EECS 338 Homework 2: Concurrency using fork()

General requirements:

- Due on the posted due date.
- Upload a single, compressed file (e.g. zip) to Canvas that contains all required files.
- Include either a single makefile that compile all programs or a separate makefile for each.
- Include a typed document with a description of any techniques that you used to control the output order.
- All work should be your own, as explained in the Academic Integrity policy from the syllabus.

Instructions: The purpose of this assignment is to become familiar with using fork() to create concurrent child processes. Each program creates three processes and prints their process IDs and parent process IDs. The order of the output is important, but it is acceptable to have the system prompt appear in the output (see tips below).

1. Create a program that creates one parent with two children. Display the IDs in the order in which the processes are created. Below is an example of how the output might look. Use the “flush(stdout)” and “sleep(#)” instructions to ensure the lines are printed in the correct order. NEW: Alternately, you may use “wait” instead of “sleep” if you wish.

```
Process 1 = 78176, parent = 50076.
Process 2 = 5708, parent = 78176.
Process 3 = 45528, parent = 78176.
```

2. Create a program that creates three processes where the 2nd process is the child of the 1st and the parent of the 3rd. Display the IDs in the order in which the processes are created. Below is an example of how the output might look. Use the “flush(stdout)” and “sleep(#)” instructions to ensure the lines are printed in the correct order. Alternately, you may use “wait” instead of “sleep” if you wish.

```
Process 1 = 45756, parent = 50076.
Process 2 = 71244, parent = 45756.
Process 3 = 18692, parent = 71244.
```

3. Create a program that creates three processes where the 2nd process is the child of the 1st and the parent of the 3rd (just like #2 above). Display the process IDs in **reverse** order of creation (3rd, 2nd, 1st). Below is an example of how the output might look. Use the “flush(stdout)” and “wait(NULL)” instructions to ensure the lines are printed in the correct order. Do not use “sleep”.

```
Process 3 = 18692, parent = 71244.
Process 2 = 71244, parent = 45756.
Process 1 = 45756, parent = 50076.
```

Tips:

- You are not required to check for fork errors (negative return value).
- Use of “fork” and “wait” is demonstrated with fork1.c, available with the lecture notes.
- Use of “getpid” is demonstrated with fork-question-2.c, available with the lecture notes and the textbook website (where it originated from).
- Use “getppid” in child processes to get the parent’s ID.
- If you compile on a local computer that doesn’t recognize “wait” (such as Cygwin), use “#include <sys/wait.h>”.
- It is OK if the prompt is displayed in the middle of your output like this:

```
Process 1 = 18817, parent = 17775.
cxf47@eecslab-3:~/hw2$ Process 2 = 18818, parent = 18817.
Process 3 = 18819, parent = 18818.
```

This may occur when the parent terminates before a child’s output is displayed. If this bothers you, a naive solution is to add an extra “sleep” to force the parent to terminate well after stdout is updated.

Grading rubric:

Item	**Points**
Makefile(s)	10
1: Basics: compiles, prints, etc.	10
1: Creating processes with fork()	10
1: Controlling print order (sleep not required for parent)	10
2: Basics: compiles, prints, etc.	10
2: Creating processes with fork()	10
2: Controlling print order (sleep not required for parent)	10
3: Basics: compiles, prints, etc.	10
3: Creating processes with fork()	10
3: Controlling print order (sleep not allowed)	10
Total	100