CS450 Group 1

Matt Angeline
Kyle Breedlove
Katrina Cortopassi
Sampurna Dhungana

Programmer's Manual

Contents

Serial.c

*int polling(char * buffer, int * count):* This function calls the polling function from the Keyboard_polling.c file written as part of R1.

Keyboard_polling.c

*int keyboard_polling(char * buffer, int * count):*This function that manages the input and output of the OS by reading and echoing the user's input in the terminal through the serial port.

Comhandle.c

int comHand(): This function is used to display the menu to the user, then asks the user for his input using sys_req(Read…). Once the input is entered from polling(), it will then check the input at the first location inside of the userInput array and determine if it equals one of the options available. Each option is labeled within the menu for the user to choose from. After each execution except for shutdown will automatically return to comhandle to allow the user to make another selection. Shutdown asks the user if they want to shut down then either enters the shutdown protocol or returns to the comhandler.

void help(): Writes each user function to screen along with an explanation on what they do using sys_req(WRITE,...).

void clearInput(): Clears the user input from the buffer.

void getTime(): This function writes the current value of time to the RTC index register, converts the result to decimal value and displays the decimal value of time to the user.

void setTime(): Prompts the user to enter a time which is then stored in memory.

*void getDate():*This function writes the current value of date to the RTC index register, converts the result to decimal value and displays the decimal value of date to the user.

void setDate(): Prompts the user to enter a date which is then stored in memory.

void version(): Displays current version of the operating system using sys_req(WRITE,...).

void shutdown(): Begins the shutdown process, by prompting the user to confirm that they want to shutdown the system using sys_req(WRITE,...).

unsigned int decToBCD(int num): converts a given integer (num) to binary coded decimal. Returns an unsigned integer.

int BCDToDec(int num): converts a given integer to its corresponding character. Returns a character.

void itoa(int num, char str):* Takes in an integer number and a character pointer as parameters. Then it gets the second number using modulo 10 and the first number by dividing by 10. Finally, it stores the characters in the character array.

processQueues.c

Struct pcb: This struct defines a PCB: its name, class, priority, status, state and stack.

Struct queue: This struct defines a queue: its head, tail and count.

queue getReadyQueue():* This function returns a pointer to the ready queue.

queue getBlockedQueue():* This function returns a pointer to the blocked queue.

queue getSuspendedReadyQueue():* This function returns a pointer to the suspended blocked queue

queue getSuspendedBlockedQueue():* This function returns a pointer to the suspended ready queue

pcb allocatePCB():* This function is used to allocate the required memory for the PCB.

int freePCB(pcb PCB):* This function is used to free the memory allocated to the PCB.

pcb setupPCB(char name[],int pcbClass,int Priority):* This function is used to allocate memory to the PCB and assign its class, priority state and status.

pcb findPCB(char[] name):* This function searches through the ready and blocked queues to find the PCB with a matching name to the char[] passed in as a parameter. It will return a pointer to this PCB or NULL if it cannot be found.

void addToReadyQueue(pcb PCB):* This function adds the PCB passed in as a parameter to the ready queue sorting by its priority compared to the priorities of the other PCBs in the queue.

Void addToSuspendedReadyQueue(pcb PCB):* This function adds the PCB passed in as a parameter to the suspended ready queue in first come first server priority.

void addToBlockedQueue(pcb PCB):* This function adds the PCB passed in as a parameter to the blocked queue following First In First Out logic.

Void addToSuspendedBlockedQueue(pcb PCB)* This function adds the PCB passed in as a parameter to the suspended blocked queue following first come first serve priority.

int removeFromQueue(pcb PCB):* This function determines which queue the PCB is in by looking at its state. It then will set the surrounding PCBs' previous and next PCB pointers to their new previous and next PCB pointers, removing it from the queue. Returns 0 on error and 1 on success.

userCommands.c

void createPCB(char [] name, int pcbClass,int priority): This function takes in the given parameters and calls setupPCB and adds it to the ready queue.

void deletePCB(char []name): This function locates the PCB within the given queue and removes it from the queue while also freeing its allocated memory.

void block(char [] name): This function locates the given pcb and sets its state to blocked, removes it from its current queue and then reinserts it into the correct queue.

void unblock(char [] name): This function locates the given pcb within the blocked queue and sets its state value to ready, then removes it from the current queue and reinserts it back into the correct queue.

void showPCB(char [] processName): This function checks to ensure that the name of the PCB is a valid PCB within the queues. It then copies each parameter within the PCB requirements and prints them to the screen.

void showReady(): This function locates the head of the ready queue, and then passes in the name of each of the members of the ready queue to the showPCB function.

void showBlocked(): This function locates the head of the blocked queue, and then passes in the name of each of the members of the block queue to the showPCB function.

Void showSuspendedBlocked(): this function locates the head of the suspended blocked queue and then passes in the name of each of the members of the suspended blocked queue into the showPCB function. This is solely used so that we can call showAll to show the other pcbs easily.

