lab: Socket Programming

1 Requirements

1. Implement C/S model: 1)Server listens to a given port (>1024, e.g. 2680) 2) Multiple clients request the same file from the server 3) Each client save the file to its local directory.
2. Implement P2P model: Each peer downloads part of the file from the server, and then distribute it to all the other peers.
3. Use Mininet to compare the overall file downloading time. Study how the number of downloading time changes with respect to the number of peers. You need to create the following star topology in Mininet. You can use one host as a server, and the other hosts as peers requesting files.

2 Clients & Server Model

code for Server

```
#include <stdio.h>  // standard input and output library
#include <stdlib.h> // this includes functions regarding memory allocation
#include <string.h> // contains string functions
#include <errno.h> //It defines macros for reporting and retrieving error conditions through error codes
#include <time.h> //contains various functions for manipulating date and time
#include <unistd.h> //contains various constants
#include <sys/types.h> //contains a number of basic derived types that should be used whenever appropriate
#include <arpa/inet.h> // defines in_addr structure
#include <sys/socket.h> // for socket creation
#include <netinet/in.h> //contains constants and structures needed for internet domain addresses
#include <pthread.h>
const int BUFFER_SIZE = 1024;
void sendfile(void *socket_fd) {
    int fd = *((int *) socket_fd);
    char buffer[BUFFER_SIZE];
    bzero(buffer,BUFFER_SIZE);
    if(recv(fd,buffer,BUFFER_SIZE,0)<0) {
        perror("Server Receive Data Failed!");
    }
    char file_name[FILENAME_MAX+1];
    bzero(file_name,FILENAME_MAX+1);
    strncpy(file_name,buffer,strlen(buffer)>FILENAME_MAX?FILENAME_MAX:strlen(buffer));
    printf("%s\n",file_name);
    FILE *fp = fopen(file_name,"r");
    if(!fp) {
        printf("File:%s Not Found!",file_name);
    }
    else {
        bzero(buffer,BUFFER_SIZE);
        int length = 0;
```

```
        while((length=fread(buffer,sizeof(char),BUFFER_SIZE,fp))>0) {
            if(send(fd,buffer,length,0)<0){
                printf("Send File:%s.\n",file_name);
                break;
            }
            bzero(buffer,BUFFER_SIZE);
        }
        fclose(fp);
        printf("File:%s Transfer Successful!\n",file_name);
    }
    close(fd);

}
int main()
{
    time_t clock;
    char dataSending[1025]; // Actually this is called packet in Network Communication, which contain data and send through.
    int clintListn = 0, clintConnt = 0;
    struct sockaddr_in ipOfServer;
    clintListn = socket(AF_INET, SOCK_STREAM, 0); // creating socket
    memset(&ipOfServer, '0', sizeof(ipOfServer));
    memset(dataSending, '0', sizeof(dataSending));
    ipOfServer.sin_family = AF_INET;
    ipOfServer.sin_addr.s_addr = htonl(INADDR_ANY);
    ipOfServer.sin_port = htons(2017);      // this is the port number of running server
    bind(clintListn, (struct sockaddr*)&ipOfServer , sizeof(ipOfServer));
    listen(clintListn , 20);

    while(1)
    {
        pthread_t thread_id;
        printf("Hi,Iam running server.Some Client hit me\n"); // whenever a request from client came. It will be processed here.
        clintConnt = accept(clintListn, (struct sockaddr*)NULL, NULL);
        if(clintConnt==-1) {
            fprintf(stderr,"Accept error!\n");
            continue;
        }
        struct sockaddr_in sa;
        int len = sizeof(sa);

        if(!getpeername(clintConnt, (struct sockaddr *)&sa, reinterpret_cast<socklen_t *>(&len)))
        {
            printf("New connection from %s:%d !\n",inet_ntoa(sa.sin_addr),ntohs(sa.sin_port));
        }

        if(pthread_create(&thread_id, NULL, reinterpret_cast<void *(*)(void *)>(sendfile), (void *)(&clintConnt))) {
            fprintf(stderr,"pthread create error");
            break;
        }
     }
    return 0;
}
```

```
83
```

when the server starts, it will listen the connection from the clients, and send the file they requests.

It has been developed to handle the multi-thread requests, when connection is established, the server will create a new thread to send the file.

