nexTTeam mpx Module 5

Programmer's Manual

Team Members

Teddy Hale
Jessica Hammersla
Mya Junkin
Philip Wenger

Table of Contents

Module 1

commandhandler.h

Purpose: Display current version for the MPX project.
Parameters: No parameters.
Preconditions: The user successfully enters the command for version.
Postconditions: The method displays the current version for the project.
Exceptions: No exceptions.
Explanation of Code: The method prints and displays the current version of the program.
void version();

Purpose: Display the name and a detailed description of all possible commands the user can execute.
Parameters: No parameters.
Preconditions: The user successfully enters the command for help.
Postconditions: The name and description for each function that a user can call is displayed.
Exceptions: No exceptions.
Explanation of Code: The method writes the name and a detailed description for every method that the user can call.
void help();

Purpose: Shutdown the MPX and terminate. When shutdown is entered, the user has the option to confirm if the user would like to shutdown.
Parameters: The user will be given a chance to confirm the shutdown. If the user says yes, the program will shutdown. If the user says no, the program will not shutdown and will return for the user to continue entering commands. If the user does not enter yes or no, the program will say that it is not a valid option.
Preconditions: The user will need to successfully enter the command to shutdown.
Postconditions: The program will return to the command handler if the shutdown is not confirmed. It will end program execution if it is confirmed to shutdown.
Exceptions: The program will return to command handler if something other than yes or no is entered in the command handler.
Explanation of Code: The function prints the question asking if the user would like to shutdown. The system then reads in the user input. If the user enters yes, the program will shutdown and return a value of 1. If the user enters not or any other response, the program will not shutdown and return a value of 0.
int shutdown();

Purpose: The method will gather the information needed to be able to set the time in the setTime method.
Parameters: The method asks for the time written as hh:mm:ss that will be sent to setTime.
Preconditions: The user will need to successfully enter the command setTime to start collecting

information needed for setTime.
Postconditions: The collected information in the wrapper will be sent to the setTime method.
Exceptions: If the information entered into the wrapper method is not formatted as hhmm:ss, then the method will terminate.
Explanation of Code: The prompt to enter the time as hh:mm:ss is displayed. The method then reads in the time entered by the user, first hour, then minute, then second. These variables are then sent to the function setTime to physically set the time.
void settimeWrapper();

Purpose: The method will gather the information needed to be able to set the time in the setDate method.
Parameters: The method asks for the time written as mm/dd/yy that will be sent to setDate.
Preconditions: The user will need to successfully enter the command setDate to start collecting information needed for setDate.
Postconditions: The collected information in the wrapper will be sent to the setDate method.
Exceptions: If the information entered into the wrapper method is not formatted as mm/dd/yy, then the method will terminate.
Explanation of Code: The prompt to enter the time as mm/dd/yy is displayed. The method then reads in the time entered by the user, first month, then day, then year. These variables are then sent to the function setDate to physically set the date.
void setdateWrapper();

Purpose: The comhand reads in methods entered by the user and transfers control to the necessary method.
Parameters: There is a buffer that reads in the input from the user for what method to send the information to.
Preconditions: The MPX project must be successfully started.
Postconditions: The command handler transfers control to the method needed by the user.
Exceptions: If none of the possible commands are entered by the user, an error message is displayed.
Explanation of Code: A pointer to each method that a user can call is created. The welcome menu is the displayed, asking for the user to enter an available command. The available commands are then displayed. The method then reads in the command entered by the user. The entered command is then entered into history. If the user enters shutdown, the command handler will handle if the user confirms that they want to quit. If a different valid command is entered, the pointer to the correct method is called. If the user enters a not valid command, a message saying there is no matching command is displayed.
int comhand();

date.h

Purpose: The method will retrieve the system date.
Parameters: The method works with the month, day, and year registers to get the date.

