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

4.15.1 Detailed Description

Definition at line 1 of file R2_Internal_Functions_And_Structures.h.

4.15.2 Member Data Documentation

4.15.2.1 nextPCB

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

Definition at line 11 of file R2_Internal_Functions_And_Structures.h.

4.15.2.2 prevPCB

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

Definition at line 12 of file R2_Internal_Functions_And_Structures.h.

4.15.2.3 priority

```
int PCB::priority
```

Definition at line 5 of file R2_Internal_Functions_And_Structures.h.

4.15.2.4 processClass

```
char PCB::processClass
```

Definition at line 4 of file R2_Internal_Functions_And_Structures.h.

4.15.2.5 processName

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

Definition at line 3 of file R2_Internal_Functions_And_Structures.h.

4.15.2.6 runningStatus

```
int PCB::runningStatus
```

Definition at line 6 of file R2_Internal_Functions_And_Structures.h.

4.15.2.7 stack

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

Definition at line 8 of file R2_Internal_Functions_And_Structures.h.

4.15.2.8 stackBase

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

Definition at line 10 of file R2_Internal_Functions_And_Structures.h.

4.15.2.9 stackTop

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

Definition at line 9 of file R2_Internal_Functions_And_Structures.h.

4.15.2.10 suspendedStatus

```
int PCB::suspendedStatus
```

Definition at line 7 of file R2_Internal_Functions_And_Structures.h.

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

- modules/R2/R2_Internal_Functions_And_Structures.h

4.16 queue Struct Reference

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

Public Attributes

- int count
- PCB ∗head
- PCB ∗tail

4.16.1 Detailed Description

Definition at line 15 of file R2_Internal_Functions_And_Structures.h.

4.16.2 Member Data Documentation

4.16.2.1 count

```
int queue::count
```

Definition at line 17 of file R2_Internal_Functions_And_Structures.h.

4.16.2.2 head

```
PCB∗ queue::head
```

Definition at line 18 of file R2_Internal_Functions_And_Structures.h.

4.16.2.3 tail

```
PCB∗ queue::tail
```

Definition at line 19 of file R2_Internal_Functions_And_Structures.h.

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

- modules/R2/R2_Internal_Functions_And_Structures.h

Chapter 5

File Documentation

5.1 include/core/asm.h File Reference

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

5.2 include/core/interrupts.h File Reference

Functions

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

5.2.1 Function Documentation

5.2.1.1 init_irq()

```
void init_irq (
            void  )
```

Definition at line 66 of file interrupts.c.

```
{
  int i;

  // Necessary interrupt handlers for protected mode
  u32int isrs[17] = {
    (u32int)divide_error,
    (u32int)debug,
    (u32int)nmi,
    (u32int)breakpoint,
    (u32int)overflow,
    (u32int)bounds,
    (u32int)invalid_op,
    (u32int)device_not_available,
    (u32int)double_fault,
    (u32int)coprocessor_segment,
    (u32int)invalid_tss,
    (u32int)segment_not_present,
    (u32int)stack_segment,
    (u32int)general_protection,
    (u32int)page_fault,
    (u32int)reserved,
    (u32int)coprocessor
  };

  // Install handlers; 0x08=sel, 0x8e=flags
  for(i=0; i<32; i++){
    if (i<17) idt_set_gate(i, isrs[i], 0x08, 0x8e);
    else idt_set_gate(i, (u32int)reserved, 0x08, 0x8e);
  }
  // Ignore interrupts from the real time clock
  idt_set_gate(0x08, (u32int)rtc_isr, 0x08, 0x8e);
}
```

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.

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

5.4.2.2 polling()

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

Definition at line 92 of file serial.c.

```
{
  // insert your code to gather keyboard input via the technique of polling.

  char keyboard_character;

  int cursor = 0;

  char log[] = {'\0', '\0', '\0', '\0'};

  int characters_in_buffer = 0;

  while (1)
  {

    if (inb(COM1 + 5) & 1)
    {                                 // is there input char?
      keyboard_character = inb(COM1); //read the char from COM1

      if (keyboard_character == '\n' || keyboard_character == '\r')
      { // HANDLEING THE CARRIAGE RETURN AND NEW LINE CHARACTERS

        buffer[characters_in_buffer] = '  \0';
        break;
      }
      else if ((keyboard_character == 127 || keyboard_character == 8) && cursor > 0)
      { // HANDELING THE BACKSPACE CHARACTER

        //serial_println("Handleing backspace character.");
        serial_print("\033[K");

        buffer[cursor - 1] = ' \0';
        serial_print("\b \b");
        serial_print(buffer + cursor);
        cursor--;

        int temp_cursor = cursor;

        while (buffer[temp_cursor + 1] != ' \0')
        {
          buffer[temp_cursor] = buffer[temp_cursor + 1];
          buffer[temp_cursor + 1] = ' \0';
          temp_cursor++;
        }

        characters_in_buffer--;
        cursor = characters_in_buffer;
      }
      else if (keyboard_character == '~' && cursor < 99)
      { //HANDLING THE DELETE KEY
        // \033[3~

        serial_print("\033[K");

        buffer[cursor + 1] = ' \0';
        serial_print("\b \b");
        serial_print(buffer + cursor);

        int temp_cursor = cursor + 1;

        while (buffer[temp_cursor + 1] != ' \0')
        {
          buffer[temp_cursor] = buffer[temp_cursor + 1];
          buffer[temp_cursor + 1] = ' \0';
          temp_cursor++;
        }

        characters_in_buffer--;
        cursor = characters_in_buffer;
      }
      else if (keyboard_character == '\033')
      { // HANDLEING FIRST CHARACTER FOR ARROW KEYS

        log[0] = keyboard_character;
      }
      else if (keyboard_character == '[' && log[0] == '\033')
      { // HANDLEING SECOND CHARACTER FOR ARROW KEYS

        log[1] = keyboard_character;
      }
      else if (log[0] == '\033' && log[1] == '[')
      { // HANDLEING LAST CHARACTER FOR ARROW KEYS
        log[2] = keyboard_character;

        if (keyboard_character == 'A')
        { //Up arrow
          //Call a history function from the commhand or do nothing
        }
        else if (keyboard_character == 'B')
        { //Down arrow
          //Call a history command from the commhand or do nothing
        }
        else if (keyboard_character == 'C' && cursor != 99)
        { //Right arrow

          serial_print("\033[C");
          cursor++;
        }
        else if (keyboard_character == 'D' && cursor != 0)
        { //Left arrow

          serial_print("\033[D");
          cursor--;
        }

        memset(log, '\0', 4);
      }
      else
      {

        if (cursor == 0 && buffer[cursor] == ' \0') //Adding character at beginning of buffer
        {
          buffer[cursor] = keyboard_character;
          serial_print(&keyboard_character);
          cursor++;
        }
        else if (buffer[cursor] == ' \0') //Adding character at the end of the buffer
        {
          buffer[cursor] = keyboard_character;
          serial_print(&keyboard_character);
          cursor++;
        }
        else //Inserting character to the middle of the buffer
        {
          char temp_buffer[strlen(buffer)];
          memset(temp_buffer, ' \0', strlen(buffer));

          int temp_cursor = 0;
          while (temp_cursor <= characters_in_buffer) //Filling the temp_buffer with all of the characters from buffer, and inserting the new character.
          {
            if (temp_cursor < cursor)
            {
              temp_buffer[temp_cursor] = buffer[temp_cursor];
            }
            else if (temp_cursor > cursor)
            {
              temp_buffer[temp_cursor] = buffer[temp_cursor - 1];
            }
            else
            { //temp_cursor == cursor
              temp_buffer[temp_cursor] = keyboard_character;
            }
            temp_cursor++;
          }

          temp_cursor = 0;
          int temp_buffer_size = strlen(temp_buffer);
          while (temp_cursor <= temp_buffer_size) //Setting the contents of the buffer equal to the temp_buffer.
          {
            buffer[temp_cursor] = temp_buffer[temp_cursor];
            temp_cursor++;
          }

          serial_print("\033[K");
          serial_print(&keyboard_character);
          serial_print(buffer + cursor + 1);
          cursor++;
        }
        characters_in_buffer++;
      }
    }
  }

  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.

```
{
  gdt_entry *new_entry = &gdt_entries[idx];
  new_entry->base_low  = (base & 0xFFFF);
  new_entry->base_mid  = (base » 16) & 0xFF;
  new_entry->base_high = (base » 24) & 0xFF;
  new_entry->limit_low = (limit & 0xFFFF);
  new_entry->flags    = (limit » 16) & 0xFF;
  new_entry->flags |= flags & 0xF0;
  new_entry->access = access;
}
```

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.

```
{
  idt_entry *new_entry = &idt_entries[idx];
  new_entry->base_low  = (base &  0xFFFF);
  new_entry->base_high = (base » 16) & 0xFFFF;
  new_entry->sselect   = sel;
  new_entry->zero = 0;
  new_entry->flags = flags;
}
```

5.5.1.4 init_gdt()

```
void init_gdt ( )
```

Definition at line 75 of file tables.c.

```
{
  gdt_ptr.limit = 5 * sizeof(gdt_entry) - 1;
  gdt_ptr.base  = (u32int) gdt_entries;

  u32int limit = 0xFFFFFFFF;
  gdt_init_entry(0, 0, 0, 0, 0);           //required null segment
  gdt_init_entry(1, 0, limit, 0x9A, 0xCF); //code segment
  gdt_init_entry(2, 0, limit, 0x92, 0xCF); //data segment
  gdt_init_entry(3, 0, limit, 0xFA, 0xCF); //user mode code segment
  gdt_init_entry(4, 0, limit, 0xF2, 0xCF); //user mode data segment

  write_gdt_ptr((u32int) &gdt_ptr, sizeof(gdt_ptr));
}
```

5.5.1.5 init_idt()

```
void init_idt ( )
```

Definition at line 43 of file tables.c.

```
{
  idt_ptr.limit = 256*sizeof(idt_descriptor) - 1;
  idt_ptr.base  = (u32int)idt_entries;
  memset(idt_entries, 0, 256*sizeof(idt_descriptor));

  write_idt_ptr((u32int)&idt_ptr);
}
```

5.5.2 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.

```
{
  u32int *addr;

  // Allocate on the kernel heap if one has been created
  if (kheap != 0){
    addr = (u32int*)alloc(size, kheap, page_align);
    if (phys_addr){
      page_entry *page = get_page((u32int)addr, kdir, 0);
      *phys_addr = (page->frameaddr*0x1000) + ((u32int)addr & 0xFFF);
    }
    return (u32int)addr;
  }
  // Else, allocate directly from physical memory
  else {
    if (page_align && (phys_alloc_addr & 0xFFFFF000)){
      phys_alloc_addr &= 0xFFFFF000;
      phys_alloc_addr += 0x1000;
    }
    addr = (u32int*)phys_alloc_addr;
    if (phys_addr){
      *phys_addr = phys_alloc_addr;
    }
    phys_alloc_addr += size;
    return (u32int)addr;
  }
}
```

5.6.2.2 alloc()

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

Definition at line 57 of file heap.c.

```
{
  no_warn(size||align||h);
  static u32int heap_addr = KHEAP_BASE;

  u32int base = heap_addr;
  heap_addr += size;

  if (heap_addr > KHEAP_BASE + KHEAP_MIN)
    serial_println("Heap is full!");

  return base;
}
```

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.

```
{
  return _kmalloc(size,0,0);
}
```

5.6.2.6 make_heap()

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

Definition at line 71 of file heap.c.

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

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.

```
{
  u32int frame  = addr/page_size;
  u32int index  = frame/32;
  u32int offset = frame%32;
  frames[index] &= ~(1 « offset);
}
```

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.

```
{
  u32int frame  = addr/page_size;
  u32int index  = frame/32;
  u32int offset = frame%32;
  return (frames[index] & (1 « offset));
}
```

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.

```
{
  u32int phys_addr;
  u32int index = addr / page_size / 1024;
  u32int offset = addr / page_size % 1024;

  //return it if it exists
  if (dir->tables[index])
    return &dir->tables[index]->pages[offset];

  //create it
  else if (make_table){
    dir->tables[index] = (page_table*)_kmalloc(sizeof(page_table), 1, &phys_addr);
    dir->tables_phys[index] = phys_addr | 0x7; //enable present, writable
    return &dir->tables[index]->pages[offset];
  }
  else return 0;
}
```

5.7.2.5 init_paging()

```
void init_paging ( )
```

Definition at line 111 of file paging.c.

```
{
  //create frame bitmap
  nframes = (u32int)(mem_size/page_size);
  frames = (u32int*)kmalloc(nframes/32);
  memset(frames, 0, nframes/32);

  //create kernel directory
  kdir = (page_dir*)_kmalloc(sizeof(page_dir), 1, 0); //page aligned
  memset(kdir, 0, sizeof(page_dir));

  //get pages for kernel heap
  u32int i = 0x0;
  for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=1){
    get_page(i,kdir,1);
  }

  //perform identity mapping of used memory
  //note: placement_addr gets incremented in get_page,
  //so we're mapping the first frames as well
  i = 0x0;
  while (i < (phys_alloc_addr+0x10000)){
    new_frame(get_page(i,kdir,1));
    i += page_size;
  }

  //allocate heap frames now that the placement addr has increased.
  //placement addr increases here for heap
  for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN);i+=PAGE_SIZE){
    new_frame(get_page(i,kdir,1));
  }

  //load the kernel page directory; enable paging
  load_page_dir(kdir);

  //setup the kernel heap
  kheap = make_heap(KHEAP_BASE, KHEAP_SIZE, KHEAP_BASE+KHEAP_MIN);
}
```

5.7.2.6 load_page_dir()

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

Definition at line 158 of file paging.c.

```
{
  cdir = new_dir;
  asm volatile ("mov %0,%%cr3":: "b"(&cdir->tables_phys[0]));
  u32int cr0;
  asm volatile ("mov %%cr0,%0": "=b"(cr0));
  cr0 |= 0x80000000;
  asm volatile ("mov %0,%%cr0":: "b"(cr0));
}
```

5.7.2.7 new_frame()

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

Definition at line 173 of file paging.c.

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

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

5.7.2.8 set_bit()

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

Definition at line 32 of file paging.c.

```
{
  u32int frame  = addr/page_size;
  u32int index  = frame/32;
  u32int offset = frame%32;
  frames[index] |= (1 « offset);
}
```

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.

```
{
  int res=0;
   int charVal=0;
   char sign = ' ';
   char c = *s;


   while(isspace(&c)){ ++s; c = *s;} // advance past whitespace


   if (*s == '-' || *s == '+') sign = *(s++); // save the sign


   while(*s != '\0'){
        charVal = *s - 48;
    res = res * 10 + charVal;
    s++;

   }


   if ( sign == '-') res=res * -1;

  return res; // return integer
}
```

5.8.1.2 isspace()

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

Definition at line 119 of file string.c.

```
{
  if (*c == ' '  ||
      *c == '\n' ||
      *c == '\r' ||
      *c == '\f' ||
      *c == '\t' ||
      *c == '\v'){
    return 1;
  }
  return 0;
}
```

5.8.1.3 memset()

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

Definition at line 137 of file string.c.

```
{
  unsigned char *p = (unsigned char *) s;
  while(n--){
    *p++ = (unsigned char) c;
  }
  return s;
}
```

5.8.1.4 strcat()

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

Definition at line 106 of file string.c.

```
{
  char *rc = s1;
  if (*s1) while(*++s1);
  while( (*s1++ = *s2++) );
  return rc;
}
```

5.8.1.5 strcmp()

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

Definition at line 79 of file string.c.

```
{

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

5.8.1.6 strcpy()

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

Definition at line 36 of file string.c.

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

5.8.1.7 strlen()

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

Definition at line 24 of file string.c.

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

5.8.1.8 strtok()

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

Definition at line 151 of file string.c.

```
{
 static char *tok_tmp = NULL;
 const char *p = s2;

 //new string
 if (s1!=NULL){
  tok_tmp = s1;
 }
 //old string cont'd
 else {
  if (tok_tmp==NULL){
   return NULL;
  }
  s1 = tok_tmp;
 }

 //skip leading s2 characters
 while ( *p && *s1 ){
  if (*s1==*p){
   ++s1;
   p = s2;
   continue;
  }
  ++p;
 }

 //no more to parse
 if (!*s1){
  return (tok_tmp = NULL);
 }

 //skip non-s2 characters
 tok_tmp = s1;
 while (*tok_tmp){
  p = s2;
  while (*p){
   if (*tok_tmp==*p++){
  *tok_tmp++ = '\0';
  return s1;
   }
  }
  ++tok_tmp;
 }

 //end of string
 tok_tmp = NULL;
 return s1;
}
```

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.

```
{
  int f;
  asm volatile ("pushf\n\t"
        "popl %0"
        : "=g"(f));
  return f & (1 « 9);
}
```

5.9.3.2 klogv()

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

Definition at line 11 of file system.c.

```
{
  char logmsg[64] = {'\0'}, prefix[] = "klogv: ";
  strcat(logmsg, prefix);
  strcat(logmsg, msg);
  serial_println(logmsg);
}
```

5.9.3.3 kpanic()

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

Definition at line 24 of file system.c.

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

5.10 kernel/core/interrupts.c File Reference

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

Macros

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

Functions

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

Variables

- idt_entry idt_entries [256]

5.10.1 Macro Definition Documentation

5.10.1.1 ICW1

```
#define ICW1 0x11
```

Definition at line 20 of file interrupts.c.

5.10.1.2 ICW4

```
#define ICW4 0x01
```

Definition at line 21 of file interrupts.c.

5.10.1.3 io_wait

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

Definition at line 28 of file interrupts.c.

5.10.1.4 PIC1

```
#define PIC1 0x20
```

Definition at line 16 of file interrupts.c.

