Assignment-3 Report

CS3.304: Advanced Operating Systems

Author: Aashrey Jain
Roll Number: 2024202012

Objective

The primary objective of this assignment was to build a a group-based file sharing system where users can share and download files from the group they belong to. This report contains the summary of the submission that I have developed along with the implementation details of a few key features.

Tracker

The tracker is the central block of the peer-to-peer system as it helps the peers find other peers, which host the files. The tracker itself does not store any file, but only metadata about the files that the peers share.

I have implemented a single tracker system that is configured to listen on the port specified in the *tracker_info.txt* file for incoming connection requests from peers. Each connection to the tracker spawns a new thread that handles all communication from that particular peer. When the peer closes the connection, the thread is terminated. The connection is not closed by the tracker upon peer *logout* command so that the user can re-login in the same session.

Design

The tracker contains the following data structures:

1. User: A class that models a user using the system. The attributes are a userID and password.
2. Class Group: A class that represents a group within the system. A group contains the following information: groupID, ownerID, joiningRequests, members, and files
3. Class torrentFile: A class that represents a file that the tracker is aware of. It contains the following information: path, numPieces, hashSequence, and seeders
4. Class Seeder: A class that represents a seeder of a file in a group. It contains the following information: socketId and isSharing

The following auxiliary structures are also maintained:

1. clientUserMapping: a mapping of socketID to userID. An entry is added to this map when the user logs in to determined which user is talking to the tracker from which socketID.
2. userMapping: a mapping of userID to User objects. An entry is added to this map when a new user is created, so that the tracker is aware of all the users existing in the system.
3. groupMapping: a mapping of groupID to Group objects. An entry is added to this map when a new group is created, so that the tracker is aware of all the groups existing in the system.

Working

As mentioned earlier, the tracker is a multi-threaded program. A new thread is spawned on each connection to the tracker to handle all commands sent by that particular peer. The initial setup consists of reading the *tracker_info.txt* file to get the desired IP and port configuration. Accordingly, a socket is created and bound to the system port to accept incoming connections. Upon each new connection, a new socketFD is generated that facilitates the communication between the tracker and that peer.

Upon receiving any command from the peer, first it is validated to ensure correctness of the incoming arguments. Validations such as number of arguments, existence checks, authentication, etc. are performed. Upon completion of all validations, the command is processed and the output is returned to the peer.

For the core part (upload and download of files), the tracker does the following:

a. During file upload, the tracker receives and stores information about a file by creating a torrentFile object and storing it in the Group object within which that file was shared. It receives the following information from the *seeder* of the file as semicolon-delimited strings:
    a. the number of pieces the file has been divided into
    b. the complete SHA-1 hash sequence of the file (concatenated hex-encoded SHA-1 hashes of the individual pieces of the file)
    c. the socketID (IP and port) of the seeder who is trying to upload the file to the group.

b. During file download, the tracker returns the following information to the *leecher* of the file as semicolon-delimited strings:
    a. The number of pieces the file has been divided into
    b. The complete SHA-1 hash sequence of the file
    c. The socketIDs of all the seeders who have previously uploaded or downloaded the file and are currently available to share it with other peers.

During *list_files* command, the tracker only returns the names of the files that have at least one seeder.

For a file to be sharable among peers, at least one seeder i.e. peer who has the complete file should be online. If not, the file cannot be shared.

Other features are implemented as required in the assignment problem statement.

Client (Peer)

The peers are the core part of this system since they are the ones communicating with the tracker to update it about the status of the files that they have. Peers also read the requested file from the filesystem and send the piece with the requested index to another peer who is downloading the file.

Design

The view of the file from the peer's perspective is as follows:

a. path
b. groupName
c. fileStatus: a single character denoting the status of the file. Possible values are:
    a. L: local file that the peer is seeding
    b. C: a file that has been downloaded completely
    c. D: a file that is currently downloading
d. numPieces

The main types of threads running in the peer are:

a. Main thread that handles user input and communicates with the tracker to perform the operation requested.
    i. For *upload_file* and *download_file* operations, a thread is created and detached from the main thread so that these operations can be performed parallelly. The main thread does not wait for these threads to complete and is available to take the next user command.
b. Peer thread that binds to a port on the system and listens to incoming requests from other peers for chunks of files that this peer might have uploaded to the tracker. This thread creates a sister thread to handle the request from the current peer and is available to accept connections from other peers on the same port.
c. File download threads (discussed in "Working" section)

Working

During initial setup, the peer takes the IP and port that it has to bind to, as command-line arguments. It creates and configures a socket and binds to the port on the local system, waiting for incoming file-transfer requests from other peers.

It also reads the *tracker_info.txt* file to read the socketIDs of the trackers, and creates and configures a socket to communicate with the tracker.

Other than handling user commands and communication with the tracker in the main thread, the peer operates mainly in two modes:

1. For incoming file requests, the peer receives the filename and index of the piece being requested. After initial validations, the peer opens the file in read mode, adjusts the offset using *lseek* system call and reads the file contents to a buffer of unsigned char datatype. It then transmits the file data over the socket in chunks of 32KB, since the maximum TCP packet size is 64KB, till the entire chunk of 512KB is sent to the peer. The seeder then closes the connection.
2. When the current peer wants to download a file shared in the group, it sends a request to the tracker and obtains the number of pieces that the file has been divided into, the SHA-1 hash sequence of the complete file and the list of peers (as socketId strings) that currently have the complete file.
    a. For each piece, the peer spawns a new thread to randomly choose a seeder and connect to it to request that piece index. These threads are not detached from the main thread since the file writing operation can only begin once all the threads have completed the download of their part of the file.
    b. A limit of 50 threads is imposed on the download to ensure system stability. If 50 threads are created, the main thread waits for all of them to terminate before creating the next 50 threads for the file download.
    c. Each thread connects to a peer selected randomly and requests a chunk of the file. It stores the chunk in a temporary file with a predetermined naming convention.
    d. Once all threads have completed i.e. all chunks have been downloaded to the respective temporary files, these files are read in order and the contents are appended to the main file that is created at the destination specified by the user. The temporary files are deleted as they are read and appended.
    e. Once the file has been assembled, a request is sent to the tracker to notify it that the download has succeeded and that the current peer is also now a seeder of that file in the group.

This concludes the implementation details of my peer-to-peer file sharing system.

More Details

Additional details regarding compiling and executing the tracker and peer processes are mentioned in the README.md file attached along with this report.