Preconditions: getdate needs to be entered by the user.
Postconditions: The method will display the system date.
Exceptions: No exceptions.
Explanation of Code: The month is retrieved from register 78, the day from register 77, and the year from register 79. The values from the registers are converted from BCD to integers, and then from integers to ASCII. The date is then displayed to the user.
void getdate();

Purpose: The method sets a new value for the month, day, and year.
Parameters: The month, day and year values entered by the user are sent to setDate.
Preconditions: The information entered into setDateWrapper needs to be entered in the correct format to send the information to setDate.
Postconditions: The new date is set and a success message is displayed to the user.
Exceptions: If any of the month, day, or year values are not valid, the method will display an error message and not set the date.
Explanation of Code: The method accepts the month, day, and year entered by the user in the setDateWrapper. If any of the values are not valid dates, an error will be displayed. If there are no errors, the three integers are converted to BCD. Once they are converted, the year is set to register 79, the month is set to register 78, and the day is set to register 77. If successfully set, a success message is displayed.
void setDate(int month, int day, int year);

polling.h

Purpose: The polling method collects the user input from the keyboard.
Parameters: A buffer reads in the input from the keyboard. A count of how many characters are in the buffer is kept track of.
Preconditions: The MPX program needs to be started.
Postconditions: The character entered will be processed by the method, and the character will be printed if necessary.
Exceptions: Some characters, such as enter, are only processed and are not displayed.
Explanation of Code: The buffer reads in the input from the user keyboard. If a character or number is entered, the character or number is displayed. If backspace is entered, the previous character is removed. If the delete is entered, the next character is removed. If enter is pressed, the line is accepted and a new line is started.
*int poll(char * buffer, int * count);*

time.h

Purpose: The method will retrieve the system time.
Parameters: The method works with the hour, minute and second registers to get the time.
Preconditions: gettime needs to be entered by the user.
Postconditions: The method will display the system time.

Exceptions: No exceptions.
Explanation of Code: The hour is retrieved from register 74, the minute from register 72, and the second from register 70. The values from the registers are converted from BCD to integers, and then from integers to ASCII. The time is then displayed to the user.
void gettime();

Purpose: The method converts a BCD to an integer.
Parameters: The method takes in an unsigned char that will be turned into an integer.
Preconditions: An unsigned char is sent to the method.
Postconditions: The unsigned char is converted to an integer and the integer is returned.
Exceptions: No exceptions.
Explanation of Code: The tens digits of the unsigned char value is and-ed with 0xF0, and then is multiplied by ten. The ones digit is then and-ed with 0x0F and is added to the previously found tens digit. The integer is then returned.
int bcdToInt(unsigned char value);

Purpose: The method sets a new value for the hour, minute, and second.
Parameters: The hour, minute, and second values entered by the user are sent to setTime.
Preconditions: The information entered into setTmeWrapper needs to be entered in the correct format to send the information to setTime.
Postconditions: The new time is set and a success message is displayed to the user.
Exceptions: If any of the hour, minute, or second values are not valid, the method will display an error message and not set the time.
Explanation of Code: The method accepts the hour, minute, and second entered by the user in the setTimeWrapper. If any of the values are not valid dates, an error will be displayed. If there are no errors, the three integers are converted to BCD. Once they are converted, the hour is set to register 74, the minute is set to register 72, and the second is set to register 70. If successfully set, a success message is displayed.
void setTime(int hours, int minutes, int seconds);

Purpose: The method converts an integer to a BCD.
Parameters: An integer is accepted by the method.
Preconditions: An integer must be the type of variable sent to the method.
Postconditions: An unsigned char that is a BCD is returned.
Exceptions: No exceptions.
Explanation of Code: The ones digits for the integer is done using mod 10. The integer is then divided by ten. The tens digit is found from the now divided integer using mod 10. The unsigned char is created by shifting the tens digit by four and adding the ones digit. The unsigned char is returned
unsigned char intToBcd(int data);

Module 2

commandhandler.h

Purpose: The create PCB wrapper collects the information necessary to setup and insert the PCB into the correct queue.
Parameters: The method asks for the name, priority, and class of the PCB to be set up.
Preconditions: The user will need to successfully enter the command to createPCB.
Postconditions: The PCB will be setup and inserted into the correct queue and a message saying it has been successful will be displayed.
Exceptions: If an invalid name, priority, or class is entered, an error message will be displayed and the PCB will not be created.
Explanation of Code: The method asks for a PCB name, and if the name is valid and not previously used, the PCB will be name will be set. The method then asks for the priority, and if the priority is valid, the PCB priority will be set. The method finally asks for the class, and if the class is valid, the PCB class will be set. The method will then be set up with the values set earlier, and the new PCB will then inserted into the proper queue. If successful, a success message is displayed. If at any point the method fails, an error message displays and the method ends.
void createPCBWrapper();

