CS 31: Intro to Systems
Deadlock

Martin Gagne
Swarthmore College
April 25, 2017
What is Deadlock?

• Deadlock is a problem that can arise:
  – When processes compete for access to limited resources
  – When threads are incorrectly synchronized

• Definition:
  – Deadlock exists among a set of threads if every thread is waiting for an event that can be caused only by another thread in the set.
Dining Philosopher Problem

Example of incorrect solution:

- When a philosopher becomes hungry
  - take a fork as soon as one becomes available (left one if both available)
  - take a second fork as soon as it becomes available
  - eat
  - put back forks on the table

If all philosophers become hungry at the same time, and all take the left fork at the same time, then they all starve.

Deadlock!
Dining Philosopher Problem

Example of incorrect solution:

- When a philosopher becomes hungry
  - take a fork as soon as one becomes available (left one if both available)
  - take a second fork as soon as it becomes available
  - eat
  - put back forks on the table

Modifying the solution to just make the philosophers put down a fork for a while and try again later may lead to livelock.
What is Deadlock?

• Set of threads are permanently blocked
  – Unblocking of one relies on progress of another
  – But none can make progress!

• Example
  – Threads A and B
  – Resources X and Y
  – A holding X, waiting for Y
  – B holding Y, waiting for X
  – Each is waiting for the other; will wait forever
Four Conditions for Deadlock

1. Mutual Exclusion
   – Only one thread may use a resource at a time.

2. Hold-and-Wait
   – Thread holds resource while waiting for another.

3. No Preemption
   – Can’t take a resource away from a thread.

4. Circular Wait
   – The waiting threads form a cycle.
Four Conditions for Deadlock

1. Mutual Exclusion
   – Only one thread may use a resource at a time.

2. Hold-and-Wait
   – Thread holds resource while waiting for another.

3. No Preemption
   – Can’t take a resource away from a thread.

4. Circular Wait
   – The waiting threads form a cycle.
Examples of Deadlock

• Memory (a reusable resource)
  – total memory = 200KB
  – T₁ requests 80KB
  – T₂ requests 70KB
  – T₁ requests 60KB (wait)
  – T₂ requests 80KB (wait)

• Messages (a consumable resource)
  – T₁: receive M₂ from P₂
  – T₂: receive M₁ from P₁
Examples of Deadlock

Cars deadlocked in an intersection

Resource Allocation Graph
Banking, Revisited

```c
struct account {
    mutex lock;
    int balance;
}

Transfer(from_acct, to_acct, amt) {
    lock(from_acct.lock);
    lock(to_acct.lock)

    from_acct.balance -= amt;
    to_acct.balance += amt;

    unlock(to_acct.lock);
    unlock(from_acct.lock);
}
```
If multiple threads are executing this code, is there a race? Could a deadlock occur?

```
struct account {
    mutex lock;
    int balance;
}

Transfer(from_acct, to_acct, amt) {
    lock(from_acct.lock);
    lock(to_acct.lock)

    from_acct.balance -= amt;
    to_acct.balance += amt;

    unlock(to_acct.lock);
    unlock(from_acct.lock);
}
```

If there’s potential for a race/deadlock, what execution ordering will trigger it?

<table>
<thead>
<tr>
<th>Clicker Choice</th>
<th>Potential Race?</th>
<th>Potential Deadlock?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>B</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>C</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>D</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Common Deadlock

**Thread 0**

Transfer(acctA, acctB, 20);

Transfer(...) {
  lock(acctA.lock);
  lock(acctB.lock);
}

**Thread 1**

Transfer(acctB, acctA, 40);

Transfer(...) {
  lock(acctB.lock);
  lock(acctA.lock);
Common Deadlock

Thread 0
Transfer(acctA, acctB, 20);

Transfer(...) {
  lock(acctA.lock);
  \text{T}_0 \text{ gets to here}
  lock(acctB.lock);
}

Thread 1
Transfer(acctA, acctB, 40);

Transfer(...) {
  lock(acctB.lock);
  \text{T}_1 \text{ gets to here}
  lock(acctA.lock);
}

\text{T}_0 \text{ holds A’s lock, will make no progress until it can get B’s.}
\text{T}_1 \text{ holds B’s lock, will make no progress until it can get A’s.}
How to Attack the Deadlock Problem

• What should you/your OS do to help you?

