Worked out Examples

6.1 Consider a system consisting of 4 resources of same type that are share by 3 processes each of which needs at most two resources. Show that the system is deadlock free

Ans:

If the system is deadlocked, it implies that each process is holding one resource and is waiting for one more. Since there are 3 processes and 4 resources, one process must be able to obtain two resources. This process requires no more resources and therefore it will return its resources when done.

6.2 Consider a system with m resources of same type being shared by n processes. Resources can be requested and released by processes only on at a time. The system is deadlock free if and only if

a. The sum of all max needs is < m+n

b. The sum of all max needs is > m+n

c. Both of above

d. None

Ans: a

6.3 A proposed solution to the Dining Philosophers deadlock problem is as follows:

```
Philosopher( int i ) {
while(1) {
think();
// grab forks if we can
lock.P()
fork[i].P();
fork[(i+1)%5].P();
lock.V();
eat();
// put down forks
lock.P();
fork[i].V();
fork[(i+1)%5].V();
lock.V();
}
```

There are five philosophers and five forks. All the lock and fork semaphores are initialized to 1.

a. **Is the second lock.P/lock.V pair necessary? Why or why not?**
b. **If the second lock.P/lock.V pair isn't necessary, are there any negative consequences to having it there?**

Ans:

a. No, it isn't necessary. There is no conflict that can occur doing the V operation, just the P
b. Yes, there are negative consequences. One innocuous one is that it can cause extra context switches if a philosopher is putting down the forks and has the lock, it will block any philosophers trying to gain access to pick up the forks until both forks are back on.

6.4 Here's a routine used in a multi-threaded process:

```
(void *)some_function( void *arg ) {
if (arg ) {
shared_data1 += 1;
shared_data2 *= shared_data1;
}
else {
cout << "error" << endl;
}
} // some_function
```

The variables shared_data1 and shared_data2 are shared by all the threads in the process. Assume a variable of class Semaphore has been declared, named lock, and is accessible to all the threads.

a. Add operations on lock to ensure mutual exclusion in the critical section of the above function. It is important that the thread be blocked for the minimum time absolutely necessary to ensure the mutual exclusion.

b. What value should lock be initialized to?

Ans:

```
(void *)some_function( void *arg ) {
if (arg ) {
lock.P();
shared_data1 += 1;
shared_data2 *= shared_data1;
lock.V();
}
else {
cout << "error" << endl;
}
} // some_function
```

What value should lock be initialized to? 1

Explanation: We don't need to protect the check of arg as it is local to the routine, nor do we need to protect anything in the else clause, as it accesses no data, so we only need to protect the lines shown. The initial value is 1 to allow exactly one process to initially enter the critical section.

6.5 Three threads need to synchronize their actions so that the following holds:

P1 -> data1 -> P2 -> data2 -> P3

meaning that P1 produces data1, and only when data1 is ready can P2 proceed, etc.

You are guaranteed that each thread will be executed only once, and there are no cycles (ie. what you see above is the whole story).

Here's the skeleton of each thread (note: where it says "do some work" or "do more stuff", it means do work independent of the shared data):

thread1() { thread2() { thread3() {

// do some work //do some work //do some work

....

....

// produce data1 //do stuff with data1 //do stuff with data2

data1 := 1; //and produce data2 cout << data2 <<endl;

data2 := data2 * data1;

//do more stuff //do more stuff // do more stuff

....

} } }

Add Semaphores and operations on them as needed, assuming that all threads can access them, to ensure the proper order of execution. Indicate the initial value for each Semaphore.

Ans:

Sempahore mutex1(0), mutex2(0);

thread1() { thread2() { thread3() {

// do some work //do some work //do some work

....

// produce data1 //do stuff with data1 //do stuff with data2

mutex2.P();

data1 := 1; //and produce data2 cout << data2 <<endl;

mutex1.V(); mutex1.P();

data2 := data2 * data1;

mutex2.V();

//do more stuff //do more stuff // do more stuff

....