Purpose: The method will remove a PCB based of the name entered by the user.
Parameters: The method reads in the name entered by the user.
Preconditions: The user will need to enter the command deletePCB.
Postconditions: The PCB entered by the user will be removed from its queue and its memory will be freed.
Exceptions: If there is no PCB with the name entered by the user, then an error message saying that PCB does not exist will be displayed.
Explanation of Code: The method asks for the name of the PCB it wants to delete. If the PCB with that name does not exist, an error message is displayed. If it does exist, then the PCB will be removed. The memory for the found PCB will then be freed. A success message saying the PCB has been successfully deleted is displayed.
void deletePCBWrapper();

Purpose: The method will change the priority for a PCB entered by the user.
Parameters: The method reads in the PCB name and the priority for the new PCB from the user.
Preconditions: The user needs to enter the command setPriority.
Postconditions: The method will send the priority off to setPriority to set the new priority.
Exceptions: If the PCB entered by the user does not exist, an error message is displayed saying there is no PCB with that name. If the new priority number is not valid, an error message is displayed.
Explanation of Code: The method asks for the name of the PCB that the user wants to change the priority of. If the PCB does not exist, an error message is displayed and the method ends. If

the PCB exists, then the method asks for the new priority of the PCB. If the priority entered is not valid, an error is displayed and the method ends. If the priority is valid, the PCB and the new priority are sent to setPriority.
void setPriorityWrapper();

Purpose: The method will collect the info to display one PCB's info.
Parameters: The method reads in the PCB name from the user,
Preconditions: The user must enter the command showPCB.
Postconditions: The PCB name will be sent off to showPCB.
Exceptions: If the name is too long, an error message will be displayed.
Explanation of Code: The method asks for the name of the PCB to be displayed. If the name is valid, the PCB is sent off to showPCB. If the name is not valid, an error message will be displayed.
void showPCBWrapper();

Purpose: The method will collect the info to block a PCB.
Parameters: The method reads in the PCB name from the user,
Preconditions: The user must enter the command block.
Postconditions: The PCB name will be sent off to block.
Exceptions: If the name is too long, an error message will be displayed. If the PCB cannot be found, an error message will be displayed saying the PCB does not exist.
Explanation of Code: The method asks for the name of the PCB to be blocked. If the name is valid and the PCB is found, the PCB is sent off to block. If the name is not valid, an error message will be displayed. If the PCB is not found, an error message will be displayed.
void blockWrapper();

Purpose: The method will collect the info to unblock a PCB.
Parameters: The method reads in the PCB name from the user,
Preconditions: The user must enter the command unblock.
Postconditions: The PCB name will be sent off to unblock.
Exceptions: If the name is too long, an error message will be displayed. If the PCB cannot be found, an error message will be displayed saying the PCB does not exist.
Explanation of Code: The method asks for the name of the PCB to be unblocked. If the name is valid and the PCB is found, the PCB is sent off to unblock. If the name is not valid, an error message will be displayed. If the PCB is not found, an error message will be displayed.
void unblockWrapper();

Purpose: The method prints the history of commands entered by the user.
Parameters: The method uses the thisHistory array to see previously entered commands.
Preconditions: The user needs to enter the command history.
Postconditions: The previous ten entered user commands are displayed.
Exceptions: No exceptions.
Explanation of Code: The method cycles through the last ten commands entered. The method

entered at the current index is printed. The index is incremented, and the next method in the history array is displayed. This continues for ten times, as long as the value at the index is not null. If the index is at 10, and ten commands have not been printed yet, the index is set back to zero.
void historyWrapper();

