
284         else
285         {
286             blocked->head = PCB_to_insert;
287             blocked->tail = PCB_to_insert;
288             blocked->count++;
289         }
290     }
291     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
292     { // Insert into suspended blocked queue
293         if (suspendedBlocked->head != NULL)
294         {
295             suspendedBlocked->tail->nextPCB = PCB_to_insert;
296             PCB_to_insert->prevPCB = suspendedBlocked->tail;
297             suspendedBlocked->tail = PCB_to_insert;
298             suspendedBlocked->count++;
299         }
300         else
301         {
302             suspendedBlocked->head = PCB_to_insert;
303             suspendedBlocked->tail = PCB_to_insert;
304             suspendedBlocked->count++;
305         }
306     }
307 }
```

5.24.1.10 removePCB()

```
int removePCB (
            PCB * PCB_to_remove )
```

Definition at line 309 of file R2_Internal_Functions_And_Structures.c.

```
310 {
311     //BENJAMIN WILL PROGRAM THIS FUNCTION
312
```

```
313     //removePCB() will remove a PCB from the queue in which it is currently stored.
314
315     if (PCB_to_remove == NULL)
316     {
317         return 1;
318     }
319     else if (PCB_to_remove == ready->head)
320     {
321         //PCB *removedNext = PCB_to_remove->nextPCB;
322
323         ready->head = PCB_to_remove->nextPCB;
324         ready->head->prevPCB = NULL;
325         PCB_to_remove->nextPCB = NULL;
326         ready->count--;
327         return 0;
328     }
329     else if (PCB_to_remove == blocked->head)
330     {
331         PCB *removedNext = PCB_to_remove->nextPCB;
332         blocked->head = removedNext;
333         removedNext->prevPCB = NULL;
334         PCB_to_remove->nextPCB = NULL;
335         blocked->count--;
336         return 0;
337     }
338     else if (PCB_to_remove == suspendedReady->head)
339     {
340         PCB *removedNext = PCB_to_remove->nextPCB;
341
342         suspendedReady->head = removedNext;
343         removedNext->prevPCB = NULL;
344         PCB_to_remove->nextPCB = NULL;
345         suspendedReady->count--;
346         return 0;
347     }
348     else if (PCB_to_remove == suspendedBlocked->head)
349     {
350         PCB *removedNext = PCB_to_remove->nextPCB;
351
352         suspendedBlocked->head = removedNext;
353         removedNext->prevPCB = NULL;
354         PCB_to_remove->nextPCB = NULL;
355         suspendedBlocked->count--;
356         return 0;
357     }
358     else if (PCB_to_remove == ready->tail)
359     {
360         PCB *removedPrev = PCB_to_remove->prevPCB;
361
362         ready->tail = removedPrev;
363         removedPrev->nextPCB = NULL;
364         PCB_to_remove->prevPCB = NULL;
365         ready->count--;
366         return 0;
367     }
368     else if (PCB_to_remove == blocked->tail)
369     {
370         PCB *removedPrev = PCB_to_remove->prevPCB;
371
372         blocked->tail = removedPrev;
373         removedPrev->nextPCB = NULL;
374         PCB_to_remove->prevPCB = NULL;
375         blocked->count--;
376         return 0;
377     }
378     else if (PCB_to_remove == suspendedReady->tail)
379     {
380         PCB *removedPrev = PCB_to_remove->prevPCB;
381
382         suspendedReady->tail = removedPrev;
383         removedPrev->nextPCB = NULL;
384         PCB_to_remove->prevPCB = NULL;
385         suspendedReady->count--;
386         return 0;
387     }
388     else if (PCB_to_remove == suspendedBlocked->tail)
389     {
390         PCB *removedPrev = PCB_to_remove->prevPCB;
391
392         suspendedBlocked->tail = removedPrev;
393         removedPrev->nextPCB = NULL;
394         PCB_to_remove->prevPCB = NULL;
395         suspendedBlocked->count--;
396         return 0;
397     }
398     else
399     {
```

```
400         // PCB *tempPrev = PCB_to_remove->prevPCB;
401         // PCB *tempNext = PCB_to_remove->nextPCB;
402
403         PCB_to_remove->prevPCB->nextPCB = PCB_to_remove->nextPCB;
404         PCB_to_remove->nextPCB->prevPCB = PCB_to_remove->prevPCB;
405
406         PCB_to_remove->nextPCB = NULL;
407         PCB_to_remove->prevPCB = NULL;
408
409         if (PCB_to_remove->runningStatus == 0 && PCB_to_remove->suspendedStatus == 1)
410         {
411             ready->count--;
412         }
413         else if (PCB_to_remove->runningStatus == -1 && PCB_to_remove->suspendedStatus == 1)
414         {
415             blocked->count--;
416         }
417         else if (PCB_to_remove->runningStatus == 0 && PCB_to_remove->suspendedStatus == 0)
418         {
419             suspendedReady->count--;
420         }
421         else if (PCB_to_remove->runningStatus == -1 && PCB_to_remove->suspendedStatus == 0)
422         {
423             suspendedBlocked->count--;
424         }
425
426         return 0;
427     }
428 }
```

5.24.1.11 setupPCB()

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

Definition at line 51 of file R2_Internal_Functions_And_Structures.c.

```
52 {
53     //COLTON WILL PROGRAM THIS FUNCTION
54
55     //setupPcb() will call allocatePCB() to create an empty PCB, initializes the PCB information, sets
       the PCB state to ready, not suspended.
56
57     PCB *returnedPCB = allocatePCB();
58
59     if (findPCB(processName)->processName == processName)
60     {
61         printMessage("There is already a PCB with this name.\n");
62
63         returnedPCB = NULL;
64     }
65     else
66     {
67
68         strcpy(returnedPCB->processName, processName);
69         returnedPCB->processClass = processClass;
70         returnedPCB->priority = processPriority;
71         returnedPCB->runningStatus = 0;
72         returnedPCB->suspendedStatus = 1;
73         returnedPCB->stackBase = returnedPCB->stack;
74         returnedPCB->stackTop = returnedPCB->stack + 1024 - sizeof(context);
75         returnedPCB->nextPCB = NULL;
76         returnedPCB->prevPCB = NULL;
77     }
78
79     return returnedPCB;
80 }
```

5.24.2 Variable Documentation

5.24.2.1 blocked

```
queue* blocked
```

Definition at line 11 of file R2_Internal_Functions_And_Structures.c.

5.24.2.2 ready

```
queue* ready
```

Definition at line 10 of file R2_Internal_Functions_And_Structures.c.

5.24.2.3 suspendedBlocked

```
queue* suspendedBlocked
```

Definition at line 13 of file R2_Internal_Functions_And_Structures.c.

5.24.2.4 suspendedReady

```
queue* suspendedReady
```

Definition at line 12 of file R2_Internal_Functions_And_Structures.c.

5.25 modules/R2/R2_Internal_Functions_And_Structures.h File Reference

Classes

- struct PCB
- struct queue

Typedefs

- typedef struct PCB PCB
- typedef struct queue queue

Functions

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

5.25.1 Typedef Documentation

5.25.1.1 PCB

```
typedef struct PCB PCB
```

5.25.1.2 queue

```
typedef struct queue queue
```

5.25.2 Function Documentation

5.25.2.1 allocatePCB()

```
PCB* allocatePCB ( )
```

Definition at line 17 of file R2_Internal_Functions_And_Structures.c.

```
18 {
19     //COLTON WILL PROGRAM THIS FUNCTION
20
21     //allocatePCB() will use sys_alloc_mem() to allocate memory for a new PCB, possible including the
       stack, and perform any reasonable initialization.
22     PCB *newPCB = (PCB *)sys_alloc_mem(sizeof(PCB));
23
24     char name[20] = "newPCB";
25     strcpy(newPCB->processName, name);
26
27     newPCB->suspendedStatus = 1;
28     newPCB->runningStatus = -1;
29     newPCB->stackTop = (newPCB->stack + 1024) - sizeof(context);
30     newPCB->stackBase = newPCB->stack;
31     newPCB->priority = 0;
32
33     // Setting the PCBs prev and next PCB
34     newPCB->nextPCB = NULL;
35     newPCB->prevPCB = NULL;
36
37     newPCB->processClass = NULL;
38
39     return newPCB;
40 }
```

5.25.2.2 allocateQueues()

```
void allocateQueues ( )
```

Definition at line 430 of file R2_Internal_Functions_And_Structures.c.

```
431 {
432     ready = sys_alloc_mem(sizeof(queue));
433     ready->count = 0;
434     ready->head = NULL;
435     ready->tail = NULL;
436 
437     blocked = sys_alloc_mem(sizeof(queue));
438     blocked->count = 0;
439     blocked->head = NULL;
440     blocked->tail = NULL;
441 
442     suspendedReady = sys_alloc_mem(sizeof(queue));
443     suspendedReady->count = 0;
444     suspendedReady->head = NULL;
445     suspendedReady->tail = NULL;
446 
447     suspendedBlocked = sys_alloc_mem(sizeof(queue));
448     suspendedBlocked->count = 0;
449     suspendedBlocked->head = NULL;
450     suspendedBlocked->tail = NULL;
451 }
```

5.25.2.3 findPCB()

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

Definition at line 82 of file R2_Internal_Functions_And_Structures.c.