To run the `server`

```
#! /usr/bin/bash
g++ -pthread server.cc -0 server
./server
```

code for Clients

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <arpa/inet.h>
const int BUFFER_SIZE=1024;
int main()
{
    int CreateSocket = 0,n = 0;
    char dataReceived[1024];
    struct sockaddr_in ipOfServer;

    memset(dataReceived, '0' ,sizeof(dataReceived));

    if((CreateSocket = socket(AF_INET, SOCK_STREAM, 0))< 0)
    {
        printf("Socket not created \n");
        return 1;
    }

    ipOfServer.sin_family = AF_INET;
    ipOfServer.sin_port = htons(2587);
    ipOfServer.sin_addr.s_addr = inet_addr("10.0.0.1");

    if(connect(CreateSocket, (struct sockaddr *)&ipOfServer,
sizeof(ipOfServer))<0)
    {
        printf("Connection failed due to port and ip problems\n");
        return 1;
    }
    char file_name[FILENAME_MAX+1]={"file"};
    char buffer[BUFFER_SIZE];
    bzero(buffer,BUFFER_SIZE);
```

```
    strncpy(buffer,file_name,strlen(file_name)>BUFFER_SIZE?
strlen(file_name):BUFFER_SIZE);
    if (send(CreateSocket,buffer,BUFFER_SIZE,0)<0) {
        perror("Send File Name Failed:");
        exit(1);
    }
    FILE *fp = fopen(file_name,"a+x");
    if(!fp) {
        printf("Can not open file:\t %s",file_name);
        exit(1);
    }
    bzero(buffer,BUFFER_SIZE);
    clock_t  start = clock();
    int length = 0;
    while((length = recv(CreateSocket,buffer,BUFFER_SIZE,0))>0) {
        if(fwrite(buffer,sizeof(char),length,fp)<length) {
            printf("Failed\n");
            break;
        }
        bzero(buffer,BUFFER_SIZE);
    }
    printf("Receive done\n");
    fclose(fp);
    printf("TIME COST %ld ms",(clock()-start)/1000);
    close(CreateSocket);
    return 0;
}

```

This client will establish a connection with the server and requests `file`. It will throw a exception with `connection error and file requests error`.

To run the client:

```
#! /usr/bin/bash
g++ client.cc -o client
./client
```

Test and Results

using mininet to create a net topology below:

```
#!/usr/bin/python

"""
Simple example of setting network and CPU parameters
"""


from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import OVSBridge
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
```

```
from mininet.log import setLogLevel, info
from mininet.cli import CLI

from sys import argv

# It would be nice if we didn't have to do this:
# pylint: disable=arguments-differ

class SingleSwitchTopo( Topo ):
    def build( self ):
        tracker = self.addHost('tracker', cpu=.25)
        switch1 = self.addSwitch('s1',stp=True)
        host1 = self.addHost('h1', cpu=.25)
        host2 = self.addHost('h2', cpu=.25)
        host3 = self.addHost('h3', cpu=.25)
        host4 = self.addHost('h4', cpu=.25)
        host5 = self.addHost('h5', cpu=.25)
        host6 = self.addHost('h6', cpu=.25)
        self.addLink(tracker, switch1, delay='5ms', loss=0, use_htb=True)
        self.addLink(host1, switch1, delay='5ms', loss=0, use_htb=True)
        self.addLink(host2, switch1,  delay='5ms', loss=0, use_htb=True)
        self.addLink(host3, switch1,  delay='5ms', loss=0, use_htb=True)
        self.addLink(host4, switch1, delay='5ms', loss=0, use_htb=True)
        self.addLink(host5, switch1,  delay='5ms', loss=0, use_htb=True)
        self.addLink(host6, switch1,  delay='5ms', loss=0, use_htb=True)

def Test():
    "Create network and run simple performance test"
    topo = SingleSwitchTopo()
    net = Mininet( topo=topo,
                   host=CPULimitedHost, link=TCLink,
                   autoStaticArp=False )
    net.start()
    info( "Dumping host connections\n" )
    dumpNodeConnections(net.hosts)
    tracker,h1,h2,h3,h4,h5,h6 =
net.getNodeByName('tracker','h1','h2','h3','h4','h5','h6')
    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    # Prevent test_simpleperf from failing due to packet loss
    Test()

```

Result for one connection

File Size	1st download cost	2nd download cost	3rd download cost	avg download cost	avg speed
1GB	3790ms	4156ms	4141ms	4029ms	254.15MB/s

Result for two connection

hostname	download cost(ms)	avg cost(ms)	avg speed(MB/s)
h1	3801 3837 3667	3768.33	271.76
h2	4573 3864 3828	4088.33	250.48

Result for four connection

hostname	download cost(ms)	avg cost(ms)	avg speed(MB/s)
h1	3539 3409 3991	3646.33	280.85
h2	3906 3630 3716	3750.67	273.06
h3	3527 3557 3875	3653	280.31
h4	3813 3711 3397	3640.33	281.32

Result for six connection

###

hostname	download cost(ms)	avg cost(ms)	avg speed(MB/s)
h1	3048 3590 3335	3324.33	308.03
h2	3606 2868 3177	3217	318.31
h3	3503 3242 3297	3347	305.94
h4	3526 31513672	3440	297.67
h5	4337 3346 3095	3592	285.07
h6	3171 3026 3068	388.33	263.69

P2P Model

code for tracker

