CS 4323 Design and Implementation of Operating Systems I

Assignment 02: Full Marks 100
(Due Date: 10/26/2019, 11:59 PM CST)

This is the mini project assignment that must be done in the group as specified. If you do not know the group members, then please refer the module **MiniGroupProjectFormation** under **Home** page in Canvas.

Note: You will not gain any points, if you do not use child, parent and exec concepts to implement the shells.

Part 1: Getting familiarize with basic Unix command

To those who are not familiar with UNIX shell environment, this part may be helpful. For those, who are very familiar with Unix shell environment and command, you can move on to **Part 3**.

1. **man**

   Online manual help for any inputted command.
   To get help about "ls" command and all possible options that can be used with it, do:

   man ls

2. **ls**

   Shows all the major files and directories in the current directory.

   Note: Everything is treated as a file in Linux. Even a directory is a file.
   ls

   Desktop Downloads Music Pictures Templates VMShare
   Documents examples.desktop OperatingSystem Public Videos

   You can use other options with ls like -a, -d, -I and so on. Use **man** command to know more about all other options.

   For example, **-l** option lists file with long format showing file permission.

   ls -l

   total 52
   drwxr-xr-x 2 shital shital 4096 Jul 29 04:57 Desktop

drwxr-xr-x 2 shital shital 4096 Jun 13 21:45 Documents
drwxr-xr-x 2 shital shital 4096 Jun 13 23:13 Downloads
-rw-r--r-- 1 shital shital 8980 Jun 13 21:27 examples.desktop
drwxr-xr-x 2 shital shital 4096 Jun 13 21:45 Music
drwxr-xr-x 3 shital shital 4096 Aug 8 19:39 OperatingSystem
drwxr-xr-x 2 shital shital 4096 Jun 13 21:45 Pictures
drwxr-xr-x 2 shital shital 4096 Jun 13 21:45 Public
drwxr-xr-x 2 shital shital 4096 Jun 13 21:45 Templates
drwxr-xr-x 2 shital shital 4096 Jun 13 21:45 Videos
drwxr-xr-x 2 shital shital 4096 Jun 13 22:22 VMShare

-i option shows file's inode index number.

928747 Desktop	926080 examples.desktop	928753 Pictures	928754 Videos
928751 Documents	928752 Music	928750 Public	1049917 VMShare
928748 Downloads	917656 OperatingSystem	928749 Templates	

For now, remember every file has associate unique inode that will be used by the file system to identify the files.

3. pwd

The directory you are standing in is called the working directory. The **pwd** command shows the path of your current directory. It prints the path of the working directory, starting from the root. Type:

pwd

and see your current directory.

4. cd

The **cd** command changes directory. If you wanted to change from the home directory to the *Desktop* directory, you would input the following command:

cd Desktop

which will change your current directory from:
shital@shital-VirtualBox:~$** to **shital@shital-VirtualBox:~/Desktop$

An absolute pathname begins with the root directory and follows the tree branch-by-branch until the path to the desired directory or file is completed.

For example, my current working directory is: /home/shital/OperatingSystem/MyImplementations

Inside this directory, there are number of other directories:

Implementing_FileReadWrite	Mutex	RemoteProcedureCall	Threads
ImplementingPRINTF	NamedPipe	Scheduling	InputFiles
ImplementingPRINT&SCAN	ordinaryPipe	Semaphore	Process
Sockets			

To move to *Scheduling* directory, you can use relative addressing:

cd Scheduling

or,

cd ./Scheduling

You can also use absolute addressing to move to *Scheduling* directory:

cd /home/shital/OperatingSystem/MyImplementations/Scheduling

The "**.**" symbol refers to the working directory itself and the "**..**" symbol refers to the working directory's parent directory.

To change the working directory to the parent of *Scheduling* directory, you can do:

cd ..

