Contents

[Introduction 2](#_Toc367107281)

[Peer Design 2](#_Toc367107282)

[Server design 3](#_Toc367107283)

[System design 4](#_Toc367107284)

[Basic communication design 5](#_Toc367107285)

[Messages 5](#_Toc367107286)

[Message Format 5](#_Toc367107287)

# Introduction

In this project we have tried to implement a p2p file sharing system. We have used C language for programming and Berkeley Sockets for the communication between the peers and indexing servers.

All peers register themselves with the indexing server. Also they periodically submit the list of files that they can share. The files in the working directory are shared. The indexing server stores all this info and gives it to peers who search for files

Any peer when it searches for files behaves like a client. It must first talk to the Indexing Server to get a list of possible Peer Servers who have the file. Once the Peer Client has this list, it connects to any one Peer Server and sends a request for getting the file.

Design

## Peer Design

The peer can behave both as a client and as a server while receiving or sending files. The peer listens to a port and on receiving a message creates a new thread for processing it. The new thread interprets the message. It then does the necessary processing like sending a reply message or append the sent file bytes to the file.

Create Socket and listen at port

Send Start signal to Iserver

List all files and Send add signal to Iserver

Wait for connections

Sleep 30 secs

List all files

New Connection Received?

For any newly added files, send ADD message to the server

For any deleted files, send REMOVE message to the server

Create child process to handle connection

Process Message

Get Message

Child Process

Fig 2. Flow chart for the Peer

## Server design

Server maintains a list of files shared by each peer in a MySql Database. The list is updated every minute by messages sent by the peers about any changes. On receiving a request the server creates a new process to handle the request. This process updates the necessary tables based on the nature of the request.

Create Socket and listen at port

Wait for connections

New Connection Received?

Create child process to handle connection

Get Message

Child Process

Process Message

Fig 3. Flow chart for the Indexing Server

We send messages for communication. The message format is interpreted at the receiving end to process it.

## System design

We have done the implementation and testing on Ubuntu Linux using gcc compiler

## Basic communication design

* Communication is implemented using Berkeley Sockets. The peers and indexing server listen to ports continuously. On receiving a connection the create a new thread or process to handle the request and go back to listening on the port.
* Indexing server : On receiving a request it creates a new process to handle the request and goes back to listening on its port
* Peers : have two listening ports. Port0 is used for normal listening and Port1 for file transfer related listening

Peer

Port 0

Port 1

I Server

Port 0

Fig 4. Listening ports for the indexing server and peers

### Messages

|  |  |  |
| --- | --- | --- |
| Message | Direction | Comment |
| Start | Peer to Iserver | Peer indicates it is active |
| Add Files | Peer to Iserver | Add a file in the list of files with the peer |
| Remove Files | Peer to Iserver | Remove a file in the list of files with the peer |
| Stop | Peer to Iserver | Peer indicates it is stopped |
| Search | Peer to Iserver | Peer queries the server for a file |
| Search Reply | Peer to Iserver | Iserver sends a list of peers with the file |
| Obtain File | Peer to Peer server | Client request for sending the file |
| Obtain File Reply | Peer server to Peer | Server sending the file as a byte stream |

### Message Format

|  |  |
| --- | --- |
| Message | Format |
| Start | SRT IP:PORT |
| Add Files | ADD “file name” |
| Remove Files | REM “file name” |
| Stop | CLO |
| Search | SRC “file name” |
| Search Reply | SRR [IP:PORT] |
| Obtain File | OBT |”filename” |k:n| |
| Obtain File Reply | OBT |”filename” |k:n|[<bytes>] |