Purpose: The method will collect the info to suspend a PCB.
Parameters: The method reads in the PCB name from the user,
Preconditions: The user must enter the command suspend.
Postconditions: The PCB name will be sent off to suspend.
Exceptions: If the name is too long, an error message will be displayed. If the PCB cannot be found, an error message will be displayed saying the PCB does not exist.
Explanation of Code: The method asks for the name of the PCB to be suspended If the name is valid and the PCB is found, the PCB is sent off to suspend. If the name is not valid, an error message will be displayed. If the PCB is not found, an error message will be displayed.
void suspendWrapper();

Purpose: The method will collect the info to resume a PCB.
Parameters: The method reads in the PCB name from the user,
Preconditions: The user must enter the command resume.
Postconditions: The PCB name will be sent off to resume.
Exceptions: If the name is too long, an error message will be displayed. If the PCB cannot be found, an error message will be displayed saying the PCB does not exist.
Explanation of Code: The method asks for the name of the PCB to be resumed. If the name is valid and the PCB is found, the PCB is sent off to resume. If the name is not valid, an error message will be displayed. If the PCB is not found, an error message will be displayed.
void resumeWrapper();

pcb.h

Purpose: The method will insert the PCB into the correct queue.
Parameters: The method accepts a PCB.
Preconditions: The PCB needs to have a state and have whether it is suspended or not.
Postconditions: The PCB will be inserted into the correct queue.
Exceptions: No exceptions.
Explanation of Code: A PCB can be inserted into one of four queue based off the state and if it is suspended. If it is in the ready state, regardless if it is suspended or not, the method the PCB gets inserted is the same. If the queue is empty, the PCB is set to the head and the tail, and the next and previous are set to null, and increase the count.. If the queue is not empty, if the priority of the PCB is less than the head's priority, set the head's previous to the PCB, the PCB's next to the head, the PCB to the head, and increase the count. If the PCB's priority is less than the current PCB's priority, set the PCB's the current previous's next to PCB, the current's previous to PCB, the PCB's next to current, and the previous to the current's previous. Set the

current to current's next, and continue checking until at the tail. If you get all the way to the tail, set the PCB's previous as the tail, the tail's next as the PCB, the PCB's next as null, set the PCB to the tail, and increment the count. If the PCB is blocked, regardless if it is suspended or not, set the tail's next to PCB, the PCB's previous to the tail, the PCB's next to null, and set the tail to the PCB.
void insertPCB(pcb Pcb);*

Purpose: The method will allocate memory for the PCB.
Parameters: The method uses the size of the struct PCB.
Preconditions: The user must select to create a PCB and enter the information correctly.
Postconditions: The method return's an allocated memory block for a PCB.
Exceptions: No exceptions.
Explanation of Code: The method uses sys_alloc_mem to allocate memory for a PCB using the size of the struct for a PCB.
pcb allocatePCB();*

Purpose: This method sets up a PCB and sets the attribute values.
Parameters: The method takes in the PCB name, the class code, and the priority code.
Preconditions: The user must select to create a PCB and enter the information correctly.
Postconditions: The method returns a setup PCB.
Exceptions: No exceptions.
Explanation of Code: The method calls to allocate memory for the PCB. The PCB's name, priority, state, suspended, and class code are set. The newly setup PCB is returned.
pcb setupPCB(char *name, int classCode, int priorityCode);*

Purpose: This method removes a PCB from a queue.
Parameters: The method accepts a PCB.
Preconditions: The user must enter to delete a PCB and enter valid values into delete PCB.
Postconditions: The method will remove the PCB. The method returns -1 if there was an error and 0 if the remove was successful.
Exceptions: No exceptions.
Explanation of Code: The method looks at the state and the suspended state of the PCB to see what queue it is in. If the PCB is the head of any of the queues, the head is set to the head's next. If the PCB is the tail, the tail is set to the tail's previous. If the PCB is in the middle of the list, set the PCB's previous next to the PCB's next and the PCB's next previous to the PCB's previous to remove the PCB from the queue. Return 0 if the PCB was removed and -1 if a PCB was not successfully removed.
int removePCB(pcb process);*

Purpose: The method changes the priority for a PCB.
Parameters: The method accepts the PCB and the priority number to change the PCB to.
Preconditions: The user must have successfully entered the information in the wrapper for it to be sent to setPriority.