Other useful UNIX commands are:

S.No.	Command	Example	Description
5.	**mkdir**	mkdir Lab1	Makes a new directory **Lab1** in the current working directory.
6.	**rmdir**	rmdir Lab1	Removes directory **Lab1** (the directory must be empty)
7.	**cp**	cp file1 ./document	Copies **file1** into a directory called **document**
8.	**rm**	rm file1	Removes/deletes file **file1** from the current directory
9.	**mv**	mv file1 newfile1 mv file1 dir/	Renames the file **file1** to **newfile1**. Move the file **file1** inisde the directory **dir**.
10.	**nano**	nano file1	nano is an editors. Just like notepad is for Windows. **file1** is a new file name. A blank screen will appear where you can write.
11.	**touch**	touch file1	Makes an empty file **file1**.
12.	**cat**	cat file1	Displays the content of the file **file1**.
13.	**clear**	clear	Clears the screen
14.	**gcc**	gcc file1.c	Compile a program **file1.c** written in C.
15.	**more** **less**	cat test | more cat test | less cat test | more -n	If you have large file **test** which will not fit in output terminal, then you can use option "more" or "less". **-n** option allows view first n lines of the file **test**.

16.	**grep**	grep “include” file1.c	Searches the input file **file1.c** for lines containing a match to a given pattern list **include**.
17.	**>** **>>**	cat test1 > test2 cat test1 >> test2	Output Redirection operator: > redirects the contents of **test1** to **test2**. The contents of **test2** will be overwritten. >> redirects the contents of **test1** to **test2**. The contents of **test1** will be appended at the end of **test2** file.
18.	**<**	sort < file1	Input redirection operator: < is used to input items from the file **file1** to the sort command sort command is used to sort the items Note: Consider the contents of **file1** is: apple banana grapes oranges watermelon avocado carrot berries
19.	**|**	cat file1 | grep “apple”	Pipe command lets you send the output of one command to another. It can redirect the standard input, output or error of one process to another for further processing The contents of file **file1** is given as input the **grep** command to search for **apple**.

Part 2: Background process using &

Till part 3, you have used shell to run the command in the foreground. Just to understand, run the following command in the terminal:

for i in {1..10} do sleep 1; done

This command makes the process sleep for 10 seconds. With *sleep 1*, it sleeps for 1 second and you are running the command 10 times. So, a total of 10 second sleep.

During this time, you will now see a blinking cursor on your terminal. The shell is working for the request in the foreground. During this time, you cannot execute any other command in the same terminal. If you want to execute any other command in the same terminal, then you have basically two options, other than opening a new terminal:

1. kill the process by using: CTRL + C.
2. suspend the process by using: CTRL + Z

Note: Very few processes have a dedicated quit function. For example, type:

```
top
```

in your terminal. The command **top** displays a dynamic real-time view of the running process and system status. You can terminate this process by typing "q".
When you suspend the process, by using CTRL + Z, the process is paused and the control is return to the terminal. You can then execute another process in the same terminal. You can try this with the server program.

You can see the status of the process in a static way by using **ps** command (as contrast to **top** command). By typing:

```
ps T
```

in the terminal, you will see the status of the process under STAT column.

The suspended process can be resumed again using: **fg** command.

Instead of doing all of these, you can run the process in the background by using **&**. Run the same command as:

```
for i in {1..10} do sleep 1; done &
```

The ampersand character **&** tells the shell not to wait for the process to complete but return the prompt to the user.

Now, immediately you can run any other process in the same terminal. You can try the following command:

```
ls
```

To see all the stopped or background processes, you can use **job** commands as:

```
jobs
```

It shows all the background processes where the first column represents the job number. You can use that number to stop any background process. For e.g.:

```
kill %1
```

kills the process which is references by %1.

Note: Please make a note that each command (running either at the foreground or the background) is a process.

If you have run a process in the foreground and want to move them to the background then you can do:

1. stop the process by CTRL + Z
2. use **bg** command to move the last stopped process in the background.

