



# **Marauders OS Programmer's Manual**

Authors: *Nathan Mullins, Riley Stauffer*

---

# Table of Contents

<b>R1</b>	<b>1</b>
<b>1. Serial.h</b>	<b>1</b>
1.1 int serial_poll(device dev, char *buffer, size_t len)	1
1.2 clear_input_buffer(device dev)	1
<b>2. itoa.h</b>	<b>2</b>
2.1 char* itoa(int value, char* destination, int base)	2
<b>3. doubly_linked_list.h</b>	<b>2</b>
3.1 void insertFront(struct Node** head, char* data)	2
3.2 void insertAfter(struct Node* prev_node, char* data)	3
3.3 void insertEnd(struct Node** head, char* data)	3
3.4 deleteNode(struct Node** head, struct Node* del_node)	4
3.5 struct Node	4
<b>4. comhand.h</b>	<b>4</b>
4.1 void comhand(void)	4
<b>5. help.h</b>	<b>5</b>
5.1 void help(const char *cmd)	5
<b>6. time.h</b>	<b>5</b>
6.3 void set_time(const char *command)	6
6.4 int isValidTimeFormat(const char* input)	6
6.5 void set_date(uint8_t day, uint8_t month, uint8_t year)	7
6.6 int hexToDec(uint8_t hex)	7
<b>7. version.h</b>	<b>7</b>
7.1 void version(void)	7
<b>R2</b>	<b>8</b>
<b>8. process_queues.h</b>	<b>8</b>
8.1 struct node	8
8.2 struct queue	8
8.3 void enqueue_pri(char* status, struct pcb* pcb)	9
8.4 void dequeue(const char* status, struct pcb* pcb)	9
8.5 char* search(struct pcb* pcb_ptr)	9
8.6 struct node* create_node(struct pcb* pcb)	9
8.7 void enqueue_reg(char* status, struct pcb* pcb)	10
<b>9. pcb.h</b>	<b>10</b>
9.1 struct pcb	10
9.2 enum state	10
9.3 struct process	11
9.4 struct pcb* allocate(void)	11
9.5 int pcb_free(void)	11

9.6 struct pcb* pcb_setup(const char* process_name, int class, int priority)	12
9.7 struct pcb* pcb_find(const char* process)	12
9.8 void pcb_insert(struct pcb* process)	12
9.9 int pcb_remove(struct pcb* process)	12
9.10 void create_pcb(const char* name, int class, int priority)	13
9.11 void delete_pcb(const char* name)	13
9.12 void block_pcb(const char* name)	13
9.13 void unblock_pcb(const char* name)	14
9.14 void suspend_pcb(const char* name)	14
9.15 void resume_pcb(const char* name)	14
9.16 void set_pcb_priority(const char* name, int priority)	14
9.17 void show_pcb(const char* name)	15
9.18 void show_ready(int status)	15
9.19 void show_blocked(int status)	15
9.20 void show_all(void)	15
<b>R3</b>	<b>16</b>
<b>10. context_switch.h</b>	<b>16</b>
10.1 struct context	16
10.2 struct context* sys_call(struct context* context_ptr)	17
<b>11. pcb.h &amp; pcb_user_commands.c continued</b>	<b>17</b>
11.1 void yield(void)	17
11.2 void Load_R3(void)	17
<b>12. sys_call_isr.s</b>	<b>18</b>
12.1 sys_call_isr	18
<b>R4</b>	<b>19</b>
<b>13. pcb.h and pcb_user_commands.c continued</b>	<b>19</b>
13.1 void load_comhand(void)	19
13.2 void load_sys_idle(void)	19
<b>14. shutdown.h</b>	<b>19</b>
14.1 void shutdown(void)	19
<b>15. alarms.h</b>	<b>20</b>
15.1 void alarm(char* formatted_time, char* message)	20
15.2 void print_message(void)	20
15.3 int check_time(void)	21

# R1

## 1. Serial.h

1.1 int serial\_poll(device dev, char \*buffer, size\_t len)

Author:

Nathan Mullins

Description

polls a device dev for data and will store it in an array one character at a time until a new line (enter key) is encountered or the buffer is full. Returns the number of bytes read from dev, or a negative number if an error has occurred.

