Advanced Concurrency Topics

Nelson Padua-Perez
Bill Pugh

Department of Computer Science
University of Maryland, College Park
public class Buffer {
    private LinkedList objects = new LinkedList();
    public synchronized add( Object x ) {
        objects.add(x);
        this.notifyAll();
    }
    public synchronized Object remove() {
        while (objects.isEmpty()) {
            try {
                this.wait();
            } catch (InterruptedException e) {}
        }
        return objects.removeFirst();
    }
}
Notes on wait and notifyAll

- Need to hold a lock on object on which you are going to wait/notifyAll
  - get an IllegalMonitorException otherwise
  - don’t catch it; this exception indicates coding error
- wait gives up the lock on the object waited on
  - no matter how many times you locked it
  - but no other locks held by that thread are given up
    - be scared of holding locks on multiple objects and using wait: easy to get deadlock
- always call wait in a loop
  - see next slide
wait idiom

synchronized(lock) {
    while (!ready)
        try { lock.wait() } catch (IE e) {
    // ready must have been true
    // and we held the lock when ready was true
    // and haven’t given it up
    TAKE ACTION;
} // release lock
What can do wrong?

```java
public class Buffer {
    private LinkedList objects = new LinkedList();
    public synchronized add( Object x ) {
        objects.add(x);
        this.notifyAll();
    }
    public synchronized Object remove() {
        if (objects.isEmpty()) {
            try {
                this.wait();
            } catch (InterruptedException e) {}
        }
        return objects.removeFirst();
    }
}
```
Sample Problem

Not easy (typical senior level quiz/midterm problem)

Bakery queue:

People show up in a bakery, takes the next available number, and receives a value in the order of their arrival

use Bakery algorithm to determine order in which consumers get items from Queue
Java Concurrency Utilities

- Lot of concurrency utilities were added in Java 1.5
- Blocking buffers
  - don’t write your own: someone else has already done it
- Exchanger
- CountDownLatch
- Semaphores
- Executors
  - thread pools
Exchanger\(<V>\)

- Allows threads to pair up to exchange object methods:
  - $V$ exchange($V$ v) - pairs up with another thread that wants to exchange a $V$
CountDownLatch

- Starts with some non-negative value
- Has several methods:
  - `getCount()` - returns current value
  - `countDown()` - decrements value
  - `await()` - waits until count reaches zero
Semaphore

Semaphore starts with some number of permits

Operations:

- acquire() - acquires one permit from the semaphore. If none are available, blocks until one can be returned
- release() - adds a permit to the semaphore
Executor

Very generic interface

One method:
  - void execute(Runnable work)

Several implementations:
  - ThreadPoolExecutor

Easy to define your own:

class DirectExecutor implements Executor {
  public void execute(Runnable r) {
    r.run();
  }
}
class ThreadPerTaskExecutor implements Executor {
    public void execute(Runnable r) {
        new Thread(r).start();
    }
}
Swing and Threads
//DON'T DO THIS!
while (isCursorBlinking()) {
    drawCursor();
    for (int i = 0; i < 300000; i++) {
        Math.sqrt((double)i); // this should really chew up some time
    }
    eraseCursor();
    for (int i = 0; i < 300000; i++) {
        Math.sqrt((double)i); // likewise
    }
}
Swing and Threads

- There is a Swing Thread
- All callbacks in response to events occur in the Swing thread
- Any modification, adjustment or reading of Swing components must be done within the Swing thread
- No operations that take longer than a few milliseconds should be done in the Swing thread
Avoiding synchronization improves performance.

Avoids the possibility that the Swing thread will be blocked by some other thread.
Doing work

- If, in response to a GUI event, you want to perform a task that might take a while such as saving a file or spell checking a document can’t do it in the event thread
- Ask another thread to do it
Better

final Runnable doUpdateCursor = new Runnable() {
    boolean shouldDraw = false;
    public void run() {
        if (shouldDraw) drawCursor(); else eraseCursor();
        shownDraw = !shouldDraw;
    }
};
Runnable doBlinkCursor = new Runnable() {
    public void run() {
        while (isCursorBlinking()) {
            EventQueue.invokeLater(doUpdateCursor);
            Thread.sleep(300);
        }
    }
};
new Thread(doBlinkCursor).start();
Recommended

Action updateCursorAction = new AbstractAction() {
    boolean shouldDraw = false;
    public void actionPerformed(ActionEvent e) {
        if (shouldDraw) drawCursor(); else eraseCursor();
        shownDraw = !shouldDraw;
    }
};

new Timer(300, updateCursorAction).start();