

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

```
1  #include <stdio.h> // standard input and output library
2  #include <stdlib.h> // this includes functions regarding memory allocation
3  #include <string.h> // contains string functions
4  #include <errno.h> //It defines macros for reporting and retrieving error
   conditions through error codes
5  #include <time.h> //contains various functions for manipulating date and time
6  #include <unistd.h> //contains various constants
7  #include <sys/types.h> //contains a number of basic derived types that should
   be used whenever appropriate
8  #include <arpa/inet.h> // defines in_addr structure
9  #include <sys/socket.h> // for socket creation
10 #include <netinet/in.h> //contains constants and structures needed for
   internet domain addresses
11 #include <pthread.h>
12 const int BUFFER_SIZE = 1024;
13 void sendfile(void *socket_fd) {
14     int fd = *((int *) socket_fd);
15     char buffer[BUFFER_SIZE];
16     bzero(buffer, BUFFER_SIZE);
17     if(recv(fd, buffer, BUFFER_SIZE, 0) < 0) {
18         perror("Server Receive Data Failed!");
19     }
20     char file_name[FILENAME_MAX+1];
21     bzero(file_name, FILENAME_MAX+1);
22     strncpy(file_name, buffer, strlen(buffer) > FILENAME_MAX?
   FILENAME_MAX:strlen(buffer));
23     printf("%s\n", file_name);
24     FILE *fp = fopen(file_name, "r");
25     if(!fp) {
26         printf("File:%s Not Found!", file_name);
27     }
28     else {
29         bzero(buffer, BUFFER_SIZE);
30         int length = 0;
```

```

31         while((length=fread(buffer, sizeof(char), BUFFER_SIZE, fp))>0) {
32             if(send(fd, buffer, length, 0)<0){
33                 printf("Send File:%s.\n", file_name);
34                 break;
35             }
36             bzero(buffer, BUFFER_SIZE);
37         }
38         fclose(fp);
39         printf("File:%s Transfer Successful!\n", file_name);
40     }
41     close(fd);
42
43 }
44 int main()
45 {
46     time_t clock;
47     char dataSending[1025]; // Actually this is called packet in Network
Communication, which contain data and send through.
48     int clintListn = 0, clintConnt = 0;
49     struct sockaddr_in ipOfServer;
50     clintListn = socket(AF_INET, SOCK_STREAM, 0); // creating socket
51     memset(&ipOfServer, '0', sizeof(ipOfServer));
52     memset(dataSending, '0', sizeof(dataSending));
53     ipOfServer.sin_family = AF_INET;
54     ipOfServer.sin_addr.s_addr = htonl(INADDR_ANY);
55     ipOfServer.sin_port = htons(2017); // this is the port number of
running server
56     bind(clintListn, (struct sockaddr*)&ipOfServer, sizeof(ipOfServer));
57     listen(clintListn, 20);
58
59     while(1)
60     {
61         pthread_t thread_id;
62         printf("Hi,Iam running server.Some Client hit me\n"); // whenever a
request from client came. It will be processed here.
63         clintConnt = accept(clintListn, (struct sockaddr*)NULL, NULL);
64         if(clintConnt==-1) {
65             fprintf(stderr, "Accept error!\n");
66             continue;
67         }
68         struct sockaddr_in sa;
69         int len = sizeof(sa);
70
71         if(!getpeername(clintConnt, (struct sockaddr *)&sa,
reinterpret_cast<socklen_t *>(&len)))
72         {
73             printf("New connection from %s:%d
!\n", inet_ntoa(sa.sin_addr), ntohs(sa.sin_port));
74         }
75
76         if(pthread_create(&thread_id, NULL, reinterpret_cast<void *(*)(void
*)>(sendfile), (void *)&clintConnt)) {
77             fprintf(stderr, "pthread create error");
78             break;
79         }
80     }
81     return 0;
82 }