Parameters

device dev - a communication port for serial communication limited to predefined devices COM1, COM2, COM3, COM4, which represent addresses for serial communication ports.

char\* buffer - user provided buffer that stores chars being polled from dev. Size is predefined.

size\_t len - length of buffer, allowable amount of characters to enter buffer without overflow.

1.2 clear\_input\_buffer(device dev)

Author:

Nathan Mullins

Description

Clears the specified device of characters to flush unwanted/extra output characters, ideally before writing then reading output to the device. Does not return anything.

Parameters

device dev - a communication port for serial communication limited to predefined devices COM1, COM2, COM3, COM4, which represent addresses for serial communication ports.

## 2. itoa.h

2.1 char\* itoa(int value, char\* destination, int base)

Author:

Nathan Mullins

Description

Converts an integer to a string value with a specified base (ie. binary, hex, decimal, etc.) Function takes in an integer of base 10, an array to store the converted integer, and a numerical base used to represent the value as a string. Returns the converted string into the destination array.

Parameters

int value - integer to convert to string. Integers must be non-negative and not exceed  $((2^{31})-1)$ , since an int is 4 bytes, or 32 bits.

char\* destination - array to store string. Size of destination must not exceed 50 bytes of storage.

int base - numerical base used to represent the value as a string. Base must be between and including 2-36 (after 36, the number of unique symbols required to represent the digits exceeds the number of standard alphanumeric characters (0-9, A-Z) available)

## 3. doubly\_linked\_list.h

3.1 void insertFront(struct Node\*\* head, char\* data)

Author:

Nathan Mullins

Description

Inserts a new node at the front/head of a doubly linked list. If the list is not empty, moves nodes over and places the new node at the beginning. Returns nothing.

Parameters

struct Node\*\* head - points to first/front node within a doubly linked list. head must be a pointer to a pointer

char\* data - array of data to be placed in the front node. data must be of type char\* and data length must be less than the length of the buffer.

3.2 void insertAfter(struct Node\* prev\_node, char\* data)

Author:

Nathan Mullins

Description

Inserts a new node after the specified node prev\_node passed into the function into the doubly linked list. If the previous node is null/does not exist, it returns immediately from function. Otherwise, returns nothing.

Parameters

struct Node\* prev\_node - points to the node that the new node will be placed after.  
prev\_node must be a pointer of type struct.

char\* data - array of data to be placed in a new node. data must be of type char\* and data length must be less than the length of the buffer.

3.3 void insertEnd(struct Node\*\* head, char\* data)

Author:

Nathan Mullins

Description

Inserts a new node at the end of the doubly linked list. If the list is empty, places the new node at the front of the list. Otherwise, traverses through the list until the next node is null, and places the new node in.

Parameters

struct Node\*\* head - points to the node that will be placed at the end of the list. must be a pointer of type struct.

char\* data - array of data to be placed in a new node. data must be of type char\* and data length must be less than the length of the buffer.

3.4 deleteNode(struct Node\*\* head, struct Node\* del\_node)

Author:

Nathan Mullins

## Description

Deletes the specified node `del_node` from the doubly linked list. If the list is empty or the `del_node` is empty, immediately returns because deletion is not possible. Otherwise, removes `del_node` and corrects pointers of previous and next nodes in the list.

## Parameters

`struct Node** head` - points to first/front node within a doubly linked list. `head` must be a pointer to a pointer

`struct Node* del_node` - points to the node that will be deleted. must be a pointer of type `struct`.

## 3.5 struct Node

### Author:

Nathan Mullins

## Description

Structure/layout of each node in a doubly linked list. Each node will contain a pointer to the next node, a pointer to the previous node, and data associated with the node.

## Parameters

`char* data` - array of data associated with a node. `data` must be of type `char*` and `data` length must be less than the length of the buffer.

