RUNTIME TERROR OS

R5

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 Alarm Struct Reference

Public Attributes

- int **hour**
- int **minute**
- int **second**
- char **message** [85]
- struct Alarm ∗ **next**
- struct Alarm ∗ **prev**

3.1.1 Detailed Description

Definition at line 15 of file userFunctions.h.

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

- modules/R1/userFunctions.h

3.2 CMCB Struct Reference

Public Attributes

- u32int **size**
- struct CMCB ∗ **prev**
- struct CMCB ∗ **next**
- char **Process_name** [10]
- u32int **address**
- int **MEMState**

3.2.1 Detailed Description

Definition at line 4 of file MCB.h.

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

- modules/R5/MCB.h

3.3 context Struct Reference

Public Attributes

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

3.3.1 Detailed Description

Definition at line 34 of file PCB.h.

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

- modules/R2/PCB.h

3.4 date_time Struct Reference

Public Attributes

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

3.4.1 Detailed Description

Definition at line 32 of file system.h.

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

- include/system.h

3.5 footer Struct Reference

Public Attributes

- header **head**

3.5.1 Detailed Description

Definition at line 18 of file heap.h.

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

- include/mem/heap.h

3.6 gdt_descriptor_struct Struct Reference

Public Attributes

- u16int **limit**
- u32int **base**

3.6.1 Detailed Description

Definition at line 25 of file tables.h.

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

- include/core/tables.h

3.7 gdt_entry_struct Struct Reference

Public Attributes

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

3.7.1 Detailed Description

Definition at line 32 of file tables.h.

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

- include/core/tables.h

3.8 header Struct Reference

Public Attributes

- int **size**
- int **index_id**

3.8.1 Detailed Description

Definition at line 13 of file heap.h.

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

- include/mem/heap.h

3.9 heap Struct Reference

Public Attributes

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

3.9.1 Detailed Description

Definition at line 35 of file heap.h.

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

- include/mem/heap.h

3.10 idt_entry_struct Struct Reference

Public Attributes

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

3.10.1 Detailed Description

Definition at line 8 of file tables.h.

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

- include/core/tables.h

3.11 idt_struct Struct Reference

Public Attributes

- u16int **limit**
- u32int **base**

3.11.1 Detailed Description

Definition at line 18 of file tables.h.

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

- include/core/tables.h

3.12 index_entry Struct Reference

Public Attributes

- int **size**
- int **empty**
- u32int **block**

3.12.1 Detailed Description

Definition at line 22 of file heap.h.

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

- include/mem/heap.h

3.13 index_table Struct Reference

Public Attributes

- index_entry **table** [TABLE_SIZE]
- int **id**

3.13.1 Detailed Description

Definition at line 29 of file heap.h.

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

- include/mem/heap.h

3.14 List Struct Reference

Public Attributes

- Alarm ∗ **head**
- Alarm ∗ **tail**

3.14.1 Detailed Description

Definition at line 24 of file userFunctions.h.

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

- modules/R1/userFunctions.h

3.15 MemList Struct Reference

Public Attributes

- CMCB ∗ **head**

3.15.1 Detailed Description

Definition at line 18 of file MCB.h.

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

- modules/R5/MCB.h

3.16 page_dir Struct Reference

Public Attributes

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

3.16.1 Detailed Description

Definition at line 36 of file paging.h.

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

- include/mem/paging.h

3.17 page_entry Struct Reference

Public Attributes

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

3.17.1 Detailed Description

Definition at line 14 of file paging.h.

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

- include/mem/paging.h

3.18 page_table Struct Reference

Public Attributes

- page_entry **pages** [1024]

3.18.1 Detailed Description

Definition at line 28 of file paging.h.

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

- include/mem/paging.h

3.19 param Struct Reference

Public Attributes

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

3.19.1 Detailed Description

Definition at line 34 of file mpx_supt.h.

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

- modules/mpx_supt.h

3.20 PCB Struct Reference

Public Attributes

- unsigned char **stack** [MEM1K]
- unsigned char ∗ **stackTop**
- struct PCB ∗ **prev**
- struct PCB ∗ **next**
- char **Process_Name** [10]
- int **Process_Class**
- int **Priority**
- int **ReadyState**
- int **SuspendedState**

3.20.1 Detailed Description

Definition at line 15 of file PCB.h.

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

- modules/R2/PCB.h

3.21 Queue Struct Reference

Public Attributes

- int **count**
- PCB ∗ **head**
- PCB ∗ **tail**

3.21.1 Detailed Description

Definition at line 27 of file PCB.h.

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

- modules/R2/PCB.h

Chapter 4

File Documentation

4.1 include/core/asm.h File Reference

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

4.2 include/core/interrupts.h File Reference

Functions

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

4.3 include/core/io.h File Reference

Macros

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

4.3.1 Macro Definition Documentation

4.3.1.1 inb

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

Value:

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

Definition at line 17 of file io.h.

4.4 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)

4.5 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**

4.6 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)

4.7 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)

4.8 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)

4.8.1 Function Documentation

4.8.1.1 atoi()

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

Description: Convert an ASCII string to an integer

Parameters

s	String

Definition at line 50 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
}
```

4.8.1.2 isspace()

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

Description: Determine if a character is whitespace.

Parameters

c	character to check

Definition at line 121 of file string.c.

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

4.8.1.3 memset()

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

Description: Set a region of memory.

Parameters

s	destination
c	byte to write
n	count

Definition at line 139 of file string.c.

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

4.8.1.4 strcat()

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

Description: Concatenate the contents of one string onto another.

Parameters

s1	destination
s2	source

Definition at line 108 of file string.c.

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

4.8.1.5 strcmp()

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

Description: String comparison

Parameters

s1	string 1
s2	string 2

Definition at line 81 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 );
}
```

4.8.1.6 strcpy()

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

Description: Copy one string to another.

Parameters

s1	destination
s2	source

Definition at line 38 of file string.c.

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

4.8.1.7 strlen()

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

Description: Returns the length of a string.

Parameters

s	input string

Definition at line 26 of file string.c.

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

4.8.1.8 strtok()

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

Description: Split string into tokens

Parameters

s1	String
s2	delimiter

Definition at line 153 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;
}
```

4.9 include/system.h File Reference

Classes

- struct date_time

Macros

- #define **NULL** 0
- #define **no_warn**(p) if (p) while (1) break
- #define **asm** __asm__
- #define **volatile** __volatile__
- #define **sti**() asm volatile ("sti"::)
- #define **cli**() asm volatile ("cli"::)
- #define **nop**() asm volatile ("nop"::)
- #define **hlt**() asm volatile ("hlt"::)
- #define **iret**() asm volatile ("iret"::)
- #define **GDT_CS_ID** 0x01
- #define **GDT_DS_ID** 0x02

Typedefs

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

Functions

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

4.10 kernel/core/interrupts.c File Reference

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

Macros

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

Functions

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

- void **do_invalid_tss** ()
- void **do_segment_not_present** ()
- void **do_stack_segment** ()
- void **do_general_protection** ()
- void **do_page_fault** ()
- void **do_reserved** ()
- void **do_coprocessor** ()

Variables

- idt_entry **idt_entries** [256]

4.11 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/comHand.h"
#include "modules/sys_proc_loader.h"
#include "modules/R1/userFunctions.h"
#include "modules/R5/MCB.h"
```

Functions

- void **kmain** (void)

4.12 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 ∗cmdBuffer, int ∗count)

Variables

- int **serial_port_out** = 0
- int **serial_port_in** = 0

4.13 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)

4.14 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]

4.15 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

4.16 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**

4.17 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)

4.17.1 Function Documentation

4.17.1.1 atoi()

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

Description: Convert an ASCII string to an integer

Parameters

s	String

Definition at line 50 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
}
```

4.17.1.2 isspace()

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

Description: Determine if a character is whitespace.

Parameters

c	character to check

Definition at line 121 of file string.c.

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

4.17.1.3 memset()

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

Description: Set a region of memory.

Parameters

s	destination
c	byte to write
n	count

Definition at line 139 of file string.c.

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

4.17.1.4 strcat()

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

Description: Concatenate the contents of one string onto another.

Parameters

s1	destination
s2	source

Definition at line 108 of file string.c.

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

4.17.1.5 strcmp()

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

Description: String comparison

Parameters

s1	string 1
s2	string 2

Definition at line 81 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 );
}
```

4.17.1.6 strcpy()

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

Description: Copy one string to another.

Parameters

s1	destination
s2	source

Definition at line 38 of file string.c.

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

4.17.1.7 strlen()

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

Description: Returns the length of a string.

Parameters

s	input string

Definition at line 26 of file string.c.

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

4.17.1.8 strtok()

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

Description: Split string into tokens

Parameters

s1	String
s2	delimiter

Definition at line 153 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;
}
```

4.18 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** ()
- u32int ∗ **sys_call** (context ∗registers)

Variables

- param **params**
- int **current_module** = -1
- u32int(∗ **student_malloc**)(u32int)
- int(∗ **student_free**)(void ∗)
- PCB ∗ **cop**
- context ∗ **initial**

4.19 modules/mpx_supt.h File Reference

```
#include <system.h>
#include "R2/PCB.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** ()
- u32int ∗ **sys_call** (context ∗registers)

4.20 modules/R1/comHand.h File Reference

Functions

- int comHand ()

4.20.1 Function Documentation

4.20.1.1 comHand()

```
int comHand ( )
```

Description: Interprets user input to call the appropriate user functions.

