README: Project 1

Author: Drew Gonsalves

Date Updated: 10/30/15

Name: Peer-to-Peer (P2P) Chunk Sharing Program

Description:

This user documentation describes a JAVA program intending to mimic the file sharing process of a Peer to Peer system. The process will create a host, who is the owner of a single file to be distributed, that will distribute ‘chunks’ of the file system evenly amongst users. The users of the system (“peers” or “clients”) will then trade amongst themselves, taking on roles as ‘downloader’ and ‘uploader’ in a circle configuration (ex: Peer 1 will upload chunks to Peer 2, and so on..). Once a peer receives all the chunks, he/she will close the connection, and then combine the chunks into one file.

Project Requirements: (Chen)

1. Start the file owner process, giving a listening port
2. Start five peer processes, one at a time, giving the file owner’s listening port, the peer’s listening port, and its download neighbor’s listening port.
3. Each peer connects to the server’s listening port. The latter creates a new thread to download one or several file chunks to the peer, while its main thread goes back to listening for new peers.
4. After receiving chunk(s) from the file owner, the peer stores them as separate file(s) and creates a summary file, listing the IDs of the chunks it has.
5. The peer then proceeds with two new threads, with one thread listening to its upload neighbor to which it will upload file chunks, and the other thread connecting to its download neighbor.
6. The peer requests for the chunk ID list from the download neighbor, compare with its own to find the missing ones, and download those from the neighbor. At the mean time, it sends its own chunk ID list to its upload neighbor, and upon request uploads chunks to the neighbor.
7. After a peer has all file chunks, it combines them for a single file.
8. A peer should output its activity to its console whenever it receives a chunk, sends a chunk, receives a chunk ID list, sends out a chunk ID list, requests for chunks, or receives such a request.

System Requirements:

1. Java Virtual Machine (JVM)

How to compile and run:

1. Run from IDE (Eclipse, Visual Studio)
   1. Main Program – ‘Main’

OR Run from command line

1. - Mac -Open 'Terminal'
   1. -'cd' to folder containing project
   2. Run ‘java –jar <PROGRAM>’
2. Follow user prompts
   1. Enter full pathname
   2. Enter number of peers (clients)
   3. Enter size of chunks (kb)

How it works:

This program works by creating a Host that takes a single, complete file and breaks the file into smaller chunks based on user input. A series of Client sessions are opened that will receive an equal number of chunks from the Host. The Clients will then open up sessions with 2 other Clients, creating an ‘uploading’ and a ‘downloading’ session, providing chunks to one and uploading his/her chunks to the other.

The program is entered through a simple Main that starts up the file sharing program called Torrent Program. The Torrent Program manages the user’s desired size of chunks and number of users that will share his specified file. The program will then start one Host and the specified number of Clients (Peers).

The program works by creating Sockets/Threads that link a serving and receiving client. A host will open a listening ServerSocket() with a client probing listening ports with a Socket(). Upon connecting, we will loop with communications of sending summary chunk ID lists that the downloader is requesting and the server providing those chunks. These operations will be controlled by a Handler, of type Thread.

The Host/Uploader will be in charge of sending his/her chunks to an inquiring Client.

The Handler Thread will maintain the connection between either the Host and Client or two Clients and will manage the operations for finding the chunks the Host has that the Client needs and serving them to the Client.

The Client/Peer will be in charge of sending his/her chunk ID summary to a host to request the chunks he or she does not have. The Client will be in charge of saving the chunks and disconnecting when he/she receives all of the chunks necessary to recreate the file. The Client will then combine the chunks into one file and store it.

Classes:

* Main – Handles starting the main program
* TorrentProgram – Handles user input and creates Host/Peers in the system
* Host – Breaks up the file into chunks, creates chunk ID lists to send to Peers evenly, and listens for incoming connections
* Peer – Connects with host, receives chunks, send chunk ID lists requesting for chunks, and stores chunks and saves file complete file
* Handler – A Thread that acts a pipe between Host/Peer or Peer/Peer connections providing a dialogue between a requesting/providing components of the system.