Postconditions: The PCB with the new priority is reinserted into the correct queue.
Exceptions: No exceptions.
Explanation of Code: The method changes the priority number for the PCB. It removes the PCB from the current queue and reinserts it into its correct place based off the new priority number. A success message is printed to the user.
*void setPriority(char *name, int priorityNum);*

Purpose: The method will find a PCB based off the PCB name.
Parameters: The method accepts a name to search for.
Preconditions: The name for the PCB must be less than eight characters.
Postconditions: The method will return a PCB if found. If the PCB is not found, then the method will return null.
Exceptions: No exceptions.
Explanation of Code: The method loops through each entry in the four queues. If the name accepted by the method matches the name of a PCB in the queue, the PCB is returned. If no PCB with the name is found, null is returned.
pcb findPCB(char *PcbName);*

Purpose: The method will display a PCB with the name accepted by the process.
Parameters: The method accepts a character array with the name for the PCB.
Preconditions: The name must be no more than eight characters.
Postconditions: The method prints the PCB.
Exceptions: If the PCB is not found, then an error message saying there is no PCB with that name is printed.
Explanation of Code: The method calls findPCB for the name entered. If the PCB is found, the method calls printOnePCB to print the PCB. If the PCB is not found, an error message is printed.
*void showPCB(char *name);*

Purpose: The method shows all ready processes.
Parameters: No parameters.
Preconditions: The method showReady must be entered by the user.
Postconditions: All ready processes will be displayed.
Exceptions: No exceptions.
Explanation of Code: The method first prints the ready queue. Starting at the head, the method printOnePCB is called for the current PCB. The PCB moves to the next PCB. As long as current is not null, the loop will continue. The same process is then done for the suspend ready queue. In the end, all ready processes are displayed.
void showReady();

Purpose: The method shows all blocked processes.
Parameters: No parameters.
Preconditions: The method showBlocked must be entered by the user.
Postconditions: All blocked processes will be displayed.

Exceptions: No exceptions.
Explanation of Code: The method first prints the blocked queue. Starting at the head, the method printOnePCB is called for the current PCB. The PCB moves to the next PCB. As long as current is not null, the loop will continue. The same process is then done for the suspended blocked queue. In the end, all ready processes are displayed.
void showBlocked();

Purpose: The method shows all processes.
Parameters: No parameters.
Preconditions: The user must enter the command showAllProcesses.
Postconditions: The method displays all processes.
Exceptions: No exceptions.
Explanation of Code: The method calls showReady to show all ready processes. The method then calls showBlocked to show all blocked processes.
void showAllProcesses();

Purpose: The method prints all the attributes for one PCB.
Parameters: The method accepts a PCB.
Preconditions: The PCB to be displayed must already be created.
Postconditions: The PCB and its attributes will be displayed.
Exceptions: No exceptions.
Explanation of Code: The method prints out all the attributes for a PCB. It displays the na,e. class, state, suspended state, and the priority for the PCB. Titles are in front of each attribute to make it clear what attribute is being displayed.
void printOnePCB(pcb Pcb);*

Purpose: The method will change the state of a PCB to block a PCB.
Parameters: The method accepts a PCB.
Preconditions: The user must enter the command block**.**
Postconditions: The PCB will be set to the state block.
Exceptions: No exceptions.
Explanation of Code: The method removes the PCB. The state is the set to blocked. The newly unblocked PCB is then inserted using insertPCB. A success message is then displayed.
void block(pcb PCB); 8*

Purpose: The method will change the state of a PCB to ready.
Parameters: The method accepts a PCB.
Preconditions: The user must enter the command unblock**.**
Postconditions: The PCB will be set to the state ready.
Exceptions: No exceptions.
Explanation of Code: The method removes the PCB. The state is the set to ready. The newly blocked PCB is then inserted using insertPCB. A success message is then displayed.
void unblock(pcb PCB);*