Definition at line 23 of file comHand.c.

```
                  {

        Help("\0");

        char cmdBuffer[100];
        int bufferSize = 99;
        int quit = 0;
        int shutdown = 0;

        while(quit != 1)    {
            memset(cmdBuffer, '\0', 100);
          sys_req(READ, DEFAULT_DEVICE, cmdBuffer, &bufferSize);
            char* FirstToken = strtok(cmdBuffer, "-");
            char* SecondToken = strtok(NULL, "-");
            char* ThirdToken = strtok(NULL, "-");
            char* FourthToken = strtok(NULL, "-");
            char* FifthToken = strtok(NULL, "-");
            if(shutdown == 0)   {
/************************************************************************************
                R1 comHand
************************************************************************************/
                    if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,NULL) == 0)     {
                        Help("\0");
                    }
                    //R1 Commands
                    else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"version") == 0 &&
        strcmp(ThirdToken,NULL) == 0) {
                        Help("Version");
                    }
                    else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"getDate") == 0 &&
        strcmp(ThirdToken,NULL) == 0) {
                        Help("GetDate");
                    }
                    else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"setDate") == 0 &&
        strcmp(ThirdToken,NULL) == 0) {
```

```
                    Help("SetDate");
                }
                else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"getTime") == 0 && strcmp(ThirdToken,NULL) == 0) {
                    Help("GetTime");
                }
                else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"setTime") == 0 && strcmp(ThirdToken,NULL) == 0) {
                    Help("SetTime");
                }
                // R2 Commands
                else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"suspend") == 0 && strcmp(ThirdToken,NULL) == 0) {
                    Help("suspend");
                }
                else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"resume") == 0 && strcmp(ThirdToken,NULL) == 0)  {
                    Help("resume");
                }
                else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"setPriority") == 0 && strcmp(ThirdToken,NULL) == 0) {
                    Help("setPriority");
                }
                else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showPCB") == 0 && strcmp(ThirdToken,NULL) == 0) {
                    Help("showPCB");
                }
                else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showAll") == 0 && strcmp(ThirdToken,NULL) == 0) {
                    Help("showAll");
                }
                else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showReady") == 0 && strcmp(ThirdToken,NULL) == 0)    {
                    Help("showReady");
                }
                else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"showBlocked") == 0 && strcmp(ThirdToken,NULL) == 0) {
                    Help("showBlocked");
                }
                // Temporary R2 commands
                // else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"createPCB") == 0 && strcmp(ThirdToken,NULL) == 0)   {
                //  Help("createPCB");
                // }
                else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"deletePCB") == 0 && strcmp(ThirdToken,NULL) == 0)    {
                    Help("deletePCB");
                }
                // else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"block") == 0 && strcmp(ThirdToken,NULL) == 0)   {
                //  Help("block");
                // }
                // else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"unblock") == 0 && strcmp(ThirdToken,NULL) == 0)     {
                //  Help("unblock");
                // }
                else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"shutdown") == 0 && strcmp(ThirdToken,NULL) == 0)     {
                    Help("shutdown");
                 }
                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"infinite") == 0 && strcmp(ThirdToken,NULL) == 0)     {
                     Help("infinte");
                 }
                 // R4 Commands
                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"loadr3") == 0 && strcmp(ThirdToken,NULL) == 0)   {
                     Help("loadr3");
                 }
                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"alarm") == 0 && strcmp(ThirdToken,NULL) == 0)    {
                     Help("alarm");
                 }
                 // Bonus Command
                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"clear") == 0 && strcmp(ThirdToken,NULL) == 0)    {
                     Help("clear");
                 }
                 // Temporary R5 Commands
                 // else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"clear") == 0 && strcmp(ThirdToken,NULL) == 0)   {
                 //  Help("heap");
                 // }
                 // else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"clear") == 0 && strcmp(ThirdToken,NULL) == 0)   {
                 //  Help("alloc");
                 // }
                 // else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"clear") == 0 && strcmp(ThirdToken,NULL) == 0)  {
                 //  Help("free");
                 // }
                 // else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"clear") == 0 && strcmp(ThirdToken,NULL) == 0)  {
                 //  Help("empty");
                 // }
                 // R5 Commands
                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"clear") == 0 && strcmp(ThirdToken,NULL) == 0)   {
                     Help("showFree");
                 }
                 else if(strcmp(FirstToken,"help") == 0 && strcmp(SecondToken,"clear") == 0 && strcmp(ThirdToken,NULL) == 0)   {
                     Help("showAlloc");
                 }






                 else if(strcmp(FirstToken,"version") == 0 && strcmp(SecondToken,NULL) == 0)
                     Version();
                 else if(strcmp(FirstToken,"clear") == 0 && strcmp(SecondToken,NULL) == 0)
                     clear();

                 else if(strcmp(FirstToken,"getDate") == 0 && strcmp(SecondToken,NULL) == 0)
                     GetDate();

                 else if(strcmp(FirstToken,"setDate") == 0){
                     if (EdgeCase(SecondToken) == 1 && EdgeCase(ThirdToken) == 1 && EdgeCase(FourthToken) == 1 && EdgeCase(FifthToken) == 1)    {
                         SetDate(atoi(SecondToken), atoi(ThirdToken), atoi(FourthToken), atoi(FifthToken));
                     }
                     else
                         printf("\x1b[31m""\nERROR: Invalid parameters for setDate \n""\x1b[0m");
                 }
                 else if(strcmp(FirstToken,"getTime") == 0 && strcmp(SecondToken,NULL) == 0) //Return the current time held by the registers.
                     GetTime();
                 else if(strcmp(FirstToken,"setTime") == 0 && strcmp(FifthToken,NULL) == 0){
                     if (EdgeCase(SecondToken) == 1 && EdgeCase(ThirdToken) == 1 && EdgeCase(FourthToken) == 1)    {
                             SetTime(atoi(SecondToken), atoi(ThirdToken), atoi(FourthToken)); //input as Hour-Minute-Seconds
                     }
                     else
                         printf("\x1b[31m""\nERROR: Invalid parameters for setTime \n""\x1b[0m");
                 }






 /**************************************************************************************
                 R2 comHand
 **************************************************************************************/
                 else if(strcmp(FirstToken,"suspend") == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     Suspend(SecondToken);
                 }
                 else if(strcmp(FirstToken,"resume") == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     Resume(SecondToken);
                 }
                 else if(strcmp(FirstToken,"setPriority") == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     if(EdgeCase(ThirdToken) == 1)    {
                         Set_Priority(SecondToken, atoi(ThirdToken));    //input as setPriority-Process_Name-Priority
                     }
                     else
                         printf("\x1b[31m""\nERROR: Invalid parameters for setPriority, priority must be entered as a integer. \n""\x1b[0m");
                 }
                 else if(strcmp(FirstToken,"showPCB") == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     Show_PCB(SecondToken);
                     printf("\n");
                 }
                 else if(strcmp(FirstToken,"showAll") == 0 && strcmp(SecondToken,NULL) == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     Show_All();
                     printf("\n");
                 }
                 else if(strcmp(FirstToken,"showReady") == 0 && strcmp(SecondToken,NULL) == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     Show_Ready();
                     printf("\n");
                 }
                 else if(strcmp(FirstToken,"showBlocked") == 0 && strcmp(SecondToken,NULL) == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     Show_Blocked();
                     printf("\n");
                 }



                 /********** R2 Temp Commands **********/
                 //Removed from active for R3/R4
                 /*
                 else if(strcmp(FirstToken,"createPCB") == 0) {
                     if( strlen(SecondToken) < 11)   {
                             Create_PCB(SecondToken, atoi(ThirdToken), atoi(FourthToken)); //input as Process_Name-Priority-Class
                     }
                     else
                         printf("\x1b[31m""\nERROR: Invalid parameters for createPCB, Process_name must only contain 10 or fewer characters. \n""\x1b[0m");
                 }
                 */
                 else if(strcmp(FirstToken,"deletePCB") == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     Delete_PCB(SecondToken);
                 }



                 //Removed from active for R3/R4
                 /*
                 else if(strcmp(FirstToken,"block") == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     Block(SecondToken);
                 }
                 else if(strcmp(FirstToken,"unblock") == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     Unblock(SecondToken);
                 }
                 */
 /*************************************************************************************
                 R3 comHand
 ************************************************************************************/
                 //Removed for R4
                 /*
                 else if(strcmp(FirstToken,"yield") == 0 && strcmp(SecondToken,NULL) == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     yield();
                     printf("\n");
                 }
                 else if(strcmp(FirstToken,"loadr3") == 0 && strcmp(SecondToken,NULL) == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     loader();
                     printf("\n");
                 }
                 */
 /*************************************************************************************
                 R4 comHand
 ************************************************************************************/
                 else if(strcmp(FirstToken,"alarm") == 0) {
                     if (EdgeCase(ThirdToken) == 1 && EdgeCase(FourthToken) == 1 && EdgeCase(FifthToken) == 1)    {
                         if (atoi(ThirdToken) < 24 && atoi(FourthToken) < 60 && atoi(FifthToken) < 60)    {
                             loaderalarm(SecondToken, atoi(ThirdToken), atoi(FourthToken), atoi(FifthToken));
                             printf("\n");   //input as Message-Hour-Minute-Seconds
                         }
                         else
                         printf("\x1b[31m""\nERROR: Invalid parameters for alarm, must be a valid time \n""\x1b[0m");
                     }
                     else
                         printf("\x1b[31m""\nERROR: Invalid parameters for alarm \n""\x1b[0m");

                 }
                 else if(strcmp(FirstToken,"loadr3") == 0 && strcmp(SecondToken,NULL) == 0 && strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     loader();
                     printf("\n");
                 }
                 else if(strcmp(FirstToken,"infinite") == 0 && strcmp(SecondToken,NULL) == 0 &&
         strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     loaderinfinite();
                     printf("\n");
                 }

 /*************************************************************************************
                 R5 comHand
 *************************************************************************************/
                 // else if(strcmp(FirstToken,"heap") == 0 && strcmp(ThirdToken,NULL) == 0 &&
         strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                 //  Init_Heap(atoi(SecondToken));
                 // }
                 // else if(strcmp(FirstToken,"alloc") == 0 && strcmp(ThirdToken,NULL) == 0 &&
         strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                 //  Alloc_Mem(atoi(SecondToken));
                 // }
                 // else if(strcmp(FirstToken,"free") == 0 && strcmp(ThirdToken,NULL) == 0 &&
         strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                 //  Free_Mem(atoi(SecondToken));
                 // }
                 // else if(strcmp(FirstToken,"empty") == 0 && strcmp(SecondToken,NULL) == 0 &&
         strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                 //  IsEmpty();
                 // }
                 else if(strcmp(FirstToken,"showFree") == 0 && strcmp(SecondToken,NULL) == 0 &&
         strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     ShowFree();
                 }
                 else if(strcmp(FirstToken,"showAlloc") == 0 && strcmp(SecondToken,NULL) == 0 &&
         strcmp(ThirdToken,NULL) == 0 && strcmp(FourthToken,NULL) == 0 && strcmp(FifthToken,NULL) == 0) {
                     ShowAlloc();
                 }

 /*************************************************************************************
                 shutdown comHand
 *************************************************************************************/
                 else if(strcmp(FirstToken,"shutdown") == 0 && strcmp(SecondToken,NULL) == 0){
                     printf("\x1b[33m""\nAre you sure you want to shutdown? [yes/no]\n""\x1b[0m");
                     shutdown = 1;
                 }
                 else    {
                     printf("\x1b[31m""\nERROR: Not a valid command \n""\x1b[0m");
                 }
             }
             else{
                 if(strcmp(FirstToken,"yes") == 0 && shutdown == 1)    {
                     quit = 1;
                 }
                 else if(strcmp(FirstToken,"no") == 0){
                     printf("\x1b[33m""\nShutdown Cancelled\x1b[0m \n");
                     shutdown = 0;
                 }
                 else
                     printf("\x1b[31m""\nERROR: Please enter \"yes\" or \"no\" \n""\x1b[0m");
             }
             sys_req(IDLE, DEFAULT_DEVICE, NULL, NULL);
     }
     getReady() -> head = NULL;
     sys_req(EXIT, DEFAULT_DEVICE, NULL, NULL);
     return 0;   //shutdown procedure
 }
```

4.21 modules/R1/userFunctions.c File Reference

```
#include <stdint.h>
#include <string.h>
#include <system.h>
#include <core/serial.h>
#include <core/io.h>
#include "../mpx_supt.h"
#include "userFunctions.h"
#include "../procsr3.h"
#include "../sys_proc_loader.h"
```

Functions

- void **clear** ()
- char ∗ itoa (int num)
- int BCDtoDec (int BCD)
- int DectoBCD (int Decimal)
- void **printf** (char msg[ ])
- int EdgeCase (char ∗pointer)
- void SetTime (int hours, int minutes, int seconds)
- void GetTime ()
- void SetDate (int day, int month, int millennium, int year)
- void GetDate ()
- void Version ()
- char toLowercase (char c)
- void Help (char ∗request)
- else **if** (strcmp(request,"showFree")==0)
- else **if** (strcmp(request,"showAlloc")==0)
- void Suspend (char ∗ProcessName)
- void Resume (char ∗ProcessName)
- void Set_Priority (char ∗ProcessName, int Priority)
- void Show_PCB (char ∗ProcessName)
- void Show_All ()
- void Show_Ready ()
- void Show_Blocked ()
- void Create_PCB (char ∗ProcessName, int Priority, int Class)
- void Delete_PCB (char ∗ProcessName)
- void Block (char ∗ProcessName)
- void Unblock (char ∗ProcessName)
- void **loader** ()
- void **loadr3** (char ∗name, u32int func)
- void **yield** ()
- void **loaderinfinite** ()
- List ∗ **getList** ()
- void **loaderalarm** (char text[ ], int hours, int minutes, int seconds)

Variables

- **else**
- List **AlarmList**

4.21.1 Function Documentation

4.21.1.1 BCDtoDec()

```
int BCDtoDec (
            int BCD )
```

Description: Changes binary number to decimal numbers.

