CSE 322 (B1)
Offline Assignment 1

Topic: Client-server application development using Socket Programming
Task: Implementation of both client and server sides of a file uploader application

1. **Requirement Specification:**
   In this assignment, you are required to build a file uploader application 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 upload file to the server. Specific requirements for the task are as follows.
   - Clients will be able to upload file only according to the constraints specified by the server. The mandatory constraints to be implemented are as follows.
     (i) File Type: This will allow a client to upload file only type/s specified by the server, such as *.java, .py, .c* etc.
     (ii) Single file/ multiple files/ folder: whether client will be able to upload single or multiple files or folders with nested subfolders.
     (iii) Fixed file names: Clients will only be able to upload file with pre-specified file names such as *server.java, client.java* etc.
     (iv) Max file size: Maximum allowable file size that a client is able to upload.

   - 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). The the server will create a folder named after the 'Student ID' and all the files received 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) Root Directory: Location in the file system where the all the incoming files will be stored by the server, i.e., all the folders named after the 'Student IDs' of incoming requests as mentioned above, will be stored under this directory. By default, the root directory will be the location from which the server program is running.
  (ii) File types: Allowable file types that a client can upload.
  (iii) Number of files and folder: Number of files a client is allowed to upload and whether a client is able to upload folder.
  (iv) Max file size: Maximum file size that a client is allowed to upload.
  (v) Allowable Student IDs: List of Student IDs who are allowed to upload files. IDs can be specified in range format such as 201305001-20130560 as well as comma separated individual IDs such as 200905100, 200805119.

- On the client side, when a user wants to upload a file (by pressing some button in the client interface), the client will first request the server to send all the constraints configured. After receiving the constraints list, the client will then prompt the user to upload files accordingly and present with him a File Dialog box. If the files/folder specified by the user mismatch with the constraints, those will be rejected and a warning will be showed to the user.

- Every file should be segmented before transfer by the client where the segment size will be maximum 512 bytes. For example, if a file is 1000 bytes, it will be segmented in two chunks, the first one is 512 bytes and the next one is 1000-512=488 bytes. The client will send each segment according to the following format:

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

  (to upload folders, define your own suitable format)

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

- *Optional:* The server program, upon receipt of files from a client will again check whether the incoming files conform to the configured constraints.

- The client must be able to resubmit files and in that case it will be prompted whether the files will be overwritten or both the copies to be retained (as we see in Windows).

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).**