Project Description

Objective: In this project you are going to create your own remote shell that simulates some services of the OS including the current Linux shell features, processes, threads, communication, scheduling, etc. The project is divided in four related phases/parts in order to build it progressively during the coming six weeks. A group of 2-3 students is required.

Phase1: Local CLI Shell **Due Date: Oct 28**

Phase2: Remote CLI Shell **Due Date: Nov 4**

Phase3: Remote Multitasking CLI Shell **Due Date: Nov 18**

Phase4: Remote Multitasking CLI Shell with Scheduling Capabilities + Report + Demo **Due Date: Dec 2**

You have to develop the following:

1- **Phase1-Local CLI Shell**: In this phase, you have to develop using C your own shell/CLI that replicates features from the linux one such as ls, ls-l, pwd, mkdir, rm, … or a program to execute including its name. You have to get my approval on the final list of features that you will develop). You can use the techniques that you have learned about creating processes (e.g. fork, excec), interprocess communications (e.g. pipes), etc.

2- **Phase2-Remote CLI Shell**: In this phase, you have to upgrade Phase1 with remote access capability to your shell (implemented in the server) through Socket communication. In this context, the client takes inputs from the user and sends requests to the server. Each input from the client can be either a command (such as ls, ls-l, pwd, mkdir, rm –r, …) or a program to execute including its name. The client will then receive the output/result from the server and print it on the screen.

3- **Phase3-Remote Multitasking CLI Shell**: In this phase, you have to upgrade Phase2 server with multitasking capability using threads.

The server should be able to serve several clients simultaneously. Each thread should handle the request, communication and computation with one client.

4- **Remote Multitasking CLI Shell with Scheduling Capabilities (will be discussed in details after finishing chapter 5)**: In this phase, you have to upgrade Phase3 with scheduling capabilities in the server, which should simulate the role of scheduler to provide concurrency for serving several clients. To avoid dealing with memory and process management, assume that you have only one CPU and only one main thread can run on that CPU; then all the requests (threads of phase3) should schedule among them to run on that main thread. All received requests (from one or more clients) will be added to the waiting queue, including all the relevant information that you find necessary for your scheduling algorithm. Each request/task will be removed from the queue by the selection algorithm, executes on the main thread for some time and then exits if done or returns back to the waiting queue for further execution if not done yet. In this context, you have to handle the following requirements:
   - If the received request is a shell command, then it can simply be executed by creating a process and call the corresponding system command to execute. Hence, it will complete execution and exit from the first round.
   - If the received request is a program, then it will run until it finishes or until the end of the allocated time based on the adopted scheduling algorithm. If the needed execution is done, then it will exit, otherwise it will return back to the queue for further execution rounds from the point it stopped (Not restarting from the beginning).
   - Your combined scheduling algorithm must include RR with different quantum based on the rounds and SJRF, similar to the ones explained in class. (you can add priority too)
   - For simulation purposes, the program to be executed can simply have a loop with sleep function in it, which will loop “n” times and print the index of each iteration. The value of n represents the amount of work to be done. Each iteration will be considered one value of time (e.g. 2 iterations will be considered 2 seconds or 2 Milliseconds).
   - Execution Scenarios: In case the server is handling a request that solely includes a command, then the server should schedule it for one round of execution. In case the server is handling a request

with the value n, this means it will simulate the execution time of a process based on the "n" value. If another command or request arrived, the scheduler module should stop the execution of the current task if the scheduling algorithm selects this option based on your set conditions, save its id and the remaining value of "n", and execute the new task. Once it is done, the server should loop over the list of waiting tasks and schedule the one that fits the best based on your scheduling algorithm and remaining amount of execution time. That being said, your server should always keep a data structure that holds all the tasks and their required information.

The solutions require the use of process creation, inter-process communication using sockets and pipes, multiprocessing, multithreading, scheduling and synchronization. In your project, you are supposed to create a fully functional remote shell including your own time-based scheduler. Don't be limited to any implementation techniques described above. You can implement the above requirements using your own techniques. Comment your code. Add a few test cases that demonstrate the functionalities of your project. Add a report including the usage of your program, test cases used and their results, and a detailed description of your implementation. Finally, be ready for a demo.