UNIT-5

Resources Access Control

* Assumption on Resources & their Usage :-

A system can have more than one processor and serially reusable resources. Serially reusable resources allocation to the job is done on non-preemptive basis and used in mutually exclusive manner. If a unit of resource R_i is allocated to a job, this unit is not available to other jobs until that job free the unit. Some resources have many units, can be used by more than one job at the same time and each of them used in a mutually exclusive manner.

① Critical Section :-

Critical section is a section of code within process, which may not be executed while another process is in corresponding section of code.

The important point is that when one process is executing shared modifiable data in its critical section, then no other process is allowed to read and write the shared data simultaneously.

Solution to critical section is mutual exclusion condition. It is a method of prevention to ensure that only one process is in a critical section. While one process executes the shared variable, all other processes desiring to do so at that time should be kept waiting, when that process has finished executing the shared variable, next process start execution which is in front of waiting queue.

(2) Enforcement of Mutual Exclusion & critical sections :-

In the enforcement of mutually exclusive accesses of resources, we use a lock-based concurrency control mechanism. When a job wants to use n_i unit of resource R_i, it executes a lock to request them, $L(R_i, n_i)$. When the job does not require the resources, it release the lock by executing an unlock, $U(R_i, n_i)$. When R_i has simply 1 unit, then we use simple notation $L(R_i)$ & $U(R_i)$ for lock and unlock respectively.

(3) Resource Conflict & Blocking :-

When the two jobs request for the resources of the same type then the two jobs conflict with one another.

When the scheduler does not grant n_i unit of resource R_i to job requesting them, the lock request of that job $L(R_i, n_i)$ fails; and the job is blocked. When the scheduler grant n_i unit of R_i for which the job is waiting, the job is unblocked and moved back to ready queue.

* Effect of Resource contention and Resource Access control :-

A resource access control protocol is a set of rules that govern-

1) When and under what conditions each request for resource is granted.
2) How job requiring resources are scheduled.

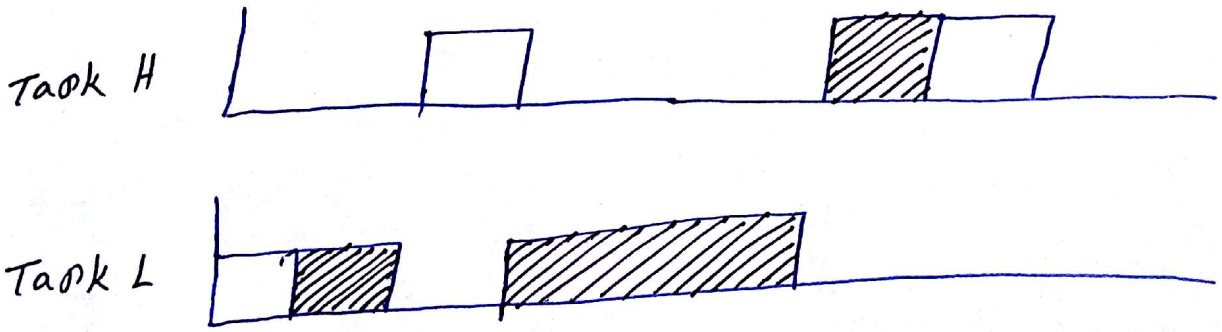
① Priority Inversion :-

A high priority task is ready to execute, but a lower priority task continues execution because it holds a lock on a shared resource that the higher priority task needs. It is an undesirable situation in which a higher priority task gets blocked.

There are two types of priority inversion -

(i) Bounded Priority Inversion
(ii) Unbounded priority Inversion.