```
83 {
84     // ANASTASE WILL PROGRAM THIS FUNCTION
85 
86     //findPCB() will search all queues for a process with a given name.
87 
88     if (strlen(processName) > 20)
89     {
90 
91         printMessage("Invalid process name.\n");
92         return NULL;
93         //return cz we have to stop if the process name is too long
94     }
95     else
96     {
97         PCB *tempPCB = ready->head;
98         int value = 0;
99         while (value < ready->count)
100         {
101             if (strcmp(tempPCB->processName, processName) == 0)
102             {
103                 return tempPCB;
104             }
105             else
106             {
107                 tempPCB = tempPCB->nextPCB;
108                 value++;
109             }
110         }
111 
112         tempPCB = blocked->head;
113         value = 0;
114         while (value < blocked->count)
115         {
116             if (strcmp(tempPCB->processName, processName) == 0)
117             {
118                 return tempPCB;
119             }
120             else
121             {
122                 tempPCB = tempPCB->nextPCB;
123                 value++;
124             }
```

```
125         }
126
127         tempPCB = suspendedBlocked->head;
128         value = 0;
129         while (value < suspendedBlocked->count)
130         {
131             if (strcmp(tempPCB->processName, processName) == 0)
132             {
133                 return tempPCB;
134             }
135             else
136             {
137                 tempPCB = tempPCB->nextPCB;
138                 value++;
139             }
140         }
141
142         tempPCB = suspendedReady->head;
143         value = 0;
144         while (value < suspendedReady->count)
145         {
146             if (strcmp(tempPCB->processName, processName) == 0)
147             {
148                 return tempPCB;
149             }
150             else
151             {
152                 tempPCB = tempPCB->nextPCB;
153                 value++;
154             }
155         }
156
157         return NULL;
158     }
159 }
```

5.25.2.4 freePCB()

```
int freePCB (
              PCB * PCB_to_free )
```

Definition at line 42 of file R2_Internal_Functions_And_Structures.c.

```
43 {
44     // ANASTASE WILL PROGRAM THIS FUNCTION
45
46     //freePCB() will use sys_free_mem() to free all memory associated with a given PCB (the stack, the
       PCB itself, etc.)
47
48     return sys_free_mem(PCB_to_free);
49 }
```

5.25.2.5 getBlocked()

```
queue* getBlocked ( )
```

Definition at line 458 of file R2_Internal_Functions_And_Structures.c.

```
459 {
460     return blocked;
461 }
```

5.25.2.6 getReady()

```
queue* getReady ( )
```

Definition at line 453 of file R2_Internal_Functions_And_Structures.c.

```
454 {
455     return ready;
456 }
```

5.25.2.7 getSuspendedBlocked()

```
queue* getSuspendedBlocked ( )
```

Definition at line 468 of file R2_Internal_Functions_And_Structures.c.

```
469 {
470     return suspendedBlocked;
471 }
```

5.25.2.8 getSuspendedReady()

```
queue* getSuspendedReady ( )
```

Definition at line 463 of file R2_Internal_Functions_And_Structures.c.

```
464 {
465     return suspendedReady;
466 }
```

5.25.2.9 insertPCB()

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

Definition at line 161 of file R2_Internal_Functions_And_Structures.c.

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

```
187                     ready->tail->nextPCB = PCB_to_insert;
188                     PCB_to_insert->prevPCB = ready->tail;
189                     ready->tail = PCB_to_insert;
190                     ready->count++;
191                     break;
192                 }
193                 else if (PCB_to_insert->priority > tempPtr->priority)
194                 { // insert at middle
195                     PCB *prevPtr = tempPtr->prevPCB;
196 
197                     prevPtr->nextPCB = PCB_to_insert;
198 
199                     PCB_to_insert->prevPCB = prevPtr;
200                     PCB_to_insert->nextPCB = tempPtr;
201 
202                     tempPtr->prevPCB = PCB_to_insert;
203 
204                     ready->count++;
205                     break;
206                 }
207                 else
208                 { // move tempPtr through the queue
209                     tempPtr = tempPtr->nextPCB;
210                 }
211                 temp++;
212             }
213         }
214         else
215         {
216             ready->head = PCB_to_insert;
217             ready->tail = PCB_to_insert;
218             ready->count++;
219         }
220     }
221     else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
222     { // Insert into suspended ready queue
223         PCB *tempPtr = suspendedReady->head;
224 
225         if (tempPtr != NULL)
226         {
227             int temp = 0;
228             while (temp < suspendedReady->count)
229             {
230                 if (PCB_to_insert->priority > suspendedReady->head->priority)
231                 { // insert at head
232                     PCB_to_insert->nextPCB = tempPtr;
233                     tempPtr->prevPCB = PCB_to_insert;
234                     suspendedReady->head = PCB_to_insert;
235                     suspendedReady->count++;
236                     break;
237                 }
238                 else if (PCB_to_insert->priority <= suspendedReady->tail->priority)
239                 { // insert at tail
240 
241                     suspendedReady->tail->nextPCB = PCB_to_insert;
242                     PCB_to_insert->prevPCB = suspendedReady->tail;
243                     suspendedReady->tail = PCB_to_insert;
244                     suspendedReady->count++;
245                     break;
246                 }
247                 else if (PCB_to_insert->priority > tempPtr->priority)
248                 { // insert at middle
249                     PCB *prevPtr = tempPtr->prevPCB;
250 
251                     prevPtr->nextPCB = PCB_to_insert;
252 
253                     PCB_to_insert->prevPCB = prevPtr;
254                     PCB_to_insert->nextPCB = tempPtr;
255 
256                     tempPtr->prevPCB = PCB_to_insert;
257 
258                     ready->count++;
259                     break;
260                 }
261                 else
262                 { // move tempPtr through the queue
263                     tempPtr = tempPtr->nextPCB;
264                 }
265                 temp++;
266             }
267         }
268         else
269         {
270             suspendedReady->count++;
271             suspendedReady->head = PCB_to_insert;
272             suspendedReady->tail = PCB_to_insert;
273         }
```

```
274     }
275     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 1)
276     { // Insert into blocked queue
277         if (blocked->head != NULL)
278         {
279             blocked->tail->nextPCB = PCB_to_insert;
280             PCB_to_insert->prevPCB = blocked->tail;
281             blocked->tail = PCB_to_insert;
282             blocked->count++;
283         }
284         else
285         {
286             blocked->head = PCB_to_insert;
287             blocked->tail = PCB_to_insert;
288             blocked->count++;
289         }
290     }
291     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
292     { // Insert into suspended blocked queue
293         if (suspendedBlocked->head != NULL)
294         {
295             suspendedBlocked->tail->nextPCB = PCB_to_insert;
296             PCB_to_insert->prevPCB = suspendedBlocked->tail;
297             suspendedBlocked->tail = PCB_to_insert;
298             suspendedBlocked->count++;
299         }
300         else
301         {
302             suspendedBlocked->head = PCB_to_insert;
303             suspendedBlocked->tail = PCB_to_insert;
304             suspendedBlocked->count++;
305         }
306     }
307 }
```

5.25.2.10 removePCB()

```
int removePCB (
            PCB * PCB_to_remove )
```

Definition at line 309 of file R2_Internal_Functions_And_Structures.c.

```
310 {
311     //BENJAMIN WILL PROGRAM THIS FUNCTION
312
313     //removePCB() will remove a PCB from the queue in which it is currently stored.
314
315     if (PCB_to_remove == NULL)
316     {
317         return 1;
318     }
319     else if (PCB_to_remove == ready->head)
320     {
321         //PCB *removedNext = PCB_to_remove->nextPCB;
322
323         ready->head = PCB_to_remove->nextPCB;
324         ready->head->prevPCB = NULL;
325         PCB_to_remove->nextPCB = NULL;
326         ready->count--;
327         return 0;
328     }
329     else if (PCB_to_remove == blocked->head)
330     {
331         PCB *removedNext = PCB_to_remove->nextPCB;
332         blocked->head = removedNext;
333         removedNext->prevPCB = NULL;
334         PCB_to_remove->nextPCB = NULL;
335         blocked->count--;
336         return 0;
337     }
338     else if (PCB_to_remove == suspendedReady->head)
339     {
340         PCB *removedNext = PCB_to_remove->nextPCB;
341
342         suspendedReady->head = removedNext;
343         removedNext->prevPCB = NULL;
344         PCB_to_remove->nextPCB = NULL;
345         suspendedReady->count--;
346         return 0;
```

```
347     }
348     else if (PCB_to_remove == suspendedBlocked->head)
349     {
350         PCB *removedNext = PCB_to_remove->nextPCB;
351 
352         suspendedBlocked->head = removedNext;
353         removedNext->prevPCB = NULL;
354         PCB_to_remove->nextPCB = NULL;
355         suspendedBlocked->count--;
356         return 0;
357     }
358     else if (PCB_to_remove == ready->tail)
359     {
360         PCB *removedPrev = PCB_to_remove->prevPCB;
361 
362         ready->tail = removedPrev;
363         removedPrev->nextPCB = NULL;
364         PCB_to_remove->prevPCB = NULL;
365         ready->count--;
366         return 0;
367     }
368     else if (PCB_to_remove == blocked->tail)
369     {
370         PCB *removedPrev = PCB_to_remove->prevPCB;
371 
372         blocked->tail = removedPrev;
373         removedPrev->nextPCB = NULL;
374         PCB_to_remove->prevPCB = NULL;
375         blocked->count--;
376         return 0;
377     }
378     else if (PCB_to_remove == suspendedReady->tail)
379     {
380         PCB *removedPrev = PCB_to_remove->prevPCB;
381 
382         suspendedReady->tail = removedPrev;
383         removedPrev->nextPCB = NULL;
384         PCB_to_remove->prevPCB = NULL;
385         suspendedReady->count--;
386         return 0;
387     }
388     else if (PCB_to_remove == suspendedBlocked->tail)
389     {
390         PCB *removedPrev = PCB_to_remove->prevPCB;
391 
392         suspendedBlocked->tail = removedPrev;
393         removedPrev->nextPCB = NULL;
394         PCB_to_remove->prevPCB = NULL;
395         suspendedBlocked->count--;
396         return 0;
397     }
398     else
399     {
400         // PCB *tempPrev = PCB_to_remove->prevPCB;
401         // PCB *tempNext = PCB_to_remove->nextPCB;
402 
403         PCB_to_remove->prevPCB->nextPCB = PCB_to_remove->nextPCB;
404         PCB_to_remove->nextPCB->prevPCB = PCB_to_remove->prevPCB;
405 
406         PCB_to_remove->nextPCB = NULL;
407         PCB_to_remove->prevPCB = NULL;
408 
409         if (PCB_to_remove->runningStatus == 0 && PCB_to_remove->suspendedStatus == 1)
410         {
411             ready->count--;
412         }
413         else if (PCB_to_remove->runningStatus == -1 && PCB_to_remove->suspendedStatus == 1)
414         {
415             blocked->count--;
416         }
417         else if (PCB_to_remove->runningStatus == 0 && PCB_to_remove->suspendedStatus == 0)
418         {
419             suspendedReady->count--;
420         }
421         else if (PCB_to_remove->runningStatus == -1 && PCB_to_remove->suspendedStatus == 0)
422         {
423             suspendedBlocked->count--;
424         }
425 
426         return 0;
427     }
428 }
```

5.25.2.11 setupPCB()

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

Definition at line 51 of file R2_Internal_Functions_And_Structures.c.

