Assignment NO. 4

Q.sc] Real-time system :-

Q.1] What is critical section problem? explain peterson's solution for solving critical section problem.

A]

→ ① consider a system consisting of n processes $\{P_0, P_1, ---, P_{n-1}\}$

② each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file & so on.

③ The important feature of the system is that, when one process is executing in its critical section, no other process is allowed to execute in its critical section.

④ That is, no two processes are executing in their critical sections at the same time.

⑤ The critical-section problem is to design a protocol that the processes can use to cooperate.

⑥ each process must request permission to enter its critical section.

⑦ The section of code implementing this request is entry section.

⑧ The critical section may be followed by an exit section. The remaining code is the remainder section.

B] Peterson's Sol$^n$. :-

① A classic software-based sol$^n$ to critical section problem known as peterson's sol$^n$.

② Peterson's sol$^n$. is restricted to two processes that alternate execution bet$^n$. their critical sections & remainder sections.

③ The processes are numbered P0 and P1. For convenience, when presenting Pi, we use Pj to denote the other process; that is, j equals 1−i.

④ In peterson's solution, we have two shared variables:

i] Boolean flag[i]: Initialized to FALSE, initially no one is interested in entering the critical section.

ii] int turn: The process whose turn is to enter the critical section.

④ The variable turn indicates whose turn it is to enter its critical section.

⑤ i.e., if turn == i, then process Pi is allowed to execute in its critical section.

⑥ The flag array is used to indicate if a process is ready to enter its critical section. For ex if flag[i] is true, this value indicates that Pi is ready to enter its critical section.

```
do {
    flag[i] = TRUE;
    turn = j;
    while (flag[j] && turn == j);
        [critical section]
    flag[i] = FALSE;
        [Remainder section]
} while (TRUE);
```

c] Peterson's soln preserves all three conditions:
A] Mutual exclusion is assured as only one process can access the critical section at any time.
B] Progress is also assured, as a process outside the critical section does not blocks other processes from entering the critical section.
c] Bounded waiting is preserved as every process gets a fair chance.

disadvantages:-
1) It involves busy waiting
2) It is limited to 2 processes.

Q.2] Write Note:-

a] Mutex Locks :-
(1) OS designers build software tools to solve the critical section problem. The simplest of these tools is mutex lock.
(2) The term mutex is short for mutual exclusion.
(3) A mutex is ~~bim~~ binary semaphore variable whose purpose is to provide locking mechanism. It is used to provide mutual exclusion to a section of code, means only one process can work on particular code section at a time.
(4) Typically, when a program is started it creates a mutex for a given resource at the beginning by requesting it from the system & the system returns a unique name.
(5) After that, any thread needing the resource must use the mutex to lock the resource from other threads while it is using the resource.

(6) If the mutex is already locked, a thread needing the resource is typically queued by the system & then given control when the mutex becomes unlocked.

(7) We use the mutex lock to protect critical regions & thus prevent race conditions. i.e. a process must acquire the lock before entering a critical section.

(8) it release the lock when it exists the critical section. The acquire() function acquires the lock & the release() function release the lock.

(9) A mutex lock has Boolean variable available whose value indicates if the lock is available or not.

(10) If lock is available, a call to acquire() succeeds & the lock is then considered unavailable.

(11) ex. ATM machine

(12) A] The definition of acquire() is as follows:

```
acquire() {
    while (!available)
        ; /* busy wait */
    available = false;
}

do {
    [acquire lock]
        critical section
    [release lock]
        remainder section
} while (true);
```

B] The defn of release() is:

```
release() {
    available = true;
}
```

(13) 1) The main disadv. of implementation given here is, it requires busy waiting.
2) While a process in in its critical section, any other process that tries to enter its critical section must loop cont. in the call to acquire().

b] Semaphores:-
① semaphore is nothing but a synchronization tool with the help of which we can ensure that the critical section can be accessed by the processes in mutually exclusive ~~manner~~ manner.
② A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operation: wait() and signal()
③ wait() operation was originally termed P; Signal() was originally called V.
④ The defn. of wait():

```
wait(s) {
    while (s<=0);    //busy wait
    s--;
}
```

⑤ ~~I~~

⑤ The defⁿ of signal() :

~~Signal()~~ Signal(s) {

s++;

}

⑥ Semaphores is simply a variable. This variable is used to solve critical section problem & to achieve process synchronization in multi processing environment.

