**CSCI 4401/5401**

**Principles of Operating Systems**

**Fall Semester 2023**

**Assignment 2 (9/5)**

**Due Monday, September 25th @ 11:59pm**

**Reading:** Tanenbaum Chapter 2

**Submission Guidelines:**

1. This assignment is worth 100 points for all students.

2. This submission will require you to submit multiple files. Files must be clearly labeled and referenced in your main submission file.

3. Screenshots must use the PrintScreen command so that they are readable.

4. It is your responsibility to make sure all files are readable and submitted on time.

**Submission:**

- Part A requires you to submit a .c script and two output files.

- Part B requires you to submit a script of your choice (e.g., Python, Java, C++, etc.) used for visualization and two graphs.

- Part C requires you to submit a .c script and two graphs.

- Part D is optional extra credit for 4401 students and required for 5401 student. This requires you to submit a single .c script and a screenshot of your running program.

**Part A. System Calls and General Programming (4401: 50 points, 5401: 45 points)**

This part requires you to understand and make system calls including fork(), pid(), and ppid(). You are to write C-language code that does fork() calls in a loop as below:

**for (i= 1 to n) {**

**fork()**

**}**

- The value of n should be input from the user, not a hardcoded variable. The example screenshot below presents a prompt for user input, but you can also do a command line argument.

- The program must also output to a file, not print to the command line, the Process ID and the Parent Process ID for each process. You can format the output how you prefer, but I recommend using csv or json to simplify your task in Part B.

*Hint:* Sometimes a parent process exits before the child makes a system call to find the parent process ID. In these cases, you will get an incorrect parent process ID. You can prevent this by either have your code sleep for a little bit (a second) before exiting or use a waitpid() call at the end of your code.

*Hint:* If your “printf”’s or “fprintf”’s act funny (print duplicate lines), try using fflush to flush out the buffers before forking.

*Hint:* You can prompt a user for input using scanf().

*Hint:* This should only be ~25 lines of code.

**1.** Submit your C code in a separate .c script.

**2.** Run your script with n = 4 as the user input. Submit your output file.

**3.** Run your script with n = 7 as the user input. Submit your output file.

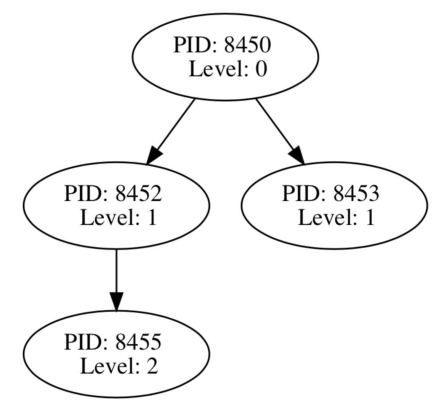
A screenshot of a computer

Description automatically generated

**Part B. Building a Process Tree (4401: 25 points, 5401: 20 points)**

This part requires general programming knowledge and requires you to work with a graphing library (e.g., graphviz). You are to write another program (in a language of your choice) to parse the file that was the output from above and build a Process Tree. Every node should represent a process and must display its Process ID. Every node should also have a “Level” field which indicates how far it is (in terms of nodes) from the root node of the process tree. For example, the grand child of the process tree’s root will have a “Level” of 2.

Here’s an example of a Process Tree that your program might build if the parameter n is 2. I built this using the pydot library in Python (you would need to install this module if you use it). I posted an example on Moodle that you can use to start. (You can also do a little bit of digging and find ways to use graphviz with C, C++, or Java if you prefer). If you choose to use Python, all you need to do is modify the example I provided to read in your output file from Part A and put the data into a dictionary.



*Hint:* If you use the example Python code I provided, all you need to do is replace the “menu” dictionary with a dictionary of process ID’s where the parent will be the key and the child(ren) will be the keys. You will need to create sub-dictionaries like in the example menu. This will require some thought.