```
52 {
53     //COLTON WILL PROGRAM THIS FUNCTION
54 
55     //setupPcb() will call allocatePCB() to create an empty PCB, initializes the PCB information, sets
       the PCB state to ready, not suspended.
56 
57     PCB *returnedPCB = allocatePCB();
58 
59     if (findPCB(processName)->processName == processName)
60     {
61         printMessage("There is already a PCB with this name.\n");
62 
63         returnedPCB = NULL;
64     }
65     else
66     {
67 
68         strcpy(returnedPCB->processName, processName);
69         returnedPCB->processClass = processClass;
70         returnedPCB->priority = processPriority;
71         returnedPCB->runningStatus = 0;
72         returnedPCB->suspendedStatus = 1;
73         returnedPCB->stackBase = returnedPCB->stack;
74         returnedPCB->stackTop = returnedPCB->stack + 1024 - sizeof(context);
75         returnedPCB->nextPCB = NULL;
76         returnedPCB->prevPCB = NULL;
77     }
78 
79     return returnedPCB;
80 }
```

5.26 modules/R2/R2commands.c File Reference

```
#include <string.h>
#include "../mpx_supt.h"
#include "../R1/R1commands.h"
#include "R2_Internal_Functions_And_Structures.h"
#include "R2commands.h"
#include <core/serial.h>
```

Functions

- void createPCB (char *processName, char processClass, int processPriority)
- void deletePCB (char *processName)
- void blockPCB (char *processName)
- void unblockPCB (char *processName)
- void suspendPCB (char *processName)
- void resumePCB (char *processName)
- void setPCBPriority (char *processName, int newProcessPriority)
- void showPCB (char *processName)
- void showQueue (PCB *pcb, int count)
- void showReady ()
- void showSuspendedReady ()
- void showSuspendedBlocked ()
- void showBlocked ()
- void showAll ()

5.26.1 Function Documentation

5.26.1.1 blockPCB()

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

Definition at line 94 of file R2commands.c.

```
95 { // ANASTASE WILL PROGRAM THIS FUNCTION
96
97     // find pcb and validate process name
98     PCB *pcb_to_block = findPCB(processName);
99
100     if (pcb_to_block != NULL)
101     {
102         pcb_to_block->runningStatus = -1; // blocked
103         removePCB(pcb_to_block);
104         insertPCB(pcb_to_block);
105
106         printMessage("The PCB was successfully blocked!\n");
107     }
108 }
```

5.26.1.2 createPCB()

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

Definition at line 12 of file R2commands.c.

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

5.26.1.3 deletePCB()

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

Definition at line 46 of file R2commands.c.

```
47 { // BENJAMIN WILL PROGRAM THIS FUNCTION
48     /*
49     The deletePCB command will remove a PCB from the appropriate queue and then free all associated
       memory.
50     This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
51     */
52     /*
53     Error Checking:
54     Name must be valid.
55     */
56
57     if (strlen(processName) > 20)
58     { // Check if the process has a valid name.
59         printMessage("The PCB could not be deleted as the name is longer than 20 characters!\n");
60     }
61
62     PCB *PCB_to_delete = findPCB(processName);
63
64     if (PCB_to_delete == NULL)
65     {
66         printMessage("The PCB you want to remove does not exist\n");
67     }
68     else if (strcmp(processName, "infinite") == 0 && PCB_to_delete->suspendedStatus != 0)
69     {
70         printMessage("In order to delete the infinite process it must be suspended first.\n");
71     }
72     else
73     {
74         int removed = removePCB(PCB_to_delete);
75         if (removed == 1)
76         {
77             printMessage("The PCB could not be unlinked.\n");
78         }
79         else
80         {
81             int result = sys_free_mem(PCB_to_delete);
82             if (result == -1)
83             {
84                 // printMessage("The PCB could not be successfully deleted\n");
85             }
86             else
87             {
88                 printMessage("The desired PCB was deleted\n");
89             }
90         }
91     }
92 }
```

5.26.1.4 resumePCB()

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

Definition at line 160 of file R2commands.c.

```
161 { // COLTON WILL PROGRAM THIS FUNCTION
162     /*
163     Places a PCB in the not suspended state and reinserts it into the appropriate queue
164     */
169
170     PCB *PCBtoResume = findPCB(processName);
171
172     if (PCBtoResume == NULL || strlen(processName) > 20)
173     {
174         printMessage("This is not a valid name.\n");
175     }
176     else
177     {
178         removePCB(PCBtoResume);
179         PCBtoResume->suspendedStatus = 1;
```

```
180         insertPCB(PCBtoResume);
181
182         printMessage("The PCB was successfully resumed!\n");
183     }
184 }
```

5.26.1.5 setPCBPriority()

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

Definition at line 186 of file R2commands.c.

```
187 { // ANASTASE WILL PROGRAM THIS FUNCTION
188
189     // Sets a PCB's priority and reinserts the process into the correct place in the correct queue
190
191     /*
192     Error Checking:
193     Name must be valid.
194     newPriority
195     */
196
197     // find the process and validate the name
198     PCB *tempPCB = findPCB(processName);
199
200     if ((tempPCB != NULL) && (newProcessPriority >= 0) && (newProcessPriority < 10))
201     {
202         tempPCB->priority = newProcessPriority;
203         removePCB(tempPCB);
204         insertPCB(tempPCB);
205
206         printMessage("The PCB's priority was successfully changed!\n");
207     }
208 }
```

5.26.1.6 showAll()

```
void showAll ( )
```

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 and blocked queues:
434         Process Name
435         Class
436         State
437         Suspended Status
438         Priority
439     */
440     /*
441     Error Checking:
442     None
443     */
444     showReady();
445     printMessage("\n");
446
447     showSuspendedReady();
448     printMessage("\n");
449
450     showBlocked();
451     printMessage("\n");
452
453     showSuspendedBlocked();
454     printMessage("\n");
455 }
```

5.26.1.7 showBlocked()

```
void showBlocked ( )
```

Definition at line 410 of file R2commands.c.

```
411 { // ANASTASE WILL PROGRAM THIS FUNCTION
412     /*
413     Displays the following information for each PCB in the blocked queue:
414         Process Name
415         Class
416         State
417         Suspended Status
418         Priority
419         HEAD
420     */
421     /*
422     Error Checking:
423     None
424     */
425
426     printMessage("The blocked queue:\n");
427     showQueue(getBlocked()->head, getBlocked()->count);
428 }
```

5.26.1.8 showPCB()

```
void showPCB (
            char * processName )
```

Definition at line 210 of file R2commands.c.

```
211 { // BENJAMIN WILL PROGRAM THIS FUNCTION
212     /*
213     Displays the following information for a PCB:
214         Process Name
215         Class
216         State
217         Suspended Status
218         Priority
219     */
220
221     /*
222     Error Checking:
223     Name must be valid.
224     */
225
226     if (strlen(processName) > 20)
227     { // Check if the process has a valid name.
228         printMessage("The PCB could not be shown as the name is longer than 20 characters!\n");
229     }
230     else
231     {
232         PCB *PCB_to_show = findPCB(processName);
233
234         if (PCB_to_show == NULL)
235         { // Check to see if the PCB exists.
236             printMessage("The PCB could not be shown, as it does not exist!\n");
237         }
238         else
239         {
240             // Print out the PCB name.
241             printMessage("The process name is: ");
242             int length = strlen(PCB_to_show->processName);
243             sys_req(WRITE, DEFAULT_DEVICE, PCB_to_show->processName, &length);
244             printMessage("\n");
245
246             // Print out PCB class
247             printMessage("The process class is: ");
248
249             if (PCB_to_show->processClass == 'a')
250             {
251                 printMessage("application.\n");
252             }
253             else
254             {
255                 printMessage("system.\n");
```

```
256             }
257
258             // Print out the PCB state
259
260             if (PCB_to_show->runningStatus == 0)
261             { // The process is ready.
262                 printMessage("The process is ready!\n");
263             }
264             else if (PCB_to_show->runningStatus == -1)
265             { // The process is blocked.
266                 printMessage("The process is blocked!\n");
267             }
268             else if (PCB_to_show->runningStatus == 1)
269             { // The process is running.
270                 printMessage("The process is running!\n");
271             }
272
273             // Print out the PCB suspended status
274
275             if (PCB_to_show->suspendedStatus == 0)
276             { // The process is suspended
277                 printMessage("The process is suspended!\n");
278             }
279             else if (PCB_to_show->suspendedStatus == 1)
280             { // The process is not suspended
281                 printMessage("The process is not suspended!\n");
282             }
283
284             // Print out the PCB priority
285             switch (PCB_to_show->priority)
286             {
287             case 0:
288                 printMessage("The process priority is 0!\n");
289                 break;
290
291             case 1:
292                 printMessage("The process priority is 1!\n");
293                 break;
294
295             case 2:
296                 printMessage("The process priority is 2!\n");
297                 break;
298
299             case 3:
300                 printMessage("The process priority is 3!\n");
301                 break;
302
303             case 4:
304                 printMessage("The process priority is 4!\n");
305                 break;
306
307             case 5:
308                 printMessage("The process priority is 5!\n");
309                 break;
310
311             case 6:
312                 printMessage("The process priority is 6!\n");
313                 break;
314
315             case 7:
316                 printMessage("The process priority is 7!\n");
317                 break;
318
319             case 8:
320                 printMessage("The process priority is 8!\n");
321                 break;
322
323             case 9:
324                 printMessage("The process priority is 9!\n");
325                 break;
326
327             default:
328                 break;
329             }
330         }
331     }
332 }
```