• Deadlock Prevention
  – Make deadlock impossible by removing a condition

• Deadlock Avoidance
  – Avoid getting into situations that lead to deadlock

• Deadlock Detection
  – Don’t try to stop deadlocks
  – Rather, if they happen, detect and resolve
How to Attack the Deadlock Problem

• What should you/your OS do to help you?

• Deadlock Prevention
  – Make deadlock impossible by removing a condition

• Deadlock Avoidance
  – Avoid getting into situations that lead to deadlock

• Deadlock Detection
  – Don’t try to stop deadlocks
  – Rather, if they happen, detect and resolve
How Can We Prevent a Traffic Jam?

• Do intersections usually look like this one?

• We have road infrastructure (mechanisms)

• We have road rules (policies)

Cars deadlocked in an intersection
Suppose we add north/south stop signs. Which condition would that eliminate?

A. Mutual exclusion

B. Hold and wait

C. No preemption

D. Circular wait

E. More than one
Deadlock Prevention

• Simply prevent any single condition for deadlock

1. Mutual exclusion
   – Make all resources sharable (e.g. find max in which the threads have a return value instead of global max)

2. Hold-and-wait
   – Get all resources simultaneously (wait until all free)
   – Only request resources when it has none
     (e.g. having a waiter that says when the philosophers can grab forks)
Deadlock Prevention

• Simply prevent any single condition for deadlock

3. No preemption
   – Allow resources to be taken away (at any time)
     (e.g. have philosophers talk to each other and have conditions under which they give a fork)

4. Circular wait
   – Order all the resources, force ordered acquisition
     (e.g. associate each fork with a number, acquire forks in order)
Which of these conditions is easiest to give up to prevent deadlocks?

A. Mutual exclusion (make everything sharable)
B. Hold and wait (must get all resources at once)
C. No preemption (resources can be taken away)
D. Circular wait (total order on resource requests)
E. I’m not willing to give up any of these!

The best solution depends on the situation!
None may be practical.
How to Attack the Deadlock Problem

• Deadlock Prevention
  – Make deadlock impossible by removing a condition

• Deadlock Avoidance
  – Avoid getting into situations that lead to deadlock

• Deadlock Detection
  – Don’t try to stop deadlocks
  – Rather, if they happen, detect and resolve
Deadlock Avoidance

• Only allow resource acquisition if there is no way it could lead to deadlock.

• This is necessarily conservative, so there will be more waiting.

• We must know max resource usage in advance.
  • How could we know this and track it?
  • Depends on the resources involved.
How to Attack the Deadlock Problem

• Deadlock Prevention
  – Make deadlock impossible by removing a condition

• Deadlock Avoidance
  – Avoid getting into situations that lead to deadlock

• Deadlock Detection
  – Don’t try to stop deadlocks
  – Rather, if they happen, detect and resolve
Deadlock Detection and Recovery

• Do nothing special to prevent/avoid deadlocks
  – If they happen, they happen
  – Periodically, try to detect if a deadlock occurred
  – Do something to resolve it

• Reasoning
  – Deadlocks rarely happen (hopefully)
  – Cost of prevention or avoidance not worth it
  – Deal with them in special way (may be very costly)
Detecting a Deadlock

• Construct resource graph

• Requires
  – Identifying all resources
  – Tracking their use
  – Periodically running detection algorithm
Recovery from Deadlock

• Abort all deadlocked threads / processes
  – Will remove deadlock, but drastic and costly
Recovery from Deadlock

• Abort all deadlocked threads / processes
  – Will remove deadlock, but drastic and costly

• Abort deadlocked threads one-at-at-time
  – Do until deadlock goes away (need to detect)
  – What order should threads be aborted?
Recovery from Deadlock

• Preempt resources (force their release)
  – Need to select thread and resource to preempt
  – Need to rollback thread to previous state
  – Need to prevent starvation

• What about resources in inconsistent states
  – Such as files that are partially written?
  – Or interrupted message (e.g., file) transfers?
Which type of deadlock-handling scheme would you expect to see in a modern OS (Linux/Windows/OS X)?