5.10.1.5 PIC2

```
#define PIC2 0xA0
```

Definition at line 17 of file interrupts.c.

5.10.2 Function Documentation

5.10.2.1 bounds()

```
void bounds ( )
```

5.10.2.2 breakpoint()

```
void breakpoint ( )
```

5.10.2.3 coprocessor()

```
void coprocessor ( )
```

5.10.2.4 coprocessor_segment()

```
void coprocessor_segment ( )
```

5.10.2.5 debug()

```
void debug ( )
```

5.10.2.6 device_not_available()

```
void device_not_available ( )
```

5.10.2.7 divide_error()

```
void divide_error ( )
```

5.10.2.8 do_bounds()

```
void do_bounds ( )
```

Definition at line 149 of file interrupts.c.

```
{
  kpanic("Bounds error");
}
```

5.10.2.9 do_breakpoint()

```
void do_breakpoint ( )
```

Definition at line 141 of file interrupts.c.

```
{
  kpanic("Breakpoint");
}
```

5.10.2.10 do_coprocessor()

```
void do_coprocessor ( )
```

Definition at line 193 of file interrupts.c.

```
{
  kpanic("Coprocessor error");
}
```

5.10.2.11 do_coprocessor_segment()

```
void do_coprocessor_segment ( )
```

Definition at line 165 of file interrupts.c.

```
{
  kpanic("Coprocessor segment error");
}
```

5.10.2.12 do_debug()

```
void do_debug ( )
```

Definition at line 133 of file interrupts.c.

```
{
  kpanic("Debug");
}
```

5.10.2.13 do_device_not_available()

```
void do_device_not_available ( )
```

Definition at line 157 of file interrupts.c.

```
{
  kpanic("Device not available");
}
```

5.10.2.14 do_divide_error()

```
void do_divide_error ( )
```

Definition at line 129 of file interrupts.c.

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

5.10.2.15 do_double_fault()

```
void do_double_fault ( )
```

Definition at line 161 of file interrupts.c.

```
{
  kpanic("Double fault");
}
```

5.10.2.16 do_general_protection()

```
void do_general_protection ( )
```

Definition at line 181 of file interrupts.c.

```
{
  kpanic("General protection fault");
}
```

5.10.2.17 do_invalid_op()

```
void do_invalid_op ( )
```

Definition at line 153 of file interrupts.c.

```
{
  kpanic("Invalid operation");
}
```

5.10.2.18 do_invalid_tss()

```
void do_invalid_tss ( )
```

Definition at line 169 of file interrupts.c.

```
{
  kpanic("Invalid TSS");
}
```

5.10.2.19 do_isr()

```
void do_isr ( )
```

Definition at line 53 of file interrupts.c.

```
{
  char in = inb(COM2);
  serial_print(&in);
  serial_println("here");
  outb(0x20,0x20); //EOI
}
```

5.10.2.20 do_nmi()

```
void do_nmi ( )
```

Definition at line 137 of file interrupts.c.

```
{
  kpanic("NMI");
}
```

5.10.2.21 do_overflow()

```
void do_overflow ( )
```

Definition at line 145 of file interrupts.c.

```
{
  kpanic("Overflow error");
}
```

5.10.2.22 do_page_fault()

```
void do_page_fault ( )
```

Definition at line 185 of file interrupts.c.

```
{
  kpanic("Page Fault");
}
```

5.10.2.23 do_reserved()

```
void do_reserved ( )
```

Definition at line 189 of file interrupts.c.

```
{
  serial_println("die: reserved");
}
```

5.10.2.24 do_segment_not_present()

```
void do_segment_not_present ( )
```

Definition at line 173 of file interrupts.c.

```
{
  kpanic("Segment not present");
}
```

5.10.2.25 do_stack_segment()

```
void do_stack_segment ( )
```

Definition at line 177 of file interrupts.c.

```
{
  kpanic("Stack segment error");
}
```

5.10.2.26 double_fault()

```
void double_fault ( )
```

5.10.2.27 general_protection()

```
void general_protection ( )
```

5.10.2.28 init_irq()

```
void init_irq (
            void  )
```

Definition at line 66 of file interrupts.c.

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

5.10.2.29 init_pic()

```
void init_pic (
            void  )
```

Definition at line 106 of file interrupts.c.

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

5.10.2.30 invalid_op()

```
void invalid_op ( )
```

5.10.2.31 invalid_tss()

```
void invalid_tss ( )
```

5.10.2.32 isr0()

```
void isr0 ( )
```

5.10.2.33 nmi()

```
void nmi ( )
```

5.10.2.34 overflow()

```
void overflow ( )
```

5.10.2.35 page_fault()

```
void page_fault ( )
```

5.10.2.36 reserved()

```
void reserved ( )
```

5.10.2.37 rtc_isr()

```
void rtc_isr ( )
```

5.10.2.38 segment_not_present()

```
void segment_not_present ( )
```

5.10.2.39 stack_segment()

```
void stack_segment ( )
```

5.10.3 Variable Documentation

5.10.3.1 idt_entries

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

Definition at line 17 of file tables.c.

5.11 kernel/core/kmain.c File Reference

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

Functions

- void kmain (void)

5.11.1 Function Documentation

5.11.1.1 kmain()

```
void kmain (
            void )
```

Definition at line 27 of file kmain.c.

```
{
   // extern uint32_t magic;
   // Uncomment if you want to access the multiboot header
   // extern void *mbd;
   // char *boot_loader_name = (char*)((long*)mbd)[16];

   // 0) Initialize Serial I/O
   // functions to initialize serial I/O can be found in serial.c
   // there are 3 functions to call

   init_serial(COM1);
   set_serial_in(COM1);
   set_serial_out(COM1);

   klogv("Starting MPX boot sequence...");
   klogv("Initialized serial I/O on COM1 device...");

   // 1) Initialize the support software by identifying the current
   //     MPX Module.  This will change with each module.
   // you will need to call mpx_init from the mpx_supt.c

   mpx_init(MODULE_R2);

   // 2) Check that the boot was successful and correct when using grub
   // Comment this when booting the kernel directly using QEMU, etc.
   //if ( magic != 0x2BADB002 ){
   //  kpanic("Boot was not error free. Halting.");
   //}

   // 3) Descriptor Tables -- tables.c
   //  you will need to initialize the global
   // this keeps track of allocated segments and pages
   klogv("Initializing descriptor tables...");

   init_gdt();
   init_idt();

   init_pic();
   sti();

   // 4)  Interrupt vector table --  tables.c
   // this creates and initializes a default interrupt vector table
   // this function is in tables.c

   init_irq();

   klogv("Interrupt vector table initialized!");

   // 5) Virtual Memory -- paging.c  -- init_paging
   //  this function creates the kernel's heap
   //  from which memory will be allocated when the program calls
   // sys_alloc_mem UNTIL the memory management module  is completed
   // this allocates memory using discrete "pages" of physical memory
   // NOTE:  You will only have about 70000 bytes of dynamic memory
   //
   klogv("Initializing virtual memory...");

   init_paging();

   // 6) Call YOUR command handler -  interface method
   klogv("Transferring control to commhand...");
   commhand();

   // 7) System Shutdown on return from your command handler

   klogv("Starting system shutdown procedure...");

   /* Shutdown Procedure */
   klogv("Shutdown complete. You may now turn off the machine. (QEMU: C-a x)");
   hlt();
}
```

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.

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

5.12.2.2 polling()

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

Definition at line 92 of file serial.c.

```
{
  // insert your code to gather keyboard input via the technique of polling.

  char keyboard_character;

  int cursor = 0;

  char log[] = {'\0', '\0', '\0', '\0'};

  int characters_in_buffer = 0;

  while (1)
  {

    if (inb(COM1 + 5) & 1)
    {                                 // is there input char?
      keyboard_character = inb(COM1); //read the char from COM1

      if (keyboard_character == '\n' || keyboard_character == '\r')
      { // HANDLEING THE CARRIAGE RETURN AND NEW LINE CHARACTERS

        buffer[characters_in_buffer] = '\0';
        break;
      }
      else if ((keyboard_character == 127 || keyboard_character == 8) && cursor > 0)
      { // HANDELING THE BACKSPACE CHARACTER

        //serial_println("Handleing backspace character.");
        serial_print("\033[K");

        buffer[cursor - 1] = '\0';
        serial_print("\b \b");
        serial_print(buffer + cursor);
        cursor--;

        int temp_cursor = cursor;

        while (buffer[temp_cursor + 1] != '\0')
        {
          buffer[temp_cursor] = buffer[temp_cursor + 1];
          buffer[temp_cursor + 1] = '\0';
          temp_cursor++;
        }

        characters_in_buffer--;
        cursor = characters_in_buffer;
      }
      else if (keyboard_character == '~' && cursor < 99)
      { //HANDLING THE DELETE KEY
        // \033[3~

        serial_print("\033[K");

        buffer[cursor + 1] = '\0';
        serial_print("\b \b");
        serial_print(buffer + cursor);

        int temp_cursor = cursor + 1;

        while (buffer[temp_cursor + 1] != '\0')
        {
          buffer[temp_cursor] = buffer[temp_cursor + 1];
          buffer[temp_cursor + 1] = '\0';
          temp_cursor++;
        }

        characters_in_buffer--;
        cursor = characters_in_buffer;
      }
      else if (keyboard_character == '\033')
      { // HANDLEING FIRST CHARACTER FOR ARROW KEYS

        log[0] = keyboard_character;
      }
      else if (keyboard_character == '[' && log[0] == '\033')
      { // HANDLEING SECOND CHARACTER FOR ARROW KEYS

        log[1] = keyboard_character;
      }
      else if (log[0] == '\033' && log[1] == '[')
      { // HANDLEING LAST CHARACTER FOR ARROW KEYS
        log[2] = keyboard_character;

        if (keyboard_character == 'A')
        { //Up arrow
          //Call a history function from the commhand or do nothing
        }
        else if (keyboard_character == 'B')
        { //Down arrow
          //Call a history command from the commhand or do nothing
        }
        else if (keyboard_character == 'C' && cursor != 99)
        { //Right arrow

          serial_print("\033[C");
          cursor++;
        }
        else if (keyboard_character == 'D' && cursor != 0)
        { //Left arrow

          serial_print("\033[D");
          cursor--;
        }

        memset(log, '\0', 4);
      }
      else
      {

        if (cursor == 0 && buffer[cursor] == ' \0') //Adding character at beginning of buffer
        {
          buffer[cursor] = keyboard_character;
          serial_print(&keyboard_character);
          cursor++;
        }
        else if (buffer[cursor] == ' \0') //Adding character at the end of the buffer
        {
          buffer[cursor] = keyboard_character;
          serial_print(&keyboard_character);
          cursor++;
        }
        else //Inserting character to the middle of the buffer
        {
          char temp_buffer[strlen(buffer)];
          memset(temp_buffer, ' \0', strlen(buffer));

          int temp_cursor = 0;
          while (temp_cursor <= characters_in_buffer) //Filling the temp_buffer with all of the characters from buffer, and inserting the new character.
          {
            if (temp_cursor < cursor)
            {
              temp_buffer[temp_cursor] = buffer[temp_cursor];
            }
            else if (temp_cursor > cursor)
            {
              temp_buffer[temp_cursor] = buffer[temp_cursor - 1];
            }
            else
            { //temp_cursor == cursor
              temp_buffer[temp_cursor] = keyboard_character;
            }
            temp_cursor++;
          }

          temp_cursor = 0;
          int temp_buffer_size = strlen(temp_buffer);
          while (temp_cursor <= temp_buffer_size) //Setting the contents of the buffer equal to the temp_buffer.
          {
            buffer[temp_cursor] = temp_buffer[temp_cursor];
            temp_cursor++;
          }

          serial_print("\033[K");
          serial_print(&keyboard_character);
          serial_print(buffer + cursor + 1);
          cursor++;
        }
        characters_in_buffer++;
      }
    }
  }

```

```
  *count = characters_in_buffer; // buffer count

  return count;
}
```

5.12.2.3 serial_print()

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

Definition at line 56 of file serial.c.

```
{
  int i;
  for (i = 0; *(i + msg) != '\0'; i++)
  {
    outb(serial_port_out, *(i + msg));
  }
  if (*msg == '\r')
    outb(serial_port_out, '\n');
  return NO_ERROR;
}
```

5.12.2.4 serial_println()

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

Definition at line 40 of file serial.c.

```
{
  int i;
  for (i = 0; *(i + msg) != '\0'; i++)
  {
    outb(serial_port_out, *(i + msg));
  }
  outb(serial_port_out, '\r');
  outb(serial_port_out, '\n');
  return NO_ERROR;
}
```

5.12.2.5 set_serial_in()

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

Definition at line 86 of file serial.c.

```
{
  serial_port_in = device;
  return NO_ERROR;
}
```

5.12.2.6 set_serial_out()

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

Definition at line 74 of file serial.c.

```
{
   serial_port_out = device;
   return NO_ERROR;
}
```

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.

```
{
  char logmsg[64] = {'\0'}, prefix[] = "klogv: ";
  strcat(logmsg, prefix);
  strcat(logmsg, msg);
  serial_println(logmsg);
}
```

5.13.1.2 kpanic()

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

Definition at line 24 of file system.c.

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

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.

```
{
  gdt_entry *new_entry = &gdt_entries[idx];
  new_entry->base_low  = (base & 0xFFFF);
  new_entry->base_mid  = (base » 16) & 0xFF;
  new_entry->base_high = (base » 24) & 0xFF;
  new_entry->limit_low = (limit & 0xFFFF);
  new_entry->flags   = (limit » 16) & 0xFF;
  new_entry->flags |= flags & 0xF0;
  new_entry->access = access;
}
```

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.

```
{
  idt_entry *new_entry = &idt_entries[idx];
  new_entry->base_low  = (base &  0xFFFF);
  new_entry->base_high = (base » 16) & 0xFFFF;
  new_entry->sselect   = sel;
  new_entry->zero = 0;
  new_entry->flags = flags;
}
```

5.14.1.3 init_gdt()

```
void init_gdt ( )
```

Definition at line 75 of file tables.c.

```
{
  gdt_ptr.limit = 5 * sizeof(gdt_entry) - 1;
  gdt_ptr.base  = (u32int) gdt_entries;

  u32int limit = 0xFFFFFFFF;
  gdt_init_entry(0, 0, 0, 0, 0);           //required null segment
  gdt_init_entry(1, 0, limit, 0x9A, 0xCF); //code segment
  gdt_init_entry(2, 0, limit, 0x92, 0xCF); //data segment
  gdt_init_entry(3, 0, limit, 0xFA, 0xCF); //user mode code segment
  gdt_init_entry(4, 0, limit, 0xF2, 0xCF); //user mode data segment

  write_gdt_ptr((u32int) &gdt_ptr, sizeof(gdt_ptr));
}
```

5.14.1.4 init_idt()

```
void init_idt ( )
```

Definition at line 43 of file tables.c.

```
{
  idt_ptr.limit = 256*sizeof(idt_descriptor) - 1;
  idt_ptr.base  = (u32int)idt_entries;
  memset(idt_entries, 0, 256*sizeof(idt_descriptor));

  write_idt_ptr((u32int)&idt_ptr);
}
```

5.14.1.5 write_gdt_ptr()

```
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.

```
{
  u32int *addr;

  // Allocate on the kernel heap if one has been created
  if (kheap != 0){
    addr = (u32int*)alloc(size, kheap, page_align);
    if (phys_addr){
      page_entry *page = get_page((u32int)addr, kdir, 0);
      *phys_addr = (page->frameaddr*0x1000) + ((u32int)addr & 0xFFF);
    }
    return (u32int)addr;
  }
  // Else, allocate directly from physical memory
  else {
    if (page_align && (phys_alloc_addr & 0xFFFFF000)){
      phys_alloc_addr &= 0xFFFFF000;
      phys_alloc_addr += 0x1000;
    }
    addr = (u32int*)phys_alloc_addr;
    if (phys_addr){
      *phys_addr = phys_alloc_addr;
    }
    phys_alloc_addr += size;
    return (u32int)addr;
  }
}
```

5.15.1.2 alloc()

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

Definition at line 57 of file heap.c.

```
{
  no_warn(size||align||h);
  static u32int heap_addr = KHEAP_BASE;

  u32int base = heap_addr;
  heap_addr += size;

  if (heap_addr > KHEAP_BASE + KHEAP_MIN)
    serial_println("Heap is full!");

  return base;
}
```

5.15.1.3 kmalloc()

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

Definition at line 52 of file heap.c.

```
{
  return _kmalloc(size,0,0);
}
```

5.15.1.4 make_heap()

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

Definition at line 71 of file heap.c.

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

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.

```
{
  u32int frame  = addr/page_size;
  u32int index  = frame/32;
  u32int offset = frame%32;
  frames[index] &= ~(1 « offset);
}
```

5.16.1.2 find_free()

```
u32int find_free ( )
```

Definition at line 68 of file paging.c.

```
{
  u32int i,j;
  for (i=0; i<nframes/32; i++)
    if (frames[i] != 0xFFFFFFFF) //if frame not full
      for (j=0; j<32; j++) //find first free bit
    if (!(frames[i] & (1 « j)))
      return i*32+j;

  return -1; //no free frames
}
```

5.16.1.3 get_bit()

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

Definition at line 56 of file paging.c.

```
{
  u32int frame  = addr/page_size;
  u32int index  = frame/32;
  u32int offset = frame%32;
  return (frames[index] & (1 « offset));
}
```

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.

