JavaOne℠

Sun's Worldwide Java Developer Conference

JavaOne℠

Sun's Worldwide Java Developer Conference

Multithreaded Applications

Thomas Ball
Staff Engineer
JavaSoft

Why Threads?

- Time wasted blocking on system calls
- Can’t interrupt calculations easily
- Difficult to manage multiple job tasks
- Difficult to use multiple processors

Threads...

- Are lightweight subprocesses
- Have separate execution paths
- Can share data
- Have a separate lifecycle
- Are scheduled

Java™ Threads

- Simplified model
  - Harder to hurt yourself
- Integrated in the Java™ language
- Platform-independent use
- System classes all thread-safe

Thread Basics

- Concurrent execution
- Scheduling
  - Preemption vs. cooperation
  - Time-slicing
- Priorities
  - wait(), sleep() and yield()

Thread Lifecycle

- New
  - Created, but not started
- Runnable
  - Started, but not necessarily running
- suspend() and resume()
- Dead
  - run() finished, or stop() called

Synchronization

- Prevents concurrent access to shared resources:

Teller A	**Teller B**
Gets balance	**Gets balance**
Adds $100	**Subtracts $100**
Stores balance	**Stores balance**

Balance equals?

- **The above is a “race condition”**

Monitors

- From Mesa/Cedar, C.F. Hoare
- Allows thread re-entry
- One per object with synchronized methods
- Usage automated through **`synchronized`** keyword
  - synchronized methods or blocks

Monitors (cont.)

```
public class Teller {
   int balance = 0;
   boolean smile = false;
   synchronized void credit(int cash) {
       balance += cash;
       smile = (balance > 10000);
   }
   synchronized void debit(int cash) {
       balance -= cash;
       smile = (balance > 10000);
   }
}
```

Monitors (cont.)

- Language integration encourages use
  - Little extra to learn
  - Compiler, runtime checking
- Language integration reduces bugs
  - Monitor guaranteed to be exited properly
  - No reference counting problems

Creating Threads

- Write the class with a run() method
  - Either subclass Thread or
  - Implement Runnable
- Create an instance
  - If subclass, use **`new <subclass>`** and call **`start()`**
  - If Runnable, use **`new Thread()`**

Killing Threads

- Exit the **run()** method, or
- Call **<thread obj>.stop()**
  - Won’t die while blocked

We’re Done, Right?

- But what about:
  - Inter-thread communication
  - Performance
  - Deadlocks
  - Race conditions
  - Native code access

Inter-thread Communication

- Roll-your-own events
- wait(), notify(), notifyAll()
- Synchronized objects

Example: SimpleQueue

```
class SimpleQueue {
  synchronized boolean nextEvent() {
      try {
           wait();
      } catch (InterruptedException e) {
           return false;
      }
      return true;
  }
}
```

Example: SimpleQueue Client

```
public static void main(String[] args) {
  SimpleQueue theQ = new SimpleQueue();
  MySubTask task = new MySubTask(theQ);
  task.start();
  while (theQ.nextEvent() {
     System.out.println(“got event”);
  }
}
```

Synchronized Objects

```
class StaticQueue {
  static theQ;
  static boolean nextEvent() {
     synchronized (theQ) {
        try {
           theQ.wait();
        }catch (InterruptedException e){
           return false;
        }
     }
     return true;
  }
}
```

Performance

- Synchronized method calls are slow:
  - Method call: 1.0
  - Synchronized method call: 6.6
  - Already-locked method call: 6.1
- So only synchronize methods which access shared instance or static data

Deadlocks

- Definition: two threads wait forever for the other to release a lock the other has locked
- Example: dining philosophers
- Diagnostic: thread and monitor dump
  - Control-\ on Solaris
  - Control-Pause on Win32
  - Debug menu on Mac

Thread Dump

```
"Screen Updater" (CONDVAR_WAIT) prio=5
  java.lang.Object.wait(Object.java:152)
  sun.awt.ScreenUpdater.nextEntry(where)
"AWT-Win32" (RUNNABLE) prio=5
  sun.awt.win32.MToolkit.run()
  java.lang.Thread.run(CONDVAR_WAIT)
"thread applet-TicTacToe.class" () prio=5
  java.lang.Object.wait()
  sun.applet.AppletPanel.getNextEvent()
```

Monitor Cache Dump

```
unknown key (key=0x75a0f0): <unowned>
sun.applet.AppletViewerPanel@1396FA0/
  143D140 (key=0x1396fa0): <unowned>
  waiters = 1
sun.awt.ScreenUpdater@1397630/143EBD8 (key=
  0x1397630): <unowned> waiters = 1
sun.awt.win32.MToolkit@1397250/143DB40
  (key=0x1397250): "AWT-Win32"
```

Race Conditions

- Symptoms:
  - Bugs on one platform, not on another
  - Bugs when run remotely
  - Bugs when run with debugger
    - Single-stepping may cause round-robining
- Any non-synchronized shared data can display a race condition

Race Conditions (cont.)

- Design/code review:
  - definitely synchronize static data access
  - probably synchronize method data access
- Don’t rely on thread priorities
  - I/O blocks thread simulators (Mac, etc.)
  - I/O doesn’t block native threads (Win32)

Native Code Access

- obj_monitor(Object)
- monitorEnter(), monitorExit()
  - **`monitorEnter(obj_monitor(Object))`**
- monitorNotify(), monitorNotifyAll()
- monitorWait()
  - **`monitorWait(obj_monitor(Object),100)`**

Native Access Code (cont.)

- Java™ code:

```
synchronized (myObj) {
        <block>
}
```

- in C:

```
monitorEnter(obj_monitor(myObj));
<block>
monitorExit(obj_monitor(myObj));
```

- Must call monitorExit() on all exits

Questions?