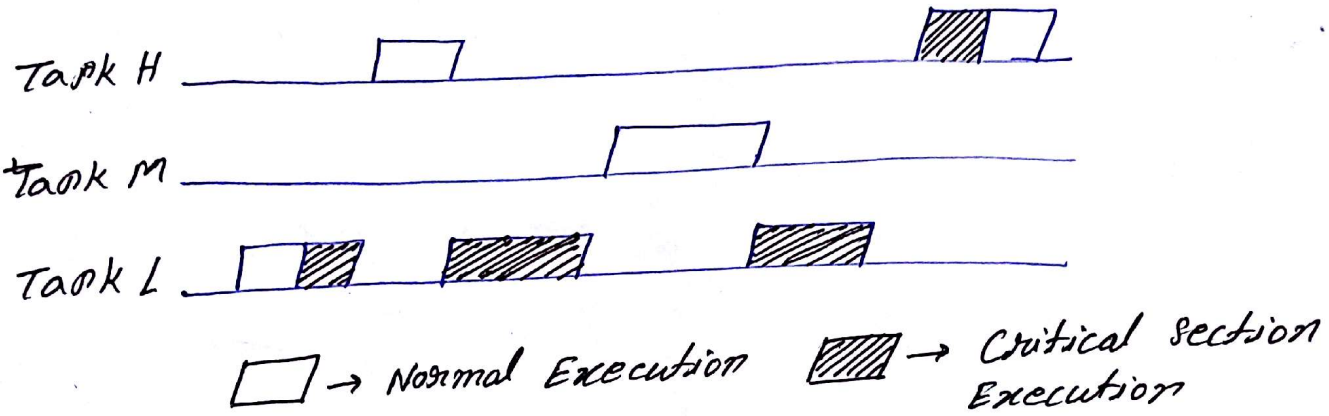
In bounded priority inversion, when low priority task L acquires a lock on a shared resource, but before releasing the shared resource it is preempted by high priority task H. Task H attempts to acquire the resource but is forced to wait for task L to finish its critical section. Task L continues running and after its completion, Task H acquires the resource and resumes execution. The worst case wait time for Task H is equal to the length of the critical section of task L.



→ Normal Execution

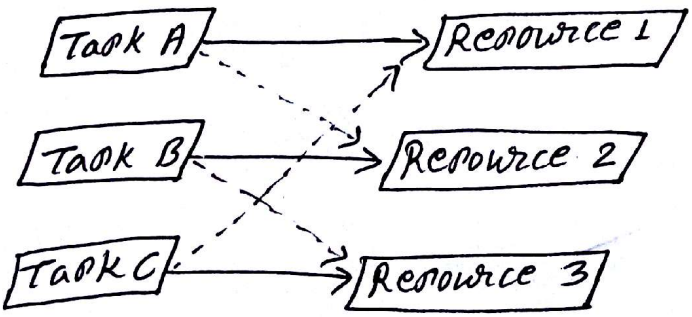
→ Critical section execution

In unbounded priority inversion, when a medium priority task M preempts task L during the execution of Task L's critical section. Task M runs until it relinquished control of the processor. only when Task M turns over control, then task L finish executing its critical section and release the shared resource; then Task H acquire the resource and resume execution. The worst case wait time for task H is is now equal to sum of the execution times of Task M and the critical section of Task L.



② Deadlock :-

Deadlock is a special case of nested resource locks, in which a circular chain of tasks waiting for resources prevents all the tasks from executing.



Suppose Task A is waiting for a resource held by Task B, while Task B is waiting for a resource held by Task C, which is waiting for resource held by Task A. None of the three tasks is able to acquire the resource, this situation is in deadlock.

* Non-preemptive critical section Protocal :-

The simplest way to control access of resources is to schedule all the critical section on the processor non-preemptively. When a job request a resource it's always allocated the resource and when a job hold a resource, it executes at the highest priority among all jobs. This protocol is called the Non-preemptive critical section (NPCS) protocol.

In this protocol, if J_i holds a resource it effectively has the highest priority in the system, i.e. it will not be preempted while holding a resource. So no deadlock.

Advantage : Bounds delays and prevents deadlock.

Dis-advantage : Every job can be blocked by a lower priority job.