5.26.1.9 showQueue()

```
void showQueue (
```

```
            PCB * pcb,
            int count )
```

Definition at line 334 of file R2commands.c.

```
335 {
336     if (count == 0)
337     {
338         // the queue is empty
339         printMessage("The queue is empty.\n");
340         return;
341     }
342     // The queue is not empty
343 
344     int value;
345     for (value = 0; value < count; value++)
346     {
347         // Print out the process
348         showPCB(pcb->processName);
349         pcb = pcb->nextPCB;
350     }
351 }
```

5.26.1.10 showReady()

```
void showReady ( )
```

Definition at line 353 of file R2commands.c.

```
354 { // COLTON WILL PROGRAM THIS FUNCTION
355     /*
356     Displays the following information for each PCB in the ready queue:
357         Process Name
358         Class
359         State
360         Suspended Status
361         Priority
362     */
363     /*
364     Error Checking:
365     None
366     */
367 
368     printMessage("The ready queue:\n");
369     showQueue(getReady()->head, getReady()->count);
370 }
```

5.26.1.11 showSuspendedBlocked()

```
void showSuspendedBlocked ( )
```

Definition at line 391 of file R2commands.c.

```
392 { // COLTON WILL PROGRAM THIS FUNCTION
393     /*
394     Displays the following information for each PCB in the suspended blocked queue:
395         Process Name
396         Class
397         State
398         Suspended Status
399         Priority
400     */
401     /*
402     Error Checking:
403     None
404     */
405 
406     printMessage("The suspended blocked queue:\n");
407     showQueue(getSuspendedBlocked()->head, getSuspendedBlocked()->count);
408 }
```

5.26.1.12 showSuspendedReady()

```
void showSuspendedReady ( )
```

Definition at line 372 of file R2commands.c.

```
373 { // COLTON WILL PROGRAM THIS FUNCTION
374     /*
375     Displays the following information for each PCB in the suspended ready queue:
376         Process Name
377         Class
378         State
379         Suspended Status
380         Priority
381     */
382     /*
383     Error Checking:
384     None
385     */
386 
387     printMessage("The suspended ready queue:\n");
388     showQueue(getSuspendedReady()->head, getSuspendedReady()->count);
389 }
```

5.26.1.13 suspendPCB()

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

Definition at line 134 of file R2commands.c.

```
135 { // COLTON WILL PROGRAM THIS FUNCTION
136     /*
137     Places a PCB in the suspended state and reinserts it into the appropriate queue
138     */
143 
144     PCB *PCBtoSuspend = findPCB(processName);
145 
146     if (PCBtoSuspend == NULL || strlen(processName) > 20)
147     {
148         printMessage("This is not a valid name.\n");
149     }
150     else
151     {
152         removePCB(PCBtoSuspend);
153         PCBtoSuspend->suspendedStatus = 0;
154         insertPCB(PCBtoSuspend);
155 
156         printMessage("The PCB was successfully suspended!\n");
157     }
158 }
```

5.26.1.14 unblockPCB()

```
void unblockPCB (
            char * processName )
```

Definition at line 110 of file R2commands.c.

```
111 { // ANASTASE WILL PROGRAM THIS FUNCTION
112 
113     /*
114     Places a PCB in the unblocked state and reinserts it into the appropriate queue.
115     */
116     /*
117     Error Checking:
118     Name must be valid.
119 
120     */
```

```
121
122     PCB *pcb_to_unblock = findPCB(processName);
123     if (pcb_to_unblock != NULL)
124     {
125         pcb_to_unblock->runningStatus = 0; // ready
126         removePCB(pcb_to_unblock);         // is this the right place to put that function?
127         insertPCB(pcb_to_unblock);
128
129         printMessage("The PCB was successfully unblocked!\n");
130     }
131 }
```

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 94 of file R2commands.c.

```
95 { // ANASTASE WILL PROGRAM THIS FUNCTION
96
97     // find pcb and validate process name
98     PCB *pcb_to_block = findPCB(processName);
99
100     if (pcb_to_block != NULL)
101     {
102         pcb_to_block->runningStatus = -1; // blocked
103         removePCB(pcb_to_block);
104         insertPCB(pcb_to_block);
105
106         printMessage("The PCB was successfully blocked!\n");
107     }
108 }
```

5.27.1.2 createPCB()

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

Definition at line 12 of file R2commands.c.

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

5.27.1.3 deletePCB()

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

Definition at line 46 of file R2commands.c.

```
47 { // BENJAMIN WILL PROGRAM THIS FUNCTION
48     /*
49     The deletePCB command will remove a PCB from the appropriate queue and then free all associated
       memory.
50     This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
51     */
52     /*
53     Error Checking:
54     Name must be valid.
55     */
56
57     if (strlen(processName) > 20)
58     { // Check if the process has a valid name.
59         printMessage("The PCB could not be deleted as the name is longer than 20 characters!\n");
60     }
61
62     PCB *PCB_to_delete = findPCB(processName);
63
64     if (PCB_to_delete == NULL)
65     {
66         printMessage("The PCB you want to remove does not exist\n");
67     }
68     else if (strcmp(processName, "infinite") == 0 && PCB_to_delete->suspendedStatus != 0)
69     {
70         printMessage("In order to delete the infinite process it must be suspended first.\n");
71     }
```

```
72      else
73      {
74          int removed = removePCB(PCB_to_delete);
75          if (removed == 1)
76          {
77              printMessage("The PCB could not be unlinked.\n");
78          }
79          else
80          {
81              int result = sys_free_mem(PCB_to_delete);
82              if (result == -1)
83              {
84                  // printMessage("The PCB could not be successfully deleted\n");
85              }
86              else
87              {
88                  printMessage("The desired PCB was deleted\n");
89              }
90          }
91      }
92  }
```

5.27.1.4 resumePCB()

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

Definition at line 160 of file R2commands.c.

```
161 { // COLTON WILL PROGRAM THIS FUNCTION
162     /*
163     Places a PCB in the not suspended state and reinserts it into the appropriate queue
164     */
169
170     PCB *PCBtoResume = findPCB(processName);
171
172     if (PCBtoResume == NULL || strlen(processName) > 20)
173     {
174         printMessage("This is not a valid name.\n");
175     }
176     else
177     {
178         removePCB(PCBtoResume);
179         PCBtoResume->suspendedStatus = 1;
180         insertPCB(PCBtoResume);
181
182         printMessage("The PCB was successfully resumed!\n");
183     }
184 }
```

5.27.1.5 setPCBPriority()

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

Definition at line 186 of file R2commands.c.

```
187 { // ANASTASE WILL PROGRAM THIS FUNCTION
188
189     // Sets a PCB's priority and reinserts the process into the correct place in the correct queue
190
191     /*
192     Error Checking:
193     Name must be valid.
194     newPriority
195     */
196
197     // find the process and validate the name
198     PCB *tempPCB = findPCB(processName);
199
```

```
200     if ((tempPCB != NULL) && (newProcessPriority >= 0) && (newProcessPriority < 10))
201     {
202         tempPCB->priority = newProcessPriority;
203         removePCB(tempPCB);
204         insertPCB(tempPCB);
205 
206         printMessage("The PCB's priority was successfully changed!\n");
207     }
208 }
```

5.27.1.6 showAll()

```
void showAll ( )
```

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 and blocked queues:
434         Process Name
435         Class
436         State
437         Suspended Status
438         Priority
439     */
440     /*
441     Error Checking:
442     None
443     */
444     showReady();
445     printMessage("\n");
446 
447     showSuspendedReady();
448     printMessage("\n");
449 
450     showBlocked();
451     printMessage("\n");
452 
453     showSuspendedBlocked();
454     printMessage("\n");
455 }
```

5.27.1.7 showBlocked()

```
void showBlocked ( )
```

Definition at line 410 of file R2commands.c.

```
411 { // ANASTASE WILL PROGRAM THIS FUNCTION
412     /*
413     Displays the following information for each PCB in the blocked queue:
414         Process Name
415         Class
416         State
417         Suspended Status
418         Priority
419         HEAD
420     */
421     /*
422     Error Checking:
423     None
424     */
425 
426     printMessage("The blocked queue:\n");
427     showQueue(getBlocked()->head, getBlocked()->count);
428 }
```

5.27.1.8 showPCB()

```
void showPCB (
            char * processName )
```

Definition at line 210 of file R2commands.c.

