OS HW7&8

8.12

a) First of all, in this figure, we can regard one line of cars as a process, and an intersection as a resource. So there are four processes with four resource.
   Then, we are going to show the four conditions of deadlock:
   1) Mutual exclusion. One resource can only be occupied by one process. In this traffic deadlock, one-line cars are occupying one intersection, in other word, one resource is occupied by one process. And other cars which want this intersection need to wait.
   2) Hold and wait or resource holding. A process(one-line cars) is currently holding one resource(intersection) and requesting additional resources which are being held by other processes. In this traffic deadlock, one-line cars are holding one intersection and requesting another intersection which is being held by other one-line car.
   3) No preemption. A resource can be released only voluntarily by the process which is holding it. In this traffic deadlock, we can find it impossible for one-line cars to give up its intersection because these roads are one-direction-travel. So they can't be released voluntarily.
   4) Circular wait. Some processes may form a circle. P0 is waiting for the resource which is held by p1, p1 is waiting for the resource which is held by p2 … so in this traffic deadlock, we can find a circle by these four processes(one-line cars), and each process want to retrieve the resource(intersection) which is holding by another process.

b) If the last car is at the end of this road, next want to pass through this intersection can't enter this intersection. In this figure, in other words, the center square can be surrounded exclude the four vertices. Only when the next position is free, next car can enter the intersection.

8.18

(b) and (d) illustrate deadlock.

In (b) and (d):

(b)'s cycle of threads and resources:

T1->R3->T3->R1

(d)'s cycle of threads and resources:

R1 and R2 both have two resources. And they are allocated to

T1 and T2, T3 and T4 respectively.

T1->R2->T4->R1

T2->R2->T3->R1

In (a), (c), (e) and (f), in this order they may complete execution:

(a): T2->T1->T3

(c): T2->T3->T1

(e): T2->T1->T3->T4

8.22

	Allocation	Max	need	Available
T0	0	2	2	4
T1	0	2	2	
T2	0	2	2	

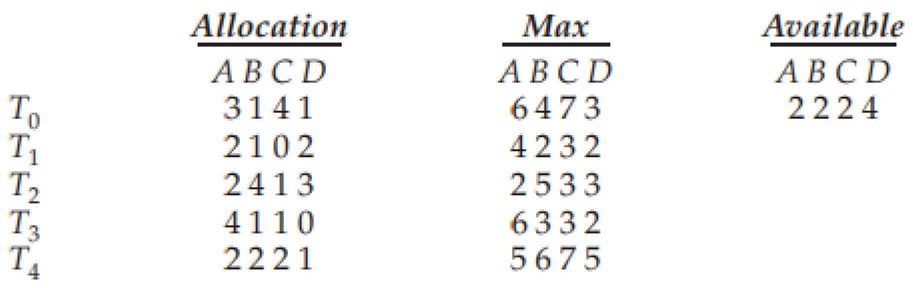
According to this form, we want to show this system is deadlock free, so we need try to find a deadlock situation. We can firstly provide these three threads with 1 resource to avoid them finish. And we still have 1 resource, we need to allocate it to one of these three thread, and each thread only need 1 resource to execute itself. So it cannot exist a deadlock. Consider the four conditions that deadlock needs. There won't be a cycle in the system. So this system is deadlock free.

8.23

If these two conditions exist, firstly, according to the a condition we can easily find there are at least n needs when each thread need one resource. And according to the b condition, we can notice that if we want a deadlock, we need to allocate all the m resources but every thread cannot be executed. So suppose that we allocate all the m resources to the n threads. Because we have less than m + n maximum needs, The number of current needs is less than m + n – m = n. In other words, at most there are n-1 needs. However, we hope that these n threads are still holding and waiting, in other words, each thread need 1 more resource. But there are at most n-1 needs, at least one thread don't need another resource.

Judging from what I mentioned above, if these two conditions hold, this system is deadlock free.

8.28



	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	

First of all, we can find the needs of each thread:

	Need			
	A	B	C	D
T0	3	3	3	2
T1	2	1	3	0
T2	0	1	2	0
T3	2	2	2	2
T4	3	4	5	4

a. According this figure, we can find there are 2A, 2B, 2C, 4D left, it is suitable the T2 and T3, we can firstly choose the T2, then T2 execute and there are 2A, 3B, 4C, 4D available. And we can choose T1, then T0, T3, T4.
   So I choose the sequence that may complete the threads:
   <T2, T1, T0, T3, T4>

b. If a request from T4 arrives for (2,2,2,4), because T4 may need 3, 4, 5, 4, so it can be granted immediately.

c. However, when T2 request (0, 1, 1, 0), because T2 only need at most 0A, 1B, 2C, 0D additionally, so it can be granted immediately.

d. If a request from thread T3 arrives for (2, 2, 1, 2), because T3 only need at most 2A, 2B, 2C, 2D additionally, so it can be granted immediately.