Project Report
CS 4350.251
Spring 2017
Group 12
Project 2
Shell

Mark McDermott
Jason Flinn
Carlo Rodriguez
Benjamin Winston

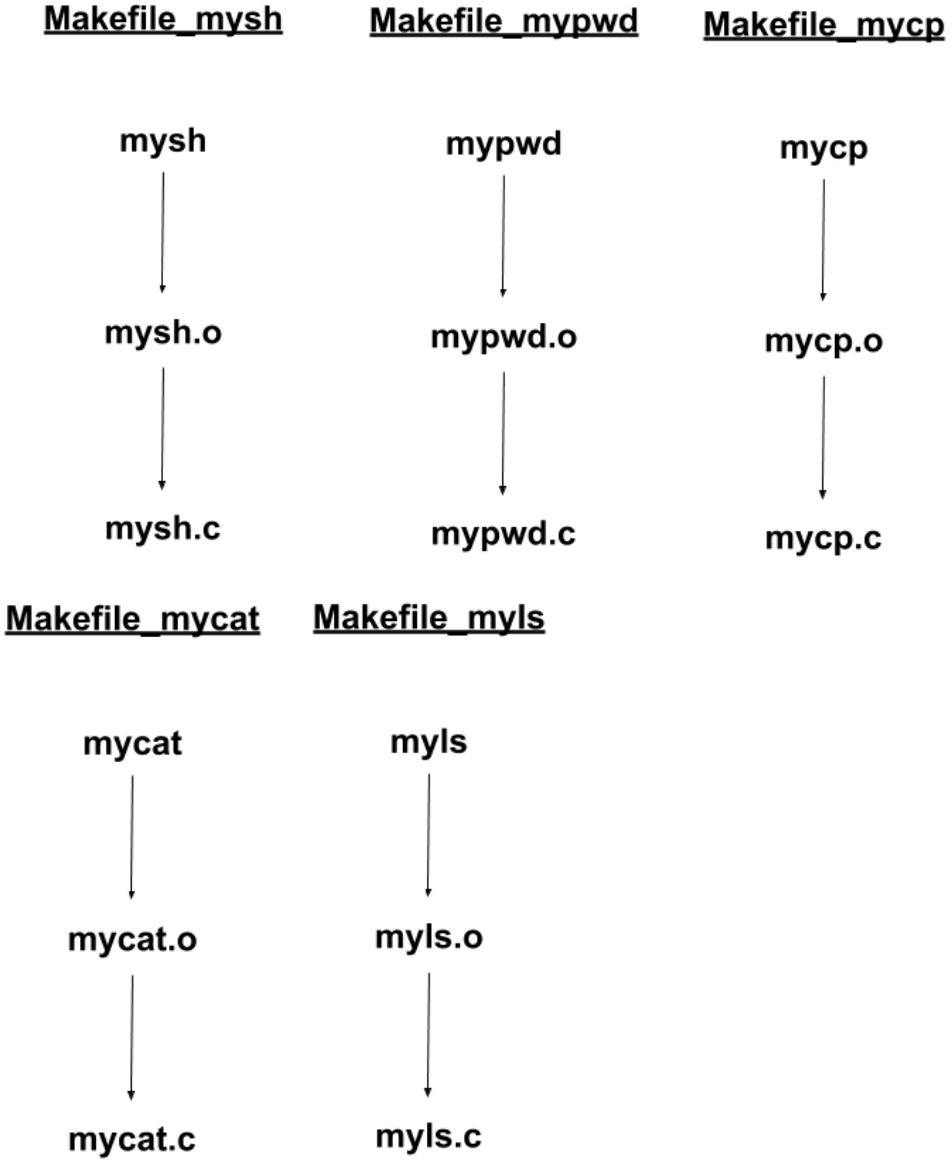
Section I

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



2) Makefiles for mysh and mypwd

Makefile_mysh

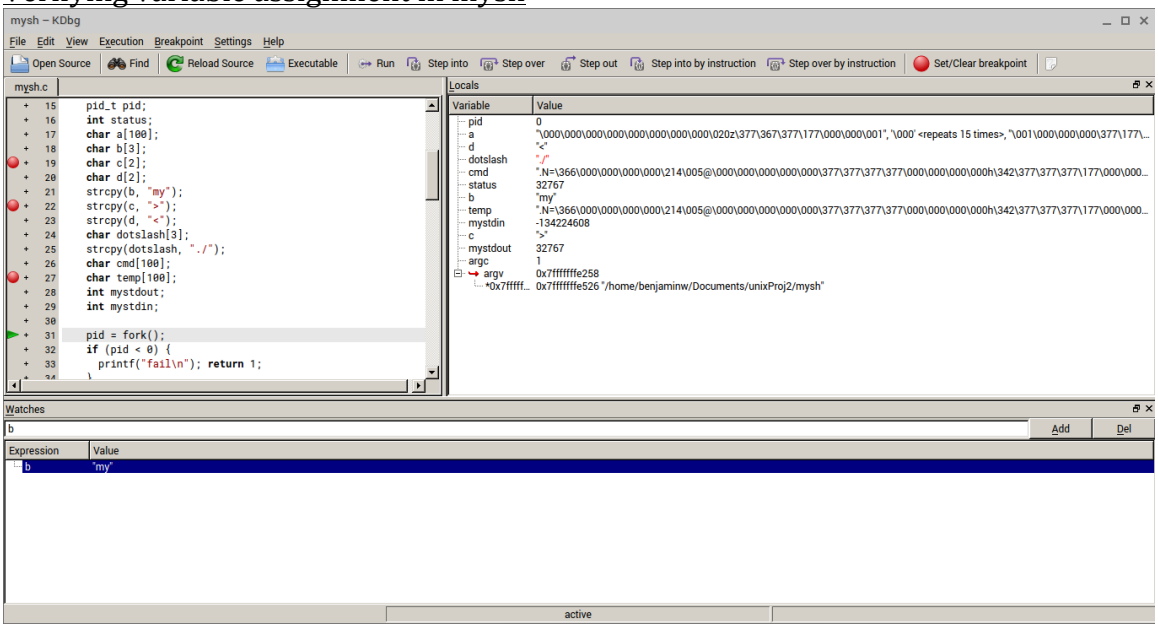
```
exe: mysh.o
    gcc -o mysh mysh.o
mysh.o: mysh.c
    gcc -g -c mysh.c
.PHONY: clean
clean:
    rm -f mysh
    rm -f mysh.o
```

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
```

3) Debugging screen shot

Verifying variable assignment in mysh



Section III

1) Description of mycat, mycp and myls

//functionality of commands and syntax of commands

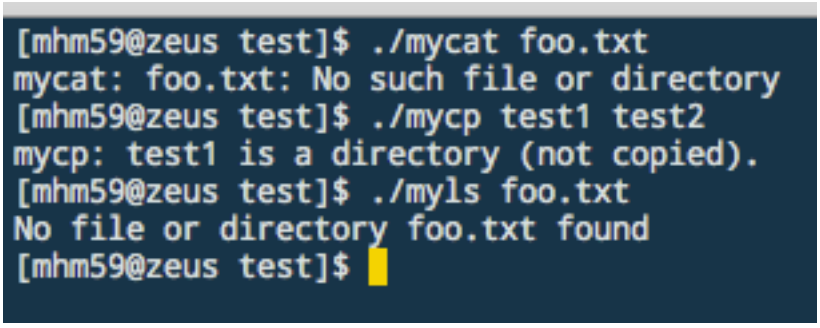
Our “mycat” program first checks for any arguments supplied, then if one or more is found, it proceeds to loop through each one. Retrieving the file info, opening the file, and printing its content using “fget”. If the file is not found, an error message is printed out. Likewise, if the supplied argument is a directory rather than a file, an error message prints out informing the user than the supplied argument was a directory. If no arguments are provided, the program handles the “./mycat <foo.txt “ and “mycat $(myls)” cases. For “./mycat <foo.txt” it doesn’t consider anything after “<” as an argument, but as stdin instead. “mycat $(myls)” manually adds line breaks on the space characters. Both cases use “read” and “STDIN_FILENO” to read and print the file.

The “mycp” program’s main function uses “getopt” to see if -R is used as an option. -R must be used to copy the directories or we throw an error. If any options besides -R are used, we also throw and error. If the source and destination arguments are not supplied, we print the message “usage: mycp <source> <destination>” to help guide the user to the correct usage. If they successfully supplied the correct arguments, we then set the “source” and “dest” char arrays equal to the appropriate arguments. Then checking if the specified source is a file or