```
211 { // BENJAMIN WILL PROGRAM THIS FUNCTION
212     /*
213     Displays the following information for a PCB:
214         Process Name
215         Class
216         State
217         Suspended Status
218         Priority
219     */
220 
221     /*
222     Error Checking:
223     Name must be valid.
224     */
225 
226     if (strlen(processName) > 20)
227     { // Check if the process has a valid name.
228         printMessage("The PCB could not be shown as the name is longer than 20 characters!\n");
229     }
230     else
231     {
232         PCB *PCB_to_show = findPCB(processName);
233 
234         if (PCB_to_show == NULL)
235         { // Check to see if the PCB exists.
236             printMessage("The PCB could not be shown, as it does not exist!\n");
237         }
238         else
239         {
240             // Print out the PCB name.
241             printMessage("The process name is: ");
242             int length = strlen(PCB_to_show->processName);
243             sys_req(WRITE, DEFAULT_DEVICE, PCB_to_show->processName, &length);
244             printMessage("\n");
245 
246             // Print out PCB class
247             printMessage("The process class is: ");
248 
249             if (PCB_to_show->processClass == 'a')
250             {
251                 printMessage("application.\n");
252             }
253             else
254             {
255                 printMessage("system.\n");
256             }
257 
258             // Print out the PCB state
259 
260             if (PCB_to_show->runningStatus == 0)
261             { // The process is ready.
262                 printMessage("The process is ready!\n");
263             }
264             else if (PCB_to_show->runningStatus == -1)
265             { // The process is blocked.
266                 printMessage("The process is blocked!\n");
267             }
268             else if (PCB_to_show->runningStatus == 1)
269             { // The process is running.
270                 printMessage("The process is running!\n");
271             }
272 
273             // Print out the PCB suspended status
274 
275             if (PCB_to_show->suspendedStatus == 0)
276             { // The process is suspended
277                 printMessage("The process is suspended!\n");
278             }
279             else if (PCB_to_show->suspendedStatus == 1)
280             { // The process is not suspended
281                 printMessage("The process is not suspended!\n");
282             }
283 
284             // Print out the PCB priority
285             switch (PCB_to_show->priority)
286             {
287             case 0:
288                 printMessage("The process priority is 0!\n");
```

```
289                 break;
290
291             case 1:
292                 printMessage("The process priority is 1!\n");
293                 break;
294
295             case 2:
296                 printMessage("The process priority is 2!\n");
297                 break;
298
299             case 3:
300                 printMessage("The process priority is 3!\n");
301                 break;
302
303             case 4:
304                 printMessage("The process priority is 4!\n");
305                 break;
306
307             case 5:
308                 printMessage("The process priority is 5!\n");
309                 break;
310
311             case 6:
312                 printMessage("The process priority is 6!\n");
313                 break;
314
315             case 7:
316                 printMessage("The process priority is 7!\n");
317                 break;
318
319             case 8:
320                 printMessage("The process priority is 8!\n");
321                 break;
322
323             case 9:
324                 printMessage("The process priority is 9!\n");
325                 break;
326
327             default:
328                 break;
329             }
330         }
331     }
332 }
```

5.27.1.9 showReady()

```
void showReady ( )
```

Definition at line 353 of file R2commands.c.

```
354 { // COLTON WILL PROGRAM THIS FUNCTION
355     /*
356     Displays the following information for each PCB in the ready queue:
357         Process Name
358         Class
359         State
360         Suspended Status
361         Priority
362     */
363     /*
364     Error Checking:
365     None
366     */
367
368     printMessage("The ready queue:\n");
369     showQueue(getReady()->head, getReady()->count);
370 }
```

5.27.1.10 showSuspendedBlocked()

```
void showSuspendedBlocked ( )
```

Definition at line 391 of file R2commands.c.

```
392 { // COLTON WILL PROGRAM THIS FUNCTION
393     /*
394     Displays the following information for each PCB in the suspended blocked queue:
395         Process Name
396         Class
397         State
398         Suspended Status
399         Priority
400     */
401     /*
402     Error Checking:
403     None
404     */
405 
406     printMessage("The suspended blocked queue:\n");
407     showQueue(getSuspendedBlocked()->head, getSuspendedBlocked()->count);
408 }
```

5.27.1.11 showSuspendedReady()

```
void showSuspendedReady ( )
```

Definition at line 372 of file R2commands.c.

```
373 { // COLTON WILL PROGRAM THIS FUNCTION
374     /*
375     Displays the following information for each PCB in the suspended ready queue:
376         Process Name
377         Class
378         State
379         Suspended Status
380         Priority
381     */
382     /*
383     Error Checking:
384     None
385     */
386 
387     printMessage("The suspended ready queue:\n");
388     showQueue(getSuspendedReady()->head, getSuspendedReady()->count);
389 }
```

5.27.1.12 suspendPCB()

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

Definition at line 134 of file R2commands.c.

```
135 { // COLTON WILL PROGRAM THIS FUNCTION
136     /*
137     Places a PCB in the suspended state and reinserts it into the appropriate queue
138     */
143 
144     PCB *PCBtoSuspend = findPCB(processName);
145 
146     if (PCBtoSuspend == NULL || strlen(processName) > 20)
147     {
148         printMessage("This is not a valid name.\n");
149     }
150     else
151     {
152         removePCB(PCBtoSuspend);
153         PCBtoSuspend->suspendedStatus = 0;
154         insertPCB(PCBtoSuspend);
155 
156         printMessage("The PCB was successfully suspended!\n");
157     }
158 }
```

5.27.1.13 unblockPCB()

```
void unblockPCB (
            char * processName )
```

Definition at line 110 of file R2commands.c.

```
111 { // ANASTASE WILL PROGRAM THIS FUNCTION
112
113     /*
114     Places a PCB in the unblocked state and reinserts it into the appropriate queue.
115     */
116     /*
117     Error Checking:
118     Name must be valid.
119
120     */
121
122     PCB *pcb_to_unblock = findPCB(processName);
123     if (pcb_to_unblock != NULL)
124     {
125         pcb_to_unblock->runningStatus = 0; // ready
126         removePCB(pcb_to_unblock);         // is this the right place to put that function?
127         insertPCB(pcb_to_unblock);
128
129         printMessage("The PCB was successfully unblocked!\n");
130     }
131 }
```

5.28 modules/R3/procsr3.c File Reference

```
#include "../include/system.h"
#include "../include/core/serial.h"
#include "../modules/mpx_supt.h"
#include "procsr3.h"
```

Macros

- #define RC_1 1
- #define RC_2 2
- #define RC_3 3
- #define RC_4 4
- #define RC_5 5

Functions

- void proc1 ()
- void proc2 ()
- void proc3 ()
- void proc4 ()
- void proc5 ()

Variables

- char * msg1 = "proc1 dispatched\n"
- char * msg2 = "proc2 dispatched\n"
- char * msg3 = "proc3 dispatched\n"
- char * msg4 = "proc4 dispatched\n"
- char * msg5 = "proc5 dispatched\n"
- int msgSize = 17
- char * er1 = "proc1 ran after it was terminated\n"
- char * er2 = "proc2 ran after it was terminated\n"
- char * er3 = "proc3 ran after it was terminated\n"
- char * er4 = "proc4 ran after it was terminated\n"
- char * er5 = "proc5 ran after it was terminated\n"
- int erSize = 34

5.28.1 Macro Definition Documentation

5.28.1.1 RC_1

```
#define RC_1 1
```

Definition at line 7 of file procsr3.c.

5.28.1.2 RC_2

```
#define RC_2 2
```

Definition at line 8 of file procsr3.c.

5.28.1.3 RC_3

```
#define RC_3 3
```

Definition at line 9 of file procsr3.c.

5.28.1.4 RC_4

```
#define RC_4 4
```

Definition at line 10 of file procsr3.c.

5.28.1.5 RC_5

```
#define RC_5 5
```

Definition at line 11 of file procsr3.c.

5.28.2 Function Documentation

5.28.2.1 proc1()

```
void proc1 ( )
```

Definition at line 27 of file procsr3.c.

```
28 {
29   int i;
30
31   // repeat forever if termination fails
32   while (1)
33   {
34     for (i = 0; i < RC_1; i++)
35     {
36       sys_req(WRITE, DEFAULT_DEVICE, msg1, &msgSize);
37       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
38     }
39     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
40     sys_req(WRITE, DEFAULT_DEVICE, er1, &erSize);
41   }
42 }
```

5.28.2.2 proc2()

```
void proc2 ( )
```

Definition at line 44 of file procsr3.c.

```
45 {
46   int i;
47
48   // repeat forever if termination fails
49   while (1)
50   {
51     for (i = 0; i < RC_2; i++)
52     {
53       sys_req(WRITE, DEFAULT_DEVICE, msg2, &msgSize);
54       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
55     }
56     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
57     sys_req(WRITE, DEFAULT_DEVICE, er2, &erSize);
58   }
59 }
```

5.28.2.3 proc3()

```
void proc3 ( )
```

Definition at line 61 of file procsr3.c.

```
62 {
63   int i;
64
65   // repeat forever if termination fails
66   while (1)
67   {
68     for (i = 0; i < RC_3; i++)
69     {
70       sys_req(WRITE, DEFAULT_DEVICE, msg3, &msgSize);
71       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
72     }
73     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
74     sys_req(WRITE, DEFAULT_DEVICE, er3, &erSize);
75   }
76 }
```

5.28.2.4 proc4()

```
void proc4 ( )
```

Definition at line 78 of file procsr3.c.

```
79 {
80   int i;
81
82   // repeat forever if termination fails
83   while (1)
84   {
85     for (i = 0; i < RC_4; i++)
86     {
87       sys_req(WRITE, DEFAULT_DEVICE, msg4, &msgSize);
88       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
89     }
90     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
91     sys_req(WRITE, DEFAULT_DEVICE, er4, &erSize);
92   }
93 }
```

5.28.2.5 proc5()

```
void proc5 ( )
```

Definition at line 95 of file procsr3.c.

