Department of Computer Science
CS240 Operating Systems, Communications and Concurrency

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**. 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. The worker thread implements the service by sending the date to the client, instead of the `main()` method as is done in practical 6. The `main()` method just listens and accepts connections and spawns worker threads. You will need to pass the connected client socket object as a parameter to the worker thread's constructor.

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 to the client.

You will need to create a `WorkerThread` class (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 from practical 6 as described above. 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
One text file containing:-
The WorkerThread Class that you created
The modified DateServer Class which spawns WorkerThreads to handle client connections.

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