MPX-Fall2020-Group9

2

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 date_time Struct Reference

`#include <system.h>`

Public Attributes

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

4.1.1 Detailed Description

Definition at line 30 of file system.h.

4.1.2 Member Data Documentation

4.1.2.1 day_m

`int date_time::day_m`

Definition at line 35 of file system.h.

4.1.2.2 day_w

```
int date_time::day_w
```

Definition at line 34 of file system.h.

4.1.2.3 day_y

```
int date_time::day_y
```

Definition at line 36 of file system.h.

4.1.2.4 hour

```
int date_time::hour
```

Definition at line 33 of file system.h.

4.1.2.5 min

```
int date_time::min
```

Definition at line 32 of file system.h.

4.1.2.6 mon

```
int date_time::mon
```

Definition at line 37 of file system.h.

4.1.2.7 sec

```
int date_time::sec
```

Definition at line 31 of file system.h.

4.1.2.8 year

```
int date_time::year
```

Definition at line 38 of file system.h.

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

- include/system.h

4.2 footer Struct Reference

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

Public Attributes

- header head

4.2.1 Detailed Description

Definition at line 16 of file heap.h.

4.2.2 Member Data Documentation

4.2.2.1 head

```
header footer::head
```

Definition at line 17 of file heap.h.

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

- include/mem/heap.h

4.3 gdt_descriptor_struct Struct Reference

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

Public Attributes

- u16int limit
- u32int base

4.3.1 Detailed Description

Definition at line 23 of file tables.h.

4.3.2 Member Data Documentation

4.3.2.1 base

```
u32int gdt_descriptor_struct::base
```

Definition at line 26 of file tables.h.

4.3.2.2 limit

```
u16int gdt_descriptor_struct::limit
```

Definition at line 25 of file tables.h.

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

- include/core/tables.h

4.4 gdt_entry_struct Struct Reference

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

Public Attributes

- u16int limit_low
- u16int base_low
- u8int base_mid
- u8int access
- u8int flags
- u8int base_high

4.4.1 Detailed Description

Definition at line 30 of file tables.h.

4.4.2 Member Data Documentation

4.4.2.1 access

```
u8int gdt_entry_struct::access
```

Definition at line 35 of file tables.h.

4.4.2.2 base_high

```
u8int gdt_entry_struct::base_high
```

Definition at line 37 of file tables.h.

4.4.2.3 base_low

```
u16int gdt_entry_struct::base_low
```

Definition at line 33 of file tables.h.

4.4.2.4 base_mid

```
u8int gdt_entry_struct::base_mid
```

Definition at line 34 of file tables.h.

4.4.2.5 flags

```
u8int gdt_entry_struct::flags
```

Definition at line 36 of file tables.h.

4.4.2.6 limit_low

```
u16int gdt_entry_struct::limit_low
```

Definition at line 32 of file tables.h.

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

- include/core/tables.h

4.5 header Struct Reference

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

Public Attributes

- int size
- int index_id

4.5.1 Detailed Description

Definition at line 11 of file heap.h.

4.5.2 Member Data Documentation

4.5.2.1 index_id

```
int header::index_id
```

Definition at line 13 of file heap.h.

4.5.2.2 size

```
int header::size
```

Definition at line 12 of file heap.h.

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

- include/mem/heap.h

4.6 heap Struct Reference

`#include <heap.h>`

Public Attributes

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

4.6.1 Detailed Description

Definition at line 33 of file heap.h.

4.6.2 Member Data Documentation

4.6.2.1 base

`u32int heap::base`

Definition at line 35 of file heap.h.

4.6.2.2 index

`index_table heap::index`

Definition at line 34 of file heap.h.

4.6.2.3 max_size

`u32int heap::max_size`

Definition at line 36 of file heap.h.

4.6.2.4 min_size

```
u32int heap::min_size
```

Definition at line 37 of file heap.h.

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

- include/mem/heap.h

4.7 idt_entry_struct Struct Reference

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

Public Attributes

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

4.7.1 Detailed Description

Definition at line 6 of file tables.h.

4.7.2 Member Data Documentation

4.7.2.1 base_high

```
u16int idt_entry_struct::base_high
```

Definition at line 12 of file tables.h.

4.7.2.2 base_low

```
u16int idt_entry_struct::base_low
```

Definition at line 8 of file tables.h.

4.7.2.3 flags

```
u8int idt_entry_struct::flags
```

Definition at line 11 of file tables.h.

4.7.2.4 sselect

```
u16int idt_entry_struct::sselect
```

Definition at line 9 of file tables.h.

4.7.2.5 zero

```
u8int idt_entry_struct::zero
```

Definition at line 10 of file tables.h.

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

- include/core/tables.h

4.8 idt_struct Struct Reference

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

Public Attributes

- u16int limit
- u32int base

4.8.1 Detailed Description

Definition at line 16 of file tables.h.

4.8.2 Member Data Documentation

4.8.2.1 base

```
u32int idt_struct::base
```

Definition at line 19 of file tables.h.

4.8.2.2 limit

```
u16int idt_struct::limit
```

Definition at line 18 of file tables.h.

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

- include/core/tables.h

4.9 index_entry Struct Reference

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

Public Attributes

- int size
- int empty
- u32int block

4.9.1 Detailed Description

Definition at line 20 of file heap.h.

4.9.2 Member Data Documentation

4.9.2.1 block

```
u32int index_entry::block
```

Definition at line 23 of file heap.h.

4.9.2.2 empty

```
int index_entry::empty
```

Definition at line 22 of file heap.h.

4.9.2.3 size

```
int index_entry::size
```

Definition at line 21 of file heap.h.

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

- include/mem/heap.h

4.10 index_table Struct Reference

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

Public Attributes

- index_entry table [0x1000]
- int id

4.10.1 Detailed Description

Definition at line 27 of file heap.h.

4.10.2 Member Data Documentation

4.10.2.1 id

```
int index_table::id
```

Definition at line 29 of file heap.h.

4.10.2.2 table

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

Definition at line 28 of file heap.h.

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

- include/mem/heap.h

4.11 page_dir Struct Reference

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

Public Attributes

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

4.11.1 Detailed Description

Definition at line 34 of file paging.h.

4.11.2 Member Data Documentation

4.11.2.1 tables

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

Definition at line 35 of file paging.h.

4.11.2.2 tables_phys

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

Definition at line 36 of file paging.h.

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

- include/mem/paging.h

4.12 page_entry Struct Reference

`#include <paging.h>`

Public Attributes

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

4.12.1 Detailed Description

Definition at line 12 of file paging.h.

4.12.2 Member Data Documentation

4.12.2.1 accessed

`u32int page_entry::accessed`

Definition at line 16 of file paging.h.

4.12.2.2 dirty

`u32int page_entry::dirty`

Definition at line 17 of file paging.h.

4.12.2.3 frameaddr

`u32int page_entry::frameaddr`

Definition at line 19 of file paging.h.

4.12.2.4 present

```
u32int page_entry::present
```

Definition at line 13 of file paging.h.

4.12.2.5 reserved

```
u32int page_entry::reserved
```

Definition at line 18 of file paging.h.

4.12.2.6 usermode

```
u32int page_entry::usermode
```

Definition at line 15 of file paging.h.

4.12.2.7 writeable

```
u32int page_entry::writeable
```

Definition at line 14 of file paging.h.

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

- include/mem/paging.h

4.13 page_table Struct Reference

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

Public Attributes

- page_entry pages [1024]

4.13.1 Detailed Description

Definition at line 26 of file paging.h.

4.13.2 Member Data Documentation

4.13.2.1 pages

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

Definition at line 27 of file paging.h.

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

- include/mem/paging.h

4.14 param Struct Reference

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

Public Attributes

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

4.14.1 Detailed Description

Definition at line 31 of file mpx_supt.h.

4.14.2 Member Data Documentation

4.14.2.1 buffer_ptr

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

Definition at line 34 of file mpx_supt.h.

4.14.2.2 count_ptr

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

Definition at line 35 of file mpx_supt.h.

4.14.2.3 device_id

```
int param::device_id
```

Definition at line 33 of file mpx_supt.h.

4.14.2.4 op_code

```
int param::op_code
```

Definition at line 32 of file mpx_supt.h.

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

- modules/mpx_supt.h

4.15 PCB Struct Reference

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

Public Attributes

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

4.15.1 Detailed Description

Definition at line 1 of file R2_Internal_Functions_And_Structures.h.

4.15.2 Member Data Documentation

4.15.2.1 nextPCB

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

Definition at line 11 of file R2_Internal_Functions_And_Structures.h.

4.15.2.2 prevPCB

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

Definition at line 12 of file R2_Internal_Functions_And_Structures.h.

4.15.2.3 priority

```
int PCB::priority
```

Definition at line 5 of file R2_Internal_Functions_And_Structures.h.

4.15.2.4 processClass

```
unsigned char PCB::processClass
```

Definition at line 4 of file R2_Internal_Functions_And_Structures.h.

4.15.2.5 processName

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

Definition at line 3 of file R2_Internal_Functions_And_Structures.h.

4.15.2.6 runningStatus

```
int PCB::runningStatus
```

Definition at line 6 of file R2_Internal_Functions_And_Structures.h.

4.15.2.7 stack

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

Definition at line 8 of file R2_Internal_Functions_And_Structures.h.

4.15.2.8 stackBase

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

Definition at line 10 of file R2_Internal_Functions_And_Structures.h.

4.15.2.9 stackTop

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

Definition at line 9 of file R2_Internal_Functions_And_Structures.h.

4.15.2.10 suspendedStatus

```
int PCB::suspendedStatus
```

Definition at line 7 of file R2_Internal_Functions_And_Structures.h.

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

- modules/R2/R2_Internal_Functions_And_Structures.h

4.16 queue Struct Reference

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

Public Attributes

- int count
- PCB *head
- PCB *tail

4.16.1 Detailed Description

Definition at line 15 of file R2_Internal_Functions_And_Structures.h.

4.16.2 Member Data Documentation

4.16.2.1 count

```
int queue::count
```

Definition at line 17 of file R2_Internal_Functions_And_Structures.h.

4.16.2.2 head

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

Definition at line 18 of file R2_Internal_Functions_And_Structures.h.

4.16.2.3 tail

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

Definition at line 19 of file R2_Internal_Functions_And_Structures.h.

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

- modules/R2/R2_Internal_Functions_And_Structures.h

Chapter 5

File Documentation

5.1 include/core/asm.h File Reference

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

5.2 include/core/interrupts.h File Reference

Functions

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

5.2.1 Function Documentation

5.2.1.1 init_irq()

```
void init_irq (
              void  )
```

Definition at line 66 of file interrupts.c.

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

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

5.2.1.2 init_pic()

```
void init_pic (
              void  )
```

Definition at line 106 of file interrupts.c.

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

5.3 include/core/io.h File Reference

Macros

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

5.3.1 Macro Definition Documentation

5.3.1.1 inb

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

Value:

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

Definition at line 15 of file io.h.

5.3.1.2 outb

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

Definition at line 8 of file io.h.

5.4 include/core/serial.h File Reference

Macros

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

Functions

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

5.4.1 Macro Definition Documentation

5.4.1.1 COM1

```
#define COM1 0x3f8
```

Definition at line 4 of file serial.h.

5.4.1.2 COM2

```
#define COM2 0x2f8
```

Definition at line 5 of file serial.h.

5.4.1.3 COM3

```
#define COM3 0x3e8
```

Definition at line 6 of file serial.h.

5.4.1.4 COM4

```
#define COM4 0x2e8
```

Definition at line 7 of file serial.h.

5.4.2 Function Documentation

5.4.2.1 init_serial()

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

Definition at line 22 of file serial.c.

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

5.4.2.2 polling()

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

Definition at line 92 of file serial.c.

```
93 {
94   // insert your code to gather keyboard input via the technique of polling.
95
96   char keyboard_character;
97
98   int cursor = 0;
99
100   char log[] = {'\0', '\0', '\0', '\0'};
101
102   int characters_in_buffer = 0;
103
104   while (1)
105   {
106
107     if (inb(COM1 + 5) & 1)
108     {                                  // 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(&keyboard_character);
206           cursor++;
207         }
208         else if (buffer[cursor] == ' \0') //Adding character at the end of the buffer
209         {
210           buffer[cursor] = keyboard_character;
211           serial_print(&keyboard_character);
212           cursor++;
213         }
214         else //Inserting character to the middle of the buffer
215         {
216           char temp_buffer[strlen(buffer)];
217           memset(temp_buffer, ' \0', strlen(buffer));
218 
219           int temp_cursor = 0;
220           while (temp_cursor <= characters_in_buffer) //Filling the temp_buffer with all of the
    characters from buffer, and inserting the new character.
221           {
222             if (temp_cursor < cursor)
223             {
224               temp_buffer[temp_cursor] = buffer[temp_cursor];
225             }
226             else if (temp_cursor > cursor)
227             {
228               temp_buffer[temp_cursor] = buffer[temp_cursor - 1];
229             }
230             else
231             { //temp_cursor == cursor
232               temp_buffer[temp_cursor] = keyboard_character;
233             }
234             temp_cursor++;
235           }
236 
237           temp_cursor = 0;
238           int temp_buffer_size = strlen(temp_buffer);
239           while (temp_cursor <= temp_buffer_size) //Setting the contents of the buffer equal to the
    temp_buffer.
240           {
241             buffer[temp_cursor] = temp_buffer[temp_cursor];
242             temp_cursor++;
243           }
244 
245           serial_print("\033[K");
246           serial_print(&keyboard_character);
247           serial_print(buffer + cursor + 1);
248           cursor++;
249         }
250         characters_in_buffer++;
251       }
252     }
253   }
254 
```

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

5.4.2.3 serial_print()

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

Definition at line 56 of file serial.c.

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

5.4.2.4 serial_println()

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

Definition at line 40 of file serial.c.

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

5.4.2.5 set_serial_in()

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

Definition at line 86 of file serial.c.

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

5.4.2.6 set_serial_out()

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

Definition at line 74 of file serial.c.

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

5.5 include/core/tables.h File Reference

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

Classes

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

Functions

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

Variables

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

5.5.1 Function Documentation

5.5.1.1 __attribute__()

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

5.5.1.2 gdt_init_entry()

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

Definition at line 57 of file tables.c.

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

5.5.1.3 idt_set_gate()

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

Definition at line 27 of file tables.c.

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

5.5.1.4 init_gdt()

```
void init_gdt ( )
```

Definition at line 75 of file tables.c.

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

5.5.1.5 init_idt()

```
void init_idt ( )
```

Definition at line 43 of file tables.c.

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

5.5.2 Variable Documentation

5.5.2.1 access

```
u8int access
```

Definition at line 3 of file tables.h.

5.5.2.2 base

```
u32int base
```

Definition at line 1 of file tables.h.

5.5.2.3 base_high

```
u8int base_high
```

Definition at line 4 of file tables.h.

5.5.2.4 base_low

```
u16int base_low
```

Definition at line 0 of file tables.h.

5.5.2.5 base_mid

```
u8int base_mid
```

Definition at line 2 of file tables.h.

5.5.2.6 flags

```
u8int flags
```

Definition at line 3 of file tables.h.

5.5.2.7 limit

```
u16int limit
```

Definition at line 0 of file tables.h.

