Project # 1: Build a minimal shell (minShell): due Friday April 3rd.

We have seen that a Unix shell (like BASH) is a sophisticated command-interpreter and scripting language that provides a powerful user interface to the operating system. In the most basic sense, a shell is a program that executes other programs.

In this project you will build a minimalist shell that supports the following requirements:

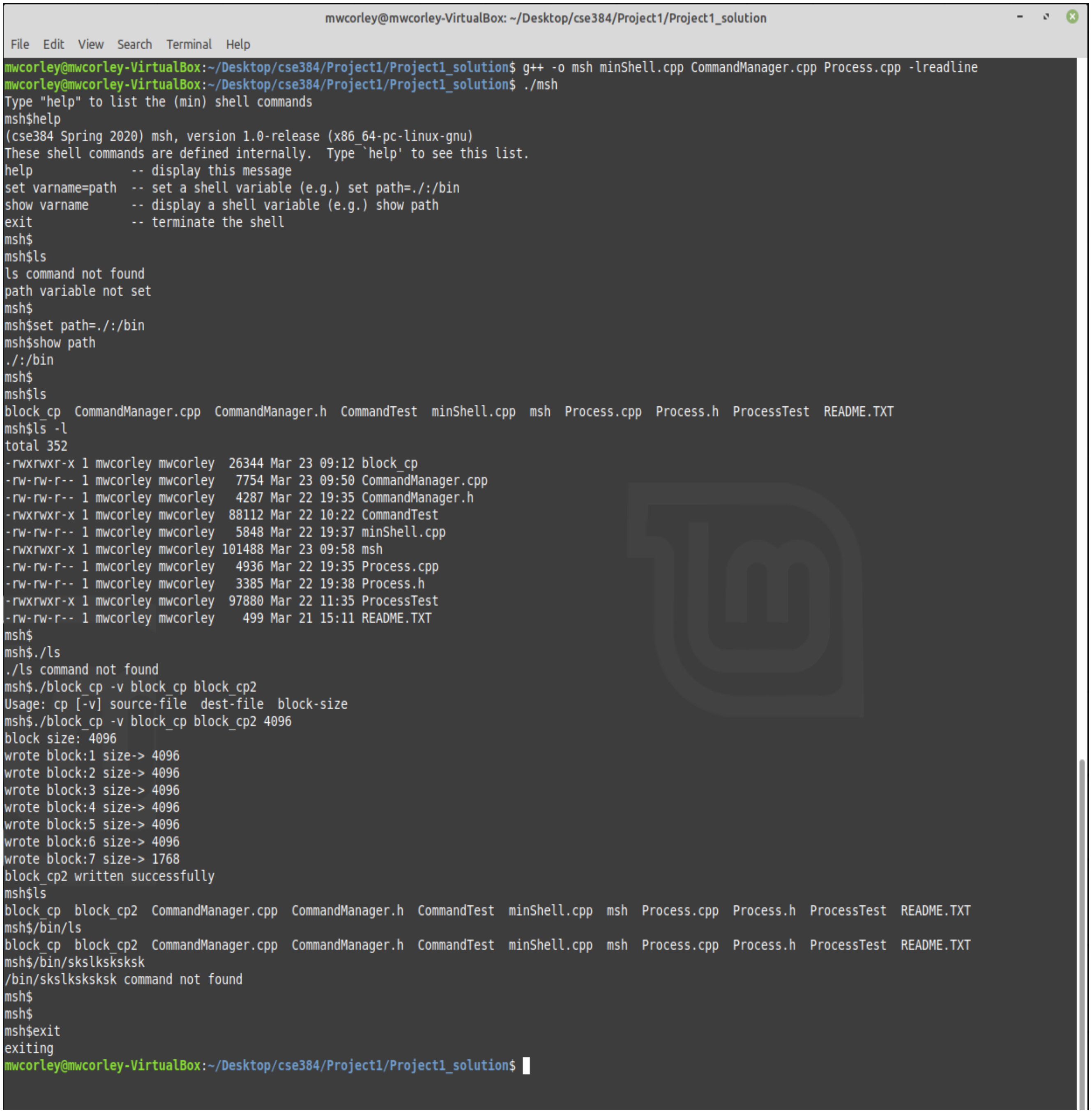
1. Display a command prompt for user input (e.g. minShell$)
2. Read user input, parse, and run commands
3. Your shell should support the following features:
    a. Shell (built-in) commands: setting/showing shell variables: support shell built-in command: *set/show*
        i. *msh$ set path=./:/bin*
        ii. *msh$ show path*
            ./:/bin
        iii. *msh$ set name=Mike*
            msh$ show name
                Mike

