1 Introduction

You should reread section 1.6.1 of the class-notes.

In this lab you will implement (in C or C++) an extremely limited shell (a.k.a. a coommand line interpreter). (All 202 labs must be written in C/C++.) Your shell will read a command name from the terminal and, depending on the command, either

1. Directly implement the command if it is built into your shell. A list of commands you are to build into your shell and their meaning is given below. I sometimes refer to these as the *simple* commands or the *type 1* commands.

2. For commands not built into your shell, invoke the four main Unix system calls for process management, namely `fork()`, `execve()`, `wait()`, and `exit` to have the (Unix) operating system on `linserv1.cims.nyu.edu` implement the command for you. I sometimes refer to the non-simple commands as *internal* commands since the real implementation is internal to Unix. I also sometimes call them *type 2* commands. Some *type 2* commands you should test are `dir`, `echo`, and `date`.

For simplicity, I have chosen the commands you are to implement, be they type 1 or 2, to be ones that take no arguments.

2 High Level Pseudocode

```
Loop {
    Obtain a command name from the user
    Replace command's trailing newline with a '\0'
    If (command is type 1)
        Implement the command directly
    else // the fun begins {
        parentOrChild = fork()  // now your lab has two!! processes running
        if (parentOrChild == 0) {   // code executed by the child process
           // Calculate arguments for execve() and execute it
           // The execve'ed command exits }
        else // code executed by the parent process
           wait() // parent waits for child to exit
    }
```

3 Some Details

3.1 Obtaining a Command from the User

- You may assume the command name is given on one line and does not exceed 50 characters.
- I suggest using `fgets()` to read the command name, but you need not follow this suggestion.
- You may assume the command is just one word. In particular, the user's command will not contain any arguments.
- I suggest that your program replaces the trailing `'\n'`. at the end of the line with an ascii `NULL` (`'\0'`).

3.2 Type 1 Commands Your shell is to Implement

- `hello`: Respond by printing a friendly greeting.
- `bye`: Acknowledge the command and terminate the run.
- `assignment`: Print "202 lab #1 (Spring 2022)".
- `author`: Print your name, your N-number, and your netid.
- `section`: Print "002" or "003" as appropriate.

3.3 Arguments to `execve()`

`execve()` takes three arguments, which you must supply. They are

1. The directory where the executable is located. For this lab the directory is always `/bin` (and hence the executable itself has address the command name given by the user prepended with `/bin/`. This is a simplification. Not all commands are in `/bin`; many are in `/usr/bin/` and some are in other directories.
2. The "name" and arguments for the executable. Specifically, the second argument is `argv` from 201. It is an array of character pointers. The first pointer points to the program name the user entered. The second (and last) pointer is NULL because each user program in this lab has no arguments.
3. The environment pointer. For lab #1, the environment pointer is always `NULL`. This is another simplification.

3.4 The `&status()` Argument to `wait()`

You should declare `status` as an `int` and include `&status` as an argument to `wait()`. However, you need not process the `status` result.