Parameters

value	Binary number to be changed to decimal

Definition at line 81 of file userFunctions.c.

```
                        {
    return (((BCD»4)*10) + (BCD & 0xF));
}
```

4.21.1.2 Block()

```
void Block (
            char * ProcessName )
```

Brief Description: Places a PCD in the blocked state and reinserts it into the correct queue.

Description: Can except a string as a pointer that is the Process Name. The specified PCB will be places in a blocked state and reinserted into the appropriate queue. An error check for a valid name occurs.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 957 of file userFunctions.c.

```
                               {
  PCB* pcb = FindPCB(ProcessName);
  if (pcb == NULL)  {
    printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
  }
  else {
    if(pcb->ReadyState == BLOCKED)  {
        printf("\x1b[32m""\nThis Process is already BLOCKED \n""\x1b[0m");
    }
    else    {
      RemovePCB(pcb);
      pcb->ReadyState = BLOCKED;
      InsertPCB(pcb);
    }
  }
}
```

4.21.1.3 Create_PCB()

```
void Create_PCB (
            char * ProcessName,
            int Priority,
            int Class )
```

Brief Description: Calls SetupPCB() and inserts PCB into appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Can accept two integers, Priority and Class. SetupPCB() will be called and the PCB will be inserted into the appropriate queue. An error check for unique and valid Process Name, an error check for valid process class, and an error check for process priority.

Parameters

Process_Name	Character pointer that matches the name of process.
Priority	integer that matches the priority number.
Class	integer that matches the class number.

Definition at line 900 of file userFunctions.c.

```
                                                                  {
  if (FindPCB(ProcessName) == NULL) {
    if(Priority >= 0 && Priority < 10){
      if(Class == 0 || Class == 1){
        PCB* pcb = SetupPCB(ProcessName, Class, Priority);
        InsertPCB(pcb);
      } else{
        printf("\x1b[31m""\nERROR: Not a valid Class \n""\x1b[0m");
      }
    } else{
      printf("\x1b[31m""\nERROR: Not a valid Priority \n""\x1b[0m");
    }
  } else{
    printf("\x1b[31m""\nERROR: This Process Name already exists \n""\x1b[0m");
  }
}
```

4.21.1.4 DectoBCD()

```
int DectoBCD (
              int Decimal )
```

Description: Changes decimal numbers to binary numbers.

Parameters

Decimal	Decimal number to be changed to binary

Definition at line 88 of file userFunctions.c.

```
                             {
    return (((Decimal/10) « 4) | (Decimal % 10));
}
```

4.21.1.5 Delete_PCB()

```
void Delete_PCB (
              char * ProcessName )
```

Brief Description: Removes PCB from appropriate queue and frees all associated memory.

Description: Can except a string as a pointer that is the Process Name. Removes PCB from the appropriate queue and then frees all associated memory. An error check to make sure process name is valid.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 926 of file userFunctions.c.

```
                                   {
  PCB* pcb = FindPCB(ProcessName);
  if (pcb == NULL)  {
    printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
  }
  else if(strcmp(pcb->Process_Name,"InfProc") == 0)   {
        if(pcb->SuspendedState == YES)  {
            RemovePCB(pcb);
            FreePCB(pcb);
        }
        else
            printf("\x1b[31m""\nERROR:This process cannot be deleted unless it is in the suspended
 state\n""\x1b[0m");
  }
  else if(pcb -> Process_Class == SYSTEM)   {
      printf("\x1b[31m""\nERROR: System Processes cannot be deleted from the system. \n""\x1b[0m");
  }
    else {
        RemovePCB(pcb);
        FreePCB(pcb);
    }
}
```

4.21.1.6 EdgeCase()

```
int EdgeCase (
               char * pointer )
```

Description: Compares pointer char to validate if it is a number or not.

Parameters

Compares	pointer char to validate if it is a number or not.

Definition at line 109 of file userFunctions.c.

```
                              {
  int valid = 0;
  if (strcmp(pointer, "00") == 0) {
    valid = 1;
    return valid;
  }
  else if (strcmp(pointer, "0") == 0) {
    valid = 1;
    return valid;
  }
  else  {
    int j;
    valid = 0;
    for(j = 0; j <= 99; j++)  {
      if(strcmp(pointer,itoa(j)) == 0)
        valid = 1;
    }
    if(valid == 0)  {
      return valid;
    }
  }
  return valid;
}
```

4.21.1.7 GetDate()

```
void GetDate ( )
```

Description: Returns the full date back to the user in decimal form.

No parameters.

Definition at line 271 of file userFunctions.c.
```
                   {

   outb(0x70,0x07);
     unsigned char day = BCDtoDec(inb(0x71));
     outb(0x70,0x08);
     unsigned char month = BCDtoDec(inb(0x71));
     outb(0x70,0x32);
     unsigned char millennium = BCDtoDec(inb(0x71));
     char msg[2] = "-";
     char msg3[10] = "Date: ";
     printf(msg3);

      printf(itoa(day));
     //sys_req(WRITE, COM1, itoa(day), &check);
     printf(msg);
     printf(itoa(month));
     //sys_req(WRITE, COM1, itoa(month), &check);
     printf(msg);
     printf(itoa(millennium));
     //sys_req(WRITE, COM1, itoa(millennium), &check);
   outb(0x70,0x09);
   if(BCDtoDec(inb(0x71)) == 0){
      printf("00");
     //sys_req(WRITE, COM1, "00", &check);
   }
   else {
         unsigned char year = BCDtoDec(inb(0x71));
          printf(itoa(year));
         //sys_req(WRITE, COM1, itoa(year), &check);
   }
     printf("\n");
 }
```

4.21.1.8 GetTime()

`void GetTime ( )`

Description: retrieve and return the time values for hours, minutes, and seconds form the clock register using inb(Port,address).

No parameters.

Definition at line 190 of file userFunctions.c.
```
                   {

     int hour;
     int minute;
     int second;
     outb(0x70,0x04);
     unsigned char hours = inb(0x71);
     outb(0x70,0x02);
     unsigned char minutes = inb(0x71);
     outb(0x70,0x00);
     unsigned char seconds = inb(0x71);
     char msg1[2] = ":";
     char msg2[10] = "Time: ";
     printf(msg2);
     hour = BCDtoDec(hours);
     printf(itoa(hour));
     //sys_req(WRITE, COM1, itoa(hour), &check);
     printf(msg1);
     minute = BCDtoDec(minutes);
     printf(itoa(minute));
     //sys_req(WRITE, COM1, itoa(minute), &check);
     printf(msg1);
     second = BCDtoDec(seconds);
     printf(itoa(second));
     //sys_req(WRITE, COM1, itoa(second), &check);
   printf("\n");
 }
```

4.21.1.9 Help()

```
void Help (
            char * request )
```

Brief Description: Gives helpful information for one of the functions

Description: Can except a string as a pointer, if the pointer is null then the function will print a complete list of avaliable commands to the console. If the pointer is a avaliable commands then instructions on how to use the command will be printed. If the command does not exist then a message explaining that it is not a valid command will be displayed.

Parameters

request	Character pointer that matches the name of the function that you need help with.

Definition at line 331 of file userFunctions.c.

```
                             {
    if (request[0] == '\0') {
        //removed for R3/R4 from active command list
        //\n createPCB \n block \n unblock
        //\n heap         alloc \n free         empty
        printf("\n to chain commands and parameters, please use \"-\" between keywords \n");
        printf("\n getDate      setDate \n getTime      setTime \n version      suspend \n resume
      setPriority \n showPCB      showAll \n showReady    showBlocked  \n deletePCB    shutdown \n alarm
         clear \n loadr3       infinte \n showFree     showAlloc \n\n");
    }
    else if (strcmp(request, "GetDate") == 0) {
        printf("\n getDate returns the current date that is loaded onto the operating system.\n");
    }
    else if (strcmp(request, "SetDate") == 0) {
        printf("\n setDate allows the user to reset the correct date into the system, as follows
      setDate-"BLU"day"RESET"-"BLU"month"RESET"-"BLU"year"RESET".\n Time must be inputed as a two digit
      number, Example 02 or 00");
    }
    else if (strcmp(request, "GetTime") == 0) {
        printf("\n getTime returns the current time as hours, minutes, seconds that is loaded onto the
      operating system.\n");
    }
    else if (strcmp(request, "SetTime") == 0) {
        printf("\n setTime allows the user to reset the correct time into the system, as follows
      setTime-"BLU"hour"RESET"-"BLU"minute"RESET"-"BLU"second"RESET".\n Time must be inputed as a two digit
      number, Example 02 or 00");
    }
    else if (strcmp(request, "Version") == 0) {
        printf("\n version returns the current operating software version that the system is
      running.\n");
    }
    else if (strcmp(request, "infinte") == 0) {
        printf("\n infinite Loads the infinite process into the ready queue.\n");
    }
    else if (strcmp(request, "loadr3") == 0)  {
        printf("\n loadr3 Loads in all five of the R3 test processes in a suspended state into the
      queue.\n");
    }
    else if (strcmp(request, "alarm") == 0)       {
        printf("\n alarm creates a user specified alarm with a user set message and time
      alarm-MSG-hour-minute-second.\n");
    }
    else if (strcmp(request, "clear") == 0)       {
        printf("\n clear erases the console of all typed commands and refreshes it with just the command
      list.\n");
    }

  else if(strcmp(request, "shutdown") == 0)   {
    printf("\n shutdown shuts down the system.\n");
  }



      /*****************************************************************************************************
              R2 Commands

      *****************************************************************************************************/
  else if(strcmp(request,"suspend") == 0) {
        printf("\n Suspend takes in the name of a PCB (suspend-NAME) then places it into the suspended
      state and reinserts it into the correct queue.\n");
```

```
}
else if(strcmp(request,"resume") == 0) {
      printf("\n Resume takes in the name of a PCB (resume-NAME) then removes it from the suspended
 state and adds it to the correct queue.\n");
}
else if(strcmp(request,"setPriority") == 0) {
      printf("\n SetPriority takes in the name of a PCB and the priority (setPrioriry-NAME-PRIORITY)
 it needs to be set to then reinstates the specified PCB into a new location by priority.\n");
}
else if(strcmp(request,"showPCB") == 0) {
      printf("\n ShowPCB takes in the name of a PCB and returns all the associated attributes to the
 user.\n");
}
else if(strcmp(request,"showAll") == 0) {
      printf("\n ShowAll takes no parameters but returns all PCB's that are currently in any of the
 queues.\n");
}
else if(strcmp(request,"showReady") == 0) {
      printf("\n ShowReady takes in no parameters but returns all PCB's and there attributes that
 currently are in the ready state.\n");
}
else if(strcmp(request,"showBlocked") == 0) {
      printf("\n ShowBlocked takes in no parameters but returns all PCB's and there attributes that
 currently are in the blocked state.\n");
}

/********************************** R2 Temp Commands
 *******************************************************/
else if(strcmp(request,"deletePCB") == 0) {
      printf("\n DeletePCB takes in the process_name (deletePCB-NAME) then deletes it from the queue
 and free's all the memory that was previously allocated to the specified PCB.\n");
}
//removed for R3/R4 from active command list
/*
else if(strcmp(request,"createPCB") == 0) {
      printf("\n CreatePCB takes in the process_name, process_class, and
 process_priority.(createPCB-NAME-PRIORITY-CLASS) Then assigns this new process into the correct
 queue.\n");
}
else if(strcmp(request,"block") == 0) {
      printf("\n Block takes in the process_name (block-NAME) then sets it's state to blocked and
 reinserts it back into the correct queue.\n");
}
else if(strcmp(request,"unblock") == 0) {
      printf("\n Unblock takes in the process_name (unblock-NAME) then sets it's state to ready and
 reinserts it back into the correct queue.\n");
}
*/


/********************************** R5 Temp Commands
 *******************************************************/
// else if(strcmp(request,"heap") == 0) {
  //  printf("\n heap initializes the memory heap for the entire system.\n");
// }
// else if(strcmp(request,"alloc") == 0) {
  //  printf("\n alloc allocates the specified amount of memory to the specific process
 (alloc-process_name-size).\n");
// }
// else if(strcmp(request,"free") == 0) {
  //  printf("\n free frees the specified memory at the address given (free-address).\n");
// }
// else if(strcmp(request,"empty") == 0) {
  //  printf("\n isempty returns true or false depending on if the heap has allocated memory.\n");
}
```

