Birla Institute of Technology & Science, Pilani Hyderabad Campus

CS F372: Operating Systems

Assignment 1 - (Process creation, execution, IPC)

Assigned: 29.08.2018 Date of Submission: 07.09.2018

Total Marks: 25

Problem 1: **Developing a Shell:** The shell or command line interpreter is the fundamental user interface to an Operating System. Every shell is structured as the following loop:

1. print out a prompt
2. read a line of input from the user
3. parse the line into the program name, and an array of parameters
4. use the fork() system call to spawn a new child process
    - the child process then uses the exec() system call to launch the specified program
    - the parent process (the shell) uses the wait() system call to wait for the child to terminate
5. When the child (i.e. the launched program) finishes, the shell repeats the loop by jumping to step 1.

Write a simple shell in C named as *myshell* that has the following properties:

- It should support the following internal commands:
    1. **cd <directory>** - Change the current default directory to <directory>. If the <directory> argument is not present, report the current directory. If the directory does not exist an appropriate error message should be reported.
    2. **echo <comment>** - Display <comment> on the display followed by a new line.
    3. **ls <directory>** - List the content of the <directory>. If <directory> argument is not supplied, list contents of current directory. If either <directory> is not present or points to a file, report the appropriate error.
    4. **wc <option> <file>** - When <option> is '-l' (without quotes) display number of lines in the file. Similarly, when <option> is '-w' or '-c', display count of words or bytes respectively in the file. Multiple arguments can also be supplied as: `wc -lw file.txt`, display both line and word count in this case. If the <file> is not present, <file> points to a directory or <option> is not a combination of [l,w,c] followed by '-', report appropriate error.
    5. **quit** - Quit the shell.

- **If any other command is executed report an error that the command does not exist.**
- **You should handle multiple occurrences of SIGINT signal. During the process of run, if user is typing Ctr+C for more than once, program should not terminate. Only quit command can exit the shell.**

When you have read a command you may need to split the actual command and the arguments. You can do this by using *strsep* function or *strtok*. The function strtok( ) will be helpful for parsing the input line into words separated by whitespaces (spaces and '\t' tab characters) and will place these words into an array of strings. Try man pages to learn more.

Once you've parsed the command, it's time to execute it. This can be done using the function *execv*. This function takes two arguments. The first is the path to an executable, while the second is an array containing all the arguments. This function turns the current process into another process. Doing so allows us to execute a command, but in return we lose our shell process. You may use any other version of *exec* i.e. *execlp* etc. The solution is to use the function *fork*, which makes a copy of the current process. When *fork* is called we have two nearly identical processes, both at the same point in the code. The only difference between the two is the return value of the fork function. The original process will get the process id of the new process, while the new process gets a return value of 0 (zero). You may use an if-statement to have the new ('child') process call *execv* to run the command, while the original ('parent') process can call the *wait* function to wait for the child to finish.

[5 Marks]

Problem 2: **Adding a System Call to the Linux Kernel:** A system call is the standard way an OS service is exported to a user program. For example, a system call may be used to give users access to internal information of a file system such as superblock or metadata stored at boot sector for file systems. The procedure to create a system call varies in different versions of Linux. When you add a system call you will have to recompile the Linux kernel, so you must have the Linux source tree installed on your system. Using the doc shared on CMS as reference and also the demo shown on the tutorial classes, add a system call to the Linux kernel (version 4.18.1) whose functionality is same as the fork() system call.

Use the following function to test your system call.

```
void myfork_test(){
     myfork();
     printf("Hello World!");
     return;
}
```

Some references:

- [obj-y] whats-meaning-of-obj-y-something-in-linux-kernel-makefile
- [ptregs] ptregs-in-syscall-table
- [custom syscall name] how-to-use-my-custom-system-call-with-its-name-and-not-with-its-number
- [terms in syscall table] parameters-of-files-syscall-32-tbl-syscall-64-tbl-in-build-linux-kernel/
- [fork - Linux kernel 2.6] where-is-the-source-for-the-fork-call-in-linux
- [fork - Linux kernel 4.6] How-does-the-fork-system-call-in-Linux-work (similar to kernel 4.18.1)

[6 Marks]

Problem 3: Write a C program that forks a child process. The parent has some integers say {11,12,13,14,15,16,17,18,19,20} stored in an array. Parent selects 2 numbers at random (say x and y, where x!=y) from the array; prints x and y; passes y to child; and sleeps for x/3 seconds. The child prints whether 'y' is a prime number or not; prints S; and goes to sleep for y%3 seconds. S is the sum of all primes observed by the child till now. The process stops when all the numbers are visited at least once or on an outside interrupt (like Ctrl+C) is received. Do not use any global variables.

[4 marks]

Problem 4: Write a C program that takes parameters n, k, r as an input from the user. Parent will fork a child which will print k processes with highest CPU usage every n seconds. You need to print those processes in the decreasing order of CPU usage percentage. You should use pipe, fork, exec family, dup2 and sleep system calls to implement the program.

After every r iterations(i.e. after the output (top k processes) has been printed r times on the screen) parent will send a PID to the already spawned child and the child will kill that process. Parent shall take the input of the PID to kill from the console. If input from console is -1 child should not kill any process.

Do not use any global variables. Look into man page of ps for BSD syntax (to see CPU usage). You can also find man pages of sort and head unix commands useful.

[6 Marks]

Problem 5: When you run grep in Unix, we get lines containing the patterns:

Say,

```
bash$ grep -n int prog.c
12: int a = 1;

17: printf("%d", a);

34: char s[10] = "interger";
```

Where, each matching line is preceded by a line number and a colon. Now write a program newgrep which just prints the line numbers. This is to be done by using Pipes.

Say,

```
bash$ newgrep int prog.c
12

17

34
```

(Hint: newgrep is to create a pipe and then create a child to exec the corresponding grep command. The parent process is to read the output from its child and throw away everything but the line numbers. For this, parent may use another feature of unix ,i.e, cut.)

[4 Marks]

Submission Instructions: You may form your own group of maximum **3 students**. All your programs must run over Ubuntu machines. Submit source files and executables. Also submit a *readme.txt* with your group details. Put all your deliverables into a tar file (like, *f20160xxx.tar*) and send it to p20150005@hyderabad.bits-pilani.ac.in as a mail attachment. Copied codes will be awarded zero marks. Every assignment will have a demo session which will be intimated later through a separate notice.

For any queries, contact:
Rajesh Kumar Shrivastava <p20150005@hyderabad.bits-pilani.ac.in>
Sanket Mishra <p20150408@hyderabad.bits-pilani.ac.in>
Bhavesh Gawri<f20150116@hyderabad.bits-pilani.ac.in>