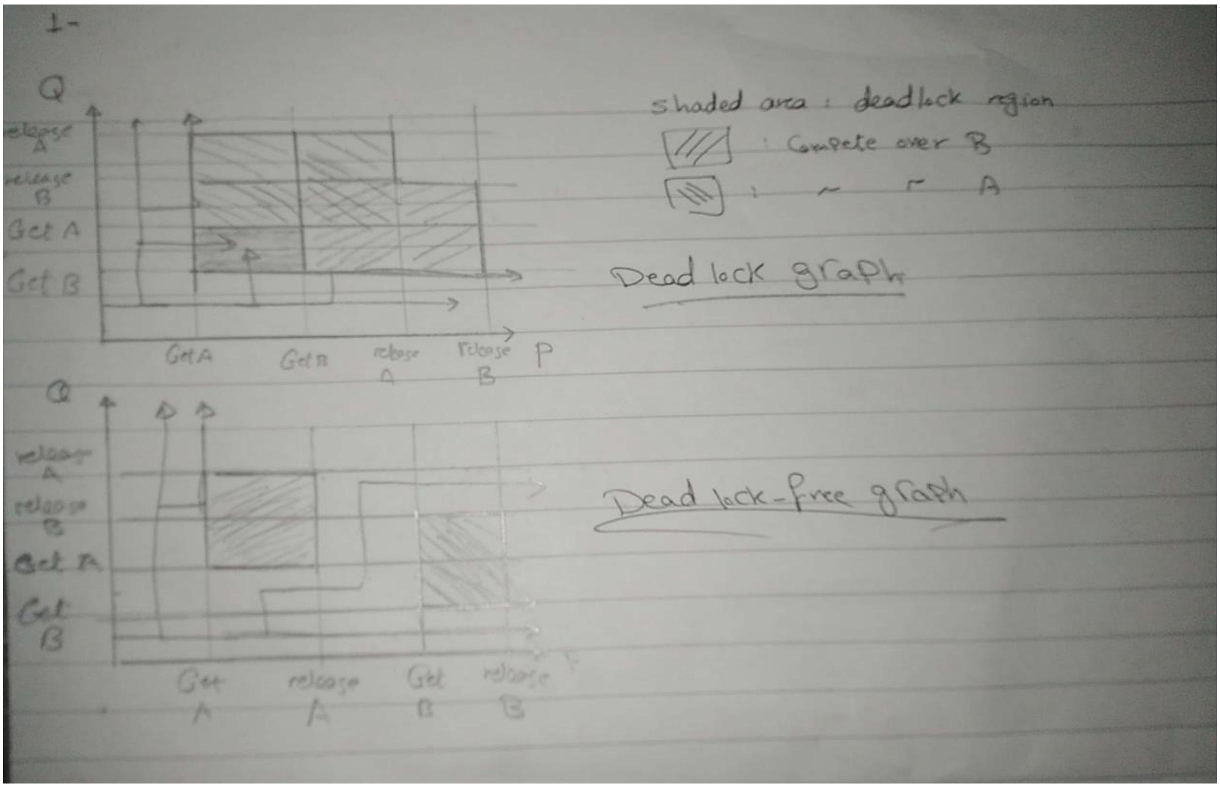
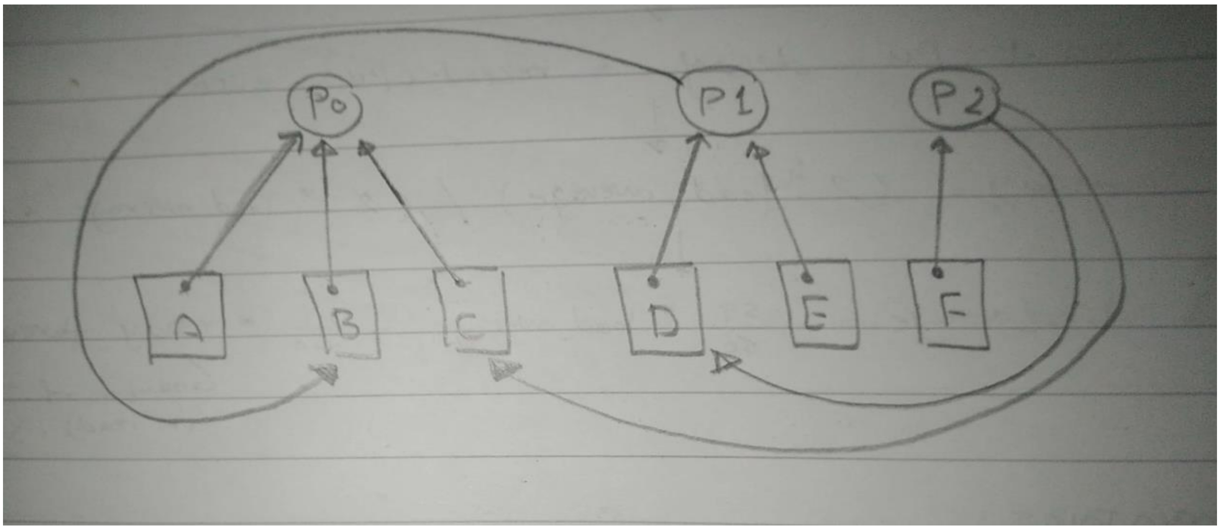
1- 