Purpose: The method frees all memory associated with a PCB.
Parameters: The method accepts a PCB.
Preconditions: The user must enter to delete a PCB.
Postconditions: All memory for the PCB will be freed. The integer 0 is returned.
Exceptions: No exceptions.
Explanation of Code: The method frees all PCB attributes using sys_free_mem: name, priority, state, suspended state, class, next PCB, previous PCB, stack top, and stack base. The PCB is then freed using sys_free_mem.
int freePCB(pcb PCB);*

Purpose: The method will change the suspended state of a PCB to suspend.
Parameters: The method accepts a PCB.
Preconditions: The user must enter the command suspend**.**
Postconditions: The PCB will be set to the suspended state suspend.
Exceptions: No exceptions.
Explanation of Code: The method removes the PCB. The state is the set to suspend. The newly suspended PCB is then inserted using insertPCB. A success message is then displayed.
void suspend(pcb PCB);*

Purpose: The method will change the suspended state of a PCB to not suspended.
Parameters: The method accepts a PCB.
Preconditions: The user must enter the command resume**.**
Postconditions: The PCB will be set to the suspended state not suspended.
Exceptions: No exceptions.
Explanation of Code: The method removes the PCB. The state is the set to not suspended. The newly not suspended PCB is then inserted using insertPCB. A success message is then displayed.
void resume(pcb PCB);*

Module 3

irq.s

Purpose: The method will complete the interrupt service routine when an interrupt is called.
Parameters: No parameters.
Preconditions: An interrupt must be alerted.
Postconditions: The interrupt will be handled and will return from the interrupt.
Exceptions: No exceptions.
Explanation of Code: The method pushes general purpose registers to the stack, and then pushes segment registers. The register that points to the top of the stack is pushed. Sys_call is then called, and the value returned from the call is used to set the stack pointed. The segment register and general purpose registers are pop, and then the method returns from the interrupt.
sys_call_isr

context.h

Purpose: The method will handle when the op code is idle and when it should exit
Parameters: The method takes in the current registers.
Preconditions: sys_call is called from sys_call_isr
Postconditions: The current operating process's stack top is returned.
Exceptions: If the current PCB is null, then the current is returned.
Explanation of Code: The method takes in the current register values. If the cop is null, then the current context is set to the registers. Else, if the op code is exit, then the PCB is removed and freed. If the op code is idle, then the cop is reinserted into the proper queue. If the current PCB is not null, meaning the ready queue has something in it, then the PCB is removed and the top of the cop is returned. Else, the current context is returned.
u32int sys_call(context *registers);*

r3commands.h

Purpose: The method cause the command handler to yield to other processes.
Parameters: No parameters.
Preconditions: A process must be in the ready queue.
Postconditions: The command handler will yield to another process.
Exceptions: No exceptions.
Explanation of Code: The one line of code causes the command handler to yield to other processes if any are in the ready queue.
void yield();

Purpose: The method will load in the five processes from procsr3.c
Parameters: No parameters.
Preconditions: The user must enter loadr3.

Postconditions: The three processes will be loaded and processed.
Exceptions: No exceptions.
Explanation of Code: For the five different processes, the method sets up the PCB. The context for the process is then set to the top of the PCB's stack. The context's registers are set to the needed values.
void loadr3();

Module 4

r4commands.h

Purpose: The method set an alarm entered by the user.
Parameters: The method takes in a time from the user and a message for what the message should be when the alarm goes off.
Preconditions: The user must enter setAlarm to set an alarm.
Postconditions: The alarm will be set and saved.
Exceptions: If the time entered for the alarm is not valid (Ex: 45:77:92 is not a valid time), then the method will not save that alarm.
Explanation of Code: The method first takes in the time the user wants for the alarm. It then checks to make sure that it is a valid time before it saves the alarm. It then asks for a message that will be associated with the alarm. A process is then created for the alarms, the total number of alarms in the system is incremented, and the current PCB for the alarm is resumed.
*void setAlarm(char * timeStatement, char * message);*