`struct Node* next` - points to the next node in a list. must be a pointer.

`struct Node* prev` - points to the previous node in a list. must be a pointer.

## 4. comhand.h

### 4.1 void comhand(void)

### Author:

Ian Jackson

## Description

Responsible for handling user input from the terminal and executing various commands based on that input. It begins by displaying a welcome message, listens for user input indefinitely, parses the input to identify specific commands, and then executes the corresponding command or displays an error message if the command is not recognized. The supported

commands include "version," "shutdown," "help," "getdate," "setdate," "gettime," "settime," and "clear," each with its own specific functionality.

Parameters

N/A

## 5. help.h

5.1 void help(const char \*cmd)

Author:

Ian Jackson

Description

Provides usage instructions/ description for all available commands given to the user.

Does not return anything.

Parameters

const char\* cmd - provides specific instruction/description about the parameter if it is a command that exists. Parameter cmd must be of length smaller than the defined buffer size.

## 6. time.h

6.1 void get\_time(void)

Author:

Riley Stauffer

Description:

Displays the current time in the form HH:MM:SS. Note, the time is displayed in GMT by default

Parameters:

N/A

6.2 void get\_date(void)

Author:

Grant Stumpf

Description:

Displays the current date in the form MM-DD-YY.

Parameters:

N/A



### 6.3 void set\_time(const char \*command)

Author:

Riley Stauffer

Description

Changes the internal time of the system. Parses input into hours, minutes, and seconds and converts those into integers. Converts the integers from decimal to BCD. Disables interrupts and writes values directly to the time registers of the system. Re-enables interrupts.

Parameters

const char \*command - user input time. If the user writes "setdate HH:MM:SS" the time will be isolated and passed to the function.

### 6.4 int isValidTimeFormat(const char\* input)

Author:

Riley Stauffer

Description

Verifies user input of the set\_date(const char \*command) function. This function passes the truncated user command from comhand.c. The input will be verified as a valid input, HH:MM:SS if the length is 8, there are ":" in the second and fifth positions, HH<24, MM<60, and SS<60.

Parameters

const char\* input - user input time, automatically truncated if the time is preceded by "settime."

### 6.5 void set\_date(uint8\_t day, uint8\_t month, uint8\_t year)

Author:

Grant Stumpf

Description

Changes the internal date of the system by disabling and enabling interrupts and writing directly to the system's time registers. Converts inputs from decimal to BCD before writing to the registers.

Parameters

uint8\_t day - day (DD) parsed from user input, uint8\_t month - month (MM) parsed from user input, uint8\_t year - year (YY) parsed from user input.

6.6 int hexToDec(uint8\_t hex)

Author:

Grant Stumpf

Description

Converts an integer in binary coded decimal (BCD) to decimal. Returns decimal value integer.

Parameters

uint8\_t hex - input of BCD value before conversion

6.7 uint8\_t decToHex(int decimal)

Author:

Riley Stauffer

Description

Converts an integer in decimal to BCD. Returns uint\_8t of input in BCD.

Parameters

int decimal - input of decimal value before conversion

## 7. version.h

7.1 void version(void)

Author:

Grant Stumpf

Description

prints the most updated/current version of the MarauderOS along with the last compilation date. Function is called by the user in the terminal by typing “version” when running MarauderOS.

Parameters

version() takes no parameters and does not return anything.

## R2

### 8. process\_queues.h

#### 8.1 struct node

Author:

Nathan Mullins

Description

Structure/layout of each node in a queue formatted as a doubly linked list. Each node will contain a pointer to the next node, a pointer to the previous node, and a pointer to a pcb.

