Project 3
CMSC 433
Programming Language Technologies and Paradigms
Fall 2002

Due Friday, November 15, 2002 at 6pm

Introduction

In project 2 you constructed a generic server. That server accepted connections one at a time and then invoked handlers one at a time. This approach works perfectly well in many situations. For example, it works well when there are a moderate numbers of client requests and all handlers execute quickly. That is because in this situation clients will tend to receive service quickly and predictably.

However, this kind of server may perform poorly in other situations. For instance, imagine a server that has at least one unresponsive handler. When that handler slows down or blocks, all other clients will also be blocked – even if the blocked clients don’t use the unresponsive handler. This is like when you get in line to order just a soda at McDonald’s and it turns out that the guy in front of you is ordering for his entire office!

Another issue arises when there are many client requests. Here, clients may have to wait a long time even if the individual jobs are small. This is something like driving on a tollway with only one toll booth during a holiday weekend. Even though it doesn’t take too long for any single car to pay the toll, the aggregate time accumulates dramatically when there are a lot of cars on the road.

These issues arise in project 2 because BasicServer is designed to handle client requests sequentially. In this project we will introduce the idea of concurrent programs (via Java Threads) as a way to improve program performance. In particular, rather than having the server process each request one at a time, we will have each request processed in its own thread, in parallel. In terms of our informal examples above, this is like having many cashiers at McDonald’s or many toll booths on the tollway; clients can still make progress when there is one slow client or many clients in aggregate.

This project has two main goals:

- To familiarize you with concurrent programming via Java Threads.
- To explore the engineering trade-offs that come with developing and using concurrent programs.
In this assignment you will create 3 applications: SeqPageProxy, ThreadedPageProxy, and BoundedThreadedPageProxy. We will test your applications with several experiments. You can also get this assignment in PDF and postscript. You will want to reuse some or all of your code from Project 2. If you wish, you can use our implementation of project 2 rather than your own; we will send out our implementation to all students via e-mail.

1 SeqPageProxy

SeqPageProxy will be an application composed of a BasicServer and some handlers; we describe each part in turn.

1.1 Handlers

Some of these handlers you have already implemented in project 2, while the remainder you will have to implement at this time. Note that you will need to consider the possible error conditions that could arise with the new handlers; if something erroneous occurs, send back a suitable message to the client via HttpServletResponse.

- **StartSessionHandler.** Matches “startSession/*” and is exclusive. Client requests are of the form http://Server:Port/startSession/key. The StartSessionHandler constructor has the following form:

```java
public StartSessionHandler(java.util.Map sessionMap,
                          HandlerOrder orderConstraint)
```

This handler is designed to start a timer that will be stopped by a later request, so as to time how long it takes to process many requests. To do this, StartSessionHandler will need to associate a Timer object with the given key in the provided sessionMap.

- **EndSessionHandler.** Matches “endSession/*”, and is exclusive. Client requests are of the form: http://Server:Port/endSession/key/log. The EndSessionHandler constructor has the following form:

```java
public EndSessionHandler(java.util.Map sessionMap,
                         java.util.Map logMap,
                         HandlerOrder orderConstraint)
```

The EndSessionHandler stops the timer that was started by StartSessionHandler for the given key; it gets the Timer object by looking it up in the provided sessionMap. When the timer is stopped, it will write an event to the log with the given name, indicating the name of the session that completed and the total elapsed time. The log is acquired by looking up log in the given logMap; if a log does not exist, then a new LocalLog is created of size 500 and stored in the map.
• StartTimerHandler. StartTimerHandler works just like it did in Project 2, having the following constructor:

```java
public StartTimerHandler(Timer timer, String pattern,
HandlerOrder orderConstraint)
```

• EndTimerHandler. EndTimerHandler works just like it did in Project 2, having the following constructor:

```java
public EndTimerHandler(Timer timer, String pattern, Log log,
HandlerOrder orderConstraint)
```

• RemoteGetFileHandler. RemoteGetFileHandler works just like it did in Project 2, having the constructor:

```java
public RemoteGetFileHandler(HandlerOrder orderConstraint)
```

• SummaryHandler. Matches "summary/.*", and is exclusive. Client requests are of the form: `http://Server:Port/summary/log`. Its constructor has the form:

```java
public SummaryHandler(java.util.Map logMap,
HandlerOrder orderConstraint)
```

SummaryHandler calculates the minimum, maximum, and median times of all the EndTimerHandler records stored in `log`; the `Log` object is acquired by looking up the name `log` in the provided `logMap`. It also calculates the same statistics for all EndSessionHandler records stored in that `log`.

