Programmer's Manual

Table of Contents

serial.h

enum statusIndex

- Author: Jack Kile
- Determines if a device has executed, is executing or preparing to execute, or has not begun to be executed yet.
- **NoExecution:** No execution of the device has occurred
- **Executed:** The device has finished execution
- **InExecution:** The device is currently or beginning execution

enum allocStatusIndex

- Author: Jordan Dennison
- Determines if a device is open or closed to the serial port
- **Open:** The device is open to the serial port
- **Close:** The device is closed off from the serial port

Struct dcbStruct

- Author: Jack Kile
- Holds all the data of the device control block
- **Int allocStatus**: Status to see if resource is in use
- **Int currOperation:** Holds the current operation (Read, Write, Idle, Exit)
- **Char ringBuffer:** Circular queue of characters
- **Int bufferSize:** Size of valid data
- **Int ringIn:** Next empty slot in the buffer to fill
- **Int ringOut:** Next character to remove from the buffer
- **Int count:** Number of chars in the buffer
- **char* userReadBuffer:** User input buffer
- **Int userBufferSize:** Max buffer size
- **Int charsRead:** Number of characters transferred so far
- **char* userWriteBuffer:** User output buffer
- **Int numTransfer:** Number of bytes to write
- **Int numWrite:** Number of bytes written so far

Struct iocbStruc

- Author: Jack Kile
- Holds the contents of a I/O control block
- **Struct pcbStruct* associatedPCB:** PCB associated with the interrupt
- **Device dev:** Device to interact with
- **Int operationType:** Holds the current operation (Read, Write, Idle, Exit)
- **Int buffSize:** Buffer size
- **Struct iocbStruct* next:** Pointer to the next iocb in the queue

Struct iocbQueue

- Author: Jordan Dennison
- Waiting queue for iocb's
- **Struct iocbStruct* head:** Beginning iocb of the waiting queue
- **Struct iocbStruct* rear:** Ending iocb of the waiting queue

serial.c

int serial_poll(device dev, char *buffer, size_t len)

- Author Jacob Comer
- Stores characters input by users in the terminal and returns a buffer containing the string of characters entered. Handles special characters to ensure the terminal behaves as expected, e.g., backspace, delete, arrow keys, and enter. Function returns control after the user selects the enter key.
- **Parameter** device dev - Serial port to read data from
- **Parameter** char* buffer - A buffer to write data into as it is read from the serial port
- **Parameter** int len - the maximum number of bytes to read
- **Returns** the number of bytes read on success, a negative number on failure

void back_arrow(device dev)

- Author Jacob Comer
- Moves the cursor backward in the terminal. Handled by recognizing the escape sequence characters sent to device handler when key is pressed.
- **Parameter** device dev - Serial port to read data from

void foward_arrow(device dev)

- Author Jacob Comer
- Moves the cursor forward in the terminal. Handled by recognizing the escape sequence characters sent to device handler when key is pressed.
- **Parameter** device dev - Serial port to read data from

void delete(int* cursor_ptr, int* cur_ptr, char* buffer, device dev)

- Author Jacob Comer
- Deletes a character and moves the remaining string back. Handled by recognizing escape sequence characters sent to device handler when key is pressed.
- **Parameter** int* cursor_ptr - Pointer to cursor spot
- **Parameter** int* cur_ptr - Pointer to current length of string
- **Parameter** char* buffer - Buffer storing string
- **Parameter** device dev - The serial port to read data from

void backspace(int* cursor_ptr, int* cur_ptr, char* buffer, device dev)

- Author Jacob Comer

- Backspaces characters in the terminal. Handled by recognizing escape sequence characters sent to device handler when key is pressed.
- **Parameter** int* cursor_ptr - Pointer to cursor spot
- **Parameter** int* cur_ptr - Pointer to current length of string
- **Parameter** char* buffer - Buffer storing string
- **Parameter** device dev - The serial port to read data from

void insert(int* cursor_ptr, int* cur_ptr, char* buffer, device dev, char c)

