Lab #1: Introduction

Due: April 7, 2006 11:59pm

Overview

This lab is meant to familiarize you with some of the basic mechanisms (that are used in the implementation of the more advanced mechanisms) that we will use to implement later projects.

Part 1: Basic Threads

Write a program, `ThreadPrint`, that takes two command-line arguments. The first argument, `n`, specifies the number of threads to create and the second, `m`, specifies the number of iterations that each thread will execute. Your program must create `n` threads, each initialized with a unique "thread number" from 0 to $n-1$. Each thread will execute a loop that iterates `m` times. On each iteration of the loop, a thread will execute

```
System.out.println("Thread " + my_thread_number);
System.out.println(my_thread_number + ": Iteration " + i);
```

Part 2: Basic Synchronization

Write a program, `ThreadPrintSynch`, that behaves as above, but that guarantees, using `synchronized`, that the two lines printed by a thread during a single iteration of its loop will print consecutively. This means that there will be no other output between the line specifying the thread number and the line specifying the iteration number.

Recall that the threads are separate objects and will need to share a common lock to enforce this synchronization.

Part 3: More Synchronization

Write a program, `ThreadPrintRobin`, that behaves as in part 1, but that ensures a round robin scheduling of the threads. Specifically, thread 0 will execute one iteration of the loop, followed by thread 1 executing a single iteration, and so on. Once thread $n-1$ has executed a single iteration, then the cycle continues with thread 0 executing its second iteration of the loop.

Part 4: Bounded Buffer

Generalize the `OneSlotBuffer` example from lecture into a more common `BoundedBuffer`. A `BoundedBuffer` allows concurrent inserts and deletes from a fixed size (set at construction) buffer in a queue-like manner (i.e., first-in/first-out behavior). If a thread attempts to remove an item from an empty buffer, then it must block until an item becomes available. Likewise, an attempt to insert into a full buffer results in blocking until space becomes available.

The Java class `ArrayBlockingQueue` implements this behavior. You cannot use `ArrayBlockingQueue` or any other class that provides similar behavior "for free". You may use a "regular" Java collection class (or array) to represent the buffer, but you must write all necessary synchronization code. If this seems an artificial restriction, then assume that you are on the Java 5 developement team and have been given the task to implement `ArrayBlockingQueue` in `java.util.concurrent` (though I am only asking that you implement a small subset of the behavior).

Part 5: Basic Sockets

For this part you will write two programs. Both programs will connect to a website and echo the contents of a remote file to the console (the second program is actually more general in that it will work with any url).

`SocketWeb`

The first progra takes three command-line arguments. The first argument specifies the name of the server (e.g., `www.csc.calpoly.edu`). The second argument is the port to connect to on the remote machine (typically port 80 for a web server). The last argument specifies the file to access, which may be a path (e.g., `index.html`).

This program should connect to the server at the specified port and request the specified file. To request a file from a web server, your client program will need to send an HTTP GET message. The format of such a message is

```
GET <filename> HTTP/1.0
```

For example, to request `index.html` from the server your client application would first send the message (followed by a blank line)

```
GET /index.html HTTP/1.0
```

After sending the `GET` request, your client program then reads the response from the server and prints it to the screen.

`URLWeb`

The second program takes a single command-line argument that specifies an entire URL (e.g., http://www.csc.calpoly.edu/index.html). It then uses the java `URL` and `URLConnection` classes to open a connection to the host and retrieve the resource. Again, the response is to be printed to the screen.

Handin

Submit your source files using handin on hornet/falcon. To do this, execute

handin akeen lab1 <your files here>

Grading will be divided as follows.

Part	Percentage
1	20
2	10
3	20
4	20
5	30