Lecture 13 (Deadlock)

Resource - hardware device or piece of information.

Resources:

- preempted - taken from process owning it with no ill effects - memory
- not preempted - cannot be taken away from its current owner without potentially causing failure

Sequence of events required to use a resource:

1. Request for resource (Resource acquisition) - associate semaphore with each resource
2. Use of resource
3. Release of resource

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;
void process A(void) {
  down(&resource_1);
  down(&resource_2);
  use_both_resources();
  up(&resource 2);
  up(&resource 1);
}
//Semaphores to protect resources
```

Types of deadlock:

- Resource deadlock - each member of the set of deadlocked processes is waiting for a resource that is owned by a deadlocked process.
- Communication deadlock - anomaly of cooperation synchronization

Deadlock - situation where two processes are each waiting for the other to finish.

Deadlock: resource1→process1→resource2→process2→resource1

Set of **processes is deadlocked** if:

- Each process in the set is waiting for an event
- That event can only be caused by another process in this set

4 necessary conditions for deadlock:

1. Mutual exclusion - each resource is either currently assigned to exactly one process or is available
2. Hold & wait - processes currently holding resources that were granted earlier can request new resources
3. No-preemption - resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
4. Circular wait - there must be a circular list of two or more processes, each of which is waiting for a resource held by the next member of the chain.

If one of them is absent, no resource deadlock is possible.

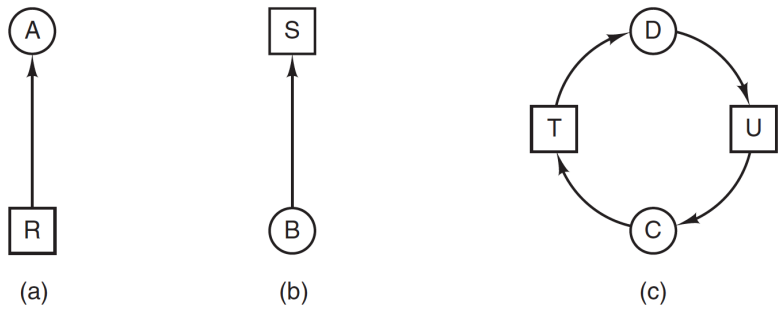

Figure 6-3. Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

4 strategies for dealing with deadlock:

1. Ignore
2. Detection & recovery (Let deadlocks occur, detect them, take action)
3. Dynamic avoidance by careful resource allocation
4. Prevention (by structurally negating one of the 4 required conditions)

Ignore

Ostrich algorithm - ignoring potential problems on the basis that they may be exceedingly rare

---

Deadlock detection & recovery

System does not attempt to prevent deadlocks from occurring. Instead, it lets them occur, tries to detect when this happens, and then takes some action to recover after the fact

Deadlock detection with one resource of each type

Construct a resource graph; if cycle exist ⇒ deadlock occur

Algorithm for detecting cycle:

1. For each node, N, in the graph, perform the following five steps with N as the starting node.
2. Initialize L to the empty list, and designate all the arcs as unmarked.
3. Add the current node to the end of L and check to see if the node now appears in L two times. If it does, the graph contains a cycle (listed in L) and the algorithm terminates.

Deadlock detection with multiple resources of each type

n - number of processes, m - number of types of resources

E = (e1, e2, ... ,em) - **existing resource vector** (total number of instances of each resource in existence)

A = (a1,a2,...,am) - **available resource vector** (giving the number of instances of resources that are currently available)

C nxm - **current allocation matrix;** Cij is the number of instances of resource j that are held by process i.

R nxm - **request matrix;** Rij is the number of instances of resource j that process i wants

$\sum_{i=1}^{n} C_{ij} + A_j = E_j$

Algorithm for detection:

1. Look for an unmarked process, Pi , for which the ith row of R is less than or equal to A.
2. If such a process is found, add the ith row of C to A, mark the process, and go back to step 1
3. If no such process exists, the algorithm terminates

If unmarked processes ⇒ deadlock occur between unmarked processes

Example:

E = (4 2 3 1) A = (2 1 0 0)

C: R:

0 0 1 0 2 0 0 1

2 0 0 1 1 0 1 0

0 1 2 0 2 1 0 0

1) p1: 2=2, 0<1, 0=0, 1<0!

2) p2: 1<2, 1<0!

3) p3: ok ⇒ A = (2 2 2 0) p3 = marked

4) p1: 2 = 2, 0<2, 0<2, 1<0!

5) p2: ok ⇒ A = (4 2 2 1) p2 = marked

6) p1: ok ⇒ A = (4 2 3 1) p1 = marked

No deadlocks.

3 possible methods of recovery:

1. Preemption - temporarily take away resource from assigned process and give it to another
2. Rollback - adding checkpoint for status of resources. **Checkpointing** a process - its state is written to a file so that it can be restarted later
3. Killing process - One possibility is to kill a process in the cycle. With a little luck, the other processes will be able to continueю Alternatively, a process not in the cycle can be chosen as the victim in order to release its resources.

In this approach, the process to be killed is carefully chosen because it is holding resources that some process in the cycle needs.

Deadlock avoidance

Safe & unsafe states. Safe state i - if there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately.

The system must be able to decide whether granting a resource is safe or not and make the allocation only when it is safe.

Banker's algorithm - The banker's algorithm considers each request as it occurs, seeing whether granting it leads to a safe state. If it does, the request is granted; otherwise, it is postponed until later. To see if a state is safe, the banker checks to see if he has enough resources to satisfy some customer.

Algorithm for multiple resources:

1. Look for a row, R, whose unmet resource needs are all smaller than or equal to A. If no such row exists, the system will eventually deadlock since no process can run to completion (assuming processes keep all resources until they exit).
2. Assume the process of the chosen row requests all the resources it needs (which is guaranteed to be possible) and finishes. Mark that process as terminated and add all of its resources to the A vector.
3. Repeat steps 1 and 2 until either all processes are marked terminated (in which case the initial state was safe) or no process is left whose resource needs can be met (in which case the system was not safe).

Example:

C:	R:
3 0 1 1	1 1 0 0
0 1 0 0	0 1 1 2
1 1 1 0	3 1 0 0
1 1 0 1	0 0 1 0

0 0 0 0 2 1 1 0

E = (6 3 4 2), A = (1 0 2 0)

1) p1: false

2) p2: false

3) p3: false

4) pr4: ok ⇒ A = (2 1 2 1) pr4 = marked

5) pr5: ok ⇒ A = (2 1 2 1) pr5 = marked

6) pr1: ok ⇒ A = (5 1 3 2) pr 1 = marked

7) pr2: ok ⇒ A = (5 2 3 2) pr 2 = marked

8) pr3: ok ⇒ A = (6 3 4 3) pr 3 = marked

All marked ⇒ state is safe.

Deadlock Prevention

One of necessary conditions are never satisfied.

1. Mutual exclusion → spool everything:

   Avoid assigning a resource unless absolutely necessary, and try to make sure that as few processes as possible may actually claim the resource.

2. Hold & wait → request all resources initially

   1. All processes should be required to request all their resources before starting execution. If everything is available, the process will be allocated whatever it needs and can run to completion
   2. Another approach is to require a process requesting a resource to first temporarily release all the resources it currently holds. Then it tries to get everything it needs all at once

3. No preemption → take resources away

   1. Take back resources that process uses (nearly impossible)
   2. Resources can be virtualized

4. Circular wait → order resource numerically

1. Rule: a process is entitled only to a single resource at any moment. If it needs a second one, it must release the first one
2. Provide a global numbering of all the resources. Every process is granted access to a resource with number that is greater than number of previously acquainted resources

Technique that can usually be employed to break communication deadlocks is **timeouts.**

Two-phase locking:

1. the process tries to lock all the records it needs, one at a time.
2. If it succeeds, it begins the second phase, performing its updates and releasing the locks. No real work is
   done in the first phase.

Livelock - process tries to be polite by giving up the locks it already acquired whenever it notices that it cannot obtain the next lock it needs

Starvation - situation in which processes are never getting service even if they are not deadlocked. Starvation can be avoided by using a first-come, first-served resource allocation policy. Process waiting the longest gets served next