* Basic Priority Inheritance Protocol :-

Priority inheritance protocol solves the problem of priority inversion. Under this protocol, if a higher priority task T^H is blocked by a lower priority task T^L, because T^L is currently executing critical section needed by T^H, ~~it temporar~~ T^L temporarily inherits the priority of T^H. When blocking ceases, i.e. T^L exits the critical section, T^L resumes its original priority.

Rules :-

① Scheduling Rule :-

a) Every ready jobs are scheduled on the processor preemptively in a priority driven manner according to their current priorities.

b) At release time t, the current priority $\pi(t)$ of every job is equal to its assigned priority. The job remain at this priority except under the condition stated in rule ③.

② Allocation Rule :-

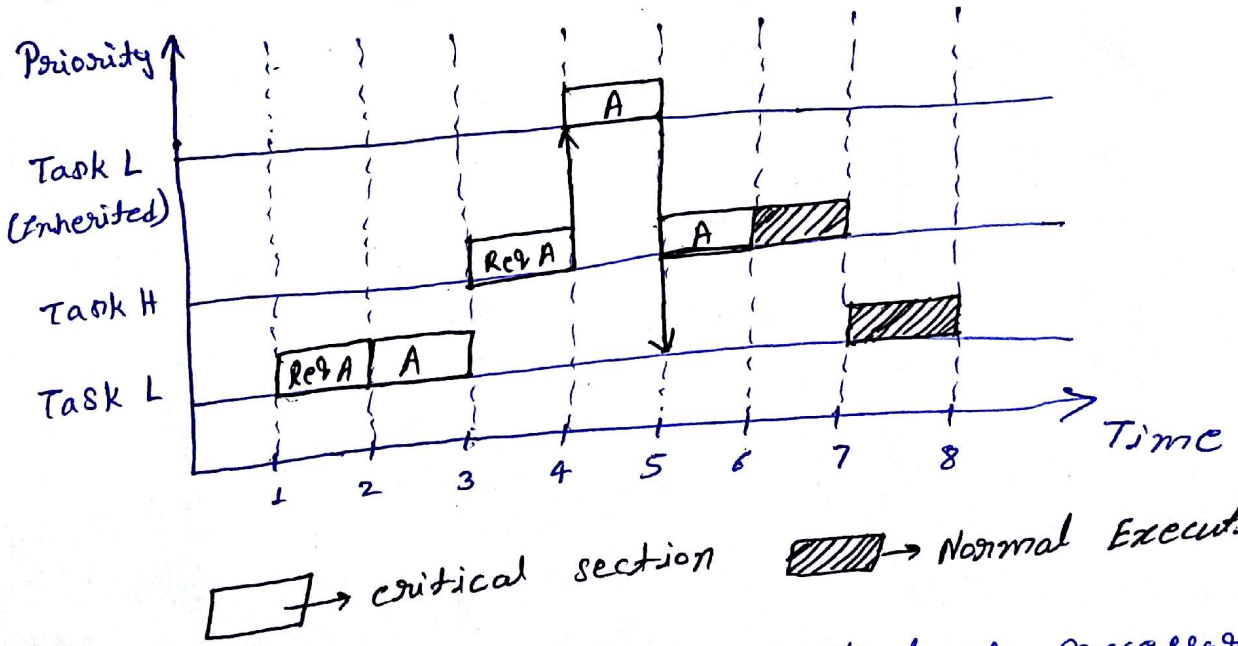
When a job request R at time t –

a) If R is free, R is allocated to job J.

b) If R is not free, the request is denied and J is blocked.

③ Priority Inheritance Rule :-

When the job J_1 becomes blocked, the job J_2 which blocks job J_1, inherits the current priority $\pi_1(t)$ of job J_1 and job J_2 executing at its inherited priority until it release resource. At that time t' the priority of job J_2 return back to its priority $\pi_2(t')$.



⇒ At time $t=1$, Task L receives control of processor and begins executing and makes a request for resource A.

⇒ At $t=2$, Task L is granted ownership of resource A and enters its critical region.

⇒ at $t=3$, Task L is preempted by Task H, a higher priority task. Task H begins executing and requests for resource A, which is owned by Task L.

