Project – 4 Phases

Operating Systems (CS-UH 3010) — Fall 2024

1 Code of Conduct

All assignments are graded, meaning we expect you to adhere to the academic integrity standards of NYU Abu Dhabi. To avoid any confusion regarding this, we will briefly state what is and isn't allowed when working on an assignment. Any documents and program code that you submit must be fully written by yourself. You can discuss your work with fellow students, as long as these discussions are restricted to general solution techniques. Put differently, these discussions should not be about concrete code you are writing, nor about specific results you wish to submit. When discussing an assignment with others, this should never lead to you possessing the complete or partial solution of others, regardless of whether the solution is in paper or digital form, and independent of who made the solution, meaning you are also not allowed to possess solutions by someone from a different year or course, by someone from another university, or code from the Internet, etc. This also implies that there is never a valid reason to share your code with fellow students, and that there is no valid reason to publish your code online in any form. Every student is responsible for the work they submit. If there is any doubt during the grading about whether a student created the assignment themselves (e.g., if the solution matches that of others), we reserve the option to let the student explain why this is the case. In case doubts remain, or we decide to directly escalate the issue, the suspected violations will be reported to the academic administration according to the policies of NYU Abu Dhabi (see https://students.nyuad.nyu.edu/campus-life/community-standards/policies/academic-integrity/).

2 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 provided as graded assignments in order to build it progressively during the semester. A group of 2 students is required.

Phase1-Prog. Assi1: Local CLI Shell **(Due Date: October 3)**

Phase2-Prog. Assi2: Remote CLI Shell **(Due Date: October 24)**

Phase3-Prog. Assi3: Remote Multitasking CLI Shell (Due Date: November 14)

Phase4-Prog. Assi4: Remote Multitasking CLI Shell with Scheduling Capabilities + Final Report + Final Demo (Due Date: December 5)

3 Implementation

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 should implement at least 15 commands and should include the composed commands including 1 pipe (…|…), 2 pipes (…|…|…), 3 pipes (…|…|…|…), input redirection, output redirection and combinations of all. You can use the techniques that you have learned about creating processes (e.g. fork, exec), inter-process 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, …) , composed commands 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 detail 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.

3 Grading

Description	**Score (/100)**
Phase 1: 25 (10 points on the single commands and 15 points on the composed commands) Phase2: 20 (proper remote execution of the commands by one client) Phase 3: 20 (proper multithreaded remote execution of the commands by multiple clients) Phase 4: 15 (proper scheduling among several clients)	80
Proper error handling, documentation, and demos	15
Report	5

You should solve and work **in group of 2** on this project. The deadline of each phase (Programming Assignment) is listed above in this document. and on NYU Brightspace.

You should directly submit your **C source files and Makefile** on NYU Brightspace, followed by scheduled demos with the lab instructor. Submissions via email are unacceptable.

Note that your program should be implemented in C and **must be runnable on the Linux operating system**.

All the phases (programming assignments) are due at 11:59pm on the due date. For late submissions, 10% will be deducted from the homework grade per late day. Late submissions will be accepted only up to 2 days late, afterwards you will receive zero points.