COP4610 Project 1 Report

JOSEPH DELEEUW

ROBERT MASSICOTTE

Design

General Structure of Code:

```
while program is running:
      get user input
      tokenize user input
      pass tokens to handleCommand function
```

Assumptions:

- Both input and output redirection will not appear together in a single command.
- Wildcards, escaped strings, and quoted strings will not appear in user input.
- No more than 255 characters per complete command will be entered by the user.

Development Process

8/29/2014

- Project is assigned
- Implemented while loop and prompt

8/30/2014

- User's input is now tokenized, with each token added to an array of strings

- Created `handleCommand` function, with parameters `argv` (the array containing the user's inputted command) and `argc` (the number of command arguments), to recognize and process commands

9/1/2014

- Implemented built-in utilities: exit, change directory, and ioacct
- Began implementing PATH search for other commands

9/2/2014

- Finished implementing PATH search for other commands

9/3/2014

- Fixed an error with "ls" when it is executed without any arguments

9/5/2014

- Modified PATH search to avoid using commands in hidden directories, such as "/.bin/ls"

9/6/2014

- Implemented input/output/append redirection

9/8/2014

- Fixed an error with "cd -" command not changing to previous working directory

9/10/2014

- Implemented ability to enter command with single pipe

9/11/2014

- Modified code to allow infinite number of pipes

9/12/2014

- Implemented ability to run commands in background using "&"

9/15/2014

- Fixed issue with ioacct command

9/17/2014

- Submitted project

Contributions

Joseph Deleeuw

- Implemented the tokenizing of the shell input string and the shell prompt
- Modularized the required built in functionality of the shell (i.e. ls, exit)
- Implemented the necessary built in functions
- Created the use cases to test the functionality of the shell

Robert Massicotte

- Implemented I/O redirect
- Implemented infinite piping
- Implemented ioacct and background processes
- Implemented bonus dictionary command

Missing Functionality

Background Processes

Background processes will execute in the same way as normal foreground processes. We tried many approaches to emulate the background process functionality of the UNIX shell, such as putting the processes in their own process group, as well as using `waitpid` with the `WNOHANG` argument. However, these approaches hindered the execution of subsequent commands. We felt that it was best to simply run background processes as foreground processes, so as not to diminish the overall integrity of our shell program. Given more time, we likely could resolve this issue.

Ioacct Command

When the ioacct command is entered, the rest of the command is executed as normal, and the number of bytes written and the number of bytes read are printed. However, the values are not always correct. We experienced a lot of difficulty implementing this command correctly. We tried to use a signal handler, which would catch the `SIGCHLD` signal when a child process terminated, to open the /proc/PID/io file associated with a child process. We found that the use of the handler hindered the execution of subsequent commands; and, as such, we thought it best to avoid that approach. Given more time, we likely could resolve this issue.

Changes to Make to Project Description

- More clarity, specifically on the approach of implementing the more difficult functions, like background processes, ioacct, and piping