- Author Jacob Comer
- Inserts a character in the middle of a string. Identifies the cursor is in the middle of the string by tracking its position and comparing it with the total length of the string. Both the array storing the buffer and the terminal output are adjust accordingly
- **Parameter** int* cursor_ptr - Pointer to cursor spot
- **Parameter** int* cur_ptr - Pointer to current length of string
- **Parameter** char* buffer - Buffer storing string
- **Parameter** device dev - The serial port to read data from
- **Parameter** char c - character passed from inb

int serial_open(device dev, int speed)

- Author: Jack Kile & Jacob Comer
- Description: Initializes the serial port associated with the specified device. Uses the device parameter to set the DCB to have actions performed on, before error checking and instantiating the Devices parameters. Uses the speed parameter to calculate the baud rate, which will be used to set the Divisor latch bits.
- Parameter device dev: Device of the port to open
- Parameter int speed: Speed used to calculate the baud rate
- Return int: Success on 0, error for anything else

int serial_close(device dev)

- Author: Jordan Dennison
- Description: Ends a session for a designated port. Executes error checking, returning error codes if errors occur, before setting the allocation status to CLOSE and disabling the correct interrupts associated with closing the device port.
- Parameter device dev: Device of the port to close
- Return int: 0 on success, error on anything else

Int serial_read(device dev, char *buf, size_t len)

- Author: Jordan Dennison
- Description: Begins the process of reading from the device passed in the parameters. Provides error checking and instantiation of the related DCB values of the device. Stores characters read in to the ring buffer. Event flag is then set to mark completion before continuing.
- Parameter device dev: Device of the port to read from

- Parameter char *buf: buffer of characters to be read in
- Parameter size_t len: length of the buffer
- Return int: 0 on success, any other value is an error

Int serial_write(device dev, char *buf, size_t len)

- Author: Jack Kile
- Description: Begins the process of writing characters to the specified port. Provides error checking before writing the first character of the buffer to the specified port. Enables the correct interrupts to continue writing the rest of the buffer.
- Parameter device dev: Device of the port to print to
- Parameter char *buf: buffer of characters to be written
- Parameter size_t len: length of the buffer
- Return int:

void serial_interrupt(void)

- Author: Jacob Comer
- Description: Disables interrupts depending on the calculated value obtained from performing an operation on the interruptID

Void serial_input_interrupt(void)

- Author Jay
- Description: Reads in the remaining characters from the device's buffer. Finishes the process started by serial read. After completing the read process, sets the correct registers and the event flag to completed before exiting.

Void serial_output_interrupt(void)

- Author: Jay
- Description: Writes the remaining characters from the device's buffer. Finishes the process started by serial write.After completing the write process, sets the correct registers and the event flag to completed before exiting.

Int is_empty_iocb(iocbQueue *q)

- Author: Jacob Comer
- Description: Checks whether or not the specified iocbQueue is empty.
- Parameter iocbQueue *q: Pointer to the queue to check the status of
- Return int:value representing whether or not the queue is empty

Void enqueue_iocb(iocbQueue* q, iocbStruct* iocb)

- Author: Jacob Comer
- Description: Takes the passed iocb and inserts it at the end of the specified IOCB queue
- Parameter iocbQueue* q: The queue to insert the iocbStruct into
- Parameter iocbStruct* iocb: the iocb to insert into the specified queue

iocbStruct* dequeue_iocb(iocbQueue* q)

- Author: Jacob Comer
- Description: Removes the first iocb in the specified queue and returns it in the return statement
- Parameter iocbQueue* q: Queue to remove the first value from
- Return iocbStruct*: the iocbStruct removed from the queue

int io_scheduler(device dev, pcbStruct* pcb, int op_code, char* buffer, int size)

- Author: Jack Kile
- Description: Determines whether or not an I/O operation should be performed immediately or inserted into a queue for later operation. Performs error checking on parameters before allocating memory to the iocb and setting its parameters. The iocb is then inserted into the queue for execution. The first value in the queue is then dequeued and inspected to see what operation needs to be performed. Once determined, the proper function is called before returning.
- Parameter device dev: Device driver used to write to the serial port
- Parameter: pcbStruct* pcb: PCB of the process associated wiht the interrupt
- Parameter: int op_code: The operation to be performed
- Parameter: char* buffer: Character buffer from the requestor
- Parameter: int size: Size of the characters contined in the buffer
- Return int: 0 on success, -1 on failure

void io_completion(void)

