CS550 Advanced Operating System

Programming Assignment 4

Meng Tang A20455416

Design Documentation

Overview

This document contains detailed description of the different entities when building the peer to peer concurrent file transfer system. It also describes the trade offs in design, limitation, and possible improvements of the program. The program is written in python3 and all the libraries used comes in the standard python library.

Communication Design:

Index server is started first and it maintains a list of registered peers. The index server uses a default port 60000, and each peer should have a number ID that matches its local directory file name. The ID is then added to the default server port to generate a new port for peer to listen for other incoming connections. Each peer is stored in an active peer list in the index server, the key is a tuple of the peer's host and the peer's port for other peer connections. This tuple would also serve as the main identification of each peer in the index server. The value of the (server,port) tuple key is another dictionary, which contains each file name of the peer in its local directory as the key, and the file size as the value. The index server when requested to download a file, it will go through every peer in its list and give a list of all the peer that has the file, along with the file size back to the original requestor. The requestor then downloads a chunk of the file from each server.

Each peer right after its registration with the index server within a thread, it will start another thread as a server, which binds to a port and listens for incoming connections. Once a connection is accepted, it goes straight to the downloading process.

Once the user chosen to exist, the peer node checks for any remaining connections to itself, it waits for 1 second before the next check. It will only check 5 times then it will start unregistering itself with the index server and close socket connection if any. The index server will remove the (server,port) tuple from its active peer list. Once the key is removed from the index server, other peers will not be able to download from the peer until it re-register itself with the index server again.

File Transmission Through Socket:

For downloading files from another peer, once a list of available peers is obtained from the server, the peers first use the file size and its chunk size to determine how many chunks it will need. The chunk size is hard coded as 2MB. Then it determines which peer from the list it will download from according to chunk number mod the peer list number. The resulted number is used as the index for the peers list to obtain a specific peer. Then it sends the peer's host and port information and the file name, along with a chunk number to a thread to do the download work. To download one chunk of the file, the requester peer will connect to the target peer, then starts the downloading process. It reads a buffer with 4096 bytes until the end when the buffer is being closed by the target peer. Then it stores the chunk number as a key and the file contents as the value in a dictionary entry. All file chunk values are stored in the same dictionary. If the chunk number is 0, then the requester will also receive a hashcode of the file from the target peer. When receiving files with an infinite while loop, the time out is set to be 15

seconds. It is tested that the longest time it takes for my computer to transfer all the chunks is about 8 seconds.

On the target peer side, once it receives an incoming connection from the requester, it starts a thread to send the file content. It uses the chunk number to locate the offset of the file the requester desired, and read the specified chunksize by the requester and send all to the buffer. If the chunk number is 0, the target peer also sends a file hash code to the requester.

Once all the chunks are added to the one large dictionary, with the key the chunk number and value the chunk content, the peer will write the content to file in order. Afterwards it opens the file again to obtain a hash code to compare with the hash code it receives from the first file chunk to determine its integrity. It then returns a successful or failure message notifying the user if they should retry or not. If no peers were given from the server, it will also return a failure message.

Multi-threading:

In the peer, the main process is the peer acting as a client to connect to the indexing server. After connection to the indexing server is successful, it will start a new thread to set up the peer as a “server” and listens for incoming connections. Once a new connection is accepted, it will start another new thread for handling file transferring communication. Once the file transfer communication is done, the thread returns back to the “server” thread part of the peer. Multiple threads will be generated if there are multiple incoming connections. The downloading file size will start thread and generate thread workers the same as the number of chunks that it needs to download. This allows the peers to download files and transfer files to other peers at the same time.

Limitation:

Currently peers can only request to download one file from other peers. There will need a bit modification to enable concurrent downloads of multiple files. The peer program has no method of recovery if connections were interrupted. The program has only been tested on one machine locally.

Further improvement:

If multiple files were to be implemented, the server side will also need to take care of the message including the list of peers. The list should be a dictionary with the server and port of the target peers as the key, then the value is another dictionary with the file name as key and file size as value. In the peer side, there will be one more layer of threads. Starts the first layer of threads that correspond to different peers, this could achieve concurrent download from multiple peers. Then start the seconds layer of threads that corresponds to the chunks of files.

There could also be improvement in the user part. While the user process might crash, session information could be stored in the config file or somewhere as data. Server can also store active peers to a file every time there is a change in the list. This can make the program more fault tolerant.

Currently there is no authentication process and encryption of information between the server and all the peers. This could be added to ensure peer data safety. Peer can generate a port number randomly every time it starts, or change the port number from time to time during its active time and update it to the server. Port number and file content could be encrypted when sending and forwarding over the internet.