7.1 Consider the traffic deadlock depicted in Figure 7.10.

a. Show that the four necessary conditions for deadlock hold in this example.

b. State a simple rule for avoiding deadlocks in this system.

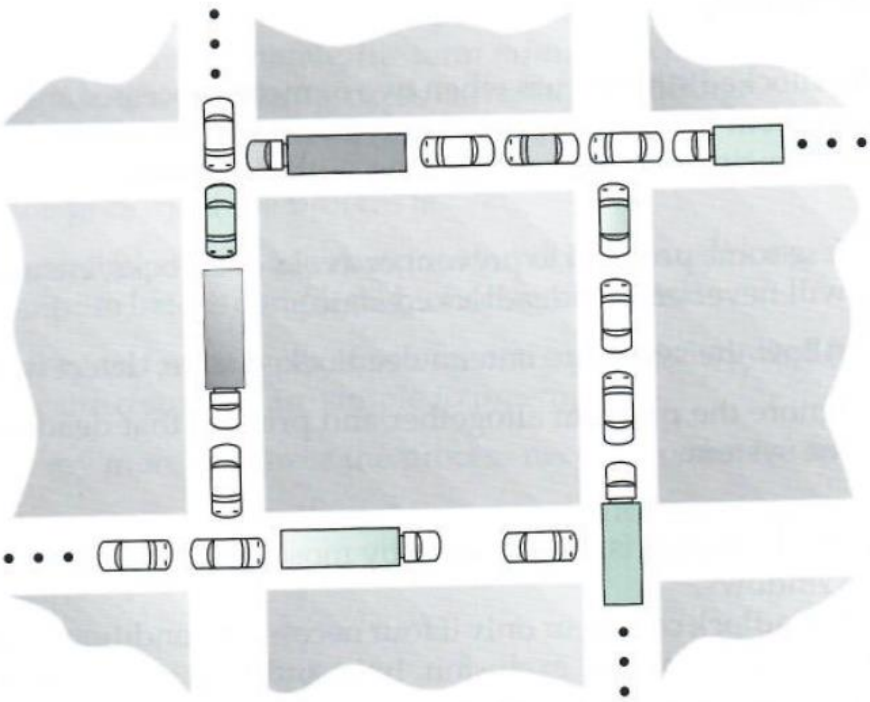

Figure 7.10 Traffic deadlock for Exercise 7.1

```
/* 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);
}
```

Figure 7.4 Deadlock example.

7.3 The program example shown in Figure 7.4 doesn't always lead to deadlock. Describe what role the CPU scheduler plays and how it can contribute to deadlock in this program.

7.13 Consider the following snapshot of a system:

	Allocation	*Max*	*Available*
	$A\ B\ C\ D$	$A\ B\ C\ D$	$A\ B\ C\ D$
P_0	2 0 0 1	4 2 1 2	3 3 2 1
P_1	3 1 2 1	5 2 5 2	
P_2	2 1 0 3	2 3 1 6	
P_3	1 3 1 2	1 4 2 4	
P_4	1 4 3 2	3 6 6 5	

Answer the following questions using the banker's algorithm:

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

b. If a request from process P_1 arrives for $(1, 1, 0, 0)$, can the request be granted immediately?

c. If a request from process P_4 arrives for $(0, 0, 2, 0)$, can the request be granted immediately?