Output should be formatted as an HTML table in two columns, with the EndTimerHandler statistics first, followed by the EndSessionHandler statistics. Here is an example that you should follow:

```html
<table>
<tr><th>Request Times</th></tr>
<tr><td>Minimum</td><td>0.056 s</td></tr>
<tr><td>Maximum</td><td>3.56 s</td></tr>
<tr><td>Median</td><td>1.04 s</td></tr>
<tr><th>Session Times</th></tr>
<tr><td>Minimum</td><td>34.5 s</td></tr>
<tr><td>Maximum</td><td>34.5 s</td></tr>
<tr><td>Median</td><td>34.5 s</td></tr>
</table>
```
1.2 The Application

To build the application, you will create the handlers as follows; unless otherwise noted, they should have order constraint NO_PREFERENCE:

1. Create two empty Maps, one for the log map, and one for the session map. In the log map, store a LocalLog of size 500, associated with key “mylog”. The session map will be empty. Create a StartSessionHandler and an EndSessionHandler, passing the log map and session map to each as necessary.

2. Create a StartTimerHandler and an EndTimerHandler with a shared Timer, having pattern remoteGetFile/.*. Pass to the EndTimerHandler the same LocalLog as was stored in the log map above. The StartTimerHandler should have order constraint EARLY and the EndTimerHandler should have order constraint LATE.

3. Create a SummaryHandler, passing it the log map created above.

4. Create a RemoteGetFileHandler.

All applications should accept the -Dport=XXX flag to set the port to use to listen for connections.

1.3 Experiment 1

Using your code, we will run two experiments. The first experiment will start a session, then make a number of requests of your server, one at a time. When the requests complete, it will end the session and invoke the summary handler to see the results.

The script to run this experiment is called exp1. To run it, you first start a slow server to which your RemoteGetFileHandler requests will be directed. The slow server is implemented in the file SlowServer.java. Then, assuming you have started the slow server on port p and you have started your SeqPageProxy on port q, from the same machine you run the script by doing

```bash
exp1 q p
```

The script performs all of requests to your server using wget, requesting remote files from the slow server. The final results will be printed to the screen.

1.4 Experiment 2

The second experiment is like the first, except that it makes all of its requests at once, rather than one at a time. However, because your server is sequential, this will not necessarily help the waiting times of requests, except that some shorter requests might (randomly) sneak in before the larger ones.

The script implementing this experiment is exp2. It is invoked exactly as exp1, described above.
2 ThreadedServer

The second application will be similar to the SeqPageServer except that the Server will be threaded. To do this you should create a ThreadedServer class that has the same interface as BasicServer:

```java
public class ThreadedServer {
    public ThreadedServer(int port);
    public void attach (HttpHandler handler);
    public void detach (HttpHandler handler);
    public void run ();
}
```

The difference will be that it will process each client request in a newly spawned thread, so that the request handlers and the main loop can operate in parallel.

The handlers you use in this application will be the same as those in SeqPageServer (from the client’s point of view.) However, you must re-code each handler to be thread-safe. A handler is thread-safe if it properly deals with the fact that its state could be shared, either by a different handler in another concurrently running thread, or another copy of the same handler in another thread. You must synchronize on data that could be shared between threads, as well as account for state that must be duplicated to allow the same handler to run multiple times concurrently. For example, there is now the possibility that multiple copies of the same StartTimerHandler and EndTimerHandler will run in different threads simultaneously; what effect will this have on the shared Timer? You will also have to be concerned with Log and Map objects that are shared between threads. There may be other concerns as well.

To simplify the grading process, we require that rather than change the code for the handlers you have, make the thread-safe versions as separate classes:

- StartSessionSyncHandler. Synchronized version of StartSessionHandler.
- EndSessionSyncHandler. Synchronized version of EndSessionHandler.
- StartTimerSyncHandler. Synchronized version of StartTimerHandler.
- EndTimerSyncHandler. Synchronized version of EndTimerHandler.
- SummarySyncHandler. Synchronized version of SummaryHandler.

You may have to create thread-safe versions of other classes as well; you may change any class that you need to, as long as the constructors to the handlers are as specified above. Make these new versions separate classes as well, rather than modifying the original ones. To keep things simple, you may wish to keep your code from problem 1 separate from the code in this problem, in a different subdirectory (but be sure to copy it all to the same directory before you submit it). Finally, depending on how you implement your synchronization, you may
not need to change some or even all of your handler classes; nonetheless, make a separate copy having the new name, as specified above.

Finally, create a version of SeqPageProxy, called ThreadedPageProxy, that uses the ThreadedServer and the thread-safe versions of the handlers (and possibly other classes).