5.5.2.8 limit_low

```
u16int limit_low
```

Definition at line 0 of file tables.h.

5.5.2.9 sselect

```
u16int sselect
```

Definition at line 1 of file tables.h.

5.5.2.10 zero

```
u8int zero
```

Definition at line 2 of file tables.h.

5.6 include/mem/heap.h File Reference

Classes

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

Macros

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

Functions

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

5.6.1 Macro Definition Documentation

5.6.1.1 KHEAP_BASE

```
#define KHEAP_BASE 0xD000000
```

Definition at line 6 of file heap.h.

5.6.1.2 KHEAP_MIN

```
#define KHEAP_MIN 0x10000
```

Definition at line 7 of file heap.h.

5.6.1.3 KHEAP_SIZE

```
#define KHEAP_SIZE 0x1000000
```

Definition at line 8 of file heap.h.

5.6.1.4 TABLE_SIZE

```
#define TABLE_SIZE 0x1000
```

Definition at line 5 of file heap.h.

5.6.2 Function Documentation

5.6.2.1 _kmalloc()

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

Definition at line 24 of file heap.c.

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

5.6.2.2 alloc()

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

Definition at line 57 of file heap.c.

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

5.6.2.3 init_kheap()

```
void init_kheap ( )
```

5.6.2.4 kfree()

```
u32int kfree ( )
```

5.6.2.5 kmalloc()

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

Definition at line 52 of file heap.c.

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

5.6.2.6 make_heap()

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

Definition at line 71 of file heap.c.

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

5.7 include/mem/paging.h File Reference

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

Classes

- struct page_entry
- struct page_table
- struct page_dir

Macros

- #define PAGE_SIZE 0x1000

Functions

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

5.7.1 Macro Definition Documentation

5.7.1.1 PAGE_SIZE

```
#define PAGE_SIZE 0x1000
```

Definition at line 6 of file paging.h.

5.7.2 Function Documentation

5.7.2.1 clear_bit()

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

Definition at line 44 of file paging.c.

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

5.7.2.2 first_free()

```
u32int first_free ( )
```

5.7.2.3 get_bit()

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

Definition at line 56 of file paging.c.

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

5.7.2.4 get_page()

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

Definition at line 85 of file paging.c.

```
86 {
87   u32int phys_addr;
88   u32int index = addr / page_size / 1024;
89   u32int offset = addr / page_size % 1024;
90
91   //return it if it exists
92   if (dir->tables[index])
93     return &dir->tables[index]->pages[offset];
94
95   //create it
96   else if (make_table){
97     dir->tables[index] = (page_table*)_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

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

5.9.1 Macro Definition Documentation

5.9.1.1 asm

```
#define asm __asm__
```

Definition at line 11 of file system.h.

5.9.1.2 cli

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

Definition at line 15 of file system.h.

5.9.1.3 GDT_CS_ID

```
#define GDT_CS_ID 0x01
```

Definition at line 20 of file system.h.

5.9.1.4 GDT_DS_ID

```
#define GDT_DS_ID 0x02
```

Definition at line 21 of file system.h.

5.9.1.5 hlt

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

Definition at line 17 of file system.h.

5.9.1.6 iret

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

Definition at line 18 of file system.h.

5.9.1.7 no_warn

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

Definition at line 7 of file system.h.

5.9.1.8 nop

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

Definition at line 16 of file system.h.

5.9.1.9 NULL

```
#define NULL 0
```

Definition at line 4 of file system.h.

5.9.1.10 sti

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

Definition at line 14 of file system.h.

5.9.1.11 volatile

```
#define volatile __volatile__
```

Definition at line 12 of file system.h.

5.9.2 Typedef Documentation

5.9.2.1 size_t

```
typedef unsigned int size_t
```

Definition at line 24 of file system.h.

5.9.2.2 u16int

```
typedef unsigned short u16int
```

Definition at line 26 of file system.h.

5.9.2.3 u32int

```
typedef unsigned long u32int
```

Definition at line 27 of file system.h.

5.9.2.4 u8int

```
typedef unsigned char u8int
```

Definition at line 25 of file system.h.

5.9.3 Function Documentation

5.9.3.1 irq_on()

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

Definition at line 42 of file system.h.

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

5.9.3.2 klogv()

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

Definition at line 11 of file system.c.

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

5.9.3.3 kpanic()

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

Definition at line 24 of file system.c.

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

5.10 kernel/core/interrupts.c File Reference

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

Macros

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

Functions

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

Variables

- idt_entry idt_entries [256]

5.10.1 Macro Definition Documentation

5.10.1.1 ICW1

```
#define ICW1 0x11
```

Definition at line 20 of file interrupts.c.

5.10.1.2 ICW4

```
#define ICW4 0x01
```

Definition at line 21 of file interrupts.c.

5.10.1.3 io_wait

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

Definition at line 28 of file interrupts.c.

5.10.1.4 PIC1

```
#define PIC1 0x20
```

Definition at line 16 of file interrupts.c.

5.10.1.5 PIC2

```
#define PIC2 0xA0
```

Definition at line 17 of file interrupts.c.

5.10.2 Function Documentation

5.10.2.1 bounds()

```
void bounds ( )
```

5.10.2.2 breakpoint()

```
void breakpoint ( )
```

5.10.2.3 coprocessor()

```
void coprocessor ( )
```

5.10.2.4 coprocessor_segment()

```
void coprocessor_segment ( )
```

5.10.2.5 debug()

```
void debug ( )
```

5.10.2.6 device_not_available()

```
void device_not_available ( )
```

5.10.2.7 divide_error()

```
void divide_error ( )
```

5.10.2.8 do_bounds()

```
void do_bounds ( )
```

Definition at line 149 of file interrupts.c.

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

5.10.2.9 do_breakpoint()

```
void do_breakpoint ( )
```

Definition at line 141 of file interrupts.c.

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

5.10.2.10 do_coprocessor()

```
void do_coprocessor ( )
```

Definition at line 193 of file interrupts.c.

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

5.10.2.11 do_coprocessor_segment()

```
void do_coprocessor_segment ( )
```

Definition at line 165 of file interrupts.c.

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

5.10.2.12 do_debug()

```
void do_debug ( )
```

Definition at line 133 of file interrupts.c.

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

5.10.2.13 do_device_not_available()

```
void do_device_not_available ( )
```

Definition at line 157 of file interrupts.c.

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

5.10.2.14 do_divide_error()

```
void do_divide_error ( )
```

Definition at line 129 of file interrupts.c.

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

5.10.2.15 do_double_fault()

```
void do_double_fault ( )
```

Definition at line 161 of file interrupts.c.

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

5.10.2.16 do_general_protection()

```
void do_general_protection ( )
```

Definition at line 181 of file interrupts.c.

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

5.10.2.17 do_invalid_op()

```
void do_invalid_op ( )
```

Definition at line 153 of file interrupts.c.

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

5.10.2.18 do_invalid_tss()

```
void do_invalid_tss ( )
```

Definition at line 169 of file interrupts.c.

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

5.10.2.19 do_isr()

```
void do_isr ( )
```

Definition at line 53 of file interrupts.c.

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

5.10.2.20 do_nmi()

```
void do_nmi ( )
```

Definition at line 137 of file interrupts.c.

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

5.10.2.21 do_overflow()

```
void do_overflow ( )
```

Definition at line 145 of file interrupts.c.

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

5.10.2.22 do_page_fault()

```
void do_page_fault ( )
```

Definition at line 185 of file interrupts.c.

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

5.10.2.23 do_reserved()

```
void do_reserved ( )
```

Definition at line 189 of file interrupts.c.

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

5.10.2.24 do_segment_not_present()

```
void do_segment_not_present ( )
```

Definition at line 173 of file interrupts.c.

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

5.10.2.25 do_stack_segment()

```
void do_stack_segment ( )
```

Definition at line 177 of file interrupts.c.

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

5.10.2.26 double_fault()

```
void double_fault ( )
```

5.10.2.27 general_protection()

```
void general_protection ( )
```

5.10.2.28 init_irq()

```
void init_irq (
              void  )
```

Definition at line 66 of file interrupts.c.

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

5.10.2.29 init_pic()

```
void init_pic (
              void  )
```

Definition at line 106 of file interrupts.c.

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

5.10.2.30 invalid_op()

```
void invalid_op ( )
```

5.10.2.31 invalid_tss()

```
void invalid_tss ( )
```

5.10.2.32 isr0()

```
void isr0 ( )
```

5.10.2.33 nmi()

```
void nmi ( )
```

5.10.2.34 overflow()

```
void overflow ( )
```

5.10.2.35 page_fault()

```
void page_fault ( )
```

5.10.2.36 reserved()

```
void reserved ( )
```

5.10.2.37 rtc_isr()

```
void rtc_isr ( )
```

5.10.2.38 segment_not_present()

```
void segment_not_present ( )
```

5.10.2.39 stack_segment()

```
void stack_segment ( )
```

5.10.3 Variable Documentation

5.10.3.1 idt_entries

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

Definition at line 17 of file tables.c.

5.11 kernel/core/kmain.c File Reference

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

Functions

- void kmain (void)

5.11.1 Function Documentation

5.11.1.1 kmain()

```
void kmain (
             void  )
```

Definition at line 28 of file kmain.c.

```
29 {
30     // extern uint32_t magic;
31     // Uncomment if you want to access the multiboot header
32     // extern void *mbd;
33     // char *boot_loader_name = (char*)((long*)mbd)[16];
34 
35 
36     // 0) Initialize Serial I/O
37     // functions to initialize serial I/O can be found in serial.c
38     // there are 3 functions to call
39 
40     init_serial(COM1);
41     set_serial_in(COM1);
42     set_serial_out(COM1);
43 
44     klogv("Starting MPX boot sequence...");
45     klogv("Initialized serial I/O on COM1 device...");
46 
47     // 1) Initialize the support software by identifying the current
48     //     MPX Module.  This will change with each module.
49     // you will need to call mpx_init from the mpx_supt.c
50 
51     mpx_init(MODULE_R1);
52 
53     // 2) Check that the boot was successful and correct when using grub
54     // Comment this when booting the kernel directly using QEMU, etc.
55     //if ( magic != 0x2BADB002 ){
56     //  kpanic("Boot was not error free. Halting.");
57     //}
58 
59     // 3) Descriptor Tables -- tables.c
60     //  you will need to initialize the global
61     // this keeps track of allocated segments and pages
62     klogv("Initializing descriptor tables...");
63 
64     init_gdt();
65     init_idt();
66 
67     init_pic();
68     sti();
69 
70      // 4)  Interrupt vector table --  tables.c
71      // this creates and initializes a default interrupt vector table
72      // this function is in tables.c
73 
74     init_irq();
75 
76      klogv("Interrupt vector table initialized!");
77 
78     // 5) Virtual Memory -- paging.c  -- init_paging
79     //  this function creates the kernel's heap
80     //  from which memory will be allocated when the program calls
81     // sys_alloc_mem UNTIL the memory management module  is completed
82     // this allocates memory using discrete "pages" of physical memory
83     // NOTE:  You will only have about 70000 bytes of dynamic memory
84     //
85     klogv("Initializing virtual memory...");
86 
87     init_paging();
88 
89     // 6) Call YOUR command handler -  interface method
90     klogv("Transferring control to commhand...");
91     commhand();
92 
93     // 7) System Shutdown on return from your command handler
94 
95     klogv("Starting system shutdown procedure...");
96 
97     /* Shutdown Procedure */
98     klogv("Shutdown complete. You may now turn off the machine. (QEMU: C-a x)");
99     hlt();
100 }
```

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(&keyboard_character);
206           cursor++;
207         }
208         else if (buffer[cursor] == ' \0') //Adding character at the end of the buffer
209         {
210           buffer[cursor] = keyboard_character;
211           serial_print(&keyboard_character);
212           cursor++;
213         }
214         else //Inserting character to the middle of the buffer
215         {
216           char temp_buffer[strlen(buffer)];
217           memset(temp_buffer, ' \0', strlen(buffer));
218 
219           int temp_cursor = 0;
220           while (temp_cursor <= characters_in_buffer) //Filling the temp_buffer with all of the
    characters from buffer, and inserting the new character.
221           {
222             if (temp_cursor < cursor)
223             {
224               temp_buffer[temp_cursor] = buffer[temp_cursor];
225             }
226             else if (temp_cursor > cursor)
227             {
228               temp_buffer[temp_cursor] = buffer[temp_cursor - 1];
229             }
230             else
231             { //temp_cursor == cursor
232               temp_buffer[temp_cursor] = keyboard_character;
233             }
234             temp_cursor++;
235           }
236 
237           temp_cursor = 0;
238           int temp_buffer_size = strlen(temp_buffer);
239           while (temp_cursor <= temp_buffer_size) //Setting the contents of the buffer equal to the
    temp_buffer.
240           {
241             buffer[temp_cursor] = temp_buffer[temp_cursor];
242             temp_cursor++;
243           }
244 
245           serial_print("\033[K");
246           serial_print(&keyboard_character);
247           serial_print(buffer + cursor + 1);
248           cursor++;
249         }
250         characters_in_buffer++;
251       }
252     }
253   }
254 
```

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

5.12.2.3 serial_print()

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

Definition at line 56 of file serial.c.

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

5.12.2.4 serial_println()

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

Definition at line 40 of file serial.c.

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

5.12.2.5 set_serial_in()

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

Definition at line 86 of file serial.c.

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

5.12.2.6 set_serial_out()

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

Definition at line 74 of file serial.c.

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

5.12.3 Variable Documentation

5.12.3.1 serial_port_in

```
int serial_port_in = 0
```

Definition at line 16 of file serial.c.

5.12.3.2 serial_port_out

```
int serial_port_out = 0
```

Definition at line 15 of file serial.c.

5.13 kernel/core/system.c File Reference

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

Functions

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

5.13.1 Function Documentation

5.13.1.1 klogv()

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

Definition at line 11 of file system.c.

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

5.13.1.2 kpanic()

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

Definition at line 24 of file system.c.

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

5.14 kernel/core/tables.c File Reference

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

Functions

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

Variables

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

5.14.1 Function Documentation

5.14.1.1 gdt_init_entry()

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

Definition at line 57 of file tables.c.

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

5.14.1.2 idt_set_gate()

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

Definition at line 27 of file tables.c.

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

5.14.1.3 init_gdt()

```
void init_gdt ( )
```

Definition at line 75 of file tables.c.

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

5.14.1.4 init_idt()

```
void init_idt ( )
```

Definition at line 43 of file tables.c.

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

5.14.1.5 write_gdt_ptr()

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

5.14.1.6 write_idt_ptr()

```
void write_idt_ptr (
             u32int  )
```

5.14.2 Variable Documentation

5.14.2.1 gdt_entries

```
gdt_entry gdt_entries[5]
```

Definition at line 13 of file tables.c.

5.14.2.2 gdt_ptr

```
gdt_descriptor gdt_ptr
```

Definition at line 12 of file tables.c.

5.14.2.3 idt_entries

```
idt_entry idt_entries[256]
```

Definition at line 17 of file tables.c.

5.14.2.4 idt_ptr

```
idt_descriptor idt_ptr
```

Definition at line 16 of file tables.c.