⑦ i) Wait() :- It is called when a process wants to access a resource. When the semaphore variable is negative, the process wait is blocked.

ii) signal() :- It is called when a process is done using a resource.

⑧ ① P operation is also called wait, sleep or down operation and V operation is also called signal, wake-up or up operation.

② Both operations are atomic and semaphore(s) is always initialized to one.

③ A critical section is surrounded by both operations to implement process synchronization.

④ Critical section of process P is in betⁿ P and V operation.

```
Process P
//some code
P(s);
//critical section
V(s)
//remainder section
```

c] classic problems of synchronization :

A] A solution to a process synchronization problem should meet three important criteria:

1] correctness: Data access synchronization & control synchronization should be performed in accordance with synchronization requirments of problem.

2] Maximum concurrency: A process should be able to operate freely except when it needs to wait for other processes to perform synchronization actions.

3] No busy waits : To avoid performance degradation, synchronization should be performed through blocking rather than through busy waits.

B] critical sections & signaling are the key elements of process synchronization, so a solution to a process synchronization problem should incorporate a suitable combination of these elements.

Q.3] What is deadlocks? explain conditions for occurance of deadlock?

→ (1) The implementation of semaphore with a waiting queue may result in a situation where two or more processes are waiting idenfinitely for an event that can be caused only by one of waiting processes.

(2) The event in question is the execution of signal() operation. When such a state is reached, these processes are said to be deadlocked.

Q.3] What is deadlock? explain conditions for occurance of deadlock.

A]

→ (1) In multiprogramming environment, several processes may compete for finite no. of resources.

(2) A process requests resources; if the resources are not available at the time, the process enters a waiting state.

(3) Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes.

This situation is called deadlock.

B] conditions:-

1] Mutual exclusion:-

At least one resource must be held in non sharable mode; i.e. only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2] Hold and wait:-

A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

3] No preemption:

Resources cannot be preemted; i.e. a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4] circular wait:
A set {P0, P1, ---Pn} of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, --- Pn-1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0.

Q.4] explain deadlock prevention techniques.

→ 1] Mutual exclusion:-
① We need to categorization of all the resources in sharable & non sharable.
② The mutual exclusion condition must hold.
③ shared resources such as read only files do not lead to deadlocks.
④ Unfortunately some resources, such as printers and tape drives, require exclusive access by single process.

2] Hold and wait:-
① To ensure that the hold-and-wait cond". never occur in the system, we must guarantee that, whenever a process request resource, it does not hold any other resources.
② Require process to request & be allocated all its resources before it begins execution.
ex.- process coping data from a tape drive to disk file, sort file and print results to a printer.

3] No preemption :-

(1) Preemption of process resource allocation can prevent this condition of deadlock, when it is possible.

(2) The third necessary cond$^n$ for deadlocks is that there be no preemption of resources that have already been allocated.

(3) To ensure that this cond$^n$ does not hold we can use following protocol :-

(1) If a process is holding some resources & requests another resource that cannot be immediately allocated to it, then all resources the process is currently holding are preempted.

(2) In other words, these resources are implicitly released. The preempted resources are added to the list of resources for which the process is waiting.

4] circular wait :-

(1) To ensure that the circular-wait condition never holds is to determines a total ordering of all resource types & to require that each process requests resources in an increasing order of enumeration.

ex - Let $R = \{R_1, R_2 \dots R_m\}$ be the set of resource types.

Q.5] explain deadlock avoidance techniques.

→ ① safe state:

① A state is safe if the system can allocate resources to each process in some order & still avoid a deadlock.

② More formally, a system is in a safe state only if there exists a safe sequence.

① If $P_i$ resource needs are not immediately available then $P_i$ can wait until all $P_j$ have finished.

② when $P_j$ is finished $P_i$ can obtain needed resources execute, return allocated resources and terminate.

③ When $P_i$ terminates, $P_{i+1}$ can obtain its needed resources & so on

③ A safe state is not a deadlock state.

④ conversely, a deadlocked state is an unsafe state.

