Deedlock detection

Deedlock preveniton

Conditions for deadlocks

What are deadlocks?

Today: Deadlocks

It is possible to implement monitors with semaphores

Locks implemented by following the monitor rules for acquiring and releasing

C++ does not provide a monitor construct, but monitors can be

Condition variables release mutex temporarily

Monitor wraps operations with a mutex

Last Class: Monitors
Deadlock is a different condition from deadlock.

Other threads are actually using it (making progress).

Starvation occurs when a thread waits indefinitely for some resource, but
making progress and thus to recover.

Deadlock detection finds instances of deadlock when threads stop availlability to prevent deadlock.

Deadlock prevention algorithms check resource requests and possibly
resources simultaneously.

Deadlock can occur when several threads complete for a finite number of

Example:

Process B:

Process A:

disk->wait();
disk->wait();
prnter->wait();

Process B:

Process A:

disk->wait();
disk->wait();
prnter->wait();

A condition where two or more threads are waiting for an
Deadlock Detection Using a Resource Allocation Graph

1. If the graph has a cycle, deadlock might exist.
2. If the graph has no cycles, no deadlock exists.

Allocation: to e (Assignment Edge)
- A directed edge from a resource to a thread indicates that the OS has allocated e to t.
- A directed edge from a thread to a resource, t e \rightarrow f, indicates that t has requested

and threads \{t_1, \ldots, t_n\}.

We define a graph with vertices that represent both resources \{R_1, \ldots, R_m\}.

1. Mutual Exclusion: at least one thread must hold a resource in non-sharable mode, i.e., the resource may only be used by one thread at a time.
2. Hold and Wait: at least one thread holds a resource and is waiting for another thread to release the resource.
3. No Preemption: A thread can only release a resource voluntarily.
4. Circular Wait: A set of waiting threads \{t_1, \ldots, t_n\} where \( t_i \) is waiting on \( T_{i+1} \) and \( T_{i+1} \) is waiting on \( t_i \).

Deadlock can happen if all the following conditions hold.
Deadlock

- When CPU utilization drops below some threshold? (May take a long time to detect)
- On a regular schedule (hourly or ...)? (May take a long time to detect deadlock)
- Whenever a resource request can't be satisfied? (Each failed request is O(n^2))
- Just before granting a resource, check if granting it would lead to a cycle? (Each)

execute this algorithm?

Detecting cycles takes O(mn^2) time, where u is |U| + |T| + |R|.

- common in database transactions
- resource to the state it was in prior to getting the resource. This technique is
- preempt resource one at a time rolling back the state of the thread holding the
- Kill all the threads one at a time, forcing them to give up resource.
- Kill all the threads in the cycle

Different ways of breaking a cycle:

- Scan the resource allocation graph for cycles, and then break the cycles.
- Detect Deadlock and Then Correct It.

Deadlock Detection Using a Resource Allocation Graph

- then we can make progress when that resource is released.
- If any instances of a resource involved in the cycle is held by a thread not in the cycle.
- Then a cycle indicates only that deadlock might exist.
- What if there are multiple Interchangeable instances of a resource?
This algorithm ensures no circular-wait condition exists.

Currently available:
If the new state is unsafe, the thread must wait even if the resource is
Grant a resource to a thread if the new state is safe

resources they have deadlocked, since some threads might not actually use the maximum
An unsafe state is not equivalent to deadlock, it just may lead to
A safe state is a state in which there is a safe sequence for the threads.

available resource plus the resources held by all threads > k.
Define a sequence of threads a sequence of threads that can still be satisfied by the currently
Threads provide advance information about the maximum resources they

Deadlock Prevention with Resource Reservation

Deadlock Prevention

Order

1. Mutual Exclusion: Make resource sharable (but not all resources can be shared)

2. Hold and Wait: Prevent deadlock: ensure that at least one of the necessary conditions doesn’t hold.

3. No Preemption:

   OS preempts (release) all the resources that the thread is currently holding.
   If a thread requests a resource that cannot be immediately allocated to it, then the

4. Circular Wait: Impose an ordering (numbering) on the resources and request them in
   Problem: not all resources can be easily preemted like printers.

   Only when all of the resources are available will the OS release the thread.
There are now 0 available drives, but each thread might need at least one drive would lead to an unsafe state.

- If t3 requests one more drive, then it must wait because allocating the unallocated tape drive:
  - t3 can complete with all its current resources, all of t1 and t2's resources, and the threads may obtain their maximum number of resources without waiting.
  - The current state is safe (there exists a safe sequence, \{t1, t2, t3\} where all
- Currently, II drives are allocated to the threads, leaving I available.
  - II tape drives.
Lecture 10: Page 13

**Summary**

- Ignore the possibility! (Most OSs use this option!!)

- Deadlock over resource managed by the program, not OS resources.

- Code concurrent programs very carefully. This only helps prevent
  of the necessary conditions never holds

- Prevention: design resource allocation strategies that guarantee that one

- Avoidance: don’t allocate a resource if it would introduce a cycle.

- Break it.

- Detection and Recovery: recognize deadlock after it has occurred and

- Deadlock: situation in which a set of threads/processes cannot proceed

- ...