4.21.1.10 itoa()

```
char* itoa (
            int num )
```

Description: An integer is taken and seperated into individual chars and then all placed into a character array. Adapted from geeksforgeeks.org.

Parameters

num	integer to be put into array Title: itoa Author: Neha Mahajan Date: 29 May, 2017 Availability: https://www.geeksforgeeks.org/implement-itoa/

Definition at line 50 of file userFunctions.c.

```
                    {

    int i,j,k,count;
    i = num;
    j = 0;
    count = 0;
    while(i){ // count number of digits
        count++;
        i /= 10;
    }

    char* arr1;
    char arr2[count];
    arr1 = (char*)sys_alloc_mem(count); //memory allocation

    while(num){ // seperate last digit from number and add ASCII
        arr2[++j] = num%10 + '0';
        num /= 10;
    }

    for(k = 0; k < j; k++){ // reverse array results
        arr1[k] = arr2[j-k];
    }
    arr1[k] = '\0';

    return(char*)arr1;
}
```

4.21.1.11 Resume()

```
void Resume (
            char * ProcessName )
```

Brief Description: Places a PCD in the not suspended state and reinserts it into the appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Places a PCB in the not suspended state and reinserts it into the appropriate queue. An error check for valid Process Name.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 483 of file userFunctions.c.

```
                                  {
    PCB* pcb = FindPCB(ProcessName);
    if (pcb == NULL)    {
      printf(RED"\nERROR: Not a valid process name \n"RESET);
    }
    else {
        if(pcb->SuspendedState == NO)   {
            printf(GRN"\nThis Process is already in the NONSUSPENDED state \n"RESET);
        }
        else if(pcb -> Process_Class == APPLICATION)    {
            pcb->SuspendedState = NO;
        }
        else
            printf("\x1b[31m""\nERROR: Cannot Alter System Process \n""\x1b[0m");
    }
}
```

4.21.1.12 Set_Priority()

```
void Set_Priority (
            char * ProcessName,
            int Priority )
```

Brief Description: Sets PCB priority and reinserts the process into the correct place in the correct queue.

Description: Can except a string as a pointer that is the Process Name. Can accept and integer than is the Priority. Sets a PCB's priority and reinserts the process into the correct place in the correct queue. An error check for valid Process Name and an error check for a valid priority 1 - 9.

Parameters

Process_Name	Character pointer that matches the name of process.
Priority	integer that matches the priority number.

Definition at line 510 of file userFunctions.c.

```
                                                    {
    PCB* pcb = FindPCB(ProcessName);
    if (pcb == NULL)    {
        printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
    }
    else if(Priority >= 10){
        printf("\x1b[31m""\nERROR: Not a valid Priority \n""\x1b[0m");
    }
    else if(pcb -> Process_Class == APPLICATION) {
        RemovePCB(pcb);
        pcb->Priority = Priority;
        InsertPCB(pcb);
    }
    else
        printf("\x1b[31m""\nERROR: Cannot Alter System Process \n""\x1b[0m");
}
```

4.21.1.13 SetDate()

```
void SetDate (
            int day,
            int month,
            int millennium,
            int year )
```

Description: Sets the date register to the new values that the user inputed, all values must be inputed as Set↵Dime(day, month, millenial, year).

Parameters

day	Integer to be set in the Day position
month	Integer to be set in the Month position
millenial	Integer to be set in the Millenial position
year	Integer to be set in the Year position

Definition at line 224 of file userFunctions.c.

```
                                                            {
  outb(0x70,0x07);
```

```
226   int tempDay = BCDtoDec(inb(0x71));
227   outb(0x70,0x08);
228   int tempMonth = BCDtoDec(inb(0x71));
229   outb(0x70,0x32);
230   int tempMillennium = BCDtoDec(inb(0x71));
231   outb(0x70,0x09);
232   int tempYear = BCDtoDec(inb(0x71));
233   cli();
234     outb(0x70,0x07);
235     outb(0x71,DectoBCD (day));
236     outb(0x70,0x08);
237     outb(0x71,DectoBCD (month));
238     outb(0x70,0x32);
239     outb(0x71,DectoBCD (millennium));
240     outb(0x70,0x09);
241     outb(0x71,DectoBCD (year));
242     sti();
243   outb(0x70,0x07);
244   unsigned char newDay = BCDtoDec(inb(0x71));
245   outb(0x70,0x08);
246   unsigned char newMonth = BCDtoDec(inb(0x71));
247   outb(0x70,0x32);
248   unsigned char newMillennium = BCDtoDec(inb(0x71));
249   outb(0x70,0x09);
250   unsigned char newYear = BCDtoDec(inb(0x71));
251   if(newDay != day || newMonth != month || newMillennium != millennium || newYear != year){
252     printf("Your input was invalid\n");
253     cli();
254         outb(0x70,0x07);
255         outb(0x71,DectoBCD (tempDay));
256         outb(0x70,0x08);
257         outb(0x71,DectoBCD (tempMonth));
258         outb(0x70,0x32);
259         outb(0x71,DectoBCD (tempMillennium));
260         outb(0x70,0x09);
261         outb(0x71,DectoBCD (tempYear));
262         sti();
263   }
264   else
265     printf("Date Set\n");
266 }
```

4.21.1.14 SetTime()

```
void SetTime (
              int hours,
              int minutes,
              int seconds )
```

Description: sets the time register to the new values that the user inputed, all values must be inputed as SetTime(↩ Hours, Minutes, Seconds).

Parameters

hours	Integer to be set in the Hour position
minutes	Integer to be set in the Minutes position
seconds	Integer to be set in the Seconds position

Definition at line 151 of file userFunctions.c.

```
151                                                         {
152   outb(0x70,0x04);
153   unsigned char tempHours = BCDtoDec(inb(0x71));
154   outb(0x70,0x02);
155   unsigned char tempMinutes = BCDtoDec(inb(0x71));
156   outb(0x70,0x00);
157   unsigned char tempSeconds = BCDtoDec(inb(0x71));
158     cli(); //outb(device + 1, 0x00); //disable interrupts
159     outb(0x70,0x04);
160     outb(0x71, DectoBCD(hours));// change to bcd
161     outb(0x70,0x02);
```

```
    outb(0x71, DectoBCD(minutes));
    outb(0x70,0x00);
    outb(0x71, DectoBCD(seconds));
    sti();  //outb(device + 4, 0x0B); //enable interrupts, rts/dsr set
  outb(0x70,0x04);
  unsigned char newHours = BCDtoDec(inb(0x71));
  outb(0x70,0x02);
  unsigned char newMinutes = BCDtoDec(inb(0x71));
  outb(0x70,0x00);
  unsigned char newSeconds = BCDtoDec(inb(0x71));
  if(newHours != hours || newMinutes != minutes || newSeconds != seconds){
    printf("Your input was invalid\n");
    cli(); //outb(device + 1, 0x00); //disable interrupts
        outb(0x70,0x04);
        outb(0x71, DectoBCD(tempHours));// change to bcd
        outb(0x70,0x02);
        outb(0x71, DectoBCD(tempMinutes));
        outb(0x70,0x00);
        outb(0x71, DectoBCD(tempSeconds));
        sti();  //outb(device + 4, 0x0B); //enable interrupts, rts/dsr set
  }
  else
    printf("Time Set\n");
}
```

4.21.1.15 Show_All()

```
void Show_All ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready and blocked queues.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the ready and blocked queues.

Definition at line 610 of file userFunctions.c.

```
                {
    Show_Ready();
    Show_Blocked();
}
```

4.21.1.16 Show_Blocked()

```
void Show_Blocked ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the blocked queue.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the blocked queue.