However, you must be careful that not all commands can run in the background. Those process may terminate automatically.

Similarly, to move the background process to the foreground, you can use **fg** command. This bring the last backgrounded process (indicated by + in the job output) to the forground. If you want to bring particular background process (for e.g., job number 1) to the foreground, then use:

```
fg %1
```

Part 3: Basic Shell Interface

The task in this assignment would be to design your own shell interface using C program. Please make sure that the program should work in CSX machine.

You need to use a fork() command to create a child process. The child process will invoke the exec command. (Please note that there are different family of functions in exec, you can explore them).

Your shell should look something like this when you first run your program:

shjoshi@CS4323shell:/home/shjoshi~$

where shjoshi is my O-key Account Username (yours should be different), followed by @CS4323shell:. It should then follow with the path of your home directory (in my case, it is /home/shjoshi, you should use your own path). Finally, you should have ~$. Please note that ~ should only appear if you are in your home directory. Otherwise, it should not appear. For example, when I am inside OS4323_Fall2020 directory inside my home directory, ~ disappears:

shjoshi@CS4323shell:/home/shjoshi/OS4323_Fall2020$cd

When you enter cd, it goes back to your home directory and ~ should appear again.

shjoshi@CS4323shell:/home/shjoshi~$

All your commands should be entered after $.

Part 4: Basic Shell Command

Your shell should support all the basic UNIX command given in part 1, including:

- redirection using >, <, >>, <<, >&, <&, 2>&1.
- Multiple pipe commands using |
  E.g.:
  cat text.txt | sort | head -n 3 | tail -n 2

  where text.txt is any text file containing anything like fruits name.

Part 5: Shell Command in Background

In this task, you need to be able to perform operations in background and should be able to display all the operations running in the background.

Part 6: History

In this task, your shell should be able to remember all the command you have entered so far. When you type history, it should be able to display all the command on the console.

Part 7: Client-Server interface

The shell should be working at the server side and the shell should get the input from the client program. So, you need to develop both client-server program.

Points distribution for this project will be as follows:

Part 3: Basic Shell Interface	**[7 points]**
Part 4: Basic Shell Command	**[18 points]**
Part 5: Shell Command in Background	**[25 points]**
Part 6: History	**[20 points]**
Part 7: Client-Server interface	**[20 points]**
Report and group coordination.	**[10 points]**

Grading Criteria:

This is a group project. So, grading will focus on students' ability to work in the group and successfully complete the work. Each member in the group should coordinate as a team rather than individual. And that's the key to score maximum points.

Points to remember:

- Failure to complete the project as a group will deduct 20 point, even though every individual has completed their part. As this is a group project, you need to learn to work in the group and meet the team's objective, and not individual objective.
- Work must be distributed uniformly among all group member and should be clearly specified in the report.
    - Be sure to submit a written report that summarizes your design and all the test cases you have performed.
    - Include each student's contribution clearly. Failure to be specific on this part will make us difficult to give fair grades. So, it is your responsibility to clearly indicate what functions have been implemented by which member in the group.
- It is not necessary that all group members will get equal points. It depends on what responsibility each student has taken, whether or not the student has delivered the task.
- It is the group's responsibility to alert instructor with any issue that is happening in the group.
    - Students are supposed to use the group created in the Canvas for all correspondence. That will serve as the proof, in case there is any dispute among the group members.
    - Time is very important. Group members are expected to respond quickly.
    - You need to have sufficient days to combine individual work. You are going to make one submission as a group and not individually.

Submission Guidelines:

- You should submit your programming assignment as a single .c file:
- Include the readMe.txt file that shows how to run your code.
- Your code should include the header information, which should include:
    - Group Number
    - Group member name
    - Email
- There should be sufficient comments in the program.
    - Each function should be clearly described along with the input arguments and return values.
- Report is to be submitted as pdf and the last page of the report should include all the code.