Shared Memory Concurrency 3

CS 62 - Spring 2016
Michael Bannister

Some slides based on those from Dan Grossman, U. of Washington

Assignments 09

- Weekly Assignment: Operating System Simulator
  - Design due 4/3 (this Sunday)
  - Full assignment due: 4/10 (next Sunday)
  - First partner assignment
  - Open design (no starter code!)
  - **Use** Eclipse and the Java library!

Deadlock

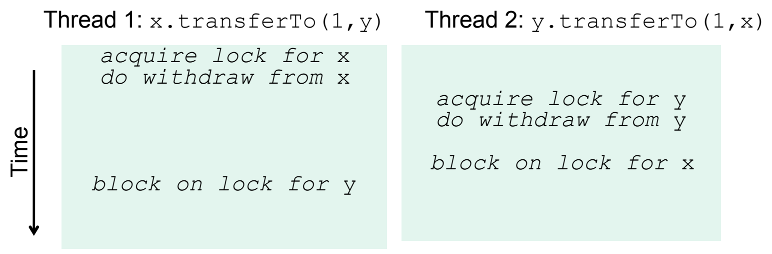
Deadlock

```
class BankAccount {
   ...
   synchronized void withdraw(int amt) {...}
   synchronized void deposit(int amt) {...}
   synchronized void transferTo(int amt, BankAccount a) {
      this.withdraw(amt);
      a.deposit(amt);
   }
}
```

- What locks are held at a.deposit(amt)?
- Is this a problem?

Deadlock

- Suppose have separate threads, each transferring to each others' account



Thread 1: `x.transferTo(1,y)`

```
acquire lock for x
do withdraw from x


block on lock for y
```

Thread 2: `y.transferTo(1,x)`

```
acquire lock for y
do withdraw from y

block on lock for x
```

Deadlock

- A deadlock occurs when there are threads $T_1, \dots, T_n$ such that:
  - For i=1,..,n-1, T_i is waiting for a resource held by T_{i+1}
  - T_n is waiting for a resource held by T_1
- In other words, there is a cycle of waiting
  - Formalize as a graph of dependencies with cycles bad
- Deadlock avoidance in programming amounts to techniques to ensure a cycle can never arise

A Last Example

- Bounded buffer is a queue with a fixed size.
  - Like event queue
  - Implemented in an array that wraps around.
- Producer threads do work and enqueue result
- Consumer threads dequeue results and perform work on them.
- Must synchronize access to the queue.

Attempt 1

```
class Buffer<E> {
  E[] array = (E[])new Object[SIZE];
  ... // front, back fields, isEmpty, isFull methods
  synchronized void enqueue(E elt) {
    if(isFull())
      ???
    else
      ... add to array and adjust back ...
  }
  synchronized E dequeue() {
    if(isEmpty()) {
      ???
    else
      ... take from array and adjust front ...
  }
}
```

Waiting

- enqueue to full buffer should not raise exception
  - Wait until there is room
- dequeue from empty buffer should not raise exception
  - Wait until there is data
- Bad approach is “spin lock”

What we want ...

- Thread should wait until has needed resources
  - Release lock and wait to be notified
- Needs operating systems support
- “Condition variable” that informs waiters when conditions have changed.
- See BoundedBuffer.java
  - uses “this” as condition variable

Concurrency Summary

- Access to shared resources introduces new kinds of bugs
  - Data races
  - Deadlocks
- Requires synchronization
  - Locks for mutual exclusion
  - Condition variables for signaling others
- Guidelines for use help avoid common pitfalls
- Getting shared-memory correct is hard!
  - But other models (e.g., message passing) not a panacea

Sleep

See sleep code example.