```
#include <iostream>
#include <string.h>
#include <vector>
#include <pthread.h>
#include <stdio.h>  // standard input and output library
#include <stdlib.h> // this includes functions regarding memory allocation
#include <string.h> // contains string functions
#include <errno.h> //It defines macros for reporting and retrieving error conditions through error codes
#include <time.h> //contains various functions for manipulating date and time
#include <unistd.h> //contains various constants
```

```
#include <sys/types.h> //contains a number of basic derived types that should be used whenever appropriate
#include <arpa/inet.h> // defines in_addr structure
#include <sys/socket.h> // for socket creation
#include <netinet/in.h> //contains constants and structures needed for internet domain addresses
#include <sys/file.h>
#include <fstream>
#include <sstream>
//pthread_mutex_t mutex;
using namespace std;
const int BUFFER_SIZE = 1024;
void sendfile(void *socket_fd) {
    int flock;
    int fd = *((int *) socket_fd);
    char buffer[BUFFER_SIZE];
    bzero(buffer,BUFFER_SIZE);
    if(recv(fd,buffer,BUFFER_SIZE,0)<0) {
        perror("Server Receive Data Failed!");
    }
    char file_name[FILENAME_MAX+1];
    bzero(file_name,FILENAME_MAX+1);
    strncpy(file_name,buffer,strlen(buffer)>FILENAME_MAX?FILENAME_MAX:strlen(buffer));
    printf("%s\n",file_name);

    FILE *fp = fopen(file_name,"r+w");
    if(!fp) {
        printf("File:%s Not Found!\n",file_name);
    }
    else {
        bzero(buffer,BUFFER_SIZE);
        int length = 0;
        while((length=fread(buffer,sizeof(char),BUFFER_SIZE,fp))>0) {
            if(send(fd,buffer,length,0)<0){
                printf("Send File:%s.\n",file_name);
                break;
            }
            bzero(buffer,BUFFER_SIZE);
        }
        fclose(fp);
        /*struct sockaddr_in sa;
        int len = sizeof(sa);
        pthread_mutex_lock(&mutex);
        if(!getpeername(fd, (struct sockaddr *)&sa, reinterpret_cast<socklen_t *>(&len)))
        {
            printf("%s:%d received file %s !\n",inet_ntoa(sa.sin_addr),ntohs(sa.sin_port),file_name);
        }
        ofstream outFile;
        outFile.open(file_name, ios::app); // 打开模式可省略
        outFile << inet_ntoa(sa.sin_addr) << ',' << ntohs(sa.sin_port) << ',' << file_name << endl;
        outFile.close();
        pthread_mutex_unlock(&mutex);
        printf("File:%s Transfer Successful!\n",file_name);*/
    }
```

```
    close(fd);

}
int main() {
    time_t clock;
    char dataSending[1025]; // Actually this is called packet in Network Communication, which contain data and send through.
    int clintListn = 0, clintConnt = 0;
    struct sockaddr_in ipOfServer;
    clintListn = socket(AF_INET, SOCK_STREAM, 0); // creating socket
    memset(&ipOfServer, '0', sizeof(ipOfServer));
    memset(dataSending, '0', sizeof(dataSending));
    ipOfServer.sin_family = AF_INET;
    ipOfServer.sin_addr.s_addr = htonl(INADDR_ANY);
    ipOfServer.sin_port = htons(2017);      // this is the port number of running server
    bind(clintListn, (struct sockaddr*)&ipOfServer , sizeof(ipOfServer));
    listen(clintListn , 20);

