# EECE.4810/EECE.5730: Operating Systems

Spring 2018

Exam 1
February 21, 2018

**Name:** _____

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

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

The exam contains 4 questions 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 4d, on page 9

You will have one hour and fifteen minutes to complete this exam.

| | |
|---|---|
| Q1: Process management | / 28 + 8 |
| Q2: Inter-process communication (IPC) | / 20 |
| Q3: Multithreading | / 26 |
| Q4: Synchronization | / 26 + 7 |
| **TOTAL SCORE** | / 100 + 15 |

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

a. (8 points) Explain (i) what an orphan process is, and (ii) why UNIX systems reassign processes to a new "parent" rather than simply terminating a process as soon as it's orphaned.

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.

b. (10 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.

c. (10 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. (20 points) ***Inter-process communication (IPC)***

a. (10 points) Name the two operations that message passing IPC systems must support, and explain how these operations function differently in direct communication than they do in indirect communication.

b. (10 points) Describe the steps necessary for two processes to establish and share a region of memory in a shared memory IPC system.

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

a. (8 points) What data or information can threads in the same process share that separate, independent processes do not share?

b. (8 points) You are designing a program to run on a system with hardware multithreading support. What characteristics determine whether part (or all) of the program should be written to run using multiple threads?

3 (continued)

c. (10 points) Given the two threads below, is it possible for x to be equal to 11 and y to be equal to 15 when both threads are done? If so, explain how; if not, explain why not.

Assume x and y are shared variables, and each line of code (but not necessarily each whole thread) is executed atomically.

| Thread 1 | Thread 2 |
|---|---|
| x = 10 | y = 15 |
| x = y - 3 | y = x + 4 |

4. (26 points) ***Synchronization***

a. (8 points) Is a critical section required to always run to completion without any interruption from other threads or processes? Explain why or why not.

b. (8 points) Programs that combine condition variables with locks for synchronization typically waste fewer processor cycles than programs that only use locks. Explain why.

4 (continued)

c. (10 points) Processes that share memory while executing concurrently are cooperating threads requiring protected access to shared data. The pseudocode below outlines the behavior of concurrent producer/consumer processes. We ran similar programs when discussing IPC, but the producer ran first; then, the consumer ran separately.

In this example, both processes concurrently access a shared array used as a circular queue, `buffer[BUFFER_SIZE]`, and two integers, `in` and `out`. `in` is the position to store a newly produced item, while `out` is the position from which the next item will be consumed. The buffer is empty if the two positions are equal, and full if `in` is one spot behind `out`.

Determine which line(s) in each program represent a critical section, and show where you would place lock()/unlock() operations to ensure the atomicity of each critical section.

| ***Producer*** | ***Consumer*** |
|---|---|
| `item next_prod;`<br>`while (true) {`<br>`  // store new data in next_prod`<br><br>`  while (((in + 1) % BUFFER_SIZE)`<br>`  == out)`<br>`  ; /* do nothing */`<br><br>`  buffer[in] = next_prod;`<br>`  in = (in + 1) % BUFFER_SIZE;`<br>`}` | `item next_cons;`<br>`while (true) {`<br>`  while (in == out)`<br>`  ; /* do nothing */`<br><br>`  next_cons = buffer[out];`<br>`  out = (out + 1) % BUFFER_SIZE;`<br><br>`  // do something with next_cons`<br>`}` |

4 (continued)

d. (7 points, ***EECE.5730 only***) We covered the concept of hand-over-hand locking in thread-safe queues that maintain one lock per node. Explain this concept and describe what potential problems it avoids.