Operating Systems (PG)
Assignment 2 - Mini-Torrent File Sharing System
Date - 4th September
Deadline - 20th September 11:55 PM

In this assignment you will be implementing a functional torrent system including your own tracker system (albeit with a much smaller feature set).

How torrents work ?

For the uninitiated, bittorrent is a P2P file sharing protocol which accounts for over 25% of traffic on the internet.

In peer-to-peer file sharing, a software client on an end-user PC requests a file, and portions of the requested file residing on peer machines are sent to the client, and then reassembled into a full copy of the requested file.

Part 1 - Creation of torrent file at client

Normally the bittorrent systems work via sharing a ".torrent" file which contains the metadata related to the file being shared in plaintext format. For this assignment you will develop your own format of .torrent file called the ".mtorrent".

It should contain the following fields:

- Tracker UR L1: The url of the tracker server1, i.e. tracker1 IP & port
- Tracker URL2: The url of the tracker server2, i.e. tracker2 IP & port
- Filename : the name with which the file would be saved on disk.
- Filesize : Complete file size
- Hash String: String whose length would be a multiple of 20. This string is a concatenation of SHA1 hashes of each piece of file.

Taking SHA1 hash of files:
You have to divide the file into logical "pieces" , wherein the size of each piece should be 512KB.

Example: 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 H1 & H2 then the corresponding hash string would be H1H2 , where H1 & H2 are 20 characters each and H1H2 is 40 characters.

Your program will create a <filename>.mtorrent file whenever it shares a file on the tracker (signifying that it is a seeder for that file).

Part 2 - Torrent tracker

A **tracker** is a special type of server, one that assists in the communication between peers using the BitTorrent protocol.

The "tracker" server for your assignment will keep track of all the available peer machines on which the files or its parts reside.

After the initial peer-to-peer file download is started, peer-to-peer communication can continue without the connection to a tracker.

Implementation Specification for tracker

- Start Tracker using:
  ./tracker <my_tracker_ip>:<my_tracker_port> <other_tracker_ip>:<other_tracker_port> <seederlist_file> <log_file>
  - seederlist_file: contains list of all the available seeders.
  - Both trackers should be started separately using the above command.

- Tracker should be able to handle multiple requests at once.

- Have an “share” functionality for the torrent clients via which they will notify the tracker of existence of files.
  - Example: Suppose a client shares a file named “A.txt” then tracker should store the following :-
    - Filename
    - Hash string of the pieces in the file
      - SHA1 hash of hash string which is used in .mtorrent file.
    - Client IP & Listen Port

- Have a “seederlist” functionality for torrent client when a file is requested for downloading. This will return the list of all available seeders for the given file.

- Have a “remove” functionality to allow removing shared file.

- Your tracker server should be fault tolerant, run two copies of the tracker which should be synchronized. The logic for synchronization should be at tracker level only.

- Seeder information should be updated if the tracker goes down as soon as it comes back up.

Part 3 - Torrent Clients

For the final part you will be further extending your client from part 1. In a nutshell, your torrent client should have the following functionalities:

- Share files to the tracker and generate a corresponding ".mtorrent" file
    - You have already done this in Part 1
- Retrieve peer information from tracker upon providing a ".mtorrent" file.
- Download files from multiple peers simultaneously, same goes for upload.
- Maintain mapping of paths of files and related .mtorrent files- this should be persistent across runs.

Implementation Specification-

Start client using - ./client <CLIENT_IP>:<UPLOAD_PORT> <TRACKER_IP_1>:<TRACKER_PORT_1> <TRACKER_IP_2>:<TRACKER_PORT_2> <log_file>

When the application starts, the available .mtorrent files and related files will be checked and request to add the client name in respective files on tracker server will be sent.

The client will have a menu driven interface to do the following:

- **Sharing a local file**:
    - *Command*: share <local file path> <filename>.mtorrent
        - The file path can be absolute or relative.
    - The application should then create a .mtorrent file according to the specifications specified in part 1. This .mtorrent file can be shared manually among clients. This .mtorrent file should never be shared to the tracker.
    - Use the "share" functionality of tracker to notify it about the shared file. (Refer Part 2)
    - After sharing, the client should start seeding the file.

- **Downloading a remote file:**
    - *Command:* get <path to .mtorrent file> <destination path>
        - Path can be absolute or relative.
        - This cannot be a blocking command.
        - Your client must also able to download multiple files simultaneously. (this will be done using multiple get commands)
    - Use the "seederlist" functionality of tracker to retrieve the peer list from where the file can be downloaded.
        - Seeder list once retrieved should be treated as static until the file is downloaded or cancelled downloading. No need to poll continuously for new peers of the file.

    - Your client *must* download the same file from multiple peers simultaneously
        - Think in terms of “pieces”.
        - Peers may or may not have the complete file, i.e. all the pieces.
        - Consider provided list of pieces from clients to be static. No need to poll for update on availability of pieces.

- **Seeding a downloaded file:** As soon as file has begun downloading, it should be notified to the tracker and the client should be able to start seeding that file. Seeding multiple files, and same file to multiple clients is possible.

- **Show downloads:**
    - *Command:* show downloads
    - This will list out the files that are downloading and files that have finished downloading.
    - To differentiate, downloading files will have [D] and downloaded files will have [S] before the file name.

- **Removing a shared file:**
    - *Command:* remove <filename.mtorrent>
    - This should remove the file information for that file from the tracker and from the persistent file database along with the .mtorrent file but not the actual file.
    - Remove only when file has been 100% downloaded.

- **Close application:** When the application is closed, a request should go to the tracker to remove the client from the seeder list of all files it is present in. Assume that application will only be closed when there are no ongoing uploads or downloads.

- **Bonus:** Ability to remove file or close application when a file is being downloaded or uploaded. This should have a graceful handling of connection if another client is getting the file that is removed.

Pointers:

1. Read up on how bittorrent works. Your viva will also include the actual bittorrent system.
2. SHA1 online - http://www.sha1-online.com/
3. Learn how to program simple a simple sockets connection in C/C++.
4. You’re free to implement multithreading or multiprocessing.
5. Your system will be offline if both trackers go down.
6. You are not allowed to use external C++ libraries except for SHA-1.
    a. For SHA-1 use openssl library.
    b. Boost library is not allowed.
7. Language - C/C++
8. Log files will be useful for debugging.