```
96 {
97   int i;
98
99   // repeat forever if termination fails
100   while (1)
101   {
102     for (i = 0; i < RC_5; i++)
103     {
104       sys_req(WRITE, DEFAULT_DEVICE, msg5, &msgSize);
105       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
106     }
107     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
108     sys_req(WRITE, DEFAULT_DEVICE, er5, &erSize);
109   }
110 }
```

5.28.3 Variable Documentation

5.28.3.1 er1

```
char* er1 = "proc1 ran after it was terminated\n"
```

Definition at line 20 of file procsr3.c.

5.28.3.2 er2

```
char* er2 = "proc2 ran after it was terminated\n"
```

Definition at line 21 of file procsr3.c.

5.28.3.3 er3

```
char* er3 = "proc3 ran after it was terminated\n"
```

Definition at line 22 of file procsr3.c.

5.28.3.4 er4

```
char* er4 = "proc4 ran after it was terminated\n"
```

Definition at line 23 of file procsr3.c.

5.28.3.5 er5

```
char* er5 = "proc5 ran after it was terminated\n"
```

Definition at line 24 of file procsr3.c.

5.28.3.6 erSize

```
int erSize = 34
```

Definition at line 25 of file procsr3.c.

5.28.3.7 msg1

```
char* msg1 = "proc1 dispatched\n"
```

Definition at line 13 of file procsr3.c.

5.28.3.8 msg2

```
char* msg2 = "proc2 dispatched\n"
```

Definition at line 14 of file procsr3.c.

5.28.3.9 msg3

```
char* msg3 = "proc3 dispatched\n"
```

Definition at line 15 of file procsr3.c.

5.28.3.10 msg4

```
char* msg4 = "proc4 dispatched\n"
```

Definition at line 16 of file procsr3.c.

5.28.3.11 msg5

```
char* msg5 = "proc5 dispatched\n"
```

Definition at line 17 of file procsr3.c.

5.28.3.12 msgSize

```
int msgSize = 17
```

Definition at line 18 of file procsr3.c.

5.29 modules/R3/procsr3.h File Reference

Functions

- void proc1 ()
- void proc2 ()
- void proc3 ()
- void proc4 ()
- void proc5 ()

5.29.1 Function Documentation

5.29.1.1 proc1()

```
void proc1 ( )
```

Definition at line 27 of file procsr3.c.

```
28 {
29   int i;
30
31   // repeat forever if termination fails
32   while (1)
33   {
34     for (i = 0; i < RC_1; i++)
35     {
36       sys_req(WRITE, DEFAULT_DEVICE, msg1, &msgSize);
37       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
38     }
39     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
40     sys_req(WRITE, DEFAULT_DEVICE, er1, &erSize);
41   }
42 }
```

5.29.1.2 proc2()

```
void proc2 ( )
```

Definition at line 44 of file procsr3.c.

```
45 {
46   int i;
47
48   // repeat forever if termination fails
49   while (1)
50   {
51     for (i = 0; i < RC_2; i++)
52     {
53       sys_req(WRITE, DEFAULT_DEVICE, msg2, &msgSize);
54       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
55     }
56     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
57     sys_req(WRITE, DEFAULT_DEVICE, er2, &erSize);
58   }
59 }
```

5.29.1.3 proc3()

```
void proc3 ( )
```

Definition at line 61 of file procsr3.c.

```
62 {
63   int i;
64
65   // repeat forever if termination fails
66   while (1)
67   {
68     for (i = 0; i < RC_3; i++)
69     {
70       sys_req(WRITE, DEFAULT_DEVICE, msg3, &msgSize);
71       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
72     }
73     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
74     sys_req(WRITE, DEFAULT_DEVICE, er3, &erSize);
75   }
76 }
```

5.29.1.4 proc4()

```
void proc4 ( )
```

Definition at line 78 of file procsr3.c.

```
79 {
80   int i;
81
82   // repeat forever if termination fails
83   while (1)
84   {
85     for (i = 0; i < RC_4; i++)
86     {
87       sys_req(WRITE, DEFAULT_DEVICE, msg4, &msgSize);
88       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
89     }
90     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
91     sys_req(WRITE, DEFAULT_DEVICE, er4, &erSize);
92   }
93 }
```

5.29.1.5 proc5()

```
void proc5 ( )
```

Definition at line 95 of file procsr3.c.

```
96 {
97   int i;
98
99   // repeat forever if termination fails
100   while (1)
101   {
102     for (i = 0; i < RC_5; i++)
103     {
104       sys_req(WRITE, DEFAULT_DEVICE, msg5, &msgSize);
105       sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
106     }
107     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
108     sys_req(WRITE, DEFAULT_DEVICE, er5, &erSize);
109   }
110 }
```

5.30 modules/R3/R3commands.c File Reference

```
#include <string.h>
#include "../mpx_supt.h"
#include <core/serial.h>
#include "../R1/R1commands.h"
#include "../R2/R2_Internal_Functions_And_Structures.h"
#include "../R2/R2commands.h"
#include "R3commands.h"
#include "procsr3.h"
```

Functions

- void yield ()
- void loadr3 ()

5.30.1 Function Documentation

5.30.1.1 loadr3()

```
void loadr3 ( )
```

Definition at line 18 of file R3commands.c.

```
19 {
20     //loadr3 will load all r3 "processes" (proc3.c file eCampus) into memory in a suspended ready state
       at any priority of your choosing.
21     // We may want to change these to use setupPCB instead of createPCB and suspendPCB
22     printMessage("Loading R3 Processes.\n\n");
23
24     createPCB("Process1", 'a', 1);
25     suspendPCB("Process1");
26     PCB *new_pcb1 = findPCB("Process1");
27     context *cp1 = (context *)(new_pcb1->stackTop);
28     memset(cp1, 0, sizeof(context));
29     cp1->fs = 0x10;
30     cp1->gs = 0x10;
31     cp1->ds = 0x10;
32     cp1->es = 0x10;
33     cp1->cs = 0x8;
34     cp1->ebp = (u32int)(new_pcb1->stack);
35     cp1->esp = (u32int)(new_pcb1->stackTop);
36     cp1->eip = (u32int)proc1; // The function correlating to the process, ie. Proc1
37     cp1->eflags = 0x202;
38
39     createPCB("Process2", 'a', 1);
40     suspendPCB("Process2");
41     PCB *new_pcb2 = findPCB("Process2");
42     context *cp2 = (context *)(new_pcb2->stackTop);
43     memset(cp2, 0, sizeof(context));
44     cp2->fs = 0x10;
45     cp2->gs = 0x10;
46     cp2->ds = 0x10;
47     cp2->es = 0x10;
48     cp2->cs = 0x8;
49     cp2->ebp = (u32int)(new_pcb2->stack);
50     cp2->esp = (u32int)(new_pcb2->stackTop);
51     cp2->eip = (u32int)proc2; // The function correlating to the process, ie. Proc1
52     cp2->eflags = 0x202;
53
54     createPCB("Process3", 'a', 1);
55     suspendPCB("Process3");
56     PCB *new_pcb3 = findPCB("Process3");
57     context *cp3 = (context *)(new_pcb3->stackTop);
```

```
58      memset(cp3, 0, sizeof(context));
59      cp3->fs = 0x10;
60      cp3->gs = 0x10;
61      cp3->ds = 0x10;
62      cp3->es = 0x10;
63      cp3->cs = 0x8;
64      cp3->ebp = (u32int)(new_pcb3->stack);
65      cp3->esp = (u32int)(new_pcb3->stackTop);
66      cp3->eip = (u32int)proc3; // The function correlating to the process, ie. Proc1
67      cp3->eflags = 0x202;
68
69      createPCB("Process4", 'a', 1);
70      suspendPCB("Process4");
71      PCB *new_pcb4 = findPCB("Process4");
72      context *cp4 = (context *)(new_pcb4->stackTop);
73      memset(cp4, 0, sizeof(context));
74      cp4->fs = 0x10;
75      cp4->gs = 0x10;
76      cp4->ds = 0x10;
77      cp4->es = 0x10;
78      cp4->cs = 0x8;
79      cp4->ebp = (u32int)(new_pcb4->stack);
80      cp4->esp = (u32int)(new_pcb4->stackTop);
81      cp4->eip = (u32int)proc4; // The function correlating to the process, ie. Proc1
82      cp4->eflags = 0x202;
83
84      createPCB("Process5", 'a', 1);
85      suspendPCB("Process5");
86      PCB *new_pcb5 = findPCB("Process5");
87      context *cp5 = (context *)(new_pcb5->stackTop);
88      memset(cp5, 0, sizeof(context));
89      cp5->fs = 0x10;
90      cp5->gs = 0x10;
91      cp5->ds = 0x10;
92      cp5->es = 0x10;
93      cp5->cs = 0x8;
94      cp5->ebp = (u32int)(new_pcb5->stack);
95      cp5->esp = (u32int)(new_pcb5->stackTop);
96      cp5->eip = (u32int)proc5; // The function correlating to the process, ie. Proc1
97      cp5->eflags = 0x202;
98  }
```

5.30.1.2 yield()

```
void yield ( )
```

Definition at line 13 of file R3commands.c.

```
14  { // temporary command - only in R3
15      asm volatile("int $60");
16  }
```

5.31 modules/R3/R3commands.h File Reference

Classes

- struct context

Typedefs

- typedef struct context context

Functions

- void yield ()
- void loadr3 ()

5.31.1 Typedef Documentation

5.31.1.1 context

```
typedef struct context context
```

5.31.2 Function Documentation

5.31.2.1 loadr3()

```
void loadr3 ( )
```

Definition at line 18 of file R3commands.c.

