Operating Systems — Homework 2
CS 4410 — Due Monday, July 24

1. Suppose that a system was running 3 processes. The first two processes do CPU bursts followed by I/O followed by another CPU burst. The third process does no I/O. The following table gives the arrival time (when the process is forked), and the durations of the program operations:

Process	**Arrival time**	**CPU**	**I/O**	**CPU**
P_1	0	15	5	10
P_2	5	35	10	20
P_3	10	70		

   (a) Assuming that context switches cost nothing, compute the average waiting time and turnaround time for this sequence of processes when scheduled using
      - FCFS
      - SJF
      - Round-robin with a quantum of 15
      - Round-robin with a quantum of 40

   (b) How long must the quantum be for round-robin to give the same schedule as FCFS?

2. Consider the following snapshot of a system:

	Allocation				Max			
	A	B	C	D	A	B	C	D
Available	1	5	2	0	3	14	12	12
P_0	0	0	1	2	0	3	1	2
P_1	1	0	0	0	1	7	5	0
P_2	1	3	5	4	2	3	5	6
P_3	0	6	3	2	0	6	5	2
P_4	0	0	1	4	0	6	5	6

   (a) Is the system in a safe state? Prove your answer.

   (b) Suppose process P_3 releases 3 resources of type B. Give an example request that would be granted immediately by the Banker's algorithm.

   (c) Continuing the supposition that P_3 releases three resources of type B, give an example request that would cause the requesting process to block when using the Banker's algorithm.

3. The program in `incr.py` runs two threads: one repeatedly increments the elements of a matrix, while the other repeatedly decrements them. Since both operations run the same number of times, the final values in the matrix should be the same as the initial values.

   (a) Due to lost updates, the final results may be incorrect. In your written document, give the range of resulting values that you see. Based on these values, what can you conclude (if anything) about the number of lost updates?

   (b) Use semaphores to correctly synchronize the program. You should use fine- grained locking, so that threads that are accessing different parts of the matrix can proceed concurrently.

4. The goal of the program in `zigzag.py` is to print out 78 characters in a specific order. This task is split into 4 threads, each of which prints out one type of character. Because the threads are not synchronized, the program can print its output in any order.

Add semaphore operations so that the output is always correct. You should retain the basic program structure — your solution should contain the same set of threads, and each thread should perform the same printing commands.

5. The program `bridge.py` simulates a one-lane bridge. Many cars may pass over the bridge simultaneously, but they must all be going in the same direction. Complete the implementation of the `OneLaneBridge` monitor.

6. The program `matchmaker.py` simulates a game server. Players connect to the service, which blocks them until a group of four players is available. It then releases the group of four players. Complete the implementation of the `Matchmaker` monitor.

   For full credit, your implementation should be fair: as soon as it becomes possible to start a game, only threads that have already called join should be allowed to play. For example, the following should be impossible: A, B, C, and D call join, then A starts, then B starts, then E calls join, then E starts, and then C starts; since E had not yet called join when the game was started, E must wait for the next game (a correct but non-fair implementation will receive most of the credit).

7. Modify your threaded implementation of `parcount.py` from HW1 so that all of the worker threads modify a single shared variable. Use synchronization mechanisms to ensure that the shared variable is updated safely. Compare the running time if your locking implementation to the running time of your previous implementation, and explain your findings.