Purpose: The method checks to see if it is time for any alarms to go off.
Parameters: The method goes through the alarm list to see if any alarms should go off.
Preconditions: None
Postconditions: The alarms that need to go off will send their alarm message.
Exceptions: None.
Explanation of Code: The method looks at the hour, minute, and second components of each alarm in the alarm list. It checks to make sure that the alarms in the list are valid. The method then looks to see if the current time is past the time for each alarm to go off. If the alarm should go off, the message associated with the alarm is displayed. The alarm is then rmoved. Once all alarms have been checked, the process for the alarms will be idle, unless no alarms are left, in which case the process exits.
void alarmProcess();

Purpose: The method keeps an infinite process idle.
Parameters: No parameters.
Preconditions: The infinite process must be created.
Postconditions: The infinite process remains infinite and a message saying it is still infinite is displayed.
Exceptions: No exceptions.
Explanation of Code: While true, the method continues the infinite process. A success message is displayed showing that the process is still infinite.
void infinite();

Module5

mcb.h

Purpose: The method initializes a heap from which memory can be allocated.
Parameters: This method takes in how large the user would like the heap to be as an integer.
Preconditions: The user must enter initializeHeap to set initialize the heap.
Postconditions: The heap will be created and ready to allocate.
Exceptions: None.
Explanation of Code: This method first calculates the needed size by adding the user-requested size to the size of an MCB that will be used to describe the heap. It then creates a heap called startHeap by calling the kmalloc method, and returns -1 if kmalloc doesn't work. An MCB called head is then initialized to describe the entire heap. Head's type is Free. Head's start address is the start address of the heap. Head's size is the actual size, calculated by adding the user-requested size to the size of an MCB. Head's name is "Starting MCB Free Block". Both next and previous are set to null for Head. The head of the memory list for free blocks is set to head, and the memory list for allocated blocks is set to Null, as there are no allocated blocks yet. *int initializeHeap(int size);*

Purpose: The method frees an allocated memory block and merges it with neighboring free blocks.
Parameters: The method takes in an address number of type ucstar
Preconditions: The user must enter freeMemory and the address of the memory block they would like to free.
Postconditions: The memory block at the user-specified address is free, and potentially merged with neighboring free memory blocks.
Exceptions: If the heap is empty, memory will not be freed and an error message will be returned. If the user-specified address isn't the address of any allocated memory block, memory will not be freed and an error message will be returned.
Explanation of Code: The method first checks the allocated memory list, and, if it is empty, it returns an error message that the heap is empty. Otherwise, if a block of memory exists at the specified address, the method searches the entire heap for an allocated memory block at the user-specified address. Once found, it unlinks the allocated block from the allocated memory list using the method removeMCB(), it changes the MCB type to free, it inserts the block into the free memory list using the method sortedInsert(), and it prints out a success message. If the neighboring block of memory is also free, the method merges both blocks into just one block of free memory. If no allocated memory block exists at the user-specified address, then an error message is returned. *void freeMem(void *toFreeAddress);*

Purpose: The method allocates a block of memory.
Parameters: The method takes in an integer for the size of the block of memory to be allocated

Preconditions: The heap must be created, the free memory list must be created, and the user must have entered allocateMemory, along with the size of memory they'd like to allocate.
Postconditions: A block of allocated memory is specified and added to the allocated memory list.
Exceptions: No free memory block large enough.
Explanation of Code: The method first calculates a value for the actual size of the block to be allocated by adding the specified size and the size of the corresponding MCB. If the head of the free memory list doesn't exist, it then returns an error. Otherwise, the method checks each free block in the free memory list to see if it is large enough for the actual size of the block to be allocated. The first block that is large enough is chosen to be allocated. If none are large enough, an error is returned to state that no blocks are large enough. The chosen block is removed from the free memory list using the method removeMCB. Then, an allocated MCB is created at the beginning of the block that was just removed, and a free MCB is created for whatever space in the block isn't to be used by the newly allocated block. Next, the method prints the size of the newly allocated MCB. Lastly, the newly allocated block and the subsequent new free block are each placed respectively in the allocated and free memory lists. A pointer to the start address of this newly allocated memory block is returned.
u32int allocateMem(int size);