Core Functions:

1. Main
   1. Main() – starts program
2. TorrentProgram
   1. Makefolder()-creates folder to save all transaction data of the system
   2. createPeers()-creates the Peers in the system
   3. createHost()-creates the one Host in the system
3. Host
   1. Loadfile()/breakfile()-manages the breaking of a file into a user specific number of chunks
   2. Run()-the core that runs upon Host creation. Manages opening up listening ServerSocket and receiving Client request and creating Handler Threads.
4. Peer
   1. getChunks()-creates the summary chunk ID request when the peer acts as a downloader
   2. recreatefile()-merges/combines all chunks into one file that is a clone of the original given to the Host
   3. run()-runs when the Peer is started. Manages the connection to the intial Host to receive intial chunks. Also manages the connection when this peer acts as a downloading peer and as an uploading peer.
5. Handler
   1. compareSummary()-determines the chunks the uploader has by taking the difference of the chunk id list the requesting downloader provides
   2. sendChunk()-sends one chunk at a time from the compareSummary() return. Made this way in order to keep the Peer run() method logic easier.
   3. Run()-Because the Handler thread is opened by a hosting host/peer, this method runs continuously listening for requesting peer chunk ID summaries to later comparesummary() against and provides the loop to send chunk data to the peer.

Testing:

File: chord.pdf

Type: PDF

Size: 195kB

Number of Users : 3

Size of Chunks : 50kb

Output from Eclipse –

Created by: Drew Gonsalves

Starting PEER-TO-PEER 'Circle' architecture system...

Directory created for P2P: /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp

Host was created...

Directory created for Host: /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp/host

Loading file at Host...

Size of file in bytes: 194676

File broken at Host into 0 chunks...

Host has Chunk ID 0 chunk file: /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp/host/chunk\_id=000\_host\_.chunk

Host has Chunk ID 1 chunk file: /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp/host/chunk\_id=001\_host\_.chunk

Host has Chunk ID 2 chunk file: /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp/host/chunk\_id=002\_host\_.chunk

Host has Chunk ID 3 chunk file: /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp/host/chunk\_id=003\_host\_.chunk

The Host is running...

Directory created for peer 0: /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp/peer0

Peer 0 was created...

Directory created for peer 1: /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp/peer1

Peer 1 was created...

Directory created for peer 2: /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp/peer2

Peer 2 was created...

Peer 1 connected to Host on port 8000

Peer 2 connected to Host on port 8000

Peer 0 connected to Host on port 8000

Peer 0 is the uploader to Peer 1 on port 20000

Peer 1 is the uploader to Peer 2 on port 20001

Peer 2 is the uploader to Peer 0 on port 20002

Peer 2 is the downloader to Peer 1 on port 20001

Peer 1 is the downloader to Peer 0 on port 20000

Connection refused. You need to initiate a server first.Peer 1 succesfully started its own UPLOAD thread on port 20001 with Peer 2

Peer 0 is the downloader to Peer 2 on port 20002

Peer 2 succesfully started its own UPLOAD thread on port 20002 with Peer 0

Peer 0 succesfully started its own UPLOAD thread on port 20000 with Peer 1

Peer 2 requested chunks from Host...

Peer 2 sent Chunk ID list to Host...

Peer 0 requested chunks from Host...

Peer 1 requested chunks from Host...

Peer 0 sent Chunk ID list to Host...

Peer 1 sent Chunk ID list to Host...

Peer 1 requested chunks from Peer 0...

Host received chunk request from Peer 0...

Host received chunk request from Peer 0...

Host received chunk request from Peer 0...

Peer 0 requested chunks from Peer 2...

Host received Chunk ID list from Peer 0...

Peer 2 requested chunks from Peer 1...

Host received Chunk ID list from Peer 0...

Host received Chunk ID list from Peer 0...

Peer 2 sent Chunk ID list to Peer 1...

Peer 1 sent Chunk ID list to Peer 0...

Peer 0 sent Chunk ID list to Peer 2...

Peer 0 received chunk request from Peer 1...

Peer 1 received chunk request from Peer 2...

Peer 0 received Chunk ID list from Peer 1...

Peer 2 received chunk request from Peer 0...

Peer 1 received Chunk ID list from Peer 2...

Peer 2 received Chunk ID list from Peer 0...

Host sent chunk ID 0 to Peer 0

Host sent chunk ID 1 to Peer 0

Host sent chunk ID 2 to Peer 0

Peer 1 received Chunk ID 0 to give Chunk ID list: [0]

Peer 0 received Chunk ID 2 to give Chunk ID list: [2]

Peer 2 received Chunk ID 1 to give Chunk ID list: [1]

Peer 2 requested chunks from Host...

Peer 1 requested chunks from Host...

Peer 0 requested chunks from Host...

Peer 2 sent Chunk ID list to Host...

Peer 1 sent Chunk ID list to Host...

Peer 0 sent Chunk ID list to Host...

Peer 0 requested chunks from Peer 2...

Peer 1 requested chunks from Peer 0...

Host received chunk request from Peer 0...

Host received chunk request from Peer 0...

Host received chunk request from Peer 0...

