**Operating Systems Coursework Report**

**1. Introduction**

This coursework required to build a simple shell in C in Linux. A shell is a user interface for access to an operating system's services. The shell had implemented 4 common built-in commands in Linux, *grep* command, *ex* command to execute outer program and an additional built-in command. This report also discussed the differences of shell in other operating system and reflection through this coursework in the end.

**2. *Built-in* Commands**

There were 4 built-in commands which were required to be implemented. They were:

a. *info*: Print simple information of this shell.

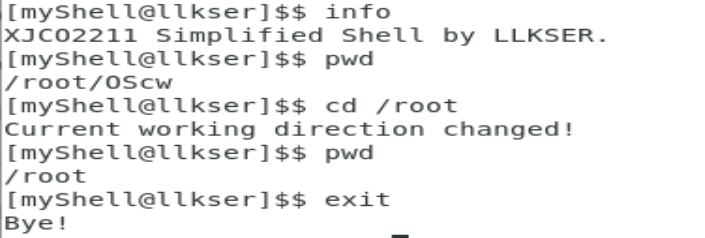
b. *exit*: Close the shell.

c. *pwd*: Print the current working directory of the shell to the standard output.

d. *cd PATH*: Change the current working directory of the shell to the directory given by the parameter PATH.

To achieve those features, the shell first read its own working directory and stored it to a string named *CWdir* via *readlink("/proc/self/exe",CWdir,255)*. Then the shell started a loop to receive user’s commands, parse those commands and execute them.

Those implemented built-in commands were showed below:



**3. String-matcher**

The shell had implemented a *grep* command: *grep -c pattern file*, which output the times of the file that satisfies the pattern match. A simple method was to compare every character in pattern string to every position in text file. Suppose the length of pattern string was n while text file was m, the method achieved an algorithm of O(n\*m).

In this shell, KMP algorithm which was modified from finite automaton had achieved O(n+m), which was far better than traditional methods. Instead of computing transition function δ(q,a) in finite automaton, KMP algorithm considered the offset of pattern string and compute function π to show if any of the offset of pattern string would be possible to match itself.

**4. Executing Outer Program**

The shell had implemented 3 *ex* commands to execute outer program. They were:

a. *ex PATH ARGS*: Execute the program specified in the PATH parameter and pass it the remaining arguments.

b. *ex ProgA | ex ProgB*: Redirect the standard output of ProgA to the standard input of ProgB.

c. *ex ProgA > a.txt*: Redirect the standard output of ProgA to the file a.txt.

To execute an outer program in Linux, the shell should first fork a new process. In C program, statement: *pid\_t pid=fork();* can fork a child process from father process. And argument *pid=0* meant the following statement was executed in child process. The statement: *wait(NULL);* kept father process waiting for the end of child process.

Then the shell executed outer program in child process by *exec* functions. The function: *execl(char\* address, char\* args[0], char\* args[1]…)* received the address of a program and its command line arguments. The *args[0]* was the name of the program. The *args[1]* and following arguments were passed to the program. What’s more, to redirect the standard output of a program to a given file, the function *dup2()* was used. The function *dup2()* changed output to the given file stream instead of standard stream. So the program would print its output in the given file.

**5. Additional Feature**

The shell had implemented an additional built-in command *ls* to list all the files under the current working directory. This feature was aimed to help users to access the file information of current working directory and check for their files easily.

The feature was implemented by functions in library <dirent.h>:

*DIR \*dirp;*

*struct dirent \*dp;*

*dirp = opendir(CWdir);*

*while ((dp = readdir(dirp)) != NULL)*

*printf("%s\n", dp->d\_name );*

*closedir(dirp);*

**6. Differences in Other Operating System**

The shell was built in C in Linux. So, it had used a lot of libraries and functions like <linux/limits.h> which were only available in Linux.

For example, function fork() created a new process in Linux while function CreateProcess() did the same job in Windows. What’s more, function readlink() was used to access current working directory of the program in Linux while function getCwd() should be called in Windows. So, if this shell was built in other operating system, the libraries and functions it used should be available in its working system.

**7. Reflection**

In this coursework, I had built a simple shell in C in Linux, through which I had learnt a lot of useful commands and functions in Linux and understood more about how a operating system was built and worked. More specifically, I learnt how to create new process and how to build a pipeline between two programs to communicate information. What’s more, in the string-matcher, I had used a more efficient algorithm KMP to achieve the feature.