Definition at line 759 of file userFunctions.c.

```
                    {
    if(getBlocked()->head == NULL)  {
         printf("\x1b[32m""\n The Blocked Queue is empty \n""\x1b[0m");
    }
    else    {
          int class, state, prior, status;
          char name[20];
          char block[] = "\x1B[34m""Blocked Queue: \n""\x1b[0m";
          char cname[] = "Name: ";
          char cclass[] = "Class: ";
          char cstate[] = "State: ";
          char cstatus[] = "Status: ";
          char cprior[] = "Priority: ";
          char line[] = "\n";
```

```

        printf(block);
        //sys_req(WRITE, COM1, block, &check );

        PCB* pcb = getBlocked()->head;

        if(pcb->next == NULL) {
          class = pcb->Process_Class;
              strcpy(name,pcb->Process_Name);
              state = pcb->ReadyState;
              status = pcb->SuspendedState;
              prior = pcb->Priority;

              printf(cname);
              printf(name);
              printf(line);

              printf(cclass);
              if(pcb->Process_Class == 0)  {
                printf("0");
              }
              else  {
                printf(itoa(class));
                //sys_req(WRITE, COM1, itoa(class), &check);
              }
              printf(line);

              printf(cstate);
              if(pcb->ReadyState == 0)  {
                printf("0");
              }
              else  {
                printf(itoa(state));
                //sys_req(WRITE, COM1, itoa(state), &check);
              }
              printf(line);

              printf(cstatus);
              if(pcb->SuspendedState == 0)  {
                printf("0");
              }
              else  {
                printf(itoa(status));
                //sys_req(WRITE, COM1, itoa(status), &check);
              }
              printf(line);

              printf(cprior);
              if(pcb->Priority == 0)  {
                printf("0");
                printf("\n\n");
              }
              else  {
                printf(itoa(prior));
                //sys_req(WRITE, COM1, itoa(prior), &check);
                printf("\n\n");
              }
        }
        else  {
          while(pcb != NULL)  {
              class = pcb->Process_Class;
                  strcpy(name,pcb->Process_Name);
                  state = pcb->ReadyState;
                  status = pcb->SuspendedState;
                  prior = pcb->Priority;

                  printf(cname);
                  printf(name);
                  printf(line);

                  printf(cclass);
                  if(pcb->Process_Class == 0)  {
                    printf("0");
                  }
                  else  {
                     printf(itoa(class));
                    //sys_req(WRITE, COM1, itoa(class), &check);
                  }
                  printf(line);

                  printf(cstate);
                  if(pcb->ReadyState == 0)  {
                    printf("0");
                  }
                  else  {
                        printf(itoa(state));
                    //sys_req(WRITE, COM1, itoa(state), &check);
                    }
                    printf(line);

                    printf(cstatus);
                    if(pcb->SuspendedState == 0)  {
                      printf("0");
                    }
                    else  {
                      printf(itoa(status));
                      //sys_req(WRITE, COM1, itoa(status), &check);
                    }
                    printf(line);

                    printf(cprior);
                    if(pcb->Priority == 0)  {
                      printf("0");
                      printf("\n\n");
                    }
                    else  {
                      printf(itoa(prior));
                      //sys_req(WRITE, COM1, itoa(prior), &check);
                      printf("\n\n");
                    }
                pcb = pcb->next;
            }
         }
    }
}
```

4.21.1.17 Show_PCB()

```
void Show_PCB (
               char * ProcessName )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of a PCB.

Description: Can except a string as a pointer that is the Process Name. The process name, claas, state, suspend status, and priority of a PCB are displayed. An error check for a valid name occurs.

Parameters

Process_Name	Character pointer that matches the name of process

Definition at line 535 of file userFunctions.c.

```
535                                     {
536     if (FindPCB(ProcessName) == NULL)   {
537         printf("\x1b[31m""\nERROR: PCB does not exist \n""\x1b[0m");
538     }
539     else    {
540 
541         char name[10];
542         char cname[] = "Name: ";
543         char cclass[] = "Class: ";
544         char cstate[] = "State: ";
545         char cstatus[] = "Status: ";
546         char cprior[] = "Priority: ";
547         char line[] = "\n";
548         PCB* pcb = FindPCB(ProcessName);
549         strcpy(name,pcb->Process_Name);
550         int class = pcb->Process_Class;
551         int state = pcb->ReadyState;
552         int status = pcb->SuspendedState;
553         int prior = pcb->Priority;
554 
555         if(name == NULL){
556             printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
557         }
558         else    {
559             printf(cname);
560             printf(ProcessName);
561             printf(line);
562             printf(cclass);
563             if(pcb->Process_Class == 0)  {
564                 printf("0");
565             }
566             else  {
567                 printf(itoa(class));
568                 //sys_req(WRITE, COM1, itoa(class), &check);
569             }
570             printf(line);
571             printf(cstate);
572             if(pcb->ReadyState == 0)  {
573                 printf("0");
574             }
575             else  {
576                 printf(itoa(state));
577                 //sys_req(WRITE, COM1, itoa(state), &check);
578             }
579             printf(line);
580             printf(cstatus);
581             if(pcb->SuspendedState == 0)  {
582                 printf("0");
583             }
584             else  {
585                 printf(itoa(status));
586                 //sys_req(WRITE, COM1, itoa(status), &check);
587             }
588             printf(line);
589             printf(cprior);
590             if(pcb->Priority == 0)  {
591                 printf("0");
592                 printf("\n\n");
593             }
594             else  {
595                 printf(itoa(prior));
596                 //sys_req(WRITE, COM1, itoa(prior), &check);
597                 printf("\n\n");
598             }
599         }
600     }
601 }
```

4.21.1.18 Show_Ready()

```
void Show_Ready ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready queue.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the ready queue.

Definition at line 622 of file userFunctions.c.

```
622                     {
623     if(getReady()->head == NULL)    {
624          printf("\x1b[32m""\n The Ready Queue is empty \n""\x1b[0m");
625     }
626     else    {
627         int class, state, prior, status;
628       char name[10];
629       char ready[] = "\x1B[34m""\nReady Queue:\n""\x1B[0m";
630       char cname[] = "Name: ";
631       char cclass[] = "Class: ";
632       char cstate[] = "State: ";
633       char cstatus[] = "Status: ";
634       char cprior[] = "Priority: ";
635       char line[] = "\n";
636 
637       printf(ready);
638       //sys_req(WRITE, COM1, ready, &check );
639 
640       PCB* pcb = getReady()->head;
641 
642         if(pcb->next == NULL)   {
643             class = pcb->Process_Class;
644             strcpy(name,pcb->Process_Name);
645             state = pcb->ReadyState;
646             status = pcb->SuspendedState;
```

```
            prior = pcb->Priority;

            printf(cname);
            printf(name);
            printf(line);

            printf(cclass);
            if(pcb->Process_Class == 0)  {
              printf("0");
            }
            else  {
              printf(itoa(class));
              //sys_req(WRITE, COM1, itoa(class), &check);
            }
            printf(line);

            printf(cstate);
            if(pcb->ReadyState == 0)  {
              printf("0");
            }
            else  {
              printf(itoa(state));
              //sys_req(WRITE, COM1, itoa(state), &check);
            }
            printf(line);

            printf(cstatus);
            if(pcb->SuspendedState == 0)  {
              printf("0");
            }
            else  {
              printf(itoa(status));
              //sys_req(WRITE, COM1, itoa(status), &check);
            }
            printf(line);

            printf(cprior);
            if(pcb->Priority == 0)  {
              printf("0");
              printf("\n\n");
            }
            else  {
              printf(itoa(prior));
              //sys_req(WRITE, COM1, itoa(prior), &check);
              printf("\n\n");
            }
      }
      else  {
        while(pcb != NULL)  {
                class = pcb->Process_Class;
                strcpy(name,pcb->Process_Name);
                state = pcb->ReadyState;
                status = pcb->SuspendedState;
                prior = pcb->Priority;

                printf(cname);
                printf(name);
                printf(line);

                printf(cclass);
                if(pcb->Process_Class == 0)  {
                  printf("0");
                }
                else  {
                  printf(itoa(class));
                  //sys_req(WRITE, COM1, itoa(class), &check);
                }
                printf(line);

                printf(cstate);
                if(pcb->ReadyState == 0)  {
                  printf("0");
                }
                else  {
                  printf(itoa(state));
                  //sys_req(WRITE, COM1, itoa(state), &check);
                }
                printf(line);

                printf(cstatus);
                if(pcb->SuspendedState == 0)  {
                  printf("0");
                }
                else  {
                  printf(itoa(status));
                  //sys_req(WRITE, COM1, itoa(status), &check);
                }
            printf(line);

            printf(cprior);
            if(pcb->Priority == 0)  {
              printf("0");
              printf("\n\n");
            }
            else  {
              printf(itoa(prior));
              //sys_req(WRITE, COM1, itoa(prior), &check);
              printf("\n\n");
            }
            pcb = pcb->next;
        }
    }
  }
}
```

4.21.1.19 Suspend()

```
void Suspend (
        char * ProcessName )
```

Brief Description: Places a PCD in the suspended state and reinserts it into the appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Places a PCB in the suspended state and reinserts it into the appropriate queue. An error check for valid Process Name.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 457 of file userFunctions.c.

```
                                {
    PCB* pcb = FindPCB(ProcessName);
    if (pcb == NULL)    {
      printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
    }
    else {
        if(pcb->SuspendedState == YES)  {
            printf("\x1b[32m""\nThis Process is already SUSPENDED \n""\x1b[0m");
        }
        else if(pcb -> Process_Class == APPLICATION)    {
            pcb->SuspendedState = YES;
        }
        else
            printf("\x1b[31m""\nERROR: Cannot Alter System Process \n""\x1b[0m");
    }
}
```

4.21.1.20 toLowercase()

```
char toLowercase (
        char c )
```

Description: If a letter is uppercase, it changes it to lowercase. (char)

Parameters

c	Character that is to be changed to its lowercase equivalent

Definition at line 314 of file userFunctions.c.

```
314                           {
315     if((c >= 65) && (c <= 90))  {
316         c = c + 32;
317     }
318     return c;
319 }
```

4.21.1.21 Unblock()

```
void Unblock (
              char * ProcessName )
```

Brief Description: Places a PCD in the unblocked state and reinserts it into the correct queue.

Description: Can except a string as a pointer that is the Process Name. The specified PCB will be places in an unblocked state and reinserted into the appropriate queue. An error check for a valid name occurs.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 983 of file userFunctions.c.

```
983                               {
984   PCB* pcb = FindPCB(ProcessName);
985   if (pcb == NULL)  {
986     printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
987   }
988   else {
989     if(pcb->ReadyState == READY)    {
990         printf("\x1b[32m""\nThis Process is already in the READY state \n""\x1b[0m");
991     }
992     else    {
993       RemovePCB(pcb);
994       pcb->ReadyState = READY;
995       InsertPCB(pcb);
996     }
997   }
998 }
```

4.21.1.22 Version()

```
void Version ( )
```

Description: Simply returns a char containing "Version: R(module).(the iteration that module is currently on).

No parameters.

Definition at line 307 of file userFunctions.c.

```
307               {
308     printf("Version: R5.2 \n");
309 }
```

4.21.2 Variable Documentation

4.21.2.1 AlarmList

```
List AlarmList
```

Initial value:

```
={
  .head = NULL,
  .tail = NULL
}
```

Definition at line 1045 of file userFunctions.c.

4.21.2.2 else

```
else
```

Initial value:

```
{
         printf("\x1b[31m""\nThe requested command does not exist please refer to the Help function for a
        full list of commands.\n""\x1b[0m")
```

Definition at line 437 of file userFunctions.c.