directory. If a file, we run the helper function, “fileCopy()”. If a directory, we check if the -r option was supplied. If not, we throw an error with “stderr” and print “mycp: %s is a directory (not copied).” to inform the user of their input error. If the -r option is supplied, we open the directory with “opendir”, get the directory info with “stat()”, create the directory with “mkdir(dest, 0755)”, then loop through each file. We set the source dest path with “malloc”, “strcpy”, and “strcat”, then run the helper function “fileCopy(sourcePath, destPath);”. Finally we deallocate the paths with “free()” and close the dir with “closedir()”.

“myls” has four helper functions: “getFileSize()”, “printFileDetail()”, “printDir()”, and “printDirDetails()”. “getFileSize()” returns the byte filesize of an inputted function. This is used for the formatting where “myls -l” needs to mimic the way “ls -l” formats the width of the filesize digits according to the longest filesize. “PrintFileDetail()” outputs file information such as permissions, number links, user name, group name, file size and time last modified. “printDir()” prints a simple list of files in a directory as just the file names. “printDirDetails()” prints the detailed “-l” style files with all the file details. The “main()” function uses “getopt” to see if “-l” is supplied, then set lflag to 1 if it is and throw an error if other options are supplied. We use “stat” to get the dir info.

2) **Two issues to watch out for**

For each command, you must pay attention to what arguments, if any, are passed and how to handle those supplied arguments correctly. Such as when an option is passed vs a filename. You also must be aware of the user’s permissions on the files they are trying to copy, output or view. Exceptions must be created to handle user input errors and/or invalid permissions



3)

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, and

then "continue" is called so we can start the while loop again.

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**

