**CS550 Advanced Operating System**

**Programming Assiggment 2**

**Meng Tang A20455416**

**Design Documentation**

**Overview**

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

**Communication Design:**

Index server is started first and it maintains a list of registered peers. The index server uses a default port 12345, 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 a 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. The value of the key is the list of files that the peer has in its local directory. When ever a peer requests to download files, it first go to the index server to ask for the peers that has the target files. The index server will return a list of tuples of the peers that has the targeted file names. The key that contains those peers’ host and port is then returned to the requester. The requester then can choose from the list and connect to the chosen host and port to download the files.

Each peer right after its registration with the index server within a thread, it will start listening of its designated host and port with another thread. Once a connection is accepted, it goes straight to the downloading process.

Once the user chosen to exist, the peer node first waits for 10 seconds and checks if all downloading threads are finished. After some timed out it then sends an unregistered message to the server along with its host and port identifier. The index server will remove the key 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 then randomly pick one from the list and tries to connect to it. If the connection failed initially, it will retry the connection after every 0.5 second. It will stop after 5 retries. A failure message will be recorded. If the connection success, it starts the downloading process.

The peer file download function sends the file list out again but to the chosen peer. All the files are transferred serially and is assumed to be returned the same order as it is requested. The requester first get a return message of the length of size of the next incoming content. Then it receives with the corresponding size buffer, which includes the file’s md5 hash code and the file content. The received file content is then used to recalculate a md5 hash code to compare with the received one. If the hash code matches, the peer will accept the file to its local directory.

After the file is downloaded to the peer’s local directory, the function also obtain a new list of the files the current peer has, then send it to the index server along with its host and port. The index server will update its list with the given host and port as the key.

**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 down, the thread returns back to the “server” thread part of the peer. Multiple threads will be generated if multiple other peers were to connect with this current peer. Since accepting file will only need one process, it will be handled by the main process which is the “client” part of the peer. This allow the peers to download files and transfer files to other peers at the same time.

**Limitation:**

Currently peer can only requests to download all of the target files from one peer. It will only make one connection to one peer for all the files in one request. And it will download files serially only.

The peer program have basic timeout mechanism to wait and retry if connection or requests fail. But no method of recovery is implemented if connection were interrupted.

The server seem to only be able to handle 4 concurrent peer connection safely. If more than 4 concurrent peers are sending request to the server, some of the message pipes will break.

**Further improvement:**

Better file sending mechanism should be implemented to account for large file transfer that the receiver would not be able to receive with only one buffer receive. More fault tolerance mechanism on the server side should be implemented to ensure stable connection for more concurrent peer requests.

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 server. Port number and file content could be encrypted when sending and forwarding over the internet.