    b. Run (execute) external commands based on a user specified minShell path variable (*not the actual system path variable*)
        i. *msh$ set path=./:/bin*
        ii. *msh$ ls -l/bin /bin*
            - searches the specified *path* variable (above) for the *ls* program, and if found it executes the command with the arguments specified e.g. *-l /bin*
            - Note: You will have to parse commands entered by the user to determine what command and arguments to properly run in your shell.
            - Your shell will need to make use of at least two system calls
                - fork – creates a new (child) process by duplicating the parent process (the minShell process) -- use “man fork” for the man page
                - execv - loads (overlays) an existing process with a new process image -- use “man execv”. *Note: there is a family of exec functions, but we will use execv for this project*
        iii. *msh$ /bin/ls* - (absolute path), should work whether or not path is set
        iv. *msh$ ./block_cp* (relative path), should run local programs (whether or not path is set)

That’s it!   When your shell supports requirements: 1. 2. 3a and 3b, then submit the code and the output as separate documents in Blackboard under the **Project 1** assignment. Please upload your code with the name “MinShellCode”, and the output with name “MinShellOutput”.

Note: It is up to you to generate your output/screenshots so that it clearly demonstrates you've met requirements 3a. and 3b. If you don't clearly demonstrate you meet 3a. and 3b. you may not get full credit.

Here is an example output from instructor solution:



mwcorley@mwcorley-VirtualBox: ~/Desktop/cse384/Project1/Project1_solution
File Edit View Search Terminal Help

```
mwcorley@mwcorley-VirtualBox:~/Desktop/cse384/Project1/Project1_solution$ g++ -o msh minShell.cpp CommandManager.cpp Process.cpp -lreadline
mwcorley@mwcorley-VirtualBox:~/Desktop/cse384/Project1/Project1_solution$ ./msh
Type "help" to list the (min) shell commands
msh$help
(cse384 Spring 2020) msh, version 1.0-release (x86_64-pc-linux-gnu)
These shell commands are defined internally.  Type `help' to see this list.
help              -- display this message
set varname=path  -- set a shell variable (e.g.) set path=./:/bin
show varname      -- display a shell variable (e.g.) show path
exit              -- terminate the shell
msh$
msh$ls
ls command not found
path variable not set
msh$
msh$set path=./:/bin
msh$show path
./:/bin
msh$
msh$ls
block_cp  CommandManager.cpp  CommandManager.h  CommandTest  minShell.cpp  msh  Process.cpp  Process.h  ProcessTest  README.TXT
msh$ls -l
total 352
-rwxrwxr-x 1 mwcorley mwcorley  26344 Mar 23 09:12 block_cp
-rw-rw-r-- 1 mwcorley mwcorley   7754 Mar 23 09:50 CommandManager.cpp
-rw-rw-r-- 1 mwcorley mwcorley   4287 Mar 22 19:35 CommandManager.h
-rwxrwxr-x 1 mwcorley mwcorley  88112 Mar 22 10:22 CommandTest
-rw-rw-r-- 1 mwcorley mwcorley   5848 Mar 22 19:37 minShell.cpp
-rwxrwxr-x 1 mwcorley mwcorley 101488 Mar 23 09:58 msh
-rw-rw-r-- 1 mwcorley mwcorley   4936 Mar 22 19:35 Process.cpp
-rw-rw-r-- 1 mwcorley mwcorley   3385 Mar 22 19:38 Process.h
-rwxrwxr-x 1 mwcorley mwcorley  97880 Mar 22 11:35 ProcessTest
-rw-rw-r-- 1 mwcorley mwcorley    499 Mar 21 15:11 README.TXT
msh$
msh$./ls
./ls command not found
msh$./block_cp -v block_cp block_cp2
Usage: cp [-v] source-file  dest-file  block-size
msh$./block_cp -v block_cp block_cp2 4096
block size: 4096
wrote block:1 size-> 4096
wrote block:2 size-> 4096
wrote block:3 size-> 4096
wrote block:4 size-> 4096
wrote block:5 size-> 4096
wrote block:6 size-> 4096
wrote block:7 size-> 1768
block_cp2 written successfully
msh$ls
block_cp  block_cp2  CommandManager.cpp  CommandManager.h  CommandTest  minShell.cpp  msh  Process.cpp  Process.h  ProcessTest  README.TXT
msh$/bin/ls
block_cp  block_cp2  CommandManager.cpp  CommandManager.h  CommandTest  minShell.cpp  msh  Process.cpp  Process.h  ProcessTest  README.TXT
msh$/bin/skslksksksk
/bin/skslksksksk command not found
msh$
msh$
msh$exit
exiting
mwcorley@mwcorley-VirtualBox:~/Desktop/cse384/Project1/Project1_solution$
```