⑤ Not all unsafe states are deadlocks, however An unsafe stay may lead to a deadlock.

⑥ As long as the state is safe, the OS can avoid unsafe states.

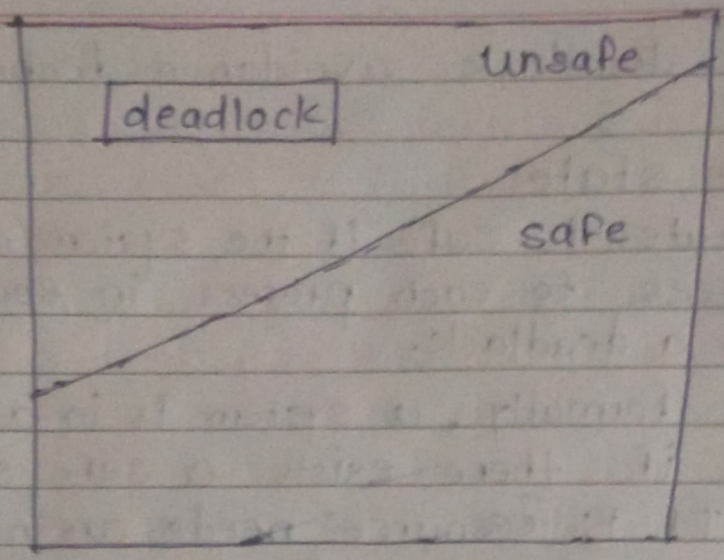
⑦ In an unsafe state, the operating system cannot prevent processes from requesting resources in such a way that deadlock occurs.

⑧ Basic fact:

① If a system is in safe state ⇒ no deadlocks

② If a system is in unsafe state ⇒ possibility of deadlock

③ Avoidance ⇒ ensure that a system will never enter an unsafe state.



