Erik Stadler & Matthew Bauman

CS 362 – Lab1

C Shell Implementation

**Requirements**

1. **The internal shell command "exit" which terminates the shell**.

Concepts: shell commands, exiting the shell

System calls: **exit()**

**Nothing needed to be done to get the exit command working in our shell. This command was already implemented. This is done by doing a string compare and if the string matches calling the exit() function.**

2. **A command with no arguments**

Example: **ls**

Details: Your shell must block until the command completes and, if the return code is

abnormal, print out a message to that effect.

Concepts: Forking a child process, waiting for it to complete, synchronous execution

System calls: **fork(), execvp(), exit(), wait()**

**All of the commands seemed to be working except the cd command. This command was implemented with a string compare like the exit command. We also added a check to ensure that the directory you were attempting to change to existed using the chdir() function.**

3. **A command with arguments**

Example: **ls -l**

Details: Argument 0 is the name of the command

Concepts: Command-line parameters

4. **A command, with or without arguments, whose output is redirected to a file**

Example: **ls -l > foo**

Details: This takes the output of the command and put it in the named file

Concepts: File operations, output redirection. System calls: **freopen()**

**This feature was working without having to make any changes.**

**5. A command, with or without arguments, whose output is appended to a file**

Example: **ls -l >> foo**

Details: This is an append, which is a variation of the output redirect (see above)

**We added another check in the redirect\_output function that checked for another ‘>’ after the first one was found. In the do\_command function we added an additional case to handle the output in the event that two ‘>’ were found compared to just one. From there, we used the freopen() command using the ‘a’ parameter versus the ‘w+’ command that was used to just redirect.**

6. **A command, with or without arguments, whose input is redirected from a file**

Example: **sort < testfile**

Details: This takes the named file as input to the command

Concepts: Input redirection, more file operations

System calls: **freopen()**

**The sort feature was already implemented without having to make any additional changes but we added a conditional that checked to make sure that the file you were attempting to sort already existed. If the file does not exist, it prints out a message that tells the user this is the case.**

7. **A command, with or without arguments, executed in the background using &.**

*For simplicity, assume that if present the & is always the last thing on the line.*

Example: **vi &**

Details: In this case, your shell must execute the command and return immediately, not blocking until the command finishes. The distinction must be made between

backgrounding a process that does not need interactive input and one that does, e.g., the

***who*** command vs. the ***vi*** command.

Concepts: Background execution, signals, signal handlers, process groups, asynchronous

execution.

System calls: **sigset(), sigaction()**

**This feature was not working and required some signal handlers to be setup. The backgrounding is handled by calling the setpgid() function.**

8. **A command, with or without arguments, whose output is piped to the input of another command.**

Example: **ls -l | more**

Details: This takes the output of the first command and makes it the input to the second command.

Concepts: Pipes, synchronous operation System calls: **pipe()**

*Note: You must check and correctly handle all return values. This means that you need to read the man pages for each function to figure out what the possible return values are, what errors they indicate, and what you must do when you get that error.*