Operating Systems Homework #2

Author: Erik Hattervig

Description:

This program, dsh, is a diagnostic shell that allows the user to find the name of a process in memory based on a given process id number. It also allows the user to view some information about the system and send signals to processes.

The program allows for the finding of a process's name via the "cmdnm" command. This command takes an argument in the form of a pid number and looks in the proc folder for the matching process and reads the process name and returns it to the user. If and error in ether the formatting of the pid or a pid that does not have a process associated with it is given, then the program will print an appropriate error message.

When the "signal" command is invoked, it takes a signal id and a process pid in the form: 'signal <signal id> <pid>' and sends the signal id to the process associated with the pid.

The "systat" command returns information about the Lunix version, memory usage, and CPU specifications from the proc folder and displays the information to the user.

The last command is "exit" and simply invokes an exit of the program.

Part 2:

Part 2 of this project added the functionality to call system commands and print information on how it ran, the ability to change directories from inside the program, the ability to redirect input and output to and from a file, the ability to pipe to commands to each other, and the ability to pipe commands over a socket from a client to a server.

To invoke a system command just type in the command and its arguments and the program will run it and print out information on how it ran such as system and user time token to complete in microseconds.

To invoke a directory change use the cd command followed by a relative or absolute path and the program will change to that directory.

To use a file redirect type in your command and then "> <filename>" to use a file as output for the command and use a command and "< <filename>" to use the file as input for he command.

To use a pipe use the syntax " <command1> | <command2>" and the two commands will be piped together with 1 having piped in input and 2 having piped out output.

To use the server pipe use the syntax " <command>)) <portNumber>" to create a server that listens to a port that is the portNumber that you specify.

To use the client pipe use the syntax " <command> ((<IP> <portNumber>" to send the command to an IP address and port that you specify.

Part 3:

Part 3 of this project added the ability to create shared memory boxes of a specified size and write, read, and copy them. To create the shared memory uses the syntax "mboxinit <number of boxes> <size of boxes in kilobytes>" to create a set of boxes of a specified size. To write to a box use the syntax "mboxwrite <boxnumber>" to write a string to a box, to exit writing to the box issue an EOF flag (ctrl-D). To read from a box use the syntax "mboxread <boxnumber>" and the shell will read till the end of the string or the end of the box that is stored in that box. To copy one box into another use the syntax "mboxcopy <box1> <box2>" and the contents of box 2 will be copied to box 1. Finally to delete all of the boxes uses the syntax "mboxdel". This will be called when exiting the shell as well to clean up any shared memory.
In order to use the shared memory the shell will create a file called "sharedMemoryID.txt" in the folder you are in. Do not delete the file as it contains the id for the shared memory you are using and deleting it can orphan the memory. In the event of a shell crash the file will still be there and running mboxdel in the folder will clean up the memory.

Usage:

From command line inside of the prog1 folder use 'make' to compile. To use, use 'dsh' from the command line.

Libraries used:

<iostream> - input and output to the console
<string> - used in parsing input from the user
<vector> - used for storing arguments from the user
<fstream> - used for reading input from the files in the proc folder
<sys/types.h> - used in invoking system calls
<signal.h> - used in sending signals to a process
<sstream> - used in turning strings into ints
<arpa/inet.h>
<cstdlib>
<cstring> - c-string library
<errno> - used for error checking
<fcntl.h>
<netinet/in.h> - used for listening to ports
<stdio> - standard template library input/output
<stdlib> - standard template library functions
<sys/resource.h>

<sys/socket.h> - for using sockets
<sys/ipc.h>
<sys/sem.h> - for using semaphores
<sys/shm.h> - for using shared memory
<sys/time> - for using time functions
<sys/types.h>
<sys/wait.h> - for using fork and wait
<time.h> - for using time functions
<usistd.h>

Program Structure:

This program operates in a loop in a function. The loop takes input from the user and parses the input using the parse function. It then determines what command was issued and does error checking before passing of the parameters to functions named after the command. If the exit command is issued, the program will exit the loop and end the program.

Files included:

prog1.cpp and makefile. The makefile contains the instructions for compiling the program, and simply dose a g++ compile on the source file prog1.cpp.

Testing:

The program has been tested using the following inputs:
<space>
<tab>
<random charaters>
cmdnm <valid pid>
cmdnm <invalid pid>
cmdnm <random letters>
cmdnm <wrong number of arguments>
signal <valid arguments>
signal <invalid arguments>
signal <wrong number of arguments>
systat
exit

Part 2:
ls
ls –al
cd <absolute dir>
cd <relative dir>

cd <invalid dir>
ls > test.txt
ls < test.txt
<cmd1> | <cmd2>
<cmd>)) 4000
<cmd> ((127.0.0.1 4000

Part 3:

Tested creating memory, writing to it, reading from it, copying it, and deleting it, all of these function appear to work correctly.
Tested reading and writing to out of bounds boxes.