UNIT – 5
Resources and Resource Access Control

1. Define Critical section.
   A segment of a job that begins at a lock and ends at a matching unlock is called a ***critical section***. Resources are released in the last-in-first-out order. Hence overlapping critical sections are properly nested.A critical section that is not included in other critical sections is an ***outermost critical section.***

2. Write about Resource Conflicts and Blocking.
   Two jobs ***conflict*** with one another, or have a ***resource conflict***, if some of the resources they require are of the same type. The jobs ***contend*** for a resource when one job requests a resource that the other job already has. The scheduler always denies a request if there are not enough free units of the resource to satisfy the request. A scheduler may deny a request even when the requested resource units are free.

3. What is Resource access-control protocol?
   A ***resource access-control protocol***, or simply an ***access-control protocol***, is a set of rules that govern
   (1) when and under what conditions each request for resource is granted and
   (2) how jobs requiring resources are scheduled.

4. When Priority Inversion can occur?
   Priority inversion can occur when the execution of some jobs or portions of jobs is nonpreemptable. Resource contentions among jobs can also cause priority inversion. Because resources are allocated to jobs on a nonpreemptive basis, a higher-priority job can be blocked by a lower-priority job if the jobs conflict, even when the execution of both jobs is preemptable.

5. What is Direct blocking?
   A higher-priority job J_h is said to be ***directly blocked*** by a lower-priority job J_l when J_l holds some resource which J_h requests and is not allocated. In the example in Figure 8–2, J_3 directly blocks J_2 at time 5. We describe the dynamic-blocking relationship among jobs using a wait-for graph.

6. Write about Wait-for graph.
   In the ***wait-for graph*** of a system, every job that requires some resource is represented by a vertex labeled by the name of the job. There is also a vertex for every resource in the system, labeled by the name and the number of units of the resource. At any time, the wait-for graph contains an (ownership) edge with label *x* from a resource vertex to a job vertex if *x* units of the resource are allocated to the job at the time. There is a (wait-for) edge with label *y* from a job vertex to a resource vertex if the job requested *y* units of the resource earlier and the request was denied. In other words, the job is waiting for the resource. So, a path in a wait-for graph from a higher-priority job to a lower priority job

represents the fact that the former is directly blocked by the latter. A cyclic path in a wait-for graph indicates a deadlock.

7. Define the Basic Priority-Inheritance Protocol

In the definition of this protocol, we call the priority that is assigned to a job according to the scheduling algorithm its ***assigned priority***. At any time t, each ready job J_l is scheduled and executes at its ***current priority*** $\pi_l(t)$, which may differ from its assigned priority and may vary with time.

In particular, the current priority $\pi_l(t)$ of a job J_l may be raised to the higher priority $\pi_h(t)$ of another job J_h . When this happens, we say that the lower-priority job J_l ***inherits*** the priority of the higher priority job J_h and that J_l executes at its ***inherited priority*** $\pi_h(t)$.

The ***priority-inheritance protocol*** is defined by the following rules.

Rules of the Basic Priority-Inheritance Protocol

1. *Scheduling Rule*: Ready jobs are scheduled on the processor preemptively in a priority-driven manner according to their current priorities. At its release time t, the current priority $\pi(t)$ of every job J is equal to its assigned priority. The job remains at this priority except under the condition stated in rule 3.
2. *Allocation Rule*: When a job J requests a resource R at time t,
   (a) if R is free, R is allocated to J until J releases the resource, and
   (b) if R is not free, the request is denied and J is blocked.
3. *Priority-Inheritance Rule*: When the requesting job J becomes blocked, the job J_l which blocks J inherits the current priority $\pi(t)$ of J. The job J_l executes at its inherited priority $\pi(t)$ until it releases R; at that time, the priority of J_l returns to its priority $\pi_l(t')$ at the time t' when it acquires the resource R.

8. Define the Basic Priority-Ceiling Protocol.

The ***priority-ceiling protocol*** extends the priority-inheritance protocol to prevent deadlocks and to further reduce the blocking time. This protocol makes two key assumptions:

1. The assigned priorities of all jobs are fixed.

2. The resources required by all jobs are known a priori before the execution of any job begins.

We now define the priority-ceiling protocol for the case when there is only 1 unit of every resource.

Rules of Basic Priority-Ceiling Protocol

1. *Scheduling Rule*:
   (a) At its release time t, the current priority $\pi(t)$ of every job J is equal to its assigned priority. The job remains at this priority except under the condition stated in rule 3.
   (b) Every ready job J is scheduled preemptively and in a priority-driven manner at its current priority $\pi(t)$.
2. *Allocation Rule*: Whenever a job J requests a resource R at time t, one of the following two conditions occurs:
   (a) R is held by another job. J's request fails and J becomes blocked.
   (b) R is free.
      (i) If J's priority $\pi(t)$ is higher than the current priority ceiling $\hat{\Pi}(t)$, R is allocated to J.

(ii) 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 J is the job holding the resource(s) whose priority ceiling is equal to $\hat{\Pi}(t)$; otherwise, J's request is denied, and J becomes blocked.

3. *Priority-Inheritance Rule*: When J becomes blocked, the job J_l which blocks J inherits the current priority $\pi(t)$ of J. J_l executes at its inherited priority until the time when it releases every resource whose priority ceiling is equal to or higher than $\pi(t)$; at that time, the priority of J_l returns to its priority $\pi_l(t')$ at the time t' when it was granted the resource(s).