    while(1)
    {
        pthread_t thread_id;
        printf("Hi,Iam the tracker.Some Client hit me\n"); // whenever a request from client came. It will be processed here.
        clintConnt = accept(clintListn, (struct sockaddr*)NULL, NULL);
        if(clintConnt==-1) {
            fprintf(stderr,"Accept error!\n");
            continue;
        }
        struct sockaddr_in sa;
        int len = sizeof(sa);

        if(!getpeername(clintConnt, (struct sockaddr *)&sa, reinterpret_cast<socklen_t *>(&len)))
        {
            printf("New connection from %s:%d !\n",inet_ntoa(sa.sin_addr),ntohs(sa.sin_port));
        }


        if(pthread_create(&thread_id, NULL, reinterpret_cast<void *(*)(void *)>(sendfile), (void *)(&clintConnt))) {
            fprintf(stderr,"pthread create error");
            break;
        }
    }
    return 0;
}

```

The tracker will listen the peer's request, return the seed torrent.

usage:

```
#! /usr/bin/bash
g++ -pthread tracker.cpp -o tracker
./tracker
```

code for peer uploader

```
//
// Created by austinguish on 2020/10/12.
//
#include <iostream>
#include <string.h>
#include <vector>
#include <stdio.h>  // standard input and output library
#include <stdlib.h> // this includes functions regarding memory allocation
#include <string.h> // contains string functions
#include <errno.h> //It defines macros for reporting and retrieving error conditions through error codes
#include <time.h> //contains various functions for manipulating date and time
#include <unistd.h> //contains various constants
#include <sys/types.h> //contains a number of basic derived types that should be used whenever appropriate
#include <arpa/inet.h> // defines in_addr structure
#include <sys/socket.h> // for socket creation
#include <netinet/in.h> //contains constants and structures needed for internet domain addresses
#include <sys/file.h>
#include <fstream>
#include <sstream>
#include <sys/stat.h>
#include <thread>
using namespace std;
const string TORRENT_NAME = "seed.torrent";
const string TRACKER_IP = "10.0.0.7";
const string TRACKER_PORT = "2017";
const int BUFFER_SIZE=1024;
inline bool is_exist (const std::string& name) {
    struct stat buffer;
    return (stat (name.c_str(), &buffer) == 0);
}
void sendfile(void *socket_fd) {
    int fd = *((int *) socket_fd);
    char buffer[BUFFER_SIZE];
    bzero(buffer,BUFFER_SIZE);
    if(recv(fd,buffer,BUFFER_SIZE,0)<0) {
        perror("Server Receive Data Failed!");
    }
    char file_name[FILENAME_MAX+1];
    bzero(file_name,FILENAME_MAX+1);
    strncpy(file_name,buffer,strlen(buffer)>FILENAME_MAX?FILENAME_MAX:strlen(buffer));
    printf("%s\n",file_name);
    FILE *fp = fopen(file_name,"r");
    if(!fp) {
        printf("File:%s Not Found!",file_name);
    }
    else {
        bzero(buffer,BUFFER_SIZE);
        int length = 0;
        long sendsize = 0;
        while((length=fread(buffer,sizeof(char),BUFFER_SIZE,fp))>0) {
```

```
            sendsize+=BUFFER_SIZE;
            if(send(fd,buffer,length,0)<0){
                printf("Send File:%s.\n",file_name);
                break;
            }
            bzero(buffer,BUFFER_SIZE);
        }
        fclose(fp);
        cout<<"send"<<sendsize<<endl;
        printf("File:%s Transfer Successful!\n",file_name);
    }
    close(fd);

}
void sub_downloader(const string fn, string dest_ip, string dest_port){
    cout<<dest_ip<<endl;
    cout<<dest_port<<endl;
    int CreateSocket = 0;
    char dataReceived[1024];
    struct sockaddr_in ipOfServer;
    stringstream strValue;
    strValue << dest_port;
    unsigned int portValue;
    strValue >> portValue;
    memset(dataReceived, '0' ,sizeof(dataReceived));

    if((CreateSocket = socket(AF_INET, SOCK_STREAM, 0))< 0)
    {
        printf("Socket not created \n");
        exit(0);
    }
    ipOfServer.sin_family = AF_INET;
    ipOfServer.sin_port = htons(portValue);
    ipOfServer.sin_addr.s_addr = inet_addr(dest_ip.c_str());