Parameters

struct node\* next - pointer to the next node in a queue

struct node\* prev - pointer to the previous node in a queue

struct pcb\* pcb - pointer to a pcb, which contains information about a process

#### 8.2 struct queue

Author:

Nathan Mullins

Description

Structure/layout of each of the 4 queues to separate processes (i.e. ready, suspended ready, blocked, suspended blocked). Queues are formatted as a doubly linked list, so they will have pointers to each end of the queue

Parameters

struct node\* front - pointer to the head/front node in a queue

struct node\* rear - pointer to the tail/rear node in a queue

#### 8.3 void enqueue\_pri(char\* status, struct pcb\* pcb)

Author:

Nathan Mullins

Description

Insert a pcb into a ready queue based on the associated priority and state of the process. Will traverse through the queue until the correct position is found. If priorities are equal, ordering is FIFO.

Returns nothing

Parameters

char\* status - state of the process (i.e. ready, suspended ready, blocked, suspended blocked)

struct pcb\* pcb - pointer to a pcb, which contains information about a process.

8.4 void dequeue(const char\* status, struct pcb\* pcb)

Author:

Nathan Mullins

Description

Find and remove a pcb from the correct queue using a pcb pointer and the state of the process.

Returns nothing

Parameters

const char\* status - state of the process to be removed (i.e. ready, suspended ready, blocked, suspended blocked)

struct pcb\* pcb - pointer to a pcb, which contains information about a process.

8.5 char\* search(struct pcb\* pcb\_ptr)

Author:

Nathan Mullins

Description

Searches for a process based on its state. Returns a char\* containing the processes execution and dispatching state (i.e. “ready not suspended”, “ready suspended”, etc.) or NULL if the process was not found.

Parameters

struct pcb\* pcb\_ptr - pointer to a pcb, which contains information about a process.

8.6 struct node\* create\_node(struct pcb\* pcb)

Author:

Nathan Mullins

Description

Creates an instance and allocates memory for a node. Returns the new node or NULL if there was an error creating the node.

Parameters

struct pcb\* pcb\_ptr - pointer to a pcb, which contains information about a process.

8.7 void enqueue\_reg(char\* status, struct pcb\* pcb)

Author:

Nathan Mullins

Description

Inserts a pcb into a blocked queue based on FIFO order. Returns nothing.

Parameters

char\* status - state of the process (i.e. ready, suspended ready, blocked, suspended blocked)  
struct pcb\* pcb - pointer to a pcb, which contains information about a process.

## 9. pcb.h

### 9.1 struct pcb

Author:

Nathan Mullins

Description

Structure/layout of the process control block. The pcb stores a processes context and points to that process.

Parameters

const char\* name\_arr; - pointer to the name of the process stored in the pcb

struct process \*process\_ptr - pointer to the process along with its context that is stored in the pcb.

### 9.2 enum state

Author:

Nathan Mullins, Ian Jackson

Description

Contains the 5 possible states (executing & dispatching) a process can be in. (i.e. READY\_NOT\_SUSPENDED, READY\_SUSPENDED, BLOCKED\_NOT\_SUSPENDED, BLOCKED\_SUSPENDED, RUNNING).

Parameters

READY\_NOT\_SUSPENDED - waiting to be executed

READY\_SUSPENDED - waiting to be loaded into memory and put in the ready not suspended

state

BLOCKED\_NOT\_SUSPENDED - waiting for resources or an event to end

BLOCKED\_SUSPENDED - waiting for resources/an event to end for a long period of time

RUNNING - currently in execution

### 9.3 struct process

Author:

Nathan Mullins

Description

Structure/layout of each process. Contains context/information such as its class, state, priority, etc.

#### Parameters

int pcb\_class - class = 0 means a user process, class = 1 means a system process  
enum state pcb\_state - state the process is currently in  
int pcb\_priority - an int between 0 (high pri) and 9 (low pri). 0 is the highest priority where 9 is the lowest priority.  
char pcb\_stack[1024] - memory space for the CPU stack of at least 1024 bytes.  
void\* stack\_ptr - pointer to current location in the stack  
struct process\* next\_process - pointer to related pcbs.

### 9.4 struct pcb\* allocate(void)

#### Author:

Nathan Mullins

#### Description

Dynamically allocates memory for the pcb, process pointer, and process name. Returns NULL if one of the memory allocations fails.

