Today: Deadlocks

- Deadlock detection
- Deadlock prevention
- Conditions for deadlocks
- What are 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
Stratification is a different condition from deadlock.

Other threads are actually using it (making progress).

Stratification 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
available 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

---

**Terminology**

**Example:**

Deadlock: A condition where two or more threads are waiting for an
event that can only be generated by these same threads.
If the graph has a cycle, deadlock might exist.

If the graph has no cycles, no deadlock exists.

allocated (\( \langle \text{resource}, \text{thread} \rangle \)) to \( t \) (Assignment Edge)
- A directed edge from a resource to a thread \( t \) indicates that the OS has
  - A directed edge from a thread to a resource \( t \) \( \rightarrow \) \( t' \) indicates that \( t' \) has requested

\[ \{ u_1, u_2, \ldots, u_m \} \]

We define a graph with vertices that represent both resources \( \{ r_1, r_2, \ldots, r_n \} \)

Deadlock Detection Using a Resource Allocation Graph

on \( t \) \( \leq \) \( n \) and \( t \) is waiting on \( r \).

4. Circular wait: A set of waiting threads \( \{ u_1, u_2, \ldots, u_m \} \) where \( u_i \) is waiting

another thread or the OS cannot force the thread to release the resource.

3. No preemption: A thread can only release a resource voluntarily?

resource(s).

2. Hold and Wait: at least one thread holds a resource and is waiting for

Hold and Wait: at least one thread holds a resource and is waiting for

non-sharable mode, i.e., the resource may only be used by one thread at a

1. Mutual exclusion: at least one thread must hold a resource in

Deadlock can happen if all the following conditions hold.

Necessary Conditions for Deadlock
deadlock

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

execute this algorithm:

Detecting cycles takes O(n^2) time, where n is |T| + |R| + 1. When should we

common in database transactions:

resource to the set R was in prior to getting the resource. This technique is
prevent 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 resources.
- Kill all 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

when we can make progress when that resource is released.

- If any instance 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.
- Then a cycle indicates only that deadlock might exist.

- What if there are multiple interchangible instances of a resource?
This algorithm ensures no circular-wait condition exists.

- If the new state is unsafe, the thread must wait even if the resource is currently available.
- Grant a resource to a thread if the new state is safe resources they have declined.
  - Since some threads might not actually use the maximum deadlock, an unsafe state is not equivalent to deadlock, it just may lead to a safe state in which there is a safe sequence for the threads.
  - Available resources plus the resources held by all $t_i > t_j$.
  - Resources that $t_i$ can still request can be satisfied by the currently available sequence of threads $\{t_1, \ldots, t_n\} as safe if for each $t_i$, the threads provide advance information about the maximum resources they may need during execution.

Prevent deadlock: ensure that at least one of the necessary conditions doesn’t hold.
There are now 0 available drives, but each thread might need at least one drive would lead to an unsafe state.

If \( f_2 \) requests one more drive, then it must wait because allocating the

more drive.

\[
\begin{array}{|c|c|c|c|}
\hline
 & 7 & 5 & 12 \\
\hline
f_2 & 4 & 4 & 8 \\
\hline
f_3 & 3 & 4 & 1 \\
\hline
\text{max needed} & \text{in use} & \text{could want} & \\
\hline
\end{array}
\]

Example (contd)

\[
\begin{array}{|c|c|c|c|}
\hline
 & 8 & 4 & 12 \\
\hline
f_2 & 4 & 4 & 8 \\
\hline
f_3 & 3 & 4 & 1 \\
\hline
\text{max needed} & \text{in use} & \text{could want} & \\
\hline
\end{array}
\]

Example

- \( f_2 \) can complete with all its current resources, all of \( f_1 \) and \( f_3 \)'s resources, and the unallocated tape drive.
- \( f_3 \) can complete with all its current resources, plus all of \( f_1 \)'s resources, and the unallocated tape drive.
- \( f_1 \) can complete with the current resource allocation.

The threads may obtain their maximum number of resources without waiting.

The current state is safe (there exists a safe sequence, \( \{ f_1, f_2, f_3 \} \) where all \( f_1, f_2, f_3 \) are \text{contending for} 12 tape drives.

- Currently, 11 drives are allocated to the threads, leaving 1 available.
- Threads \( f_1, f_2 \), and \( f_3 \) are contending for 12 tape drives.
IEEE the possibility? Most OSes use this option!

- Avoidance: don't allocate a resource if it would introduce a cycle.
- Break it.
- Detection and recovery: recognize deadlock after it has occurred and because each requires resources held by another member of the set.
- Deadlock: situation in which a set of threads/processes cannot proceed.

Summary