CST 334: Operating Systems

Dr. Glenn Bruns

# MSH 3

**Purpose**. Did you know that the 'cd' command is a bash builtin? Do you know why it is? Try running 'cd' in your msh and see what you get. Probably your current directory did not change. The purpose of this assignment is to understand how bash implements the 'cd' command, and to get more experience with the Linux API.

**Instructions**. You will extend your msh code, or the starter file msh3.c that I provide, in the two following ways:

1. Add a “built-in” command ‘cd'. This command should work in two ways, just like in bash: 1) if no argument is supplied, change to the user’s home directory, and 2) if an argument is supplied, change to the directory specified by the argument. Hint: to get the user’s home directory, your code will have to read the value of the HOME environment variable. Investigate the ‘getenv’ library function and ‘chdir’ system call. Here is the sort of output your new msh should produce:

$ ./msh

msh> ls

backup Makefile msh msh.c msh-hw-2.c README.txt temp

msh> pwd

/home/CLASSES/brunsglenn/ctests/msh

msh> cd backup

msh> pwd

/home/CLASSES/brunsglenn/ctests/msh/backup

msh> cd ..

msh> pwd

/home/CLASSES/brunsglenn/ctests/msh

msh> cd baz

msh: cd: No such file or directory

msh> cd

msh> pwd

/home/CLASSES/brunsglenn

msh> exit

$

Some system calls set the error number variable 'errno'. See the previous homework on how to use function strerror() to get an error message from the errno value. I used this in my code for errors related to changing directory.

1. Allow msh to read input from a file rather than from standard input. If msh is invoked from bash with a command-line argument, then msh should attempt to open the file specified by the command-line argument, and use that file as input. Your msh should terminate when the end of the file is reached (note that ctrl-d means “end-of-file” in bash). However, msh should still be able to be invoked without a command-line argument (as shown in the previous part of this exercise).

Hint: you are probably already using function fgets to read input from the user. Notice that the last argument of fgets is a “stream”, of type FILE \*. When you call fgets you are probably supplying stdin as the input, which tells fgets to get input from the keyboard. Look at the man pages for functions fgets and fopen.

Here is an example of how msh should work with a filename on the command line:

$ ls

backup Makefile msh msh.c msh-hw-2.c README.txt temp

$ cat temp

ls

$ ./msh temp

backup Makefile msh msh.c msh-hw-2.c README.txt temp

$

Note that no prompts are produced when msh is run from a script.

**Starter code**. On mlc104, the directory /home/CLASSES/brunsglenn/cst334/hw/hw4 contains file msh3.c that you can use as your starting point. Note that this code is a solution to the previous msh assignment.

**Testing your code**. On mlc104, the directory /home/CLASSES/brunsglenn/cst334/hw/hw4 contains test files test1.sh, …, test6.sh. Copy these and the Makefile to the directory where you developed your file msh.c. Each test should give exit status 0, like this:

$ ./test1.sh

$ echo $?

0

You need to run test1.sh first, as it will compile your code and produce binary file 'msh3' that is used by the other tests. The directory also contains a Makefile. If you enter the command 'make', the target 'tests' in Makefile will run, causing each test to run. If you enter the command 'make clean', temporary files created by testing will be deleted.

Make sure to test your code for reasonable behavior beyond the unit tests I provide.

**Submitting**. Submit your msh3.c on iLearn.

**Grading**. 10 points for each of 6 test cases, and 10 points for tidy code. Some of the test cases I run may differ a little from the test cases I have supplied you with. I do this so that you will think about test cases beyond the ones I provide you with.