EECE.4810/EECE.5730: Operating Systems

Spring 2019

Exam 1
February 25, 2019

Name: _____

Section: **EECE.4810 (undergraduate)** **EECE.5730 (graduate)**

For this exam, you may use two 8.5" x 11" double-sided pages of notes. All electronic devices (e.g., cell phones, calculators, laptops) are prohibited. If you have a cell phone, please turn it off before the start of the exam to avoid distracting other students.

The exam contains 4 sections for a total of 100 points. Please answer the questions in the spaces provided. If you need additional space, use the back of the page on which the question is written and clearly indicate that you have done so.

Please note that students enrolled in EECE.5730 must complete two extra problems, which are worth a total of 15 points:

- Question 1d, on page 3
- Question 2c, on page 5

You will have two hours to complete this exam.

Q1: Process management	/ 32 + 8
Q2: Inter-process communication (IPC)	/ 22 + 7
Q3: Multithreading	/ 26
Q4: Synchronization	/ 20
TOTAL SCORE	/ 100 + 15

1. (28 + **8** points) ***Process management***

a. (8 points) Explain (i) what a zombie process is, and (ii) why all processes are technically zombie processes for at least a brief time. (Hint: think about which of the five process states zombie processes must exist in.)

Questions 1b, 1c, and 1d refer to the two programs ***pr1*** and ***pr2*** listed on the extra sheet provided with the exam. Assume ***pr1*** always executes first and is invoked as follows:

```
./pr1 4810
```

b. (12 points) How many unique processes do the programs ***pr1*** and ***pr2*** create when executed, including the initial process? Draw a process tree to support your answer.

1 (continued)

Remember, Questions 1b, 1c, and 1d refer to the two programs ***pr1*** and ***pr2*** listed on the extra sheet provided with the exam. Assume ***pr1*** always executes first and is invoked as follows:

```
./pr1 4810
```

c. (12 points) What will these programs print? (If there are multiple possible output sequences, choose one valid sequence and display it.)

d. (8 points, ***EECE.5730 only***) Are there multiple possible output sequences for this program? (In other words, could the same output statements print in a different order each time you run the program?) If so, explain why; if not, explain why your answer to part (c) is the only possible output.

2. (22 points) ***Inter-process communication (IPC)***

a. (10 points) What benefits does message passing IPC offer over shared memory IPC? Specifically, what does the kernel handle that might make it easier for the programmer to write cooperating processes than it would be using shared memory?

b. (12 points) Explain why, in the POSIX shared memory example we discussed in class, (i) the shared region was first established as a memory-mapped file, and (ii) why it is beneficial to map the region into memory, as opposed to simply working with a shared file.

2 (continued)

c. (7 points, ***EECE.5730 only***) Can you write a set of cooperating processes that model message passing IPC, communicating without invoking the kernel for the actual send/receive operations? If so, explain how, including the differences between modeling indirect and direct communication. If not, explain why not.

3. (26 points) ***Multithreading***

a. (8 points) Is it possible for two concurrent threads to execute without running in parallel? Explain why or why not.

b. (8 points) Name the four sections of a process's address space and explain why they can or cannot be shared by multiple threads in the same process.

3 (continued)

c. (10 points) Given the two threads below, is it possible for v1 to be equal to -2 and v2 to be equal to 10 when both threads are done? If so, explain how; if not, explain why not.

Assume v1 and v2 are shared variables that are both initialized to 0 before either thread starts. Do not assume each line of code is executed atomically—each instruction executes atomically, but a statement may represent multiple instructions.

Thread 1	Thread 2
v1 = v1 + 2	v1 = v1 - 2
v2 = v2 + 10	v2 = v2 - 10

4. (20 points) ***Synchronization***

a. (10 points) Is it possible to establish a critical section without synchronization primitives such as locks? Explain your answer, including a description of the three properties a critical section must satisfy and how that can or cannot be done without synchronization primitives.

4 (continued)

b. (10 points) In an in-class example, we briefly discussed the idea of thread joining—in other words, a parent thread waiting for a child thread to finish. This functionality could be implemented using a lock, a condition variable, and a shared variable. The parent thread would set the variable to 0 before starting the child thread; the child would set the variable to 1 when it's finished. The synchronization primitives would protect access to the variable and prevent the parent from wasting cycles on busy waiting until the child finishes.

   Complete the pseudocode below to show where lock/unlock and condition variable wait/signal operations would be used to force the parent to wait for its child to finish.

```
/* Global variables */
int done;           // Shared variable
lock L;             // Lock
cond_var CV;        // Condition variable

/* Child thread—complete with lock/CV operations */
void *child(void *arg) {
   ...              // Assume thread does actual work here
   /* COMPLETE SECTION BELOW */



   done = 1;        // Indicate thread complete



}

/* Parent thread—complete with lock/CV operations */
int main() {
   ...
   done = 0;        // Clear shared variable
   create(child);   // Create and start child thread
   /* COMPLETE SECTION BELOW */



   while (_______________________________) {  // Wait for shared var
                                              //   to change

   }



}
```