#### Parameters

N/A

### 9.5 int pcb\_free(void)

#### Author:

Ian Jackson

#### Description

Frees memory allocated by a process using the given sys\_free\_mem function. Returns 0 for success and non-zero for an error.

#### Parameters

N/A

### 9.6 struct pcb\* pcb\_setup(const char\* process\_name, int class, int priority)

#### Author:

Riley Stauffer

#### Description

Setup a pcb by verifying correct/allowed values for parameters, calling pcb\_allocate() to dynamically allocate memory for the pcb, and setting the pcbs contents using the parameters. Returns the pcb or NULL if an error has occurred.

#### Parameters

const char\* process\_name - pointer containing the name of the process  
int class - class (user or system) of a process passed as an int (0 or 1)  
int priority - an int between 0 (high pri) and 9 (low pri). 0 is the highest priority where 9 is the lowest priority.

9.7 struct pcb\* pcb\_find(const char\* process)

Author:

Nathan Mullins

Description

Searches through all queues for a process with the provided name. Returns the pcb associated with the name if it exists, or NULL if it could not be found.

Parameters

const char\* process - name of a process to search for in the queues

9.8 void pcb\_insert(struct pcb\* process)

Author:

Ian Jackson

Description

Insert a pcb into the correct queue using the enqueue functions. If the processes state is of type ready, then enqueue with priority will be used to insert. If the processes state is of type blocked, then enqueue with FIFO will be used to insert. If the process's state is running, nothing the process is not inserted. Returns nothing.

Parameters

struct pcb\* process - pointer to the pcb to insert

9.9 int pcb\_remove(struct pcb\* process)

Author:

Grant Stumpf

Description

Remove a pcb from the appropriate queue using the dequeue function. Returns 0 on success and 1 on error.

Parameters

struct pcb\* process - pointer to the pcb to be removed.

9.10 void create\_pcb(const char\* name, int class, int priority)

Author:

Ian Jackson

#### Description

Checks each passed parameter for validity. If all checks pass, calls `pcb_setup` and `pcb_insert` to create and insert the pcb into the correct queue.

#### Parameters

`const char* process_name` - pointer containing the name of the process.

`int class` - class (user or system) of a process passed as an int (0 or 1).

`int priority` - an int between 0 (high pri) and 9 (low pri). 0 is the highest priority where 9 is the lowest priority.

#### 9.11 void `delete_pcb(const char* name)`

##### Author:

Nathan Mullins

#### Description

Uses `find_pcb` to find the pcb to delete, checks if process can be deleted (i.e. if process is not of type system class), calls `pcb_remove` to delete the pcb, and `pcb_free` to free the memory the pcb was holding.

#### Parameters

`const char* name` - pointer to the name of the pcb to delete.

#### 9.12 void `block_pcb(const char* name)`

##### Author:

Ian Jackson

#### Description

This function is used to block a PCB with the given name. It removes the PCB from its current queue, changes its state to blocked (suspended or not), and inserts it back into the relevant queue.

#### Parameters

`const char* name`: a pointer to a string containing the name of the PCB to block.

#### 9.13 void `unblock_pcb(const char* name)`

##### Author:

Ian Jackson

#### Description

This function is used to unblock a previously blocked PCB with the given name. It follows a similar process to `block_pcb()` but changes the state to ready instead of blocked.

#### Parameters

`const char* name`: a pointer to a string containing the name of the PCB to unblock.



#### 9.14 void suspend\_pcb(const char\* name)

Author:

Ian Jackson

Description

This function suspends a PCB with the given name. It removes the PCB from its current queue, changes its state to suspended (ready suspended or blocked suspended), and inserts it back into the relevant queue. Note that system processes cannot be suspended.

Parameters

const char\* name: a pointer to a string containing the name of the PCB to suspend.

#### 9.15 void resume\_pcb(const char\* name)

Author:

Ian Jackson

Description

This function resumes a suspended PCB with the given name. It reverses the suspension process by changing the state from suspended to not suspended and placing the PCB back into the relevant queue. System processes cannot be suspended or resumed.