- Author: Jacob Comer
- Description: Performs the io completion sequence whenever the event flag is set in such a way to indicate that an operation has completed. Moves the pcb to the ready queue, sets the context, and frees memory associated with the iocb.

comhand.c

void titleCard()
- Author: Jordan Dennison
- Prints a string of characters to the terminal that form the teams logo

char* itoa(int value, char* strArray)
- Author: Jordan Dennison
- Takes the integer value passed to the function and calculates the ascii equivalent and stores it in a character array to be returned
- **Parameter** int value - the integer value to convert
- **Parameter** char* strArray - existing string array to store the converted value into
- **Return** character array containing the converted integer to ascii value

void comhand()
- Author: Jack Kile
- Takes user input from the terminal to determine what command is to be ran
- Checks if user input is an integer in a range from 1-13 to determine if it is valid user input, shows error message and reloads menu if so
- When 1 is entered, the help command is executed
- When 2 is entered, the version command is executed
- When 3 is entered, the get time command is executed
- When 4 is entered the set time command is executed
- When 5 is entered the get date command is executed
- When 6 is entered the set date command is executed
- When 7 is entered the initialize heap command is executed
- When 8 is entered the allocate memory command is executed
- When 9 is entered the free memory command is executed
- When 10 is entered the show free memory command is executed
- When 11 is entered the show allocated memory command is executed
- When 12 is entered the show all memory command is executed
- When 13 is entered the create alarm command is executed
- When 14 is entered the pcb commands menu is displayed
- When 15 is entered the shutdown command is executed
- Terminates once the shutdown command is selected

void version_command()
- Author: Jack Kile
- Prints the current version and compilation date of the mpx system
- Current version is formatted as Version #.#
- Compilation date is formatted as Month Day, Year

void gett_command()

- Author: Jordan Dennison
- Utilizes getTime function from rtc.c to return a struct containing the current values of hour, minute, and second from the corresponding registers. These values are then formatted into a human readable format and printed to the terminal.

void getd_command()

- Author: Jordan Dennison
- Utilizes getDate function from rtc.c to return a struct containing the current values of day, month, and year from the corresponding registers. These values are then formatted into a human readable format and printed to the terminal.

void sett_command()

- Author: Jordan Dennison
- Prompts the user to enter hour, minute, and second values. These values are converted into integers using atoi, stored into variables, and then individually validated using if statements based on expected value ranges. If the values are successfully validated, they are set into the registers using the setTime function from rtc.c. The time is then printed to the terminal using the gett_command.

void setd_command()

- Author: Jordan Dennison
- Prompts the user to enter day, month, and year values. These values are converted into integers using atoi, stored into variables, and then individually validated using if statements based on expected value ranges. If the values are successfully validated, they are set into the registers using the setDate function from rtc.c. The date is then printed to the terminal using the getd_command.

void help_command()

- Author: Jay Bhardwaj
- Prints the description for all the available commands implemented in MPX.

int shutdown_command(void)

- Author: Jay Bhardwaj
- Shuts down MPX if the user selects “1” and returns back to the menu if the user selects “0” when a confirmation prompt is displayed.
- **Returns** an integer value of either 1 or 0 depending on the input by the user

void pcbMenu (void)

- Author: Jordan Dennison
- Prints a list of functions for possible actions the user can perform relating to the PCB.
- Takes uses input and validates before running the corresponding function.
- When 1 is entered, the PCB help command is executed

- When 2 is entered, the Load Process function is executed
- When 3 is entered, the Load Suspended Process function is executed
- When 4 is entered, the Delete PCB function is executed
- When 5 is entered the Block PCB function is executed
- When 6 is entered the Unblock PCB function is executed
- When 7 is entered the Suspend PCB function is executed
- When 8 is entered the Resume PCB function is executed
- When 9 is entered the Set PCB Priority function is executed
- When 10 is entered the Show PCB function is executed
- When 11 is entered the Show Ready function is executed
- When 12 is entered the Show Blocked function is executed
- When 13 is entered the Show All function is executed
- When 14 is entered the Yield CPU function is executed
- When 15 is entered the menu is terminated and the user is returned to the main menu

void pcbMenu_help_command (void)