2-

- Deadlock prevention
- Deadlock avoidance
- Deadlock detection

3-



4- We can use linear ordering of resources to prevent any cycles and hence, prevent deadlock as follows.

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

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

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

6.3 DEADLOCK AVOIDANCE

5-

a)

Process	*Max*	*Hold*	*Need*
P1	70	45	25
P2	60	40	20
P3	60	15	45
P4	60	25	35

Available memory = 150 - 45 - 40 - 15 = 50 units

We now need to check whether granting additional 25 units is a safe state or not. Process 4's new holding value will be 25. The available memory will become 25 units. The maximum total memory needed by all processes is 70 + 60 + 60 + 60 = 250 units. Therefore, the available memory of 25 units is not enough to satisfy the maximum needs of all processes, and granting the request would result in an UNSAFE state.

b)

Process	*Max Allocation*	*Current Allocation*	*Need*	*Work*
P1	70	45	25	105
P2	60	40	20	110
P3	60	15	45	135
P4	60	35	25	115

Assuming that all processes are not finished. We keep checking until all processes are marked as finished or the system becomes unsafe. We first find a process whose need is less than or equal to the current work, and mark it as finished, then release its resources. We repeat this until either all processes are finished or we cannot find any process whose need is less than or equal to the current work. It is obvious that no process can finish since there are not enough resources to satisfy any process's needs. Therefore, it is NOT safe to grant the fourth process a maximum memory need of 60 and an initial need of 35 units.

7- banker's algorithm can be used to solve the problem. First, define N = C – A = [2;1;6;5]. The available resources = 7.

- For p1: N1 = 2, available = 5. Can finish
- For p2: N2 = 1, available = 6. Can finish
- For p3: N3 = 6, available = 9. Can finish
- For p4: N4 = 5, available = 10. Can finish.

The number of units needed = 10.

8- a)

i. Resource ordering
ii. Banker's algorithm
iii. Restart thread and release all resources if the thread needs to wait
iv. Detect deadlock and rollback thread's action
v. Reserve all resources in advance
vi. Detect deadlock and kill the thread, releasing all resources

Resource ordering method allows for maximum concurrency as the resources are carefully allocated. Banker's algorithm allows the maximum number of processes to be running given that the system will not be in an unsafe state. The third approach allows the threads to run as soon as the resources are available, but it has an excessive overhead due to all the operations involved. The fourth method involves rollback back from a deadlock situation which has way more overhead the restarting the thread. Reserving resources in advance requires that all threads are aware of the resources they will need and they can't run unless all the resources are available which is not efficient if the thread can proceed with the resources available at the moment until the other resources are available. The last approach is the least efficient as it causes loosing all the work of the threads the caused the deadlock.

b)

i. Resource ordering

ii. Banker’s algorithm
iii. Restart thread and release all resources if the thread needs to wait
iv. Detect deadlock and kill the thread
v. Reserve all resources in advance
vi. Detect deadlock and roll back

The first approach doesn’t have any overhead. Banker’s algorithm requires additional overhead to check the state of the system. The third approach requires restarting the thread and releasing the resources. In the detection method, more overhead is involved in order to detect and kill the thread. Reserving resources results in great overhead to reserve resources in advance. The las approach requires the most overhead to rollback from a deadlock situation.

9-

a. I think that this solution is inefficient. In this solution, a philosopher will repeatedly pick up and put down his left fork if the right one is not available. This will waste cpu cycles. Also, if the philosopher can never reach the right fork before the philosopher next to him, this will lead to starvation.

b. This solution is also inefficient as it will result in low concurrency. Let’s say that we have 2 philosophers and 2 forks. If one of them could reach both of the forks, the other one will stay idle. Even though if each one of them can eat with only one fork until the other fork is not used anymore. The solution may also lead to a deadlock if each one of them managed to reach one fork but not the other.