Gamya OS 5th

UNIT-V DEADLOCKS

S Q.WHAT IS DEAD LOCK AND WHAT ARE THE NECESSARY CONDITIONS TO DEAD LOCK OCCUR?

A) A process request resources, if the resources are not available at that time, then the processes enters into the waiting state. Because the requested resources are held by the other waiting process. Now both processes are in waiting state. This situation is called as "deadlock"(always waiting).

Four necessary conditions are there to dead lock occur each and every dead lock must satisfy the following 4 conditions.

1.MUTUAL EXCLUSION:-mutual exclusion means,only one process at a time can use the resources the resources are in non-sharable.if another process request the resource the requesting process must wait until the resource has been released.

2.HOLD AND WAIT:-it means a process that is holding atleast one resource and waiting for another resource.

3.NO-PREMPTION:-it means the process does not release the resource at he middle of processing the resource will release only the process completed its task.

4.CIRCULAR WAIT:-{p0,p1,p2,……..pn-1}is a set of waiting processes p0 is a waiting for a resource i.e; held by p1.p1 is waiting for a resource that is held by p2 and so on pn-1 is waiting for a resource by p0.

Q.EXPLAIN ABOUT RESOURCE-ALLOCATION GRAPH?

Deadlocks can be described in terms of a directed graph called a *system resource- allocation graph*. This graph consists of a set of vertices *V* and a set of edges *E*.

1) V is partitioned into two types:

a) $P = \{P1, P2, \ldots, Pn\}$, the set consisting of all the processes in the system.

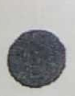
b) $R = \{R1, R2, \ldots, Rm\}$, the set consisting of all resource types in the system.

2) request edge – directed edge $P1$ Rj

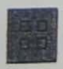
3) assignment edge – directed edge Rj Pi

The following symbols are used in the resource allocation graph :

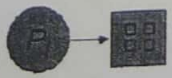
a) Process



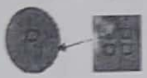
b) Resource Type with 4 instances



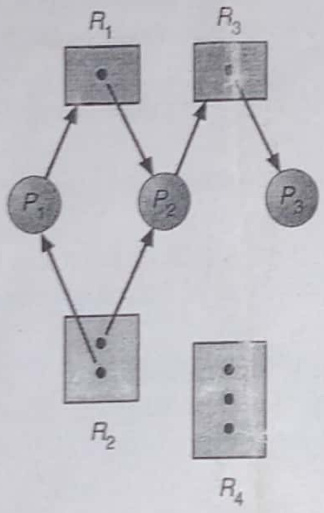
c) P_i requests instance of Rj



d) P_i is holding an instance of R_j



Example of a Resource Allocation Graph :



1
2
3
4
5
6

The above graph is describing the following situation :

1) The sets P, R and E are :

$P = \{ P1, P2, P3 \}$

$R = \{ R1, R2, R3 \}$

$E = \{ P1 \rightarrow R1, P2 \rightarrow R3, R1 \rightarrow P2, R2 \rightarrow P2, R2 \rightarrow P1, R3 \rightarrow P3 \}$

2) Resource instances :

- One instance of resource type $R1$

- Two instances of resources type $R2$

- One instance of resource type $R3$

- Three instances of resource type $R4$

3) Process states :

- Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1.
- Process P_2 is holding an instance of R_1 and R_2 and is waiting for an instance of resource type R_3.
- Process P_3 is holding an instance of R_3.

If the graph consists ***no cycles****, then no process in the system is deadlocked. If the graph* ***contains a cycle****, then a deadlock may exist.*

E **Q) HOW TO PREVENT DEADLOCK ?**

Deadlock prevention is a set of methods for ensuring that at least one of the necessary conditions cannot hold.

a) Mutual Exclusion: The mutual exclusion condition must hold for non sharable resources. For example, a printer can not be accessed by more than one process at a time. But Read-only files are good example for sharable resources and can not be involved in deadlock. A process never needs to wait for sharable resources. Hence we can not prevent deadlocks by denying the mutual exclusion condition because some resources are basically non sharable.

b) Hold and Wait: To ensure that the hold and wait condition never occurs in the system. There are two protocols to break the hold and wait.