2.1 Experiment 3
We will re-run experiment 2, using the script exp2. However, you will use ThreadedPageProxy as your server rather than SeqPageProxy. Now that the server is multi-threaded, the shorter jobs should complete sooner, despite the presence of longer ones. The differences will be shown by the summary handler.

3 BoundedThreadedServer
This application will be the same as the ThreadedServer except that the Server will only allow \( k \) client request Threads to run at any one time (make \( k \) settable by a command-line argument if you wish, but have it default to 10). The general idea is sketched out below in pseudocode (this is not Java code!):

```java
while (true) {
    if (max # of threads running) {
        wait until one completes
    } else {
        accept connection
        process request
        ....
    }
}
```

The number of threads \( k \) should be settable by a user of BoundedThreadedServer:

```java
public class BoundedThreadedServer {
    public BoundedThreadedServer(int port, int maxThreads);
    public void attach (HttpHandler handler);
    public void detach (HttpHandler handler);
    public void run ();
}
```

3.1 Experiment 4
Create a version of ThreadedPageProxy, called BoundedThreadedPageProxy, that uses the BoundedThreadedServer rather than the ThreadedServer. You should create a command-line option for setting the number of threads, as follows:
java -DnumThreads=10 -Dport=13304 BoundedThreadedPageProxy

Here, numThreads indicates the number of threads to use (so BoundedThreadedPageProxy will use this value to set $k$ in BoundedThreadedServer) and we use the port argument as usual to specify the port on which to accept connections.

We will re-run experiment 3, again with script exp2, using this version of the server and observe how the performance differs, as shown by the summary handler.

4 Amendments

The project is now due on **November 15**, rather than November 13.

4.1 Testing

Here is what you must test:

- 2 tests for each new SyncHandler (see list above).
- 2 new tests for each Server: ThreadedServer and BoundedThreadedServer

You can submit others for a little bit of extra credit if you wish.

Comments on the tests themselves:

- Handler tests – should have several threads run simultaneously. For example, create a Thread subclass that runs a startTimerHandler followed by an endTimerHandler on a particular HttpRequest object that you create. Then run two of these threads simultaneously, with each sharing the same Log (to write the elapsed time to). You can then check that both times were successfully written to the log.

- Server tests – handle several connections simultaneously, check that you don't exceed max number of threads, etc. To do the server tests, you should start up your server (and the slowserver if necessary), and then run the JUnit test. This test will connect to the server, send a request, and check that the output makes sense. To do this, use the URL and URLconnection classes, as we did for RemoteGetFileHandler. Be sure when you open the URLconnection you only get the input stream and not the output stream; doing the latter will result in an IOException being thrown. The RemoteGetFileHandler code is the right place to start.

In general, don't worry about checking exact output; since there will be timing variability, just check things that you know are invariant* such as the fact that a record should or should not be written to a Log, that an entry with a certain key should be in a Map, etc.
4.2 Cautions and Tips

- If you pass in synchronized versions of objects to the constructors of your handlers, please document this in the handler itself. That is, point out that for the handler to work in a synchronized setting, the objects it receives in its constructor must be synchronized. If you have constructed your handler so that this isn’t the case, you might mention that the handler will use objects in a synchronized way, whether or not the objects are internally synchronized.

- The ThreadedServer class does not implement Runnable, even though it has a run method. It was not intended for this to be run in multiple threads, but instead for it to spawn multiple threads to be implemented by another class.

- In general, do not change the interface of any class whose interface we have specified. You will lose points!

- When you download the scripts, be sure to make them executable or they won’t run. That is, do something like chmod 755 exp1.

- While you are encouraged to use the scripts we have provided for testing, you will probably want to run smaller tests on your own. In particular, take a look at SlowServer to see how it works, and/or use some of the wget commands from the scripts individually.

4.3 Bugs

- We did not specify the constructor to BoundedThreadedServer; it should be BoundedThreadedServer(int port, int maxThreads);

- One of the calls to the “turtle” target in the exp2 script did not have a & after it, so it was not running in parallel. This has been fixed in the script on-line (please use this one).

- We have been inconsistent in how we have been referring to the servers and the applications. The server for the second part of the project is ThreadedServer, and its application is ThreadedPageProxy; for the third part is should be BoundedThreadedServer and BoundedThreadedPageProxy.

- The originally provided version of BasicServer.java was incorrect. Please grab the new version at http://www.cs.umd.edu/class/fall2002/cmsc433-0201/hw3/java/BasicServer.java.

- The originally provided version of SlowServer.java was incorrect, as it was not multi-threaded. Please grab the new version.

- We did not specify what command-line arguments to use. See Sections 1.2 and 3.1.
• The pattern for the StartTimerHandler and EndTimerHandler for the applications was incorrectly stated as .*; it should be remoteGetFile/.*; see Section 1.2.