```
{
  u32int phys_addr;
  u32int index = addr / page_size / 1024;
  u32int offset = addr / page_size % 1024;

  //return it if it exists
  if (dir->tables[index])
    return &dir->tables[index]->pages[offset];

  //create it
  else if (make_table){
    dir->tables[index] = (page_table*)_kmalloc(sizeof(page_table), 1, &phys_addr);
    dir->tables_phys[index] = phys_addr | 0x7; //enable present, writable
    return &dir->tables[index]->pages[offset];
  }
  else return 0;
}
```

5.16.1.5 init_paging()

```
void init_paging ( )
```

Definition at line 111 of file paging.c.

```
{
  //create frame bitmap
  nframes = (u32int)(mem_size/page_size);
  frames = (u32int*)kmalloc(nframes/32);
  memset(frames, 0, nframes/32);

  //create kernel directory
  kdir = (page_dir*)_kmalloc(sizeof(page_dir), 1, 0); //page aligned
  memset(kdir, 0, sizeof(page_dir));

  //get pages for kernel heap
  u32int i = 0x0;
  for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN); i+=1){
    get_page(i,kdir,1);
  }

  //perform identity mapping of used memory
  //note: placement_addr gets incremented in get_page,
  //so we're mapping the first frames as well
  i = 0x0;
  while (i < (phys_alloc_addr+0x10000)){
    new_frame(get_page(i,kdir,1));
    i += page_size;
  }

  //allocate heap frames now that the placement addr has increased.
  //placement addr increases here for heap
  for(i=KHEAP_BASE; i<(KHEAP_BASE+KHEAP_MIN);i+=PAGE_SIZE){
    new_frame(get_page(i,kdir,1));
  }

  //load the kernel page directory; enable paging
  load_page_dir(kdir);

  //setup the kernel heap
  kheap = make_heap(KHEAP_BASE, KHEAP_SIZE, KHEAP_BASE+KHEAP_MIN);
}
```

5.16.1.6 load_page_dir()

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

Definition at line 158 of file paging.c.

```
{
  cdir = new_dir;
  asm volatile ("mov %0,%%cr3":: "b"(&cdir->tables_phys[0]));
  u32int cr0;
  asm volatile ("mov %%cr0,%0": "=b"(cr0));
  cr0 |= 0x80000000;
  asm volatile ("mov %0,%%cr0":: "b"(cr0));
}
```

5.16.1.7 new_frame()

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

Definition at line 173 of file paging.c.

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

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

5.16.1.8 set_bit()

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

Definition at line 32 of file paging.c.

```
{
  u32int frame  = addr/page_size;
  u32int index  = frame/32;
  u32int offset = frame%32;
  frames[index] |= (1 « offset);
}
```

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.

```
{
  int res=0;
   int charVal=0;
   char sign = ' ';
   char c = *s;


   while(isspace(&c)){ ++s; c = *s;} // advance past whitespace


   if (*s == '-' || *s == '+') sign = *(s++); // save the sign


   while(*s != '\0'){
        charVal = *s - 48;
    res = res * 10 + charVal;
    s++;

   }


   if ( sign == '-') res=res * -1;

  return res; // return integer
}
```

5.17.1.2 isspace()

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

Definition at line 119 of file string.c.

```
{
  if (*c == ' '  ||
      *c == '\n' ||
      *c == '\r' ||
      *c == '\f' ||
      *c == '\t' ||
      *c == '\v'){
    return 1;
  }
  return 0;
}
```

5.17.1.3 memset()

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

Definition at line 137 of file string.c.

```
{
  unsigned char *p = (unsigned char *) s;
  while(n--){
    *p++ = (unsigned char) c;
  }
  return s;
}
```

5.17.1.4 strcat()

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

Definition at line 106 of file string.c.

```
{
  char *rc = s1;
  if (*s1) while(*++s1);
  while( (*s1++ = *s2++) );
  return rc;
}
```

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.

```
{
  static char *tok_tmp = NULL;
  const char *p = s2;

  //new string
  if (s1!=NULL){
    tok_tmp = s1;
  }
  //old string cont'd
  else {
    if (tok_tmp==NULL){
      return NULL;
    }
    s1 = tok_tmp;
  }

  //skip leading s2 characters
  while ( *p && *s1 ){
    if (*s1==*p){
      ++s1;
      p = s2;
      continue;
    }
    ++p;
  }

  //no more to parse
  if (!*s1){
    return (tok_tmp = NULL);
  }

  //skip non-s2 characters
  tok_tmp = s1;
  while (*tok_tmp){
    p = s2;
    while (*p){
      if (*tok_tmp==*p++){
    *tok_tmp++ = '\0';
    return s1;
      }
    }
    ++tok_tmp;
  }

  //end of string
  tok_tmp = NULL;
  return s1;
}
```

5.18 modules/mpx_supt.c File Reference

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

Functions

- int sys_req (int op_code, int device_id, char *buffer_ptr, int *count_ptr)
- void mpx_init (int cur_mod)
- void sys_set_malloc (u32int(*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.

```
{
  char msg[30];
  int count=0;

    memset( msg, ’\0’, sizeof(msg));
    strcpy(msg, "IDLE PROCESS EXECUTING.\n");
    count = strlen(msg);

  while(1){
    sys_req( WRITE, DEFAULT_DEVICE, msg, &count);
    sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
  }
}
```

5.18.1.2 mpx_init()

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

Definition at line 106 of file mpx_supt.c.

```
{

  current_module = cur_mod;
  if (cur_mod == MEM_MODULE)
        mem_module_active = TRUE;

  if (cur_mod == IO_MODULE)
        io_module_active = TRUE;
}
```

5.18.1.3 sys_alloc_mem()

```
void∗ sys_alloc_mem (
              u32int size )
```

Definition at line 144 of file mpx_supt.c.

```
{
  if (!mem_module_active)
    return (void *) kmalloc(size);
  else
    return (void *) (*student_malloc)(size);
}
```

5.18.1.4 sys_free_mem()

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

Definition at line 158 of file mpx_supt.c.

```
{
  if (mem_module_active)
    return (*student_free)(ptr);
  // otherwise we don't free anything
  return -1;
}
```

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.

```
{
    int return_code =0;

  if (op_code == IDLE || op_code == EXIT){
    // store the process's operation request
    // triger interrupt 60h to invoke
    params.op_code = op_code;
    asm volatile ("int $60");
  }// idle or exit

  else if (op_code == READ || op_code == WRITE) {
    // validate buffer pointer and count pointer
    if (buffer_ptr == NULL)
      return_code = INVALID_BUFFER;
    else if (count_ptr == NULL || *count_ptr <= 0)
      return_code = INVALID_COUNT;

    // if parameters are valid store in the params structure
    if ( return_code == 0){
      params.op_code = op_code;
      params.device_id = device_id;
      params.buffer_ptr = buffer_ptr;
      params.count_ptr = count_ptr;

      if (!io_module_active){
        // if default device
        if (op_code == READ)
          return_code = *(polling(buffer_ptr, count_ptr));

        else //must be WRITE
          return_code = serial_print(buffer_ptr);

      } else {// I/O module is implemented
        asm volatile ("int $60");
      } // NOT IO_MODULE
    }
  } else return_code = INVALID_OPERATION;

  return return_code;
}// 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.

```
{
  student_free = func;
}
```

5.18.1.7 sys_set_malloc()

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

Definition at line 124 of file mpx_supt.c.

```
{
  student_malloc = func;
}
```

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.

```
{
  char msg[30];
  int count=0;

    memset( msg, '\0', sizeof(msg));
    strcpy(msg, "IDLE PROCESS EXECUTING.\n");
    count = strlen(msg);

  while(1){
    sys_req( WRITE, DEFAULT_DEVICE, msg, &count);
    sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
  }
}
```

5.19.2.2 mpx_init()

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

Definition at line 106 of file mpx_supt.c.

```
{

  current_module = cur_mod;
  if (cur_mod == MEM_MODULE)
        mem_module_active = TRUE;

  if (cur_mod == IO_MODULE)
        io_module_active = TRUE;
}
```

5.19.2.3 sys_alloc_mem()

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

Definition at line 144 of file mpx_supt.c.

```
{
  if (!mem_module_active)
    return (void *) kmalloc(size);
  else
    return (void *) (*student_malloc)(size);
}
```

5.19.2.4 sys_free_mem()

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

Definition at line 158 of file mpx_supt.c.

```
{
  if (mem_module_active)
    return (*student_free)(ptr);
  // otherwise we don't free anything
  return -1;
}
```

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.

```
{
    int return_code =0;

  if (op_code == IDLE || op_code == EXIT){
    // store the process's operation request
    // triger interrupt 60h to invoke
    params.op_code = op_code;
    asm volatile ("int $60");
  }// idle or exit

  else if (op_code == READ || op_code == WRITE) {
    // validate buffer pointer and count pointer
    if (buffer_ptr == NULL)
      return_code = INVALID_BUFFER;
    else if (count_ptr == NULL || *count_ptr <= 0)
      return_code = INVALID_COUNT;

    // if parameters are valid store in the params structure
    if ( return_code == 0){
      params.op_code = op_code;
      params.device_id = device_id;
      params.buffer_ptr = buffer_ptr;
      params.count_ptr = count_ptr;

      if (!io_module_active){
        // if default device
        if (op_code == READ)
          return_code = *(polling(buffer_ptr, count_ptr));

        else //must be WRITE
          return_code = serial_print(buffer_ptr);

      } else {// I/O module is implemented
        asm volatile ("int $60");
      } // NOT IO_MODULE
    }
  } else return_code = INVALID_OPERATION;

  return return_code;
}// 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.

```
{
  student_free = func;
}
```

5.19.2.7 sys_set_malloc()

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

Definition at line 124 of file mpx_supt.c.

```
{
  student_malloc = func;
}
```

5.20 modules/R1/commhand.c File Reference

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

Functions

- int commhand ()

5.20.1 Function Documentation

5.20.1.1 commhand()

```
int commhand ( )
```

Definition at line 10 of file commhand.c.

```
{

    char welcomeMSG[] = "\nWelcome to our CS 450 Project!\nType help to see what you can do!\n\n";
    int welcomeLength = strlen(welcomeMSG);
    sys_req(WRITE, DEFAULT_DEVICE, welcomeMSG, &welcomeLength);

    char cmdBuffer[100];
    int bufferSize;
    allocateQueues();

    int quitFlag = 0;

    while (!quitFlag)
    {
        //get a command: cal polling fx

        memset(cmdBuffer, '\0', 100);

        bufferSize = 99; // reset size before each call to read

        sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);

        char newLine[] = "\n";
        int newLineCount = 1;
        sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);

        if (strcmp(cmdBuffer, "help") == 0)
        {
            help();
        }
        else if (strcmp(cmdBuffer, "version") == 0)
        {
            version();
        }
        else if (strcmp(cmdBuffer, "getDate") == 0)
        {
            getDate();
        }
        else if (strcmp(cmdBuffer, "setDate") == 0)
        {
            setDate();
        }
        else if (strcmp(cmdBuffer, "getTime") == 0)
        {
            getTime();
        }
        else if (strcmp(cmdBuffer, "setTime") == 0)
        {
            setTime();
        }
        else if (strcmp(cmdBuffer, "createPCB") == 0)
        {
            char processName[20];
            char processClass;
            int processPriority;

            char nameMsg[] = "Please enter a name for the PCB you wish to create. (The name can be no more than 20 characters)\n";
            int nameMsgLen = strlen(nameMsg);
            sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
            sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
            sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
            strcpy(processName, cmdBuffer);
```

```
       memset(cmdBuffer, ' \0', 100);

       char classMsg[] = "Please enter a class for the PCB you wish to create. ('a' for application or 's' for system)\n";
       int classMsgLen = strlen(classMsg);
       sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
       sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
       sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
       if (strcmp(cmdBuffer, "a") == 0)
       {
           processClass = 'a';
       }
       else if (strcmp(cmdBuffer, "s") == 0)
       {
           processClass = 's';
       }
       else
       {
           processClass = '\0';
       }
       memset(cmdBuffer, ' \0', 100);

       char priorityMsg[] = "Please enter a priority for the PCB you wish to create. (The priorities range from 0 to 9)\n";
       int priorityMsgLen = strlen(priorityMsg);
       sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
       sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
       sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
       processPriority = atoi(cmdBuffer);

        createPCB(processName, processClass, processPriority);
    }
    else if (strcmp(cmdBuffer, "deletePCB") == 0)
    {
        char processName[20];

        char nameMsg[] = "Please enter the name for the PCB you wish to delete. (The name can be no more than 20 characters)\n";
        int nameMsgLen = strlen(nameMsg);
        sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
        sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
        sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
        strcpy(processName, cmdBuffer);

        deletePCB(processName);
    }
    else if (strcmp(cmdBuffer, "blockPCB") == 0)
    {
        char processName[20];

        char nameMsg[] = "Please enter the name for the PCB you wish to block. (The name can be no more than 20 characters)\n";
        int nameMsgLen = strlen(nameMsg);
        sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
        sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
        sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
        strcpy(processName, cmdBuffer);

        blockPCB(processName);
    }
    else if (strcmp(cmdBuffer, "unblockPCB") == 0)
    {
        char processName[20];

        char nameMsg[] = "Please enter the name for the PCB you wish to unblock. (The name can be no more than 20 characters)\n";
        int nameMsgLen = strlen(nameMsg);
        sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
        sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
        sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
        strcpy(processName, cmdBuffer);

        unblockPCB(processName);
    }
    else if (strcmp(cmdBuffer, "suspendPCB") == 0)
    {
        char processName[20];

        char nameMsg[] = "Please enter the name for the PCB you wish to suspend. (The name can be no more than 20 characters)\n";
        int nameMsgLen = strlen(nameMsg);
        sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
        sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
        sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
        strcpy(processName, cmdBuffer);

        suspendPCB(processName);
    }
    else if (strcmp(cmdBuffer, "resumePCB") == 0)
    {
        char processName[20];

        char nameMsg[] = "Please enter the name for the PCB you wish to resume. (The name can be no more than 20 characters)\n";
        int nameMsgLen = strlen(nameMsg);
        sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
        sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
        sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
        strcpy(processName, cmdBuffer);

        resumePCB(processName);
    }
    else if (strcmp(cmdBuffer, "setPCBPriority") == 0)
    {
        char processName[20];
        int newProcessPriority;

        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";
        int nameMsgLen = strlen(nameMsg);
        sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
        sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
        sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
        strcpy(processName, cmdBuffer);

        char priorityMsg[] = "Please enter a priority for the PCB you wish to change priorities for. (The priorities range from 0 to 9)\n";
        int priorityMsgLen = strlen(priorityMsg);
        sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
        sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
        sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
        newProcessPriority = atoi(cmdBuffer);

        setPCBPriority(processName, newProcessPriority);
    }
    else if (strcmp(cmdBuffer, "showPCB") == 0)
    {
        char processName[20];

        char nameMsg[] = "Please enter the name for the PCB you wish to see. (The name can be no more than 20 characters)\n";
        int nameMsgLen = strlen(nameMsg);
        sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
        sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
        sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
        strcpy(processName, cmdBuffer);

        showPCB(processName);
    }
    else if (strcmp(cmdBuffer, "showReady") == 0)
    {
        showReady();
    }
    else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
    {
        showSuspendedReady();
    }
    else if (strcmp(cmdBuffer, "showSuspendedBlocked") == 0)
    {
        showSuspendedBlocked();
    }
    else if (strcmp(cmdBuffer, "showBlocked") == 0)
    {
        showBlocked();
    }
    else if (strcmp(cmdBuffer, "showAll") == 0)
    {
        showAll();
    }
    else if (strcmp(cmdBuffer, "quit") == 0)
    {
        quitFlag = quit();

        sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
    }
    else
    {
        char message[] = "Unrecognized Command\n";

        int tempBuffer = strlen(message);

        sys_req(WRITE, DEFAULT_DEVICE, (char *)message, &tempBuffer);
    }

    // process the command: take array buffer chars and make a string. Decide what the cmd wants to do
    // see if quit was entered: if string == quit = 1
}

return 0;
}
```

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.