Q.7] Write note:-

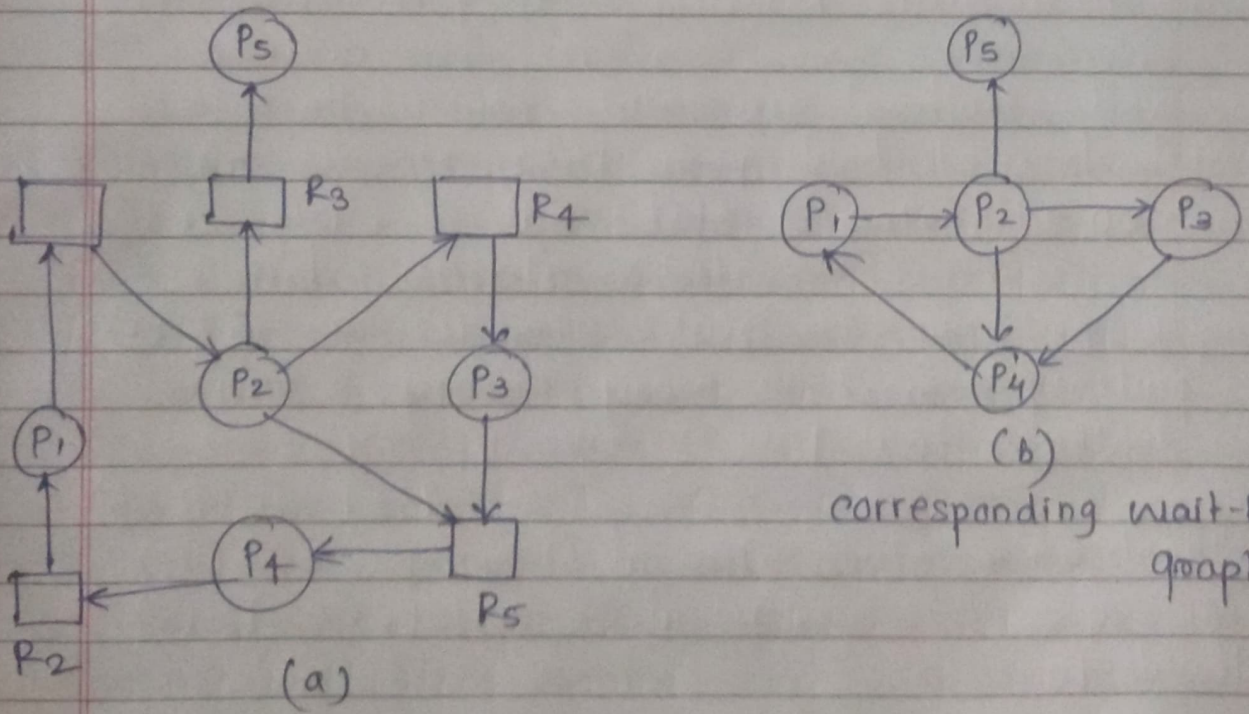
a] wait-for graph -

(1) If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a varient of the resource-allocation graph called a wait-for graph.

(2) We obtain this graph from the resource-allocation graph by removing the resource nodes & collapsing the appropriate edges.

(3) deadlock exists in the system if and only if the wait-for graph contains a cycle.

(4) To detect deadlock, the system needs to maintain the wait for graph & preriodacaly invoke an algorithm that searches for a cycle in a graph.




(a)
Resource allocation graph

(b)
corresponding wait-for graph

b] Recovery for deadlock –

(1) When a detection algorithm determines that deadlock exists, several alternatives are available.

(2) One possibility is to ~~form~~ inform the operator that a deadlock has occured & to let the operator deal with deadlock manually.

(3) Another possibility is to let the system recover from the deadlock automatically.

(4) There are two options for breaking a dealock:-

1) One is simply to abort one or more processes to break the circular wait.

2) The other is to preemp some resources from one / more of the deadlock processes.

Q. 6] explain Banker's alg. with suitable ex.

→ (1) The resource-allocation-graph algorithm is not applicable to a resource allocation system with multiple instances of each resource type.
(2) The banker's alg. is applicable to a resource allocation system with multiple instances of each resource type.
(3) The name was choosen because the alg. could be used in banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.
(4) When a new process enters the system, it must declare the max. no. of instances of each resource type that it may need.
(5) This no. may not exceed the total no. of resources in system.
(6) When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in safe state.
(7) If it will, the resources are allocated; otherwise the process must wait until some other process releases enough resources.

ex.
To illustrate the use of banker's algorithm. consider a system with five processes $P_0$ through $P_4$ and three resource types A, B & C. Resource type A has ten instances, resource type B has five instances & resource type C has seven instances; Suppose that, at time $T_0$,

the following snapshot of the system has been taken:

| | Allocation | Max | available |
|---|---|---|---|
| | A B C | A B C | A B C |
| $P_0$ | 0 1 0 | 7 5 3 | 3 3 2 |
| $P_1$ | 2 0 0 | 3 2 2 | |
| $P_2$ | 3 0 2 | 9 0 2 | |
| $P_3$ | 2 1 1 | 2 2 2 | |
| $P_4$ | 0 0 2 | 4 3 3 | |

The content of matrix Need is defined to be Max − allocation and is as follows:

| | Need |
|---|---|
| | A B C |
| $P_0$ | 7 4 3 |
| $P_1$ | 1 2 2 |
| $P_2$ | 6 0 0 |
| $P_3$ | 0 1 1 |
| $P_4$ | 4 3 1 |

We claim that the system is currently in safe state. Indeed, the sequence $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfies the safety criteria. Suppose now that process $P_1$ requests one additional Instance of resource type A & two Instances of resource type C, so $Request_1 = (1,0,2)$. To decide whether this request can be $(1,0,2) \leq (3,3,2)$, which is true. We then pretend that this request has been Fulfilled & we arrive at the following new state:

| | Allocation | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C |
| $P_0$ | 0 | 1 | 0 | 7 | 4 | 3 | 2 | 3 | 0 |
| $P_1$ | 3 | 0 | 2 | 0 | 2 | 0 | | | |
| $P_2$ | 3 | 0 | 2 | 6 | 0 | 0 | | | |
| $P_3$ | 2 | 1 | 1 | 0 | 1 | 1 | | | |
| $P_4$ | 0 | 0 | 2 | 4 | 3 | 1 | | | |

We must determine whether this new system state is safe. To do so, we execute our safety alg. & find that the sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies the safety requirment. Hence, we can Immediately grant the request of process $P_1$.