5.15 kernel/mem/heap.c File Reference

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

Functions

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

Variables

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

5.15.1 Function Documentation

5.15.1.1 _kmalloc()

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

Definition at line 24 of file heap.c.

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

5.15.1.2 alloc()

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

Definition at line 57 of file heap.c.

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

5.15.1.3 kmalloc()

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

Definition at line 52 of file heap.c.

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

5.15.1.4 make_heap()

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

Definition at line 71 of file heap.c.

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

5.15.2 Variable Documentation

5.15.2.1 __end

```
void __end
```

Definition at line 18 of file heap.c.

5.15.2.2 _end

```
void _end
```

Definition at line 18 of file heap.c.

5.15.2.3 curr_heap

```
heap* curr_heap = 0
```

Definition at line 15 of file heap.c.

5.15.2.4 end

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

5.15.2.5 kdir

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

Definition at line 21 of file paging.c.

5.15.2.6 kheap

```
heap* kheap = 0
```

Definition at line 14 of file heap.c.

5.15.2.7 phys_alloc_addr

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

Definition at line 22 of file heap.c.

5.16 kernel/mem/paging.c File Reference

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

Functions

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

Variables

- u32int mem_size = 0x4000000
- u32int page_size = 0x1000
- u32int nframes
- u32int *frames
- page_dir *kdir = 0
- page_dir *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>
```

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 ()

Variables

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

5.18.1 Function Documentation

5.18.1.1 idle()

```
void idle ( )
```

Definition at line 173 of file mpx_supt.c.

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

5.18.1.2 mpx_init()

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

Definition at line 106 of file mpx_supt.c.

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

5.18.1.3 sys_alloc_mem()

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

Definition at line 144 of file mpx_supt.c.

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

5.18.1.4 sys_free_mem()

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

Definition at line 158 of file mpx_supt.c.

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

5.18.1.5 sys_req()

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

Definition at line 49 of file mpx_supt.c.

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

5.18.1.6 sys_set_free()

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

Definition at line 134 of file mpx_supt.c.

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

5.18.1.7 sys_set_malloc()

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

Definition at line 124 of file mpx_supt.c.

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

5.18.2 Variable Documentation

5.18.2.1 current_module

```
int current_module = -1
```

Definition at line 18 of file mpx_supt.c.

5.18.2.2 io_module_active

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

Definition at line 19 of file mpx_supt.c.

5.18.2.3 mem_module_active

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

Definition at line 20 of file mpx_supt.c.

5.18.2.4 params

```
param params
```

Definition at line 15 of file mpx_supt.c.

5.18.2.5 student_free

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

Definition at line 28 of file mpx_supt.c.

5.18.2.6 student_malloc

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

Definition at line 24 of file mpx_supt.c.

5.19 modules/mpx_supt.h File Reference

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

Classes

- struct param

Macros

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

Functions

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

5.19.1 Macro Definition Documentation

5.19.1.1 COM_PORT

```
#define COM_PORT 222
```

Definition at line 29 of file mpx_supt.h.

5.19.1.2 DEFAULT_DEVICE

```
#define DEFAULT_DEVICE 111
```

Definition at line 28 of file mpx_supt.h.

5.19.1.3 EXIT

```
#define EXIT 0
```

Definition at line 6 of file mpx_supt.h.

5.19.1.4 FALSE

```
#define FALSE 0
```

Definition at line 13 of file mpx_supt.h.

5.19.1.5 IDLE

```
#define IDLE 1
```

Definition at line 7 of file mpx_supt.h.

5.19.1.6 INVALID_BUFFER

```
#define INVALID_BUFFER 1000
```

Definition at line 25 of file mpx_supt.h.

5.19.1.7 INVALID_COUNT

```
#define INVALID_COUNT 2000
```

Definition at line 26 of file mpx_supt.h.

5.19.1.8 INVALID_OPERATION

```
#define INVALID_OPERATION 4
```

Definition at line 10 of file mpx_supt.h.

5.19.1.9 IO_MODULE

```
#define IO_MODULE 10
```

Definition at line 21 of file mpx_supt.h.

5.19.1.10 MEM_MODULE

```
#define MEM_MODULE 11
```

Definition at line 22 of file mpx_supt.h.

5.19.1.11 MODULE_F

```
#define MODULE_F 9
```

Definition at line 20 of file mpx_supt.h.

5.19.1.12 MODULE_R1

```
#define MODULE_R1 0
```

Definition at line 15 of file mpx_supt.h.

5.19.1.13 MODULE_R2

```
#define MODULE_R2 1
```

Definition at line 16 of file mpx_supt.h.

5.19.1.14 MODULE_R3

```
#define MODULE_R3 2
```

Definition at line 17 of file mpx_supt.h.

5.19.1.15 MODULE_R4

```
#define MODULE_R4 4
```

Definition at line 18 of file mpx_supt.h.

5.19.1.16 MODULE_R5

```
#define MODULE_R5 8
```

Definition at line 19 of file mpx_supt.h.

5.19.1.17 READ

```
#define READ 2
```

Definition at line 8 of file mpx_supt.h.

5.19.1.18 TRUE

```
#define TRUE 1
```

Definition at line 12 of file mpx_supt.h.

5.19.1.19 WRITE

```
#define WRITE 3
```

Definition at line 9 of file mpx_supt.h.

5.19.2 Function Documentation

5.19.2.1 idle()

```
void idle ( )
```

Definition at line 173 of file mpx_supt.c.

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

5.19.2.2 mpx_init()

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

Definition at line 106 of file mpx_supt.c.

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

5.19.2.3 sys_alloc_mem()

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

Definition at line 144 of file mpx_supt.c.

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

5.19.2.4 sys_free_mem()

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

Definition at line 158 of file mpx_supt.c.

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

5.19.2.5 sys_req()

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

Definition at line 49 of file mpx_supt.c.

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

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

5.19.2.6 sys_set_free()

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

Definition at line 134 of file mpx_supt.c.

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

5.19.2.7 sys_set_malloc()

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

Definition at line 124 of file mpx_supt.c.

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

5.20 modules/R1/commhand.c File Reference

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

Functions

- int commhand ()

5.20.1 Function Documentation

5.20.1.1 commhand()

```
int commhand ( )
```

Definition at line 10 of file commhand.c.

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

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

```
153             resumePCB(processName);
154         }
155         else if (strcmp(cmdBuffer, "setPCBPriority") == 0)
156         {
157             char processName[20];
158             int newProcessPriority;
159 
160             char nameMsg[] = "Please enter the name for the PCB you wish to change priorities for. (The
    name can be no more than 20 characters)\n";
161             int nameMsgLen = strlen(nameMsg);
162             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
163             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
164             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
165             strcpy(processName, cmdBuffer);
166 
167             char priorityMsg[] = "Please enter a priority for the PCB you wish to change priorities for.
    (The priorities range from 0 to 9)\n";
168             int priorityMsgLen = strlen(priorityMsg);
169             sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
170             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
171             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
172             newProcessPriority = atoi(cmdBuffer);
173 
174             setPCBPriority(processName, newProcessPriority);
175         }
176         else if (strcmp(cmdBuffer, "showPCB") == 0)
177         {
178             char processName[20];
179 
180             char nameMsg[] = "Please enter the name for the PCB you wish to see. (The name can be no
    more than 20 characters)\n";
181             int nameMsgLen = strlen(nameMsg);
182             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
183             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
184             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
185             strcpy(processName, cmdBuffer);
186 
187             showPCB(processName);
188         }
189         else if (strcmp(cmdBuffer, "showReady") == 0)
190         {
191             showReady();
192         }
193         else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
194         {
195             showSuspendedReady();
196         }
197         else if (strcmp(cmdBuffer, "showSuspendedBlocked") == 0)
198         {
199             showSuspendedBlocked();
200         }
201         else if (strcmp(cmdBuffer, "showBlocked") == 0)
202         {
203             showBlocked();
204         }
205         else if (strcmp(cmdBuffer, "showAll") == 0)
206         {
207             showAll();
208         }
209         else if (strcmp(cmdBuffer, "quit") == 0)
210         {
211             quitFlag = quit();
212 
213             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
214         }
215         else
216         {
217             char message[] = "Unrecognized Command\n";
218 
219             int tempBuffer = strlen(message);
220 
221             sys_req(WRITE, DEFAULT_DEVICE, (char *)message, &tempBuffer);
222         }
223 
224         // process the command: take array buffer chars and make a string. Decide what the cmd wants to
    do
225         // see if quit was entered: if string == quit = 1
226     }
227 
228     return 0;
229 }
```

5.21 modules/R1/commhand.h File Reference

Functions

- int commhand ()

5.21.1 Function Documentation

5.21.1.1 commhand()

```
int commhand ( )
```

Definition at line 10 of file commhand.c.

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

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

```
148             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
149             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
150             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
151             strcpy(processName, cmdBuffer);
152 
153             resumePCB(processName);
154         }
155         else if (strcmp(cmdBuffer, "setPCBPriority") == 0)
156         {
157             char processName[20];
158             int newProcessPriority;
159 
160             char nameMsg[] = "Please enter the name for the PCB you wish to change priorities for. (The
    name can be no more than 20 characters)\n";
161             int nameMsgLen = strlen(nameMsg);
162             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
163             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
164             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
165             strcpy(processName, cmdBuffer);
166 
167             char priorityMsg[] = "Please enter a priority for the PCB you wish to change priorities for.
    (The priorities range from 0 to 9)\n";
168             int priorityMsgLen = strlen(priorityMsg);
169             sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
170             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
171             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
172             newProcessPriority = atoi(cmdBuffer);
173 
174             setPCBPriority(processName, newProcessPriority);
175         }
176         else if (strcmp(cmdBuffer, "showPCB") == 0)
177         {
178             char processName[20];
179 
180             char nameMsg[] = "Please enter the name for the PCB you wish to see. (The name can be no
    more than 20 characters)\n";
181             int nameMsgLen = strlen(nameMsg);
182             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
183             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
184             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
185             strcpy(processName, cmdBuffer);
186 
187             showPCB(processName);
188         }
189         else if (strcmp(cmdBuffer, "showReady") == 0)
190         {
191             showReady();
192         }
193         else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
194         {
195             showSuspendedReady();
196         }
197         else if (strcmp(cmdBuffer, "showSuspendedBlocked") == 0)
198         {
199             showSuspendedBlocked();
200         }
201         else if (strcmp(cmdBuffer, "showBlocked") == 0)
202         {
203             showBlocked();
204         }
205         else if (strcmp(cmdBuffer, "showAll") == 0)
206         {
207             showAll();
208         }
209         else if (strcmp(cmdBuffer, "quit") == 0)
210         {
211             quitFlag = quit();
212 
213             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
214         }
215         else
216         {
217             char message[] = "Unrecognized Command\n";
218 
219             int tempBuffer = strlen(message);
220 
221             sys_req(WRITE, DEFAULT_DEVICE, (char *)message, &tempBuffer);
222         }
223 
224         // process the command: take array buffer chars and make a string. Decide what the cmd wants to
    do
225         // see if quit was entered: if string == quit = 1
226     }
227 
228     return 0;
229 }
```

5.22 modules/R1/R1commands.c File Reference

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

Functions

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

5.22.1 Function Documentation

5.22.1.1 BCDtoChar()

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

Definition at line 489 of file R1commands.c.

```
490 {
491
492     int val1 = (test / 16);
493     int val2 = (test % 16);
494
495     buffer[0] = val1 + '0';
496     buffer[1] = val2 + '0';
497
498     return 0;
499 }
```

5.22.1.2 getDate()

```
void getDate ( )
```

Definition at line 239 of file R1commands.c.

```
240 {
241
242     char buffer[4] = " \0\0\0";
243     int count = 4;
244     char divider = '/';
245     char newLine[1] = "\n";
246     int newLineCount = 1;
247
248     outb(0x70, 0x07); // getting Day of month value
249     BCDtoChar(inb(0x71), buffer);
250     buffer[2] = divider;
251     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
252     memset(buffer, ' \0', count);
253
254     outb(0x70, 0x08); // getting Month value
255     BCDtoChar(inb(0x71), buffer);
256     buffer[2] = divider;
257     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
258     memset(buffer, ' \0', count);
259
260     outb(0x70, 0x32); // getting Year value second byte
261     BCDtoChar(inb(0x71), buffer);
262     buffer[2] = ' \0';
263     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
264     memset(buffer, ' \0', count);
265
266     outb(0x70, 0x09); // getting Year value first byte
267     BCDtoChar(inb(0x71), buffer);
268     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
269     memset(buffer, ' \0', count);
270
271     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
272     memset(newLine, '\0', newLineCount);
273 }
```

5.22.1.3 getTime()

```
void getTime ( )
```

Definition at line 86 of file R1commands.c.

```
87 {
88
89     char buffer[4] = " \0\0\0";
90     int count = 4;
91     char divider = ':';
92     char newLine[1] = "\n";
93     int newLineCount = 1;
94
95     outb(0x70, 0x04); // getting Hour value
96     BCDtoChar(inb(0x71), buffer);
97     buffer[2] = divider;
98     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
99     memset(buffer, ' \0', count);
100
101     outb(0x70, 0x02); // getting Minute value
102     BCDtoChar(inb(0x71), buffer);
103     buffer[2] = divider;
104     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
105     memset(buffer, ' \0', count);
106
107     outb(0x70, 0x00); // getting Second value
108     BCDtoChar(inb(0x71), buffer);
109     buffer[2] = ' \0';
110     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
111     memset(buffer, ' \0', count);
112
113     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
114     memset(newLine, '\0', newLineCount);
115 }
```

5.22.1.4 help()

```
int help ( )
```

Definition at line 11 of file R1commands.c.

```
12 {
13
14     // Help Description section
15     char helpDesc[] = "help: Returns basic command information.\n";
16
17     int tempBuffer = strlen(helpDesc);
18
19     sys_req(WRITE, DEFAULT_DEVICE, (char *)helpDesc, &tempBuffer);
20     memset(helpDesc, '\0', tempBuffer);
21
22     // Version Description section
23     char versionDesc[] = "version: Returns the current version of the software.\n";
24
25     tempBuffer = strlen(versionDesc);
26
27     sys_req(WRITE, DEFAULT_DEVICE, (char *)versionDesc, &tempBuffer);
28     memset(versionDesc, '\0', tempBuffer);
29
30     // getTime Description section
31     char getTimeDesc[] = "getTime: Returns the current set time.\n";
32
33     tempBuffer = strlen(getTimeDesc);
34
35     sys_req(WRITE, DEFAULT_DEVICE, (char *)getTimeDesc, &tempBuffer);
36     memset(getTimeDesc, '\0', tempBuffer);
37
38     // setTime Description section
39     char setTimeDesc[] = "setTime: Allows the user to change the set time.\n";
40
41     tempBuffer = strlen(setTimeDesc);
42
43     sys_req(WRITE, DEFAULT_DEVICE, (char *)setTimeDesc, &tempBuffer);
44     memset(setTimeDesc, '\0', tempBuffer);
45
46     // getDate Description section
47     char getDateDesc[] = "getDate: Returns the current set date.\n";
48
49     tempBuffer = strlen(getDateDesc);
50
51     sys_req(WRITE, DEFAULT_DEVICE, (char *)getDateDesc, &tempBuffer);
52     memset(getDateDesc, '\0', tempBuffer);
53
54     // setDate Description section
55     char setDateDesc[] = "setDate: Allows the user to change the set date.\n";
56
57     tempBuffer = strlen(setDateDesc);
58
59     sys_req(WRITE, DEFAULT_DEVICE, (char *)setDateDesc, &tempBuffer);
60     memset(setDateDesc, '\0', tempBuffer);
61
62     // quit Description section
63     char quitDesc[] = "quit: Allows the user to shut the system down.\n";
64
65     tempBuffer = strlen(quitDesc);
66
67     sys_req(WRITE, DEFAULT_DEVICE, (char *)quitDesc, &tempBuffer);
68     memset(quitDesc, '\0', tempBuffer);
69
70     return 0;
71 }
```

5.22.1.5 intToBCD()

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

Definition at line 483 of file R1commands.c.