Peer 1 sent Chunk ID list to Peer 0...

Peer 0 sent Chunk ID list to Peer 2...

Host received Chunk ID list from Peer 0...

Host received Chunk ID list from Peer 0...

Host received Chunk ID list from Peer 0...

Peer 2 requested chunks from Peer 1...

Peer 2 sent Chunk ID list to Peer 1...

Host sent chunk ID 3 to Peer 0

Peer 0 received chunk request from Peer 1...

Peer 2 received chunk request from Peer 0...

Peer 0 received Chunk ID list from Peer 1...

Peer 2 received Chunk ID list from Peer 0...

Peer 1 received chunk request from Peer 2...

Peer 1 received Chunk ID list from Peer 2...

Peer 0 sent chunk ID 2 to Peer 1

Peer 2 sent chunk ID 1 to Peer 0

Peer 1 sent chunk ID 0 to Peer 2

Peer 1 received Chunk ID 3 to give Chunk ID list: [0, 3]

Peer 0 received Chunk ID 1 to give Chunk ID list: [1, 2]

Peer 2 received Chunk ID 0 to give Chunk ID list: [0, 1]

Peer 0 requested chunks from Host...

Peer 2 requested chunks from Host...

Peer 0 sent Chunk ID list to Host...

Peer 2 sent Chunk ID list to Host...

Peer 0 requested chunks from Peer 2...

Peer 2 requested chunks from Peer 1...

Host received chunk request from Peer 0...

Host received chunk request from Peer 0...

Peer 2 sent Chunk ID list to Peer 1...

Peer 0 sent Chunk ID list to Peer 2...

Host received Chunk ID list from Peer 0...

Host received Chunk ID list from Peer 0...

Peer 1 received chunk request from Peer 2...

Peer 1 received Chunk ID list from Peer 2...

Peer 2 received chunk request from Peer 0...

Peer 2 received Chunk ID list from Peer 0...

Peer 2 sent chunk ID 0 to Peer 0

Peer 1 sent chunk ID 3 to Peer 2

Peer 1 received Chunk ID 2 to give Chunk ID list: [0, 2, 3]

Peer 1 requested chunks from Host...

Peer 1 sent Chunk ID list to Host...

Peer 1 requested chunks from Peer 0...

Host received chunk request from Peer 0...

Peer 1 sent Chunk ID list to Peer 0...

Host received Chunk ID list from Peer 0...

Peer 0 received chunk request from Peer 1...

Peer 0 received Chunk ID list from Peer 1...

Peer 2 received Chunk ID 3 to give Chunk ID list: [0, 1, 3]

Peer 0 received Chunk ID 0 to give Chunk ID list: [0, 1, 2]

Peer 2 requested chunks from Host...

Peer 0 requested chunks from Host...

Peer 2 sent Chunk ID list to Host...

Peer 0 sent Chunk ID list to Host...

Peer 0 sent chunk ID 1 to Peer 1

Peer 0 requested chunks from Peer 2...

Peer 2 requested chunks from Peer 1...

Peer 0 sent Chunk ID list to Peer 2...

Peer 2 sent Chunk ID list to Peer 1...

Host received chunk request from Peer 0...

Host received Chunk ID list from Peer 0...

Host received chunk request from Peer 0...

Peer 1 received chunk request from Peer 2...

Host received Chunk ID list from Peer 0...

Peer 1 received Chunk ID list from Peer 2...

Peer 2 received chunk request from Peer 0...

Peer 2 received Chunk ID list from Peer 0...

Peer 2 sent chunk ID 3 to Peer 0

Peer 1 sent chunk ID 2 to Peer 2

Peer 1 received Chunk ID 1 to give Chunk ID list: [0, 1, 2, 3]

Peer 2 received Chunk ID 2 to give Chunk ID list: [0, 1, 2, 3]

Peer 0 received Chunk ID 3 to give Chunk ID list: [0, 1, 2, 3]

Combined chunks in Peer 1 to form file chord.pdf in /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp/peer1

Thread hosted by Peer 0 was disconnected due to all chunks being received...

Thread hosted by Host was disconnected due to all chunks being received...

Combined chunks in Peer 2 to form file chord.pdf in /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp/peer2

Combined chunks in Peer 0 to form file chord.pdf in /Users/gonsalves-admin/Documents/School/CNT5106C-Network/Proj1/torrent\_tmp/peer0

Thread hosted by Peer 1 was disconnected due to all chunks being received...

Thread hosted by Host was disconnected due to all chunks being received...

Thread hosted by Host was disconnected due to all chunks being received...

Thread hosted by Peer 2 was disconnected due to all chunks being received...