9. List the rules for defining the Basic Stack-Based, Priority-Ceiling Protocol.

Rules Defining Basic Stack-Based, Priority-Ceiling Protocol

0. *Update of the Current Ceiling*: Whenever all the resources are free, the ceiling of the system is Ω. The ceiling $\hat{\Pi}(t)$ is updated each time a resource is allocated or freed.
1. *Scheduling Rule*: After a job is released, it is blocked from starting execution until its assigned priority is higher than the current ceiling $\hat{\Pi}(t)$ of the system. At all times, jobs that are not blocked are scheduled on the processor in a priority-driven, preemptive manner according to their assigned priorities.
2. *Allocation Rule*: Whenever a job requests a resource, it is allocated the resource.

10. List the rules for defining the Ceiling-Priority Protocol

Rules Defining the Ceiling-Priority Protocol

1. *Scheduling Rule*:

(a) Every job executes at its assigned priority when it does not hold any resource. Jobs of the same priority are scheduled on the FIFO basis.

(b) The priority of each job holding any resource is equal to the highest of the priority ceilings of all resources held by the job.

2. *Allocation Rule*: Whenever a job requests a resource, it is allocated the resource.

11. List the rules of Basic Preemption-Ceiling Protocol

Rules of Basic Preemption-Ceiling Protocol

1 and 3. The *scheduling rule* (i.e., rule 1) and *priority inheritance rule* (i.e., rule 3) are the same as the corresponding rules of the priority-ceiling protocol.

2. *Allocation Rule*: Whenever a job J requests resource R at time t, one of the following two conditions occurs:
   (a) R is held by another job. J's request fails, and J becomes blocked.
   (b) R is free.
      (i) If J's preemption level $\psi(t)$ is higher than the current preemption ceiling $\hat{\Psi}(t)$ of the system, R is allocated to J.
      (ii) If J's preemption level $\psi(t)$ is not higher than the ceiling $\hat{\Psi}(t)$ of the system, R is allocated to J only if J is the job holding the resource(s) whose preemption ceiling is equal to $\hat{\Psi}(t)$; otherwise, J's request is denied, and J becomes blocked.

12. List the rules of Basic Stack-Based, Preemption-Ceiling Protocol

The stack-based preemption-ceiling protocol is called the Stack-Based Protocol (SBP). It is defined by the following rules.

Rules of Basic Stack-Based, Preemption-Ceiling Protocol

0. *Update of the Current Ceiling*: Whenever all the resources are free, the preemption ceiling of the system is Ω. The preemption ceiling $\hat{\Psi}(t)$ is updated each time a resource is allocated or freed.
1. *Scheduling Rule*: After a job is released, it is blocked from starting execution until its preemption level is higher than the current ceiling $\hat{\Psi}(t)$ of the system and the preemption level of the executing job. At any time t, jobs that are not blocked are scheduled on the processor in a priority-driven, preemptive manner according to their assigned priorities.
2. *Allocation Rule*: Whenever a job J requests for a resource R, it is allocated the resource.
3. *Priority-Inheritance Rule*: When some job is blocked from starting, the blocking job inherits the highest priority of all the blocked jobs.

13. List the Allocation rules of Multiple-Unit Priority-Ceiling Protocol.

Allocation Rule of Multiple-Unit Priority-(Preemption-) Ceiling Protocol

Whenever a job J requests k units of resource R at time t, one of the following two conditions occurs:

(a) Less than k units of R are free. J's request fails and J becomes directly blocked.
(b) k or more units of R are free.
  (i) If J's priority $\pi(t)$ [preemption level $\psi(t)$] is higher than the current priority ceiling $\hat{\Pi}(t)$ [preemption ceiling $\hat{\Psi}(t)$] of the system at the time, k units of R are allocated to J until it releases them.
  (ii) If J's priority $\pi(t)$ [preemption level $\psi(t)$] is not higher than the system ceiling $\hat{\Pi}(t)$ [$\hat{\Psi}(t)$], k units of R are allocated to J only if J holds the resource(s) whose priority ceiling (preemption ceiling) is equal to $\hat{\Pi}(t)$ [$\hat{\Psi}(t)$]; otherwise, J's request is denied, and J becomes blocked.

14. Write about Priority-Inheritance Rule

Let us examine a system where there are 3 units of resource *R*, and there are four jobs, each requiring 1 unit of *R*. Suppose that at the time when the highest priority job J_1 requests a unit of *R*, all 3 units are held by the other three jobs. Now, all three lower-priority jobs block J_1. In this case, it is reasonable to let J_2 inherit J_1's priority until it releases its units of *R*. Indeed, an important special case is when a job can request and hold at most 1 unit of every resource at a time. In this case, the following priority-inheritance rule works well.

Priority-Inheritance Rule

When the requesting job *J* becomes blocked at *t* , the job with the highest priority among all the jobs holding the resource *R* that has the highest priority ceiling among all the resources inherits *J* 's priority until it releases its unit of *R*. In general, a job may request and hold arbitrary numbers of resources.

15. List the rules for Convex-Ceiling Protocol.

Rules of Convex-Ceiling Protocol

1. *Scheduling Rule*: At any time, jobs that are not suspended are scheduled on the processor in a preemptive, priority-driven manner. Upon its release, the current priority of every job J_i is its assigned priority π_i. It executes at this priority except when the inheritance rule is applied.
2. *Allocation Rule*: When a job J_i requests to access a data object R for the first time,
   (a) if J_i's priority is higher than the system ceiling $\hat{\Pi}(t)$, J is allowed to continue execution and access R;
   (b) if J_i's priority is not higher than $\hat{\Pi}(t)$,
      i. if $\hat{\Pi}(t)$ is equal to $\Pi(J_i, t)$, J_i is allowed to access R;
      ii. otherwise, J is suspended.
3. *Priority-Inheritance Rule*: When J_i becomes suspended, the job J_l whose priority-ceiling function is equal to the system ceiling at the time inherits the current priority $\pi_i(t)$ of J_i