```
484 {
485
486     return (((test / 10) << 4) | (test % 10));
487 }
```

5.22.1.6 quit()

```
int quit ( )
```

Definition at line 501 of file R1commands.c.

```
502 {
503     int flag = 0;
504 
505     char quitMsg[] = "Are you sure you want to shutdown? y/n\n";
506     int quitMsgLength = strlen(quitMsg);
507     sys_req(WRITE, DEFAULT_DEVICE, quitMsg, &quitMsgLength);
508 
509     char quitAns[] = "\0\0";
510     int quitAnsLength = 1;
511     sys_req(READ, DEFAULT_DEVICE, quitAns, &quitAnsLength);
512     char answer = quitAns[0];
513 
514     if (answer == 'y' || answer == 'Y')
515     {
516         flag = 1;
517     }
518     else if (answer == 'n' || answer == 'N')
519     {
520         flag = 0;
521     }
522     else
523     {
524         char error[] = "Invalid input!\n";
525         int errorLength = strlen(error);
526         sys_req(WRITE, DEFAULT_DEVICE, error, &errorLength);
527     }
528 
529     return flag;
530 }
```

5.22.1.7 setDate()

```
int setDate ( )
```

Definition at line 275 of file R1commands.c.

```
276 {
277 
278     int count = 4; // used to print year
279 
280     char spacer[1] = "\n"; // used to space out terminal outputs
281     int spaceCount = 1;
282 
284     char instruction1[] = "Please type the desired year. I.E.: yyyy.\n";
285     int length = strlen(instruction1);
286 
287     sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
288     memset(instruction1, '\0', length);
289 
290     char year[5] = "\0\0\0\0\0"; // year buffer
291 
292     int flag = 0; // thrown if input is invalid
293 
294     do
295     {
296         sys_req(READ, DEFAULT_DEVICE, year, &count);
297         if (atoi(year) > 0)
298         {
299 
300             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
301             flag = 0;
302 
303             char yearUpper[3] = "\0\0\0";
304             char yearLower[3] = "\0\0\0";
305 
306             yearUpper[0] = year[0];
307             yearUpper[1] = year[1];
308             yearLower[0] = year[2];
309             yearLower[1] = year[3];
310 
311             cli();
312 
```

```
313             outb(0x70, 0x32); // Setting first byte year value
314             outb(0x71, intToBCD(atoi(yearUpper)));
315 
316             outb(0x70, 0x09); // Setting second byte year value
317             outb(0x71, intToBCD(atoi(yearLower)));
318 
319             sti();
320         }
321         else
322         {
323             char invalid[] = "Invalid year.\n";
324             int lengthInval = strlen(invalid);
325             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
326             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
327             memset(invalid, '\0', lengthInval);
328             flag = 1;
329         }
330     } while (flag == 1);
331 
333     char instruction2[] = "Please type the desired month. I.E.: mm.\n";
334     length = strlen(instruction2);
335 
336     sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
337     memset(instruction2, '\0', length);
338 
339     char month[4] = "\0\0\n\0";
340     count = 4; // used to print month
341 
342     do
343     {
344         sys_req(READ, DEFAULT_DEVICE, month, &count);
345         if (atoi(month) < 13 && atoi(month) > 0)
346         {
347 
348             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
349             flag = 0;
350 
351             cli();
352 
353             outb(0x70, 0x08); // Setting month value
354             outb(0x71, intToBCD(atoi(month)));
355 
356             sti();
357         }
358         else
359         {
360             char invalid[] = "Invalid month.\n";
361             int lengthInval = strlen(invalid);
362             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
363             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
364             memset(invalid, '\0', lengthInval);
365             flag = 1;
366         }
367     } while (flag == 1);
368 
370     char instruction3[] = "Please type the desired day of month. I.E.: dd.\n";
371 
372     length = strlen(instruction3);
373     sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
374     memset(instruction3, '\0', length);
375 
376     char day[4] = "\0\0\n\0";
377     count = 4; // used to print day
378 
379     do
380     {
381         sys_req(READ, DEFAULT_DEVICE, day, &count);
382         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
383         if ((atoi(year) % 4 == 0 && atoi(year) % 100 != 0) || atoi(year) % 400 == 0)
384         { // checking for leap year
385 
386             char leapYear[] = "This is a leap year. February has 29 days.\n";
387             length = strlen(leapYear);
388 
389             sys_req(WRITE, DEFAULT_DEVICE, leapYear, &length);
390             memset(leapYear, '\0', length);
391 
392             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)
393             {
394                 flag = 1;
395                 char invalid[] = "Invalid day.\n";
396                 length = strlen(invalid);
397                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
398                 memset(invalid, '\0', length);
399             }
400             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
```

```
      atoi(day) > 30)
401           {
402               flag = 1;
403               char invalid[] = "Invalid day.\n";
404               length = strlen(invalid);
405               sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
406               memset(invalid, '\0', length);
407           }
408           else if ((atoi(month) == 2) && atoi(day) > 29)
409           {
410               flag = 1;
411               char invalid[] = "Invalid day.\n";
412               length = strlen(invalid);
413               sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
414               memset(invalid, '\0', length);
415           }
416           else
417           {
418
419               flag = 0;
420               cli();
421
422               outb(0x70, 0x07); // Setting day of month value
423               outb(0x71, intToBCD(atoi(day)));
424
425               sti();
426           }
427       }
428       else if (atoi(year) % 4 != 0 || atoi(year) % 400 != 0)
429       { // checking for leap year
430
431           char noLeap[] = "This is not a leap year.\n";
432           length = strlen(noLeap);
433           sys_req(WRITE, DEFAULT_DEVICE, noLeap, &length);
434           memset(noLeap, '\0', length);
435
436           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)
437           {
438               flag = 1;
439               char invalid[] = "Invalid day.\n";
440               length = strlen(invalid);
441               sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
442               memset(invalid, '\0', length);
443           }
444           else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
      atoi(day) > 30)
445           {
446               flag = 1;
447               char invalid[] = "Invalid day.\n";
448               length = strlen(invalid);
449               sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
450               memset(invalid, '\0', length);
451           }
452           else if ((atoi(month) == 2) && atoi(day) > 28)
453           {
454               flag = 1;
455               char invalid[] = "Invalid day.\n";
456               length = strlen(invalid);
457               sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
458               memset(invalid, '\0', length);
459           }
460           else
461           {
462
463               cli();
464
465               outb(0x70, 0x07); // Setting day of month value
466               outb(0x71, intToBCD(atoi(day)));
467
468               sti();
469           }
470       }
471
472    } while (flag == 1);
473
474    char exitMessage[] = "The date has been set.\n";
475    int exitLength = strlen(exitMessage);
476    sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
477    memset(exitMessage, '\0', exitLength);
478    memset(spacer, '\0', spaceCount);
479
480    return 0;
481 }
```

5.22.1.8 setTime()

```
int setTime ( )
```

Definition at line 117 of file R1commands.c.

```
118 {
119
120     int count = 4; // counter for printing
121
122     char spacer[1] = "\n"; // used to space out terminal outputs
123     int spaceCount = 1;
124
126     char instruction1[] = "Please type the desired hours. I.E.: hh.\n";
127
128     int length = strlen(instruction1);
129
130     sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
131     memset(instruction1, '\0', length);
132
133     char hour[4] = "\0\0\n\0";
134
135     int flag = 0;
136
137     do
138     {
139         sys_req(READ, DEFAULT_DEVICE, hour, &count);
140         if (atoi(hour) < 24 && atoi(hour) >= 0)
141         {
142
143             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
144             flag = 0;
145         }
146         else
147         {
148             char invalid[] = "Invalid hours.\n";
149             int lengthInval = strlen(invalid);
150             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
151             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
152             memset(invalid, '\0', lengthInval);
153             flag = 1;
154         }
155     } while (flag == 1);
156
158     char instruction2[] = "Please type the desired minutes. I.E.: mm.\n";
159
160     length = strlen(instruction2);
161
162     sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
163     memset(instruction2, '\0', length);
164
165     char minute[4] = "\0\0\n\0";
166
167     do
168     {
169         sys_req(READ, DEFAULT_DEVICE, minute, &count);
170         if (atoi(minute) < 60 && atoi(minute) >= 0)
171         {
172
173             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
174             flag = 0;
175         }
176         else
177         {
178             char invalid[] = "Invalid minutes.\n";
179             int lengthInval = strlen(invalid);
180             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
181             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
182             memset(invalid, '\0', lengthInval);
183             flag = 1;
184         }
185     } while (flag == 1);
186
188     char instruction3[] = "Please type the desired seconds. I.E.: ss.\n";
189
190     length = strlen(instruction3);
191
192     sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
193     memset(instruction3, '\0', length);
194
195     char second[4] = "\0\0\n\0";
196
197     do
198     {
199         sys_req(READ, DEFAULT_DEVICE, second, &count);
200         if (atoi(second) < 60 && atoi(second) >= 0)
```

```
201         {
202
203             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
204             flag = 0;
205         }
206         else
207         {
208             char invalid[] = "Invalid seconds.\n";
209             int lengthInval = strlen(invalid);
210             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
211             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
212             memset(invalid, '\0', lengthInval);
213             flag = 1;
214         }
215     } while (flag == 1);
216
217     cli();
218
219     outb(0x70, 0x04); // Hour
220     outb(0x71, intToBCD(atoi(hour)));
221
222     outb(0x70, 0x02); // Minute
223     outb(0x71, intToBCD(atoi(minute)));
224
225     outb(0x70, 0x00); // Second
226     outb(0x71, intToBCD(atoi(second)));
227
228     sti();
229
230     char exitMessage[] = "The time has been set.\n";
231     int exitLength = strlen(exitMessage);
232     sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
233     memset(exitMessage, '\0', exitLength);
234     memset(spacer, '\0', spaceCount);
235
236     return 0;
237 }
```

5.22.1.9 version()

```
int version ( )
```

Definition at line 73 of file R1commands.c.

```
74 {
75
76     char version[] = "Version 2.0\n";
77
78     int tempBuffer = strlen(version);
79
80     sys_req(WRITE, DEFAULT_DEVICE, (char *)version, &tempBuffer);
81     memset(version, '\0', tempBuffer);
82
83     return 0;
84 }
```

5.23 modules/R1/R1commands.h File Reference

Functions

- void help ()
- void version ()
- void getTime ()
- void setTime ()
- void getDate ()
- void setDate ()
- unsigned int change_int_to_binary (int test)
- int BCDtoChar (unsigned char test, char *buffer)
- int quit ()

5.23.1 Function Documentation

5.23.1.1 BCDtoChar()

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

Definition at line 489 of file R1commands.c.

```
490 {
491
492     int val1 = (test / 16);
493     int val2 = (test % 16);
494
495     buffer[0] = val1 + '0';
496     buffer[1] = val2 + '0';
497
498     return 0;
499 }
```

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 239 of file R1commands.c.

```
240 {
241
242     char buffer[4] = " \0\0\0\0";
243     int count = 4;
244     char divider = '/';
245     char newLine[1] = "\n";
246     int newLineCount = 1;
247
248     outb(0x70, 0x07); // getting Day of month value
249     BCDtoChar(inb(0x71), buffer);
250     buffer[2] = divider;
251     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
252     memset(buffer, ' \0', count);
253
254     outb(0x70, 0x08); // getting Month value
255     BCDtoChar(inb(0x71), buffer);
256     buffer[2] = divider;
257     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
258     memset(buffer, ' \0', count);
259
260     outb(0x70, 0x32); // getting Year value second byte
261     BCDtoChar(inb(0x71), buffer);
262     buffer[2] = ' \0';
263     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
264     memset(buffer, ' \0', count);
265
266     outb(0x70, 0x09); // getting Year value first byte
267     BCDtoChar(inb(0x71), buffer);
268     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
269     memset(buffer, ' \0', count);
270
271     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
272     memset(newLine, '\0', newLineCount);
273 }
```

5.23.1.4 getTime()

```
void getTime ( )
```

Definition at line 86 of file R1commands.c.

```
87 {
88
89     char buffer[4] = " \0\0\0";
90     int count = 4;
91     char divider = ':';
92     char newLine[1] = "\n";
93     int newLineCount = 1;
94
95     outb(0x70, 0x04); // getting Hour value
96     BCDtoChar(inb(0x71), buffer);
97     buffer[2] = divider;
98     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
99     memset(buffer, ' \0', count);
100
101     outb(0x70, 0x02); // getting Minute value
102     BCDtoChar(inb(0x71), buffer);
103     buffer[2] = divider;
104     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
105     memset(buffer, ' \0', count);
106
107     outb(0x70, 0x00); // getting Second value
108     BCDtoChar(inb(0x71), buffer);
109     buffer[2] = ' \0';
110     sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
111     memset(buffer, ' \0', count);
112
113     sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
114     memset(newLine, '\0', newLineCount);
115 }
```

5.23.1.5 help()

```
void help ( )
```

Definition at line 11 of file R1commands.c.

```
12 {
13
14     // Help Description section
15     char helpDesc[] = "help: Returns basic command information.\n";
16
17     int tempBuffer = strlen(helpDesc);
18
19     sys_req(WRITE, DEFAULT_DEVICE, (char *)helpDesc, &tempBuffer);
20     memset(helpDesc, '\0', tempBuffer);
21
22     // Version Description section
23     char versionDesc[] = "version: Returns the current version of the software.\n";
24
25     tempBuffer = strlen(versionDesc);
26
27     sys_req(WRITE, DEFAULT_DEVICE, (char *)versionDesc, &tempBuffer);
28     memset(versionDesc, '\0', tempBuffer);
29
30     // getTime Description section
31     char getTimeDesc[] = "getTime: Returns the current set time.\n";
32
33     tempBuffer = strlen(getTimeDesc);
34
35     sys_req(WRITE, DEFAULT_DEVICE, (char *)getTimeDesc, &tempBuffer);
36     memset(getTimeDesc, '\0', tempBuffer);
37
38     // setTime Description section
39     char setTimeDesc[] = "setTime: Allows the user to change the set time.\n";
40
41     tempBuffer = strlen(setTimeDesc);
42
43     sys_req(WRITE, DEFAULT_DEVICE, (char *)setTimeDesc, &tempBuffer);
44     memset(setTimeDesc, '\0', tempBuffer);
45
46     // getDate Description section
47     char getDateDesc[] = "getDate: Returns the current set date.\n";
```

```
48
49      tempBuffer = strlen(getDateDesc);
50
51      sys_req(WRITE, DEFAULT_DEVICE, (char *)getDateDesc, &tempBuffer);
52      memset(getDateDesc, '\0', tempBuffer);
53
54      // setDate Description section
55      char setDateDesc[] = "setDate: Allows the user to change the set date.\n";
56
57      tempBuffer = strlen(setDateDesc);
58
59      sys_req(WRITE, DEFAULT_DEVICE, (char *)setDateDesc, &tempBuffer);
60      memset(setDateDesc, '\0', tempBuffer);
61
62      // quit Description section
63      char quitDesc[] = "quit: Allows the user to shut the system down.\n";
64
65      tempBuffer = strlen(quitDesc);
66
67      sys_req(WRITE, DEFAULT_DEVICE, (char *)quitDesc, &tempBuffer);
68      memset(quitDesc, '\0', tempBuffer);
69
70      return 0;
71  }
```

5.23.1.6 quit()

```
int quit ( )
```

Definition at line 501 of file R1commands.c.