4.22 modules/R1/userFunctions.h File Reference

Classes

- struct Alarm
- struct List

Macros

- #define **RED** "\x1B[31m"
- #define **GRN** "\x1B[32m"
- #define **YEL** "\x1B[33m"
- #define **BLU** "\x1B[34m"
- #define **MAG** "\x1B[35m"
- #define **CYN** "\x1B[36m"
- #define **WHT** "\x1B[37m"
- #define **RESET** "\x1B[0m"

Typedefs

- typedef struct Alarm **Alarm**
- typedef struct List **List**

Functions

- void SetTime (int hours, int minutes, int seconds)
- void GetTime ()
- int DectoBCD (int Decimal)
- void **clear** ()
- char ∗ itoa (int num)
- void SetDate (int day, int month, int millennium, int year)
- int BCDtoDec (int BCD)
- void GetDate ()
- void Version ()
- void Help (char ∗request)
- void **printf** (char msg[ ])
- int EdgeCase (char ∗pointer)
- char toLowercase (char c)
- void Suspend (char ∗ProcessName)
- void Resume (char ∗ProcessName)
- void Set_Priority (char ∗ProcessName, int Priority)
- void Show_PCB (char ∗ProcessName)
- void Show_All ()
- void Show_Ready ()
- void Show_Blocked ()
- void Create_PCB (char ∗ProcessName, int Priority, int Class)
- void Delete_PCB (char ∗ProcessName)
- void Block (char ∗ProcessName)
- void Unblock (char ∗ProcessName)
- void **loader** ()
- void **loadr3** (char ∗name, u32int func)
- void **yield** ()
- void **loaderinfinite** ()
- List ∗ **getList** ()
- void **loaderalarm** ()

4.22.1 Function Documentation

4.22.1.1 BCDtoDec()

```
int BCDtoDec (
            int BCD )
```

Description: Changes binary number to decimal numbers.

Parameters

value	Binary number to be changed to decimal

Definition at line 81 of file userFunctions.c.

```
                        {
    return (((BCD»4)*10) + (BCD & 0xF));
```

```
}
```

4.22.1.2 Block()

```
void Block (
              char * ProcessName )
```

Brief Description: Places a PCD in the blocked state and reinserts it into the correct queue.

Description: Can except a string as a pointer that is the Process Name. The specified PCB will be places in a blocked state and reinserted into the appropriate queue. An error check for a valid name occurs.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 957 of file userFunctions.c.

```
                                {
  PCB* pcb = FindPCB(ProcessName);
  if (pcb == NULL)  {
    printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
  }
  else {
    if(pcb->ReadyState == BLOCKED)  {
        printf("\x1b[32m""\nThis Process is already BLOCKED \n""\x1b[0m");
    }
    else    {
      RemovePCB(pcb);
      pcb->ReadyState = BLOCKED;
      InsertPCB(pcb);
    }
  }
}
```

4.22.1.3 Create_PCB()

```
void Create_PCB (
              char * ProcessName,
              int Priority,
              int Class )
```

Brief Description: Calls SetupPCB() and inserts PCB into appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Can accept two integers, Priority and Class. SetupPCB() will be called and the PCB will be inserted into the appropriate queue. An error check for unique and valid Process Name, an error check for valid process class, and an error check for process priority.

Parameters

Process_Name	Character pointer that matches the name of process.
Priority	integer that matches the priority number.
Class	integer that matches the class number.

Definition at line 900 of file userFunctions.c.

```
                                                               {
  if (FindPCB(ProcessName) == NULL) {
    if(Priority >= 0 && Priority < 10){
      if(Class == 0 || Class == 1){
        PCB* pcb = SetupPCB(ProcessName, Class, Priority);
        InsertPCB(pcb);
      } else{
        printf("\x1b[31m""\nERROR: Not a valid Class \n""\x1b[0m");
      }
    } else{
      printf("\x1b[31m""\nERROR: Not a valid Priority \n""\x1b[0m");
    }
  } else{
    printf("\x1b[31m""\nERROR: This Process Name already exists \n""\x1b[0m");
  }
}
```

4.22.1.4 DectoBCD()

```
int DectoBCD (
              int Decimal )
```

Description: Changes decimal numbers to binary numbers.

Parameters

Decimal	Decimal number to be changed to binary

Definition at line 88 of file userFunctions.c.

```
                            {
     return (((Decimal/10) « 4) | (Decimal % 10));
}
```

4.22.1.5 Delete_PCB()

```
void Delete_PCB (
              char * ProcessName )
```

Brief Description: Removes PCB from appropriate queue and frees all associated memory.

Description: Can except a string as a pointer that is the Process Name. Removes PCB from the appropriate queue and then frees all associated memory. An error check to make sure process name is valid.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 926 of file userFunctions.c.

```
                                         {
  PCB* pcb = FindPCB(ProcessName);
  if (pcb == NULL)  {
    printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
  }
  else if(strcmp(pcb->Process_Name,"InfProc") == 0)   {
        if(pcb->SuspendedState == YES)  {
            RemovePCB(pcb);
            FreePCB(pcb);
        }
        else
            printf("\x1b[31m""\nERROR:This process cannot be deleted unless it is in the suspended
 state\n""\x1b[0m");
  }
  else if(pcb -> Process_Class == SYSTEM)   {
      printf("\x1b[31m""\nERROR: System Processes cannot be deleted from the system. \n""\x1b[0m");
  }
    else {
        RemovePCB(pcb);
        FreePCB(pcb);
    }
}
```

4.22.1.6 EdgeCase()

```
int EdgeCase (
            char * pointer )
```

Description: Compares pointer char to validate if it is a number or not.

Parameters

Compares	pointer char to validate if it is a number or not.

Definition at line 109 of file userFunctions.c.

```
                              {
  int valid = 0;
  if (strcmp(pointer, "00") == 0) {
    valid = 1;
    return valid;
  }
  else if (strcmp(pointer, "0") == 0) {
    valid = 1;
    return valid;
  }
  else  {
    int j;
    valid = 0;
    for(j = 0; j <= 99; j++)  {
      if(strcmp(pointer,itoa(j)) == 0)
        valid = 1;
    }
    if(valid == 0)  {
      return valid;
    }
  }
  return valid;
}
```

4.22.1.7 GetDate()

```
void GetDate ( )
```

Description: Returns the full date back to the user in decimal form.

No parameters.

Definition at line 271 of file userFunctions.c.

```
                 {
```

```
272
273   outb(0x70,0x07);
274     unsigned char day = BCDtoDec(inb(0x71));
275     outb(0x70,0x08);
276     unsigned char month = BCDtoDec(inb(0x71));
277     outb(0x70,0x32);
278     unsigned char millennium = BCDtoDec(inb(0x71));
279     char msg[2] = "-";
280     char msg3[10] = "Date: ";
281     printf(msg3);
282
283      printf(itoa(day));
284     //sys_req(WRITE, COM1, itoa(day), &check);
285     printf(msg);
286     printf(itoa(month));
287     //sys_req(WRITE, COM1, itoa(month), &check);
288     printf(msg);
289     printf(itoa(millennium));
290     //sys_req(WRITE, COM1, itoa(millennium), &check);
291   outb(0x70,0x09);
292   if(BCDtoDec(inb(0x71)) == 0){
293      printf("00");
294     //sys_req(WRITE, COM1, "00", &check);
295   }
296   else {
297         unsigned char year = BCDtoDec(inb(0x71));
298          printf(itoa(year));
299         //sys_req(WRITE, COM1, itoa(year), &check);
300   }
301     printf("\n");
302 }
```

4.22.1.8 GetTime()

```
void GetTime ( )
```

Description: retrieve and return the time values for hours, minutes, and seconds form the clock register using inb(Port,address).

No parameters.

Definition at line 190 of file userFunctions.c.

```
190                {
191
192     int hour;
193     int minute;
194     int second;
195     outb(0x70,0x04);
196     unsigned char hours = inb(0x71);
197     outb(0x70,0x02);
198     unsigned char minutes = inb(0x71);
199     outb(0x70,0x00);
200     unsigned char seconds = inb(0x71);
201     char msg1[2] = ":";
202     char msg2[10] = "Time: ";
203     printf(msg2);
204     hour = BCDtoDec(hours);
205     printf(itoa(hour));
206     //sys_req(WRITE, COM1, itoa(hour), &check);
207     printf(msg1);
208     minute = BCDtoDec(minutes);
209     printf(itoa(minute));
210     //sys_req(WRITE, COM1, itoa(minute), &check);
211     printf(msg1);
212     second = BCDtoDec(seconds);
213     printf(itoa(second));
214     //sys_req(WRITE, COM1, itoa(second), &check);
215   printf("\n");
216 }
```

4.22.1.9 Help()

```
void Help (
              char * request )
```

Brief Description: Gives helpful information for one of the functions

Description: Can except a string as a pointer, if the pointer is null then the function will print a complete list of avaliable commands to the console. If the pointer is a avaliable commands then instructions on how to use the command will be printed. If the command does not exist then a message explaining that it is not a valid command will be displayed.

Parameters

request	Character pointer that matches the name of the function that you need help with.