```
benjaminw@Snuss:~/Documents/unixProj2$ ./myls -l
-rwxrwxr-x 1 benjaminw benjaminw  8786 Apr 24 05:12 mycat
-rwxrwxr-x 1 benjaminw benjaminw 14010 Apr 25 09:59 myls
-rw-rw-r-- 1 benjaminw benjaminw  9032 Apr 25 07:32 mysh.o
-rwxrwxr-x 1 benjaminw benjaminw 16559 Apr 25 07:32 mysh
-rwxrwxr-x 1 benjaminw benjaminw  8631 Apr 24 05:08 mypwd
-rw-rw-r-- 1 benjaminw benjaminw   211 Apr 24 05:01 mypwd.c
-rw-rw-r-- 1 benjaminw benjaminw   387 Apr 24 05:12 mycat.c
-rw-rw-r-- 1 benjaminw benjaminw  4832 Apr 25 09:55 myls.c
-rw-rw-r-- 1 benjaminw benjaminw   112 Apr 25 07:31 Makefile_sh
-rw-rw-r-- 1 benjaminw benjaminw   120 Apr 25 07:36 Makefile_mypwd
-rw-rw-r-- 1 benjaminw benjaminw  1729 Apr 24 05:08 mysh.c
-rw-rw-r-- 1 benjaminw benjaminw   112 Apr 25 09:44 Makefile_myls
benjaminw@Snuss:~/Documents/unixProj2$
```

3) **ls -l**

```
[root@localhost Unix-Project-2]# gcc -o mysh mysh.c
[root@localhost Unix-Project-2]# ./mysh
>ls -l
total 44
-rw-rw-r--. 1 Car234 Car234   387 Apr 23 22:12 mycat.c
-rw-r--r--. 1 root   root    2841 Apr 25 01:29 mycp.c
-rw-rw-r--. 1 Car234 Car234   211 Apr 24 18:51 mypwd.c
-rwxr-xr-x. 1 root   root   13608 Apr 25 02:12 mysh
-rw-rw-r--. 1 Car234 Car234  2299 Apr 24 21:08 mysh.c
-rw-rw-r--. 1 Car234 Car234  5050 Apr 23 22:12 project2-report.docx
-rw-rw-r--. 1 Car234 Car234  2337 Apr 23 22:12 README.md
```

4) **mycat <a_file >another_file**

```
[root@localhost Unix-Project-2]# ./mysh
>mycat <mycat.c >somefile.txt
>mycat somefile.txt
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[]) {

  char *filename = argv[1];
  char buffer[BUFSIZ];
  char ch;
  FILE *fp;

  if (argc > 1) {
    fp = fopen(filename, "r");
    while (fgets(buffer, BUFSIZ, fp) != NULL) {
      printf("%s",buffer);
    };
  } else {
    while (read(STDIN_FILENO, &ch, 1) != 0) {
      printf("%c",ch);
    }
  }

  return 0;
}
```

5) **myls | mycat**

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

```
[root@localhost Unix-Project-2]# ./mysh
>mycd ../..
>pwd
/home/Car234
```

7) **mypwd**

```
>mypwd
/home/Car234/UnixProj2/Unix-Project-2
```