**4.** Submit your code to graph that graphs the output from Part A.

**5.** Submit an image of your graph that use the output from n = 4 in Part A.

**6.** Submit an image of your graph that use the output from n = 7 in Part A.

**Part C. System Calls: fork() and waitpid(). (4401: 25 points, 5401: 20 points)**

Modify your code from Part A to make sure that the parent waits for the child to terminate soon after the fork and then breaks from the loop and exits. Here’s some skeletal pseudo code for that loop:

**for (i= 1 to n) {**

**fork()**

**………**

**if parent:**

**waitpid(pid)**

**break**

**}**

**7.** Submit a separate file that contains your modified C code from Part A.

**8.** Run the above modified source code on n=4 and n=7. Then, run the code from Part

B on the output file produced by the modified source code. Include these 2 modified network graphs.

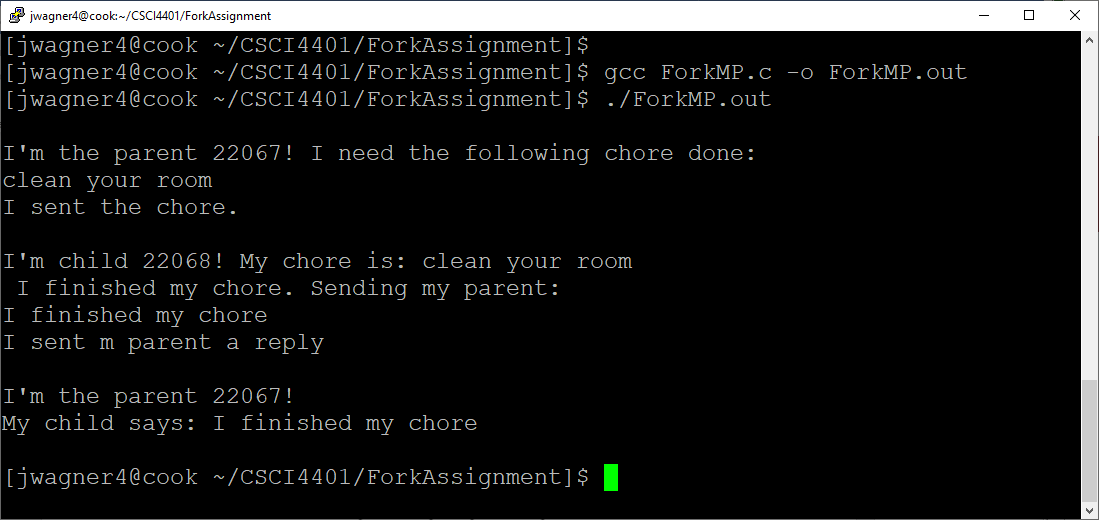
**Part D. Message Passing between Processes.** **(4401: +15 points e.c., 5401: 15 points)**

Write a single C program that will exchange messages between a parent and child process using the msgsnd() and msgrcv() system calls. This will require you to first create a message queue with the msgget() system call. Below is a screenshot that demonstrates what this process should do.

*Step 1.* First, call fork() to start the child process. If it is the parent process, print a simple hello message from the parent including the process ID, and prompt for a chore to be entered into the console. Enter the chore into the console (e.g., “clean your room”) and send that message using msgsnd(). Print a message that confirms this was completed.

*Step 2.* If it is the child process, read the message with msgrcv(). You may want to use blocking (i.e., 0 & IPC\_NOWAIT in the example from class) to make sure the parent has written the message. The child should print a simple hello message including the process ID along with the message it received. The child should then prompt for a reply message to be entered into the console. Enter the reply (e.g., “I finished my chore”) and send this message with msgsnd(). Print a message that confirms this was completed. *Hint:* You may want to use a separate message queue structure for each process.

*Step 3.* If it is the parent process, print another hello including the process ID. Then, read the message with msgrcv() and print it.



**9.** Submit your C code.

**10.** Submit a screenshot of your running code with the messages you entered into the console.