1. Purpose

This assignment implements a very simple multi-threaded server over sockets. The server will return the current date to the client in a multi-threaded manner. You will turn in two servers—one where you explicitly create and manage your own thread and another using a Java thread pool.

2. Description

See below for the single threaded Date Server which is shown in the Silberschatz OS book, Edition 10, figure 3.28.

```java
import java.net.*;
import java.io.*;
public class DateServer
{
  public static void main(String[] args)
   {
    try
      {
      ServerSocket sock = new ServerSocket(6013);

      /* now listen for connections */
      while (true)
          {
        Socket client = sock.accept();
            PrintWriter pout = new PrintWriter(client.getOutputStream(),
                        true);

        /* write the Date to the socket */
        pout.println(new java.util.Date().toString());

        /* close the socket and resume */
        /* listening for connections */
        client.close();
          }
    }
    catch (IOException ioe)
     {
      System.err.println(ioe);
    }
   }
}
```

For this assignment, create two multi-threaded servers: DateServerMT.java and DateServerMTP.java. For DateServerMT create a new thread on the server for each request. For the DateServerMTP utilize a Java-based thread pool from java.util.concurrent.Executors to run the client requests.

The Java Server should take the port that it will be listening on as an argument. When testing in the Linux lab you should use the last 5 digits of your student ID as a port so that it does not collide with other students.

Example Execution:
        Java DateServerMT.class 60013
This server will take requests on port 60013 and handle them in a multi-threaded manner.

---

3. Turn in

Turn in the following files: DateServerMT.java and DateServerMTP.java.

Please zip this in the one zip file with the files from the other part of this program.