} } }

6.6 A system has four processes P1 through P4 and two resource types R1 and R2. It has 2 units of R1 and 3 units of R2. Given that:

P1 requests 2 units of R2 and 1 unit of R1

P2 holds 2 units of R1 and 1 unit of R2

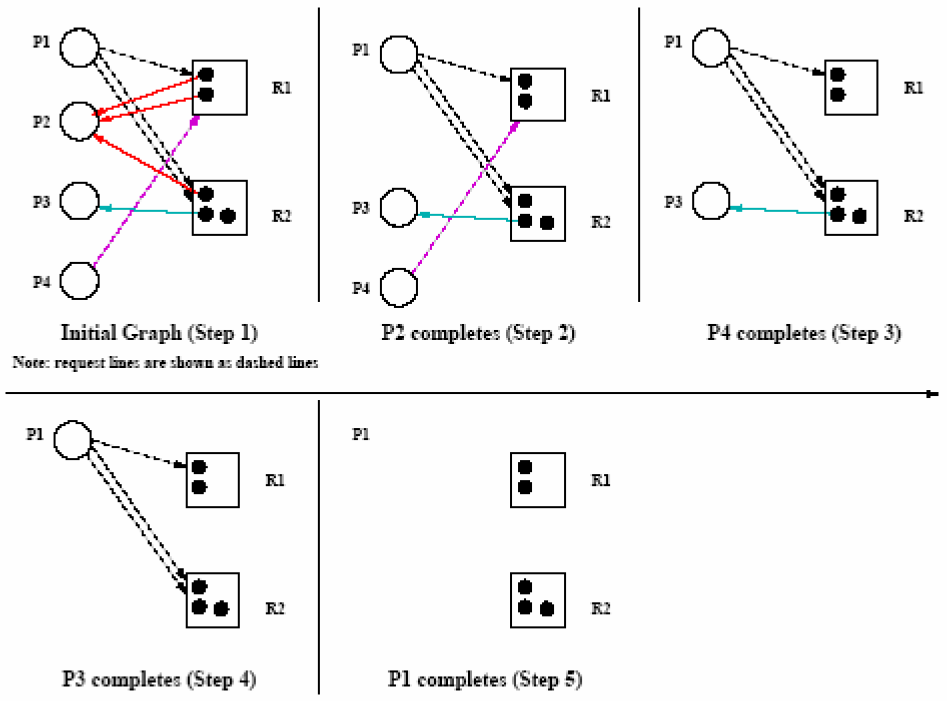
P3 holds 1 unit of R2

P4 requests 1 unit of R1

Show the resource graph for this state of the system. Is the system in deadlock, and if so, which processes are involved?

Ans:

Here is one complete solution:



There are many other orders in which things could complete. But one rule is that P2 must complete before P1 completes, as it hold the all the R1 resources and P1 needs one of them.

6.7 A system has five processes P1 through P5 and four resource types R1 through R4.

There are 2 units of each resource type. Given that:

P1 holds 1 unit of R1 and requests 1 unit of R4

P2 holds 1 unit of R3 and requests 1 unit of R2

P3 holds one unit of R2 and requests 1 unit of R3

P4 requests 1 unit of R4

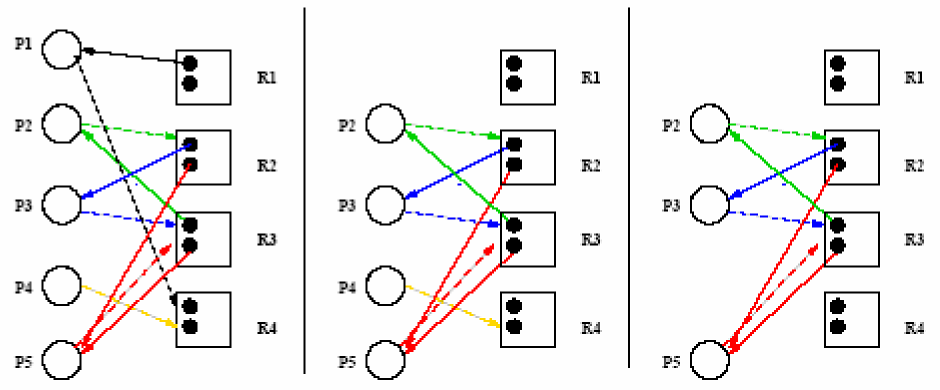
P5 holds one unit of R3 and 1 unit of R2, and requests 1 unit of R3

Show the resource graph for this state of the system. Is the system in deadlock, and if

so, which processes are involved?