Parameters

const char\* name: a pointer to a string containing the name of the PCB to resume.

#### 9.16 void set\_pcb\_priority(const char\* name, int priority)

Author:

Riley Stauffer

Description

This function allows you to change the priority of a PCB with the given name. It first looks up the PCB by name and then updates its priority. It performs checks to ensure the validity of the name and priority value.

Parameters

const char\* name: a pointer to a string containing the name of the PCB to change the priority of.  
int priority: the level of priority assigned to the PCB, must be between 0-9.

#### 9.17 void show\_pcb(const char\* name)

Author:

Nathan Mullins, Ian Jackson

Description

This function is used to display information about a specific PCB identified by its name. It first looks up the PCB associated with the given name using `pcb_find(name)`. If the PCB exists, it extracts information such as process name, class, state, and priority, and then converts and prints this information.

Parameters

`const char* name`: pointer to string containing the name of the PCB to display.

9.18 `void show_ready(int status)`

Author:

Grant Stumpf, Ian Jackson

Description

This function displays information about PCBs that are in the ready state. It can also display a header for the information if the status parameter is set to 1. It iterates through the ready queue and suspended ready queue and calls `show_pcb()` for each PCB found.

Parameters

`int status`: an integer flag (0 or 1) indicating whether to display the header.

9.19 `void show_blocked(int status)`

Author:

Grant Stumpf, Ian Jackson

Description

Similar to `show_ready()`, this function displays information about PCBs that are in the blocked state. It can also display a header if the status parameter is set to 1. It iterates through the blocked queue and suspended blocked queue and calls `show_pcb()` for each PCB found.

Parameters

`int status`: an integer flag (0 or 1) indicating whether to display the header.

9.20 `void show_all(void)`

Author:

Grant Stumpf, Ian Jackson

Description

This function displays information about all PCBs, both in the ready and blocked states. It prints a header and then calls `show_ready(0)` and `show_blocked(0)` to display information about all PCBs.

Parameters

N/A

## R3

### 10. context\_switch.h

#### 10.1 struct context

Author:

Nathan Mullins

Description

Structure/layout of the context on a stack of a process. The context holds a processes general purpose registers, segment registers, and status registers.

Parameters

- int ESP - general purpose register, Stack Pointer (in the SS segment)
- int EAX - general purpose register, Accumulator
- int EBX - general purpose register, Pointer to data in DS
- int ECX - general purpose register, Counter for string/loop operations
- int EDX - general purpose register. I/O pointer
- int ESI - general purpose register, Pointer to data in segment pointed to by DS, source pointer
- int EDI - general purpose register, Pointer to data/destination in segment pointed to by ES
- int EBP - general purpose register, Pointer to data on the stack (in the SS)
- int DS - segment register, data segment
- int ES - segment register, data segment
- int FS- segment register (data register)
- int GS- segment register (data register)
- int SS - segment register, stack segment
- int EIP - status and control register, instruction pointer
- int CS - status and control register, code segment
- int EFLAGS - status and control register, flags register

#### 10.2 struct context\* sys\_call(struct context\* context\_ptr)

Author:

Nathan Mullins

Description

Handle all system calls and context switching. If the op code is anything other than IDLE or EXIT, it will return the current process's context and change the EAX register to -1. If the op code is IDLE and the instruction pointer of previous context is different than the instruction pointer of the current process (i.e. the process executing is different than the process executing on the last system call), then the

next process in the ready queue (if there is one) will be loaded to be run next. If the op code is EXIT, the currently running process will be deleted, and a new process (if it exists) will be loaded to run next.

Parameters

Context\_ptr - pointer to the context of the currently running process. The context will pass in the processes general purpose registers, segment registers, and status registers to be used.

## 11. pcb.h & pcb\_user\_commands.c continued

### 11.1 void yield(void)

Author:

All members

Description

Forces comhand to yield/give up the CPU. Allows processes in the ready queue to execute.

