

DiamondDogs

Operating System

CS 450 MPX

Programmers Manual

Version 4.0

Oct 20 2021

Dakota Lacy, Jacob Taylor, Easton Thewes, Leah Cave

Contents

MPX Project Overview:	4
Operating System Functions:	4
run_ch()	4
run_startup()	4
run_getdate()	4
run_gettime()	5
run_help()	5
run_version()	5
run_settime()	5
run_setdate()	6
run_shutdown()	6
get_command()	7
keycap()	7
itoa()	7
polling()	8
setupPCB()	9
findPCB()	9
allocatePCB()	9
freePCB()	9
removePCB()	10
insertPCB()	10
get_pcb_data()	10
blockPCB()	10
unblockPCB()	11
suspendPCB()	11
resumePCB()	11
setPriority()	11

showPCB()	12
printReady()	12
printBlocked()	12
printAll()	12
get_name()	13
get_class()	13
get_prio()	13
print()	13
initQueues()	14
deletePCB()	14
deleteAll()	14
resumeAll()	14
alarm()	15
add_alarm()	15
check_alarm()	15
get_current_time()	16
loadr3()	16
sys_call_isr()	16
sys_call()	16
create_infinite()	17

i. MPX Project Overview:

The MPX Project is designed to implement various functions of an operating system. This module includes our first version of the command menu, allowing the user to interact with the OS through the terminal. The user will use the keyboard to send input and whenever a system response is generated, it will display on the terminal window.

ii. Operating System Functions:

I. `run_ch()`

i. Functionality:

This function initiates the control switching to our command handler. This module prints the OS specifications, the menu and determines what command to execute based on the user's input.

ii. Input Parameters:

None

iii. Returns:

None

II. `run_startup()`

i. Functionality:

This module builds and outputs the startup design and system details.

ii. Parameters:

None

iii. Returns:

None

III. `run_getdate()`

i. Functionality:

Getdate uses the *outb* function to access the register value for each of the 3 parts of date, and then converts them in order to be able to output the correct system response.

ii. Parameters:

- None
- iii. Returns:
 - None

IV. `run_gettime()`

- i. Functionality:

Gettime utilizes the same logic as getdate in order to access the individual registers corresponding to each piece of the time.
- ii. Parameters:
 - None
- iii. Returns:
 - Int

V. `run_help()`

- i. Functionality:

The help function takes in the user input buffer where it parses the command to determine the menu option to include. If nothing follows the initial help command, it outputs the main menu of the system.
- ii. Parameters:
 - char* commandBuff - buffer holding UI command.
 - int bufferSize - size of corresponding buffer.
- iii. Returns:
 - Int

VI. `run_version()`

- i. Functionality:

This module writes the current system version to the terminal.
- ii. Parameters:
 - None
- iii. Returns:
 - Int

VII. `run_settime()`

- i. Functionality:

Settime allocates memory in character arrays for the hours, minutes and seconds declaration. The function parses through the commandBuff to read in the three values for hour, minute, and second. Those values are run through the 'atoi' function which are then converted to BCD to be placed into the registers

- ii. Parameters:
char * commandBuff - buffer holding UI command.
- iii. Returns:
Int

VIII. run_setdate()

- i. Functionality:
Settime allocates memory in character arrays for the month, day, and year declaration. The function parses through commandBuff to read in the three values for month, day, and year. Those values are converted to integers with the 'atoi' function and then converted to BCD to be placed into the registers
- ii. Parameters:
char * commandBuff - buffer holding UI command.
- iii. Returns:
Int

IX. run_shutdown()

- i. Functionality:
Shutdown confirms with the user with a yes or no option on whether they want to shut down the program or not. If yes is entered, then it returns, 1, indicating shutdown, if no is entered, a 0 is returned and the program keeps running
- ii. Parameters:
char* buffer - Used to read confirmation input
Int size - size of buffer
- iii. Returns:
Int - 1 tells command handler to shutdown, 0 keeps the program running

X. `get_command()`

i. Functionality:

The function starts at the 0 index of `commandBuff` and searches for the first ' ' character or a null terminator and places those characters in a command array. From there we `strcmp` the command array to different commands to determine what command was typed. Once determined, that command is run.

ii. Parameters:

`Char * commandBuff` - buffer holding UI command

`Int bufferSize` - size of `commandBuff`

iii. Returns:

`Int`

XI. `keycap()`

i. Functionality:

Called from within polling to handle keyboard logic. Through a series of if statements, the key capture can determine what letter or number is pressed, as well as special characters like backspace, delete, and enter.

ii. Parameters:

`char* buffer` - A buffer that inputted characters are placed into and removed from.

