EECE.4811/EECE.5811: Operating Systems

Spring 2022

Individual Program 2

Due **5:00 PM**, **Monday, 3/14/22**

1. Introduction

This program will provide you with practice programming with semaphores. The format will be a bit different from our other programs, as you will use problems taken directly from our textbook.

This assignment is worth a total of 40 points. The given grading rubric in Section 4 applies to students in both EECE.4811 and EECE.5811.

2. Project Submission and Deliverables

Your submission must meet the following requirements:

- Your solution may be written in C or C++.
  - However, given that all starter files are written in C, you will likely find it easiest to complete these programs in that language.
- Your code must run on the Linux machines in Ball 410, but you may write it elsewhere.
- You must submit an archive file *(.zip, .tar, or .tgz format only)* containing:
  - All source files you complete. You should have three such files:
    - `fork-join.c`
    - `rendezvous.c`
    - `barrier.c`
  - Appropriate build files for your project—either a makefile to be used with `make`, or appropriate files to be used with `cmake`
    - Note that you will have three different executables—one for each problem—and your build system should account for this requirement.
  - A README file that briefly describes the contents of your submission, as well as directions for compiling and running your code.
- Programs must be submitted via Blackboard.
- You must work individually on this program—this is not a group assignment.

3. Specification

General overview: Chapter 31 of the OSTEP text, which covers semaphores, contains several homework problems on the last page (p. 20). You will solve the first three problems, which are briefly described below (and described in more detail in the textbook).

1. The fork/join problem (5 points): A very basic problem—a parent thread starts a child thread and waits for it to finish—for which the solution is basically given in the text. (I'll leave it up to you to find that solution.) A correctly completed program should print the following three lines of output in order:

   ```
   parent: begin
   child
   parent: end
   ```

2. The rendezvous problem (10 points): A small extension to the fork/join problem. The note about using `sleep(1)` calls to test your solution means you should add delay in various places to ensure the code works because of your semaphore usage, not just because the threads happen to execute in the proper order.

3. Barrier synchronization (15 points): Implementation of a higher-level synchronization primitive used to ensure an arbitrary number of threads all reach the same point in their code before any thread is allowed to proceed.

The Blackboard item describing this assignment provides links to the appropriate textbook chapter, as well as the GitHub repository where you can find starter code for each problem.

4. Grading Rubric

Due to the simplicity of this assignment, the grading rubric is relatively simple as well:

- **10 points**: Your program compiles successfully using the build system you provided, according to the directions in your README file.
- **5 points**: You provide a working solution for the fork/join problem in `fork-join.c`.
- **10 points**: You provide a working solution for the rendezvous problem in `rendezvous.c`.
- **15 points**: You provide a working barrier and a complete version of the program started in `barrier.c`.