- Author: Jay Bhardwaj
- Prints a descriptions of all of the command options from the pcbMenu command

void deletePCB (void)

- Author: Jacob Comer
- Prompts the user to enter a name of the process they want to delete. Displays errors if it is an invalid process name or if it is a kernel level process. The pcb is then removed from its queue and the removed pcb's memory is freed.

void blockPCB (void)

- Author: Jack Kile
- Prompts the user to enter a name of a process they want to block. Users will return to the menu if an invalid name is entered. The process is then removed from its previous queue, the execution state is set to blocked, and then inserted into the blocked queue.

void unblockPCB (void)

- Author: Jack Kile
- Prompts the user to enter a name of a process they want to unblock. Users will return to the menu if an invalid name is entered. The process is then removed from its previous queue, the execution state is set to ready, and then inserted into the ready queue.

void suspendPCB (void)

- Author: Jacob Comer
- Prompts the user to enter a name of a process they want to suspend. Users will return to the menu if an invalid name is entered. The process is then removed from its previous queue, the suspension status is set to suspend, and then inserted into either the ready suspended or blocked suspended queue.

void resumePCB (void)

- Author: Jacob Comer
- Prompts the user to enter a name of a process they want to resume. Users will return to the menu if an invalid name is entered. The process is then removed from its previous queue, the suspension status is set to active, and then inserted into either the ready or blocked queue.

void setPcbPriority (void)

- Author: Jack Kile
- Prompts the user to enter a name of a process they want to change the priority of. Users will return to the menu if an invalid name is entered. The process is then removed from its queue, the priority is changed, and then inserted into the correct position of the same queue.

void pcbShow (void)

- Author: Jay Bhardwaj
- Prompts the user to enter the name of a PCB of their choice when pcbShow(0) is called; when pcbShow(1) is called, the function then prints all the processes in ready state. Similarly, when pcbShow(2) is called, the function then prints all the processes in the blocked state. Lastly, pcbShow(1) and pcbShow(2) are called together to show all the available processes.

void allocate(void)

- Author: Jack Kile
- Prompts the user to ender the size of memory they want to allocate (in hexadecimal). If memory is successfully allocated there will be a message displayed indicating so, and if it is unsuccessfully allocated an error message will be displayed.

void free(void)

- Author: Jack Kile & Jay Bhardwaj
- Prompts the user to enter the starting address of the block of memory they want to free (in hexadecimal). If memory is successfully freed there will be a message displayed indicating so, and if it is unsuccessfully allocated.

void show_mem(int func)

- Author: Jay Bhardwaj
- **Parameter** int func - Indicates which type of memory you would like to view. 0 for all memory blocks, 1 for free memory, and 2 for allocated memory.
- The function traverses the list of memory control blocks and prints out the starting address and size of each block. The type of memory to be displayed is determined by the parameter.

void init_heap(void)

- Author: Jordan Dennison
- The function prompts the user to enter the amount of bytes they would like to allocate for the heap. The maximum amount of bytes that can be allocated is 10,000 bytes, and if any amount higher than that is entered then an error message will be displayed. If the heap is successfully allocated a message indicating so will be displayed. If it is unsuccessful then an error message is displayed.