Definition at line 331 of file userFunctions.c.

```
                                {
    if (request[0] == '\0') {
        //removed for R3/R4 from active command list
        //\n createPCB \n block \n unblock
        //\n heap          alloc \n free          empty
        printf("\n to chain commands and parameters, please use \"-\" between keywords \n");
        printf("\n getDate      setDate \n getTime      setTime \n version      suspend \n resume
    setPriority \n showPCB      showAll \n showReady    showBlocked  \n deletePCB    shutdown \n alarm
        clear \n loadr3       infinte \n showFree     showAlloc \n\n");
    }
    else if (strcmp(request, "GetDate") == 0) {
        printf("\n getDate returns the current date that is loaded onto the operating system.\n");
    }
    else if (strcmp(request, "SetDate") == 0) {
        printf("\n setDate allows the user to reset the correct date into the system, as follows
    setDate-"BLU"day"RESET"-"BLU"month"RESET"-"BLU"year"RESET".\n Time must be inputed as a two digit
    number, Example 02 or 00");
    }
    else if (strcmp(request, "GetTime") == 0) {
        printf("\n getTime returns the current time as hours, minutes, seconds that is loaded onto the
    operating system.\n");
    }
    else if (strcmp(request, "SetTime") == 0) {
        printf("\n setTime allows the user to reset the correct time into the system, as follows
    setTime-"BLU"hour"RESET"-"BLU"minute"RESET"-"BLU"second"RESET".\n Time must be inputed as a two digit
    number, Example 02 or 00");
    }
    else if (strcmp(request, "Version") == 0) {
        printf("\n version returns the current operating software version that the system is
    running.\n");
    }
    else if (strcmp(request, "infinte") == 0) {
        printf("\n infinite Loads the infinite process into the ready queue.\n");
    }
    else if (strcmp(request, "loadr3") == 0)  {
        printf("\n loadr3 Loads in all five of the R3 test processes in a suspended state into the
    queue.\n");
    }
    else if (strcmp(request, "alarm") == 0)       {
        printf("\n alarm creates a user specified alarm with a user set message and time
    alarm-MSG-hour-minute-second.\n");
    }
    else if (strcmp(request, "clear") == 0)       {
        printf("\n clear erases the console of all typed commands and refreshes it with just the command
    list.\n");
    }

  else if(strcmp(request, "shutdown") == 0)   {
    printf("\n shutdown shuts down the system.\n");
  }



    /*****************************************************************************************************
             R2 Commands

    *****************************************************************************************************/
  else if(strcmp(request,"suspend") == 0) {
        printf("\n Suspend takes in the name of a PCB (suspend-NAME) then places it into the suspended
    state and reinserts it into the correct queue.\n");
```

```
}
else if(strcmp(request,"resume") == 0) {
      printf("\n Resume takes in the name of a PCB (resume-NAME) then removes it from the suspended
 state and adds it to the correct queue.\n");
}
else if(strcmp(request,"setPriority") == 0) {
      printf("\n SetPriority takes in the name of a PCB and the priority (setPrioriry-NAME-PRIORITY)
 it needs to be set to then reinstates the specified PCB into a new location by priority.\n");
}
else if(strcmp(request,"showPCB") == 0) {
      printf("\n ShowPCB takes in the name of a PCB and returns all the associated attributes to the
 user.\n");
}
else if(strcmp(request,"showAll") == 0) {
      printf("\n ShowAll takes no parameters but returns all PCB's that are currently in any of the
 queues.\n");
}
else if(strcmp(request,"showReady") == 0) {
      printf("\n ShowReady takes in no parameters but returns all PCB's and there attributes that
 currently are in the ready state.\n");
}
else if(strcmp(request,"showBlocked") == 0) {
      printf("\n ShowBlocked takes in no parameters but returns all PCB's and there attributes that
 currently are in the blocked state.\n");
}

/********************************* R2 Temp Commands
 *******************************************************/
else if(strcmp(request,"deletePCB") == 0) {
      printf("\n DeletePCB takes in the process_name (deletePCB-NAME) then deletes it from the queue
 and free's all the memory that was previously allocated to the specified PCB.\n");
}
//removed for R3/R4 from active command list
/*
else if(strcmp(request,"createPCB") == 0) {
      printf("\n CreatePCB takes in the process_name, process_class, and
 process_priority.(createPCB-NAME-PRIORITY-CLASS) Then assigns this new process into the correct
 queue.\n");
}
else if(strcmp(request,"block") == 0) {
      printf("\n Block takes in the process_name (block-NAME) then sets it's state to blocked and
 reinserts it back into the correct queue.\n");
}
else if(strcmp(request,"unblock") == 0) {
      printf("\n Unblock takes in the process_name (unblock-NAME) then sets it's state to ready and
 reinserts it back into the correct queue.\n");
}
*/


/********************************* R5 Temp Commands
 *******************************************************/
// else if(strcmp(request,"heap") == 0) {
  //  printf("\n heap initializes the memory heap for the entire system.\n");
// }
// else if(strcmp(request,"alloc") == 0) {
  //  printf("\n alloc allocates the specified amount of memory to the specific process
 (alloc-process_name-size).\n");
// }
// else if(strcmp(request,"free") == 0) {
  //  printf("\n free frees the specified memory at the address given (free-address).\n");
// }
// else if(strcmp(request,"empty") == 0) {
  //  printf("\n isempty returns true or false depending on if the heap has allocated memory.\n");
}
```

4.22.1.10 itoa()

```
char* itoa (
              int num )
```

Description: An integer is taken and seperated into individual chars and then all placed into a character array. Adapted from geeksforgeeks.org.

Parameters

num	integer to be put into array Title: itoa Author: Neha Mahajan Date: 29 May, 2017 Availability: https://www.geeksforgeeks.org/implement-itoa/

Definition at line 50 of file userFunctions.c.

```
                      {

    int i,j,k,count;
    i = num;
    j = 0;
    count = 0;
    while(i){ // count number of digits
        count++;
        i /= 10;
    }

    char* arr1;
    char arr2[count];
    arr1 = (char*)sys_alloc_mem(count); //memory allocation

    while(num){ // seperate last digit from number and add ASCII
        arr2[++j] = num%10 + '0';
        num /= 10;
    }

    for(k = 0; k < j; k++){ // reverse array results
        arr1[k] = arr2[j-k];
    }
    arr1[k] = '\0';

    return(char*)arr1;
}
```

4.22.1.11 Resume()

```
void Resume (
               char * ProcessName )
```

Brief Description: Places a PCD in the not suspended state and reinserts it into the appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Places a PCB in the not suspended state and reinserts it into the appropriate queue. An error check for valid Process Name.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 483 of file userFunctions.c.

```
                               {
    PCB* pcb = FindPCB(ProcessName);
    if (pcb == NULL)    {
      printf(RED"\nERROR: Not a valid process name \n"RESET);
    }
    else {
        if(pcb->SuspendedState == NO)    {
            printf(GRN"\nThis Process is already in the NONSUSPENDED state \n"RESET);
        }
        else if(pcb -> Process_Class == APPLICATION)     {
            pcb->SuspendedState = NO;
        }
        else
            printf("\x1b[31m""\nERROR: Cannot Alter System Process \n""\x1b[0m");
    }
}
```

4.22.1.12 Set_Priority()

```
void Set_Priority (
            char * ProcessName,
            int Priority )
```

Brief Description: Sets PCB priority and reinserts the process into the correct place in the correct queue.

Description: Can except a string as a pointer that is the Process Name. Can accept and integer than is the Priority. Sets a PCB's priority and reinserts the process into the correct place in the correct queue. An error check for valid Process Name and an error check for a valid priority 1 - 9.

Parameters

Process_Name	Character pointer that matches the name of process.
Priority	integer that matches the priority number.

Definition at line 510 of file userFunctions.c.

```
                                                    {
    PCB* pcb = FindPCB(ProcessName);
    if (pcb == NULL)    {
        printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
    }
    else if(Priority >= 10){
        printf("\x1b[31m""\nERROR: Not a valid Priority \n""\x1b[0m");
    }
    else if(pcb -> Process_Class == APPLICATION) {
        RemovePCB(pcb);
        pcb->Priority = Priority;
        InsertPCB(pcb);
    }
    else
        printf("\x1b[31m""\nERROR: Cannot Alter System Process \n""\x1b[0m");
}
```

4.22.1.13 SetDate()

```
void SetDate (
            int day,
            int month,
            int millennium,
            int year )
```

Description: Sets the date register to the new values that the user inputed, all values must be inputed as Set←Dime(day, month, millenial, year).

Parameters

day	Integer to be set in the Day position
month	Integer to be set in the Month position
millenial	Integer to be set in the Millenial position
year	Integer to be set in the Year position

Definition at line 224 of file userFunctions.c.

```
                                                            {
  outb(0x70,0x07);
  int tempDay = BCDtoDec(inb(0x71));
  outb(0x70,0x08);
  int tempMonth = BCDtoDec(inb(0x71));
  outb(0x70,0x32);
  int tempMillennium = BCDtoDec(inb(0x71));
  outb(0x70,0x09);
  int tempYear = BCDtoDec(inb(0x71));
  cli();
    outb(0x70,0x07);
    outb(0x71,DectoBCD (day));
    outb(0x70,0x08);
    outb(0x71,DectoBCD (month));
    outb(0x70,0x32);
    outb(0x71,DectoBCD (millennium));
    outb(0x70,0x09);
    outb(0x71,DectoBCD (year));
    sti();
  outb(0x70,0x07);
  unsigned char newDay = BCDtoDec(inb(0x71));
  outb(0x70,0x08);
  unsigned char newMonth = BCDtoDec(inb(0x71));
  outb(0x70,0x32);
  unsigned char newMillennium = BCDtoDec(inb(0x71));
  outb(0x70,0x09);
  unsigned char newYear = BCDtoDec(inb(0x71));
  if(newDay != day || newMonth != month || newMillennium != millennium || newYear != year){
    printf("Your input was invalid\n");
    cli();
        outb(0x70,0x07);
        outb(0x71,DectoBCD (tempDay));
        outb(0x70,0x08);
        outb(0x71,DectoBCD (tempMonth));
        outb(0x70,0x32);
        outb(0x71,DectoBCD (tempMillennium));
        outb(0x70,0x09);
        outb(0x71,DectoBCD (tempYear));
        sti();
  }
  else
    printf("Date Set\n");
}
```

4.22.1.14 SetTime()

```
void SetTime (
              int hours,
              int minutes,
              int seconds )
```

Description: sets the time register to the new values that the user inputed, all values must be inputed as SetTime(↩ Hours, Minutes, Seconds).

Parameters

hours	Integer to be set in the Hour position
minutes	Integer to be set in the Minutes position
seconds	Integer to be set in the Seconds position

Definition at line 151 of file userFunctions.c.

```
                                                    {
  outb(0x70,0x04);
  unsigned char tempHours = BCDtoDec(inb(0x71));
  outb(0x70,0x02);
  unsigned char tempMinutes = BCDtoDec(inb(0x71));
  outb(0x70,0x00);
  unsigned char tempSeconds = BCDtoDec(inb(0x71));
    cli(); //outb(device + 1, 0x00); //disable interrupts
    outb(0x70,0x04);
    outb(0x71, DectoBCD(hours));// change to bcd
    outb(0x70,0x02);
    outb(0x71, DectoBCD(minutes));
    outb(0x70,0x00);
    outb(0x71, DectoBCD(seconds));
    sti();  //outb(device + 4, 0x0B); //enable interrupts, rts/dsr set
  outb(0x70,0x04);
  unsigned char newHours = BCDtoDec(inb(0x71));
  outb(0x70,0x02);
  unsigned char newMinutes = BCDtoDec(inb(0x71));
  outb(0x70,0x00);
  unsigned char newSeconds = BCDtoDec(inb(0x71));
  if(newHours != hours || newMinutes != minutes || newSeconds != seconds){
    printf("Your input was invalid\n");
    cli(); //outb(device + 1, 0x00); //disable interrupts
        outb(0x70,0x04);
        outb(0x71, DectoBCD(tempHours));// change to bcd
        outb(0x70,0x02);
        outb(0x71, DectoBCD(tempMinutes));
        outb(0x70,0x00);
        outb(0x71, DectoBCD(tempSeconds));
        sti();  //outb(device + 4, 0x0B); //enable interrupts, rts/dsr set
  }
  else
    printf("Time Set\n");
}
```

4.22.1.15 Show_All()

```
void Show_All ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready and blocked queues.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the ready and blocked queues.

Definition at line 610 of file userFunctions.c.

```
                {
    Show_Ready();
    Show_Blocked();
}
```

4.22.1.16 Show_Blocked()

```
void Show_Blocked ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the blocked queue.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the blocked queue.

