CSE 322 (A2)
Offline Assignment 1

Topic: Client-server application development using Socket Programming
Task: Implementation of both client and server sides of an application simulating a simple prototype of cricinfo.

1. **Requirement Specification:**
In this assignment, you are required to build a prototype of cricinfo on client-server architecture. You are required to write two programs (in whatever language you prefer that supports socket programming): (i) the client and (ii) the server. The client program will enable remote users to see over by over information of a match simulated by the server. Specific requirements for the task are as follows.

- The server will simulate multiple T20 matches at a particular time. The simulation will be done in simpler way in an over by over manner. The server will generate possible outcomes in a random manner for six deliveries in an over. No extra deliveries (wide/no ball etc.) needs to be simulated. So, the possible outcome limits to **dot, 1, 2, 3, 4, 6 and out** for any delivery. Other rules of cricket are applicable as necessary. So, after each over the server will generate the following things. The outcome of six deliveries with total run and wicket after each over. At the end of the match, it will tell which team has won the match. A delay should introduced after end of each over.

- On startup, the client program will prompt the user for the server's 'IP Address' and 'Port Number' as well as the 'Student ID'. Then the client will initiate a connection request with the server and send the 'Student ID'.

- Upon receipt of the connection request from a client, the server will save the <IP Address, Student ID> mapping (in any suitable data structure). Then the server will create a folder named after the 'Student ID' and all the answers submitted through this connection will be saved under this folder. The 'Student IDs' are to be treated as numbers and should be within the range pre-configured in the server. If the 'Student ID' of an incoming request in not within the list, the server should notify the requesting client accordingly and ask him to send a valid ID. (*Optional and Bonus:* If two different Student IDs try to register from the same IP or same Student ID tries to register from two different IPs, the server admin will be explicitly asked, through a prompt, whether to accept or reject the connection request. In case of rejection, the requesting client will be notified accordingly).

- The server program must provide interface for the following configurable options:

(i) Set-up a match: A match is set up only by setting two strings as team names. In this way, multiple matches can be set up.

(ii) Number of Concurrent Matches: Number of concurrent matches a client can subscribe to.

(iii) Allowable Student IDs: List of Student IDs who are allowed to upload files.
IDs can be specified in range format such as 201305091-201305120 as well as comma separated individual IDs such as 200905100, 200805119.

- On login, the client is shown a list of the concurrent matches that are currently simulated by the server. The client chooses the matches he want to get information of from the server. But, the number cannot exceed the limit set by the server.
- The server sends a file to the client after the end of simulation of every over according to the matches. And in the end, the file should contain the name of the winning team.
- Every file should be segmented before transfer by the client where the segment size will be maximum 256 bytes. For example, if a file is 1000 bytes, it will be segmented in two chunks, the first three packets are 256 bytes and the last one is 1000-768=232 bytes. The client will send each segment according to the following format:

  File Name::Starting Byte Number::Size of Segment::File Data

  The client, upon receiving each segment, will send an acknowledgment to the client which will cause the server to send the next segment, i.e., each segment is to be individually acknowledged. (*Optional and Bonus*: If a server does not receive an acknowledgement for a sent segment, it will resend it after waiting for a predefined time interval).

- If a client machine crashes during the simulation of match(s), then upon restarting the client program and the resending the connection request to the server, the server will send the file containing the information starting from the overs after the crash was held.

2. **Programming Issues**

- You must use socket programming in your implementation, both for client and server.
- One of the primary objectives of this lab is to learn how to write message passing protocol. Therefore, all the prompts for user will be generated by client and client should know all the message formats. **You should not implement the client as a dumb terminal.**
- You may use any programming language you wish (Java, C#, Python) as long as it supports socket abstraction to access OS's native TCP service.
- Use object oriented programming. In that case, you will have at least two class files *client.class* and *server.class*.
- Use of GUI is desirable and recommended.
- Take care to handle exceptions. Unwanted action by client should not crash the server.

3. **Ethical Issues**

Since all of you will be doing the same assignment, experience tells us that there is high chance of copying. **Let us warn you that any case of plagiarism (copying) will be handled severely with nearly zero tolerance and may even result in suspension from the course irrespective of whether you were the server (source of code) or the client (who copied the code).**