Department of Computer Science
CS240 Operating Systems, Communications and Concurrency – Dermot Kelly
Practical Number 10

In practical 6 you implemented a version of a simple **client/server application** using a **java socket based** solution. A client communicated with a server to obtain the date.

In practical 7, 8 and 9 you implemented **multithreaded** simulations for the Producer Consumer problem, the Dining Philosophers problem and the Readers/Writers problem.

Using your code and experience from these practicals, **revisit** the client/server date application (practical 6 - socket based solution only – files `DateServer.java & DateClient.java`) and make the necessary alterations to make the server **multithreaded**. Create a p10 directory in your cs240 folder and copy DateServer.java and DateClient.java to it.

The server should be able to accept multiple connections from clients and return values for the date to the respective clients simultaneously. Note there is **no need to change the client code**, only the server code. You will need to change the `main()` method in the server code so that after accepting a client socket connection, instead of carrying on to process the request, it should create a new worker thread and the `run()` method of the new worker thread should contain the code for processing the client request and closing the connection. You will need to define a new class "`class WorkerThread extends Thread`" as part of the DateServer.java file. The `run()` method of the worker thread class should implement the service by sending the date to the client, instead of the `main()` method as was done in practical 6. The `main()` method should just listen and accept connections and spawn worker threads to service them.

The `main()` method should also keep count of the number of connections made and pass the number of each connection to the worker thread's constructor so that it can print a message like "`finished processing client n`" to its output stream after responding with the Date to the client.

You will need to pass each connected client socket object (returned from `sock.accept() in main()`) as a parameter to the worker thread's constructor, the header of the constructor might look like this:-

```
public WorkerThread(Socket client, int connectionNumber) {
```

You will need to create a `WorkerThread` class as mentioned above(based on the templates provided in earlier practicals). Obviously the behaviour of a `WorkerThread`, as defined in its `run()` method, needs to be modified to implement part of the functionality given in the `DateServer` class `main()` method from practical 6. Easiest to delete all the code in the `run()` method and constructor of a thread class created from one of the earlier labs and paste the required parts from the `main()`method of DateServer.java from practical 6 with some additional modifications.

Although the application functionality of this example is fairly simple, applications involving communication and multithreading are conceptually difficult for the programmer. You can use this reference framework for introducing you to java multithreaded servers and to help construct more complex distributed multithreaded code in the future.

SUBMIT ON MOODLE
A screenshot with
1) a long directory listing of your p10 directory.
2) an editor window showing your new WorkerThread class & run() method.
3) an editor window showing the modified DateServer main() method.
4) a shell window showing an example of the client/server interaction running.

This is your last assignment for this course. I hope you enjoyed the labs and learned some new things.

```java
// This is the Client Code from Practical 6, save as DateClient.java
import java.net.*;
import java.io.*;

public class DateSocketClient {
	public static void main(String[] args) throws IOException {
		try {

			System.out.println("Creating a socket connection to server on port 6013");
			Socket sock = new Socket("127.0.0.1",6013);
			InputStream in = sock.getInputStream();

			// bin is the buffered input stream from the server
			BufferedReader bin = new BufferedReader(new InputStreamReader(in));

			String line;
			System.out.println("Reading data from Server over socket connection");
			while ( (line = bin.readLine()) != null)
				System.out.println("The date received from the server was: "+line);
			sock.close();
		}
		catch (IOException ioe) {
			System.err.println(ioe);
		}
	}
}
```

This file does not need to be altered.

```
// This is the Server code from practical 6, save as DateServer.java
import java.net.*;
import java.io.*;

public class DateSocketServer {

	public static void main(String[] args) throws IOException {
		try {
			// This creates a listener socket
			ServerSocket sock = new ServerSocket(6013);
			while (true) {
				System.out.println("Server waiting for an incoming socket connection on port 
6013");

				Socket client = sock.accept();

				System.out.println("Connection accepted - sending response");
				// pout is the output stream to the client
				PrintWriter pout = new PrintWriter(client.getOutputStream(),true);
				// The response sent is the server's system date/time displayed as a string
				pout.println(new java.util.Date().toString());
				client.close();
			}
		}
		catch (IOException ioe) {
			System.err.println(ioe);
		}
	}
}
```

This code needs to be modified, the shaded area (servicing the incoming request) needs to be done in the run method of a thread instead. The main() method should keep count of incoming connections and create a WorkerThread to service each one instead.