Protocol 1 – Each process to request and be allocated all its resources before it begins execution.

Protocol 2 – A process request some resources only when the process has none. A process may request some resources and use them. Before it can request any additional resources, it must release all the resources which are hold by it.

By using an example we differentiate between these two protocols: A process that copies data from a tape drive to a disk file, sorts the disk file and then prints the results to a printer.

According to protocol 1, the process must request all the resources such as tape drive, disk drive and printer before staring its execution. If they are available, they will be allocated to it. Otherwise, the process has to wait to start its execution until getting the all resources. Though all resources are allocated initially, the process can use the printer at the end.

According to protocol 2, the process can request initially only the tape drive and disk file. It copies from the tape drive from disk and release both. Again the process request the disk file and printer. After copying the disk file to the printer, it releases two resources and terminates.

Disadvantages: These two protocols have two main disadvantages.

1) *Resource utilization may be low* – many of the resources may be allocated but unused for a long period. In the example, the process release the tape drive and disk file, and then again request the disk file printer. If they are available at that time we can allocate them. Otherwise the process must wait.
2) Starvation is possible – A process that needs several popular resources may have to wait indefinitely because at least one of the resources that it needs is always allocated to some other process.

c) No Preemption: The third necessary condition is that there is no preemption of resources that have already been allocated. To ensure that this condition does not hold, we can use the following two protocols:

Protocol 1 – If a process is holding some resources and requests another resource that be immediately allocated to it. All the resources currently being are preempt preempted resources are added to the list of resources for which the process is waiting process will be restarted when it regain its old resources as well as the new ones.

Protocol 2 – A process requests some resources, we first check whether they are available. If they are available, we can allocate them. If they are not available, first we check whether they are allocated to some other process that is waiting for additional resources. If so, preempt them from that process and allocated to the requesting process. If they are not available, the requesting process must wait. It may be happen that while it is waiting, some of its existing resources may be preempted due to the requesting of some other process.

d) Circular Wait: The fourth and final condition for deadlocks is the circular-wait. To ensure that this condition never holds in the system. To do so, each process requests resources in an increasing order of enumeration.

Let R = { R_1, R_2, R_3, ..., R_m } be the set of resource types. We assign to each resource type unique integer number, which allows us to compare two resources and to determine whether one precedes another in our ordering.

We define a one-to-one function F: R->N, Where N is the set of natural numbers. For example, F (tape drive) =1

F (disk drive) =5

F (printer) =12

To prevent the deadlocks we can follow the following protocols:

Protocol 1- Each process can request resources in an increasing order only. A process can request the resources if and only if F (R_j) > F (R_i). If several instances of the same resource type are needed, as single request for all of them must be issued.

Protocol 2 – Whenever a process requests an instance of resource type R_j , it has released any resources R_i such that F (R_i) F (R_j).

If these two protocols are used, then the circular wait condition cannot be hold. Let the set of processes { P_0, P_1, ..., P_n }where P_i is waiting for a resource R_i , which is held by P_{i+1}. Process P_{i+1} is holding resource R_i, while requesting resource R_{i+1}, we must have F (R_i) < F (R_{i+1}), for all i. But this condition means that F (R_0) < F (R_1) < .... F (R_n) < F (R_0). By transitivity, F (R_0) < F (R_0), which is impossible. Therefore, there can be no circular wait.

If we follow these protocols to fail one of the 4 conditions for the deadlock, we can easily prevent deadlock.

Q.EXPLAIN ABOUT AVOIDANCE WITH THE HELP OF BANKER'S ALGORITHM? (OR) EXPLAIN ABOUT SAFETY ALGORITHM AND RESOURCE REQUEST ALGORITHM?