RTC.h

enum rtcIndex

- Author: Jordan Dennison
- Data type to create an enum specific to the RTC indexes. Depending on the value entered, will utilize the corresponding RTC register value index for the data specified.
- **Seconds -** Contains the RTCs index value for seconds
- **Minutes -** Contains the RTCs index value for minutes
- **Hours -** Contains the RTCs index value for hours
- **DayOfWeek -** Contains the RTCs index value for the day of the week
- **DayOfMonth -** Contains the RTCs index value for day of the month
- **Month** - Contains the RTCs index value for month
- **Year -** Contains the RTCs index value for year

struct timeStruct

- Author: Jordan Dennison
- Data structure to create an object to store values relating to the time (ex: hour, minute, second)
- **hour -** integer value to store the hour of a timeStruct
- **minute -** integer value to store the minute of a timeStruct
- **second -** integer value to store the second of a timeStruct

struct dateStruct

- Author: Jordan Dennison
- Data structure to create an object to store values relating to the date (ex: day, month, year)
- **day -** integer value to store the given day of a dateStruct
- **month -** integer value to store the given month of a dateStruct
- **year -** integer value to store the given year of a dateStruct

RTC.c

int bcdToD(int bcd)

- Author: Jordan Dennison
- Converts the given Binary Coded Decimal value to a decimal value
- **Parameter** int bcd - the Binary Coded Decimal value to be converted
- **Returns** the converted decimal value equivalent to the entered binary coded decimal value

int dToBCD(int decimal)

- Author: Jordan Dennison
- Converts the given decimal value to a Binary Coded Decimal
- **Parameter** int decimal - the decimal value to be converted
- **Returns** the converted binary coded decimal value equivalent to the entered decimal value

int read(rtcIndex index)

- Author: Jordan Dennison
- Uses the function outb to write the passed RTC index into the index register. This value is then returned using the inb function to read the data register. (Can be used to return the values from time or date registers in the RTC)
- **Parameter** rtcIndex index - the index value of the register to be read from
- **Returns** the value stored in the passed rtc index

void write(rtcIndex index, int value)

- Author: Jordan Dennison
- Disables interrupts, followed by writing the passed index value to the index register using outb. The new value for the index is then written into the correct register using outb. After this is written, the interrupts are then reenabled.
- **Parameter** rtcIndex index - The index of the value to be changed
- **Parameter** int value - The new value for the index to be changed to

void setTime(int hour, int minute, int second)

- Author: Jordan Dennison
- Utilizes the write function to set the hour, minute, and second registers in the RTC. These values will first have to be converted from decimal values to binary coded decimal to make them compatible with the registers. This will be completed using the dToBCD function. After the completion the new time will be set.
- **Parameter** int hour - value to change the hour index to
- **Parameter** int minute - value to change the minute index to
- **Parameter** int second - value to change the second index to

void setDate(int day, int month, int year)

- Author: Jordan Dennison
- Utilizes the write function to set the day, month, and year registers in the RTC. These values will first have to be converted from decimal values to binary coded decimal to make them compatible with the registers. This will be completed using the dToBCD function. After the completion the new date will be set.
- **Parameter** int day - value to change the day index to
- **Parameter** int month - value to change the month index to
- **Parameter** int year - value to change the year index to
- **Returns**

timeStruct getTime()

- Author: Jordan Dennison
- Creates a new timeStruct to store the current values of hour, minute, and second. This will be done using the read function to read the corresponding registers for each of these values. These values will then need to be converted using the bcdToD function to convert the binary coded decimal values to decimal before they are stored into the struct. This struct is then returned.
- **Returns** timeStruct containing the current register values for hour, minute, and second

dateStruct getDate()

- Author: Jordan Dennison
- Creates a new dateStruct to store the current values of day, month, and year. This will be done using the read function to read the corresponding registers for each of these values. These values will then need to be converted using the bcdToD function to convert the binary coded decimal values to decimal before they are stored into the struct. This struct is then returned.
- **Returns** dateStruct containing the current register values for day, month and year

pcb.h

enum execIndex

- Author: Jacob Comer
- Enum for execution states

enum disIndex

- Author: Jacob Comer
- Enum for suspension states

enum classIndex

- Author: Jacob Comer
- Enum for classes (User vs Kernel)

struct pcbStruct

- Author: Jacob Comer
- Struct representing a PCB. Stores the PCB's name, class, execution state, suspension state, priority, stack, and stack pointer. The PCB struct behaves as a node in the queue, so the struct also stores a pointer to the next PCB.

pcb.c

pcbStruct* pcb_allocate(void)

- Author: Jacob Comer
- Allocates memory for a new PCB. Initializes stack to binary zero and initializes stack pointer to last byte of stack.
- **Returns** pcbStruct* - pointer to newly allocated PCB.