```

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`

```
1  #! /usr/bin/bash
2  g++ -pthread server.cc -o server
3  ./server
```

code for Clients

```
1  #include <sys/socket.h>
2  #include <sys/types.h>
3  #include <netinet/in.h>
4  #include <netdb.h>
5  #include <stdio.h>
6  #include <string.h>
7  #include <stdlib.h>
8  #include <unistd.h>
9  #include <errno.h>
10 #include <time.h>
11 #include <arpa/inet.h>
12 const int BUFFER_SIZE=1024;
13 int main()
14 {
15     int CreateSocket = 0,n = 0;
16     char dataReceived[1024];
17     struct sockaddr_in ipOfServer;
18
19     memset(dataReceived, '0' ,sizeof(dataReceived));
20
21     if((CreateSocket = socket(AF_INET, SOCK_STREAM, 0))< 0)
22     {
23         printf("Socket not created \n");
24         return 1;
25     }
26
27     ipOfServer.sin_family = AF_INET;
28     ipOfServer.sin_port = htons(2587);
29     ipOfServer.sin_addr.s_addr = inet_addr("10.0.0.1");
30
31     if(connect(CreateSocket, (struct sockaddr *)&ipOfServer,
32 sizeof(ipOfServer))<0)
33     {
34         printf("Connection failed due to port and ip problems\n");
35         return 1;
36     }
37     char file_name[FILENAME_MAX+1]="file";
38     char buffer[BUFFER_SIZE];
39     bzero(buffer,BUFFER_SIZE);
```

```

39     strncpy(buffer, file_name, strlen(file_name)>BUFFER_SIZE?
strlen(file_name):BUFFER_SIZE);
40     if (send(CreateSocket, buffer, BUFFER_SIZE, 0)<0) {
41         perror("Send File Name Failed:");
42         exit(1);
43     }
44     FILE *fp = fopen(file_name, "a+x");
45     if(!fp) {
46         printf("Can not open file:\t %s", file_name);
47         exit(1);
48     }
49     bzero(buffer, BUFFER_SIZE);
50     clock_t start = clock();
51     int length = 0;
52     while((length = recv(CreateSocket, buffer, BUFFER_SIZE, 0))>0) {
53         if(fwrite(buffer, sizeof(char), length, fp)<length) {
54             printf("Failed\n");
55             break;
56         }
57         bzero(buffer, BUFFER_SIZE);
58     }
59     printf("Receive done\n");
60     fclose(fp);
61     printf("TIME COST %ld ms", (clock()-start)/1000);
62     close(CreateSocket);
63     return 0;
64 }
65

```

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:

```

1  #!/usr/bin/bash
2  g++ client.cc -o client
3  ./client

```

Test and Results

using mininet to create a net topology below:

```

1  #!/usr/bin/python
2
3  """
4  Simple example of setting network and CPU parameters
5  """
6
7
8  from mininet.topo import Topo
9  from mininet.net import Mininet
10 from mininet.node import OVSBridge
11 from mininet.node import CPULimitedHost
12 from mininet.link import TCLink
13 from mininet.util import dumpNodeConnections

```

```

14 from mininet.log import setLogLevel, info
15 from mininet.cli import CLI
16
17 from sys import argv
18
19 # It would be nice if we didn't have to do this:
20 # pylint: disable=arguments-differ
21
22 class SingleSwitchTopo( Topo ):
23     def build( self ):
24         tracker = self.addHost('tracker', cpu=.25)
25         switch1 = self.addSwitch('s1', stp=True)
26         host1 = self.addHost('h1', cpu=.25)
27         host2 = self.addHost('h2', cpu=.25)
28         host3 = self.addHost('h3', cpu=.25)
29         host4 = self.addHost('h4', cpu=.25)
30         host5 = self.addHost('h5', cpu=.25)
31         host6 = self.addHost('h6', cpu=.25)
32         self.addLink(tracker, switch1, delay='5ms', loss=0, use_htb=True)
33         self.addLink(host1, switch1, delay='5ms', loss=0, use_htb=True)
34         self.addLink(host2, switch1, delay='5ms', loss=0, use_htb=True)
35         self.addLink(host3, switch1, delay='5ms', loss=0, use_htb=True)
36         self.addLink(host4, switch1, delay='5ms', loss=0, use_htb=True)
37         self.addLink(host5, switch1, delay='5ms', loss=0, use_htb=True)
38         self.addLink(host6, switch1, delay='5ms', loss=0, use_htb=True)
39
40     def Test():
41         "Create network and run simple performance test"
42         topo = SingleSwitchTopo()
43         net = Mininet( topo=topo,
44                       host=CPULimitedHost, link=TCLink,
45                       autoStaticArp=False )
46         net.start()
47         info( "Dumping host connections\n" )
48         dumpNodeConnections(net.hosts)
49         tracker, h1, h2, h3, h4, h5, h6 =
50         net.getNodeByName('tracker', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6')
51         CLI(net)
52         net.stop()
53
54 if __name__ == '__main__':
55     setLogLevel( 'info' )
56     # Prevent test_simpleperf from failing due to packet loss
57     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

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