The bankers algorithm contains 2 algorithms one is safety and resource algorithm at the time of resource allocation first safety algorithm executes if the system is safe state then rersource request algorithm executed if the system is in unsafe then the process must be wait for the resource.

The two algorithms requires some data stuctures.those are
AVAILABLE:-it indicates the no.of available resource of each type if available [j]=k then there are k resources are available.
MAX:-it indicates the maximum demand of each process if max[I,j]=k then k instances of resources
ALLOCATION:-it indicates the no.of resources of each type [I,j]=k then k instances of resources
NEED:-if need then [i,j]=k then instances of resources i.e;
Need[i,j]=maxi[i,j]-allocation [i,j]

SAFETY ALGORITHM:-
This algorithm finds whether the system is in a safe or not

```
STEP 1:-let work and finish the vectors of length m,n respectively initialize
        Work=available
        Finish[i]=false
for(i=1,2,….,n)
STEP2:-find an I such that both
1.finish [i]=false
2.need<=work
If no such I exist then goto step 4.
STEP3:-work=work+allocation[i]
         Finish[i]=true
         Goto step2
STEP4:-if finish[i]=true for all I then the system is in safe state.
```

RESOURCE REQUEST ALGORITHM:-

```
STEP1:-if(request<=need)then goto step2 otherwise raise an error condition.
STEP2:-if(request<=available)then go to step3
Otherwise the resources are not available.
STEP3:-if the requested resources are allocated to the process then
1.availble=available=available-request
2.allocation=alloacation+request
3.need=need-request.
```

S **Q.EXPLAIN ABOUT DEADLOCK DETECTION?**

A. Sometimes we have not available the protocols which will ensure "NO Deadlock" in our system. In this case, deadlock may arrive into our system. Then we have to use deadlock detection & recovery procedures. Deadlock is an algorithm which is invoked periodically & it determines whether deadlock has occurred in our system or not. If the algorithm detects a deadlock in the system, we need to use the recovery procedure.

The following is the deadlock detection algorithm. Detection algorithm uses information about processes, resource allocation & outstanding requests from each process and availability of resources.

Deadlock detection algorithm uses the following data structures:

Available: A vector of length *m* indicates the number of available resources of each type.

Allocation: An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.

Request: An *n* x *m* matrix indicates the current request of each process. If *Request*

[i, *j*] = *k*, then process P_i is requesting *k* more instances of resource type. *Rj*.

Algorithm:

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively Initialize: (a) *Work* = *Availa*

   (b) For *i* = 1,2, …, *n*, if $Allocation_i$ 0, then

   Finish[i] = false; otherwise, *Finish*[i] = *true*.

2. Find an index *i* such that both: (a) *Finish*[*i*] == *false* (b) $Request_i$ *Work*

   If no such *i* exists, go to step 4.

3. *Work* = *Work* + $Allocation_i$

   Finish[*i*] = *true*

   go to step 2.

4. If *Finish*[*i*] == false, for some *i*, 1 *i* *n*, then the system is in deadlock state.

   Moreover, if *Finish*[*i*] == *false*, then P_i is deadlocked.

Q) HOW TO RECOVERY FROM DEADLOCK?

S After the deadlock detection algorithm determines that a deadlock exists, and then two possibilities exist. One is that a deadlock has occurred and the user needs to deal with deadlock. The other is to let the system recover from the deadlock. There are two options for breaking a deadlock. One solution is to abort one or more processes to break the circular wait (i.e., to kill one of its processes). The second [illegible]ion is to preempt some resources from one or more of the deadlock processes.

1) Proce[illegible] Termination: To eliminate the deadlocks by aborting a process, we use one of the two methods. In both methods, the system reclaims all resources allocated to the terminated process.

a) *Abort all deadlock processes*: This method will break the deadlock cycle. But at a great expense, these processes may have computed for a long of time and the results of these partial computations must be discarded and may be recommended later.

b) *Abort one process at a time until the dead cycle is eliminated*: This method incurs considerable overhead , since after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.