Helper code: **(Process package, CommandManager package, and minShell.cpp executive)**

https://mwcorley79.github.io/MikeCorley/lecture18/Project1_helpers/Project1_helpers.tar.gz

Process.h/.cpp (package): runs external programs (see interface/test stub in .cpp file)

CommandManager.h/.cpp (package): supports creation and management of shell variables, and searching the path variable

minShell.cpp (executive): implements minShell (*msh*) using the services of *Process* package and *CommandManager* package

*****Note:***** I've given lots of help. All you should really need do to complete Project 1 is implement the two functions: ***DoProcessExternalCommand, DoProcessBuiltInCommand*** in minShell.cpp***,*** which handle the (built-ins) set and show shell variables, and executing external programs respectively. View the test stubs in Process.cpp and Command Manager.cpp to see how you might implement these functions.

You are not required to use my helper code. You build all of your code if you like, but you need meet all of requirements specified above

Background and thoughts and Starting

The basic processing logic for a minimalist shell might be summarized by the following steps:

1. Display prompt to the user (e.g. minShell$)
2. Wait for a command from the user
3. parse the command
4. execute the command
5. return to step 1

To begin building your minShell you might start by building a program in C or C++ that run (executes) other programs. To do that, consider the steps (1-5) outlined above:

1. Write a while loop that will display a prompt, and then waits/reads input from the user.
    a. Reading input from the user means the user types a command line and presses enter/return
2. Parse the command line
    a. Parsing the command means to extracting the parts and determine what actions(s) to take. i.e. set/get shell variables or run/execute an external command. For example:
        i. *msh$ set path=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin*
        ii. *msh$ ls -l*

3. Executing the parsed command requires the creation of a new (child) process. Creating a new process is performed with the *fork* system call, followed by an *exec* system call to load the process image specified in the command
    a. When a shell executes a command, it first creates a copy of itself (the child process by calling *fork*). It then calls *exec i*n the child process to load/run the command by loading specified program image into (overlaying) child process with the specified program image (***See Figure 1 below for an illustration of the process).***
        i. Please read the following Lecture from USNA to begin getting a grasp on the concept: https://www.usna.edu/Users/cs/aviv/classes/ic221/s16/lec/14/lec.html

            - Note: I intend to give a significant amount of helper code. That will be posted soon, and a notification will be sent via piazza

This is the basic idea:

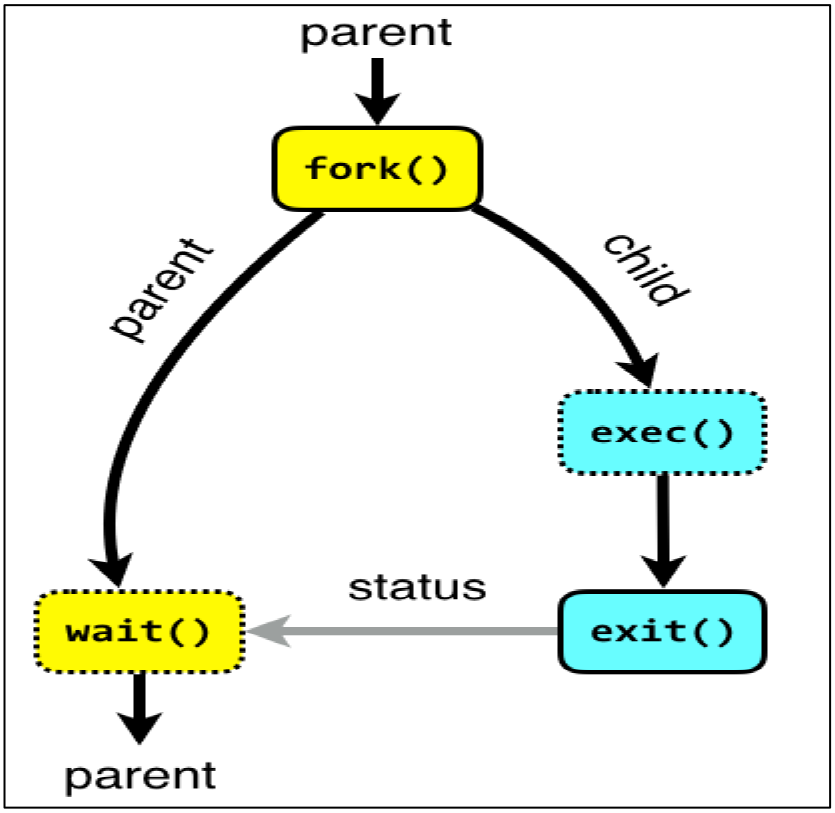


Figure 1: Command Execution

Image source: http://www.it.uu.se/education/course/homepage/os/vt18/module-2/process-management/