Project Phase 1

Due Date: 26th February 2026

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/).

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. The spreadsheet for group selection has been shared with you during the lab. The deadline for group selection and filling this sheet is on Monday 16th February 11:59 pm. A **late penalty of 10%** will be applied in case the group hasn't registered in the sheet by the deadline.

Your task for phase 1 is to create an application using C your own shell/CLI named *myshell* where you parse command line input from the user and execute the command as child process. The objective is that you become familiar with fork(), exec(), wait(), dup2(), pipe() in Linux.

The design for *myshell* program will essentially consist of parsing through the input command strings, and executing the commands using system calls. Consider the following elements:

1- Shell Commands with No Arguments (ls, ps etc)

Each command irrespective of whether they have arguments or no arguments will be executed in an individual *child* processes spawned for its purpose from the main process. The main (parent myshell process) will fork and wait till all its child processes are finishes to prompt the user again for input. A simple pseudo code to describe this functionality is given below:

```
If fork doesn't equal 0
        In parent; wait for all child processes
Else
        Execute command
```

The system call used to execute the command will be execvp(). The first element of the input array will be the program name. execvp() will automatically search for this program in the path environment, and execute.

2- Shell Commands with Arguments (ls -l, ps aux etc)

This will be similar to the previous category, except that the command and its arguments will be provided as a string array arguments to execvp().

3- Redirecting output

Should implement output and error redirection (>, 2>). You can use dup2() system call to achieve this.

4- Redirecting input

Should implement input redirection. You can implement this feature using dup2() system call.

5- Pipes

It connects the standard output of one command to the standard input of another. You do this by separating the two commands with the pipe symbol |. Your shell has to wait for the last process in the pipe-line to terminate to show the prompt for the next command. This can be achieved using pipe() system call. Your shell should technically support *n* number of pipes.

6- Program to Execute

As seen in the exec() examples in the lab, your shell should also be able to execute any program executable with its name such as ./hello for instance.

7- Composed Compounds Combinations

You should also implement combinations of composed commands including all input redirection, output redirection, and pipes including **all** the following:

- `command < input.txt`
- `command > output.txt`
- `command 2> error.log`
- `command1 < input.txt | command2`
- `command1 | command2 > output.txt`
- `command1 | command2 2> error.log`
- `command < input.txt > output.txt`
- `command1 < input.txt | command2 > output.txt`
- `command1 < input.txt | command2 | command3 > output.txt`
- `command1 | command2 | command3 2> error.log`
- `command1 < input.txt | command2 2> error.log | command3 > output.txt`

8- Error handling

Make sure to handle different kinds of potential errors including missing arguments, incorrect commands, etc.. Here are some error examples that are expected to be handled:

- **Missing input file**: `command <` (Input file not specified.)
- **Missing output file**: `command >` (Output file not specified.)
- **Missing error redirection file**: `command 2>` (Error output file not specified.)
- **Missing command after pipe**: `command1 |` (Command missing after pipe.)
- **Empty command between pipes**: `command1 | | command2` (Empty command between pipes.)
- **Invalid command**: `invalid_command` (Command not found.)
- **Invalid command in pipe sequence**: `command1 | invalid_command | command3` (Command not found in pipe sequence.)
- **Output redirection missing target**: `< input.txt | command1 >` (Output file not specified after redirection.)
- **File not found**: `command < non_existent_file.txt` (File not found.)

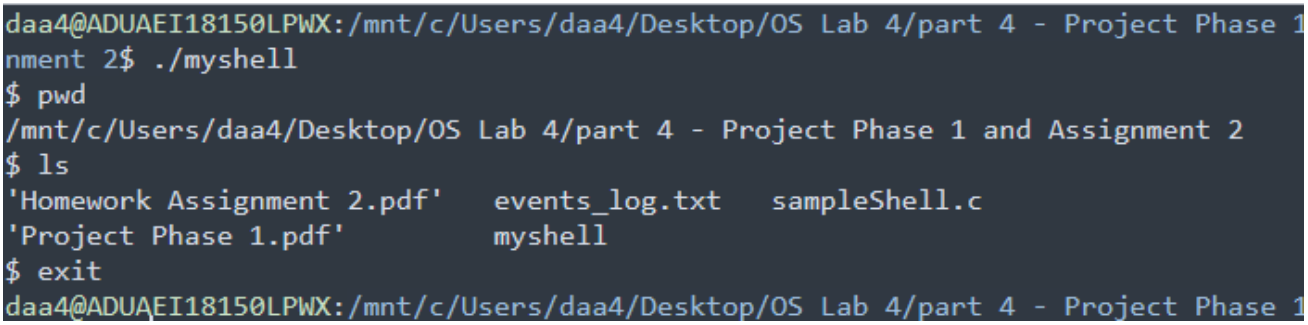
Report Guidelines

With each project phase code submission, you will be preparing a report as well. The following table shows the required guidelines.

Section	Description
Title Page	Includes the phase number, group member names, and NetIDs.
Architecture and Design	Describes the high-level structure of the system, key design decisions, reasons for using specific algorithms/data structures, file structure, and code organization.
Implementation Highlights	Explains the core functionalities, including important functions, algorithms, and logic. Includes references to critical code snippets. Also describes how errors and edge cases were handled.
Execution Instructions	Provides instructions on how to compile and run the program.
Testing	Lists the test cases performed. Includes screenshots and text explanations detailing the tests conducted and the output received.
Challenges	Describes any difficulties and challenges encountered during development and explains how they were resolved.
Division of Tasks	Clarify how the tasks/responsibilities were divided across both team members
References	Cites any resources used during development.

Shell Output Format

Your shell should look like a regular Unix shell. You shouldn't have any extra elements in the display. So, it should follow the following screenshot: When you run the myshell executable, it just shows you a $ on a new line, takes in the command, displays the output on the following line and display another $ in the following line. Remember that "exit" command should exit your shell.



Grading Rubric

Description	Points (/30)
Successful compilation with a Makefile on remote Linux Server	1
Single Commands (without & with arguments)	3
Input, output and error redirection	3
Pipes	5
Composed Compound Commands	10
Error Handling	3
Detailed Meaningful Comments (throughout the entire code fully explaining your logic and code implementation)	2
Code modularity, quality, efficiency and organization	1.5
Report	1.5

Noteworthy Points

- Make sure to use separate compilation and apply best practices in your code submission. Aim for high efficiency, reduction of redundant code, and scalability to facilitate updates in future phases.

- Extensive and proper commenting must be done for all the program. Avoid short comments before huge code blocks. They should be meaningful, detailed and explaining all your code logic.

- Make sure to use **exit** to exit the program

- Your submission **must** be working on the **remote Linux server** successfully.

- Submissions:
  A .zip file containing the following:
  - C files + Make file
  - Report (**Must be in PDF format**)