```

11 #include <sys/types.h> //contains a number of basic derived types that
    should be used whenever appropriate
12 #include <arpa/inet.h> // defines in_addr structure
13 #include <sys/socket.h> // for socket creation
14 #include <netinet/in.h> //contains constants and structures needed for
    internet domain addresses
15 #include <sys/file.h>
16 #include <fstream>
17 #include <sstream>
18 //pthread_mutex_t mutex;
19 using namespace std;
20 const int BUFFER_SIZE = 1024;
21 void sendfile(void *socket_fd) {
22     int flock;
23     int fd = *((int *) socket_fd);
24     char buffer[BUFFER_SIZE];
25     bzero(buffer, BUFFER_SIZE);
26     if(recv(fd, buffer, BUFFER_SIZE, 0) < 0) {
27         perror("Server Receive Data Failed!");
28     }
29     char file_name[FILENAME_MAX+1];
30     bzero(file_name, FILENAME_MAX+1);
31     strncpy(file_name, buffer, strlen(buffer) > FILENAME_MAX?
FILENAME_MAX:strlen(buffer));
32     printf("%s\n", file_name);
33
34     FILE *fp = fopen(file_name, "r+w");
35     if(!fp) {
36         printf("File:%s Not Found!\n", file_name);
37     }
38     else {
39         bzero(buffer, BUFFER_SIZE);
40         int length = 0;
41         while((length=fread(buffer, sizeof(char), BUFFER_SIZE, fp)) > 0) {
42             if(send(fd, buffer, length, 0) < 0){
43                 printf("Send File:%s.\n", file_name);
44                 break;
45             }
46             bzero(buffer, BUFFER_SIZE);
47         }
48         fclose(fp);
49         /*struct sockaddr_in sa;
50         int len = sizeof(sa);
51         pthread_mutex_lock(&mutex);
52         if(!getpeername(fd, (struct sockaddr *)&sa,
reinterpret_cast<socklen_t *>(&len)))
53         {
54             printf("%s:%d received file %s
!\n", inet_ntoa(sa.sin_addr), ntohs(sa.sin_port), file_name);
55         }
56         ofstream outFile;
57         outFile.open(file_name, ios::app); // 打开模式可省略
58         outFile << inet_ntoa(sa.sin_addr) << ',' << ntohs(sa.sin_port) <<
', ' << file_name << endl;
59         outFile.close();
60         pthread_mutex_unlock(&mutex);
61         printf("File:%s Transfer Successful!\n", file_name);*/
62     }
}

```