```
{

    char welcomeMSG[] = "\nWelcome to our CS 450 Project!\nType help to see what you can do!\n\n";
    int welcomeLength = strlen(welcomeMSG);
    sys_req(WRITE, DEFAULT_DEVICE, welcomeMSG, &welcomeLength);

    char cmdBuffer[100];
    int bufferSize;
    allocateQueues();

    int quitFlag = 0;

    while (!quitFlag)
    {
        //get a command: cal polling fx

        memset(cmdBuffer, '\0', 100);

        bufferSize = 99; // reset size before each call to read

        sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);

        char newLine[] = "\n";
        int newLineCount = 1;
        sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);

        if (strcmp(cmdBuffer, "help") == 0)
        {
            help();
        }
        else if (strcmp(cmdBuffer, "version") == 0)
        {
            version();
        }
        else if (strcmp(cmdBuffer, "getDate") == 0)
        {
            getDate();
        }
        else if (strcmp(cmdBuffer, "setDate") == 0)
        {
            setDate();
        }
        else if (strcmp(cmdBuffer, "getTime") == 0)
        {
            getTime();
        }
        else if (strcmp(cmdBuffer, "setTime") == 0)
        {
            setTime();
        }
        else if (strcmp(cmdBuffer, "createPCB") == 0)
        {
            char processName[20];
            char processClass;
            int processPriority;

            char nameMsg[] = "Please enter a name for the PCB you wish to create. (The name can be no more than 20 characters)\n";
            int nameMsgLen = strlen(nameMsg);
            sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
            sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
            sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
            strcpy(processName, cmdBuffer);
            memset(cmdBuffer, ’ \0’, 100);

            char classMsg[] = "Please enter a class for the PCB you wish to create. (’a’ for application or ’s’ for system)\n";
            int classMsgLen = strlen(classMsg);
            sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);
            sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
            sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
            if (strcmp(cmdBuffer, "a") == 0)
            {
                processClass = ’a’;
            }
            else if (strcmp(cmdBuffer, "s") == 0)
            {
                processClass = ’s’;
            }
            else
            {
                processClass = ’\0’;
            }
            memset(cmdBuffer, ’ \0’, 100);

            char priorityMsg[] = "Please enter a priority for the PCB you wish to create. (The priorities range from 0 to 9)\n";
            int priorityMsgLen = strlen(priorityMsg);
            sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
            sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
            sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
            processPriority = atoi(cmdBuffer);

             createPCB(processName, processClass, processPriority);
         }
         else if (strcmp(cmdBuffer, "deletePCB") == 0)
         {
             char processName[20];

             char nameMsg[] = "Please enter the name for the PCB you wish to delete. (The name can be no more than 20 characters)\n";
             int nameMsgLen = strlen(nameMsg);
             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
             strcpy(processName, cmdBuffer);

             deletePCB(processName);
         }
         else if (strcmp(cmdBuffer, "blockPCB") == 0)
         {
             char processName[20];

             char nameMsg[] = "Please enter the name for the PCB you wish to block. (The name can be no more than 20 characters)\n";
             int nameMsgLen = strlen(nameMsg);
             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
             strcpy(processName, cmdBuffer);

             blockPCB(processName);
         }
         else if (strcmp(cmdBuffer, "unblockPCB") == 0)
         {
             char processName[20];

             char nameMsg[] = "Please enter the name for the PCB you wish to unblock. (The name can be no more than 20 characters)\n";
             int nameMsgLen = strlen(nameMsg);
             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
             strcpy(processName, cmdBuffer);

             unblockPCB(processName);
         }
         else if (strcmp(cmdBuffer, "suspendPCB") == 0)
         {
             char processName[20];

             char nameMsg[] = "Please enter the name for the PCB you wish to suspend. (The name can be no more than 20 characters)\n";
             int nameMsgLen = strlen(nameMsg);
             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
             strcpy(processName, cmdBuffer);

             suspendPCB(processName);
         }
         else if (strcmp(cmdBuffer, "resumePCB") == 0)
         {
             char processName[20];

             char nameMsg[] = "Please enter the name for the PCB you wish to resume. (The name can be no more than 20 characters)\n";
             int nameMsgLen = strlen(nameMsg);
             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
             strcpy(processName, cmdBuffer);

             resumePCB(processName);
         }
         else if (strcmp(cmdBuffer, "setPCBPriority") == 0)
         {
             char processName[20];
             int newProcessPriority;

             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";
             int nameMsgLen = strlen(nameMsg);
             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
             strcpy(processName, cmdBuffer);

             char priorityMsg[] = "Please enter a priority for the PCB you wish to change priorities for. (The priorities range from 0 to 9)\n";
             int priorityMsgLen = strlen(priorityMsg);
             sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
             newProcessPriority = atoi(cmdBuffer);

             setPCBPriority(processName, newProcessPriority);
         }
         else if (strcmp(cmdBuffer, "showPCB") == 0)
         {
             char processName[20];

             char nameMsg[] = "Please enter the name for the PCB you wish to see. (The name can be no more than 20 characters)\n";
             int nameMsgLen = strlen(nameMsg);
             sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
             sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
             strcpy(processName, cmdBuffer);

             showPCB(processName);
         }
         else if (strcmp(cmdBuffer, "showReady") == 0)
         {
             showReady();
         }
         else if (strcmp(cmdBuffer, "showSuspendedReady") == 0)
         {
             showSuspendedReady();
         }
         else if (strcmp(cmdBuffer, "showSuspendedBlocked") == 0)
         {
             showSuspendedBlocked();
         }
         else if (strcmp(cmdBuffer, "showBlocked") == 0)
         {
             showBlocked();
         }
         else if (strcmp(cmdBuffer, "showAll") == 0)
         {
             showAll();
         }
         else if (strcmp(cmdBuffer, "quit") == 0)
         {
             quitFlag = quit();

             sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
         }
         else
         {
             char message[] = "Unrecognized Command\n";

             int tempBuffer = strlen(message);

             sys_req(WRITE, DEFAULT_DEVICE, (char *)message, &tempBuffer);
         }

         // process the command: take array buffer chars and make a string. Decide what the cmd wants to do
         // see if quit was entered: if string == quit = 1
     }

     return 0;
 }
```

5.22 modules/R1/R1commands.c File Reference

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

Functions

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

5.22.1 Function Documentation

5.22.1.1 BCDtoChar()

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

Definition at line 593 of file R1commands.c.

```
{

    int val1 = (test / 16);
    int val2 = (test % 16);

    buffer[0] = val1 + '0';
    buffer[1] = val2 + '0';

    return 0;
}
```

5.22.1.2 getDate()

```
void getDate ( )
```

Definition at line 343 of file R1commands.c.

```
{

    char buffer[4] = " \0\0\0";
    int count = 4;
    char divider = '/';
    char newLine[1] = "\n";
    int newLineCount = 1;

    outb(0x70, 0x07); // getting Day of month value
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = divider;
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, ' \0', count);

    outb(0x70, 0x08); // getting Month value
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = divider;
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, ' \0', count);

    outb(0x70, 0x32); // getting Year value second byte
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = ' \0';
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, ' \0', count);

    outb(0x70, 0x09); // getting Year value first byte
    BCDtoChar(inb(0x71), buffer);
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, ' \0', count);

    sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
    memset(newLine, '\0', newLineCount);
}
```

5.22.1.3 getTime()

```
void getTime ( )
```

Definition at line 190 of file R1commands.c.

```
{

    char buffer[4] = " \0\0\0";
    int count = 4;
    char divider = ':';
    char newLine[1] = "\n";
    int newLineCount = 1;

    outb(0x70, 0x04); // getting Hour value
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = divider;
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, ' \0', count);

    outb(0x70, 0x02); // getting Minute value
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = divider;
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, ' \0', count);

    outb(0x70, 0x00); // getting Second value
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = ' \0';
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, ' \0', count);

    sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
    memset(newLine, '\0', newLineCount);
}
```

5.22.1.4 help()

```
int help ( )
```

Definition at line 11 of file R1commands.c.

```
{

    // Help Description section
    char helpDesc[] = "help: Returns basic command information.\n";

    int tempBuffer = strlen(helpDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)helpDesc, &tempBuffer);
    memset(helpDesc, ’\0’, tempBuffer);

    // Version Description section
    char versionDesc[] = "version: Returns the current version of the software.\n";

    tempBuffer = strlen(versionDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)versionDesc, &tempBuffer);
    memset(versionDesc, ’\0’, tempBuffer);

    // getTime Description section
    char getTimeDesc[] = "getTime: Returns the current set time.\n";

    tempBuffer = strlen(getTimeDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)getTimeDesc, &tempBuffer);
    memset(getTimeDesc, ’\0’, tempBuffer);

    // setTime Description section
    char setTimeDesc[] = "setTime: Allows the user to change the set time.\n";

    tempBuffer = strlen(setTimeDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)setTimeDesc, &tempBuffer);
    memset(setTimeDesc, ’\0’, tempBuffer);

    // getDate Description section
    char getDateDesc[] = "getDate: Returns the current set date.\n";

    tempBuffer = strlen(getDateDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)getDateDesc, &tempBuffer);
    memset(getDateDesc, ’\0’, tempBuffer);

    // setDate Description section
    char setDateDesc[] = "setDate: Allows the user to change the set date.\n";

    tempBuffer = strlen(setDateDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)setDateDesc, &tempBuffer);
    memset(setDateDesc, ’\0’, tempBuffer);

    // createPCb Description section
    char createPCBDesc[] = "createPCB: Will create a PCB and put it into the ready queue by default.\n";

    tempBuffer = strlen(createPCBDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)createPCBDesc, &tempBuffer);
    memset(createPCBDesc, ’\0’, tempBuffer);

    // deletePCB Description section
    char deletePCBDesc[] = "deletePCB: Will delete a specific PCB from what ever queue it is in. \n";

    tempBuffer = strlen(deletePCBDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)deletePCBDesc, &tempBuffer);
    memset(deletePCBDesc, ’\0’, tempBuffer);

    // blockPCB Description section
    char blockPCBDesc[] = "blockPCB: Will change a specific PCB’s state to blocked. \n";

    tempBuffer = strlen(blockPCBDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)blockPCBDesc, &tempBuffer);
    memset(blockPCBDesc, ’\0’, tempBuffer);

    // unblockPCB Description section
    char unblockPCBDesc[] = "unblockPCB: Will change a specific PCB’s state to ready. \n";

    tempBuffer = strlen(unblockPCBDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)unblockPCBDesc, &tempBuffer);
```

```
memset(unblockPCBDesc, '\0', tempBuffer);

// suspendPCB Description section
char suspendPCBDesc[] = "suspendPCB: Will suspend a specific PCB. \n";

tempBuffer = strlen(suspendPCBDesc);

sys_req(WRITE, DEFAULT_DEVICE, (char *)suspendPCBDesc, &tempBuffer);
 memset(suspendPCBDesc, '\0', tempBuffer);

 // resumePCB Description section
 char resumePCBDesc[] = "resumePCB: Will unsuspend a specific PCB. \n";

 tempBuffer = strlen(resumePCBDesc);

 sys_req(WRITE, DEFAULT_DEVICE, (char *)resumePCBDesc, &tempBuffer);
 memset(resumePCBDesc, '\0', tempBuffer);

 // setPCBPriority Description section
 char setPCBPriorityDesc[] = "setPCBPriority: Will change the priority of a specific PCB. \n";

 tempBuffer = strlen(setPCBPriorityDesc);

 sys_req(WRITE, DEFAULT_DEVICE, (char *)setPCBPriorityDesc, &tempBuffer);
 memset(setPCBPriorityDesc, '\0', tempBuffer);

 // showPCB Description section
 char showPCBDesc[] = "showPCB: Will display the name, class, state, suspended status, and priority of a specific PCB. \n";

 tempBuffer = strlen(showPCBDesc);

 sys_req(WRITE, DEFAULT_DEVICE, (char *)showPCBDesc, &tempBuffer);
 memset(showPCBDesc, '\0', tempBuffer);

 // showReady Description section
 char showReadyDesc[] = "showReady: Will display the name, class, state, suspended status, and priority of every PCB in the ready queue.\n";

 tempBuffer = strlen(showReadyDesc);

 sys_req(WRITE, DEFAULT_DEVICE, (char *)showReadyDesc, &tempBuffer);
 memset(showReadyDesc, '\0', tempBuffer);

 // showSuspendedReady Description section
 char showSuspendedReadyDesc[] = "showSuspendedReady: Will display the name, class, state, suspended status, and priority of every PCB in the suspended ready queue.\n";

 tempBuffer = strlen(showSuspendedReadyDesc);

 sys_req(WRITE, DEFAULT_DEVICE, (char *)showSuspendedReadyDesc, &tempBuffer);
 memset(showSuspendedReadyDesc, '\0', tempBuffer);

 // showSuspendedBlocked Description section
 char showSuspendedBlockedDesc[] = "showSuspendedBlocked: Will display the name, class, state, suspended status, and priority of every PCB in the suspended blocked queue.\n";

 tempBuffer = strlen(showSuspendedBlockedDesc);

 sys_req(WRITE, DEFAULT_DEVICE, (char *)showSuspendedBlockedDesc, &tempBuffer);
 memset(showSuspendedBlockedDesc, '\0', tempBuffer);

 // showBlocked Description section
 char showBlockedDesc[] = "showBlocked: Will display the name, class, state, suspended status, and priority of every PCB in the blocked queue.\n";

 tempBuffer = strlen(showBlockedDesc);

 sys_req(WRITE, DEFAULT_DEVICE, (char *)showBlockedDesc, &tempBuffer);
 memset(showBlockedDesc, '\0', tempBuffer);

 // showAll Description section
 char showAllDesc[] = "showReady: Will display the name, class, state, suspended status, and priority of every PCB in all 4 queues.\n";

 tempBuffer = strlen(showAllDesc);

 sys_req(WRITE, DEFAULT_DEVICE, (char *)showAllDesc, &tempBuffer);
 memset(showAllDesc, '\0', tempBuffer);

 // quit Description section
 char quitDesc[] = "quit: Allows the user to shut the system down.\n";

 tempBuffer = strlen(quitDesc);

 sys_req(WRITE, DEFAULT_DEVICE, (char *)quitDesc, &tempBuffer);
 memset(quitDesc, '\0', tempBuffer);
```

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

5.22.1.5 intToBCD()

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

Definition at line 587 of file R1commands.c.

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

5.22.1.6 quit()

```
int quit ( )
```

Definition at line 605 of file R1commands.c.

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

5.22.1.7 setDate()

```
int setDate ( )
```

Definition at line 379 of file R1commands.c.

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

```

    char instruction1[] = "Please type the desired year. I.E.: yyyy.\n";
    int length = strlen(instruction1);

    sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
    memset(instruction1, ’\0’, length);

    char year[5] = "\0\0\0\0\0"; // year buffer

    int flag = 0; // thrown if input is invalid

    do
    {
        sys_req(READ, DEFAULT_DEVICE, year, &count);
        if (atoi(year) > 0)
        {

            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            flag = 0;

            char yearUpper[3] = "\0\0\0";
            char yearLower[3] = "\0\0\0";

            yearUpper[0] = year[0];
            yearUpper[1] = year[1];
            yearLower[0] = year[2];
            yearLower[1] = year[3];

            cli();

            outb(0x70, 0x32); // Setting first byte year value
            outb(0x71, intToBCD(atoi(yearUpper)));

            outb(0x70, 0x09); // Setting second byte year value
            outb(0x71, intToBCD(atoi(yearLower)));

            sti();
        }
        else
        {
            char invalid[] = "Invalid year.\n";
            int lengthInval = strlen(invalid);
            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
            memset(invalid, ’\0’, lengthInval);
            flag = 1;
        }
    } while (flag == 1);

    char instruction2[] = "Please type the desired month. I.E.: mm.\n";
    length = strlen(instruction2);

    sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
    memset(instruction2, ’\0’, length);

    char month[4] = "\0\0\n\0";
    count = 4; // used to print month

    do
    {
        sys_req(READ, DEFAULT_DEVICE, month, &count);
        if (atoi(month) < 13 && atoi(month) > 0)
        {

            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            flag = 0;

            cli();

            outb(0x70, 0x08); // Setting month value
            outb(0x71, intToBCD(atoi(month)));

            sti();
        }
        else
        {
            char invalid[] = "Invalid month.\n";
            int lengthInval = strlen(invalid);
            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
            memset(invalid, ’\0’, lengthInval);
            flag = 1;
        }
    } while (flag == 1);

    char instruction3[] = "Please type the desired day of month. I.E.: dd.\n";

    length = strlen(instruction3);
    sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
    memset(instruction3, '\0', length);

    char day[4] = "\0\0\n\0";
    count = 4; // used to print day

    do
    {
        sys_req(READ, DEFAULT_DEVICE, day, &count);
        sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
        if ((atoi(year) % 4 == 0 && atoi(year) % 100 != 0) || atoi(year) % 400 == 0)
        { // checking for leap year

            char leapYear[] = "This is a leap year. February has 29 days.\n";
            length = strlen(leapYear);

            sys_req(WRITE, DEFAULT_DEVICE, leapYear, &length);
            memset(leapYear, '\0', length);

            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)
            {
                flag = 1;
                char invalid[] = "Invalid day.\n";
                length = strlen(invalid);
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
                memset(invalid, '\0', length);
            }
            else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) && atoi(day) > 30)
            {
                flag = 1;
                char invalid[] = "Invalid day.\n";
                length = strlen(invalid);
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
                memset(invalid, '\0', length);
            }
            else if ((atoi(month) == 2) && atoi(day) > 29)
            {
                flag = 1;
                char invalid[] = "Invalid day.\n";
                length = strlen(invalid);
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
                memset(invalid, '\0', length);
            }
            else
            {

                flag = 0;
                cli();

                outb(0x70, 0x07); // Setting day of month value
                outb(0x71, intToBCD(atoi(day)));

                sti();
            }
        }
        else if (atoi(year) % 4 != 0 || atoi(year) % 400 != 0)
        { // checking for leap year

            char noLeap[] = "This is not a leap year.\n";
            length = strlen(noLeap);
            sys_req(WRITE, DEFAULT_DEVICE, noLeap, &length);
            memset(noLeap, '\0', length);

            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)
            {
                flag = 1;
                char invalid[] = "Invalid day.\n";
                length = strlen(invalid);
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
                memset(invalid, '\0', length);
            }
            else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) && atoi(day) > 30)
            {
                flag = 1;
                char invalid[] = "Invalid day.\n";
                length = strlen(invalid);
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
                memset(invalid, '\0', length);
            }
            else if ((atoi(month) == 2) && atoi(day) > 28)
            {
                flag = 1;
```

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

5.22.1.8 setTime()

```
int setTime ( )
```

Definition at line 221 of file R1commands.c.

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

```

    do
    {
        sys_req(READ, DEFAULT_DEVICE, minute, &count);
        if (atoi(minute) < 60 && atoi(minute) >= 0)
        {

            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            flag = 0;
        }
        else
        {
            char invalid[] = "Invalid minutes.\n";
            int lengthInval = strlen(invalid);
            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
            memset(invalid, '\0', lengthInval);
            flag = 1;
        }
    } while (flag == 1);

    char instruction3[] = "Please type the desired seconds. I.E.: ss.\n";

    length = strlen(instruction3);

    sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
    memset(instruction3, '\0', length);

    char second[4] = "\0\0\n\0";

    do
    {
        sys_req(READ, DEFAULT_DEVICE, second, &count);
        if (atoi(second) < 60 && atoi(second) >= 0)
        {

            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            flag = 0;
        }
        else
        {
            char invalid[] = "Invalid seconds.\n";
            int lengthInval = strlen(invalid);
            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
            memset(invalid, '\0', lengthInval);
            flag = 1;
        }
    } while (flag == 1);

    cli();

    outb(0x70, 0x04); // Hour
    outb(0x71, intToBCD(atoi(hour)));

    outb(0x70, 0x02); // Minute
    outb(0x71, intToBCD(atoi(minute)));

    outb(0x70, 0x00); // Second
    outb(0x71, intToBCD(atoi(second)));

    sti();

    char exitMessage[] = "The time has been set.\n";
    int exitLength = strlen(exitMessage);
    sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
    memset(exitMessage, '\0', exitLength);
    memset(spacer, '\0', spaceCount);

    return 0;
}
```

5.22.1.9 version()

```
int version ( )
```

Definition at line 177 of file R1commands.c.