    if(connect(CreateSocket, (struct sockaddr *)&ipOfServer,
sizeof(ipOfServer))<0)
    {
        printf("Connection failed due to port and ip problems\n");
    }
    const char *file_name = fn.c_str();
    char buffer[BUFFER_SIZE];
    bzero(buffer,BUFFER_SIZE);
    strncpy(buffer,file_name,strlen(file_name)>BUFFER_SIZE?
strlen(file_name):BUFFER_SIZE);
    if (send(CreateSocket,buffer,BUFFER_SIZE,0)<0) {
        perror("Send File Name Failed:");
        exit(1);
    }
    ofstream outFile;
    outFile.open(file_name); // 打开模式可省略
    bzero(buffer,BUFFER_SIZE);
    int length = 0;
    while((length = recv(CreateSocket,buffer,BUFFER_SIZE,0))>0) {
        outFile<<buffer;
        bzero(buffer,BUFFER_SIZE);
    }
    printf("Receive done\n");
```

```
    outFile.close();
    close(CreateSocket);
}
void downloader(){
    int download_cnt = 0;
    ifstream inFile("seed.torrent", ios::in);
    string lineStr;
    vector<vector<string>> strArray;
    while (getline(inFile, lineStr))
    {
        stringstream ss(lineStr);
        string str;
        vector<string> lineArray;
        // 按照逗号分隔
        while (getline(ss, str, ','))
            lineArray.push_back(str);
        strArray.push_back(lineArray);
    }
    thread downloadThreads[5];
    for (auto i : strArray)
    {
        if(is_exist(i[0])) continue;else {
            downloadThreads[download_cnt] =
thread(sub_downloader,i[0],i[1],i[2]);
            downloadThreads[download_cnt].join();
        }
        ++download_cnt;

    }

}
void share(int port_num){
    time_t clock;
    char dataSending[1025]; // Actually this is called packet in Network
Communication, which contain data and send through.
    int clintListn = 0, clintConnt = 0;
    struct sockaddr_in ipOfServer;
    clintListn = socket(AF_INET, SOCK_STREAM, 0); // creating socket
    memset(&ipOfServer, '0', sizeof(ipOfServer));
    memset(dataSending, '0', sizeof(dataSending));
    ipOfServer.sin_family = AF_INET;
    ipOfServer.sin_addr.s_addr = htonl(INADDR_ANY);
    ipOfServer.sin_port = htons(port_num);      // this is the port number
of running server
    bind(clintListn, (struct sockaddr*)&ipOfServer , sizeof(ipOfServer));
    listen(clintListn , 20);

    while(1)
    {
        pthread_t thread_id;
        printf("Hi,Iam the tracker.Some Client hit me\n"); // whenever a
request from client came. It will be processed here.
        clintConnt = accept(clintListn, (struct sockaddr*)NULL, NULL);
        if(clintConnt==-1) {
            fprintf(stderr,"Accept error!\n");
            continue;
        }
        struct sockaddr_in sa;
```

```
        int len = sizeof(sa);

        if(!getpeername(clintConnt, (struct sockaddr *)&sa,
reinterpret_cast<socklen_t *>(&len)))
        {
            printf("New connection from %s:%d
!\n",inet_ntoa(sa.sin_addr),ntohs(sa.sin_port));
        }


        if(pthread_create(&thread_id, NULL, reinterpret_cast<void *(*)(void
*)>(sendfile), (void *)(&clintConnt))) {
            fprintf(stderr,"pthread create error");
            break;
        }
    }


}
int main (int argc,char *argv[]){
    sub_downloader(TORRENT_NAME,TRACKER_IP,TRACKER_PORT);
    stringstream strValue;
    strValue << argv[1];
    unsigned int portValue;
    strValue >> portValue;
    cout<<portValue;
    thread upload(share,portValue);
    upload.join();
};

```

the peer will request seed.torrent from the tracker, and then wait for other peer's connection. Send the

file to requests

```
#! /usr/bin/bash
make clean
make all
make copy
```

code for peerdownloader

```
//
// Created by austinguish on 2020/10/12.
//
#include <iostream>
#include <string.h>
#include <vector>
#include <stdio.h>  // standard input and output library
#include <stdlib.h> // this includes functions regarding memory allocation
#include <string.h> // contains string functions
#include <errno.h> //It defines macros for reporting and retrieving error
conditions through error codes
```

