1 Purpose

The purpose of this assignment is to learn about processes and using system calls on a UNIX system. This is a very good chance to understand how your shell operates.

2 Exercises

This assignment will be broken down into 8 parts. This breakdown is just so you can get a handle on the entire assignment. You only need to turn in the last part since it will include all the functionality from the previous parts.

2.1 Simple shell

To start you should create a program that serves as a simple interactive shell. This shell should read the command line and try to run the program contained there. For example,

```
% /usr/bin/ls
% /usr/bin/date
% /usr/bin/uptime
```

Your shell should not look for another command to run until the previous command has finished. Use the `waitpid()` system call to be able to know when the command is completed.

2.2 Built-in commands

Now you are ready to build some commands that the shell needs to understand. Your shell needs to be able to do at least the following commands:

- `cd` changes the working directory of the shell. Basically, this is a wrapper for the `chdir()` system call.
- `exit` stops the shell's execution
- `pwd` displays the current working directory. This should be a wrapper for the `getcwd()` system call.
- `version` displays the version of the shell. This should include your name at a minimum.

2.3 Commands with arguments

Your shell should be augmented so that it can run programs with arguments. These arguments will be passed to the programs in the `argv` part of `main()` (i.e., int main(int argc, char **argv)). For example,

```
% /usr/bin/ls -l
% /usr/bin/echo Testing A B C
% /usr/bin/cat .bashrc .bash_profile
```

Arguments are always separated by whitespace. You do not have to deal with quotes to try to include spaces within the arguments.

2.4 Commands with relative paths

Next, your shell should be able to find the location of an executable by searching the `PATH` environment variable. You can use the `getenv()` system call to find the value an environment variable. The format of the `PATH` variable is a listing of directories to search separated by colons. You should search the list in the order they appear in `PATH`. Thus, if `PATH` is `/bin:/usr/bin:/usr/local/bin`, and the command that was entered was `ls`, you should try to run `/bin/ls`, followed by `/usr/bin/ls`, followed by `/usr/local/bin/ls`. Do not try to execute `ls` without a leading path. Also, do not try to execute `./ls` unless `.` is part of the `PATH` variable.

The `PATH` variable should be searched for any command that uses a relative path. Absolute paths should always ignore the `PATH` variable. Absolute paths are indicated by a command that begins with `/`, `.`, or `..`. These indicate the user explicitly does not want to check `PATH`.

```
% ls
% ./a.out
% echo Testing A B C
```

2.5 Background commands

If the command line has a trailing `&`, you should immediately try to look for another command to run. The shell should reap this command when it can, so you still need to use the `waitpid()` system call, but with different parameters. Your program should be able to handle at least 5 commands in the background. If your shell cannot handle another background command and the user asks for it, you must give an appropriate error message. For example:

```
% ./a.out &
% ./b.out &
% ./c.out &
% ./d.out &
% ./e.out &
% ./f.out &
Too many background processes
%
```

It should be very easy to handle more than 5 commands, and I encourage you to develop a data structure that is more flexible. Five is just the absolute minimum.

2.6 Input redirection

If the command line has a <, you should use the information preceding the < as the command, and the information following as a file name. For simplicity, the command line will have whitespace separating the parameters, the <, and the file name. The file name should be used as standard input to the command being run. For example,

```
% cat < input
% mail < text
% grep secret < confidential
```

2.7 Output redirection

If the command line has a >, you should use it like the < except that the file name should be used to capture the standard output of the command. Note that you will have to do a little more processing at this point because you may see a command with both < and >. For example,

```
% ls > files
% echo < input > output
% grep secret > violation < confidential
```

You should use the `open()` system call to use the output file. If it does not exist, the permissions should be the same as the current working directory. If the file does exist, its contents should be erased and replaced with the output from the command.

2.8 Pipes

If the command line has a |, you should split the command into two parts. The first part (before the |) should be command and optionally a set of arguments. Your program should interpret these as it normally does. The second part (after the |) is also a command with optional set of arguments that should be interpreted normally. The difference is that the output of the first part should serve as the input to the second part. Remember to use the `pipe()` system call like we discussed in class. For simplicity, a command with a | will not contain a < or a >. For example,

```
% ls -l | awk {print $5}
% ps -ef | grep root
% find . -name *.cpp | xargs grep TODO
```

3 Helpful hints

Again, the man pages are going to be very helpful to you. You will get the exact format of the commands along with the parameters they should be passed and the behavior of the command.

Your textbook gives a simple skeleton to start with. You may wish to use it to start off with. It is not necessary.

4 Constraints

I have listed a bunch of simplifications in the command line to try to make it easier to parse. You may try to make it accept more sophisticated input if you wish.

This assignment is probably the most common CS assignment for Operating Systems, and you will be able to find lots of code on the web to do this stuff. In fact, the code for all the commonly used shells is open source. Don't cheat by using those online resources that give you code.

You may only use C or C++ to code this assignment.