```
502 {
503     int flag = 0;
504
505     char quitMsg[] = "Are you sure you want to shutdown? y/n\n";
506     int quitMsgLength = strlen(quitMsg);
507     sys_req(WRITE, DEFAULT_DEVICE, quitMsg, &quitMsgLength);
508
509     char quitAns[] = "\0\0";
510     int quitAnsLength = 1;
511     sys_req(READ, DEFAULT_DEVICE, quitAns, &quitAnsLength);
512     char answer = quitAns[0];
513
514     if (answer == 'y' || answer == 'Y')
515     {
516         flag = 1;
517     }
518     else if (answer == 'n' || answer == 'N')
519     {
520         flag = 0;
521     }
522     else
523     {
524         char error[] = "Invalid input!\n";
525         int errorLength = strlen(error);
526         sys_req(WRITE, DEFAULT_DEVICE, error, &errorLength);
527     }
528
529     return flag;
530 }
```

5.23.1.7 setDate()

```
void setDate ( )
```

Definition at line 275 of file R1commands.c.

```
276 {
277
278     int count = 4; // used to print year
279
280     char spacer[1] = "\n"; // used to space out terminal outputs
```

```
281     int spaceCount = 1;
282
284     char instruction1[] = "Please type the desired year. I.E.: yyyy.\n";
285     int length = strlen(instruction1);
286
287     sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
288     memset(instruction1, '\0', length);
289
290     char year[5] = "\0\0\0\0"; // year buffer
291
292     int flag = 0; // thrown if input is invalid
293
294     do
295     {
296         sys_req(READ, DEFAULT_DEVICE, year, &count);
297         if (atoi(year) > 0)
298         {
299
300             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
301             flag = 0;
302
303             char yearUpper[3] = "\0\0";
304             char yearLower[3] = "\0\0";
305
306             yearUpper[0] = year[0];
307             yearUpper[1] = year[1];
308             yearLower[0] = year[2];
309             yearLower[1] = year[3];
310
311             cli();
312
313             outb(0x70, 0x32); // Setting first byte year value
314             outb(0x71, intToBCD(atoi(yearUpper)));
315
316             outb(0x70, 0x09); // Setting second byte year value
317             outb(0x71, intToBCD(atoi(yearLower)));
318
319             sti();
320         }
321         else
322         {
323             char invalid[] = "Invalid year.\n";
324             int lengthInval = strlen(invalid);
325             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
326             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
327             memset(invalid, '\0', lengthInval);
328             flag = 1;
329         }
330     } while (flag == 1);
331
333     char instruction2[] = "Please type the desired month. I.E.: mm.\n";
334     length = strlen(instruction2);
335
336     sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
337     memset(instruction2, '\0', length);
338
339     char month[4] = "\0\0\n\0";
340     count = 4; // used to print month
341
342     do
343     {
344         sys_req(READ, DEFAULT_DEVICE, month, &count);
345         if (atoi(month) < 13 && atoi(month) > 0)
346         {
347
348             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
349             flag = 0;
350
351             cli();
352
353             outb(0x70, 0x08); // Setting month value
354             outb(0x71, intToBCD(atoi(month)));
355
356             sti();
357         }
358         else
359         {
360             char invalid[] = "Invalid month.\n";
361             int lengthInval = strlen(invalid);
362             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
363             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
364             memset(invalid, '\0', lengthInval);
365             flag = 1;
366         }
367     } while (flag == 1);
368
370     char instruction3[] = "Please type the desired day of month. I.E.: dd.\n";
```

```
371
372     length = strlen(instruction3);
373     sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
374     memset(instruction3, '\0', length);
375
376     char day[4] = "\0\0\n\0";
377     count = 4; // used to print day
378
379     do
380     {
381         sys_req(READ, DEFAULT_DEVICE, day, &count);
382         sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
383         if ((atoi(year) % 4 == 0 && atoi(year) % 100 != 0) || atoi(year) % 400 == 0)
384         { // checking for leap year
385
386             char leapYear[] = "This is a leap year. February has 29 days.\n";
387             length = strlen(leapYear);
388
389             sys_req(WRITE, DEFAULT_DEVICE, leapYear, &length);
390             memset(leapYear, '\0', length);
391
392             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)
393             {
394                 flag = 1;
395                 char invalid[] = "Invalid day.\n";
396                 length = strlen(invalid);
397                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
398                 memset(invalid, '\0', length);
399             }
400             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
    atoi(day) > 30)
401             {
402                 flag = 1;
403                 char invalid[] = "Invalid day.\n";
404                 length = strlen(invalid);
405                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
406                 memset(invalid, '\0', length);
407             }
408             else if ((atoi(month) == 2) && atoi(day) > 29)
409             {
410                 flag = 1;
411                 char invalid[] = "Invalid day.\n";
412                 length = strlen(invalid);
413                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
414                 memset(invalid, '\0', length);
415             }
416             else
417             {
418
419                 flag = 0;
420                 cli();
421
422                 outb(0x70, 0x07); // Setting day of month value
423                 outb(0x71, intToBCD(atoi(day)));
424
425                 sti();
426             }
427         }
428         else if (atoi(year) % 4 != 0 || atoi(year) % 400 != 0)
429         { // checking for leap year
430
431             char noLeap[] = "This is not a leap year.\n";
432             length = strlen(noLeap);
433             sys_req(WRITE, DEFAULT_DEVICE, noLeap, &length);
434             memset(noLeap, '\0', length);
435
436             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)
437             {
438                 flag = 1;
439                 char invalid[] = "Invalid day.\n";
440                 length = strlen(invalid);
441                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
442                 memset(invalid, '\0', length);
443             }
444             else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) &&
    atoi(day) > 30)
445             {
446                 flag = 1;
447                 char invalid[] = "Invalid day.\n";
448                 length = strlen(invalid);
449                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
450                 memset(invalid, '\0', length);
451             }
452             else if ((atoi(month) == 2) && atoi(day) > 28)
453             {
```

```
454                 flag = 1;
455                 char invalid[] = "Invalid day.\n";
456                 length = strlen(invalid);
457                 sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
458                 memset(invalid, '\0', length);
459             }
460             else
461             {
462 
463                 cli();
464 
465                 outb(0x70, 0x07); // Setting day of month value
466                 outb(0x71, intToBCD(atoi(day)));
467 
468                 sti();
469             }
470         }
471 
472     } while (flag == 1);
473 
474     char exitMessage[] = "The date has been set.\n";
475     int exitLength = strlen(exitMessage);
476     sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
477     memset(exitMessage, '\0', exitLength);
478     memset(spacer, '\0', spaceCount);
479 
480     return 0;
481 }
```

5.23.1.8 setTime()

```
void setTime ( )
```

Definition at line 117 of file R1commands.c.

```
118 {
119 
120     int count = 4; // counter for printing
121 
122     char spacer[1] = "\n"; // used to space out terminal outputs
123     int spaceCount = 1;
124 
126     char instruction1[] = "Please type the desired hours. I.E.: hh.\n";
127 
128     int length = strlen(instruction1);
129 
130     sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
131     memset(instruction1, '\0', length);
132 
133     char hour[4] = "\0\0\n\0";
134 
135     int flag = 0;
136 
137     do
138     {
139         sys_req(READ, DEFAULT_DEVICE, hour, &count);
140         if (atoi(hour) < 24 && atoi(hour) >= 0)
141         {
142 
143             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
144             flag = 0;
145         }
146         else
147         {
148             char invalid[] = "Invalid hours.\n";
149             int lengthInval = strlen(invalid);
150             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
151             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
152             memset(invalid, '\0', lengthInval);
153             flag = 1;
154         }
155     } while (flag == 1);
156 
158     char instruction2[] = "Please type the desired minutes. I.E.: mm.\n";
159 
160     length = strlen(instruction2);
161 
162     sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
163     memset(instruction2, '\0', length);
164 
```

```
165     char minute[4] = "\0\0\n\0";
166
167     do
168     {
169         sys_req(READ, DEFAULT_DEVICE, minute, &count);
170         if (atoi(minute) < 60 && atoi(minute) >= 0)
171         {
172
173             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
174             flag = 0;
175         }
176         else
177         {
178             char invalid[] = "Invalid minutes.\n";
179             int lengthInval = strlen(invalid);
180             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
181             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
182             memset(invalid, '\0', lengthInval);
183             flag = 1;
184         }
185     } while (flag == 1);
186
188     char instruction3[] = "Please type the desired seconds. I.E.: ss.\n";
189
190     length = strlen(instruction3);
191
192     sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
193     memset(instruction3, '\0', length);
194
195     char second[4] = "\0\0\n\0";
196
197     do
198     {
199         sys_req(READ, DEFAULT_DEVICE, second, &count);
200         if (atoi(second) < 60 && atoi(second) >= 0)
201         {
202
203             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
204             flag = 0;
205         }
206         else
207         {
208             char invalid[] = "Invalid seconds.\n";
209             int lengthInval = strlen(invalid);
210             sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
211             sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
212             memset(invalid, '\0', lengthInval);
213             flag = 1;
214         }
215     } while (flag == 1);
216
217     cli();
218
219     outb(0x70, 0x04); // Hour
220     outb(0x71, intToBCD(atoi(hour)));
221
222     outb(0x70, 0x02); // Minute
223     outb(0x71, intToBCD(atoi(minute)));
224
225     outb(0x70, 0x00); // Second
226     outb(0x71, intToBCD(atoi(second)));
227
228     sti();
229
230     char exitMessage[] = "The time has been set.\n";
231     int exitLength = strlen(exitMessage);
232     sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
233     memset(exitMessage, '\0', exitLength);
234     memset(spacer, '\0', spaceCount);
235
236     return 0;
237 }
```

5.23.1.9 version()

```
void version ( )
```

Definition at line 73 of file R1commands.c.

```
74 {
```

```
75
76      char version[] = "Version 2.0\n";
77
78      int tempBuffer = strlen(version);
79
80      sys_req(WRITE, DEFAULT_DEVICE, (char *)version, &tempBuffer);
81      memset(version, '\0', tempBuffer);
82
83      return 0;
84  }
```

5.24 modules/R2/R2_Internal_Functions_And_Structures.c File Reference

```
#include <string.h>
#include "../mpx_supt.h"
#include "R2_Internal_Functions_And_Structures.h"
```

Functions

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

Variables

- queue *ready
- queue *blocked
- queue *suspendedReady
- queue *suspendedBlocked

5.24.1 Function Documentation

5.24.1.1 allocatePCB()

```
PCB* allocatePCB ( )
```

Definition at line 14 of file R2_Internal_Functions_And_Structures.c.

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

5.24.1.2 allocateQueues()

```
void allocateQueues ( )
```

Definition at line 296 of file R2_Internal_Functions_And_Structures.c.

```
297 {
298     ready = sys_alloc_mem(sizeof(queue));
299     blocked = sys_alloc_mem(sizeof(queue));
300     suspendedReady = sys_alloc_mem(sizeof(queue));
301     suspendedBlocked = sys_alloc_mem(sizeof(queue));
302 }
```

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         char error_message[30] = "Invalid process name.\n";
92         int error_size = strlen(error_message);
93         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
94         return NULL;
95         //return cz we have to stop if the process name is too long
96     }
97     else
98     {
99
100         // searching in ready queue
101
```

```
102         PCB *found_ready_pcb; // this is a pointer to another pointer (** starts). Need testing!
103         found_ready_pcb = searchPCB(ready, processName);
104         if (found_ready_pcb != NULL)
105         {
106             return found_ready_pcb;
107         }
108 
109         // searching PCB in blocked queue
110         PCB *found_blocked_pcb;
111         found_blocked_pcb = searchPCB(blocked, processName);
112         if (found_blocked_pcb != NULL)
113         {
114             return found_blocked_pcb;
115         }
116 
117         // searching PCB in suspendedReady queue
118         PCB *found_suspended_ready_pcb;
119         found_suspended_ready_pcb = searchPCB(suspendedReady, processName);
120         if (found_suspended_ready_pcb != NULL)
121         {
122             return found_suspended_ready_pcb;
123         }
124 
125         // searching PCB in suspendedBlocked queue
126         PCB *found_suspended_blocked_pcb;
127         found_suspended_blocked_pcb = searchPCB(suspendedBlocked, processName);
128         if (found_suspended_blocked_pcb != NULL)
129         {
130             return found_suspended_blocked_pcb;
131         }
132 
133         char errMsg[] = "The process was not found.\n";
134         int errMsgLen = strlen(errMsg);
135         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
136         return NULL;
137     }
138 }
```

5.24.1.4 freePCB()

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

Definition at line 39 of file R2_Internal_Functions_And_Structures.c.