2) *Resume Preemption*: In this method, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken. Here, we need to consider 3 things:

a) *Selection of a victim*: In this, we need to find out which resources and processes are to be preempted. In process termination, we determine the order of preemption to minimize the cost. Cost factors may like number of resources a deadlock process is holding and amount of time a deadlock process has consumed during its execution.

b) *Rollback*: We must rollback the process to some safe state and restart it from that state. It is difficult to determine the safe state. So, the solution is a rollback.

c) *Starvation*: When victim selection is made usually on cost factors, it may happen that the same process is always picked as a victim. As a result, starvation occurs. So we must ensure that a victim is selected for a small / finite number of times. The common solution is to include the number of rollbacks in the cost factor.

E

Q).EXPLAIN DISK SCHEDULING ALGORITHMS

1) The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.

2) Access time has two major components

   a) *Seek time* is the time for the disk are to move the heads to the cylinder containing the desired sector.

   b) *Rotational latency* is the additional time waiting for the disk to rotate the desired sector to the disk head.

3) Minimize seek time

4) Seek time   seek distance

5) Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

6) Several algorithms exist to schedule the servicing of disk I/O requests.

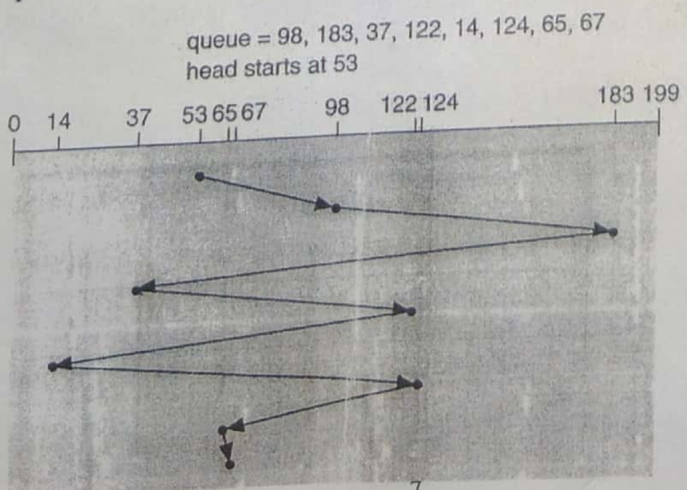
7) We illustrate them with a request queue (0-199).

   98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

1. FCFS Scheduling:

The simplest form of disk scheduling is First-come, First-served (FCFS). This algorithm is basically fair, but it generally does not provide the fastest service.Ex:

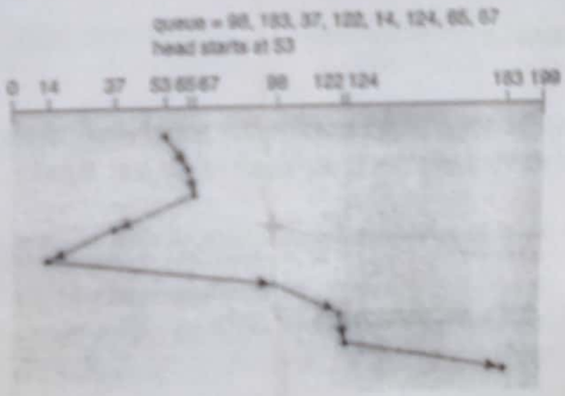


To the above example, total head movement of 640 cylinders.

2. SSTF (Shortest Seek Time First) Scheduling:

a) SSTF selects the request with the minimum seek time from the current head position. b) SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

Ex: This shows total head movement of 236 cylinders.