```
19 {
20     //loadr3 will load all r3 "processes" (proc3.c file eCampus) into memory in a suspended ready state
       at any priority of your choosing.
21     // We may want to change these to use setupPCB instead of createPCB and suspendPCB
22     printMessage("Loading R3 Processes.\n\n");
23
24     createPCB("Process1", 'a', 1);
25     suspendPCB("Process1");
26     PCB *new_pcb1 = findPCB("Process1");
27     context *cp1 = (context *)(new_pcb1->stackTop);
28     memset(cp1, 0, sizeof(context));
29     cp1->fs = 0x10;
30     cp1->gs = 0x10;
31     cp1->ds = 0x10;
32     cp1->es = 0x10;
33     cp1->cs = 0x8;
34     cp1->ebp = (u32int)(new_pcb1->stack);
35     cp1->esp = (u32int)(new_pcb1->stackTop);
36     cp1->eip = (u32int)proc1; // The function correlating to the process, ie. Proc1
37     cp1->eflags = 0x202;
38
39     createPCB("Process2", 'a', 1);
40     suspendPCB("Process2");
41     PCB *new_pcb2 = findPCB("Process2");
42     context *cp2 = (context *)(new_pcb2->stackTop);
43     memset(cp2, 0, sizeof(context));
44     cp2->fs = 0x10;
45     cp2->gs = 0x10;
46     cp2->ds = 0x10;
47     cp2->es = 0x10;
48     cp2->cs = 0x8;
49     cp2->ebp = (u32int)(new_pcb2->stack);
50     cp2->esp = (u32int)(new_pcb2->stackTop);
51     cp2->eip = (u32int)proc2; // The function correlating to the process, ie. Proc1
52     cp2->eflags = 0x202;
53
54     createPCB("Process3", 'a', 1);
55     suspendPCB("Process3");
56     PCB *new_pcb3 = findPCB("Process3");
57     context *cp3 = (context *)(new_pcb3->stackTop);
58     memset(cp3, 0, sizeof(context));
59     cp3->fs = 0x10;
60     cp3->gs = 0x10;
61     cp3->ds = 0x10;
62     cp3->es = 0x10;
63     cp3->cs = 0x8;
64     cp3->ebp = (u32int)(new_pcb3->stack);
65     cp3->esp = (u32int)(new_pcb3->stackTop);
66     cp3->eip = (u32int)proc3; // The function correlating to the process, ie. Proc1
67     cp3->eflags = 0x202;
68
69     createPCB("Process4", 'a', 1);
70     suspendPCB("Process4");
```

```
71      PCB *new_pcb4 = findPCB("Process4");
72      context *cp4 = (context *)(new_pcb4->stackTop);
73      memset(cp4, 0, sizeof(context));
74      cp4->fs = 0x10;
75      cp4->gs = 0x10;
76      cp4->ds = 0x10;
77      cp4->es = 0x10;
78      cp4->cs = 0x8;
79      cp4->ebp = (u32int)(new_pcb4->stack);
80      cp4->esp = (u32int)(new_pcb4->stackTop);
81      cp4->eip = (u32int)proc4; // The function correlating to the process, ie. Proc1
82      cp4->eflags = 0x202;
83
84      createPCB("Process5", 'a', 1);
85      suspendPCB("Process5");
86      PCB *new_pcb5 = findPCB("Process5");
87      context *cp5 = (context *)(new_pcb5->stackTop);
88      memset(cp5, 0, sizeof(context));
89      cp5->fs = 0x10;
90      cp5->gs = 0x10;
91      cp5->ds = 0x10;
92      cp5->es = 0x10;
93      cp5->cs = 0x8;
94      cp5->ebp = (u32int)(new_pcb5->stack);
95      cp5->esp = (u32int)(new_pcb5->stackTop);
96      cp5->eip = (u32int)proc5; // The function correlating to the process, ie. Proc1
97      cp5->eflags = 0x202;
98  }
```

5.31.2.2 yield()

```
void yield ( )
```

Definition at line 13 of file R3commands.c.

```
14  { // temporary command - only in R3
15      asm volatile("int $60");
16  }
```

5.32 modules/R4/R4commands.c File Reference

```
#include <string.h>
#include "../mpx_supt.h"
#include <core/serial.h>
#include <core/io.h>
#include "../R2/R2_Internal_Functions_And_Structures.h"
#include "../R2/R2commands.h"
#include "../R3/R3commands.h"
#include "R4commands.h"
#include "../R1/R1commands.h"
```

Functions

- void alarmPCB ()
- void infinitePCB ()
- void infiniteFunc ()
- void allocateAlarmQueue ()
- alarm * allocateAlarms ()
- alarmList * getAlarms ()
- void addAlarm ()
- int convertTime (char *hours, char *minutes, char *seconds)
- void iterateAlarms ()

Variables

- alarmList * alarms

5.32.1 Function Documentation

5.32.1.1 addAlarm()

```
void addAlarm ( )
```

Definition at line 85 of file R4commands.c.

```
86 {
87
88     unblockPCB("Alarm");
89
90     printMessage("Please enter a name for the alarm you want to create.\n\n");
91
92     alarm *Alarm_to_insert = allocateAlarms();
93
94     int nameLength = strlen(Alarm_to_insert->alarmName);
95     sys_req(READ, DEFAULT_DEVICE, Alarm_to_insert->alarmName, &nameLength);
96
97     printMessage("Please type the desired hours. I.E.: hh.\n");
98
99     char hour[4] = "\0\0\n\0";
100
101     int flag = 0;
102
103     do
104     {
105         int hourLength = strlen(hour);
106         sys_req(READ, DEFAULT_DEVICE, hour, &hourLength);
107         if (atoi(hour) < 24 && atoi(hour) >= 0)
108         {
109
110             printMessage("\n");
111             flag = 0;
112         }
113         else
114         {
115             printMessage("\nInvalid hours.\n");
116
117             flag = 1;
118         }
119     } while (flag == 1);
120
122     printMessage("Please type the desired minutes. I.E.: mm.\n");
123
124     char minute[4] = "\0\0\n\0";
125
126     do
127     {
128         int minuteLength = strlen(minute);
129         sys_req(READ, DEFAULT_DEVICE, minute, &minuteLength);
130         if (atoi(minute) < 60 && atoi(minute) >= 0)
131         {
132             printMessage("\n");
133             flag = 0;
134         }
135         else
136         {
137             printMessage("\nInvalid minutes.\n");
138             flag = 1;
139         }
140     } while (flag == 1);
141
143     printMessage("Please type the desired seconds. I.E.: ss.\n");
144
145     char second[4] = "\0\0\n\0";
146
147     do
148     {
149         int secondLength = strlen(second);
150         sys_req(READ, DEFAULT_DEVICE, second, &secondLength);
```

```
151         if (atoi(second) < 60 && atoi(second) >= 0)
152         {
153 
154             printMessage("\n");
155             flag = 0;
156         }
157         else
158         {
159             printMessage("\nInvalid seconds.\n");
160             flag = 1;
161         }
162     } while (flag == 1);
163 
164     // Storing time in the alarm to insert
165     Alarm_to_insert->alarmTime = convertTime(hour, minute, second);
166 
167     // Inserting the alarm
168     if (getAlarms()->head != NULL)
169     {
170         getAlarms()->tail->nextAlarm = Alarm_to_insert;
171         Alarm_to_insert->prevAlarm = getAlarms()->tail;
172         getAlarms()->tail = Alarm_to_insert;
173         getAlarms()->count++;
174     }
175     else
176     {
177         getAlarms()->head = Alarm_to_insert;
178         getAlarms()->tail = Alarm_to_insert;
179         getAlarms()->count++;
180     }
181 }
```

5.32.1.2 alarmPCB()

```
void alarmPCB ( )
```

Definition at line 16 of file R4commands.c.

```
17 {
18     if (alarms->head == NULL && findPCB("Alarm")->runningStatus != -1)
19     {
20         blockPCB("Alarm");
21     }
22     else
23     {
24         iterateAlarms();
25     }
26 }
```

5.32.1.3 allocateAlarmQueue()

```
void allocateAlarmQueue ( )
```

Definition at line 56 of file R4commands.c.

```
57 {
58     alarms = sys_alloc_mem(sizeof(alarmList));
59     alarms->count = NULL;
60     alarms->head = NULL;
61     alarms->tail = NULL;
62 }
```

5.32.1.4 allocateAlarms()

```
alarm* allocateAlarms ( )
```

Definition at line 64 of file R4commands.c.

```
65 {
66     alarm *newAlarm = (alarm *)sys_alloc_mem(sizeof(alarm));
67 
68     char name[20] = "newAlarm";
69     strcpy(newAlarm->alarmName, name);
70 
71     newAlarm->alarmTime = 0;
72 
73     // Setting the alarms prev and next PCB
74     newAlarm->nextAlarm = NULL;
75     newAlarm->prevAlarm = NULL;
76 
77     return newAlarm;
78 }
```

5.32.1.5 convertTime()

```
int convertTime (
            char * hours,
            char * minutes,
            char * seconds )
```

Definition at line 183 of file R4commands.c.

```
184 {
185     int result = (atoi(hours) * 3600);
186     result += (atoi(minutes) * 60);
187     result += (atoi(seconds));
188 
189     return result;
190 }
```

5.32.1.6 getAlarms()

```
alarmList* getAlarms ( )
```

Definition at line 80 of file R4commands.c.

```
81 {
82     return alarms;
83 }
```

5.32.1.7 infiniteFunc()

```
void infiniteFunc ( )
```

Definition at line 45 of file R4commands.c.

```
46 {
47     while (1)
48     {
49 
50         printMessage("Infinite Process Executing.\n");
51 
52         sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
53     }
54 }
```

5.32.1.8 infinitePCB()

```
void infinitePCB ( )
```

Definition at line 28 of file R4commands.c.

```
29 {
30     createPCB("infinite", 'a', 1);
31     PCB *new_pcb = findPCB("infinite");
32     context *cp = (context *)(new_pcb->stackTop);
33     memset(cp, 0, sizeof(context));
34     cp->fs = 0x10;
35     cp->gs = 0x10;
36     cp->ds = 0x10;
37     cp->es = 0x10;
38     cp->cs = 0x8;
39     cp->ebp = (u32int)(new_pcb->stack);
40     cp->esp = (u32int)(new_pcb->stackTop);
41     cp->eip = (u32int)infiniteFunc; // The function correlating to the process, ie. Proc1
42     cp->eflags = 0x202;
43 }
```

5.32.1.9 iterateAlarms()

```
void iterateAlarms ( )
```

Definition at line 192 of file R4commands.c.

