Operating System Homework 7 & 8

Jinyan Xu, 3160101126, Information Security

8.12 Consider the traffic deadlock depicted in Figure 8.11.

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

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

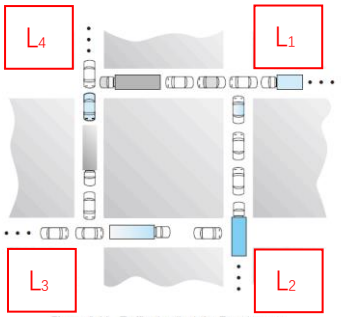

Figure 8.11 Traffic deadlock for Exercise 8.12.

Answer:

a. Regard roadways as resources and cars as processes.

Mutual Exclusion: Only one line of cars at a time can use the roadway.

Hold-and-wait: Each line of cars holds one roadway and wait for the next roadway.

No preemption: The roadway can't be released until the whole line of cars have passed it.

Circular wait: There're 4 lines of cars, L_1 waits for L_2, L_2 waits for L_3, L_3 waits for L_4, L_4 waits for L_1.

b. Each car detects the intersection before entering, if it can't pass, it will wait until the intersection is free.

8.18 Which of the six resource-allocation graphs shown in Figure 8.12 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.

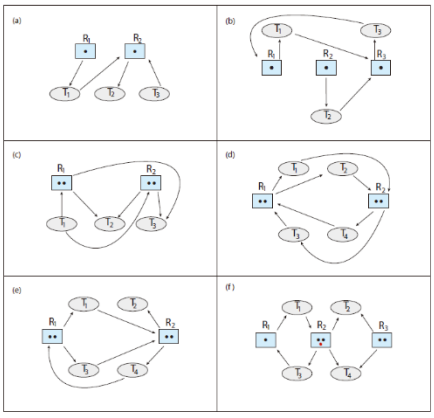

Figure 8.12 Resource-allocation graphs for Exercise 8.18.

Answer:

Note: I added a resource to R_2 in graph (f).

Graph (a): No deadlock, possible order of thread execution: **T_2, T_1, T_3**.

Graph (b): Deadlock, because there is a cycle: **$T_1 \rightarrow R_3 \rightarrow T_3 \rightarrow R_1 \rightarrow T_1$**.

Graph (c): No deadlock, possible order of thread execution: **T_2, T_3, T_1**.
Graph (d): Deadlock, there're two cycle: **T_1→R_2→T_3→R_1→T_1**, **T_2→R_2→T_4→R_1→T_2**.
Graph (e): Possible deadlock, there're two cycles: **T_1→R_2→T_4→R_1→T_1**, **T_3→R_2→T_4→R_1→T_3**, if T_2 releases R_2, the deadlock can be avoided.
Graph (f): Possible deadlock, the cycle is: **T_1→R_2→T_3→R_1**, if T_2 or T_4 release R_2, the deadlock can be avoided.

8.22 Consider a system consisting of four resources of the same type that are shared by three threads, each of which needs at most two resources. Show that the system is deadlock free.

Answer:
In any case, there is always a thread can have 2 resources, which means this thread will release its resources in bounded time, so the other threads which wait for the resource can always get what they want. In conclusion, this system is deadlock free.

8.23 Consider a system consisting of m resources of the same type being shared by n threads. A thread can request or release only one resource at a time. Show that the system is deadlock free if the following two conditions hold:
a. The maximum need of each thread is between one resource and m resources.
b. The sum of all maximum needs is less than m + n.

Answer:
From a & b, we can get this: $1 \le \sum_{i=1}^{n} Max_i \le m + n$.
Consider deadlock happens: $\sum_{i=1}^{n} Allocation_i = m$.
In this case, $1 \le \sum_{i=1}^{n} Need_i + \sum_{i=1}^{n} Allocation_i = \sum_{i=1}^{n} Need_i + m = \sum_{i=1}^{n} Max_i \le m + n$.
So, $1 - \mathrm{m} \le \sum_{i=1}^{n} Need_i \le n$, there must exist some threads that their needs are 0. For these threads, when they finish, it can release at least 1 resource. So, the other threads waiting for the resource can always get the resource they need. In conclusion, this system is deadlock free.

8.28 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	

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 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?

Answer:

a. The content of the matrix Need:

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

Both T_2 & T_3 can run. Let T_2 run first.

T_2 gets the resource.

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

Then, the remains threads can run in any order, one possible order is: **T_2, T_0, T_1, T_3, T_4**. So, the system is in safe state.

b. A request from thread T_4 arrives for (2, 2, 2, 4).

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

Then the available matrix is equal to 0, neither thread can run. So, the system is in unsafe state, which means this request can't be granted immediately.

c. A request from thread T_2 arrives for (0, 1, 1, 0).

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

Threads can run in this order: T_2, T_3, T_0, T_1, T_4. So, the system is in safe state, which means this request can be granted immediately.

d. A request from thread T_3 arrives for (2, 2, 1, 2).

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

Threads can run in this order: T_3, T_0, T_1, T_2, T_4. So, the system is in safe state, which means this request can be granted immediately.