AOS Assignment 4

Distributed File Sharing System

Deadlines :

1. **Interim Submission :** `7th October 2023, 11:59:00 PM`
2. **Final Submission :** `12th October 2023, 11:59:00 PM`

Guidelines :

- **Languages Allowed**: **C/C++**
- All Programs must use system calls only. Use of system() commands, exec family of commands, filesystem library are strictly prohibited.
- Modularize and Indent your codes. Also add comments wherever necessary to promote readability.
- Handle error cases wherever required. (If not done marks will be deducted).
- If the code doesn't compile, no marks will be rewarded.
- Segmentation faults at the time of grading will be penalized.
- Add a **README.md** File (compulsory) which contains instruction to execute your code, working procedure of your code , assumptions you have made and short description about about each feature you have implemented .
- Viva will be conducted at the time of evaluation, so prepare well.
- Keep your imported headers CLEAN. Basically, do not import libraries you are not using, especially banned ones.
- If there is something that can be achieved with only system calls, you are supposed to achieve it only using system calls. As such, there are certain libraries like filesystem that are banned. They should not be in the list of libraries that you have imported at the start of your codes. **We will not be taking into consideration the scenario that you have imported a banned library and not used it, it should simply not be there.** If unsure, make sure to ask on Moodle before using a library that may be banned.

⚠️ **ZERO tolerance** towards any kind of code plagiarism. We will thoroughly check your code with all current/previous year submissions and AI Plagiarism Check will also be done (`So No ChatGPT code :[`)

Pre-requisites:

Socket Programming, SHA1 hash, Multi-threading .

Goal :

In this assignment, you need to build a group based file sharing system where users can share, download files from the group they belong to. Download should be parallel with multiple pieces from multiple peers with support for fallback multi-tracker system, parallel downloading and custom piece selection algorithm.

Architecture Overview:

The Following Functionalities should be present in the system :

1. **Synchronized trackers (2 tracker system) :**
    a. Maintain information of clients with their files (shared by client) to assist the clients for the communication between peers.
    b. Trackers should be synchronized i.e all the trackers if online should be in sync with each other.
2. **Clients:**
    a. User should create an account and register with tracker
    b. Login using user credentials.
    c. The clients which create the group, by default should become owner of that group. (If Owner leaves the group, some other group member should become owner of that group).
    d. Fetch list of all Groups in server.
    e. Request to join a group.
    f. Leave Group.
    g. List/Accept Group join requests(if owner)

h. Share file across group: Share the filename and SHA1 hash of the complete file as well as piecewise SHA1 with the tracker.

i. Fetch list of all sharable files in a Group.

j. ***Download file*** **[Core Part]**

- Retrieve peer information from tracker for the file.
- **Core Part:** Download file from multiple peers (different pieces of file from different peers - piece selection algorithm) simultaneously and all the files which client downloads will be shareable to other users in the same group.
- Your algorithm should ensure that pieces are downloaded from more than 1 peer (If available).
- Ensure file integrity from SHA1 comparison.
- Users should be able to download files concurrently in their respective sessions.

k. Show downloads

l. Stop sharing file

m. Stop sharing all files(Logout)

n. Whenever client logins, all previously shared files before logout should automatically be on sharing mode.

Working:

1. At Least one tracker will always be online.
2. Client needs to create an account (user_id and password) in order to be part of the network.
3. Client can create any number of groups (group_id should be different) and hence will be owner of those groups.
4. Client needs to be part of the group from which it wants to download the file
5. Client will send join request to join a group
6. Owner Client Will *Accept*/*Reject* the request.
7. After joining group, client can see list of all the shareable files in the group.
8. Client can share file in any group (as an owner or member; note: file will not get uploaded to tracker but only the <ip>:<port> of the client for that file)

9. Client can send the download command to tracker with the group_id and filename, and tracker will send the details of the group members which are currently sharing that particular file.
10. After fetching the peer info from the tracker, client will communicate with peers about the portions of the file they contain and hence accordingly decide which part of file to take from which peer (You need to design your own Piece Selection Algorithm)
11. As soon as a chunk of file gets downloaded it should be available for sharing (the client becomes a 'leecher')
12. After logout, the client should temporarily stop sharing their own currently shared files/file chunks till the next login.
13. All trackers need to be in sync with each other, so that any seeding/sharing information is available to all.

Commands:

1. Tracker:

- **Run Tracker:** `./tracker tracker_info.txt tracker_no`

  tracker_info.txt - Contains ip , port details of all the trackers

- **Close Tracke**r: `quit`

2. Client:

- **Run Client**: `./client <IP>:<PORT> tracker_info.txt`

  tracker_info.txt - Contains ip, port details of all the trackers

- **Create User Account:** `create_user <user_id> <passwd>`
- **Login:** `login <user_id> <passwd>`
- **Create Group:** `create_group <group_id>`
- **Join Group:** `join_group <group_id>`
- **Leave Group:** `leave_group <group_id>`
- **List pending join:** `list_requests <group_id>`
- **Accept Group Joining Request:** `accept_request <group_id> <user_id>`
- **List All Group In Network:** `list_groups`

- **List All sharable Files In Group:** `list_files <group_id>`
- **Upload File:** `upload_file <file_path> <group_id>`
- **Download File:** `download_file <group_id> <file_name> <destination_path>`
- **Logout:** `logout`
- **Show downloads:** `show_downloads`
    - Output format:

      [D] [grp_id] filename

      [C] [grp_id] filename

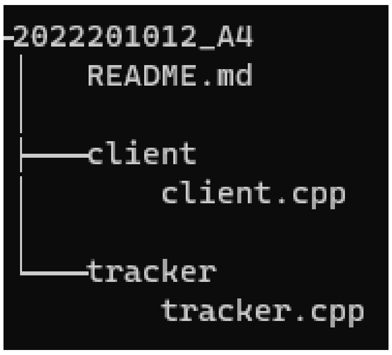
      D (Downloading), C (Complete)
- **Stop sharing:** `stop_share <group_id> <file_name>`

Note :

- **Piece Division:** You have to divide the file into logical "pieces", wherein the size of each piece should be `512KB` .
- **File/Piece Integrity** : SHA1: Suppose the file size is 1024KB, then divide it into two pieces of 512KB each and take SHA1 hash of each part, assume that the hashes are HASH1 & HASH2 then the corresponding hash string would be H1H2 , where H1 & H2 are starting 20 characters of HASH1 & HASH2 respectively and hence H1H2 is 40 characters
- Authentication for login needs to be done.
- You can implement your own version of a torrent architecture. You don't need to implement the exact same protocols mentioned in the BitTorrent paper. You can create your own algorithm and architecture.
- You need to design your own **Piece Selection Algorithm.**

Submission Format :

- Zip file should contain 2 folders (tracker and client). You are allowed to use ***make*** command for this assignment.
- **Submission Format : <Roll_Number>_A4.zip**
- **Hierarchy** of **<Roll_Number>_A4**



```
-2022201012_A4
 |    README.md
 |
 ├───client
 |        client.cpp
 |
 └───tracker
          tracker.cpp
```

- All the files for client should be under client folder .
- All the files for tracker should be under tracker folder .
- Apart from these there should not be any other folder/subfolder .