```
{

    char version[] = "Version 2.0\n";

    int tempBuffer = strlen(version);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)version, &tempBuffer);
    memset(version, '\0', tempBuffer);

    return 0;
}
```

5.23 modules/R1/R1commands.h File Reference

Functions

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

5.23.1 Function Documentation

5.23.1.1 BCDtoChar()

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

Definition at line 593 of file R1commands.c.

```
{

    int val1 = (test / 16);
    int val2 = (test % 16);

    buffer[0] = val1 + '0';
    buffer[1] = val2 + '0';

    return 0;
}
```

5.23.1.2 change_int_to_binary()

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

5.23.1.3 getDate()

```
void getDate ( )
```

Definition at line 343 of file R1commands.c.

```
{

    char buffer[4] = "\0\0\0\0";
    int count = 4;
    char divider = '/';
    char newLine[1] = "\n";
    int newLineCount = 1;

    outb(0x70, 0x07); // getting Day of month value
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = divider;
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, '\0', count);

    outb(0x70, 0x08); // getting Month value
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = divider;
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, '\0', count);

    outb(0x70, 0x32); // getting Year value second byte
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = '\0';
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, '\0', count);

    outb(0x70, 0x09); // getting Year value first byte
    BCDtoChar(inb(0x71), buffer);
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, '\0', count);

    sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
    memset(newLine, '\0', newLineCount);
}
```

5.23.1.4 getTime()

```
void getTime ( )
```

Definition at line 190 of file R1commands.c.

```
{

    char buffer[4] = "\0\0\0";
    int count = 4;
    char divider = ':';
    char newLine[1] = "\n";
    int newLineCount = 1;

    outb(0x70, 0x04); // getting Hour value
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = divider;
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, '\0', count);

    outb(0x70, 0x02); // getting Minute value
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = divider;
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, '\0', count);

    outb(0x70, 0x00); // getting Second value
    BCDtoChar(inb(0x71), buffer);
    buffer[2] = '\0';
    sys_req(WRITE, DEFAULT_DEVICE, buffer, &count);
    memset(buffer, '\0', count);

    sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineCount);
    memset(newLine, '\0', newLineCount);
}
```

5.23.1.5 help()

```
void help ( )
```

Definition at line 11 of file R1commands.c.

```
{

    // Help Description section
    char helpDesc[] = "help: Returns basic command information.\n";

    int tempBuffer = strlen(helpDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)helpDesc, &tempBuffer);
    memset(helpDesc, ’\0’, tempBuffer);

    // Version Description section
    char versionDesc[] = "version: Returns the current version of the software.\n";

    tempBuffer = strlen(versionDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)versionDesc, &tempBuffer);
    memset(versionDesc, ’\0’, tempBuffer);

    // getTime Description section
    char getTimeDesc[] = "getTime: Returns the current set time.\n";

    tempBuffer = strlen(getTimeDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)getTimeDesc, &tempBuffer);
    memset(getTimeDesc, ’\0’, tempBuffer);

    // setTime Description section
    char setTimeDesc[] = "setTime: Allows the user to change the set time.\n";

    tempBuffer = strlen(setTimeDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)setTimeDesc, &tempBuffer);
    memset(setTimeDesc, ’\0’, tempBuffer);

    // getDate Description section
    char getDateDesc[] = "getDate: Returns the current set date.\n";

    tempBuffer = strlen(getDateDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)getDateDesc, &tempBuffer);
    memset(getDateDesc, ’\0’, tempBuffer);

    // setDate Description section
    char setDateDesc[] = "setDate: Allows the user to change the set date.\n";

    tempBuffer = strlen(setDateDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)setDateDesc, &tempBuffer);
    memset(setDateDesc, ’\0’, tempBuffer);

    // createPCb Description section
    char createPCBDesc[] = "createPCB: Will create a PCB and put it into the ready queue by default.\n";

    tempBuffer = strlen(createPCBDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)createPCBDesc, &tempBuffer);
    memset(createPCBDesc, ’\0’, tempBuffer);

    // deletePCB Description section
    char deletePCBDesc[] = "deletePCB: Will delete a specific PCB from what ever queue it is in. \n";

    tempBuffer = strlen(deletePCBDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)deletePCBDesc, &tempBuffer);
    memset(deletePCBDesc, ’\0’, tempBuffer);

    // blockPCB Description section
    char blockPCBDesc[] = "blockPCB: Will change a specific PCB’s state to blocked. \n";

    tempBuffer = strlen(blockPCBDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)blockPCBDesc, &tempBuffer);
    memset(blockPCBDesc, ’\0’, tempBuffer);

    // unblockPCB Description section
    char unblockPCBDesc[] = "unblockPCB: Will change a specific PCB’s state to ready. \n";

    tempBuffer = strlen(unblockPCBDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)unblockPCBDesc, &tempBuffer);
    memset(unblockPCBDesc, '\0', tempBuffer);

    // suspendPCB Description section
    char suspendPCBDesc[] = "suspendPCB: Will suspend a specific PCB. \n";

    tempBuffer = strlen(suspendPCBDesc);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)suspendPCBDesc, &tempBuffer);
     memset(suspendPCBDesc, '\0', tempBuffer);

     // resumePCB Description section
     char resumePCBDesc[] = "resumePCB: Will unsuspend a specific PCB. \n";

     tempBuffer = strlen(resumePCBDesc);

     sys_req(WRITE, DEFAULT_DEVICE, (char *)resumePCBDesc, &tempBuffer);
     memset(resumePCBDesc, '\0', tempBuffer);

     // setPCBPriority Description section
     char setPCBPriorityDesc[] = "setPCBPriority: Will change the priority of a specific PCB. \n";

     tempBuffer = strlen(setPCBPriorityDesc);

     sys_req(WRITE, DEFAULT_DEVICE, (char *)setPCBPriorityDesc, &tempBuffer);
     memset(setPCBPriorityDesc, '\0', tempBuffer);

     // showPCB Description section
     char showPCBDesc[] = "showPCB: Will display the name, class, state, suspended status, and priority of a specific PCB. \n";

     tempBuffer = strlen(showPCBDesc);

     sys_req(WRITE, DEFAULT_DEVICE, (char *)showPCBDesc, &tempBuffer);
     memset(showPCBDesc, '\0', tempBuffer);

     // showReady Description section
     char showReadyDesc[] = "showReady: Will display the name, class, state, suspended status, and priority of every PCB in the ready queue.\n";

     tempBuffer = strlen(showReadyDesc);

     sys_req(WRITE, DEFAULT_DEVICE, (char *)showReadyDesc, &tempBuffer);
     memset(showReadyDesc, '\0', tempBuffer);

     // showSuspendedReady Description section
     char showSuspendedReadyDesc[] = "showSuspendedReady: Will display the name, class, state, suspended status, and priority of every PCB in the suspended ready queue.\n";

     tempBuffer = strlen(showSuspendedReadyDesc);

     sys_req(WRITE, DEFAULT_DEVICE, (char *)showSuspendedReadyDesc, &tempBuffer);
     memset(showSuspendedReadyDesc, '\0', tempBuffer);

     // showSuspendedBlocked Description section
     char showSuspendedBlockedDesc[] = "showSuspendedBlocked: Will display the name, class, state, suspended status, and priority of every PCB in the suspended blocked queue.\n";

     tempBuffer = strlen(showSuspendedBlockedDesc);

     sys_req(WRITE, DEFAULT_DEVICE, (char *)showSuspendedBlockedDesc, &tempBuffer);
     memset(showSuspendedBlockedDesc, '\0', tempBuffer);

     // showBlocked Description section
     char showBlockedDesc[] = "showBlocked: Will display the name, class, state, suspended status, and priority of every PCB in the blocked queue.\n";

     tempBuffer = strlen(showBlockedDesc);

     sys_req(WRITE, DEFAULT_DEVICE, (char *)showBlockedDesc, &tempBuffer);
     memset(showBlockedDesc, '\0', tempBuffer);

     // showAll Description section
     char showAllDesc[] = "showReady: Will display the name, class, state, suspended status, and priority of every PCB in all 4 queues.\n";

     tempBuffer = strlen(showAllDesc);

     sys_req(WRITE, DEFAULT_DEVICE, (char *)showAllDesc, &tempBuffer);
     memset(showAllDesc, '\0', tempBuffer);

     // quit Description section
     char quitDesc[] = "quit: Allows the user to shut the system down.\n";

     tempBuffer = strlen(quitDesc);

     sys_req(WRITE, DEFAULT_DEVICE, (char *)quitDesc, &tempBuffer);
     memset(quitDesc, '\0', tempBuffer);

     return 0;
 }
```

5.23.1.6 quit()

```
int quit ( )
```

Definition at line 605 of file R1commands.c.

```
{
    int flag = 0;

    char quitMsg[] = "Are you sure you want to shutdown? y/n\n";
    int quitMsgLength = strlen(quitMsg);
    sys_req(WRITE, DEFAULT_DEVICE, quitMsg, &quitMsgLength);

    char quitAns[] = "\0\0";
    int quitAnsLength = 1;
    sys_req(READ, DEFAULT_DEVICE, quitAns, &quitAnsLength);
    char answer = quitAns[0];

    if (answer == 'y' || answer == 'Y')
    {
        flag = 1;
    }
    else if (answer == 'n' || answer == 'N')
    {
        flag = 0;
    }
    else
    {
        char error[] = "Invalid input!\n";
        int errorLength = strlen(error);
        sys_req(WRITE, DEFAULT_DEVICE, error, &errorLength);
    }

    return flag;
}
```

5.23.1.7 setDate()

```
void setDate ( )
```

Definition at line 379 of file R1commands.c.

```
{

    int count = 4; // used to print year

    char spacer[1] = "\n"; // used to space out terminal outputs
    int spaceCount = 1;

    char instruction1[] = "Please type the desired year. I.E.: yyyy.\n";
    int length = strlen(instruction1);

    sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
    memset(instruction1, '\0', length);

    char year[5] = "\0\0\0\0\0"; // year buffer

    int flag = 0; // thrown if input is invalid

    do
    {
        sys_req(READ, DEFAULT_DEVICE, year, &count);
        if (atoi(year) > 0)
        {

            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            flag = 0;

            char yearUpper[3] = "\0\0\0";
            char yearLower[3] = "\0\0\0";

            yearUpper[0] = year[0];
            yearUpper[1] = year[1];
            yearLower[0] = year[2];
            yearLower[1] = year[3];

            cli();

            outb(0x70, 0x32); // Setting first byte year value
            outb(0x71, intToBCD(atoi(yearUpper)));

            outb(0x70, 0x09); // Setting second byte year value
            outb(0x71, intToBCD(atoi(yearLower)));

            sti();
        }
        else
        {
            char invalid[] = "Invalid year.\n";
            int lengthInval = strlen(invalid);
            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
            memset(invalid, '\0', lengthInval);
            flag = 1;
        }
    } while (flag == 1);

    char instruction2[] = "Please type the desired month. I.E.: mm.\n";
    length = strlen(instruction2);

    sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
    memset(instruction2, '\0', length);

    char month[4] = "\0\0\n\0";
    count = 4; // used to print month

    do
    {
        sys_req(READ, DEFAULT_DEVICE, month, &count);
        if (atoi(month) < 13 && atoi(month) > 0)
        {

            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            flag = 0;

            cli();

            outb(0x70, 0x08); // Setting month value
            outb(0x71, intToBCD(atoi(month)));

            sti();
        }
        else
        {
            char invalid[] = "Invalid month.\n";
            int lengthInval = strlen(invalid);
            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
            memset(invalid, '\0', lengthInval);
            flag = 1;
        }
    } while (flag == 1);

    char instruction3[] = "Please type the desired day of month. I.E.: dd.\n";

    length = strlen(instruction3);
    sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
    memset(instruction3, '\0', length);

    char day[4] = "\0\0\n\0";
    count = 4; // used to print day

    do
    {
        sys_req(READ, DEFAULT_DEVICE, day, &count);
        sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
        if ((atoi(year) % 4 == 0 && atoi(year) % 100 != 0) || atoi(year) % 400 == 0)
        { // checking for leap year

            char leapYear[] = "This is a leap year. February has 29 days.\n";
            length = strlen(leapYear);

            sys_req(WRITE, DEFAULT_DEVICE, leapYear, &length);
            memset(leapYear, '\0', length);

            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)
            {
                flag = 1;
                char invalid[] = "Invalid day.\n";
                length = strlen(invalid);
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
                memset(invalid, '\0', length);
            }
            else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) && atoi(day) > 30)
            {
                flag = 1;
                char invalid[] = "Invalid day.\n";
                length = strlen(invalid);
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
                memset(invalid, '\0', length);
            }
            else if ((atoi(month) == 2) && atoi(day) > 29)
            {
                flag = 1;
                char invalid[] = "Invalid day.\n";
                length = strlen(invalid);
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
                memset(invalid, '\0', length);
            }
            else
            {

                flag = 0;
                cli();

                outb(0x70, 0x07); // Setting day of month value
                outb(0x71, intToBCD(atoi(day)));

                sti();
            }
        }
        else if (atoi(year) % 4 != 0 || atoi(year) % 400 != 0)
        { // checking for leap year

            char noLeap[] = "This is not a leap year.\n";
            length = strlen(noLeap);
            sys_req(WRITE, DEFAULT_DEVICE, noLeap, &length);
            memset(noLeap, '\0', length);

            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)
            {
                flag = 1;
                char invalid[] = "Invalid day.\n";
                length = strlen(invalid);
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
                memset(invalid, '\0', length);
            }
            else if ((atoi(month) == 4 || atoi(month) == 6 || atoi(month) == 9 || atoi(month) == 11) && atoi(day) > 30)
            {
                flag = 1;
                char invalid[] = "Invalid day.\n";
                length = strlen(invalid);
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
                memset(invalid, '\0', length);
            }
            else if ((atoi(month) == 2) && atoi(day) > 28)
            {
                flag = 1;
                char invalid[] = "Invalid day.\n";
                length = strlen(invalid);
                sys_req(WRITE, DEFAULT_DEVICE, invalid, &length);
                memset(invalid, '\0', length);
            }
            else
            {

                cli();

                outb(0x70, 0x07); // Setting day of month value
                outb(0x71, intToBCD(atoi(day)));

                sti();
            }
        }

    } while (flag == 1);

    char exitMessage[] = "The date has been set.\n";
    int exitLength = strlen(exitMessage);
    sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
    memset(exitMessage, '\0', exitLength);
    memset(spacer, '\0', spaceCount);

    return 0;
}
```

5.23.1.8 setTime()

```
void setTime ( )
```

Definition at line 221 of file R1commands.c.

```
{

    int count = 4; // counter for printing

    char spacer[1] = "\n"; // used to space out terminal outputs
    int spaceCount = 1;

    char instruction1[] = "Please type the desired hours. I.E.: hh.\n";

    int length = strlen(instruction1);

    sys_req(WRITE, DEFAULT_DEVICE, instruction1, &length);
    memset(instruction1, '\0', length);

    char hour[4] = "\0\0\n\0";

    int flag = 0;

    do
    {
        sys_req(READ, DEFAULT_DEVICE, hour, &count);
        if (atoi(hour) < 24 && atoi(hour) >= 0)
        {

            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            flag = 0;
        }
        else
        {
            char invalid[] = "Invalid hours.\n";
            int lengthInval = strlen(invalid);
            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
            memset(invalid, '\0', lengthInval);
            flag = 1;
        }
    } while (flag == 1);

    char instruction2[] = "Please type the desired minutes. I.E.: mm.\n";

    length = strlen(instruction2);

    sys_req(WRITE, DEFAULT_DEVICE, instruction2, &length);
    memset(instruction2, '\0', length);

    char minute[4] = "\0\0\n\0";

    do
    {
        sys_req(READ, DEFAULT_DEVICE, minute, &count);
        if (atoi(minute) < 60 && atoi(minute) >= 0)
        {

            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            flag = 0;
        }
        else
        {
            char invalid[] = "Invalid minutes.\n";
            int lengthInval = strlen(invalid);
            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
            memset(invalid, '\0', lengthInval);
            flag = 1;
        }
    } while (flag == 1);

    char instruction3[] = "Please type the desired seconds. I.E.: ss.\n";

    length = strlen(instruction3);

    sys_req(WRITE, DEFAULT_DEVICE, instruction3, &length);
    memset(instruction3, '\0', length);

    char second[4] = "\0\0\n\0";

    do
    {
        sys_req(READ, DEFAULT_DEVICE, second, &count);
        if (atoi(second) < 60 && atoi(second) >= 0)
        {

            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            flag = 0;
        }
        else
        {
            char invalid[] = "Invalid seconds.\n";
            int lengthInval = strlen(invalid);
            sys_req(WRITE, DEFAULT_DEVICE, spacer, &spaceCount);
            sys_req(WRITE, DEFAULT_DEVICE, invalid, &lengthInval);
            memset(invalid, '\0', lengthInval);
            flag = 1;
        }
    } while (flag == 1);

    cli();

    outb(0x70, 0x04); // Hour
    outb(0x71, intToBCD(atoi(hour)));

    outb(0x70, 0x02); // Minute
    outb(0x71, intToBCD(atoi(minute)));

    outb(0x70, 0x00); // Second
    outb(0x71, intToBCD(atoi(second)));

    sti();

    char exitMessage[] = "The time has been set.\n";
    int exitLength = strlen(exitMessage);
    sys_req(WRITE, DEFAULT_DEVICE, exitMessage, &exitLength);
    memset(exitMessage, '\0', exitLength);
    memset(spacer, '\0', spaceCount);

    return 0;
}
```

5.23.1.9 version()

```
void version ( )
```

Definition at line 177 of file R1commands.c.