A. Deadlock prevention

B. Deadlock avoidance

C. Deadlock detection/recovery

D. Something else
Which type of deadlock-handling scheme would you expect to see in a modern OS (Linux/Windows/OS X)?

A. Deadlock prevention
B. Deadlock avoidance
C. Deadlock detection/recovery
D. Something else

“Ostrich Algorithm”
How to Attack the Deadlock Problem

• Deadlock Prevention
  – Make deadlock impossible by removing a condition

• Deadlock Avoidance
  – Avoid getting into situations that lead to deadlock

• Deadlock Detection
  – Don’t try to stop deadlocks
  – Rather, if they happen, detect and resolve

• These all have major drawbacks…
Other Thread Complications

- Deadlock is not the only problem
- Performance: too much locking?
- Priority inversion
- ...
Priority Inversion

• Problem: Low priority thread holds lock, high priority thread waiting for lock.
  – What needs to happen: boost low priority thread so that it can finish, release the lock
  – What sometimes happens in practice: low priority thread not scheduled, can’t release lock

• Example: Mars Pathfinder (1997)
Sojourner Rover on Mars
Mars Rover

• Three periodic tasks:
  1. Low priority: collect meteorological data
  2. Medium priority: communicate with NASA
  3. High priority: data storage/movement

• Tasks 1 and 3 require exclusive access to a hardware bus to move data.
  – Bus protected by a mutex.
Mars Rover

• Failsafe timer (watchdog): if high priority task doesn’t complete in time, reboot system

• Observation: uh-oh, this thing seems to be rebooting a lot, we’re losing data…

JPL engineers later confessed that one or two system resets had occurred in their months of pre-flight testing. They had never been reproducible or explainable, and so the engineers, in a very human-nature response of denial, decided that they probably weren't important, using the rationale "it was probably caused by a hardware glitch".
What Happened: Priority Inversion

Low priority task, running happily.
What Happened: Priority Inversion

Low priority task acquires mutex lock.
What Happened: Priority Inversion

Medium task starts up, takes CPU.

Blocked
What Happened: Priority Inversion

High priority task tries to acquire mutex, can’t because it’s already held.
What Happened: Priority Inversion

High priority task tries to acquire mutex, can’t because it’s already held.

Low priority task can’t give up the lock because it can’t run - medium task trumps it.
What Happened: Priority Inversion

High priority is taking too long.
Reboot!
Solution: Priority Inheritance

High priority task tries to acquire mutex, can’t because it’s already held.

Give to blue red’s (higher) priority!
Solution: Priority Inheritance

High priority finishes in time.

Release lock, revert to low priority.
Deadlock Summary

• Deadlock occurs when threads are waiting on each other and cannot make progress.

• Deadlock requires four conditions:
  – Mutual exclusion, hold and wait, no resource preemption, circular wait

• Approaches to dealing with deadlock:
  – Ignore it – Living life on the edge (most common!)
  – Prevention – Make one of the four conditions impossible
  – Avoidance – Control allocation
  – Detection and Recovery – Look for a cycle, preempt/abort
That’s all the material for this course!

• But wait, didn’t I say I would do one more thing at the beginning of the course

• Let’s go there just for fun!
Pacman

- Pacman freaks out if you complete level 255
- Why?
Mars Pathfinder (1997)

• Frequently locked up and stopped responding
  – (automatic reboot)

• “Priority inversion” in parallel software
Pokémon Yellow

- Cleverly “hacked”,
  game completed in 1:36
- “Buffer overflow” exploit
Buffer Overflow

Two varieties:

• Buffer Overread

• Buffer Overwrite
Buffer Overflow

Two varieties:

• Buffer Overread
  – use buffer overflow to read more memory than should be available

• Buffer Overwrite
Heartbleed

HOW THE HEARTBLEED BUG WORKS:

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "POTATO" (6 LETTERS).

User Meg wants these 6 letters: POTATO.

Source: https://xkcd.com/1354/
Heartbleed

Source: https://xkcd.com/1354/
Heartbleed

Source: https://xkcd.com/1354/
Buffer Overflow

Two varieties:

• **Buffer Overread**
  – use buffer overflow to read more memory than should be available

• **Buffer Overwrite**
  – use buffer overflow to write to places you shouldn’t normally be allowed (where?)