Design Document

Table of Contents

1 Requirements 1

1.1 A central indexing server 1

1.2 A peer 1

2 Architecture 1

2.1 Indexing Server 2

2.1.1 Data Structures 2

2.1.2 API’s 3

2.2 Peer 3

2.2.1 Data Structures 4

2.2.2 API’s 4

2.2.3 Messages from peer to server 4

3 Code flow 5

3.1 Flow for registering a file 5

3.2 Flow for retrieving a peer id and obtain a file 5

# Requirements

## A central indexing server

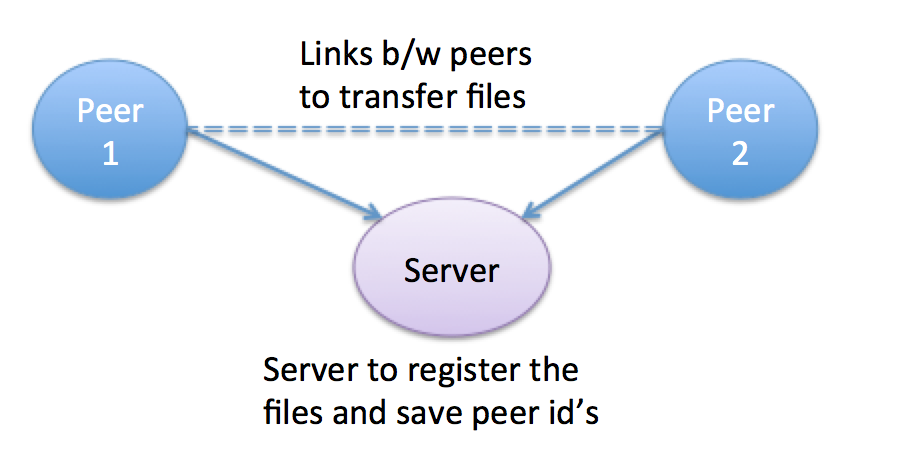
This server indexes the contents of all of the peers that register with it and also provides search facility to peers.

## A peer

A peer is both a client and a server. As a client, the use specifies a file name with the indexing server using “lookup”.

As a server, the peer waits for requests from other peers and sends the requested file when receiving a request.

# Architecture



## Indexing Server

Indexing server would be receiving the requests for registering the files and looking up the peers that have a certain file.

Now, whenever server will receive a request:

1. It would spawn a thread to serve it, be it to register or to lookup a file.
2. In case of registration, server will maintain a list of peer’s along with their files. The key of the list will be the peer’s identity (peer\_id) which is PEER\_IP:PORT combination.
3. For searching, server will go over the list made in step 2 and return the peer\_id that has the file or else it will send a message having contents as “none”. Peer can decipher this using strncmp and take appropriate action.
4. The number of threads is limited by a static variable num\_threads. In case, server as reached maximum number of threads it will send an error message to the requesting peer.
5. Peer can again retry in some time. This is done to give the control to the peer to decide on if server doesn’t have enough resources rather than blocking the peer and consuming its resources.

### Data Structures

|  |  |
| --- | --- |
| /\* Optional parameters \*/  **struct opt\_params\_t** {  int bandwidth;  int demand;  };  /\* List of files that a peer has \*/  **struct file\_list** {  char \*file\_name;  struct file\_list \*link;  }; | /\* List of peers \*/  **struct peer\_list** {  struct file\_list \*list;  char \*peer\_id;  struct opt\_params\_t \*params;  struct peer\_list \*link;  };  **struct msg\_data** {  /\* client fd used by server thread for replying \*/  int client\_fd;  /\* Message received from the client \*/  char \*buffer;  }; |

### API’s

/\* Function to search the files and return the peer\_id's

\* that have the respective files to the client

\*/

void \*search (void \*data);

Sends peer id if a file is found or else sends “none” in the message.

/\* Function invoked to register files with the server \*/

void \*registry (void \*data);

Sends a message saying “Registered <file\_name>” to the peer.

/\* Function to register a peer \*/

void \*deregister (void \*data);

/\* Function to register a file \*/

void \*deregister\_file (void \*data);

## Peer

Peer is a client who can have following three tasks:

1. Register its files with the indexing server
2. Send a request to the server for obtaining the peer’s having a particular file.
3. Connect to another peer that has the file the peer is looking for.
4. Act as a server to send file to a requesting peer.

Every peer will spawn three threads:

Thread 1: Listen to a port as a server

Thread 2: Start a thread to register files with the indexing server

Thread 3: Start a thread to lookup a file from the server and obtain it from the peer.

### Data Structures

#define MAX\_CONN\_LISTEN 100 /\* Maximum clients I can listen to \*/

#define MAX\_MSG\_SIZE 1024 /\* Maximum size of the message \*/

struct peer\_data {

/\* File names \*/

char \*file\_name;

/\* Ip of the indexing server \*/

char \*server\_ip;

/\* Port of the indexing server \*/

char \*server\_port;

/\* My own IP for acting as a server \*/

char \*my\_ip;

/\* My own Port for acting as a server \*/

char \*my\_port;

};

### API’s

/\* Function to lookup and obtain a file from peer \*/

void \*obtain (void \*data);

/\* Function to start a peer as a server \*/

void \*server (void \*data);

/\* Function to register a peer as a server \*/

void \*register\_peer (void \*data);

### Messages from peer to server

Following is the format of messages to be sent to the server to register a peer or lookup a file:

1. Registration Request : register:peer\_id:<files>

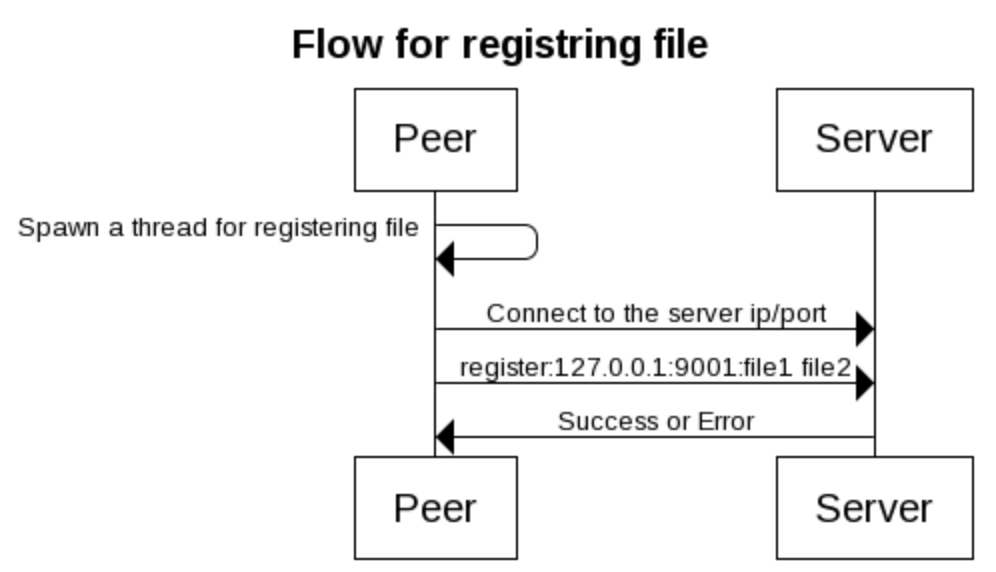
2. Lookup Request : lookup:<file>

3. Deregister file: deregister\_file:<file>

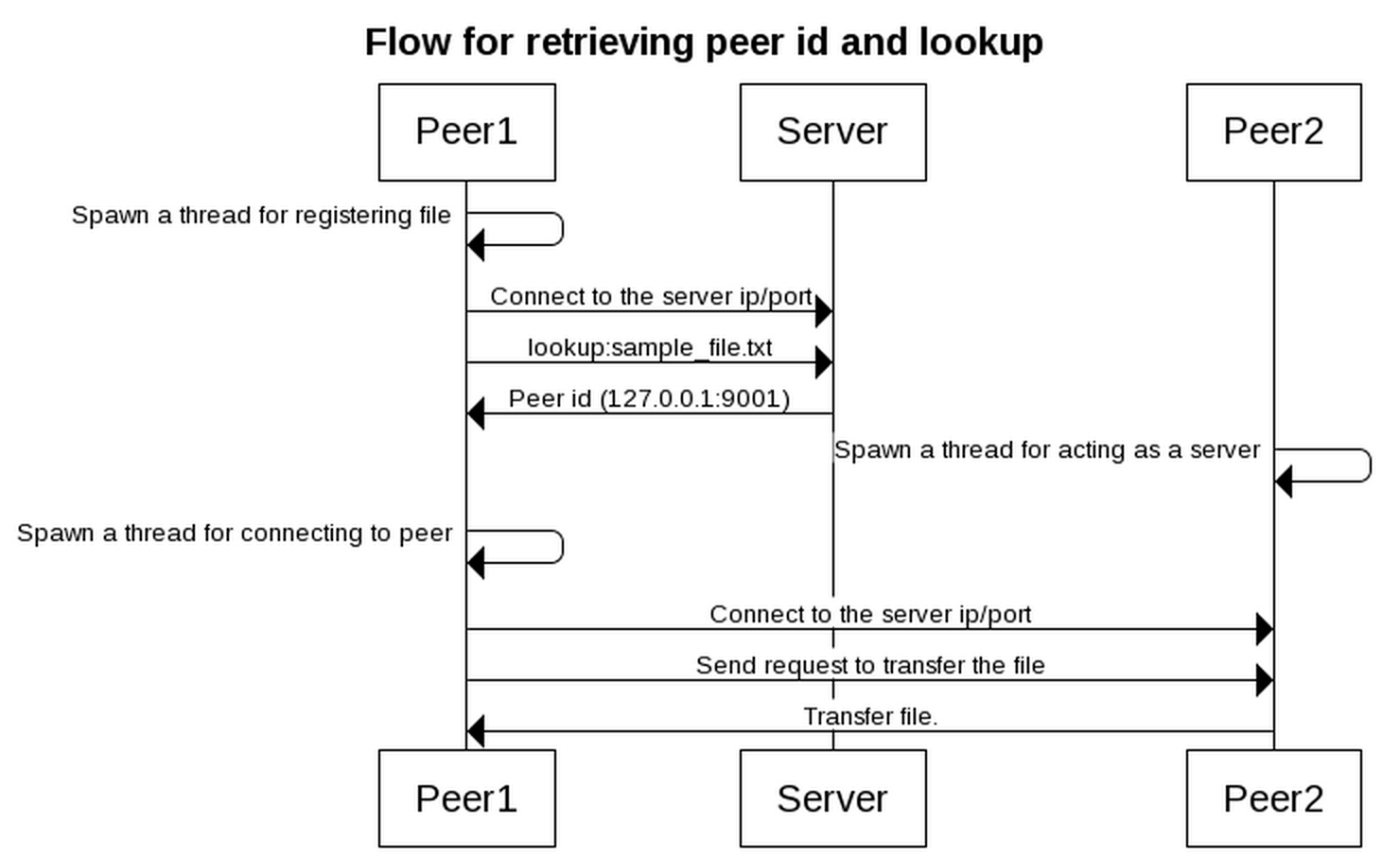
4. Deregister peer : deregister\_peer:<peer\_id>

# Code flow

## Flow for registering a file



## Flow for retrieving a peer id and obtain a file



End of Document