Project Report

CS 4350.251

Spring 2017

Group 12

Project 2

Shell

Mark McDermott

Jason Flinn

Carlo Rodriguez

Benjamin Winston

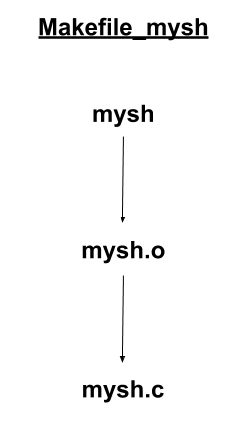
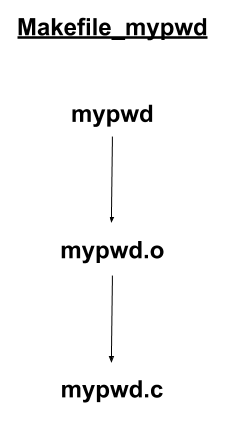
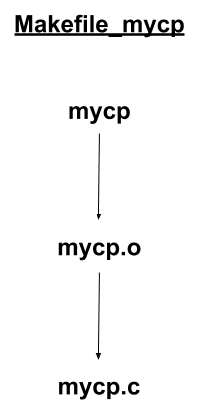
**Section I**

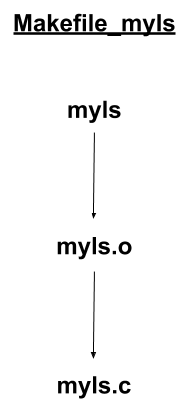
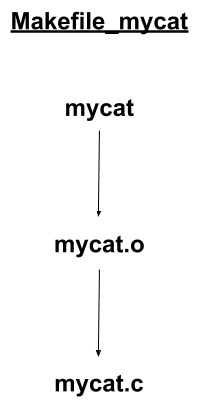
In this project, we built our own shell and basic Linux commands. Mark McDermott coded mycp and mycat and wrote section I of the report. Jason Flinn coded myls and wrote section III of the report. Carlo Rodriguez coded mysh and wrote section IV of the report. Benjamin Winston coded mypwd and wrote section II of the report. Our group met twice at Doc’s Backyard restaurant and coordinated over a Slack chat message channel.

**Section II**

<Makefiles and debugger screen shot - Ben>

1. **Makefile dependency graphs**

**  **

****

1. **Makefiles for mysh and mypwd**

Makefile\_mysh

exe: mysh.o

gcc -o mysh mysh.o

mysh.o: mysh.c

gcc -g -c mysh.c

.PHONY: clean

clean:

rm -f mysh

rm -f mysh.o

Makefile\_mypwd

exe: mypwd.o

gcc -o mypwd mypwd.o

mypwd.o: mypwd.c

gcc -g -c mypwd.c

.PHONY: clean

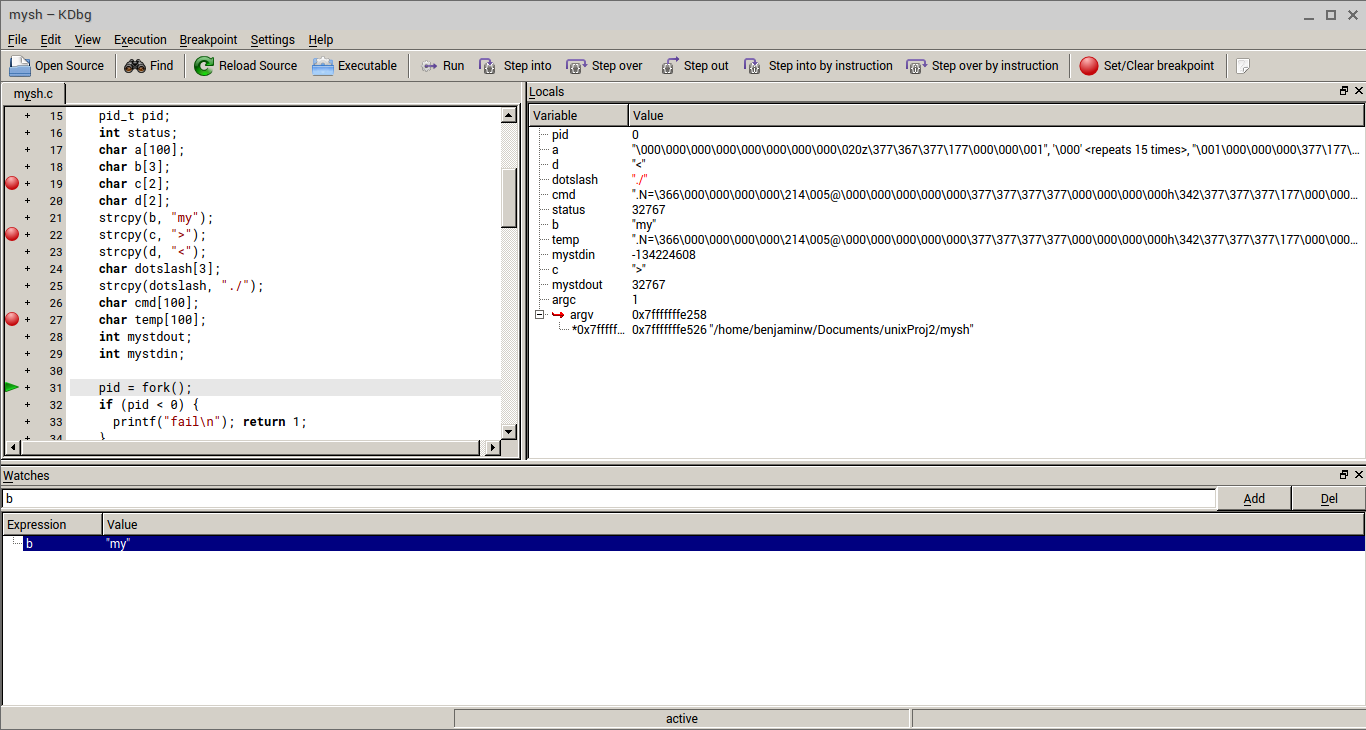
clean:

rm -f mypwd

rm -f mypwd.o

1. **Debugging screen shot**

Verifying variable assignment in mysh

****

**Section III**

**Section IV**

1) **Description of mysh.c**

Our "mysh.c" program uses the "fork()", "pipe()", and the "execvp()" functions to create a shell that takes in user input and gives the correct response in regards to the commands and options they pass to the shell. We incorporated an infinite "while" loop so that the user can constantly feed input to the shell. First, the shell will create variables that hold the strings "my", ">", "<", "|", and "./". These characters are detected later for their proper usage. The shell also creates integer variables to manipulate stdin and stdout.

Our shell then forks into a child process and reads in the users desired commands and options. It will parse each command and option using the "strtok()" function by separating by a space. Each command and option are put into a pointer pointer char variable called "res". From this point, the first element of "res" will be put into a variable called "cmd", and that variable, and maybe the an element from "res", will be compared with the variables from the first paragraph ("my", ">", "<", "|", and "./").

First, it will compare to check if the cmd variable starts with the characters "my". It does this so that it can concatenate the "./" char to the beginning of cmd so that it can run the executables created from the custom commands needed for the project.

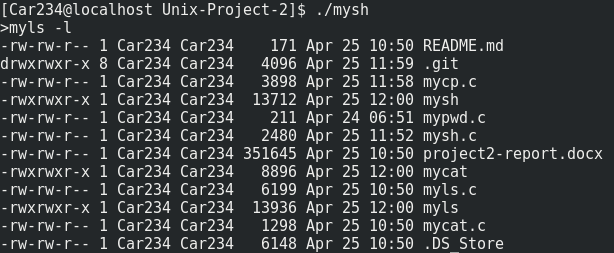
Second, it will compare to check if "res[1]" contains a "<" character for input redirection. If so, it will increment "res[1]" to remove the "<" character and open res[1] as a read only stdin and instantiate that into the variable "mystdin". It will close stdin(0) and dup2 to change stdin to "mystdin" instead.