```

63     close(fd);
64
65 }
66 int main() {
67     time_t clock;
68     char dataSending[1025]; // Actually this is called packet in Network
    Communication, which contain data and send through.
69     int clintListn = 0, clintConnt = 0;
70     struct sockaddr_in ipOfServer;
71     clintListn = socket(AF_INET, SOCK_STREAM, 0); // creating socket
72     memset(&ipOfServer, '0', sizeof(ipOfServer));
73     memset(dataSending, '0', sizeof(dataSending));
74     ipOfServer.sin_family = AF_INET;
75     ipOfServer.sin_addr.s_addr = htonl(INADDR_ANY);
76     ipOfServer.sin_port = htons(2017); // this is the port number of
    running server
77     bind(clintListn, (struct sockaddr*)&ipOfServer, sizeof(ipOfServer));
78     listen(clintListn, 20);
79
80     while(1)
81     {
82         pthread_t thread_id;
83         printf("Hi,Iam the tracker.Some Client hit me\n"); // whenever a
    request from client came. It will be processed here.
84         clintConnt = accept(clintListn, (struct sockaddr*)NULL, NULL);
85         if(clintConnt==-1) {
86             fprintf(stderr,"Accept error!\n");
87             continue;
88         }
89         struct sockaddr_in sa;
90         int len = sizeof(sa);
91
92         if(!getpeername(clintConnt, (struct sockaddr *)&sa,
    reinterpret_cast<socklen_t *>(&len)))
93         {
94             printf("New connection from %s:%d
    !\n",inet_ntoa(sa.sin_addr),ntohs(sa.sin_port));
95         }
96
97
98         if(pthread_create(&thread_id, NULL, reinterpret_cast<void *(*)(void
    *)>(sendfile), (void *)(&clintConnt))) {
99             fprintf(stderr,"pthread create error");
100             break;
101         }
102     }
103     return 0;
104 }
105

```

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

usage:

```

1  #! /usr/bin/bash
2  g++ -pthread tracker.cpp -o tracker
3  ./tracker

```


code for peer uploader

```
1  //
2  // Created by austinguish on 2020/10/12.
3  //
4  #include <iostream>
5  #include <string.h>
6  #include <vector>
7  #include <stdio.h> // standard input and output library
8  #include <stdlib.h> // this includes functions regarding memory allocation
9  #include <string.h> // contains string functions
10 #include <errno.h> //It defines macros for reporting and retrieving error
    conditions through error codes
11 #include <time.h> //contains various functions for manipulating date and
    time
12 #include <unistd.h> //contains various constants
13 #include <sys/types.h> //contains a number of basic derived types that
    should be used whenever appropriate
14 #include <arpa/inet.h> // defines in_addr structure
15 #include <sys/socket.h> // for socket creation
16 #include <netinet/in.h> //contains constants and structures needed for
    internet domain addresses
17 #include <sys/file.h>
18 #include <fstream>
19 #include <sstream>
20 #include <sys/stat.h>
21 #include <thread>
22 using namespace std;
23 const string TORRENT_NAME = "seed.torrent";
24 const string TRACKER_IP = "10.0.0.7";
25 const string TRACKER_PORT = "2017";
26 const int BUFFER_SIZE=1024;
27 inline bool is_exist (const std::string& name) {
28     struct stat buffer;
29     return (stat (name.c_str(), &buffer) == 0);
30 }
31 void sendfile(void *socket_fd) {
32     int fd = *((int *) socket_fd);
33     char buffer[BUFFER_SIZE];
34     bzero(buffer,BUFFER_SIZE);
35     if(recv(fd,buffer,BUFFER_SIZE,0)<0) {
36         perror("Server Receive Data Failed!");
37     }
38     char file_name[FILENAME_MAX+1];
39     bzero(file_name,FILENAME_MAX+1);
40     strncpy(file_name,buffer,strlen(buffer)>FILENAME_MAX?
    FILENAME_MAX:strlen(buffer));
41     printf("%s\n",file_name);
42     FILE *fp = fopen(file_name,"r");
43     if(!fp) {
44         printf("File:%s Not Found!",file_name);
45     }
46     else {
47         bzero(buffer,BUFFER_SIZE);
48         int length = 0;
49         long sendsize = 0;
50         while((length=fread(buffer,sizeof(char),BUFFER_SIZE,fp))>0) {
```