Definition at line 759 of file userFunctions.c.

```
                    {
    if(getBlocked()->head == NULL)  {
        printf("\x1b[32m""\n The Blocked Queue is empty \n""\x1b[0m");
    }
    else    {
          int class, state, prior, status;
          char name[20];
          char block[] = "\x1B[34m""Blocked Queue: \n""\x1b[0m";
          char cname[] = "Name: ";
          char cclass[] = "Class: ";
          char cstate[] = "State: ";
          char cstatus[] = "Status: ";
          char cprior[] = "Priority: ";
          char line[] = "\n";

          printf(block);
          //sys_req(WRITE, COM1, block, &check );

          PCB* pcb = getBlocked()->head;

          if(pcb->next == NULL) {
            class = pcb->Process_Class;
                strcpy(name,pcb->Process_Name);
                state = pcb->ReadyState;
                status = pcb->SuspendedState;
                prior = pcb->Priority;

                printf(cname);
                printf(name);
                printf(line);

                printf(cclass);
                if(pcb->Process_Class == 0)  {
                  printf("0");
                }
                else  {
                  printf(itoa(class));
                  //sys_req(WRITE, COM1, itoa(class), &check);
                }
                printf(line);

                printf(cstate);
                if(pcb->ReadyState == 0)  {
                  printf("0");
                }
                else  {
                  printf(itoa(state));
                  //sys_req(WRITE, COM1, itoa(state), &check);
                }
                printf(line);

                printf(cstatus);
                if(pcb->SuspendedState == 0)  {
                  printf("0");
                }
                else  {
                  printf(itoa(status));
                  //sys_req(WRITE, COM1, itoa(status), &check);
                }
                printf(line);

                printf(cprior);
                if(pcb->Priority == 0)  {
                  printf("0");
                  printf("\n\n");
                }
                else  {
                  printf(itoa(prior));
                  //sys_req(WRITE, COM1, itoa(prior), &check);
                  printf("\n\n");
                }
          }
          else  {
            while(pcb != NULL)  {
                class = pcb->Process_Class;
                    strcpy(name,pcb->Process_Name);
                    state = pcb->ReadyState;
                    status = pcb->SuspendedState;
                    prior = pcb->Priority;

                    printf(cname);
                    printf(name);
                    printf(line);

                    printf(cclass);
                    if(pcb->Process_Class == 0)  {
                      printf("0");
                    }
                    else  {
                       printf(itoa(class));
                      //sys_req(WRITE, COM1, itoa(class), &check);
                    }
                    printf(line);

                    printf(cstate);
                    if(pcb->ReadyState == 0)  {
                      printf("0");
                    }
                    else  {
                          printf(itoa(state));
                      //sys_req(WRITE, COM1, itoa(state), &check);
                    }
                    printf(line);

                    printf(cstatus);
                    if(pcb->SuspendedState == 0)  {
                      printf("0");
                    }
                    else  {
                      printf(itoa(status));
                      //sys_req(WRITE, COM1, itoa(status), &check);
                    }
                    printf(line);

                    printf(cprior);
                    if(pcb->Priority == 0)  {
                      printf("0");
                      printf("\n\n");
                    }
                    else  {
                      printf(itoa(prior));
                      //sys_req(WRITE, COM1, itoa(prior), &check);
                      printf("\n\n");
                    }
                pcb = pcb->next;
            }
        }
    }
}
```

4.22.1.17 Show_PCB()

```
void Show_PCB (
              char * ProcessName )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of a PCB.

Description: Can except a string as a pointer that is the Process Name. The process name, claas, state, suspend status, and priority of a PCB are displayed. An error check for a valid name occurs.

Parameters

Process_Name	Character pointer that matches the name of process

Definition at line 535 of file userFunctions.c.

```
                                    {
    if (FindPCB(ProcessName) == NULL)   {
        printf("\x1b[31m""\nERROR: PCB does not exist \n""\x1b[0m");
    }
    else    {

        char name[10];
        char cname[] = "Name: ";
        char cclass[] = "Class: ";
        char cstate[] = "State: ";
        char cstatus[] = "Status: ";
        char cprior[] = "Priority: ";
        char line[] = "\n";
        PCB* pcb = FindPCB(ProcessName);
        strcpy(name,pcb->Process_Name);
        int class = pcb->Process_Class;
        int state = pcb->ReadyState;
        int status = pcb->SuspendedState;
        int prior = pcb->Priority;

        if(name == NULL){
            printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
        }
        else    {
            printf(cname);
            printf(ProcessName);
            printf(line);
            printf(cclass);
            if(pcb->Process_Class == 0)  {
                printf("0");
            }
            else  {
                printf(itoa(class));
                //sys_req(WRITE, COM1, itoa(class), &check);
            }
            printf(line);
            printf(cstate);
            if(pcb->ReadyState == 0)  {
                printf("0");
            }
            else  {
                printf(itoa(state));
                //sys_req(WRITE, COM1, itoa(state), &check);
            }
            printf(line);
            printf(cstatus);
            if(pcb->SuspendedState == 0)  {
                printf("0");
            }
            else  {
                printf(itoa(status));
                //sys_req(WRITE, COM1, itoa(status), &check);
            }
            printf(line);
            printf(cprior);
            if(pcb->Priority == 0)  {
                printf("0");
                printf("\n\n");
            }
            else  {
                printf(itoa(prior));
                //sys_req(WRITE, COM1, itoa(prior), &check);
                printf("\n\n");
            }
        }
    }
}
```

4.22.1.18 Show_Ready()

```
void Show_Ready ( )
```

Brief Description: Displays the process name, class, state, suspended status, and priority of all PCB in the ready queue.

Description: The process name, claas, state, suspend status, and priority of each of he PCB's in the ready queue.

Definition at line 622 of file userFunctions.c.

```
                    {
    if(getReady()->head == NULL)    {
         printf("\x1b[32m""\n The Ready Queue is empty \n""\x1b[0m");
    }
    else    {
        int class, state, prior, status;
      char name[10];
      char ready[] = "\x1B[34m""\nReady Queue:\n""\x1B[0m";
      char cname[] = "Name: ";
      char cclass[] = "Class: ";
      char cstate[] = "State: ";
      char cstatus[] = "Status: ";
      char cprior[] = "Priority: ";
      char line[] = "\n";

      printf(ready);
      //sys_req(WRITE, COM1, ready, &check );

      PCB* pcb = getReady()->head;

        if(pcb->next == NULL)   {
            class = pcb->Process_Class;
            strcpy(name,pcb->Process_Name);
            state = pcb->ReadyState;
            status = pcb->SuspendedState;
            prior = pcb->Priority;

            printf(cname);
            printf(name);
            printf(line);

            printf(cclass);
            if(pcb->Process_Class == 0)  {
              printf("0");
            }
            else  {
              printf(itoa(class));
              //sys_req(WRITE, COM1, itoa(class), &check);
            }
            printf(line);

            printf(cstate);
            if(pcb->ReadyState == 0)  {
              printf("0");
            }
            else  {
              printf(itoa(state));
              //sys_req(WRITE, COM1, itoa(state), &check);
            }
            printf(line);

            printf(cstatus);
            if(pcb->SuspendedState == 0)  {
              printf("0");
            }
            else  {
              printf(itoa(status));
              //sys_req(WRITE, COM1, itoa(status), &check);
            }
            printf(line);

            printf(cprior);
            if(pcb->Priority == 0)  {
              printf("0");
              printf("\n\n");
            }
            else  {
              printf(itoa(prior));
              //sys_req(WRITE, COM1, itoa(prior), &check);
              printf("\n\n");
            }
      }
      else  {
        while(pcb != NULL)  {
                class = pcb->Process_Class;
                strcpy(name,pcb->Process_Name);
                state = pcb->ReadyState;
                status = pcb->SuspendedState;
                prior = pcb->Priority;

                printf(cname);
                printf(name);
                printf(line);

                printf(cclass);
                if(pcb->Process_Class == 0)  {
                  printf("0");
                }
                else  {
                  printf(itoa(class));
                  //sys_req(WRITE, COM1, itoa(class), &check);
                }
                printf(line);

                printf(cstate);
                if(pcb->ReadyState == 0)  {
                  printf("0");
                }
                else  {
                  printf(itoa(state));
                  //sys_req(WRITE, COM1, itoa(state), &check);
                }
                printf(line);

                printf(cstatus);
                if(pcb->SuspendedState == 0)  {
                  printf("0");
                }
                else  {
                  printf(itoa(status));
                  //sys_req(WRITE, COM1, itoa(status), &check);
                }
                printf(line);

                printf(cprior);
                if(pcb->Priority == 0)  {
                  printf("0");
                  printf("\n\n");
                }
                else  {
                  printf(itoa(prior));
                  //sys_req(WRITE, COM1, itoa(prior), &check);
                  printf("\n\n");
                }
                pcb = pcb->next;
            }
        }
    }
}
```

4.22.1.19 Suspend()

```
void Suspend (
              char * ProcessName )
```

Brief Description: Places a PCD in the suspended state and reinserts it into the appropriate queue.

Description: Can except a string as a pointer that is the Process Name. Places a PCB in the suspended state and reinserts it into the appropriate queue. An error check for valid Process Name.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 457 of file userFunctions.c.

```
                                {
    PCB* pcb = FindPCB(ProcessName);
    if (pcb == NULL)    {
      printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
    }
    else {
        if(pcb->SuspendedState == YES)  {
            printf("\x1b[32m""\nThis Process is already SUSPENDED \n""\x1b[0m");
        }
        else if(pcb -> Process_Class == APPLICATION)    {
            pcb->SuspendedState = YES;
        }
        else
            printf("\x1b[31m""\nERROR: Cannot Alter System Process \n""\x1b[0m");
    }
}
```

4.22.1.20 toLowercase()

```
char toLowercase (
              char c )
```

Description: If a letter is uppercase, it changes it to lowercase. (char)

Parameters

c	Character that is to be changed to its lowercase equivalent

Definition at line 314 of file userFunctions.c.

```
                              {
    if((c >= 65) && (c <= 90))  {
        c = c + 32;
    }
    return c;
}
```

4.22.1.21 Unblock()

```
void Unblock (
              char * ProcessName )
```

Brief Description: Places a PCD in the unblocked state and reinserts it into the correct queue.

Description: Can except a string as a pointer that is the Process Name. The specified PCB will be places in an unblocked state and reinserted into the appropriate queue. An error check for a valid name occurs.

Parameters

Process_Name	Character pointer that matches the name of process.

Definition at line 983 of file userFunctions.c.

```
                                  {
  PCB* pcb = FindPCB(ProcessName);
  if (pcb == NULL)  {
    printf("\x1b[31m""\nERROR: Not a valid process name \n""\x1b[0m");
  }
  else {
    if(pcb->ReadyState == READY)    {
        printf("\x1b[32m""\nThis Process is already in the READY state \n""\x1b[0m");
    }
    else    {
      RemovePCB(pcb);
      pcb->ReadyState = READY;
      InsertPCB(pcb);
    }
  }
}
```

4.22.1.22 Version()

```
void Version ( )
```

Description: Simply returns a char containing "Version: R(module).(the iteration that module is currently on).

No parameters.

Definition at line 307 of file userFunctions.c.

```
                 {
    printf("Version: R5.2 \n");
}
```

4.23 modules/sys_proc_loader.c File Reference

```
#include <stdint.h>
#include <string.h>
#include <system.h>
#include <core/serial.h>
#include <core/io.h>
#include "mpx_supt.h"
#include "R1/userFunctions.h"
#include "procsr3.h"
#include "R1/comHand.h"
#include "sys_proc_loader.h"
```

Functions

- void **sysLoader** ()
- void **loadSysProc** (char ∗name, u32int func, int priority)
- void **InfiniteProc** ()
- void **AlarmProc** ()

4.24 modules/sys_proc_loader.h File Reference

Functions

- void **sysLoader** ()
- void **loadSysProc** (char ∗name, u32int func, int priority)
- void **InfiniteProc** ()
- void **AlarmProc** ()

Index