DiamondDogs

Operating System

CS 450 MPX Programmers Manual

Version 1.0

Sept 9 2021

Dakota Lacy, Jacob Taylor, Easton Thewes, Leah Cave

Contents

1. 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.

2. 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 as 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 a 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