```
#include <time.h> //contains various functions for manipulating date and time
#include <unistd.h> //contains various constants
#include <sys/types.h> //contains a number of basic derived types that should be used whenever appropriate
#include <arpa/inet.h> // defines in_addr structure
#include <sys/socket.h> // for socket creation
#include <netinet/in.h> //contains constants and structures needed for internet domain addresses
#include <sys/file.h>
#include <fstream>
#include <sstream>
#include <sys/stat.h>
#include <thread>
using namespace std;
const string TORRENT_NAME = "seed.torrent";
const string TRACKER_IP = "10.0.0.7";
const string TRACKER_PORT = "2017";
const int BUFFER_SIZE=1024;
inline bool is_exist (const std::string& name) {
    struct stat buffer;
    return (stat (name.c_str(), &buffer) == 0);
}
void sub_downloader(const string fn, string dest_ip, string dest_port){
    cout<<dest_ip<<endl;
    cout<<dest_port<<endl;
    clock_t start = clock();
    int CreateSocket = 0;
    char dataReceived[1024];
    struct sockaddr_in ipOfServer;
    stringstream strValue;
    strValue << dest_port;
    unsigned int portValue;
    strValue >> portValue;
    memset(dataReceived, '0' ,sizeof(dataReceived));

    if((CreateSocket = socket(AF_INET, SOCK_STREAM, 0))< 0)
    {
        printf("Socket not created \n");
        exit(0);

    }
    ipOfServer.sin_family = AF_INET;
    ipOfServer.sin_port = htons(portValue);
    ipOfServer.sin_addr.s_addr = inet_addr(dest_ip.c_str());

    if(connect(CreateSocket, (struct sockaddr *)&ipOfServer, sizeof(ipOfServer))<0)
    {
        printf("Connection failed due to port and ip problems\n");
    }
    const char *file_name = fn.c_str();
    char buffer[BUFFER_SIZE];
    bzero(buffer,BUFFER_SIZE);
    strncpy(buffer,file_name,strlen(file_name)>BUFFER_SIZE? strlen(file_name):BUFFER_SIZE);
    if (send(CreateSocket,buffer,BUFFER_SIZE,0)<0) {
        perror("Send File Name Failed:");
```

```
        exit(1);
    }
    FILE *fp = fopen(file_name, "wb+");
    if (fp == NULL) {
        printf("File open error");
        return;
    }
    bzero(buffer,BUFFER_SIZE);
    int length = 0;
    long recvlength = 0;
    while((length = recv(CreateSocket,buffer,BUFFER_SIZE,0))>0) {
        if(fwrite(buffer,sizeof(char),length,fp)<length) {
            printf("Failed\n");
            break;
        }
        recvlength+=BUFFER_SIZE;
        bzero(buffer,BUFFER_SIZE);
    }
    fclose(fp);
    cout<<"rev"<<recvlength<<endl;
    printf("Receive done\n");
    close(CreateSocket);
    cout<<"Time cost:"<<clock()-start<<endl;
}
void downloader(){
    int download_cnt = 0;
    ifstream inFile("seed.torrent", ios::in);
    string lineStr;
    vector<vector<string>> strArray;
    while (getline(inFile, lineStr))
    {
        stringstream ss(lineStr);
        string str;
        vector<string> lineArray;
        // 按照逗号分隔
        while (getline(ss, str, ','))
            lineArray.push_back(str);
        strArray.push_back(lineArray);
    }
    thread downloadThreads[5];
    for (auto i : strArray)
    {
        if(is_exist(i[0])) continue;else {
            downloadThreads[download_cnt] =
thread(sub_downloader,i[0],i[1],i[2]);
            downloadThreads[download_cnt].join();
        }
        ++download_cnt;

    }

}

int main (int argc,char *argv[]){
    sub_downloader(TORRENT_NAME,TRACKER_IP,TRACKER_PORT);
    thread download(downloader);
    download.join();
};
```

```
121
```

it will read the torrent, connect to the server who saved the part file, and download it. It will create sub_thread to download the fro each server.

result

hostname	total download_time(ms)	total_size(MB)	Avg_speed(MB/s)
h1	2617.073	853.33	326.07
h2	2486.907	853.33	343.12
h3	2440.584	853.33	349.73
h4	2672.707	853.33	319.24
h5	2844.547	853.33	299.94
h6	2675.27	853.33	319.00

conclusion

1. in client-server model with client adding in the transportation queue, the speed decreased and the latency for server to answer the client increased
2. in p2p model,with the peer added, the download speed has increased due to the file distributed storage. The main server's workload decreased and the downloader can download at a very high speed, 14% faster than client-server model.