Advance Systems Programming
In-class Activity 1

Important Note 1: If you are an on-campus student, we expect you to attend the in-class activity via Zoom and work in a team of size two. Students in the EDGE section can work on the solution offline. Each student must submit a solution of their own on ELearning by the due date. Please write down the name of your teammate, If any, in your submission.

In this activity, you are going to write a C program (in pseudocode) that simulates a shell. A shell controls a terminal and executes commands entered on the command line, which can be as simple as "execute a single program" or "execute multiple programs based on a specific set of rules". It is important to note that the shell can execute any program if all necessary command line arguments of that program are provided along with the name of the program (executable).

Important Note 2: You do not need to write any command line parsing logic in your solution. You can assume that the command line has already been divided into tokens and you can refer to them the way you want. So, please focus on using the system calls properly to achieve the expected behavior.

Assume that the shell has the following structure:

Display the command line symbol ($)
While a command or command sequence to read on the command line do
    Parse the command line into tokens
    Process the command line (using a combination of fork, exec, wait/waitpid, pipe, close, dup/dup2)
    Display the command line symbol ($)
End while

You are going to use the system calls we have covered (fork, exec, wait/waitpid, pipe, close, dup/dup2) to implement (in pseudocode) the following 6 cases of "Process the command line", where prog, prog1, and prog2 represent the names of some executables. You can assume that directories that host the executables are included in the PATH environment variable. Also, you can assume that the input (command line) has already been parsed and divided into tokens (you do not need to write the parsing logic).

Case 1: Output redirection

$ prog args > file

Execute prog in a child process and redirect the output on the standard output to the file

Case 2: Input redirection

$ prog args < file

Execute prog in a child process and redirect standard input to the file as if the standard input is coming from the file

Case 3: Pipe

$ prog1 args1 | prog2 args2

Execute prog1 and prog2 in parallel (in two different child processes) and redirect standard output of the process running prog1 to the standard input of the process running prog2.

Case 4: Sequential

$ prog1 args1 ; prog2 args2

First execute prog1 in a child process and once it terminates execute prog2 in another child process.

Case 5: Conditional AND

$ prog1 args1 && prog2 args2

First execute prog1 in a child process and if the return status is 0 then execute prog2 in another child process.

Case 6: Conditional OR

$ prog1 args1 || prog2 args2

First execute prog1 in a child process and if the return status is not zero then execute prog2 in another child process.

To get full credit, you should use the mentioned system calls appropriately. Note that you are not allowed to use the system function as doing so does not demonstrate your knowledge of the specific systems calls we mention above!