```
{

    char version[] = "Version 2.0\n";

    int tempBuffer = strlen(version);

    sys_req(WRITE, DEFAULT_DEVICE, (char *)version, &tempBuffer);
    memset(version, '\0', tempBuffer);

    return 0;
}
```

5.24 modules/R2/R2_Internal_Functions_And_Structures.c File Reference

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

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

Functions

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

Variables

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

5.24.1 Function Documentation

5.24.1.1 allocatePCB()

```
PCB* allocatePCB ( )
```

Definition at line 14 of file R2_Internal_Functions_And_Structures.c.

```
{
    //COLTON WILL PROGRAM THIS FUNCTION

    //allocatePCB() will use sys_alloc_mem() to allocate memory for a new PCB, possible including the stack, and perform any reasonable initialization.
    PCB *newPCB = (PCB *)sys_alloc_mem(sizeof(PCB));

    char name[20] = "newPCB";
    strcpy(newPCB->processName, name);

    newPCB->suspendedStatus = 1;
    newPCB->runningStatus = -1;
    newPCB->stackTop = (newPCB->stackTop + 1024);
    newPCB->stackBase = newPCB->stackBase;
    newPCB->priority = 0;

    // Setting the PCBs prev and next PCB
    newPCB->nextPCB = NULL;
    newPCB->prevPCB = NULL;

    newPCB->processClass = NULL;

    return newPCB;
}
```

5.24.1.2 allocateQueues()

```
void allocateQueues ( )
```

Definition at line 377 of file R2_Internal_Functions_And_Structures.c.

```
{
    ready = sys_alloc_mem(sizeof(queue));
    ready->count = 0;
    ready->head = NULL;
    ready->tail = NULL;
    blocked = sys_alloc_mem(sizeof(queue));
    blocked->count = 0;
    blocked->head = NULL;
    blocked->tail = NULL;
    suspendedReady = sys_alloc_mem(sizeof(queue));
    suspendedReady->count = 0;
    suspendedReady->head = NULL;
    suspendedReady->tail = NULL;
    suspendedBlocked = sys_alloc_mem(sizeof(queue));
    suspendedBlocked->count = 0;
    suspendedBlocked->head = NULL;
    suspendedBlocked->tail = NULL;
}
```

5.24.1.3 findPCB()

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

Definition at line 77 of file R2_Internal_Functions_And_Structures.c.

```
{
    // ANASTASE WILL PROGRAM THIS FUNCTION

    //findPCB() will search all queues for a process with a given name.

    if (strlen(processName) > 20)
    {

        char error_message[30] = "Invalid process name.\n";
        int error_size = strlen(error_message);
        sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
        return NULL;
        //return cz we have to stop if the process name is too long
    }
    else
    {
        PCB *tempPCB = ready->head;
        int value = 0;
        while (value <= ready->count)
        {
            if (strcmp(tempPCB->processName, processName) == 0)
            {
                 return tempPCB;
             }
             else
             {
                 tempPCB = tempPCB->nextPCB;
                 value++;
             }
         }

         tempPCB = blocked->head;
         value = 0;
         while (value <= blocked->count)
         {
             if (strcmp(tempPCB->processName, processName) == 0)
             {
                 return tempPCB;
             }
             else
             {
                 tempPCB = tempPCB->nextPCB;
                 value++;
             }
         }

        tempPCB = suspendedBlocked->head;
        value = 0;
        while (value <= suspendedBlocked->count)
        {
            if (strcmp(tempPCB->processName, processName) == 0)
            {
                return tempPCB;
            }
            else
            {
                tempPCB = tempPCB->nextPCB;
                value++;
            }
        }

        tempPCB = suspendedReady->head;
        value = 0;
        while (value <= suspendedReady->count)
        {
            if (strcmp(tempPCB->processName, processName) == 0)
            {
                return tempPCB;
            }
            else
            {
                tempPCB = tempPCB->nextPCB;
                value++;
            }
        }

        return NULL;
    }
}
```

5.24.1.4 freePCB()

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

Definition at line 39 of file R2_Internal_Functions_And_Structures.c.

```
{
    // ANASTASE WILL PROGRAM THIS FUNCTION

    //freePCB() will use sys_free_mem() to free all memory associated with a given PCB (the stack, the PCB itself, etc.)

    return sys_free_mem(PCB_to_free);
}
```

5.24.1.5 getBlocked()

```
queue* getBlocked ( )
```

Definition at line 402 of file R2_Internal_Functions_And_Structures.c.

```
{
    return blocked;
}
```

5.24.1.6 getReady()

```
queue* getReady ( )
```

Definition at line 397 of file R2_Internal_Functions_And_Structures.c.

```
{
    return ready;
}
```

5.24.1.7 getSuspendedBlocked()

```
queue* getSuspendedBlocked ( )
```

Definition at line 412 of file R2_Internal_Functions_And_Structures.c.

```
{
    return suspendedBlocked;
}
```

5.24.1.8 getSuspendedReady()

```
queue* getSuspendedReady ( )
```

Definition at line 407 of file R2_Internal_Functions_And_Structures.c.

```
{
    return suspendedReady;
}
```

5.24.1.9 insertPCB()

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

Definition at line 158 of file R2_Internal_Functions_And_Structures.c.

```
{
    //BENJAMIN WILL PROGRAM THIS FUNCTION

    //insertPCB() will insert a PCB into the appropriate queue.
    //Note: The ready queue is a priority queue and the blocked queue is a FIFO queue.

    if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 1)
    { // Insert into ready queue
        PCB *tempPtr = ready->head;

        if (tempPtr != NULL)
        {
            int temp = 0;
            while (temp <= ready->count)
            {
                if (PCB_to_insert->priority < tempPtr->priority)
                {
                    tempPtr = tempPtr->nextPCB;
                }
                else if (PCB_to_insert->priority >= tempPtr->priority)
                {
                    PCB_to_insert->nextPCB = tempPtr;
                    PCB_to_insert->prevPCB = tempPtr->prevPCB;
                    tempPtr->prevPCB = PCB_to_insert;
                }
                else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
                {
                    tempPtr->nextPCB = PCB_to_insert;
                    PCB_to_insert->prevPCB = tempPtr;
                    ready->tail = PCB_to_insert;
                }
                temp++;
            }
            ready->count++;
        }
        else
        {
            ready->count++;
            ready->head = PCB_to_insert;
            ready->tail = PCB_to_insert;
        }
    }
    else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
    { // Insert into suspended ready queue
        PCB *tempPtr = suspendedReady->head;

        if (tempPtr != NULL)
        {
            int temp = 0;
            while (temp <= suspendedReady->count)
            {
                if (PCB_to_insert->priority < tempPtr->priority)
                {
                    tempPtr = tempPtr->nextPCB;
                }
                else if (PCB_to_insert->priority >= tempPtr->priority)
                {
                    PCB_to_insert->nextPCB = tempPtr;
                    PCB_to_insert->prevPCB = tempPtr->prevPCB;
                    tempPtr->prevPCB = PCB_to_insert;
                }
                else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
                {
                    tempPtr->nextPCB = PCB_to_insert;
                    PCB_to_insert->prevPCB = tempPtr;
                    suspendedReady->tail = PCB_to_insert;
                }
                temp++;
            }
            suspendedReady->count++;
        }
        else
        {
            suspendedReady->count++;
            suspendedReady->head = PCB_to_insert;
            suspendedReady->tail = PCB_to_insert;
        }
    }
    else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 1)
    { // Insert into blocked queue
        PCB *tempPtr = blocked->tail;

        tempPtr->nextPCB = PCB_to_insert;
        PCB_to_insert->prevPCB = tempPtr;
        blocked->tail = PCB_to_insert;
    }
    else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
    { // Insert into suspended blocked queue
        PCB *tempPtr = suspendedBlocked->tail;

        tempPtr->nextPCB = PCB_to_insert;
        PCB_to_insert->prevPCB = tempPtr;
        suspendedBlocked->tail = PCB_to_insert;
    }
}
```

5.24.1.10 removePCB()

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

Definition at line 255 of file R2_Internal_Functions_And_Structures.c.

```
{
    //BENJAMIN WILL PROGRAM THIS FUNCTION

    //removePCB() will remove a PCB from the queue in which it is currently stored.

    PCB *removedPCB = findPCB(PCB_to_remove->processName);
    if (removedPCB == NULL)
    {
        return 1;
    }
    else if (removedPCB == ready->head)
    {
        PCB *removedNext = removedPCB->nextPCB;

        ready->head = removedNext;
        removedNext->prevPCB = NULL;
        removedPCB->nextPCB = NULL;
        ready->count--;
        return 0;
    }
    else if (removedPCB == blocked->head)
    {
        PCB *removedNext = removedPCB->nextPCB;
        blocked->head = removedNext;
        removedNext->prevPCB = NULL;
        removedPCB->nextPCB = NULL;
        blocked->count--;
        return 0;
    }
    else if (removedPCB == suspendedReady->head)
    {
        PCB *removedNext = removedPCB->nextPCB;

        suspendedReady->head = removedNext;
        removedNext->prevPCB = NULL;
        removedPCB->nextPCB = NULL;
        suspendedReady->count--;
        return 0;
    }
    else if (removedPCB == suspendedBlocked->head)
    {
        PCB *removedNext = removedPCB->nextPCB;

        suspendedBlocked->head = removedNext;
        removedNext->prevPCB = NULL;
        removedPCB->nextPCB = NULL;
        suspendedBlocked->count--;
        return 0;
    }
    else if (removedPCB == ready->tail)
    {
        PCB *removedPrev = removedPCB->prevPCB;

        ready->tail = removedPrev;
        removedPrev->nextPCB = NULL;
        removedPCB->prevPCB = NULL;
        ready->count--;
        return 0;
    }
    else if (removedPCB == blocked->tail)
    {
        PCB *removedPrev = removedPCB->prevPCB;

        blocked->tail = removedPrev;
        removedPrev->nextPCB = NULL;
        removedPCB->prevPCB = NULL;
        blocked->count--;
        return 0;
    }
    else if (removedPCB == suspendedReady->tail)
    {
        PCB *removedPrev = removedPCB->prevPCB;

        suspendedReady->tail = removedPrev;
        removedPrev->nextPCB = NULL;
        removedPCB->prevPCB = NULL;
        suspendedReady->count--;
        return 0;
    }
    else if (removedPCB == suspendedBlocked->tail)
    {
        PCB *removedPrev = removedPCB->prevPCB;

        suspendedBlocked->tail = removedPrev;
        removedPrev->nextPCB = NULL;
        removedPCB->prevPCB = NULL;
        suspendedBlocked->count--;
        return 0;
    }
    else
    {
        PCB *tempPrev = removedPCB->prevPCB;
        PCB *tempNext = removedPCB->nextPCB;

        tempPrev->nextPCB = tempNext;
        tempNext->prevPCB = tempPrev;

        removedPCB->nextPCB = NULL;
        removedPCB->prevPCB = NULL;

        if (removedPCB->runningStatus == 0 && removedPCB->suspendedStatus == 1)
        {
            ready->count--;
        }
        else if (removedPCB->runningStatus == -1 && removedPCB->suspendedStatus == 1)
        {
            blocked->count--;
        }
        else if (removedPCB->runningStatus == 0 && removedPCB->suspendedStatus == 0)
        {
            suspendedReady->count--;
        }
        else if (removedPCB->runningStatus == -1 && removedPCB->suspendedStatus == 0)
        {
            suspendedBlocked->count--;
        }

        return 0;
    }
}
```

5.24.1.11 setupPCB()

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

Definition at line 48 of file R2_Internal_Functions_And_Structures.c.

```
{
    //COLTON WILL PROGRAM THIS FUNCTION

    //setupPcb() will call allocatePCB() to create an empty PCB, initializes the PCB information, sets the PCB state to ready, not suspended.

    PCB *returnedPCB = allocatePCB();

    if (findPCB(processName)->processName == processName)
    {
        char message[] = "There is already a PCB with this name.\n";
        int messLength = strlen(message);
        sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);

        returnedPCB = NULL;
    }
    else
    {

        strcpy(returnedPCB->processName, processName);
        returnedPCB->processClass = processClass;
        returnedPCB->priority = processPriority;
        returnedPCB->runningStatus = 0;
        returnedPCB->suspendedStatus = 1;
    }

    return returnedPCB;
}
```

5.24.2 Variable Documentation

5.24.2.1 blocked

```
queue* blocked
```

Definition at line 8 of file R2_Internal_Functions_And_Structures.c.

5.24.2.2 ready

```
queue* ready
```

Definition at line 7 of file R2_Internal_Functions_And_Structures.c.

5.24.2.3 suspendedBlocked

```
queue* suspendedBlocked
```

Definition at line 10 of file R2_Internal_Functions_And_Structures.c.

5.24.2.4 suspendedReady

```
queue* suspendedReady
```

Definition at line 9 of file R2_Internal_Functions_And_Structures.c.

5.25 modules/R2/R2_Internal_Functions_And_Structures.h File Reference

Classes

- struct PCB
- struct queue

Typedefs

- typedef struct PCB PCB
- typedef struct queue queue

Functions

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

5.25.1 Typedef Documentation

5.25.1.1 PCB

```
typedef struct PCB PCB
```

5.25.1.2 queue

```
typedef struct queue queue
```

5.25.2 Function Documentation

5.25.2.1 allocatePCB()

```
PCB∗ allocatePCB ( )
```

Definition at line 14 of file R2_Internal_Functions_And_Structures.c.

```
{
    //COLTON WILL PROGRAM THIS FUNCTION

    //allocatePCB() will use sys_alloc_mem() to allocate memory for a new PCB, possible including the stack, and perform any reasonable initialization.
    PCB *newPCB = (PCB *)sys_alloc_mem(sizeof(PCB));

    char name[20] = "newPCB";
    strcpy(newPCB->processName, name);

    newPCB->suspendedStatus = 1;
    newPCB->runningStatus = -1;
    newPCB->stackTop = (newPCB->stackTop + 1024);
    newPCB->stackBase = newPCB->stackBase;
    newPCB->priority = 0;

    // Setting the PCBs prev and next PCB
    newPCB->nextPCB = NULL;
    newPCB->prevPCB = NULL;

    newPCB->processClass = NULL;

    return newPCB;
}
```

5.25.2.2 allocateQueues()

```
void allocateQueues ( )
```

Definition at line 377 of file R2_Internal_Functions_And_Structures.c.

```
{
    ready = sys_alloc_mem(sizeof(queue));
    ready->count = 0;
    ready->head = NULL;
    ready->tail = NULL;
    blocked = sys_alloc_mem(sizeof(queue));
    blocked->count = 0;
    blocked->head = NULL;
    blocked->tail = NULL;
    suspendedReady = sys_alloc_mem(sizeof(queue));
    suspendedReady->count = 0;
    suspendedReady->head = NULL;
    suspendedReady->tail = NULL;
    suspendedBlocked = sys_alloc_mem(sizeof(queue));
    suspendedBlocked->count = 0;
    suspendedBlocked->head = NULL;
    suspendedBlocked->tail = NULL;
}
```

5.25.2.3 findPCB()

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

Definition at line 77 of file R2_Internal_Functions_And_Structures.c.

```
{
    // ANASTASE WILL PROGRAM THIS FUNCTION

    //findPCB() will search all queues for a process with a given name.

    if (strlen(processName) > 20)
    {

        char error_message[30] = "Invalid process name.\n";
        int error_size = strlen(error_message);
        sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
        return NULL;
        //return cz we have to stop if the process name is too long
    }
    else
    {
        PCB *tempPCB = ready->head;
        int value = 0;
        while (value <= ready->count)
        {
            if (strcmp(tempPCB->processName, processName) == 0)
            {
                return tempPCB;
            }
            else
            {
                tempPCB = tempPCB->nextPCB;
                value++;
            }
        }

        tempPCB = blocked->head;
        value = 0;
        while (value <= blocked->count)
        {
            if (strcmp(tempPCB->processName, processName) == 0)
            {
                return tempPCB;
            }
            else
            {
                tempPCB = tempPCB->nextPCB;
                value++;
            }
        }

        tempPCB = suspendedBlocked->head;
        value = 0;
        while (value <= suspendedBlocked->count)
        {
            if (strcmp(tempPCB->processName, processName) == 0)
            {
                return tempPCB;
            }
            else
            {
                tempPCB = tempPCB->nextPCB;
                value++;
            }
        }

        tempPCB = suspendedReady->head;
        value = 0;
        while (value <= suspendedReady->count)
        {
            if (strcmp(tempPCB->processName, processName) == 0)
            {
                return tempPCB;
            }
            else
            {
                tempPCB = tempPCB->nextPCB;
                value++;
            }
        }

        return NULL;
    }
}
```

5.25.2.4 freePCB()

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

Definition at line 39 of file R2_Internal_Functions_And_Structures.c.

```
{
    // ANASTASE WILL PROGRAM THIS FUNCTION

    //freePCB() will use sys_free_mem() to free all memory associated with a given PCB (the stack, the PCB itself, etc.)

    return sys_free_mem(PCB_to_free);
}
```

5.25.2.5 getBlocked()

```
queue* getBlocked ( )
```

Definition at line 402 of file R2_Internal_Functions_And_Structures.c.

