Operating Systems: Exercises on deadlocks

1. Consider the following snapshot of a system:

	Allocation	*Max*
	A B C D	*A B C D*
T_0	1 2 0 2	4 3 1 6
T_1	0 1 1 2	2 4 2 4
T_2	1 2 4 0	3 6 5 1
T_3	1 2 0 1	2 6 2 3
T_4	1 0 0 1	3 1 1 2

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.

a. ***Available*** = (2, 2, 2, 3)

b. ***Available*** = (4, 4, 1, 1)

c. ***Available*** = (3, 0, 1, 4)

d. ***Available*** = (1, 5, 2, 2)

2. Consider the following snapshot of a system:

	Allocation	*Max*	*Available*
	A B C D	*A B C D*	*A B C D*
T_0	3 1 4 1	6 4 7 3	2 2 2 4
T_1	2 1 0 2	4 2 3 2	
T_2	2 4 1 3	2 5 3 3	
T_3	4 1 1 0	6 3 3 2	
T_4	2 2 2 1	5 6 7 5	

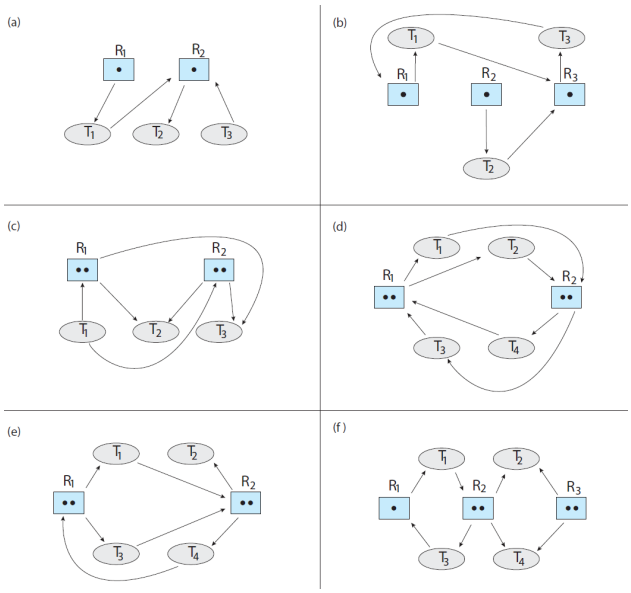
(a) Illustrate that the system is in a safe state by demonstrating an order in which the threads may complete.

(b) If a request from thread T_4 arrives for (2, 2, 2, 4), can the request be granted immediately?

(c) If a request from thread T_2 arrives for (0, 1, 1, 0), can the request be granted immediately?

(d) If a request from thread T_3 arrives for (2, 2, 1, 2), can the request be granted immediately?

3. Which of the six resource-allocation graphs shown below illustrate deadlock? For those situations that are deadlocked, provide the cycle of threads and resources. Where there is not a deadlock situation, illustrate the order in which the threads may complete execution



4. The program example shown below doesn't always lead to deadlock. Describe what role the CPU scheduler plays and how it can contribute to deadlock in this program.

```
/* thread_one runs in this function */
void *do_work_one(void *param)
{
   pthread_mutex_lock(&first_mutex);
   pthread_mutex_lock(&second_mutex);
   /**
    * Do some work
    */
   pthread_mutex_unlock(&second_mutex);
   pthread_mutex_unlock(&first_mutex);

   pthread_exit(0);
}

/* thread_two runs in this function */
void *do_work_two(void *param)
{
   pthread_mutex_lock(&second_mutex);
   pthread_mutex_lock(&first_mutex);
   /**
    * Do some work
    */
   pthread_mutex_unlock(&first_mutex);
   pthread_mutex_unlock(&second_mutex);

   pthread_exit(0);
}
```