Deadlock

Questions answered in this lecture:
What are the four necessary conditions for deadlock?
How can deadlock be prevented?
How can deadlock be avoided?
How can deadlock be detected and recovered from?

Deadlock

Deadlock Example
Two threads access two shared variables, A and B.
Variable A is protected by lock x, variable B by lock y.

How to add lock and unlock statements?

```c
int A, B;
lock_t x, y;
Thread 1
A += 10;
lock(x);
B += 20;
lock(y);
A += B;
unlock(y);
A += 30;
unlock(x);
Thread 2
B += 10;
lock(y);
A += 20;
lock(x);
A += B;
unlock(x);
B += 30;
unlock(y);
```

Deadlock: Why does it happen?

Informal: Every entity is waiting for resource held by another entity; none release until it gets what it is waiting for.

What can go wrong?
Representing Deadlock

Two common ways of representing deadlock
• Vertices:
  - Threads (or processes) in system
  - Resources (anything of value, including locks and semaphores)
• Edges: Indicate thread is waiting for the other

Wait-For Graph

```
T1 "waiting for" T2
```

Resource-Allocation Graph

```
T1 wants y held by T2
  ^
  |  x wants
  |  held by
T1
```

Conditions for Deadlock

Mutual exclusion
• Resource can not be shared
• Requests are delayed until resource is released

Hold-and-wait
• Thread holds one resource while waits for another

No preemption
• Resources are released voluntarily after completion

Circular wait
• Circular dependencies exist in "waits-for" or "resource-allocation" graphs

ALL four conditions MUST hold

Handing Deadlock

Deadlock prevention
• Ensure deadlock does not happen
• Ensure at least one of 4 conditions does not occur

Deadlock avoidance
• Ensure deadlock does not happen
• Use information about resource requests to dynamically avoid unsafe situations

Deadlock detection and recovery
• Allow deadlocks, but detect when occur
• Recover and continue

Ignore
• Easiest and most common approach

Deadlock Prevention #1

Approach
• Ensure 1 of 4 conditions cannot occur
• Negate each of the 4 conditions

No single approach is appropriate (or possible) for all circumstances

No mutual exclusion --> Make resource sharable
• Example: Read-only files
Deadlock Prevention #2

No Hold-and-wait -- Two possibilities
1) Only request resources when have none
   - Release resource before requesting next one

Thread 1
lock(x);
A += 10;
unlock(x);
lock(y);
B += 20;
unlock(y);
lock(x);
A += 30;
unlock(x);

Thread 2
lock(y);
B += 10;
unlock(y);
lock(x);
A += 20;
unlock(x);
lock(y);
B += 30;
unlock(y);

Deadlock Prevention #2

No Hold-and-wait
2) Atomically acquire all resources at once
   - Example #1: Single lock to protect all

Thread 1
lock(z);
A += 10;
B += 20;
A += B;
unlock(z);
A += 30;
unlock(z);

Thread 2
lock(z);
A += 10;
B += 20;
A += B;
A += 30;
unlock(z);

Deadlock Prevention #2

Problems w/ acquiring many resources atomically
   - Low resource utilization
     - Must make pessimistic assumptions about resource usage
     if (cond1) {
       lock(x);
     }
     if (cond2) {
       lock(y);
     }
   
   - Starvation
     - If need many resources, others might keep getting one of them
Deadlock Prevention #3

No "no preemption" --> Preempt resources
Example: A waiting for something held by B, then take resource away from B and give to A
- Only works for some resources (e.g., CPU and memory)
- Not possible if resource cannot be saved and restored
  - Can’t take away a lock without causing problems

Deadlock Prevention #4

No circular wait --> Impose ordering on resources
- Give all resources a ranking; must acquire highest ranked first
- How to change Example?

Deadlock Avoidance

Dijkstra’s Banker’s Algorithm
Avoid unsafe states of processes holding resources
- Unsafe states might lead to deadlock if processes make certain future requests
- When process requests resource, only give if doesn’t cause unsafe state
- Problem: Requires processes to specify all possible future resource demands

Banker’s Algorithm Example

Scenario:
- Three processes, P0, P1, and P2
- Five available resources, N=5
- Each process may need maximum of 4 resources simultaneously
Not safe example: P0 has 2, P1 has 1, P2 has 1
- Why could this lead to deadlock?
- Implication: Avoid this state, allow only states with enough resources left to satisfy claim of at least 1 process
- Claim: Maximum need - currently loaned to this process
Example:
- P0 requests: Allow?
- P1 requests: Allow?
- P2 requests: Allow?
- P0 requests: Allow?
- P1 requests: Allow?
- P0 requests: Allow?
- P0 releases 2
- Allow any others now?
Deadlock Detection and Recovery

Detection
- Maintain wait-for graph of requests
- Run algorithm looking for cycles
  - When should algorithm be run?

Recovery: Terminate deadlock
- Reboot system (Abort all processes)
- Abort all deadlocked processes
- Abort one process in cycle

Challenges
- How to take resource away from process? Undo effects of
  process (e.g., removing money from account)
  - Must roll-back state to safe state (checkpoint memory of job)
- Could starve process if repeatedly abort it