Int pcb_free(struct pcb*)

- Author: Jordan Dennison
- Takes a pointer to a PCB to be the PCB to free from memory
- Utilizes sys_free_mem() to free all memory related to the PCB
- **Returns** integer value indicating success or failure of freeing memory
    - 0 on success
    - Non-Zero on failure

struct pcb* pcb_setup(const char*, int, int)

- Author: Jay Bhardwaj
- Uses pcb_allocate to initialize a process with data and sets ot ready, not suspended.
- **Returns** pcbStruct* - pointer to a initialized PCB.

struct pcb* pcb_find(const char* pcbName)

- Author: Jordan Dennison
- Takes the name of the PCB to be searched for as a parameter
- Iterates through each of the four queues (ready_queue, blocked_queue, ready_Suspended, and blocked_Suspended) using strcmp() to compare the PCB nodes name to the name to be search for until it has iterated through every queue, or has found the desired node.
- **Parameter** const char* pcbName - The name of the PCB to be searched for
- **Returns** Pointer to the existing PCB node in the queue if found, otherwise return NULL

void pcb_insert(pcbStruct* pcb_in)

- Author: Jack Kile
- Checks the given pcb for its execution state (ready or blocked) and its suspension status (suspended or active). Based on those results it will put in one of the four queues: ready_queue, blocked_queue, ready_Suspended, or blocked_Suspended. The helper function insert_help() is called once it is determined what queue the pcb should be inserted into.
- **Parameter** pcbStruct* pcb_in - pcb to be inserted into its appropriate queue

void insert_help(struct Queue* q, pcbStruct* pcb_in)

- Author: Jacob Comer
- Acts as a helper function for pcb_insert. The function takes a queue and pcb as input and handles the insertion of the pcb in priority queues. PCBs are placed in the queue based on their priority, higher priorities PCBs are placed ahead of lower priority PCBs. PCBs of the same priority are inserted into the queue FIFO.
- **Parameter** Queue *q - queue to insert PCB into
- **Parameter** pcbStruct* pcb_in - PCB to be inserted into queue

int pcb_remove(pcbStruct* pcb_in)

- Author: Jacob Comer
- Finds the queue the PCB is stored using conditional logic based on the PCB's execution state and suspended state. Pcb_remove then calls remove_help which removes the PCB from the appropriate queue.
- **Parameter** pcbStruct* pcb_in - PCB to remove from queue
- **Returns** an integer representing whether or not the PCB was successfully removed. A return value of 0 represents a successful removal, otherwise a 1 represents a failure to remove the PCB.

int remove_help(struct Queue* q, pcbStruct* pcb_in)

- Author: Jacob Comer
- Serves as a helper function to pcb_remove. The function receives as input the PCB to remove and the correct queue that the PCB is to be removed from. The function then removes the PCB from the queue and handles the necessary pointer reassignment to maintain the queue.
- **Parameter** struct Queue* q - queue where PCB is to be removed
- **Parameter** pcbStruct* pcb_in - PCB to be removed from queue
- **Returns** an integer representing whether or not the PCB was successfully removed. A return value of 0 represents a successful removal, otherwise a 1 represents a failure to remove the PCB.

struct Queue* getProcess(int type)

- Author: Jay Bhardwaj
- Takes integer as a parameter to determine which of the four queues to return a pointer to.
- When 1 is selected, returns pointer to the ready_queue
- When 2 is selected, returns pointer to the blocked_queue
- When 3 is selected, returns pointer to the ready_suspended queue
- When 4 is selected, returns pointer to the blocked_suspended queue
- **Parameter** int type - integer to designate which queue pointer to return
- **Returns** pointer to the corresponding queue

queue.h

struct Queue

- Author: Jacob Comer
- Struct representing a queue. Stores a pointer to the front and rear PCBs in the queue.

queue.c

int is_empty(Queue* q)