Purpose: The method inserts a specified block of memory into either the free or allocated memory list.
Parameters: The method takes in either the free or allocated memory list, and it takes in a block of memory.
Preconditions: The heap must be created, and both free and allocated memory lists must exist.
Postconditions: A new block of memory is added to one of the memory lists. **Exceptions:** None.
Explanation of Code: The method first checks to see if the specified memory list is empty, and, if so, it places the incoming memory block at the head of the list. If the list is not empty, then it checks to see if the incoming block's address is higher than the block currently at the head of the list. If so, it becomes the new head, and the old head is pushed down the list. If not, the method goes down the blocks in the list to see if the incoming block's address is higher than the next block's address, and, as soon as it is, it places the incoming block before the next block. If it has the lowest address, it is placed at the bottom of the list.
void sortedInsert(memoryList curList,MCB* newBlock);*

Purpose: The method returns the address of a specified block of memory **Parameters:** The method takes in an MCB
Preconditions: The heap must be created and the specified MCB must refer to an existing block
of memory.
Postconditions: An address of the specified memory block is found and returned.
Exceptions: None.

Explanation of Code: This method increments a ucstar value address until it is as large as the size of the specified MCB, and it returns that value address.
unsigned long getAddress(MCB mcb);*

Purpose: The method displays to the user every allocated block of memory. **Parameters:** None.
Preconditions: The heap must be created, the allocated memory list must be created, and the user must have entered showAllocated.
Postconditions: A list of all allocated memory blocks in the heap is printed.
Exceptions: None.
Explanation of Code: First, the method prints out a list header specifying that the allocated memory blocks are being displayed. Then, beginning at the head of the allocated memory list, the method sends each allocated memory block into the method printOneMCB which displays the information about that block.
void showAllocated();

Purpose: The method displays to the user every free block of memory. **Parameters:** None.
Preconditions: The heap must be created, the free memory list must be created, and the user must have entered showFree.
Postconditions: A list of all free memory blocks in the heap is printed.
Exceptions: None.
Explanation of Code: First, the method prints out a list header specifying that the free memory blocks are being displayed. Then, beginning at the head of the free memory list, the method sends each free memory block into the method printOneMCB which displays the information about that block.
void showFree();

Purpose: The method prints information regarding a single block of memory **Parameters:** The method takes in a pointer to an MCB of a memory block.
Preconditions: The heap must be created, and the user must have entered either showFree or showAllocated.
Postconditions: Information regarding a memory block, including the starting address and the size, is displayed to the user.
Exceptions: None.
Explanation of Code: The method first displays the text “Address:”; then, on the next line, it displays the address of the specified MCB; then, on the next line, it displays the text “size:”; then, on the last line, displays the size of the MCB. It then creates another new line and completes.
void printOneMCB(MCB currentMCB);*

Purpose: Informs the user of whether there are any allocated blocks of memory within the heap or not.
Parameters: None.
Preconditions: The user must have entered isEmpty.
Postconditions: A message is displayed informing whether the heap doesn't exist, is completely free, or contains allocated portions.
Exceptions: None.
Explanation of Code: The method checks to see if there is a heap. If not, the integer 2 is returned. If so, the method then checks to see if the allocated memory list holds a memory block at its head. If not, the integer 1 is returned. If so, the integer 0 is returned.
int checkIfEmpty();

Purpose: The method removes a specified MCB from any memory list. **Parameters:** The method takes a pointer to an MCB which is to be removed.
Preconditions: The heap must be created, and the specified MCB must exist.
Postconditions: A memory list contains one less MCB.
Exceptions: None.
Explanation of Code: First this method checks to see if the specified MCB is the head of the free memory list. If so, it sets the head of the free memory list to the specified MCB's next MCB. If not, it checks to see if the specified MCB is the head of the allocated memory list. If so, it sets the head of the allocated memory list to the specified MCB's next MCB. If not, it checks to see if the specified MCB is at the end of a list. If so, it sets the specified MCB's previous MCB's next MCB to null. If not, it sets ths specified MCB's previous MCB's next MCB to the specified MCB's next MCB, and the specified MCB's next MCB's previous MCB to the specified MCB's previous MCB. No matter where in any list the specified MCB is located, it is removed and its neighboring MCBs are linked past it.
void removeMCB(MCB mcb);*