OS Assignment III

1.

A. **How does the signal() operation associated with monitors differ from the corresponding operation defined for Semaphores?**

The signal( ) operations associated with monitors is not persistent.
If a signal is performed and if there are no waiting threads, then the signal is simply ignored and the system does not remember the fact that the signal took place. If a subsequent wait operation is performed, then the corresponding thread simply blocks.

Whereas in semaphores, every signal results in a corresponding increment of the semaphore value even if there are no waiting threads. A future wait operation would immediately succeed because of the earlier increment.

B. **What is the meaning of busy waiting? What are the other kinds of waiting in operating system? Can busy waiting be avoided altogether? Explain.**

Busy waiting means that a process is waiting for a condition to be satisfied in a tight loop without relinquish the processor.

Alternatively, a process could wait by relinquishing the processor, and block on a condition and wait to be awakened at some appropriate time in the future.

Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached.

2.

A. **Show that if wait() and signal() semaphore operations are not executed atomically then mutual exclusion may be violated.**

A wait operation atomically decrements the value associated with a semaphore. If two wait operations are executed on a semaphore when its value is 1, if the two operations are not performed atomically, then it is possible that both operations might proceed to decrement the semaphore value, thereby violating mutual exclusion.

B. **Explain why spinlocks are not appropriate for single processor systems yet are often used in multiprocessor environments.**

Spinlocks are not appropriate for single-processor systems because the condition that would break a process out of the spinlock can be obtained only by executing a different process. If the process is not relinquishing the processor, other processes do not get the opportunity to set the program condition required for the first process to make progress. In a multiprocessor system, other processes execute on other processors and thereby modify the program state in order to release the first process from the spinlock.

3. **Implement a Bounded producer consumer solution with Monitors. Assume the condition variables as : full, empty, initialization code for producer and consumer and procedures - Enter and Remove in the Monitor implementation.**

```
monitor ProducerConsumer {

      condition full, empty;
      int count = 0;

      procedure enter () { // produce
            if (count == N)
                  wait(full);
            produceItem(item);
            count++;
            if (N == 1)
                  signal(empty);
      }

      procedure remove () { // consume
            if (count == 0)
                  wait(empty);
            consumeItem(item);
            count--;
            if (count == N-1)
                  signal(full);
      }

}

function producer () {
      while (YES) {
            ProducerConsumer.enter();
      }
}

function consumer () {
      while (YES) {
            ProducerConsumer.remove();
      }
}
```

4. Consider the following snapshot of the following:

	ALLOCATION				MAX				AVAILABLE			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

Answer the following questions using the banker's algorithm.

i. What is the content of the matrix Need.

NEED = MAX - ALLOCATION

	NEED			
	A	**B**	**C**	**D**
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

ii. Is the system in a safe state?

Yes. With Available being equal to (1, 5, 2, 0), either process P0 or P3 could run. Once process P3 runs, it releases its resources, which allow all other existing processes to run.

One example of such order would be P0, P2, P3, P4, P1.

iii. If a request from process P1 arrives for (0, 4, 2, 0) can the request be granted immediately?

The request can be granted immediately.

This results in the value of Available being (1, 1, 0, 0). One ordering of processes that can finish is P0, P2, P3, P1, and P4.

5. **What are the Conditions for Deadlock to occur? Briefly explain. In a system, the following state of processes and resources are given: R1→ P1, P1→ R2, P2→ R3, R2→ P2, R3→ P3, P3→ R4, P4→ R3, R4→ P4, P4→ R1, R1→ P5. Draw Resource Allocation Graph for the system and check for deadlock condition. Explain your answer.**

Deadlock can happen when one or more of the following conditions are satisfied:

- More than one process have a lock at a time. (Mutual exclusion)
- Resources cannot be pre-empted. (No pre-emption)
- Process waiting for resources that are assigned to some other process. (Hold and wait)
- Cyclic waiting of the processes.

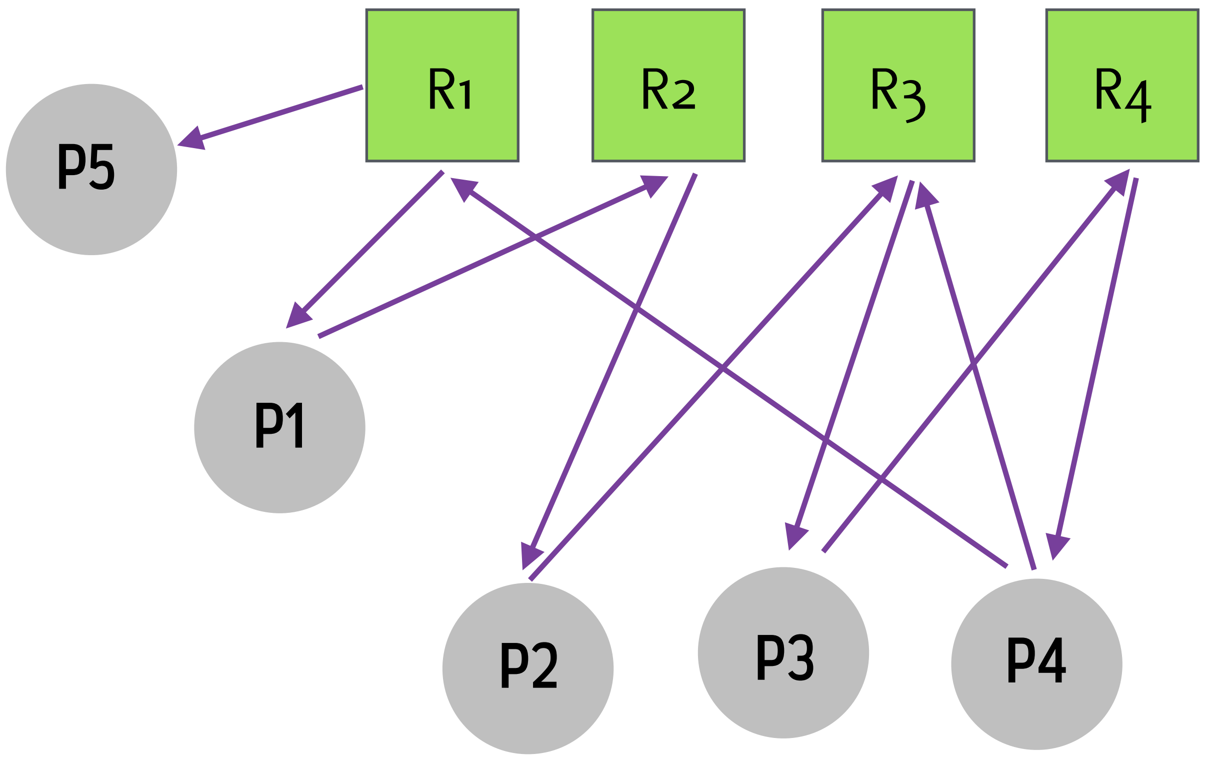
Cycles in the resource allocation graph include:

R1→ P1→ R2→ P2→ R3→ P3→ R4→ P4→ R3→ P3→ R4→ P4→ R1

This cycle has more than one instances of the resource R3 and R4, so it's not necessarily deadlocked.

R3→ P3→ R4→ P4→ R3

This cycle has only one instances of the resources R3 and R4, so it might be deadlocked.


R1
R2
R3
R4
P5
P1
P2
P3
P4