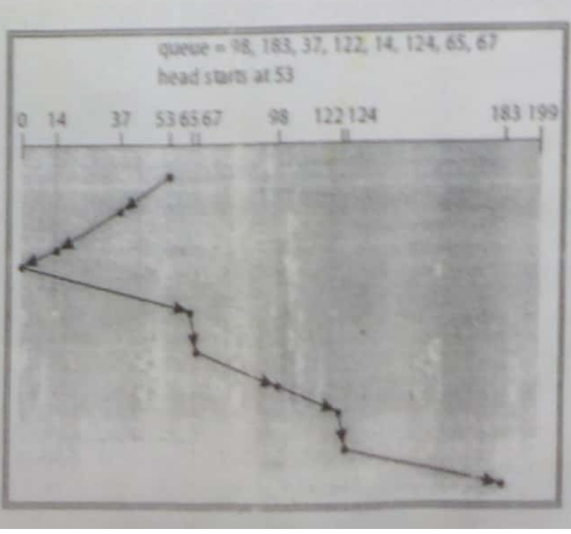


3) SCAN Scheduling:

a) In the SCAN algorithm, the disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

b) Sometimes this algorithm is also called the *elevator algorithm* because the disk arm behaves just like elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.

Ex: This algorithm shows total head movement of 208 cylinders.

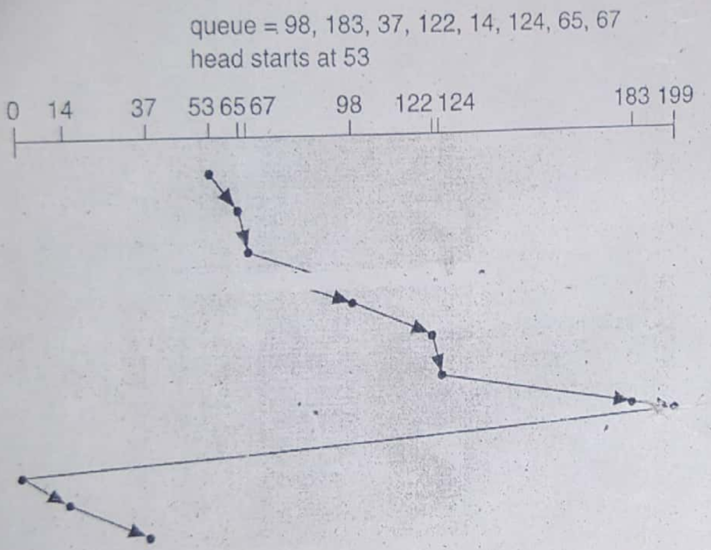


Scheduling:

eduling algorithm provides a more uniform wait time than SCAN.

ead moves from one end of the disk to the other, servicing requests along the way. When it reaches er end, however, it immediately returns to the beginning of the disk, without servicing any requests on eturn trip.

The C-SCAN scheduling algorithm treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

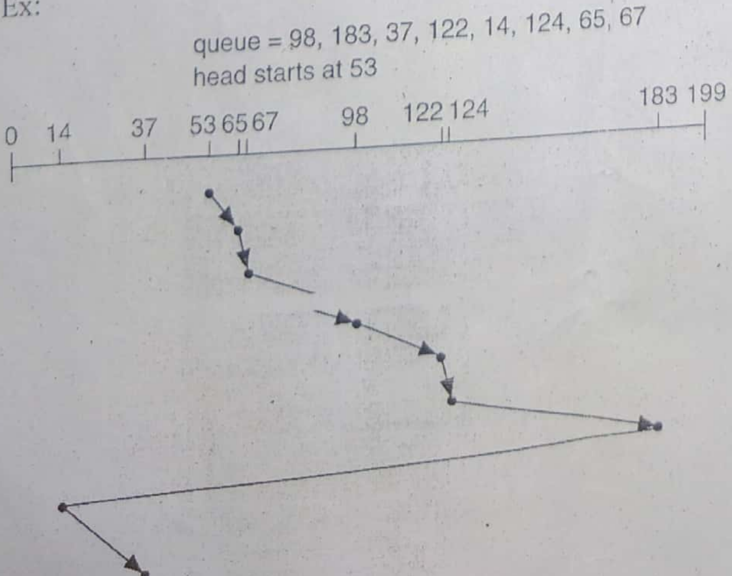


5) C-LOOK Scheduling:

a) This C-LOOK is the version of C-SCAN

b) More commonly, the arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

Ex:


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199