- Author: Jacob Comer
- Checks if a given queue is empty. Returns a 1 if a given queue is empty and a 0 otherwise.
- **Parameter** Queue* q - Queue to be checked if empty or not
- **Returns** an integer representing whether or not the queue is empty

void enqueue(Queue* q, pcbStruct* pcb)

- Author: Jacob Comer
- Inserts a given PCB (node-like structure) at the end of a given queue. If the function is empty it simply sets the given PCB as the front and rear or the queue. Otherwise it iterates through to the end of the queue and inserts the PCB at the rear.
- **Parameter** Queue* q - Queue to insert PCB into
- **Parameter** - pcbStruct* pcb - pcb to insert into end of queue

pcbStruct* dequeue(Queue* q)

- Author: Jacob Comer
- Removes the PCB found at the front of a given queue and returns it. If the queue is empty, the function returns NULL.
- **Parameter** Queue* q - Queue to remove PCB from
- **Returns** pcbStruct* - PCB that was removed from front of queue

void enqueue_alarm (alarm_Queue* q, alarmStruct* alarm)

- Author : Jacob Comer
- Inserts a given alarm into the end of the alarm queue
- **Parameter** alarm_Queue* q - Queue to insert alarm into
- **Parameter** alarmStruct - alarm to be inserted into queue

void dequeue_alarm(alarm_Queue* q, alarmStruct* alarm)

- Author : Jacob Comer
- Removes a specified alarm from the alarm queue
- **Parameter** alarm_Queue* q - Queue to insert alarm into
- **Parameter** alarmStruct - alarm to be removed from the queue

void is_alarm_empty(alarm_Queue* q)

- Author : Jacob Comer
- Checks if a given alarm queue is empty
- **Parameter** alarm_Queue* q - Queue to insert alarm into
- **Returns** integer where 1 represents an empty queue and 0 represents otherwise

sys_call.h

struct context

- Author: Jacob Comer
- Contains int variables that hold the values of the CPU registers. The values are populated once a currently running process is yielded and the context is saved within this struct.

sys_call.c

struct context *sys_call(struct *context currContext)

- Author: Jacob Comer
- Handles context switches for processes.
- If the currently running process is set to idle, its context is stored in its pcb stack and the pcb is inserted into the ready queue.
- If the currently running process is set to exit, the process's memory is freed.
- The next process in the ready queue is loaded into memory via the register values stored in the pcb's stack.
- In the case where a process is set to idle and there are no other processes in the ready queue to be ran, the current process is returned to continue running. If there are no other processes in the queue and the current process is exiting, the first context is returned.
- Sys_req utilizes sys_call each time it executes. The passed context's eax register determines if the process is to be set to idle, exit, or neither. In the case that a context switch is not needed, the eax value is set to -1, and returned to sys_req to perform other operations.

sys_call_isr

- Author: Jacob Comer
- Set of assembly instructions that pushes the currently running processes register values to the CPU stack, calls sys_call where a context switch is performed, then pops the new process's context off the CPU stack and into the CPU to execute.

r3_commands.c

void CPU_yield(void)

- Author: Jack Kile
- The function sys_req() is called with an op code of IDLE. This will execute any processes that are in the ready queue.

void load_process(void)

- Author: Jay Bhardwaj and Jordan Dennison
- Loads four non-suspended processes into the ready queue. Four processes are initialized with a correlating context. The context's register values are all populated with different values. The register cs is set to 0x08. The registers ds, es, fs, gs, and ss are all set to 0x10. The ebp is set to an int cast of the correlating process' stack. The eip is set to an int cast of its correlating process. The eflags are set to 0x0202. The remaining registers eax, ebx, ecx, edx, esi, and edi are set to 0x00. Then the process is inserted into the ready queue. This is repeated for each process.

void load_suspended_process(void)

- Author: Jordan Dennison
- Loads one suspended process into the ready queue. The process is initialized with a correlating context. The context's register values are all populated with different values. The register cs is set to 0x08. The registers ds, es, fs, gs, and ss are all set to 0x10. The ebp is set to an int cast of the correlating process' stack. The eip is set to an int cast of its correlating process. The eflags are set to 0x0202. The remaining registers eax, ebx, ecx, edx, esi, and edi are set to 0x00. Then the process is inserted into the ready queue.