⇒ at $t=4$, task L inherits the priority of Task H and resumes its critical section.

⇒ at $t=5$, Task L releases resource A and is lowered back to its original priority. Task H acquires resource A and begins executing its critical section.

⇒ At $t=6$, Task H releases resource A and continues executing normally.

⇒ At $t=7$, Task H finishes executing and Task L continues executing normally.

⇒ At $t=8$, Task L finishes executing.

⇓

Example: Refer from text book.

---

* Priority Ceiling Protocal :-

Priority Ceiling Protocal solves the priority inversion problem, without getting into deadlock. When a task T_i attempts to execute one of its critical sections, it will be suspended unless its priority is higher than the priority ceiling of all semaphores currently locked by tasks other than T_i.

If task T_i is unable to enter its critical section for this reason, the task that holds the locks on the semaphore with the highest priority ceiling is said to be blocking T_i and hence inherits the priority of T_i.

As long as a task T_i is not attempting to enter one of its critical sections, it will preempt every task that has a lower priority.

Rules :-

1) Scheduling Rule :

(i) At its release time t, the current priority $\pi(t)$ of every jobs j is equal to its assigned priority. The job remain at this priority except condition in rule (3).

(ii) Every ready job j is scheduled preemptively in a priority-driven manner at its current priority $\pi(t)$.

2) Allocation Rule :

Whenever a job j request a resource R at time t, one of the following two condition occure :

(i) If R is held by another job, j's request fails and j becomes blocked.

(ii) R is free :

a) If j's priority $\pi(t)$ is higher than the current priority ceiling $\hat{\pi}(t)$, then R is allocated to j.

b) If j's priority $\pi(t)$ is not higher than the ceiling $\hat{\pi}(t)$ of the system, R is allocated to j only if the job j is holding the resources, otherwise, request is denied and j becomes blocked.

3) Priority-inheritance Rule :-

When j becomes blocked, the job J_i which blocks j, inherits the current priority $\pi(t)$ of job j. The job J_i executes at its inherited priority until it release resource. After that, job J_i returns to its priority $\pi_i(t)$, which is the priority when it has granted the resources.

Stack-Based Priority ceiling Protocal :-

It is simpler than the basic priority ceiling protocal. These protocal come from the stack sharing capability and to simplify the priority ceiling protocal.

The stack-based priority ceiling protocal is deadlock free. whenever a job is preempted, all the resources needed by the preempting job are free. once a job is preempted, it can only resume when it returns to the top of the stack. Worst case blocking time of this protocal is the same as for basic priority ceiling protocal.

Rules :-

1) Update Priority Ceiling :-
whenever all resources are free, the ceiling of the system is $\hat{\pi}(t) = \Omega$. The value of $\hat{\pi}(t)$ is updated whenever resource is allocated & freed.

2) Scheduling Rule :-
After a job is released, it is blocked from starting & execution until its assigned priority is higher than $\hat{\pi}(t)$.

3) Allocation Rule :-
whenever a job requests a resource, it is allocated the resource.

* Use of Priority-Ceiling Protocol in Dynamic-Priority Systems :-

In a dynamic-priority system, the priorities of the periodic tasks change with time while the resources required by each task remain constant. As a consequence, the priority ceilings of the resources may change with time; and we update the priority ceiling of each resource and the ceiling of the system each time task priority change.

As an example, the EDF schedule of two tasks $T_1 = (2, 0.9)$ and $T_2 = (5, 2.3)$. In its first two periods (i.e. from time 0 to 4), T_1 has priority 1 while T_2 has priority 2; but from time 4 to 5, T_2 has priority 1 and T_1 has priority 2. Suppose that the task T_1 required a resource x while T_2 does not. The priority ceiling of x is 1 from time 0 to 4 and becomes 2 from time 4 to 5, and so on.

* Preemption Ceiling Protocol
* Access Control in multiple-Unit Resources
* Controlling Concurrent Accesses to Data objects

Please Refer from text book.

—— * ——