

## The Final Answer (I hope!)

```
public class ProducerConsumer {
    private boolean valueReady = false;
    private Object value;
    private Object fullLock=new Object(), emptyLock=new Object();

    void produce(Object o) { // no longer synchronized method
        synchronized (emptyLock) {
            while (valueReady) {
                emptyLock.wait();
            }
            synchronized (this) { value = o; valueReady = true; }
            synchronized (fullLock) { fullLock.notify(); }
        }
    }
}
```

CMSC 433, Fall 2002 - Michael Hicks

64

## The Final Answer (I hope!)

```
Object consume() { // no longer synchronized method
    synchronized (fullLock) {
        while (!valueReady) {
            fullLock.wait();
        }
    }
    Object o;
    synchronized (this) {
        valueReady = false;
        o = value;
        value = null;
    }
    synchronized (emptyLock) { emptyLock.notify(); }
    return o;
}
}
```

CMSC 433, Fall 2002 - Michael Hicks

65

## notify() vs. notifyAll()

- Very tricky to use notify() correctly
  - notifyAll() generally much safer
- To use notify() correctly, should:
  - have all waiters be equal
    - each notify only needs to wake up 1 thread
    - doesn't matter which thread it is
  - handle exceptions correctly

CMSC 433, Fall 2002 - Michael Hicks

66

## Thread Cancellation

- Example scenarios: want to cancel thread
  - whose processing the user no longer needs (i.e. she has hit the “cancel” button)
  - that computes a partial result and other threads have encountered errors, ... etc.
- Java used to have support Thread.kill()
  - But it and Thread.stop() are deprecated
  - Use Thread.interrupt() instead

CMSC 433, Fall 2002 - Michael Hicks

67

## Why no Thread.kill()?

- What if the thread is holding a lock when it is killed? The system could
  - free the lock, but the datastructure it is protecting might be now inconsistent
  - keep the lock, but this could lead to deadlock
- A thread needs to perform its own cleanup
  - Use `InterruptedException` and `isInterrupted()` to discover when it should cancel

CMSC 433, Fall 2002 - Michael Hicks

68

## Cancellation Example

```
public class CancellableReader extends Thread {
    private FileInputStream dataFile;
    public void run() {
        try {
            while (!Thread.interrupted()) {
                try {
                    int c = dataFile.read();
                    if (c == -1) break;
                    else process(c);
                } catch (IOException ex) { break; }
            }
        } finally { // cleanup here }
    }
}
```

*This could acquire locks, be on a wait set, etc.*

*What if the thread is blocked on a lock or wait set, or sleeping when interrupted?*

CMSC 433, Fall 2002 - Michael Hicks

69

## InterruptedException

- Will be thrown if interrupted either while doing or attempting to do a wait, sleep, or join
  - But not when blocked (or blocking on) on a lock or I/O
- Must reset invariants before cancelling
  - E.g. closing file descriptors, notifying other waiters, etc.

CMSC 433, Fall 2002 - Michael Hicks

70

## InterruptedException Example

- Threads t1 and t2 are waiting
- Thread t3 performs a notify
  - thread t1 is selected
- Before t1 can acquire lock, t1 is interrupted
- t1's call to wait throws InterruptedException
  - t1 doesn't process notification
  - t2 doesn't wake up

CMSC 433, Fall 2002 - Michael Hicks

71

## Handling InterruptedException

```
synchronized (this) {  
    while (!ready) {  
        try { wait(); }  
        catch (InterruptedException e) {  
            // make shared state acceptable  
            notify();  
            // cancel processing  
            return;  
        }  
        // do whatever  
    }  
}
```

CMSC 433, Fall 2002 - Michael Hicks

72

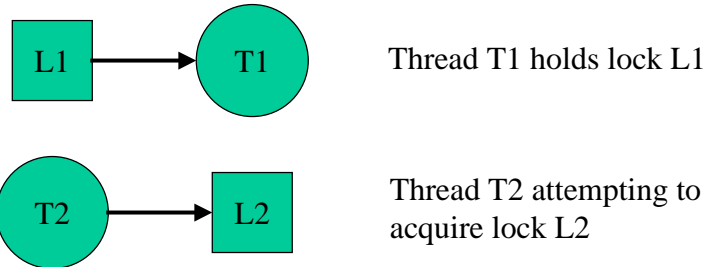
## Deadlock

- Quite possible to create code that deadlocks
  - Thread 1 holds lock on **A**
  - Thread 2 holds lock on **B**
  - Thread 1 is trying to acquire a lock on **B**
  - Thread 2 is trying to acquire a lock on **A**
  - Deadlock!
- Not easy to detect when deadlock has occurred
  - other than by the fact that nothing is happening

CMSC 433, Fall 2002 - Michael Hicks

73

## Deadlock: Wait graphs

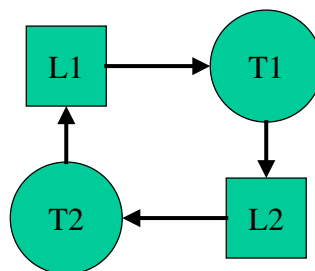


Deadlock occurs when there is a cycle in the graph

CMSC 433, Fall 2002 - Michael Hicks

74

## Wait graph example



T1 holds lock on **L1**  
T2 holds lock on **L2**  
T1 is trying to acquire a lock on **L2**  
T2 is trying to acquire a lock on **L1**

CMSC 433, Fall 2002 - Michael Hicks

75

## Livelock

- Deadlock arises when *blocked* threads cannot execute
- *Livelock* occurs when threads actually are executing, but no work gets done.
  - E.g. one thread keeps performing work that gets undone by another thread, so it must keep starting over

## Field Visibility

- Threads might cache values
- Obtaining a lock forces the thread to get fresh values
- Releasing a lock forces the thread to flush out all pending writes
- **volatile** variables are never cached
- **sleep(...)** doesn't force fresh values
- Many compilers don't perform these optimizations
  - but some do (Hotspot?)
- Problem might also occur with multiple CPUs

## Guidelines to simple/safe multi-threaded programming

- Synchronize access to shared data
- Don't hold a lock on more than one object at a time
  - could cause deadlock
- Hold a lock for as little time as possible
  - reduces blocking waiting for locks
- While holding a lock, don't call a method you don't understand
  - e.g., a method provided by someone else, especially if you can't be sure what it locks

CMSC 433, Fall 2002 - Michael Hicks

78

## Guidelines (cont.)

- Have to go beyond these guidelines for more complex situations
  - but need to understand threading and synchronization well
- We'll discuss threads more from the textbook *Concurrent Programming in Java* and from a talk at a Java conference by Bill Pugh and Doug Lea

CMSC 433, Fall 2002 - Michael Hicks

79

This document was created with Win2PDF available at <http://www.daneprairie.com>.  
The unregistered version of Win2PDF is for evaluation or non-commercial use only.