Ans:

Here is one complete solution:



Initial Graph (Step 1) P1 completes (Step 2) P4 completes (Step 3)

Note: request lines are shown as dashed lines

At this point no other processes can complete, so the system is deadlocked, and process P2, P3, and P5 are involved.

An alternate ordering of reducing the graph, which leads ot the same result is to have P4 complete first, then P1.

6.8 Given 5 total units of the resource, tell whether the following system is in a safe or unsafe state.

Process	**Used**	**Max**
P1	**1**	**2**
P2	**1**	**3**
P3	**2**	**4**
P4	**0**	**5**

The system is in a safe state. Here's why. You have 1 unit left. If you give it to P1, it can complete and release the two units. Then P2 and P3 can complete in either order. When P3 is done, it frees up enough resources for P4 to complete.

6.9 Given a total of 5 units of resource type 1 and 4 units of resource type 2, tell whether the following system is in a safe or unsafe state. Show your work.

	Type 1		**Type 2**	
Process	**Used**	**Max**	**Used**	**Max**
P1	**1**	**2**	**1**	**3**
P2	**1**	**3**	**1**	**2**

P3	**2**	**4**	**1**	**4**

Ans:

The system is in an unsafe state. In order to be sure a process can complete, it will need its request of *all* resource types to be satisfied, so you need to look at the need for each process for all resource types and see if what is available can satisfy any process's needs.

In this example we have the following needs:

Process	Type 1	Type 2
P1	1	2
P2	2	1
P3	2	3

As we have only 1 unit of each resource type available, and no process has a need that can be satisfied, so the system is unsafe.

6.10 Given a total of 10 units of a resource type, and given the safe state shown below, should process 2 be granted a request of 2 additional resources? Show your work.

Process	**Used**	**Max**
P1	**2**	**5**
P2	**1**	**6**
P3	**2**	**6**
P4	**1**	**2**
P5	**1**	**4**

Ans:

The system is in an unsafe state. If we give the 2 units to P2, we would have the following needs:

Process	Need
P1	3
P2	3
P3	4
P4	1
P5	3

We have 1 unit left, which we can give to P4, which will release only 2 units. Of the processes remaining, no process can complete with only 2 units, so the system is unsafe.

6.11 Given a total of 10 units of a resource type, and given the safe state shown below, should process 2 be granted a request of 2 additional resources? Show your work.

Process	**Used**	**Max**
P1	**2**	**5**
P2	**1**	**6**
P3	**2**	**6**
P4	**1**	**2**
P5	**1**	**4**

Ans:

The system is in an unsafe state. If we give the 2 units to P2, we would have the following needs:

Process	Need
P1	3
P2	3
P3	4
P4	1
P5	3

We have 1 unit left, which we can give to P4, which will release only 2 units. Of the processes remaining, no process can complete with only 2 units, so the system is unsafe.

6.12 List three examples of deadlocks that are not related to a computer-system environment.

Ans:

- Two cars crossing a single-lane bridge from opposite directions.
- A person going down a ladder while another person is climbing up the ladder.
- Two trains traveling toward each other on the same track.

6.13 Consider a system consisting of four resources of the same type that

are shared by three processes, each of which needs at most two resources. Show that the system is deadlockfree.

Ans:

Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

6.14 When a process is rolled out of memory, it loses its ability to use the CPU (at least for a while). Describe another situation where a process loses its ability to use the CPU, but where the process does not get rolled out.

a. When an interrupt occurs.
b. When thrashing occurs.
c. When deadlock occurs.
d. While swapping.

Ans: a