EECE.4810/EECE.5730: Operating Systems

Spring 2019

Exam 2
April 1, 2019

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 3 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 1c, on page 3
- Question 3e, on page 11

You will have two hours to complete this exam.

S1: Monitors & semaphores	/ 20 + 7
S2: Deadlock	/ 30
S3: Scheduling	/ 50 + 8
TOTAL SCORE	/ 100 + 15

1. (20 **+ 7** points) ***Monitors and semaphores***

a. (10 points) An inefficient solution to the multiple producer/multiple consumer problem you solved in Program 2 might look like the pseudocode below, which uses a single lock, L, and a single condition variable, CV:

Producer	Consumer
lock(L)	lock L
while (buffer full)	while (buffer empty)
wait(L, CV)	wait(L, CV)
Write data to buffer	Read data from buffer
signal(CV)	signal(CV)
unlock(L)	unlock(L)

Explain what makes this solution inefficient and how you can solve this issue.

1 (continued)

b. (10 points) In Program 2, you implemented a monitor using locks and condition variables. Could you implement a monitor for the same problem (bounded buffer, multiple producers & consumers) using only semaphores for synchronization? If not, explain why not; if so, explain for what purposes you would need semaphores, how you would initialize those semaphores, and under what conditions you would call up() or down() on each of them.

c. (7 points, ***EECE.5730 only***) A barrier is a synchronization construct that forces a group of threads to wait for all threads to reach the barrier before allowing any of them to proceed. Use the space below to describe how a barrier can be implemented using semaphores for a given number of threads, N. (Hint: there are two general solutions—a relatively inefficient one using N semaphores, and a more efficient one using two semaphores and an integer.)

2. (30 points) ***Deadlock***

a. (20 points) Your system is currently running 3 processes: A, B, and C. Each process requests two types of resources: files and network connections. Assume the system can handle up to 5 open files and 5 network connections (abbreviated "conn.") in total, across all processes.

Each process makes the resource requests shown in the table below, which lists total requested resources, initial state (resources the process is already using at this point), and a list of individual requests. Each individual request consists of a unique ID (A1 or B2, for example) and the resources requested (files, connections, or both).

Each process only frees resources when it ends, which occurs after all its requests are complete.

Use the Banker's Algorithm to find an order in which all resource requests can be satisfied and list that order in the space provided. You must show the total amount of each resource type used at each step of your solution. Requests from the same process must be in order (for example, A1 must be satisfied before A2), but requests from different processes may be in any order (A1 may be satisfied before or after B1).

	Process A	**Process B**	**Process C**
Total requested resources	5 files 3 connections	5 files 5 connections	5 files 3 connections
Initial state (already using)	1 file	2 connections	1 file
Individual resource requests	A1: 2 files, 1 conn. A2: 2 conn. A3: 1 file	B1: 4 files B2: 1 file, 1 conn. B3: 2 conn.	C1: 3 conn. C2: 2 files C3: 2 files

2 (continued)
Additional space to solve Question 2a:

2 (continued)

b. (10 points) You are running a program with three concurrent threads, each of which start execution by operating on two semaphores, S1 and S2, as shown. Assume both semaphores are initialized to 0:

Thread 1	Thread 2	Thread 3
`down(&S1);`	`down(&S1);`	`down(&S2);`
`up(&S2);`	`up(&S2);`	`down(&S2);`
…	…	`up(&S1);`
		`up(&S1);`
		…

Explain why these three threads are guaranteed to deadlock, and show how you could rewrite the threads to remove the deadlock condition.

3. (50 + 7 points) ***Scheduling***

a. (20 points) Consider the following set of processes, with the length of the CPU burst time given in milliseconds. Processes may begin executing 1 ms after they arrive (i.e., a process arriving at time 5 could start executing at time 6). Any process arriving at time 0 is in the ready queue when the scheduler makes a decision about the first process to run.

Process	Burst	Priority	Arrival time
P1	2	5	0
P2	7	4	0
P3	3	3	5
P4	8	2	7
P5	4	1	9

Determine the turnaround time for each process using each of the following scheduling algorithms: (i) round-robin (RR) with time quantum = 1 ms, (ii) shortest job first (SJF), and (iii) a *preemptive* priority scheme in which lower numbers indicate higher priority.

Your solution **must contain some work that shows start/end times for each algorithm**—either a table or Gantt chart like the ones shown in class.

You do not have to calculate the average turnaround time for each algorithm.

Use the space on this page and the following page to solve this problem.

3 (continued)
Additional space to solve Question 3a:

3 (continued)

b. (10 points) When are scheduling decisions made in non-preemptive scheduling algorithms? When are scheduling decisions made in preemptive scheduling algorithms.

c. (10 points) Which of the scheduling algorithms we discussed can cause starvation and why? How can you prevent starvation in these algorithms?

3 (continued)

d. (10 points) Which scheduling algorithm would you choose for the set of processes described in each table? Briefly explain your answer in each case.

i.

Process	Burst	Arrival time
P1	10	0
P2	2	0
P3	15	1
P4	3	2
P5	8	3

ii.

Process	Burst	Arrival time
P1	10	0
P2	10	0
P3	10	5
P4	9	7
P5	9	9

3 (continued)

e. (8 points, ***EECE.5730 only***) In class, we described how some scheduling algorithms "degrade to first-come, first-served (FCFS)" under certain conditions. Explain what it means for an algorithm to degrade to FCFS, and give an example demonstrating this concept.