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

5.24.1.5 getBlocked()

```
queue* getBlocked ( )
```

Definition at line 309 of file R2_Internal_Functions_And_Structures.c.

```
310 {
311     return blocked;
312 }
```

5.24.1.6 getReady()

```
queue* getReady ( )
```

Definition at line 304 of file R2_Internal_Functions_And_Structures.c.

```
305 {
306     return ready;
307 }
```

5.24.1.7 getSuspendedBlocked()

```
queue* getSuspendedBlocked ( )
```

Definition at line 319 of file R2_Internal_Functions_And_Structures.c.

```
320 {
321     return suspendedBlocked;
322 }
```

5.24.1.8 getSuspendedReady()

```
queue* getSuspendedReady ( )
```

Definition at line 314 of file R2_Internal_Functions_And_Structures.c.

```
315 {
316     return suspendedReady;
317 }
```

5.24.1.9 insertPCB()

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

Definition at line 173 of file R2_Internal_Functions_And_Structures.c.

```
174 {
175     //BENJAMIN WILL PROGRAM THIS FUNCTION
176
177     //insertPCB() will insert a PCB into the appropriate queue.
178     //Note: The ready queue is a priority queue and the blocked queue is a FIFO queue.
179
180     if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 1)
181     { // Insert into ready queue
182
183         queue *ready = getReady();
184         PCB *tempPtr = ready->head;
185
186         if (tempPtr != NULL)
187         {
188             int temp = 0;
189             while (temp < ready->count)
190             {
191                 if (PCB_to_insert->priority < tempPtr->priority)
192                 {
193                     tempPtr = tempPtr->nextPCB;
194                 }
195                 else if (PCB_to_insert->priority >= tempPtr->priority)
196                 {
197                     PCB_to_insert->nextPCB = tempPtr;
198                     PCB_to_insert->prevPCB = tempPtr->prevPCB;
```

```
199                     tempPtr->prevPCB = PCB_to_insert;
200                 }
201                 else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
202                 {
203                     tempPtr->nextPCB = PCB_to_insert;
204                     PCB_to_insert->prevPCB = tempPtr;
205                 }
206                 temp++;
207             }
208             ready->count++;
209         }
210         else
211         {
212             tempPtr = PCB_to_insert;
213             ready->count++;
214         }
215     }
216     else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
217     { // Insert into suspended ready queue
218         queue *suspendedReady = getSuspendedReady();
219         PCB *tempPtr = suspendedReady->head;
220 
221         if (tempPtr != NULL)
222         {
223             int temp = 0;
224             while (temp < suspendedReady->count)
225             {
226                 if (PCB_to_insert->priority < tempPtr->priority)
227                 {
228                     tempPtr = tempPtr->nextPCB;
229                 }
230                 else if (PCB_to_insert->priority >= tempPtr->priority)
231                 {
232                     PCB_to_insert->nextPCB = tempPtr;
233                     PCB_to_insert->prevPCB = tempPtr->prevPCB;
234                     tempPtr->prevPCB = PCB_to_insert;
235                 }
236                 else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
237                 {
238                     tempPtr->nextPCB = PCB_to_insert;
239                     PCB_to_insert->prevPCB = tempPtr;
240                 }
241                 temp++;
242             }
243             suspendedReady->count++;
244         }
245         else
246         {
247             tempPtr = PCB_to_insert;
248             suspendedReady->count++;
249         }
250     }
251     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 1)
252     { // Insert into blocked queue
253         queue *blocked = getBlocked();
254         PCB *tempPtr = blocked->tail;
255 
256         tempPtr->nextPCB = PCB_to_insert;
257         PCB_to_insert->prevPCB = tempPtr;
258     }
259     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
260     { // Insert into suspended blocked queue
261         queue *suspendedBlocked = getSuspendedBlocked();
262         PCB *tempPtr = suspendedBlocked->tail;
263 
264         tempPtr->nextPCB = PCB_to_insert;
265         PCB_to_insert->prevPCB = tempPtr;
266     }
267 }
```

5.24.1.10 removePCB()

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

Definition at line 269 of file R2_Internal_Functions_And_Structures.c.

```
270 {
271     //BENJAMIN WILL PROGRAM THIS FUNCTION
```

```
272
273     //removePCB() will remove a PCB from the queue in which it is currently stored.
274
275     PCB *tempPrev = PCB_to_remove->prevPCB;
276     PCB *tempNext = PCB_to_remove->nextPCB;
277
278     tempPrev->nextPCB = tempNext;
279     tempNext->prevPCB = tempPrev;
280
281     PCB_to_remove->nextPCB = NULL;
282     PCB_to_remove->prevPCB = NULL;
283
284     int result = sys_free_mem(PCB_to_remove);
285
286     if (result == -1)
287     {
288         return 1;
289     }
290     else
291     {
292         return 0;
293     }
294 }
```

5.24.1.11 searchPCB()

```
PCB* searchPCB (
            queue * PCB_container,
            char * processName )
```

Definition at line 140 of file R2_Internal_Functions_And_Structures.c.

```
141 {
142     // PCB_container has PCB*head and PCB*tail pointers
143     //queue*tempQueue;
144
145     PCB *tempPtr = PCB_container->head;
146
147     int count = PCB_container->count; // tempQueue->count;
148
149     int found = 0; // not found signal
150     // detecting buffer overflow
151
152     int value = 0;
153     while (value <= count)
154     {
155         if (strcmp(tempPtr->processName, processName) == 0)
156         {
157             found = 1; // found signal
158             return tempPtr;
159             break;
160         }
161
162         tempPtr = tempPtr->nextPCB; // don't know why this line is giving assignment from incompatible
      pointer type error.
163         value++;
164     }
165
166     if (found == 0)
167     {
168         return NULL; // Why are this return not recognized??
169     }
170     return tempPtr; // for testing.
171 }
```

5.24.1.12 setupPCB()

```
PCB* setupPCB (
            char * processName,
```

```
            unsigned char processClass,
            int processPriority )
```

Definition at line 48 of file R2_Internal_Functions_And_Structures.c.

```
49 {
50     //COLTON WILL PROGRAM THIS FUNCTION
51
52     //setupPcb() will call allocatePCB() to create an empty PCB, initializes the PCB information, sets
       the PCB state to ready, not suspended.
53
54     PCB *tempPCB = allocatePCB();
55
56     PCB *returnedPCB;
57
58     if (findPCB(processName)->processName == processName)
59     {
60
61         char message[] = "There is already a PCB with this name.\n";
62         int messLength = strlen(message);
63         sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
64
65         returnedPCB = NULL;
66     }
67     else
68     {
69
70         strcpy(tempPCB->processName, processName);
71         tempPCB->processClass = processClass;
72         tempPCB->priority = processPriority;
73         tempPCB->runningStatus = 0;
74         tempPCB->suspendedStatus = 1;
75
76         returnedPCB = tempPCB;
77     }
78
79     return returnedPCB;
80 }
```

5.24.2 Variable Documentation

5.24.2.1 blocked

```
queue* blocked
```

Definition at line 8 of file R2_Internal_Functions_And_Structures.c.

5.24.2.2 ready

```
queue* ready
```

Definition at line 7 of file R2_Internal_Functions_And_Structures.c.

5.24.2.3 suspendedBlocked

```
queue* suspendedBlocked
```

Definition at line 10 of file R2_Internal_Functions_And_Structures.c.

5.24.2.4 suspendedReady

```
queue* suspendedReady
```

Definition at line 9 of file R2_Internal_Functions_And_Structures.c.

5.25 modules/R2/R2_Internal_Functions_And_Structures.h File Reference

Classes

- struct PCB
- struct queue

Typedefs

- typedef struct PCB PCB
- typedef struct queue queue

Functions

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

5.25.1 Typedef Documentation

5.25.1.1 PCB

```
typedef struct PCB PCB
```

5.25.1.2 queue

```
typedef struct queue queue
```

5.25.2 Function Documentation

5.25.2.1 allocatePCB()

```
PCB* allocatePCB ( )
```

Definition at line 14 of file R2_Internal_Functions_And_Structures.c.

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

5.25.2.2 allocateQueues()

```
void allocateQueues ( )
```

Definition at line 296 of file R2_Internal_Functions_And_Structures.c.

```
297 {
298     ready = sys_alloc_mem(sizeof(queue));
299     blocked = sys_alloc_mem(sizeof(queue));
300     suspendedReady = sys_alloc_mem(sizeof(queue));
301     suspendedBlocked = sys_alloc_mem(sizeof(queue));
302 }
```

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         char error_message[30] = "Invalid process name.\n";
92         int error_size = strlen(error_message);
```

```
93          sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
94          return NULL;
95          //return cz we have to stop if the process name is too long
96      }
97      else
98      {
99
100         // searching in ready queue
101
102         PCB *found_ready_pcb; // this is a pointer to another pointer (** starts). Need testing!
103         found_ready_pcb = searchPCB(ready, processName);
104         if (found_ready_pcb != NULL)
105         {
106             return found_ready_pcb;
107         }
108
109         // searching PCB in blocked queue
110         PCB *found_blocked_pcb;
111         found_blocked_pcb = searchPCB(blocked, processName);
112         if (found_blocked_pcb != NULL)
113         {
114             return found_blocked_pcb;
115         }
116
117         // searching PCB in suspendedReady queue
118         PCB *found_suspended_ready_pcb;
119         found_suspended_ready_pcb = searchPCB(suspendedReady, processName);
120         if (found_suspended_ready_pcb != NULL)
121         {
122             return found_suspended_ready_pcb;
123         }
124
125         // searching PCB in suspendedBlocked queue
126         PCB *found_suspended_blocked_pcb;
127         found_suspended_blocked_pcb = searchPCB(suspendedBlocked, processName);
128         if (found_suspended_blocked_pcb != NULL)
129         {
130             return found_suspended_blocked_pcb;
131         }
132
133         char errMsg[] = "The process was not found.\n";
134         int errMsgLen = strlen(errMsg);
135         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
136         return NULL;
137     }
138 }
```

5.25.2.4 freePCB()

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

Definition at line 39 of file R2_Internal_Functions_And_Structures.c.

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

5.25.2.5 getBlocked()

```
queue* getBlocked ( )
```

Definition at line 309 of file R2_Internal_Functions_And_Structures.c.

```
310 {
311     return blocked;
312 }
```

5.25.2.6 getReady()

```
queue* getReady ( )
```

Definition at line 304 of file R2_Internal_Functions_And_Structures.c.

```
305 {
306     return ready;
307 }
```

5.25.2.7 getSuspendedBlocked()

```
queue* getSuspendedBlocked ( )
```

Definition at line 319 of file R2_Internal_Functions_And_Structures.c.

```
320 {
321     return suspendedBlocked;
322 }
```

5.25.2.8 getSuspendedReady()

```
queue* getSuspendedReady ( )
```

Definition at line 314 of file R2_Internal_Functions_And_Structures.c.

```
315 {
316     return suspendedReady;
317 }
```

5.25.2.9 insertPCB()

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

Definition at line 173 of file R2_Internal_Functions_And_Structures.c.

```
174 {
175     //BENJAMIN WILL PROGRAM THIS FUNCTION
176 
177     //insertPCB() will insert a PCB into the appropriate queue.
178     //Note: The ready queue is a priority queue and the blocked queue is a FIFO queue.
179 
180     if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 1)
181     { // Insert into ready queue
182 
183         queue *ready = getReady();
184         PCB *tempPtr = ready->head;
185 
186         if (tempPtr != NULL)
187         {
188             int temp = 0;
189             while (temp < ready->count)
190             {
191                 if (PCB_to_insert->priority < tempPtr->priority)
192                 {
193                     tempPtr = tempPtr->nextPCB;
194                 }
195                 else if (PCB_to_insert->priority >= tempPtr->priority)
196                 {
197                     PCB_to_insert->nextPCB = tempPtr;
198                     PCB_to_insert->prevPCB = tempPtr->prevPCB;
```

```
199                     tempPtr->prevPCB = PCB_to_insert;
200                 }
201                 else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
202                 {
203                     tempPtr->nextPCB = PCB_to_insert;
204                     PCB_to_insert->prevPCB = tempPtr;
205                 }
206                 temp++;
207             }
208             ready->count++;
209         }
210         else
211         {
212             tempPtr = PCB_to_insert;
213             ready->count++;
214         }
215     }
216     else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
217     { // Insert into suspended ready queue
218         queue *suspendedReady = getSuspendedReady();
219         PCB *tempPtr = suspendedReady->head;
220 
221         if (tempPtr != NULL)
222         {
223             int temp = 0;
224             while (temp < suspendedReady->count)
225             {
226                 if (PCB_to_insert->priority < tempPtr->priority)
227                 {
228                     tempPtr = tempPtr->nextPCB;
229                 }
230                 else if (PCB_to_insert->priority >= tempPtr->priority)
231                 {
232                     PCB_to_insert->nextPCB = tempPtr;
233                     PCB_to_insert->prevPCB = tempPtr->prevPCB;
234                     tempPtr->prevPCB = PCB_to_insert;
235                 }
236                 else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
237                 {
238                     tempPtr->nextPCB = PCB_to_insert;
239                     PCB_to_insert->prevPCB = tempPtr;
240                 }
241                 temp++;
242             }
243             suspendedReady->count++;
244         }
245         else
246         {
247             tempPtr = PCB_to_insert;
248             suspendedReady->count++;
249         }
250     }
251     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 1)
252     { // Insert into blocked queue
253         queue *blocked = getBlocked();
254         PCB *tempPtr = blocked->tail;
255 
256         tempPtr->nextPCB = PCB_to_insert;
257         PCB_to_insert->prevPCB = tempPtr;
258     }
259     else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
260     { // Insert into suspended blocked queue
261         queue *suspendedBlocked = getSuspendedBlocked();
262         PCB *tempPtr = suspendedBlocked->tail;
263 
264         tempPtr->nextPCB = PCB_to_insert;
265         PCB_to_insert->prevPCB = tempPtr;
266     }
267 }
```

5.25.2.10 removePCB()

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

Definition at line 269 of file R2_Internal_Functions_And_Structures.c.

```
270 {
271     //BENJAMIN WILL PROGRAM THIS FUNCTION
```

```
272
273     //removePCB() will remove a PCB from the queue in which it is currently stored.
274
275     PCB *tempPrev = PCB_to_remove->prevPCB;
276     PCB *tempNext = PCB_to_remove->nextPCB;
277
278     tempPrev->nextPCB = tempNext;
279     tempNext->prevPCB = tempPrev;
280
281     PCB_to_remove->nextPCB = NULL;
282     PCB_to_remove->prevPCB = NULL;
283
284     int result = sys_free_mem(PCB_to_remove);
285
286     if (result == -1)
287     {
288         return 1;
289     }
290     else
291     {
292         return 0;
293     }
294 }
```

5.25.2.11 searchPCB()

```
PCB* searchPCB (
            queue * PCB_container,
            char * processName )
```

Definition at line 140 of file R2_Internal_Functions_And_Structures.c.

```
141 {
142     // PCB_container has PCB*head and PCB*tail pointers
143     //queue*tempQueue;
144
145     PCB *tempPtr = PCB_container->head;
146
147     int count = PCB_container->count; // tempQueue->count;
148
149     int found = 0; // not found signal
150     // detecting buffer overflow
151
152     int value = 0;
153     while (value <= count)
154     {
155         if (strcmp(tempPtr->processName, processName) == 0)
156         {
157             found = 1; // found signal
158             return tempPtr;
159             break;
160         }
161
162         tempPtr = tempPtr->nextPCB; // don't know why this line is giving assignment from incompatible
      pointer type error.
163         value++;
164     }
165
166     if (found == 0)
167     {
168         return NULL; // Why are this return not recognized??
169     }
170     return tempPtr; // for testing.
171 }
```

5.25.2.12 setupPCB()

```
PCB* setupPCB (
            char * processName,
```

```
            unsigned char processClass,
            int processPriority )
```

Definition at line 48 of file R2_Internal_Functions_And_Structures.c.

```
49 {
50     //COLTON WILL PROGRAM THIS FUNCTION
51 
52     //setupPcb() will call allocatePCB() to create an empty PCB, initializes the PCB information, sets
       the PCB state to ready, not suspended.
53 
54     PCB *tempPCB = allocatePCB();
55 
56     PCB *returnedPCB;
57 
58     if (findPCB(processName)->processName == processName)
59     {
60 
61         char message[] = "There is already a PCB with this name.\n";
62         int messLength = strlen(message);
63         sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
64 
65         returnedPCB = NULL;
66     }
67     else
68     {
69 
70         strcpy(tempPCB->processName, processName);
71         tempPCB->processClass = processClass;
72         tempPCB->priority = processPriority;
73         tempPCB->runningStatus = 0;
74         tempPCB->suspendedStatus = 1;
75 
76         returnedPCB = tempPCB;
77     }
78 
79     return returnedPCB;
80 }
```

5.26 modules/R2/R2commands.c File Reference

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

Functions

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

5.26.1 Function Documentation

5.26.1.1 blockPCB()

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

Definition at line 99 of file R2commands.c.