Void showSuspendedReady(): This function locates the head of the suspended ready queue and then passes in the name of each of the members of the suspended ready queue into the showPCB function. This is solely used so that we can call showAll to show the other pcbs easily.

void showAll(): This function calls showReady() then showBlocked() then showSuspendedReady() and showSuspendedBlocked() and writes all pcbs to the screen in the order that they occur within their queues. It displays the pcbs in the ready queue first then the blocked queue, then the suspended queues.

void Suspend(char [] name): This function locates the given pcb and changes its status to “Suspended”.

void Resume(char [] name): This function locates the given pcb and changes its status to “Not Suspended”.

void setPriority(char name[], int PCBpriority): This function locates the given pcb and changes the priority number to the new given number.

void ResumeAll(): This function resumes all of the processes that are in the suspended queues.

Processes.c (given)

Void Proc1(): This function enters the for loop and prints out the preprogrammed message to display one time.

Void Proc2(): This function enters the for loop and prints out the preprogrammed message to display two times.

Void Proc3(): This function enters the for loop and prints out the preprogrammed message to display three times.

Void Proc4(): This function enters the for loop and prints out the preprogrammed message to display four times.

Void Proc5(): This function enters the for loop and prints out the preprogrammed message to display five times.

userCommandsR3.c

Void yield(): This function will call the interrupt on line for line 60 which gives up control of the CPU. This function is R3 specific; after this it is removed from user handling.

pcb loadr3():* This function will load each of the processes as an individual PCB and place them into the suspended ready queue.

Interupts.c

Idt_set_gate: This is used to initialize the interrupt on line 60.

infiniteCommands.c (R4)

Struct alarm (infiniteCommands.h): It is a struct which defines an alarm and its properties (name of the alarm, its message and the time when the alarm should be triggered).

Void foreverIdle(): This function calls sys_req IDLE and prevents the system from breaking.

Void checkAlarm(): This function searches the array storing created alarms, and checks if any alarms that have been set have a time equal to the current time. If the times do match, it prints the message set with the alarm and removes the alarm from the array.

Void setAlarm(): This function asks the user to input a name, message, and time for a new alarm. It then adds the new alarm into the alarmList at the first available location.

Void removeAlarm(): This function removes a given alarm from the list of alarms set.

int checkInput(char name[], int hr, int min, int sec): This function checks that values given by the user when creating a new alarm are valid.

kmain.c

Alter comhand call to processes: We had to comment out the call to comHandle and created a PCB for each of the processes, comhandle, idle forever, and checkAlarm. We used a similar method as used in the R3 LoadR3 function to create these PCBs, setting the names of each of them to represent what each call does.

irq.s

Sys_call_isr (given): This function is the assembly language that was given during the R3/R4 introduction. The general purpose of this function was to store the information of each of the flags to ensure the information is saved and can be placed back into the system at the correct time.

mpx_supt.c

Sys_call: Checks if the global variable Current Operating Process has been called before if it updates the current information stored into COP or it will free what is in COP depending on if the code is idling or exiting respectively. It then checks to see if anything is in the ready queue if there is something it will start running each of the processes within the system one at a time and if nothing is in the ready queue will return the old context that was stored.

heapManager.c

Struct cmcb: Contains an address, type, size, nextCMCB pointer, and previous CMCB pointer (prevCMCB) of a CMCB.

cmcb placeInFreeList(cmcb* toAdd):* Takes the CMCB and places it into the free list, if the CMCB is currently in the Allocated list and needs to be removed it automatically will remove it.

cmcb placeInAllocList(cmcb* toAdd):* Takes a CMCB and places it into the Allocated list.

cmcb findCMCB(u32int addr):* Takes an address and returns the cmcb with that address or NULL if one does not exist.

Void initializeHeap(int size): Takes a size and allocates that many bytes in memory. Also creates the beginFree CMCB pointer that starts as containing the entirety of the initialized heap.

U32int allocateMemory(int size): Takes a size and allocates that much memory if available, in a First Fit algorithm, creating a new CMCB if a free block needs to be split as it contains excess memory.

U32int freeMemory(u32int addr): Takes the address of a memory as a parameter and clears it making it a free memory.

Int isEmpty(): If the allocated list has an item within it it will return a 1 stating that the allocated list is not empty otherwise it will return a 0 stating that it is empty.

Void showFreeMemory(): Shows the list of free memories in the heap.

Void showAllocatedMemory(): Shows the list of allocated memories in the heap.

Void merge(): Merges two or more free memories whenever they are adjacent to each other.

R6/Serial.c

Int Com_Open(): This creates and initializes the port to be open for the device and sets up the interrupts to the correct information.

Int Com_Read(): Takes the information from the stored data and reads it to the screen storing it within a buffer otherwise.

Int Com_Write(): This copies the keystrokes and prints them out to the user on the screen. This will also populate the buffer and print each character as it is pressed causing the interrupt to happen

Void Top_Handler(): This checks each interrupt and chooses the correct handler.

Int Input_h(): This handles the interrupt for the reading

Int Output_h(): This handles the interrupt for the writing

Int Com_Close(): Closes the port and disables the interrupts