SE350: Operating Systems

Lecture 5: Deadlock
Topics

• What is deadlock?
• What are the necessary conditions for deadlock?
• How can we prevent deadlock?
• How can we detect deadlock and recover from it?
Definitions

• Resource: any (passive) entity needed by a thread to do its job (CPU, disk space, memory, lock)
  • Preemptable: can be taken away by OS
  • Non-preemptable: must leave with thread

• Starvation: thread waits indefinitely

• Deadlock: circular waiting for resources
  • Deadlock leads to starvation, but not vice versa
Example: Two Locks

// Thread A
lock1.acquire();
lock2.acquire();
lock2.release();
lock1.release();

// Thread B
lock2.acquire();
lock1.acquire();
lock1.release();
lock2.release();

Deadlock won't always happen with this code, but it might
Bidirectional Bounded Buffer

// Thread A
buffer1.put(data);
buffer1.put(data);
buffer2.get();
buffer2.get();

// Thread B
buffer2.put(data);
buffer2.put(data);
buffer1.get();
buffer1.get();

Deadlock could happen if buffer1 and buffer2 both start almost full.
Two Locks and a Condition Variable

// Thread A
lock1.acquire();
... 
lock2.acquire();
... 
while (need to wait)
    condition.wait(lock2);
... 
lock2.release();
... 
lock1.release();

// Thread B
lock1.acquire();
... 
lock2.acquire();
... 
condition.signal(lock2);
... 
lock2.release();
... 
lock1.release();
Two Locks and a Condition Variable (cont.)
Dining Philosophers Politicians!

- Each politician needs two chopsticks to eat
- Each grabs chopstick on the right first
- Deadlock if all grab chopstick at same time
- Deadlock depends on the order of execution
  - No deadlock if one was left-handed
Necessary Conditions for Deadlock

- **Limited resources**
  - Finite num. of threads can simultaneously use a resource

- **No preemption**
  - Thread’s resource ownership cannot be revoked

- **Multiple independent requests** (wait while holding)
  - Thread holds a resource while waiting for another
  - First acquire one resources then try to acquire the other

- **Circular chain** of requests
  - Each thread is waiting for a resource hold by another
Questions

- How does Dining Politicians meet the necessary conditions for deadlock?
  - Limited resources
  - No preemption
  - Multiple independent requests (wait while holding)
  - Circular chain of requests

- How can we modify this to prevent deadlock?
  - Have an infinite pool of chopsticks
  - Take chopstick away from politicians if in deadlock
  - Grab both chopsticks at once or neither
  - Put one left handed politician
Preventing Deadlock

- Exploit or limit program behavior
  - Limit program from anything that might lead to deadlock
- Predict the future
  - If we know what program will do, we can tell if granting a resource might lead to deadlock
- Detect and recover
  - If we can rollback threads, we can fix deadlock if it occurs
Exploit or Limit Program Behavior

• Provide enough resources
• Preempt resources
• Eliminate “wait while holding”
• Eliminate circular waiting
Provide Enough Resources

• Question: How many chopsticks are enough?
  • One additional chopstick anywhere on the table!
Eliminate Wait While Holding

- Release lock when calling out of module

```cpp
Module::foo() {
    lock.acquire();
    doSomeStuff();
    otherModule->bar();
    lock.release();
}

Module::doSomeStuff() {
    x = x + 1;
}
```

```cpp
Module::foo() {
    doSomeStuff();
    otherModule->bar();
}

Module::doSomeStuff() {
    lock.acquire();
    x = x + 1;
    lock.release();
}
```
Eliminate Circular Waiting

• Lock ordering: always acquire locks in a fixed order

• Question: Can we use resource ordering to eliminate deadlock in dining politicians?
  • Number chopsticks from 1 to N
  • Pick lower-numbered chopstick before higher-numbered
Example

// Thread 1
Acquire A
Acquire C
Wait for B

// Thread 2
Acquire B
Wait for A

Have to stall here – otherwise, we're doomed!

Preventing deadlock means waiting even when the resource you are asking for is available, if some of the resources you will (or may) need are not available.
Deadlock Dynamics

- **Safe state:**
  - For any possible sequence of resource requests, there is at least one processing order that eventually succeeds.
  - May require waiting even when resources are available!

- **Unsafe state:**
  - At least one sequence of future resource requests leads to deadlock no matter what processing order is tried.

- **Doomed state:**
  - All possible processing orders lead to deadlock.
Possible System States

- Process can be in a safe, unsafe, or deadlocked state
Question

• What are the doomed states for Dining Politicians?
  • Each politicians holds one chopstick

• What are the safe states?
  • All the other states

• What are the unsafe states?
  • Only the doomed state
Communal Dining Politicians

- N chopsticks in middle of table;
- N politicians, each needs 2 chopsticks and can grab 1 at a time
- When does grabbing a chopstick lead to deadlock?
  - It's the last one, and no one has 2 chopsticks
Communal Mutant Dining Politicians

• N chopsticks in the middle of the table
• N politicians, each needs $k > 2$ chopsticks and can grab 1 at a time
• When does grabbing a chopstick lead to deadlock?
  • It's the last one, and no one would have $k$
  • It's the next to the last, and no one would have $k-1$
  • ...

...
Preventing Deadlock

• Exploit or limit program behavior
  • Limit program from anything that might lead to deadlock

• Predict the future
  • If we know what program will do, we can tell if granting a resource might lead to deadlock

• Detect and recover
  • If we can rollback threads, we can fix deadlock if it occurs
Banker’s Algorithm

- Grant request iff result is a safe state
- Sum of maximum resource needs of current threads can be greater than the total resources
  - Provided there is some way for all the threads to finish without getting into deadlock
- Example: proceed iff
  - total available resources - # allocated >= max remaining that might be needed by this thread in order to finish
  - Guarantees this thread can finish
Detect and Repair

- Scan wait graph, detect cycles, fix cycles
  - Terminate a thread to free up resources
    - Not always possible, e.g., could lead to inconsistent state
  - Proceed without the resource
    - Requires robust exception handling code
    - E.g., Amazon will say you can buy a book, if the inventory subsystem doesn’t reply quickly enough (wrong answer quickly is better than the right answer slowly)
- Roll back actions of deadlocked threads
  - Common technique in databases
  - Transaction allow rollbacks, restart to beginning of transaction
Acknowledgment

- This lecture is a slightly modified version of the one prepared by Tom Anderson.