alarm.h

struct alarmStruct

- Author: Jacob Comer
- Struct representing an alarm in the alarm queue. Holds the message to be displayed to the user, two structs for the given date and time, and a pointer to the next alarm in the queue.

alarm.c

void create_alarm(void)

- Author: Jacob Comer
- Creates a new user defined alarm to be inserted into the the alarm queue. The function additionally creates a pcb to be inserted into the ready queue if one does not already exist. The pcb is given a context to store in its stack, and its instruction pointer points to check_alarm.

void check_alarm(void)

- Author: Jack Kile
- Checks to see if the alarm's given date and time has surpassed the system's current date and time. If so the process will display the given message from the user. If that is not the case, then the process will become idle and eventually be checked again until that alarm is reached. If the alarm queue is empty at the beginning, or empty after traversing the alarm queue, then the process will exit and control will be given to the process with the highest priority.

void set_alarm_time(void)

- Author: Jordan Dennison
- Prompts the user to enter the desired time (hour, minute, and second) and date (day, month, and year) for the alarm to be inserted into the alarm struct. These values are then validated using military time format and standard date format, prompting the user again if they need to re-enter, before being stored into the alarm struct, specifically within the contained timestruct and datestruct in the alarm struct.

string.c

void printf(char text[], int size)

- Author: Jordan Dennison
- **Parameter** char[] text - The character array the user would like to print to the terminal
- **Parameter** int size - The size of character array to be printed to the terminal
- Passes the text and size parameters to the function sys_req(WRITE) in the appropriate parameter locations as a means of printing a character array to the terminal.

void printfRED(char text[], int size)

- Author: Jordan Dennison
- **Parameter** char[] text - The character array the user would like to print to the terminal
- **Parameter** int size - The size of character array to be printed to the terminal
- Passes the text and size parameters to the function sys_req(WRITE) in the appropriate parameter locations as a means of printing a character array to the terminal. This time in the color red!

mem.h

struct mcbStruct

- Author: Jack Kile
- Struct representing a memory control block in the memory queue. It holds the starting address of a block of memory, the size of the block of memory, a type indicating whether the memory is free or allocated, a pointer to the next block in memory, and a pointer to the previous block in memory.

enum memIndex

- Author: Jack Kile
- Enum for memory types, free and allocated

mem.c

void initialize_heap(size_t size)

- Author: Jack Kile
- **Parameter** size_t size - Size of main memory to allocate to the memory manager
- The function allocates a large block of free memory to the memory manager for our system to use, essentially determining how much main memory we have. The size is determined by the parameter and this large block, after being initialized as a mcb, of free memory is added into our memory queue.

void *allocate_memory(size_t size)

- Author: Jacob Comer
- **Parameter** size_t size - Size of memory to allocate to a block
- **Return** void* - Pointer to the starting address of the allocated block
- The function allocates a block of memory from our memory queue, based on the size given by the parameter. The list is traversed until there is a free block large enough to allocate the memory desired.

int free_memory(void *start)

- Author: Jordan Dennison
- **Parameter** void *start - Starting address of the block to be freed

- **Returns** int - Integer determining if memory was successfully freed, 1 if successful and -1 if unsuccessful
- The function traverses the memory queue to find an allocated block that matches the starting address given by the parameter. If found, the memory will be freed, and potentially merged with its previous and/or next blocks of memory if they are also free blocks of memory.

int htoi(char *hexNumber)

- Author: Jay Bhardwaj
- **Parameter** char *hexNumber - Hexadecimal character pointer to be converted into an integer
- **Return** int - Integer converted from hexadecimal
- Traverses each character in from the hexadecimal string and calculates its integer value.

char* itoh(int intNum, char* strArray)

- Author: Jordan Dennison
- **Parameter** int intNum - Integer value to be converted into hexadecimal
- **Parameter** char* strArray - Character pointer to store the converted integer
- **Return** char* - The converted hexadecimal character pointer
- Initializes the parameter strArray's first 8 characters to represent 0x0000000. After initialization the characters are assigned different values based on the calculations within the function.