```
100 { // ANASTASE WILL PROGRAM THIS FUNCTION
101
102     // find pcb and validate process name
103     PCB *pcb_to_block = findPCB(processName);
104
105     if (pcb_to_block != NULL)
106     {
107         pcb_to_block->runningStatus = -1; // blocked
108         removePCB(pcb_to_block);
109         insertPCB(pcb_to_block);
110     }
111 }
```

5.26.1.2 createPCB()

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

Definition at line 11 of file R2commands.c.

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

5.26.1.3 deletePCB()

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

Definition at line 51 of file R2commands.c.

```
52 { // BENJAMIN WILL PROGRAM THIS FUNCTION
53     /*
54     The deletePCB command will remove a PCB from the appropriate queue and then free all associated
       memory.
55     This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
56     */
57     /*
58     Error Checking:
59     Name must be valid.
60     */
61
62     if (strlen(processName) > 20)
63     { // Check if the process has a valid name.
64         char errMsg[100];
65         strcpy(errMsg, "The PCB could not be deleted as the name is longer than 20 characters!\n");
66         int errLen = strlen(errMsg);
67         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
68     }
69
70     PCB *PCB_to_delete = findPCB(processName);
71
72     if (PCB_to_delete == NULL)
73     {
74         char errMsg[42] = "The PCB you want to remove does not eist\n";
75         int errMsgLen = 42;
76         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
77     }
78     else
79     {
80         int result = removePCB(PCB_to_delete);
81
82         if (result == 1)
83         {
84             char errMsg[50];
85             strcpy(errMsg, "The PCB could not be successfully deleted\n");
86             int errLen = strlen(errMsg);
87             sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
88         }
89         else
90         {
91             char msg[50];
92             strcpy(msg, "The desired PCB was deleted\n");
93             int msgLen = strlen(msg);
94             sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
95         }
96     }
97 }
```

5.26.1.4 resumePCB()

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

Definition at line 161 of file R2commands.c.

```
162 { // COLTON WILL PROGRAM THIS FUNCTION
163     /*
164     Places a PCB in the not suspended state and reinserts it into the appropriate queue
165     */
170
171     PCB *PCBtoResume = findPCB(processName);
172
173     if (PCBtoResume == NULL || strlen(processName) > 20)
174     {
175         char nameError[] = "This is not a valid name.\n";
176         int printCount = strlen(nameError);
177         sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
178     }
179     else
180     {
```

```
181         removePCB(PCBtoResume);
182         PCBtoResume->suspendedStatus = 1;
183         insertPCB(PCBtoResume);
184     }
185 }
```

5.26.1.5 setPCBPriority()

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

Definition at line 187 of file R2commands.c.

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

5.26.1.6 showAll()

```
void showAll ( )
```

Definition at line 578 of file R2commands.c.

```
579 { // COLTON WILL PROGRAM THIS FUNCTION
580     /*
581     Displays the following information for each PCB in the ready and blocked queues:
582         Process Name
583         Class
584         State
585         Suspended Status
586         Priority
587     */
588     /*
589     Error Checking:
590     None
591     */
592
593     showReady();
594     showSuspendedReady();
595     showBlocked();
596     showSuspendedBlocked();
597 }
```

5.26.1.7 showBlocked()

```
void showBlocked ( )
```

Definition at line 525 of file R2commands.c.

```
526 { // ANASTASE WILL PROGRAM THIS FUNCTION
527     /*
528     Displays the following information for each PCB in the blocked queue:
529         Process Name
530         Class
531         State
532         Suspended Status
533         Priority
534         HEAD
535     */
536     /*
537     Error Checking:
538     None
539     */
540 
541     // check
542 
543     char print_message[30] = "The blocked queue:\n";
544     int message_size = strlen(print_message);
545     sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
546 
547     // printPCBs(blocked);
548     queue *tempQueue = getBlocked();
549     PCB *tempPtr = tempQueue->head; //PCB_container->head;
550     int count = tempQueue->count;
551 
552     if (count == 0)
553     {
554         // the queue is empty
555         char error_message[30] = "The queue is empty.\n";
556         int error_size = strlen(error_message);
557         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
558         return;
559     }
560     // The queue is not empty
561 
562     int value = 0;
563     // Testing purpose
564     //char print_message[38]="The blocke queue testing:\n";
565     //int message_size=strlen(print_message);
566     //sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
567 
568     while (value <= count)
569     { // testing for <== or <
570         // Print out the process
571         showPCB(tempPtr->processName);
572         // increment pcb*tempPtr, the loop variable.
573         tempPtr = tempPtr->nextPCB;
574         value++;
575     }
576 }
```

5.26.1.8 showPCB()

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

Definition at line 209 of file R2commands.c.

```
210 { // BENJAMIN WILL PROGRAM THIS FUNCTION
211     /*
212     Displays the following information for a PCB:
213         Process Name
214         Class
215         State
216         Suspended Status
217         Priority
218     */
219 
220     /*
221     Error Checking:
```

```
222     Name must be valid.
223     */
224
225     if (strlen(processName) > 20)
226     { // Check if the process has a valid name.
227         char errMsg[100];
228         strcpy(errMsg, "The PCB could not be shown as the name is longer than 20 characters!\n");
229         int errLen = strlen(errMsg);
230         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
231     }
232     else
233     {
234
235         PCB *PCB_to_show = findPCB(processName);
236
237         if (PCB_to_show == NULL)
238         { // Check to see if the PCB exists.
239             char errMsg[100];
240             strcpy(errMsg, "The PCB could not be shown, as it does not exist!\n");
241             int errLen = strlen(errMsg);
242             sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
243         }
244         else
245         {
246             // Print out the PCB name.
247             char nameMsg[50];
248             strcpy(nameMsg, "The process name is: ");
249             int nameMsgLen = strlen(nameMsg);
250             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
251             char name[20];
252             strcpy(name, PCB_to_show->processName);
253             int nameLen = strlen(name);
254             sys_req(WRITE, DEFAULT_DEVICE, name, &nameLen);
255             char newLine[1];
256             strcpy(newLine, "\n");
257             int newLineLen = 1;
258             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);
259
260             // Print out PCB class
261             char classMsg[50];
262             strcpy(classMsg, "The process class is: ");
263             int classMsgLen = strlen(classMsg);
264             sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
265
266             if (PCB_to_show->processClass == 'a')
267             {
268                 char appMsg[50];
269                 strcpy(appMsg, "application");
270                 int appMsgLen = strlen(appMsg);
271                 sys_req(WRITE, DEFAULT_DEVICE, appMsg, &appMsgLen);
272             }
273             else
274             {
275                 char sysMsg[50];
276                 strcpy(sysMsg, "system");
277                 int sysMsgLen = strlen(sysMsg);
278                 sys_req(WRITE, DEFAULT_DEVICE, sysMsg, &sysMsgLen);
279             }
280             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);
281
282             // Print out the PCB state
283
284             if (PCB_to_show->runningStatus == 0)
285             { // The process is ready.
286                 char stateMsg[50];
287                 strcpy(stateMsg, "The process is ready!\n");
288                 int stateMsgLen = strlen(stateMsg);
289                 sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
290             }
291             else if (PCB_to_show->runningStatus == -1)
292             { // The process is blocked.
293                 char stateMsg[50];
294                 strcpy(stateMsg, "The process is blocked!\n");
295                 int stateMsgLen = strlen(stateMsg);
296                 sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
297             }
298             else if (PCB_to_show->runningStatus == 1)
299             { // The process is running.
300                 char stateMsg[50];
301                 strcpy(stateMsg, "The process is running!\n");
302                 int stateMsgLen = strlen(stateMsg);
303                 sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
304             }
305
306             // Print out the PCB suspended status
307
308             if (PCB_to_show->suspendedStatus == 0)
```

```
309             { // The process is suspended
310                 char susMsg[50];
311                 strcpy(susMsg, "The process is suspended!\n");
312                 int susMsgLen = strlen(susMsg);
313                 sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
314             }
315             else if (PCB_to_show->suspendedStatus == 1)
316             { // The process is not suspended
317                 char susMsg[50];
318                 strcpy(susMsg, "The process is not suspended!\n");
319                 int susMsgLen = strlen(susMsg);
320                 sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
321             }
322 
323             // Print out the PCB priority
324             char priorityMsg[50];
325             int priorityMsgLen = 0;
326 
327             switch (PCB_to_show->priority)
328             {
329             case 0:
330                 strcpy(priorityMsg, "The process priority is 0!\n");
331                 priorityMsgLen = strlen(priorityMsg);
332                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
333                 break;
334 
335             case 1:
336                 strcpy(priorityMsg, "The process priority is 1!\n");
337                 priorityMsgLen = strlen(priorityMsg);
338                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
339                 break;
340 
341             case 2:
342                 strcpy(priorityMsg, "The process priority is 2!\n");
343                 priorityMsgLen = strlen(priorityMsg);
344                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
345                 break;
346 
347             case 3:
348                 strcpy(priorityMsg, "The process priority is 3!\n");
349                 priorityMsgLen = strlen(priorityMsg);
350                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
351                 break;
352 
353             case 4:
354                 strcpy(priorityMsg, "The process priority is 4!\n");
355                 priorityMsgLen = strlen(priorityMsg);
356                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
357                 break;
358 
359             case 5:
360                 strcpy(priorityMsg, "The process priority is 5!\n");
361                 priorityMsgLen = strlen(priorityMsg);
362                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
363                 break;
364 
365             case 6:
366                 strcpy(priorityMsg, "The process priority is 6!\n");
367                 priorityMsgLen = strlen(priorityMsg);
368                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
369                 break;
370 
371             case 7:
372                 strcpy(priorityMsg, "The process priority is 7!\n");
373                 priorityMsgLen = strlen(priorityMsg);
374                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
375                 break;
376 
377             case 8:
378                 strcpy(priorityMsg, "The process priority is 8!\n");
379                 priorityMsgLen = strlen(priorityMsg);
380                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
381                 break;
382 
383             case 9:
384                 strcpy(priorityMsg, "The process priority is 9!\n");
385                 priorityMsgLen = strlen(priorityMsg);
386                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
387                 break;
388 
389             default:
390                 break;
391             }
392         }
393     }
394 }
```

5.26.1.9 showReady()

```
void showReady ( )
```

Definition at line 396 of file R2commands.c.

```
397 { // COLTON WILL PROGRAM THIS FUNCTION
398     /*
399     Displays the following information for each PCB in the ready queue:
400         Process Name
401         Class
402         State
403         Suspended Status
404         Priority
405     */
406     /*
407     Error Checking:
408     None
409     */
410 
411     char message[] = "Printing the ready queue:\n";
412     int messLength = strlen(message);
413     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
414 
415     queue *tempQueue = getReady();
416     PCB *tempPCB = tempQueue->head;
417 
418     int loop = 0;
419     int count = tempQueue->count;
420 
421     if (count == 0)
422     {
423         // the queue is empty
424         char error_message[30] = "The queue is empty.\n";
425         int error_size = strlen(error_message);
426         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
427         return;
428     }
429 
430     while (loop <= count && tempPCB->nextPCB != NULL && count > 0)
431     {
432         showPCB(tempPCB->processName);
433         PCB *tempNext = tempPCB->nextPCB;
434         loop++;
435         tempPCB = tempNext;
436     }
437 }
```

5.26.1.10 showSuspendedBlocked()

```
void showSuspendedBlocked ( )
```

Definition at line 482 of file R2commands.c.

```
483 { // COLTON WILL PROGRAM THIS FUNCTION
484     /*
485     Displays the following information for each PCB in the suspended blocked queue:
486         Process Name
487         Class
488         State
489         Suspended Status
490         Priority
491     */
492     /*
493     Error Checking:
494     None
495     */
496 
497     char message[] = "Printing the suspended blocked queue:\n";
498     int messLength = strlen(message);
499     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
500 
501     queue *tempQueue = getSuspendedBlocked();
```

```
502     PCB *tempPCB = tempQueue->head;
503
504     int loop = 0;
505     int count = tempQueue->count;
506
507     if (count == 0)
508     {
509         // the queue is empty
510         char error_message[30] = "The queue is empty.\n";
511         int error_size = strlen(error_message);
512         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
513         return;
514     }
515
516     while (loop < count && tempPCB->nextPCB != NULL && count > 0)
517     {
518         showPCB(tempPCB->processName);
519         PCB *tempNext = tempPCB->nextPCB;
520         loop++;
521         tempPCB = tempNext;
522     }
523 }
```

5.26.1.11 showSuspendedReady()

```
void showSuspendedReady ( )
```

Definition at line 439 of file R2commands.c.

```
440 { // COLTON WILL PROGRAM THIS FUNCTION
441     /*
442     Displays the following information for each PCB in the suspended ready queue:
443         Process Name
444         Class
445         State
446         Suspended Status
447         Priority
448     */
449     /*
450     Error Checking:
451     None
452     */
453
454     char message[] = "Printing the suspended ready queue:\n";
455     int messLength = strlen(message);
456     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
457
458     queue *tempQueue = getSuspendedReady();
459     PCB *tempPCB = tempQueue->head;
460
461     int loop = 0;
462     int count = tempQueue->count;
463
464     if (count == 0)
465     {
466         // the queue is empty
467         char error_message[30] = "The queue is empty.\n";
468         int error_size = strlen(error_message);
469         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
470         return;
471     }
472
473     while (loop < count && tempPCB->nextPCB != NULL && count > 0)
474     {
475         showPCB(tempPCB->processName);
476         PCB *tempNext = tempPCB->nextPCB;
477         loop++;
478         tempPCB = tempNext;
479     }
480 }
```

5.26.1.12 suspendPCB()

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

Definition at line 135 of file R2commands.c.

```
136 { // COLTON WILL PROGRAM THIS FUNCTION
137     /*
138     Places a PCB in the suspended state and reinserts it into the appropriate queue
139     */
144
145     PCB *PCBtoSuspend = findPCB(processName);
146
147     if (PCBtoSuspend == NULL || strlen(processName) > 20)
148     {
149         char nameError[] = "This is not a valid name.\n";
150         int printCount = strlen(nameError);
151         sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
152     }
153     else
154     {
155         removePCB(PCBtoSuspend);
156         PCBtoSuspend->suspendedStatus = 0;
157         insertPCB(PCBtoSuspend);
158     }
159 }
```

5.26.1.13 unblockPCB()

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

Definition at line 113 of file R2commands.c.

```
114 { // ANASTASE WILL PROGRAM THIS FUNCTION
115
116     /*
117     Places a PCB in the unblocked state and reinserts it into the appropriate queue.
118     */
119     /*
120     Error Checking:
121     Name must be valid.
122
123     */
124
125     PCB *pcb_to_unblock = findPCB(processName);
126     if (pcb_to_unblock != NULL)
127     {
128         pcb_to_unblock->runningStatus = 0; // ready
129         removePCB(pcb_to_unblock);         // is this the right place to put that function?
130         insertPCB(pcb_to_unblock);
131     }
132 }
```

5.27 modules/R2/R2commands.h File Reference

Functions

- void createPCB (char *processName, unsigned 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 99 of file R2commands.c.