```
{
    return blocked;
}
```

5.25.2.6 getReady()

```
queue* getReady ( )
```

Definition at line 397 of file R2_Internal_Functions_And_Structures.c.

```
{
    return ready;
}
```

5.25.2.7 getSuspendedBlocked()

```
queue* getSuspendedBlocked ( )
```

Definition at line 412 of file R2_Internal_Functions_And_Structures.c.

```
{
    return suspendedBlocked;
}
```

5.25.2.8 getSuspendedReady()

```
queue* getSuspendedReady ( )
```

Definition at line 407 of file R2_Internal_Functions_And_Structures.c.

```
{
    return suspendedReady;
}
```

5.25.2.9 insertPCB()

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

Definition at line 158 of file R2_Internal_Functions_And_Structures.c.

```
{
    //BENJAMIN WILL PROGRAM THIS FUNCTION

    //insertPCB() will insert a PCB into the appropriate queue.
    //Note: The ready queue is a priority queue and the blocked queue is a FIFO queue.

    if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 1)
    { // Insert into ready queue
        PCB *tempPtr = ready->head;

        if (tempPtr != NULL)
        {
            int temp = 0;
            while (temp <= ready->count)
            {
                if (PCB_to_insert->priority < tempPtr->priority)
                {
                    tempPtr = tempPtr->nextPCB;
                }
                else if (PCB_to_insert->priority >= tempPtr->priority)
                {
                    PCB_to_insert->nextPCB = tempPtr;
                    PCB_to_insert->prevPCB = tempPtr->prevPCB;
                    tempPtr->prevPCB = PCB_to_insert;
                }
                else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
                {
                    tempPtr->nextPCB = PCB_to_insert;
                    PCB_to_insert->prevPCB = tempPtr;
                    ready->tail = PCB_to_insert;
                }
                temp++;
            }
            ready->count++;
        }
        else
        {
            ready->count++;
            ready->head = PCB_to_insert;
            ready->tail = PCB_to_insert;
        }
    }
    else if (PCB_to_insert->runningStatus == 0 && PCB_to_insert->suspendedStatus == 0)
    { // Insert into suspended ready queue
        PCB *tempPtr = suspendedReady->head;

        if (tempPtr != NULL)
        {
            int temp = 0;
            while (temp <= suspendedReady->count)
            {
                if (PCB_to_insert->priority < tempPtr->priority)
                {
                    tempPtr = tempPtr->nextPCB;
                }
                else if (PCB_to_insert->priority >= tempPtr->priority)
                {
                    PCB_to_insert->nextPCB = tempPtr;
                    PCB_to_insert->prevPCB = tempPtr->prevPCB;
                    tempPtr->prevPCB = PCB_to_insert;
                }
                else if (PCB_to_insert->priority < tempPtr->priority && tempPtr->nextPCB == NULL)
                {
                    tempPtr->nextPCB = PCB_to_insert;
                    PCB_to_insert->prevPCB = tempPtr;
                    suspendedReady->tail = PCB_to_insert;
                }
                temp++;
            }
            suspendedReady->count++;
        }
        else
        {
            suspendedReady->count++;
            suspendedReady->head = PCB_to_insert;
            suspendedReady->tail = PCB_to_insert;
        }
    }
    else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 1)
    { // Insert into blocked queue
        PCB *tempPtr = blocked->tail;

        tempPtr->nextPCB = PCB_to_insert;
        PCB_to_insert->prevPCB = tempPtr;
        blocked->tail = PCB_to_insert;
    }
    else if (PCB_to_insert->runningStatus == -1 && PCB_to_insert->suspendedStatus == 0)
    { // Insert into suspended blocked queue
        PCB *tempPtr = suspendedBlocked->tail;

        tempPtr->nextPCB = PCB_to_insert;
        PCB_to_insert->prevPCB = tempPtr;
        suspendedBlocked->tail = PCB_to_insert;
    }
}
```

5.25.2.10 removePCB()

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

Definition at line 255 of file R2_Internal_Functions_And_Structures.c.

```
{
    //BENJAMIN WILL PROGRAM THIS FUNCTION

    //removePCB() will remove a PCB from the queue in which it is currently stored.

    PCB *removedPCB = findPCB(PCB_to_remove->processName);
    if (removedPCB == NULL)
    {
        return 1;
    }
    else if (removedPCB == ready->head)
    {
        PCB *removedNext = removedPCB->nextPCB;

        ready->head = removedNext;
        removedNext->prevPCB = NULL;
        removedPCB->nextPCB = NULL;
        ready->count--;
        return 0;
    }
    else if (removedPCB == blocked->head)
    {
        PCB *removedNext = removedPCB->nextPCB;
        blocked->head = removedNext;
        removedNext->prevPCB = NULL;
        removedPCB->nextPCB = NULL;
        blocked->count--;
        return 0;
    }
    else if (removedPCB == suspendedReady->head)
    {
        PCB *removedNext = removedPCB->nextPCB;

        suspendedReady->head = removedNext;
        removedNext->prevPCB = NULL;
        removedPCB->nextPCB = NULL;
        suspendedReady->count--;
        return 0;
    }
    else if (removedPCB == suspendedBlocked->head)
    {
        PCB *removedNext = removedPCB->nextPCB;

        suspendedBlocked->head = removedNext;
        removedNext->prevPCB = NULL;
        removedPCB->nextPCB = NULL;
        suspendedBlocked->count--;
        return 0;
    }
    else if (removedPCB == ready->tail)
    {
        PCB *removedPrev = removedPCB->prevPCB;

        ready->tail = removedPrev;
        removedPrev->nextPCB = NULL;
        removedPCB->prevPCB = NULL;
        ready->count--;
        return 0;
    }
    else if (removedPCB == blocked->tail)
    {
        PCB *removedPrev = removedPCB->prevPCB;

        blocked->tail = removedPrev;
        removedPrev->nextPCB = NULL;
        removedPCB->prevPCB = NULL;
        blocked->count--;
        return 0;
    }
    else if (removedPCB == suspendedReady->tail)
    {
        PCB *removedPrev = removedPCB->prevPCB;

        suspendedReady->tail = removedPrev;
        removedPrev->nextPCB = NULL;
        removedPCB->prevPCB = NULL;
        suspendedReady->count--;
        return 0;
    }
    else if (removedPCB == suspendedBlocked->tail)
    {
        PCB *removedPrev = removedPCB->prevPCB;

        suspendedBlocked->tail = removedPrev;
        removedPrev->nextPCB = NULL;
        removedPCB->prevPCB = NULL;
        suspendedBlocked->count--;
        return 0;
    }
    else
    {
        PCB *tempPrev = removedPCB->prevPCB;
        PCB *tempNext = removedPCB->nextPCB;

        tempPrev->nextPCB = tempNext;
        tempNext->prevPCB = tempPrev;

        removedPCB->nextPCB = NULL;
        removedPCB->prevPCB = NULL;

        if (removedPCB->runningStatus == 0 && removedPCB->suspendedStatus == 1)
        {
            ready->count--;
        }
        else if (removedPCB->runningStatus == -1 && removedPCB->suspendedStatus == 1)
        {
            blocked->count--;
        }
        else if (removedPCB->runningStatus == 0 && removedPCB->suspendedStatus == 0)
        {
            suspendedReady->count--;
        }
        else if (removedPCB->runningStatus == -1 && removedPCB->suspendedStatus == 0)
        {
            suspendedBlocked->count--;
        }

        return 0;
    }
}
```

5.25.2.11 setupPCB()

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

Definition at line 48 of file R2_Internal_Functions_And_Structures.c.

```
{
    //COLTON WILL PROGRAM THIS FUNCTION

    //setupPcb() will call allocatePCB() to create an empty PCB, initializes the PCB information, sets the PCB state to ready, not suspended.

    PCB *returnedPCB = allocatePCB();

    if (findPCB(processName)->processName == processName)
    {
        char message[] = "There is already a PCB with this name.\n";
        int messLength = strlen(message);
        sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);

        returnedPCB = NULL;
    }
    else
    {

        strcpy(returnedPCB->processName, processName);
        returnedPCB->processClass = processClass;
        returnedPCB->priority = processPriority;
        returnedPCB->runningStatus = 0;
        returnedPCB->suspendedStatus = 1;
    }

    return returnedPCB;
}
```

5.26 modules/R2/R2commands.c File Reference

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

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

Functions

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

5.26.1 Function Documentation

5.26.1.1 blockPCB()

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

Definition at line 113 of file R2commands.c.

```
{ // ANASTASE WILL PROGRAM THIS FUNCTION

    // find pcb and validate process name
    PCB *pcb_to_block = findPCB(processName);

    if (pcb_to_block != NULL)
    {
        pcb_to_block->runningStatus = -1; // blocked
        removePCB(pcb_to_block);
        insertPCB(pcb_to_block);

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

5.26.1.2 createPCB()

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

Definition at line 11 of file R2commands.c.

```
{ // BENJAMIN WILL PROGRAM THIS FUNCTION
    /*
    The createPCB command will call setupPCB() and insert the PCB in the appropriate queue
    */
    /*
    Error Checking:
    Name must be unique and valid.
    Class must be valid.
    Priority must be valid.
    */

    if (findPCB(processName) != NULL || strlen(processName) > 20)
    { // Check if the process has a unique name, and if it has a valid name.
        char errMsg[125];
        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");
        int errLen = strlen(errMsg);
        sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
    }
    else if (processClass != 'a' && processClass != 's')
    { // Check if the process has a valid class.
        char errMsg[100];
        strcpy(errMsg, "The PCB could not be created as it does not have a valid class!\n");
        int errLen = strlen(errMsg);
        sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
    }
    else if (processPriority < 0 || processPriority > 9)
    { // Check if the process has a valid priority.
        char errMsg[100];
        strcpy(errMsg, "The PCB could not be created as it does not have a valid priority!\n");
        int errLen = strlen(errMsg);
        sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
    }
    else
    { // Make the PCB
        PCB *createdPCB = setupPCB(processName, processClass, processPriority);

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

        insertPCB(createdPCB);
    }
}
```

5.26.1.3 deletePCB()

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

Definition at line 56 of file R2commands.c.

```
{ // BENJAMIN WILL PROGRAM THIS FUNCTION
    /*
    The deletePCB command will remove a PCB from the appropriate queue and then free all associated memory.
    This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
    */
    /*
    Error Checking:
    Name must be valid.
    */

    if (strlen(processName) > 20)
    { // Check if the process has a valid name.
        char errMsg[100];
        strcpy(errMsg, "The PCB could not be deleted as the name is longer than 20 characters!\n");
        int errLen = strlen(errMsg);
        sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
    }

    PCB *PCB_to_delete = findPCB(processName);

    if (PCB_to_delete == NULL)
    {
        char errMsg[42] = "The PCB you want to remove does not exist\n";
        int errMsgLen = 42;
        sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
    }
    else
    {
        int removed = removePCB(PCB_to_delete);
        if (removed == 1)
        {
            char errMsg[] = "The PCB could not be unlinked.\n";
            int errMsgLen = strlen(errMsg);
            sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
        }
        else
        {
            int result = sys_free_mem(PCB_to_delete);
            if (result == -1)
            {
                char errMsg[50];
                strcpy(errMsg, "The PCB could not be successfully deleted\n");
                int errLen = strlen(errMsg);
                sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
            }
            else
            {
                char msg[50];
                strcpy(msg, "The desired PCB was deleted\n");
                int msgLen = strlen(msg);
                sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
            }
        }
    }
}
```

5.26.1.4 resumePCB()

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

Definition at line 187 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Places a PCB in the not suspended state and reinserts it into the appropriate queue
    */

    PCB *PCBtoResume = findPCB(processName);

    if (PCBtoResume == NULL || strlen(processName) > 20)
    {
        char nameError[] = "This is not a valid name.\n";
        int printCount = strlen(nameError);
        sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
    }
    else
    {
        removePCB(PCBtoResume);
        PCBtoResume->suspendedStatus = 1;
        insertPCB(PCBtoResume);

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

5.26.1.5 setPCBPriority()

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

Definition at line 217 of file R2commands.c.

```
{ // ANASTASE WILL PROGRAM THIS FUNCTION

    // Sets a PCB's priority and reinserts the process into the correct place in the correct queue

    /*
    Error Checking:
    Name must be valid.
    newPriority
    */

    // find the process and validate the name
    PCB *tempPCB = findPCB(processName);

    if ((tempPCB != NULL) && (newProcessPriority >= 0) && (newProcessPriority < 10))
    {
        tempPCB->priority = newProcessPriority;
        removePCB(tempPCB);
        insertPCB(tempPCB);

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

5.26.1.6 showAll()

```
void showAll ( )
```

Definition at line 612 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Displays the following information for each PCB in the ready and blocked queues:
        Process Name
        Class
        State
        Suspended Status
        Priority
    */
    /*
    Error Checking:
    None
    */

    showReady();
    showSuspendedReady();
    showBlocked();
    showSuspendedBlocked();
}
```

5.26.1.7 showBlocked()

```
void showBlocked ( )
```

Definition at line 559 of file R2commands.c.

```
{ // ANASTASE WILL PROGRAM THIS FUNCTION
    /*
    Displays the following information for each PCB in the blocked queue:
        Process Name
        Class
        State
        Suspended Status
        Priority
        HEAD
    */
    /*
    Error Checking:
    None
    */

    // check

    char print_message[30] = "The blocked queue:\n";
    int message_size = strlen(print_message);
    sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);

    // printPCBs(blocked);
    queue *tempQueue = getBlocked();
    PCB *tempPtr = tempQueue->head; //PCB_container->head;
    int count = tempQueue->count;

    if (count == 0)
    {
        // the queue is empty
        char error_message[30] = "The queue is empty.\n";
        int error_size = strlen(error_message);
        sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
        return;
    }
    // The queue is not empty

    int value = 0;
    // Testing purpose
    //char print_message[38]="The blocke queue testing:\n";
    //int message_size=strlen(print_message);
    //sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);

    while (value < count)
    { // testing for <== or <
        // Print out the process
        showPCB(tempPtr->processName);
        // increment pcb*tempPtr, the loop variable.
        tempPtr = tempPtr->nextPCB;
        value++;
    }
}
```

5.26.1.8 showPCB()

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

Definition at line 243 of file R2commands.c.

```
{ // BENJAMIN WILL PROGRAM THIS FUNCTION
    /*
    Displays the following information for a PCB:
        Process Name
        Class
        State
        Suspended Status
        Priority
    */

    /*
    Error Checking:
    Name must be valid.
    */

    if (strlen(processName) > 20)
    { // Check if the process has a valid name.
        char errMsg[100];
        strcpy(errMsg, "The PCB could not be shown as the name is longer than 20 characters!\n");
        int errLen = strlen(errMsg);
        sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
    }
    else
    {

        PCB *PCB_to_show = findPCB(processName);

        if (PCB_to_show == NULL)
        { // Check to see if the PCB exists.
            char errMsg[100];
            strcpy(errMsg, "The PCB could not be shown, as it does not exist!\n");
            int errLen = strlen(errMsg);
            sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
        }
        else
        {
            // Print out the PCB name.
            char nameMsg[50];
            strcpy(nameMsg, "The process name is: ");
            int nameMsgLen = strlen(nameMsg);
            sys_req(WRITE, DEFAULT_DEVICE, nameMsg, &nameMsgLen);
            char name[20];
            strcpy(name, PCB_to_show->processName);
            int nameLen = strlen(name);
            sys_req(WRITE, DEFAULT_DEVICE, name, &nameLen);
            char newLine[1];
            strcpy(newLine, "\n");
            int newLineLen = 1;
            sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);

            // Print out PCB class
            char classMsg[50];
            strcpy(classMsg, "The process class is: ");
            int classMsgLen = strlen(classMsg);
            sys_req(WRITE, DEFAULT_DEVICE, classMsg, &classMsgLen);

            if (PCB_to_show->processClass == 'a')
            {
                char appMsg[50];
                strcpy(appMsg, "application");
                int appMsgLen = strlen(appMsg);
                sys_req(WRITE, DEFAULT_DEVICE, appMsg, &appMsgLen);
            }
            else
            {
                char sysMsg[50];
                strcpy(sysMsg, "system");
                int sysMsgLen = strlen(sysMsg);
                sys_req(WRITE, DEFAULT_DEVICE, sysMsg, &sysMsgLen);
            }
            sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);

            // Print out the PCB state

            if (PCB_to_show->runningStatus == 0)
            { // The process is ready.
                char stateMsg[50];
                strcpy(stateMsg, "The process is ready!\n");
                int stateMsgLen = strlen(stateMsg);
                sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
            }
            else if (PCB_to_show->runningStatus == -1)
            { // The process is blocked.
                char stateMsg[50];
                strcpy(stateMsg, "The process is blocked!\n");
                int stateMsgLen = strlen(stateMsg);
                sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
            }
            else if (PCB_to_show->runningStatus == 1)
            { // The process is running.
                char stateMsg[50];
                strcpy(stateMsg, "The process is running!\n");
                int stateMsgLen = strlen(stateMsg);
                sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
            }

            // Print out the PCB suspended status

            if (PCB_to_show->suspendedStatus == 0)
            { // The process is suspended
                char susMsg[50];
                strcpy(susMsg, "The process is suspended!\n");
                int susMsgLen = strlen(susMsg);
                sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
            }
            else if (PCB_to_show->suspendedStatus == 1)
            { // The process is not suspended
                char susMsg[50];
                strcpy(susMsg, "The process is not suspended!\n");
                int susMsgLen = strlen(susMsg);
                sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
            }

            // Print out the PCB priority
            char priorityMsg[50];
            int priorityMsgLen = 0;

            switch (PCB_to_show->priority)
            {
            case 0:
                strcpy(priorityMsg, "The process priority is 0!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 1:
                strcpy(priorityMsg, "The process priority is 1!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 2:
                strcpy(priorityMsg, "The process priority is 2!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 3:
                strcpy(priorityMsg, "The process priority is 3!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 4:
                strcpy(priorityMsg, "The process priority is 4!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 5:
                strcpy(priorityMsg, "The process priority is 5!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 6:
                strcpy(priorityMsg, "The process priority is 6!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 7:
                strcpy(priorityMsg, "The process priority is 7!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 8:
                strcpy(priorityMsg, "The process priority is 8!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 9:
                strcpy(priorityMsg, "The process priority is 9!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            default:
                break;
            }
        }
    }
}
```