Parameters

N/A

### 11.2 void Load\_R3(void)

Author:

Nathan Mullins, Ian Jackson

Description

Creates/allocates memory for the 5 test processes in test\_processes.h, places them in the ready non-suspended queue, and initializes values for each processes context.

Parameters

N/A

## 12. sys\_call\_isr.s

### 12.1 sys\_call\_isr

Author:

Nathan Mullins, Ian Jackson

Description

System call interrupt handler that handles an interrupt service routine. Pushes a processes general purpose registers, segment registers, and status registers in the reverse order of the defined context struct,

calls `sys_call` to handle the possible context switch, and then pops the next processes registers in the order of the context struct. Returns from the interrupt service routine

Parameters

N/A

## R4

### 13. `pcb.h` and `pcb_user_commands.c` continued

#### 13.1 `void load_comhand(void)`

Author:

Ian Jackson, Nathan Mullins

Description

Load the command handler as a function into the ready queue by allocating memory, inserting in the appropriate position, and giving the newly created process context. Command handler is a system process that always runs at the highest priority.

Parameters

N/A

#### 13.2 `void load_sys_idle(void)`

Author:

Ian Jackson, Nathan Mullins

Description

Load a system idle process into the ready queue by allocating memory, inserting in the appropriate position, and giving the newly created process context. A sys idle process is a system process that runs at the lowest priority. It will only be dispatched if no other processes are available to execute

Parameters

N/A

### 14. `shutdown.h`

#### 14.1 `void shutdown(void)`

Author:

Ian Jackson

Description

Shutdown/shutoff the MarauerOS operating system by clearing the ready queue (removing all processes from the ready queue) and calling `sys_req(exit)`.

Parameters

N/A

## 15. alarms.h

15.1 void `alarm(char* formatted_time, char* message)`

Author:

Nathan Mullins, Grant Stumpf, Riley Stauffer

Description

This function is responsible for creating alarm processes in the system. It checks the validity of the time format using `isValidTimeFormat(formatted_time)` and, if the format is invalid, sends an error message. If the time format is valid and there is space for a new alarm process, it sets global variables for the message and time, and then creates a new PCB representing the alarm process. The function sets up the context for the alarm process, including its code segment, data segment, stack pointers, and instruction pointer, and initializes its registers and flags. If the system already has five alarm processes, it sends an error message indicating the maximum limit is reached and returns. Finally, it triggers the IDLE system request to let the system scheduler decide when to execute the alarm process. This function essentially allows the creation of alarm processes that execute a `print_message` function when the specified time is reached, effectively serving as alarm clocks in the system.

Parameters

`char* formatted_time`: string of time for alarm to ring, in form “hh:mm:ss”

`char* message`: string of whatever message the user wants to give when the alarm sounds

15.2 void `print_message(void)`

Author:

Nathan Mullins, Grant Stumpf, Riley Stauffer

Description

This function checks the system time each time the alarm gains CPU time, and waits for a specific time to match a user-defined time. Once the user-specified time matches the system time, it prints a global message and exits, effectively setting an alarm.

Parameters

N/A

15.2 int check\_time(void)

Author:

Nathan Mullins, Grant Stumpf, Riley Stauffer

Description

This function converts the system time into hexadecimal and compares it with a user-specified time provided in the global time array. If the system time matches or is later than the user-specified time, it returns 1; otherwise, it returns 0, effectively determining whether it's time to trigger the alarm. So, the return integer acts as a boolean since only 1 or 0 will be returned.

Parameters

N/A

15.3 int check\_running\_processes(void)

Author:

Nathan Mullins, Grant Stumpf, Riley Stauffer

Description

Check for which alarm is running, and return its corresponding number as an int, i.e. if alarm 3 is running 3 will be returned. This will iterate through the array containing alarm PCBs, and find which alarm exists and is not in the ready queue (i.e. if not in the ready and the pcb exists, it must be executing).

Parameters

N/A