```
100 { // ANASTASE WILL PROGRAM THIS FUNCTION
101
102     // find pcb and validate process name
103     PCB *pcb_to_block = findPCB(processName);
104
105     if (pcb_to_block != NULL)
106     {
107         pcb_to_block->runningStatus = -1; // blocked
108         removePCB(pcb_to_block);
109         insertPCB(pcb_to_block);
110     }
111 }
```

5.27.1.2 createPCB()

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

Definition at line 11 of file R2commands.c.

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

5.27.1.3 deletePCB()

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

Definition at line 51 of file R2commands.c.

```
52 { // BENJAMIN WILL PROGRAM THIS FUNCTION
53     /*
54     The deletePCB command will remove a PCB from the appropriate queue and then free all associated
       memory.
55     This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
56     */
57     /*
58     Error Checking:
59     Name must be valid.
60     */
61
62     if (strlen(processName) > 20)
63     { // Check if the process has a valid name.
64         char errMsg[100];
65         strcpy(errMsg, "The PCB could not be deleted as the name is longer than 20 characters!\n");
66         int errLen = strlen(errMsg);
67         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
68     }
69
70     PCB *PCB_to_delete = findPCB(processName);
71
72     if (PCB_to_delete == NULL)
73     {
74         char errMsg[42] = "The PCB you want to remove does not eist\n";
75         int errMsgLen = 42;
76         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
77     }
78     else
79     {
80         int result = removePCB(PCB_to_delete);
81
82         if (result == 1)
83         {
84             char errMsg[50];
85             strcpy(errMsg, "The PCB could not be successfully deleted\n");
86             int errLen = strlen(errMsg);
87             sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
88         }
89         else
90         {
91             char msg[50];
92             strcpy(msg, "The desired PCB was deleted\n");
93             int msgLen = strlen(msg);
94             sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
95         }
96     }
97 }
```

5.27.1.4 resumePCB()

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

Definition at line 161 of file R2commands.c.

```
162 { // COLTON WILL PROGRAM THIS FUNCTION
163     /*
164     Places a PCB in the not suspended state and reinserts it into the appropriate queue
165     */
170
171     PCB *PCBtoResume = findPCB(processName);
172
173     if (PCBtoResume == NULL || strlen(processName) > 20)
174     {
175         char nameError[] = "This is not a valid name.\n";
176         int printCount = strlen(nameError);
177         sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
178     }
179     else
180     {
```

```
181         removePCB(PCBtoResume);
182         PCBtoResume->suspendedStatus = 1;
183         insertPCB(PCBtoResume);
184     }
185 }
```

5.27.1.5 setPCBPriority()

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

Definition at line 187 of file R2commands.c.

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

5.27.1.6 showAll()

```
void showAll ( )
```

Definition at line 578 of file R2commands.c.

```
579 { // COLTON WILL PROGRAM THIS FUNCTION
580     /*
581     Displays the following information for each PCB in the ready and blocked queues:
582         Process Name
583         Class
584         State
585         Suspended Status
586         Priority
587     */
588     /*
589     Error Checking:
590     None
591     */
592
593     showReady();
594     showSuspendedReady();
595     showBlocked();
596     showSuspendedBlocked();
597 }
```

5.27.1.7 showBlocked()

```
void showBlocked ( )
```

Definition at line 525 of file R2commands.c.

```
526 { // ANASTASE WILL PROGRAM THIS FUNCTION
527     /*
528     Displays the following information for each PCB in the blocked queue:
529         Process Name
530         Class
531         State
532         Suspended Status
533         Priority
534         HEAD
535     */
536     /*
537     Error Checking:
538     None
539     */
540 
541     // check
542 
543     char print_message[30] = "The blocked queue:\n";
544     int message_size = strlen(print_message);
545     sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
546 
547     // printPCBs(blocked);
548     queue *tempQueue = getBlocked();
549     PCB *tempPtr = tempQueue->head; //PCB_container->head;
550     int count = tempQueue->count;
551 
552     if (count == 0)
553     {
554         // the queue is empty
555         char error_message[30] = "The queue is empty.\n";
556         int error_size = strlen(error_message);
557         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
558         return;
559     }
560     // The queue is not empty
561 
562     int value = 0;
563     // Testing purpose
564     //char print_message[38]="The blocke queue testing:\n";
565     //int message_size=strlen(print_message);
566     //sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);
567 
568     while (value <= count)
569     { // testing for <== or <
570         // Print out the process
571         showPCB(tempPtr->processName);
572         // increment pcb*tempPtr, the loop variable.
573         tempPtr = tempPtr->nextPCB;
574         value++;
575     }
576 }
```

5.27.1.8 showPCB()

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

Definition at line 209 of file R2commands.c.

```
210 { // BENJAMIN WILL PROGRAM THIS FUNCTION
211     /*
212     Displays the following information for a PCB:
213         Process Name
214         Class
215         State
216         Suspended Status
217         Priority
218     */
219 
220     /*
221     Error Checking:
```

```
222     Name must be valid.
223     */
224
225     if (strlen(processName) > 20)
226     { // Check if the process has a valid name.
227         char errMsg[100];
228         strcpy(errMsg, "The PCB could not be shown as the name is longer than 20 characters!\n");
229         int errLen = strlen(errMsg);
230         sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
231     }
232     else
233     {
234
235         PCB *PCB_to_show = findPCB(processName);
236
237         if (PCB_to_show == NULL)
238         { // Check to see if the PCB exists.
239             char errMsg[100];
240             strcpy(errMsg, "The PCB could not be shown, as it does not exist!\n");
241             int errLen = strlen(errMsg);
242             sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
243         }
244         else
245         {
246             // Print out the PCB name.
247             char nameMsg[50];
248             strcpy(nameMsg, "The process name is: ");
249             int nameMsgLen = strlen(nameMsg);
250             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
251             char name[20];
252             strcpy(name, PCB_to_show->processName);
253             int nameLen = strlen(name);
254             sys_req(WRITE, DEFAULT_DEVICE, name, &nameLen);
255             char newLine[1];
256             strcpy(newLine, "\n");
257             int newLineLen = 1;
258             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);
259
260             // Print out PCB class
261             char classMsg[50];
262             strcpy(classMsg, "The process class is: ");
263             int classMsgLen = strlen(classMsg);
264             sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
265
266             if (PCB_to_show->processClass == 'a')
267             {
268                 char appMsg[50];
269                 strcpy(appMsg, "application");
270                 int appMsgLen = strlen(appMsg);
271                 sys_req(WRITE, DEFAULT_DEVICE, appMsg, &appMsgLen);
272             }
273             else
274             {
275                 char sysMsg[50];
276                 strcpy(sysMsg, "system");
277                 int sysMsgLen = strlen(sysMsg);
278                 sys_req(WRITE, DEFAULT_DEVICE, sysMsg, &sysMsgLen);
279             }
280             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);
281
282             // Print out the PCB state
283
284             if (PCB_to_show->runningStatus == 0)
285             { // The process is ready.
286                 char stateMsg[50];
287                 strcpy(stateMsg, "The process is ready!\n");
288                 int stateMsgLen = strlen(stateMsg);
289                 sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
290             }
291             else if (PCB_to_show->runningStatus == -1)
292             { // The process is blocked.
293                 char stateMsg[50];
294                 strcpy(stateMsg, "The process is blocked!\n");
295                 int stateMsgLen = strlen(stateMsg);
296                 sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
297             }
298             else if (PCB_to_show->runningStatus == 1)
299             { // The process is running.
300                 char stateMsg[50];
301                 strcpy(stateMsg, "The process is running!\n");
302                 int stateMsgLen = strlen(stateMsg);
303                 sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
304             }
305
306             // Print out the PCB suspended status
307
308             if (PCB_to_show->suspendedStatus == 0)
```

```
309             { // The process is suspended
310                 char susMsg[50];
311                 strcpy(susMsg, "The process is suspended!\n");
312                 int susMsgLen = strlen(susMsg);
313                 sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
314             }
315             else if (PCB_to_show->suspendedStatus == 1)
316             { // The process is not suspended
317                 char susMsg[50];
318                 strcpy(susMsg, "The process is not suspended!\n");
319                 int susMsgLen = strlen(susMsg);
320                 sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
321             }
322 
323             // Print out the PCB priority
324             char priorityMsg[50];
325             int priorityMsgLen = 0;
326 
327             switch (PCB_to_show->priority)
328             {
329             case 0:
330                 strcpy(priorityMsg, "The process priority is 0!\n");
331                 priorityMsgLen = strlen(priorityMsg);
332                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
333                 break;
334 
335             case 1:
336                 strcpy(priorityMsg, "The process priority is 1!\n");
337                 priorityMsgLen = strlen(priorityMsg);
338                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
339                 break;
340 
341             case 2:
342                 strcpy(priorityMsg, "The process priority is 2!\n");
343                 priorityMsgLen = strlen(priorityMsg);
344                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
345                 break;
346 
347             case 3:
348                 strcpy(priorityMsg, "The process priority is 3!\n");
349                 priorityMsgLen = strlen(priorityMsg);
350                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
351                 break;
352 
353             case 4:
354                 strcpy(priorityMsg, "The process priority is 4!\n");
355                 priorityMsgLen = strlen(priorityMsg);
356                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
357                 break;
358 
359             case 5:
360                 strcpy(priorityMsg, "The process priority is 5!\n");
361                 priorityMsgLen = strlen(priorityMsg);
362                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
363                 break;
364 
365             case 6:
366                 strcpy(priorityMsg, "The process priority is 6!\n");
367                 priorityMsgLen = strlen(priorityMsg);
368                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
369                 break;
370 
371             case 7:
372                 strcpy(priorityMsg, "The process priority is 7!\n");
373                 priorityMsgLen = strlen(priorityMsg);
374                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
375                 break;
376 
377             case 8:
378                 strcpy(priorityMsg, "The process priority is 8!\n");
379                 priorityMsgLen = strlen(priorityMsg);
380                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
381                 break;
382 
383             case 9:
384                 strcpy(priorityMsg, "The process priority is 9!\n");
385                 priorityMsgLen = strlen(priorityMsg);
386                 sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
387                 break;
388 
389             default:
390                 break;
391             }
392         }
393     }
394 }
```

5.27.1.9 showReady()

```
void showReady ( )
```

Definition at line 396 of file R2commands.c.

```
397 { // COLTON WILL PROGRAM THIS FUNCTION
398     /*
399     Displays the following information for each PCB in the ready queue:
400         Process Name
401         Class
402         State
403         Suspended Status
404         Priority
405     */
406     /*
407     Error Checking:
408     None
409     */
410 
411     char message[] = "Printing the ready queue:\n";
412     int messLength = strlen(message);
413     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
414 
415     queue *tempQueue = getReady();
416     PCB *tempPCB = tempQueue->head;
417 
418     int loop = 0;
419     int count = tempQueue->count;
420 
421     if (count == 0)
422     {
423         // the queue is empty
424         char error_message[30] = "The queue is empty.\n";
425         int error_size = strlen(error_message);
426         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
427         return;
428     }
429 
430     while (loop <= count && tempPCB->nextPCB != NULL && count > 0)
431     {
432         showPCB(tempPCB->processName);
433         PCB *tempNext = tempPCB->nextPCB;
434         loop++;
435         tempPCB = tempNext;
436     }
437 }
```

5.27.1.10 showSuspendedBlocked()

```
void showSuspendedBlocked ( )
```

Definition at line 482 of file R2commands.c.

```
483 { // COLTON WILL PROGRAM THIS FUNCTION
484     /*
485     Displays the following information for each PCB in the suspended blocked queue:
486         Process Name
487         Class
488         State
489         Suspended Status
490         Priority
491     */
492     /*
493     Error Checking:
494     None
495     */
496 
497     char message[] = "Printing the suspended blocked queue:\n";
498     int messLength = strlen(message);
499     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
500 
501     queue *tempQueue = getSuspendedBlocked();
```

```
502     PCB *tempPCB = tempQueue->head;
503
504     int loop = 0;
505     int count = tempQueue->count;
506
507     if (count == 0)
508     {
509         // the queue is empty
510         char error_message[30] = "The queue is empty.\n";
511         int error_size = strlen(error_message);
512         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
513         return;
514     }
515
516     while (loop < count && tempPCB->nextPCB != NULL && count > 0)
517     {
518         showPCB(tempPCB->processName);
519         PCB *tempNext = tempPCB->nextPCB;
520         loop++;
521         tempPCB = tempNext;
522     }
523 }
```

5.27.1.11 showSuspendedReady()

```
void showSuspendedReady ( )
```

Definition at line 439 of file R2commands.c.

```
440 { // COLTON WILL PROGRAM THIS FUNCTION
441     /*
442     Displays the following information for each PCB in the suspended ready queue:
443         Process Name
444         Class
445         State
446         Suspended Status
447         Priority
448     */
449     /*
450     Error Checking:
451     None
452     */
453
454     char message[] = "Printing the suspended ready queue:\n";
455     int messLength = strlen(message);
456     sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);
457
458     queue *tempQueue = getSuspendedReady();
459     PCB *tempPCB = tempQueue->head;
460
461     int loop = 0;
462     int count = tempQueue->count;
463
464     if (count == 0)
465     {
466         // the queue is empty
467         char error_message[30] = "The queue is empty.\n";
468         int error_size = strlen(error_message);
469         sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
470         return;
471     }
472
473     while (loop < count && tempPCB->nextPCB != NULL && count > 0)
474     {
475         showPCB(tempPCB->processName);
476         PCB *tempNext = tempPCB->nextPCB;
477         loop++;
478         tempPCB = tempNext;
479     }
480 }
```

5.27.1.12 suspendPCB()

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

Definition at line 135 of file R2commands.c.

```
136 { // COLTON WILL PROGRAM THIS FUNCTION
137     /*
138     Places a PCB in the suspended state and reinserts it into the appropriate queue
139     */
140 
141     PCB *PCBtoSuspend = findPCB(processName);
142 
143     if (PCBtoSuspend == NULL || strlen(processName) > 20)
144     {
145         char nameError[] = "This is not a valid name.\n";
146         int printCount = strlen(nameError);
147         sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
148     }
149     else
150     {
151         removePCB(PCBtoSuspend);
152         PCBtoSuspend->suspendedStatus = 0;
153         insertPCB(PCBtoSuspend);
154     }
155 }
```

5.27.1.13 unblockPCB()

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

Definition at line 113 of file R2commands.c.

```
114 { // ANASTASE WILL PROGRAM THIS FUNCTION
115 
116     /*
117     Places a PCB in the unblocked state and reinserts it into the appropriate queue.
118     */
119     /*
120     Error Checking:
121     Name must be valid.
122 
123     */
124 
125     PCB *pcb_to_unblock = findPCB(processName);
126     if (pcb_to_unblock != NULL)
127     {
128         pcb_to_unblock->runningStatus = 0; // ready
129         removePCB(pcb_to_unblock);         // is this the right place to put that function?
130         insertPCB(pcb_to_unblock);
131     }
132 }
```

5.28 README.md File Reference