EI331 Course Project

By: Wang Haoxuan

Instructor: Wu ChenTao

November 4, 2018

I. PROJECT 2

A. UNIX Shell Programming

1. Introduction

In this part of the whole project, we are expected to accomplish a simple shell. The shell shuold be able to:

- Create the chlid process and execute the the command in the child.
- Provide a history feature.
- Add support of input and ouput direction.
- Allow the parent and child process to communicate via a pip.

The outline of the shell program is already given and we are expected to make it better.

2. Reading the Commands

It is quite a complex problem in c to get the commands we need. I first get in the whole string by $getchar()$, which read in characters one by one. Then, through the function $explain_command()$, I get the final parsed command in the form of $char**$, which is the size of the total words in the command and contains a string in each element. Data in this form can be called by the $execvp()$ function to help us implement basic shell commands. This is the outline of how our shell should work, and the following shows the extendeed functions required by the assignment.

3. Executing Commands in a Child Process

This function is implemented by examing whether symbol & is at the end of the command. If it is, we are required not to wait for the child process to finish in the parent process, which means the $waitpid$ function is not called. This could be done in a simple if clause. When the & is used, our shell would look a litte different as the "$osh>$" is output in a different manner.

4. Creating a History Feature

Our shell needs to execute the previous command when the user types in !!. A good way to implement this is by storing the previous command in a global variable, when the detected command is !!, we just use the previous command. Also, when we type in !! two times in a row, we need to use the last command that is not !!. This would also include a judgement so that we do not store !! in the previous command variable. Such a structure is put into the $main()$ function for simplication.

5. Redirecting Input and Output

This function is implemented by using $dup2$ and child process. $dup2$ redirects the output or input channel from the standard port to the port we defined. After redefining the port, we need to modify the argument so that it does not cantain $>$ or $<$ and the redirected location. In this way, we are able to read from or write to other locations rather than the shell. The specific function is defined in $my_redirect()$.

6. Communication via a Pipe

Pipe is similar to using the < and > at the same time. Thus the functon can be implemented in two ways. One is to establish a new file and store the read information into the file and then write it out. The other, which is a more formal way, is to use *pipe*() and a child process and a grandchild process, we redirect the pipe's two ends to implement the function.

The shell's function is shown as below:

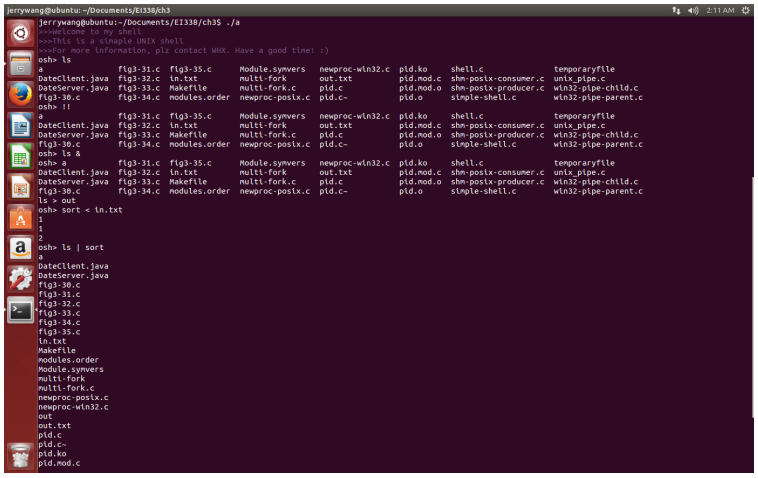

FIG. 1: Shell Functioning

B. Linux Kernel Module for Task Information

This section requires us to write to and read from the */proc* file system. We use a global variable l_pid to store the information. *sscanf* is used to write information, and a struct of pid_task is used to read the information. It's attributes such as command, pid, and state are needed to be read. We push a string of "123" into proc file system and use *dmesg* to see the output in kernel mode. Result is shown below:

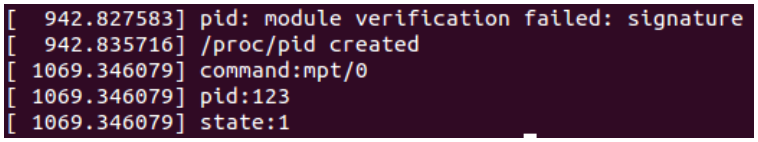

FIG. 2: Output in Kernel Mode