Dead Locks

What is a Deadlock?

In a multiprogramming environment, several processes may compete for a finite number of resources

A process requests resources; if the resources are not available at that time, the process enters a waiting state

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 a deadlock

Deadlock can arise if four conditions hold simultaneously

• Mutual exclusion: only one process at a time can use a resource

• Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes

• No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task

• Circular wait: there exists a set {P0, P1, ..., Pn} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, ..., Pn–1is waiting for a resource that is held by Pn , and Pn is waiting for a resource that is held by P0

Types of resources

• Physical resources - printers, tape drives, memory space, and CPU cycles

• Logical resources - semaphores, mutex locks, and files

Each process utilizes a resource as follows:

• request

• use

• release

Resource-Allocation Graph

request edge – directed edge Pi→ Rj

assignment edge – directed edge Rj →Pi

If graph contains no cycles---no deadlock

If graph contains a cycle

• if only one instance per resource type, then deadlock

• if several instances per resource type, possibility of deadlock

A cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock

Methods for Handling Deadlocks

Ensure that the system will never enter a deadlock state:

• Deadlock prevention

• Deadlock avoidance

Deadlock prevention

• provides a set of methods to ensure that at least one of the necessary conditions cannot hold

Deadlock avoidance

• the OS be given additional information in advance concerning which resources a process will request and use

• with this additional knowledge, the OS can decide the request

• each process declare the maximum number of resources of each type that it may need

• dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition

• Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes

Deadlock prevention:

1) Mutual Exclusion

• at least one resource must be nonsharable

• Read-only files are a good example of a sharable resource

2) Hold and Wait - must guarantee that whenever a process requests a resource, it does not hold any other resources

i) Process to request all its resources before it begins execution

ii) An alternative protocol allows a process to request resources only when it has none.

• Example

• Consider a process that copies data from a DVD drive to a file on disk, sorts the file, and then prints the results to a printer

• In first method, the process must initially request the DVD drive, disk file, and printer

• In second method, initially only the DVD drive and disk file are requested

3) No Preemption

• If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released

• Preempted resources are added to the list of resources for which the process is waiting

• Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

4) Circular Wait

• Impose a total ordering of all resource types and to require that each process requests resources in an increasing order of enumeration

• Let R={R1, R2, ..., Rm} be the set of resource types

• a process can initially request any number of instances of a resource type say, R_i

• After that, the process can request instances of resource type R_j if

• and only if $F(R_j)>F(R_i)$

Safe State

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

• A safe state is not a deadlocked state

• A system is in a safe state only if there exists a safe sequence

• If a system is in safe state □ no deadlocks

• If a system is in unsafe state □ possibility of deadlock

• Avoidance □ ensure that a system will never enter an unsafe state.

Claim edge $P_i \rightarrow R_j$ indicated that process P_i may request resource R_j at some time in the future represented by a dashed line

Refer Bankers Algorithm

Deadlock Detection

• Allow system to enter deadlock state

• Detection algorithm

• Recovery scheme

Wait-for graph

• obtain this graph from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges

Refer Deadlock Detection Algorithm

Recovery From Deadlock

1) Process Termination

• Abort all deadlocked processes

• Abort one process at a time until the deadlock cycle is eliminated

• In which order should we choose to abort?

1. Priority of the process
2. How long process has computed, and how much longer to completion
3. Resources the process has used
4. Resources process needs to complete
5. How many processes will need to be terminated
6. Is process interactive or batch

2) Resource Preemption

• Selecting a victim – minimize cost

• Rollback – return to some safe state, restart process for that state

• Starvation – same process may always be picked as victim, include number of rollback in cost factor