Third, it will compare to check if "res[1]" contains a "|" character for piping. the shell will then use the "pipe()" function to create interprocess communication using the variable "pipefd[2]". From here, another child process will be created for the "reading" process. The child will use "dup2" to put reassign stdin to "pipefd[0]", and then close "pipefd[1]" since it does not need to do any writing. The child will then use execvp to do the first command. Once the child dies, the parent uses "dup2" on "pipefd[1]" and "1" to reassign stdout and then close "pipefd[0] because it does not need to do any reading. execvp will then be called on the second command.

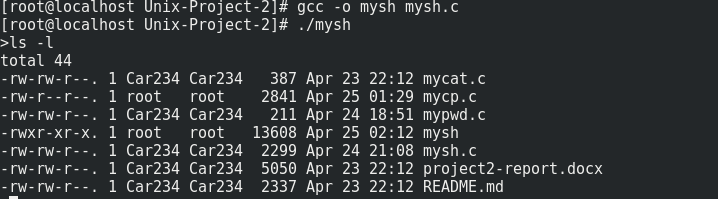
Last, it will compare to check if "res[2]" contains a ">" character for output redirection. If so, it will increment "res[2]" to delete the ">" character and open the file named in "res[2]". If the file does not exist, it is created and given a "write only" permission. This is opened into the integer variable mystdout. Stdout (1) will be closed, and dup2 is called to output to "mystdout".

At this point, if there isn't any piping, "execvp" will be called using cmd and the res variable. The process will then start over again after the command is called and successfully runs or fails.

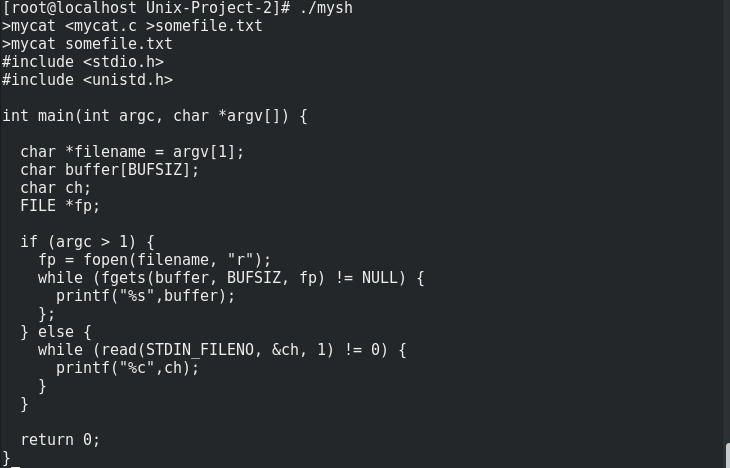
2) **myls -l**

****

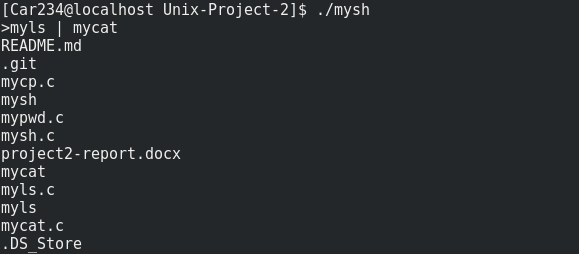
3) **ls -l**



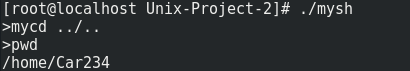
4) **mycat <a\_file >another\_file**

****

5) **myls | mycat**

****

6) **mycd ../..**

****

7) **mypwd**

**C:\Users\Carlo\Downloads\mypwd.png**