`Int location` - Keeps track of what position the cursor is on after each keystroke

`Int length` - Keeps track of the length of the buffer

iii. Returns:

`Int location` - Returns the location of the cursor so that it can be tracked, and sent back to `keyCap`. If -1 is returned, enter has been pressed and causes a break to happen in polling, stopping user input.

XII. `itoa()`

i. Functionality:

Convert an integer to a string

ii. Parameters:

`Int num` - integer to be converted

char* str - char* for the string to be placed into

iii. Returns:

char* str - returns string with converted integer

XIII. `polling()`

i. Functionality:

To check for ready byte that indicates that the user has pressed the keyboard. From there `keyCap` is called to handle what was pressed.

ii. Parameters:

char* buffer - buffer to place char values into

int* count - size of buffer

iii. Returns:

int* count - returns the size of the buffer

XIV. setupPCB()

i. Functionality:

To create a new pcb and fill its contents with the user inputted name, class, and priority. Checks to make sure that a pcb with the same name is not already in one of the queues and allocates space for pcb.

ii. Parameters:

char* name - user inputted name of process

Int class - integer value of class

Int priority - integer value of user inputted priority

iii. Returns:

PCB* newPCB - returns the pcb it just created

XV. findPCB()

i. Functionality:

Searches through the four queues and compares the names to every pcb to check to see if there is a pcb with the same name inside the queues.

ii. Parameters:

char* name - user inputted name to search for.

iii. Returns:

PCB* current - returns NULL pointer if one does not exist or the pcb with the same name if it does exist.

XVI. allocatePCB()

i. Functionality:

Allocates a section in memory for the PCB

ii. Parameters:

None

iii. Returns:

PCB* temp - PCB that was just allocated

XVII. freePCB()

i. Functionality:

Free's the PCB from memory

ii. Parameters:

PCB* pcb

- iii. Returns:
void

XVIII. removePCB()

- i. Functionality:
Is called by deletePCB() to remove the pcb from whatever queue it is inside
- ii. Parameters:
PCB* pcb
- iii. Returns:
Int - Returns 1 if the removal is successful

XIX. insertPCB()

- i. Functionality:
Determines what queue the pcb needs to be placed into, and adds the pcb to the queue in the correct place, depending on if the queue is priority or fifo.
- ii. Parameters:
PCB* pcb
- iii. Returns:
PCB* pcb - Returns the pcb that was just inserted

XX. get_pcb_data()

- i. Functionality:
This function calls get_name, get_class, and get_prio to gather the information needed to create the PCB. It then checks to make sure a pcb with the same name does not exist, then calls setupPCB to create it
- ii. Parameters:
Char* commandBuff - command buffer
- iii. Returns:
Int - returns 1 for success, another number for error

XXI. blockPCB()

- i. Functionality:

Moves a pcb out of the ready or suspended ready into the blocked or suspended blocked queue

- ii. Parameters:
char* commandBuff - command buffer
- iii. Returns:
PCB* pcb - returns pcb that it just blocked

XXII. unblockPCB()

- i. Functionality:
Returns the pcb to the ready or suspended ready from the blocked or suspended blocked queue.
- ii. Parameters:
char* commandBuff - command buffer
- iii. Returns:
PCB* pcb - returns pcb that it just unblocked

XXIII. suspendPCB()

- i. Functionality:
Moves pcb from ready or blocked queue to suspended ready or suspended blocked queue
- ii. Parameters:
char* commandBuff - command buffer
- iii. Returns:
PCB* pcb -returns pcb that was just suspended

XXIV. resumePCB()

- i. Functionality:
Returns pcb from the suspended ready or blocked ready to the ready or blocked queue
- ii. Parameters:
char* commandBuff - command buffer
- iii. Returns:
PCB* pcb - returns pcb that it just resumed

XXV. setPriority()

- i. Functionality:

Changes priority of user inputted pcb to user inputted priority

- ii. Parameters:
char* commandBuff -
- iii. Returns:
Int - returns 1 for success

XXVI. showPCB()

- i. Functionality:
Shows contents of pcb
- ii. Parameters:
char* commandBuff - command buffer
- iii. Returns:
void

XXVII. printReady()

- i. Functionality:
Print current ready queue
- ii. Parameters:
none
- iii. Returns:
none

XXVIII. printBlocked()

- i. Functionality:
Prints all the pcb's in the blocked queue.
- ii. Parameters:
none
- iii. Returns:
none

XXIX. printAll()

- i. Functionality:
Shows all the pcb's in every queue.
- ii. Parameters:
none
- iii. Returns:

none

XXX. `get_name()`

- i. **Functionality:**
Called in `get_pcb_data()` to parse the command buffer and get the inputted name
- ii. **Parameters:**
`char* commandBuff` - command buffer
- iii. **Returns:**
`char* name_ptr` - copied name

XXXI. `get_class()`

- i. **Functionality:**
Called in `get_pcb_data()` to parse the command buffer and get the inputted class number
- ii. **Parameters:**
`char* commandBuff` - command buffer
- iii. **Returns:**
`char* class_ptr`

XXXII. `get_prio()`

- i. **Functionality:**
Called in `get_pcb_data()` to parse the command buffer and get the inputted priority number
- ii. **Parameters:**
`char* commandBuff` - command buffer
- iii. **Returns:**
`prio_ptr`

XXXIII. `print()`

- i. **Functionality:**
Called by `printReady`, `printBlocked`, and `printAll`, to print the individual pcb and the contents within it
- ii. **Parameters:**
`PCB* pcb` - current pcb that needs to be printed

- iii. Returns:
none

XXXIV. `initQueues()`

- i. Functionality:
Initializes the four queues.
- ii. Parameters:
none
- iii. Returns:
none

XXXV. `deletePCB()`

- i. Functionality:
Delete specified PCB from queue.
- ii. Parameters:
PCB* pcb - current pcb that needs to be deleted
- iii. Returns:
none

XXXVI. `deleteAll()`

- i. Functionality:
Deletes all processes from every queue in order to allow for shutdown.
- ii. Parameters:
none
- iii. Returns:
none

XXXVII. `resumeAll()`

- i. Functionality:
Resumes all processes in the suspended ready queue.
- ii. Parameters:
none
- iii. Returns:
none

XXXVIII. alarm()

i. Functionality:

Checks to see if an alarm PCB is inside of the ready queue and adds one if it isn't inside. Handles all user input for the message and the user imputed time. Calls add_alarm to add alarm to the system.

ii. Parameters:

char* buffer - Command buffer from command handler

iii. Returns:

none

XXXIX. add_alarm()

i. Functionality:

Retrieves alarm data to add into the system.

ii. Parameters:

char* hr - Pointer variable representing hours

char* min - Pointer variable representing min

char* sec - Variable representing seconds

char message[50] - input buffer

iii. Returns:

none

XL. check_alarm()

i. Functionality:

Function called by the alarm PCB to check the alarm times versus the current time of the system. If an alarm needs to be triggered, the function prints the message the user imputed.

ii. Parameters:

none

iii. Returns:

none

XLII. `get_current_time()`

- i. **Functionality:**
Get the current time of the system.
- ii. **Parameters:**
none
- iii. **Returns:**
Char* getTime - Char* of the hours, minutes, and seconds concatenated together to compare against alarm times.

XLIII. `loadr3()`

- i. **Functionality:**
This function loads all described processes into the system and places them into their correct queue.
- ii. **Parameters:**
none
- iii. **Returns:**
none

XLIV. `sys_call_isr()`

- i. **Functionality:**
Pushes and pops all the CPU registers and calls the sys_call function.
- ii. **Parameters:**
none
- iii. **Returns:**
none

XLV. `sys_call()`

- i. **Functionality:**
Gets operating processes context data and changes the currently operating process based on what processes are inside of the ready queue.
- ii. **Parameters:**
context* registers - pointer to processes context
- iii. **Returns:**
unsigned pointer to context of process.

XLV. `create_infinite()`

- i. **Functionality:**
Initializes a process to run continuously.
- ii. **Parameters:**
none
- iii. **Returns:**
none