```
193 {
194     char hours[4] = "\0\0\0\0";
195     outb(0x70, 0x04); // getting current Hour value
196     BCDtoChar(inb(0x71), hours);
197 
198     char minutes[4] = "\0\0\0\0";
199     outb(0x70, 0x02); // getting current Minute value
200     BCDtoChar(inb(0x71), minutes);
201 
202     char seconds[4] = "\0\0\0\0";
203     outb(0x70, 0x00); // getting current Minute value
204     BCDtoChar(inb(0x71), seconds);
205 
206     int currentTime = convertTime(hours, minutes, seconds);
207 
208     alarm *tempAlarm = getAlarms()->head;
209 
210     while (tempAlarm != NULL)
211     {
212         if (currentTime >= getAlarms()->head->alarmTime)
213         {
214             // do something for alarm.
215             printMessage(getAlarms()->head->alarmName);
216             getAlarms()->head = getAlarms()->head->nextAlarm;
217         }
218         else if (currentTime >= getAlarms()->tail->alarmTime)
219         {
220             printMessage(getAlarms()->tail->alarmName);
221             getAlarms()->tail = getAlarms()->tail->prevAlarm;
222         }
223         else if (currentTime >= tempAlarm->alarmTime)
224         {
225             printMessage(tempAlarm->alarmName);
226             tempAlarm->prevAlarm->nextAlarm = tempAlarm->nextAlarm;
227             tempAlarm->nextAlarm->prevAlarm = tempAlarm->prevAlarm;
228             tempAlarm->nextAlarm = NULL;
229             tempAlarm->prevAlarm = NULL;
230         }
231         else
232         {
233             // iterates if not time
234             tempAlarm = tempAlarm->nextAlarm;
235         }
236     }
237 }
```

5.32.2 Variable Documentation

5.32.2.1 alarms

```
alarmList* alarms
```

Definition at line 14 of file R4commands.c.

5.33 modules/R4/R4commands.h File Reference

Classes

- struct alarm
- struct alarmList

Typedefs

- typedef struct alarm alarm
- typedef struct alarmList alarmList

Functions

- void alarmPCB ()
- void infinitePCB ()
- void infiniteFunc ()
- void allocateAlarmQueue ()
- alarm * allocateAlarms ()
- alarmList * getAlarms ()
- void addAlarm ()
- int convertTime (char *hours, char *minutes, char *seconds)
- void iterateAlarms ()

5.33.1 Typedef Documentation

5.33.1.1 alarm

```
typedef struct alarm alarm
```

5.33.1.2 alarmList

```
typedef struct alarmList alarmList
```

5.33.2 Function Documentation

5.33.2.1 addAlarm()

```
void addAlarm ( )
```

Definition at line 85 of file R4commands.c.

```
86 {
87
88     unblockPCB("Alarm");
89
90     printMessage("Please enter a name for the alarm you want to create.\n\n");
91
92     alarm *Alarm_to_insert = allocateAlarms();
93
94     int nameLength = strlen(Alarm_to_insert->alarmName);
95     sys_req(READ, DEFAULT_DEVICE, Alarm_to_insert->alarmName, &nameLength);
96
97     printMessage("Please type the desired hours. I.E.: hh.\n");
98
99     char hour[4] = "\0\0\n\0";
100
101     int flag = 0;
102
103     do
104     {
105         int hourLength = strlen(hour);
106         sys_req(READ, DEFAULT_DEVICE, hour, &hourLength);
107         if (atoi(hour) < 24 && atoi(hour) >= 0)
108         {
109
110             printMessage("\n");
111             flag = 0;
112         }
113         else
114         {
115             printMessage("\nInvalid hours.\n");
116
117             flag = 1;
118         }
119     } while (flag == 1);
120
122     printMessage("Please type the desired minutes. I.E.: mm.\n");
123
124     char minute[4] = "\0\0\n\0";
125
126     do
127     {
128         int minuteLength = strlen(minute);
129         sys_req(READ, DEFAULT_DEVICE, minute, &minuteLength);
130         if (atoi(minute) < 60 && atoi(minute) >= 0)
131         {
132             printMessage("\n");
133             flag = 0;
134         }
135         else
136         {
137             printMessage("\nInvalid minutes.\n");
138             flag = 1;
139         }
140     } while (flag == 1);
141
143     printMessage("Please type the desired seconds. I.E.: ss.\n");
144
145     char second[4] = "\0\0\n\0";
146
147     do
148     {
149         int secondLength = strlen(second);
150         sys_req(READ, DEFAULT_DEVICE, second, &secondLength);
151         if (atoi(second) < 60 && atoi(second) >= 0)
152         {
153
154             printMessage("\n");
155             flag = 0;
156         }
157         else
158         {
```

```
159             printMessage("\nInvalid seconds.\n");
160             flag = 1;
161         }
162     } while (flag == 1);
163 
164     // Storing time in the alarm to insert
165     Alarm_to_insert->alarmTime = convertTime(hour, minute, second);
166 
167     // Inserting the alarm
168     if (getAlarms()->head != NULL)
169     {
170         getAlarms()->tail->nextAlarm = Alarm_to_insert;
171         Alarm_to_insert->prevAlarm = getAlarms()->tail;
172         getAlarms()->tail = Alarm_to_insert;
173         getAlarms()->count++;
174     }
175     else
176     {
177         getAlarms()->head = Alarm_to_insert;
178         getAlarms()->tail = Alarm_to_insert;
179         getAlarms()->count++;
180     }
181 }
```

5.33.2.2 alarmPCB()

```
void alarmPCB ( )
```

Definition at line 16 of file R4commands.c.

```
17 {
18     if (alarms->head == NULL && findPCB("Alarm")->runningStatus != -1)
19     {
20         blockPCB("Alarm");
21     }
22     else
23     {
24         iterateAlarms();
25     }
26 }
```

5.33.2.3 allocateAlarmQueue()

```
void allocateAlarmQueue ( )
```

Definition at line 56 of file R4commands.c.

```
57 {
58     alarms = sys_alloc_mem(sizeof(alarmList));
59     alarms->count = NULL;
60     alarms->head = NULL;
61     alarms->tail = NULL;
62 }
```

5.33.2.4 allocateAlarms()

```
alarm* allocateAlarms ( )
```

Definition at line 64 of file R4commands.c.

```
65 {
66     alarm *newAlarm = (alarm *)sys_alloc_mem(sizeof(alarm));
67 
68     char name[20] = "newAlarm";
69     strcpy(newAlarm->alarmName, name);
70 
71     newAlarm->alarmTime = 0;
72 
73     // Setting the alarms prev and next PCB
74     newAlarm->nextAlarm = NULL;
75     newAlarm->prevAlarm = NULL;
76 
77     return newAlarm;
78 }
```

5.33.2.5 convertTime()

```
int convertTime (
            char * hours,
            char * minutes,
            char * seconds )
```

Definition at line 183 of file R4commands.c.

```
184 {
185     int result = (atoi(hours) * 3600);
186     result += (atoi(minutes) * 60);
187     result += (atoi(seconds));
188 
189     return result;
190 }
```

5.33.2.6 getAlarms()

```
alarmList* getAlarms ( )
```

Definition at line 80 of file R4commands.c.

```
81 {
82     return alarms;
83 }
```

5.33.2.7 infiniteFunc()

```
void infiniteFunc ( )
```

Definition at line 45 of file R4commands.c.

```
46 {
47     while (1)
48     {
49 
50         printMessage("Infinite Process Executing.\n");
51 
52         sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
53     }
54 }
```

5.33.2.8 infinitePCB()

```
void infinitePCB ( )
```

Definition at line 28 of file R4commands.c.

```
29 {
30     createPCB("infinite", 'a', 1);
31     PCB *new_pcb = findPCB("infinite");
32     context *cp = (context *)(new_pcb->stackTop);
33     memset(cp, 0, sizeof(context));
34     cp->fs = 0x10;
35     cp->gs = 0x10;
36     cp->ds = 0x10;
37     cp->es = 0x10;
38     cp->cs = 0x8;
39     cp->ebp = (u32int)(new_pcb->stack);
40     cp->esp = (u32int)(new_pcb->stackTop);
41     cp->eip = (u32int)infiniteFunc; // The function correlating to the process, ie. Proc1
42     cp->eflags = 0x202;
43 }
```

5.33.2.9 iterateAlarms()

```
void iterateAlarms ( )
```

Definition at line 192 of file R4commands.c.

```
193 {
194     char hours[4] = "\0\0\0\0";
195     outb(0x70, 0x04); // getting current Hour value
196     BCDtoChar(inb(0x71), hours);
197 
198     char minutes[4] = "\0\0\0\0";
199     outb(0x70, 0x02); // getting current Minute value
200     BCDtoChar(inb(0x71), minutes);
201 
202     char seconds[4] = "\0\0\0\0";
203     outb(0x70, 0x00); // getting current Minute value
204     BCDtoChar(inb(0x71), seconds);
205 
206     int currentTime = convertTime(hours, minutes, seconds);
207 
208     alarm *tempAlarm = getAlarms()->head;
209 
210     while (tempAlarm != NULL)
211     {
212         if (currentTime >= getAlarms()->head->alarmTime)
213         {
214             // do something for alarm.
215             printMessage(getAlarms()->head->alarmName);
216             getAlarms()->head = getAlarms()->head->nextAlarm;
217         }
218         else if (currentTime >= getAlarms()->tail->alarmTime)
219         {
220             printMessage(getAlarms()->tail->alarmName);
221             getAlarms()->tail = getAlarms()->tail->prevAlarm;
222         }
223         else if (currentTime >= tempAlarm->alarmTime)
224         {
225             printMessage(tempAlarm->alarmName);
226             tempAlarm->prevAlarm->nextAlarm = tempAlarm->nextAlarm;
227             tempAlarm->nextAlarm->prevAlarm = tempAlarm->prevAlarm;
228             tempAlarm->nextAlarm = NULL;
229             tempAlarm->prevAlarm = NULL;
230         }
231         else
232         {
233             // iterates if not time
234             tempAlarm = tempAlarm->nextAlarm;
235         }
236     }
237 }
```

5.34 modules/utilities.c File Reference

5.35 modules/utilities.h File Reference

5.36 README.md File Reference