5.26.1.9 showReady()

```
void showReady ( )
```

Definition at line 430 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Displays the following information for each PCB in the ready queue:
        Process Name
        Class
        State
        Suspended Status
        Priority
    */
    /*
    Error Checking:
    None
    */

    char message[] = "Printing the ready queue:\n";
    int messLength = strlen(message);
    sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);

    queue *tempQueue = getReady();
    PCB *tempPCB = tempQueue->head;

    int loop = 0;
    int count = tempQueue->count;

    if (count == 0)
    {
        // the queue is empty
        char error_message[30] = "The queue is empty.\n";
        int error_size = strlen(error_message);
        sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
        return;
    }

    while (loop < count)
    {
        showPCB(tempPCB->processName);
        PCB *tempNext = tempPCB->nextPCB;
        loop++;
        tempPCB = tempNext;
    }
}
```

5.26.1.10 showSuspendedBlocked()

```
void showSuspendedBlocked ( )
```

Definition at line 516 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Displays the following information for each PCB in the suspended blocked queue:
        Process Name
        Class
        State
        Suspended Status
        Priority
    */
    /*
    Error Checking:
    None
    */

    char message[] = "Printing the suspended blocked queue:\n";
    int messLength = strlen(message);
    sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);

    queue *tempQueue = getSuspendedBlocked();
    PCB *tempPCB = tempQueue->head;

    int loop = 0;
    int count = tempQueue->count;

    if (count == 0)
    {
        // the queue is empty
        char error_message[30] = "The queue is empty.\n";
        int error_size = strlen(error_message);
        sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
        return;
    }

    while (loop < count)
    {
        showPCB(tempPCB->processName);
        PCB *tempNext = tempPCB->nextPCB;
        loop++;
        tempPCB = tempNext;
    }
}
```

5.26.1.11 showSuspendedReady()

```
void showSuspendedReady ( )
```

Definition at line 473 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Displays the following information for each PCB in the suspended ready queue:
        Process Name
        Class
        State
        Suspended Status
        Priority
    */
    /*
    Error Checking:
    None
    */

    char message[] = "Printing the suspended ready queue:\n";
    int messLength = strlen(message);
    sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);

    queue *tempQueue = getSuspendedReady();
    PCB *tempPCB = tempQueue->head;

    int loop = 0;
    int count = tempQueue->count;

    if (count == 0)
    {
        // the queue is empty
        char error_message[30] = "The queue is empty.\n";
        int error_size = strlen(error_message);
        sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
        return;
    }

    while (loop < count)
    {
        showPCB(tempPCB->processName);
        PCB *tempNext = tempPCB->nextPCB;
        loop++;
        tempPCB = tempNext;
    }
}
```

5.26.1.12 suspendPCB()

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

Definition at line 157 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Places a PCB in the suspended state and reinserts it into the appropriate queue
    */

    PCB *PCBtoSuspend = findPCB(processName);

    if (PCBtoSuspend == NULL || strlen(processName) > 20)
    {
        char nameError[] = "This is not a valid name.\n";
        int printCount = strlen(nameError);
        sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
    }
    else
    {
        removePCB(PCBtoSuspend);
        PCBtoSuspend->suspendedStatus = 0;
        insertPCB(PCBtoSuspend);

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

5.26.1.13 unblockPCB()

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

Definition at line 131 of file R2commands.c.

```
{ // ANASTASE WILL PROGRAM THIS FUNCTION

    /*
    Places a PCB in the unblocked state and reinserts it into the appropriate queue.
    */
    /*
    Error Checking:
    Name must be valid.

    */

    PCB *pcb_to_unblock = findPCB(processName);
    if (pcb_to_unblock != NULL)
    {
        pcb_to_unblock->runningStatus = 0; // ready
        removePCB(pcb_to_unblock);          // is this the right place to put that function?
        insertPCB(pcb_to_unblock);

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

5.27 modules/R2/R2commands.h File Reference

Functions

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

5.27.1 Function Documentation

5.27.1.1 blockPCB()

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

Definition at line 113 of file R2commands.c.

```
{ // ANASTASE WILL PROGRAM THIS FUNCTION

    // find pcb and validate process name
    PCB *pcb_to_block = findPCB(processName);

    if (pcb_to_block != NULL)
    {
        pcb_to_block->runningStatus = -1; // blocked
        removePCB(pcb_to_block);
        insertPCB(pcb_to_block);

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

5.27.1.2 createPCB()

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

Definition at line 11 of file R2commands.c.

```
{ // BENJAMIN WILL PROGRAM THIS FUNCTION
    /*
    The createPCB command will call setupPCB() and insert the PCB in the appropriate queue
    */
    /*
    Error Checking:
    Name must be unique and valid.
    Class must be valid.
    Priority must be valid.
    */

    if (findPCB(processName) != NULL || strlen(processName) > 20)
    { // Check if the process has a unique name, and if it has a valid name.
        char errMsg[125];
        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");
        int errLen = strlen(errMsg);
        sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
    }
    else if (processClass != 'a' && processClass != 's')
    { // Check if the process has a valid class.
        char errMsg[100];
        strcpy(errMsg, "The PCB could not be created as it does not have a valid class!\n");
        int errLen = strlen(errMsg);
        sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
    }
    else if (processPriority < 0 || processPriority > 9)
    { // Check if the process has a valid priority.
        char errMsg[100];
        strcpy(errMsg, "The PCB could not be created as it does not have a valid priority!\n");
        int errLen = strlen(errMsg);
        sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
    }
    else
    { // Make the PCB
        PCB *createdPCB = setupPCB(processName, processClass, processPriority);

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

        insertPCB(createdPCB);
    }
}
```

5.27.1.3 deletePCB()

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

Definition at line 56 of file R2commands.c.

```
{ // BENJAMIN WILL PROGRAM THIS FUNCTION
    /*
    The deletePCB command will remove a PCB from the appropriate queue and then free all associated memory.
    This method will need to find the pcb, unlink it from the appropriate queue, and then free it.
    */
    /*
    Error Checking:
    Name must be valid.
    */

    if (strlen(processName) > 20)
    { // Check if the process has a valid name.
        char errMsg[100];
        strcpy(errMsg, "The PCB could not be deleted as the name is longer than 20 characters!\n");
        int errLen = strlen(errMsg);
        sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
    }

    PCB *PCB_to_delete = findPCB(processName);

    if (PCB_to_delete == NULL)
    {
        char errMsg[42] = "The PCB you want to remove does not exist\n";
        int errMsgLen = 42;
        sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
    }
    else
    {
        int removed = removePCB(PCB_to_delete);
        if (removed == 1)
        {
            char errMsg[] = "The PCB could not be unlinked.\n";
            int errMsgLen = strlen(errMsg);
            sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errMsgLen);
        }
        else
        {
            int result = sys_free_mem(PCB_to_delete);
            if (result == -1)
            {
                char errMsg[50];
                strcpy(errMsg, "The PCB could not be successfully deleted\n");
                int errLen = strlen(errMsg);
                 sys_req(WRITE, DEFAULT_DEVICE, errMsg, &errLen);
             }
             else
             {
                 char msg[50];
                 strcpy(msg, "The desired PCB was deleted\n");
                 int msgLen = strlen(msg);
                 sys_req(WRITE, DEFAULT_DEVICE, msg, &msgLen);
             }
         }
     }
 }
```

5.27.1.4 resumePCB()

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

Definition at line 187 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Places a PCB in the not suspended state and reinserts it into the appropriate queue
    */

    PCB *PCBtoResume = findPCB(processName);

    if (PCBtoResume == NULL || strlen(processName) > 20)
    {
        char nameError[] = "This is not a valid name.\n";
        int printCount = strlen(nameError);
        sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
    }
    else
    {
        removePCB(PCBtoResume);
        PCBtoResume->suspendedStatus = 1;
        insertPCB(PCBtoResume);

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

5.27.1.5 setPCBPriority()

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

Definition at line 217 of file R2commands.c.

```
{ // ANASTASE WILL PROGRAM THIS FUNCTION

    // Sets a PCB's priority and reinserts the process into the correct place in the correct queue

    /*
    Error Checking:
    Name must be valid.
    newPriority
    */

    // find the process and validate the name
    PCB *tempPCB = findPCB(processName);

    if ((tempPCB != NULL) && (newProcessPriority >= 0) && (newProcessPriority < 10))
    {
        tempPCB->priority = newProcessPriority;
        removePCB(tempPCB);
        insertPCB(tempPCB);

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

5.27.1.6 showAll()

```
void showAll ( )
```

Definition at line 612 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Displays the following information for each PCB in the ready and blocked queues:
        Process Name
        Class
        State
        Suspended Status
        Priority
    */
    /*
    Error Checking:
    None
    */

    showReady();
    showSuspendedReady();
    showBlocked();
    showSuspendedBlocked();
}
```

5.27.1.7 showBlocked()

```
void showBlocked ( )
```

Definition at line 559 of file R2commands.c.

```
{ // ANASTASE WILL PROGRAM THIS FUNCTION
    /*
    Displays the following information for each PCB in the blocked queue:
        Process Name
        Class
        State
        Suspended Status
        Priority
        HEAD
    */
    /*
    Error Checking:
    None
    */

    // check

    char print_message[30] = "The blocked queue:\n";
    int message_size = strlen(print_message);
    sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);

    // printPCBs(blocked);
    queue *tempQueue = getBlocked();
    PCB *tempPtr = tempQueue->head; //PCB_container->head;
    int count = tempQueue->count;

    if (count == 0)
    {
        // the queue is empty
        char error_message[30] = "The queue is empty.\n";
        int error_size = strlen(error_message);
        sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
        return;
    }
    // The queue is not empty

    int value = 0;
    // Testing purpose
    //char print_message[38]="The blocke queue testing:\n";
    //int message_size=strlen(print_message);
    //sys_req(WRITE, DEFAULT_DEVICE, print_message, &message_size);

    while (value < count)
    { // testing for <== or <
        // Print out the process
        showPCB(tempPtr->processName);
606         // increment pcb*tempPtr, the loop variable.
607         tempPtr = tempPtr->nextPCB;
608         value++;
609     }
610 }
```

5.27.1.8 showPCB()

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

Definition at line 243 of file R2commands.c.

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

```c
                strcpy(sysMsg, "system");
                int sysMsgLen = strlen(sysMsg);
                sys_req(WRITE, DEFAULT_DEVICE, sysMsg, &sysMsgLen);
            }
            sys_req(WRITE, DEFAULT_DEVICE, newLine, &newLineLen);

            // Print out the PCB state

            if (PCB_to_show->runningStatus == 0)
            { // The process is ready.
                char stateMsg[50];
                strcpy(stateMsg, "The process is ready!\n");
                int stateMsgLen = strlen(stateMsg);
                sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
            }
            else if (PCB_to_show->runningStatus == -1)
            { // The process is blocked.
                char stateMsg[50];
                strcpy(stateMsg, "The process is blocked!\n");
                int stateMsgLen = strlen(stateMsg);
                sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
            }
            else if (PCB_to_show->runningStatus == 1)
            { // The process is running.
                char stateMsg[50];
                strcpy(stateMsg, "The process is running!\n");
                int stateMsgLen = strlen(stateMsg);
                sys_req(WRITE, DEFAULT_DEVICE, stateMsg, &stateMsgLen);
            }

            // Print out the PCB suspended status

            if (PCB_to_show->suspendedStatus == 0)
            { // The process is suspended
                char susMsg[50];
                strcpy(susMsg, "The process is suspended!\n");
                int susMsgLen = strlen(susMsg);
                sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
            }
            else if (PCB_to_show->suspendedStatus == 1)
            { // The process is not suspended
                char susMsg[50];
                strcpy(susMsg, "The process is not suspended!\n");
                int susMsgLen = strlen(susMsg);
                sys_req(WRITE, DEFAULT_DEVICE, susMsg, &susMsgLen);
            }

            // Print out the PCB priority
            char priorityMsg[50];
            int priorityMsgLen = 0;

            switch (PCB_to_show->priority)
            {
            case 0:
                strcpy(priorityMsg, "The process priority is 0!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 1:
                strcpy(priorityMsg, "The process priority is 1!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 2:
                strcpy(priorityMsg, "The process priority is 2!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 3:
                strcpy(priorityMsg, "The process priority is 3!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 4:
                strcpy(priorityMsg, "The process priority is 4!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 5:
                strcpy(priorityMsg, "The process priority is 5!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 6:
                strcpy(priorityMsg, "The process priority is 6!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 7:
                strcpy(priorityMsg, "The process priority is 7!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 8:
                strcpy(priorityMsg, "The process priority is 8!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            case 9:
                strcpy(priorityMsg, "The process priority is 9!\n");
                priorityMsgLen = strlen(priorityMsg);
                sys_req(WRITE, DEFAULT_DEVICE, priorityMsg, &priorityMsgLen);
                break;

            default:
                break;
            }
        }
    }
}
```

5.27.1.9 showReady()

```
void showReady ( )
```

Definition at line 430 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Displays the following information for each PCB in the ready queue:
        Process Name
        Class
        State
        Suspended Status
        Priority
    */
    /*
    Error Checking:
    None
    */

    char message[] = "Printing the ready queue:\n";
    int messLength = strlen(message);
    sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);

    queue *tempQueue = getReady();
    PCB *tempPCB = tempQueue->head;

    int loop = 0;
    int count = tempQueue->count;

    if (count == 0)
    {
        // the queue is empty
        char error_message[30] = "The queue is empty.\n";
        int error_size = strlen(error_message);
        sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
        return;
    }

    while (loop < count)
    {
        showPCB(tempPCB->processName);
        PCB *tempNext = tempPCB->nextPCB;
        loop++;
        tempPCB = tempNext;
    }
}
```

5.27.1.10 showSuspendedBlocked()

```
void showSuspendedBlocked ( )
```

Definition at line 516 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Displays the following information for each PCB in the suspended blocked queue:
        Process Name
        Class
        State
        Suspended Status
        Priority
    */
    /*
    Error Checking:
    None
    */

    char message[] = "Printing the suspended blocked queue:\n";
    int messLength = strlen(message);
    sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);

    queue *tempQueue = getSuspendedBlocked();
    PCB *tempPCB = tempQueue->head;

    int loop = 0;
    int count = tempQueue->count;

    if (count == 0)
    {
        // the queue is empty
        char error_message[30] = "The queue is empty.\n";
        int error_size = strlen(error_message);
        sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
        return;
    }

    while (loop < count)
    {
        showPCB(tempPCB->processName);
        PCB *tempNext = tempPCB->nextPCB;
        loop++;
        tempPCB = tempNext;
    }
}
```

5.27.1.11 showSuspendedReady()

```
void showSuspendedReady ( )
```

Definition at line 473 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Displays the following information for each PCB in the suspended ready queue:
        Process Name
        Class
        State
        Suspended Status
        Priority
    */
    /*
    Error Checking:
    None
    */

    char message[] = "Printing the suspended ready queue:\n";
    int messLength = strlen(message);
    sys_req(WRITE, DEFAULT_DEVICE, message, &messLength);

    queue *tempQueue = getSuspendedReady();
    PCB *tempPCB = tempQueue->head;

    int loop = 0;
    int count = tempQueue->count;

    if (count == 0)
    {
        // the queue is empty
        char error_message[30] = "The queue is empty.\n";
        int error_size = strlen(error_message);
        sys_req(WRITE, DEFAULT_DEVICE, error_message, &error_size);
        return;
    }

    while (loop < count)
    {
        showPCB(tempPCB->processName);
        PCB *tempNext = tempPCB->nextPCB;
        loop++;
        tempPCB = tempNext;
    }
}
```

5.27.1.12 suspendPCB()

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

Definition at line 157 of file R2commands.c.

```
{ // COLTON WILL PROGRAM THIS FUNCTION
    /*
    Places a PCB in the suspended state and reinserts it into the appropriate queue
    */

    PCB *PCBtoSuspend = findPCB(processName);

    if (PCBtoSuspend == NULL || strlen(processName) > 20)
    {
        char nameError[] = "This is not a valid name.\n";
        int printCount = strlen(nameError);
        sys_req(WRITE, DEFAULT_DEVICE, nameError, &printCount);
    }
    else
    {
        removePCB(PCBtoSuspend);
        PCBtoSuspend->suspendedStatus = 0;
        insertPCB(PCBtoSuspend);

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

5.27.1.13 unblockPCB()

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

Definition at line 131 of file R2commands.c.

```
{ // ANASTASE WILL PROGRAM THIS FUNCTION

    /*
    Places a PCB in the unblocked state and reinserts it into the appropriate queue.
    */
    /*
    Error Checking:
    Name must be valid.

    */

    PCB *pcb_to_unblock = findPCB(processName);
    if (pcb_to_unblock != NULL)
    {
        pcb_to_unblock->runningStatus = 0; // ready
        removePCB(pcb_to_unblock);         // is this the right place to put that function?
        insertPCB(pcb_to_unblock);

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

5.28 README.md File Reference