Deadlock and Starvation

6.1 PRINCIPLES OF DEADLOCK

1. Draw a joint progress diagram that may cause a deadlock for one of its progress paths and show a progress path that will cause the deadlock.
Draw another joint progress diagram that is deadlock-free for all of its progress paths.

2. What are the 3 common approaches that can deal with all types of deadlock?

3. In the code below, three processes compete for six resources labeled A to F.

a. Using a resource allocation graph, show the possibility of a deadlock in this implementation.

```
void P0()
{
  while(true) {
    get(A);
    get(B);
    get(C);
    // critical
section
    // use A, B, C
    release(A);
    release (B);
    release (C);
  }
}
```

```
void P1()
{
  while(true) {
    get(D);
    get(E);
    get(B);
    // critical
section
    // use D, E, B
    release(D);
    release (E);
    release (B);
  }
}
```

```
void P2()
{
  while(true) {
    get(C);
    get(F);
    get(D);
    // critical
section
    // use C, F, D
    release(C);
    release (F);
    release (D);
  }
}
```

6.2 DEADLOCK PREVENTION

4. Modify the order of some of the get requests in the previous question to prevent the possibility of any deadlock. You cannot move requests across procedures, only change the order inside each procedure. Use a resource allocation graph to justify your answer.

6.3 DEADLOCK AVOIDANCE

5. Consider a system with a total of 150 units of memory, allocated to three processes as shown below:

Process	Max	Hold
1	70	45
2	60	40
3	60	15

Apply the banker's algorithm to determine whether it would be safe to grant each of the following requests. If yes, indicate a sequence of terminations that could be guaranteed possible. If no, show the reduction of the resulting allocation table.

a. A fourth process arrives, with a maximum memory need of 60 and an initial need of 25 units.
b. A fourth process arrives, with a maximum memory need of 60 and an initial need of 35 units.

6.4 DEADLOCK DETECTION

6. Apply the deadlock detection algorithm to the following data and show the results.

$$\text{Available} = (2 \quad 1 \quad 0 \quad 0)$$

$$\text{Request} = \begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{pmatrix} \qquad \text{Allocation} = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix}$$

7. Consider a system consisting of four processes and a single resource. The current state of the claim and allocation matrices are as follows:

$$\mathbf{C} = \begin{pmatrix} 3 \\ 2 \\ 9 \\ 7 \end{pmatrix} \qquad \mathbf{A} = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 2 \end{pmatrix}$$

What is the minimum number of units of the resource needed to be available for this state to be safe?

6.5 AN INTEGRATED DEADLOCK STRATEGY

8. Consider the following ways of handling deadlock: (1) banker's algorithm, (2) detect deadlock and kill the thread, releasing all resources, (3) reserve all resources in advance, (4) restart thread and release all resources if the thread needs to wait, (5) resource ordering, and (6) detect deadlock and roll back thread's actions.

a. One criterion to use in evaluating different approaches to deadlock is which approach permits the greatest concurrency. In other words, which approach allows most threads to make progress without waiting when there is no deadlock? Give a rank order from 1 to 6 for each of the ways of handling deadlock just listed, where 1 allows the greatest degree of concurrency. Comment on your order.

b. Another criterion is efficiency; in other words, which requires the least processor overhead. Rank order the approaches from 1 to 6, with 1 being the most efficient, assuming that deadlock is a very rare event. Comment on your order. Does your ordering change if deadlocks occur frequently?

6.6 DINING PHILOSOPHERS' PROBLEM

9. Comment on the following solution to the dining philosophers' problem.

a. A hungry philosopher first picks up his left fork; if his right fork is also available, he picks up his right fork and starts eating; otherwise, he puts down his left fork again and repeats the cycle.

b. A hungry philosopher picks up both forks at the same time (in a critical section), only if both forks are available.

GENERAL QUESTIONS

10. Consider the following snapshot of a system. There are no outstanding unsatisfied requests for resources.

available			
r1	r2	r3	r4
2	1	0	0

	current allocation				maximum demand				still needs			
process	r1	r2	r3	r4	r1	r2	r3	r4	r1	r2	r3	r4
p1	0	0	1	2	0	0	1	2				
p2	2	0	0	0	2	7	5	0				
p3	0	0	3	4	6	6	5	6				
p4	2	3	5	4	4	3	5	6				
p5	0	3	3	2	0	6	5	2				

a. Compute what each process still might request and display in the columns labeled “still needs”
b. Is this system currently in a safe or unsafe state? Why?
c. Is this system currently deadlocked? Why or why not?
d. Which processes, if any, are or may become deadlocked?
e. If a request from p3 arrives for (0, 1, 0, 0), can that request be safely granted immediately? Which processes, if any, are or may become deadlocked if this whole request is granted immediately?