```

51         sendsize+=BUFFER_SIZE;
52         if(send(fd,buffer,length,0)<0){
53             printf("Send File:%s.\n",file_name);
54             break;
55         }
56         bzero(buffer,BUFFER_SIZE);
57     }
58     fclose(fp);
59     cout<<"send"<<sendsize<<endl;
60     printf("File:%s Transfer Successful!\n",file_name);
61 }
62 close(fd);
63
64 }
65 void sub_downloader(const string fn, string dest_ip, string dest_port){
66     cout<<dest_ip<<endl;
67     cout<<dest_port<<endl;
68     int CreateSocket = 0;
69     char dataReceived[1024];
70     struct sockaddr_in ipOfServer;
71     stringstream strValue;
72     strValue << dest_port;
73     unsigned int portValue;
74     strValue >> portValue;
75     memset(dataReceived, '0' ,sizeof(dataReceived));
76
77     if((CreateSocket = socket(AF_INET, SOCK_STREAM, 0))< 0)
78     {
79         printf("Socket not created \n");
80         exit(0);
81     }
82     ipOfServer.sin_family = AF_INET;
83     ipOfServer.sin_port = htons(portValue);
84     ipOfServer.sin_addr.s_addr = inet_addr(dest_ip.c_str());
85
86     if(connect(CreateSocket, (struct sockaddr *)&ipOfServer,
87 sizeof(ipOfServer))<0)
88     {
89         printf("Connection failed due to port and ip problems\n");
90     }
91     const char *file_name = fn.c_str();
92     char buffer[BUFFER_SIZE];
93     bzero(buffer,BUFFER_SIZE);
94     strncpy(buffer,file_name,strlen(file_name)>BUFFER_SIZE?
95 strlen(file_name):BUFFER_SIZE);
96     if (send(CreateSocket,buffer,BUFFER_SIZE,0)<0) {
97         perror("Send File Name Failed:");
98         exit(1);
99     }
100     ofstream outFile;
101     outFile.open(file_name); // 打开模式可省略
102     bzero(buffer,BUFFER_SIZE);
103     int length = 0;
104     while((length = recv(CreateSocket,buffer,BUFFER_SIZE,0))>0) {
105         outFile<<buffer;
106         bzero(buffer,BUFFER_SIZE);
107     }
108     printf("Receive done\n");

```

```

107     outFile.close();
108     close(CreateSocket);
109 }
110 void downloader(){
111     int download_cnt = 0;
112     ifstream inFile("seed.torrent", ios::in);
113     string lineStr;
114     vector<vector<string>> strArray;
115     while (getline(inFile, lineStr))
116     {
117         stringstream ss(lineStr);
118         string str;
119         vector<string> lineArray;
120         // 按照逗号分隔
121         while (getline(ss, str, ','))
122             lineArray.push_back(str);
123         strArray.push_back(lineArray);
124     }
125     thread downloadThreads[5];
126     for (auto i : strArray)
127     {
128         if(is_exist(i[0])) continue;else {
129             downloadThreads[download_cnt] =
130             thread(sub_downloader,i[0],i[1],i[2]);
131             downloadThreads[download_cnt].join();
132             ++download_cnt;
133         }
134     }
135 }
136 }
137 void share(int port_num){
138     time_t clock;
139     char dataSending[1025]; // Actually this is called packet in Network
140     // Communication, which contain data and send through.
141     int clintListn = 0, clintConnt = 0;
142     struct sockaddr_in ipOfServer;
143     clintListn = socket(AF_INET, SOCK_STREAM, 0); // creating socket
144     memset(&ipOfServer, '0', sizeof(ipOfServer));
145     memset(dataSending, '0', sizeof(dataSending));
146     ipOfServer.sin_family = AF_INET;
147     ipOfServer.sin_addr.s_addr = htonl(INADDR_ANY);
148     ipOfServer.sin_port = htons(port_num); // this is the port number
149     // of running server
150     bind(clintListn, (struct sockaddr*)&ipOfServer, sizeof(ipOfServer));
151     listen(clintListn, 20);
152     while(1)
153     {
154         pthread_t thread_id;
155         printf("Hi,Iam the tracker.Some Client hit me\n"); // whenever a
156         // request from client came. It will be processed here.
157         clintConnt = accept(clintListn, (struct sockaddr*)NULL, NULL);
158         if(clintConnt!=-1) {
159             fprintf(stderr,"Accept error!\n");
160             continue;
161         }
162         struct sockaddr_in sa;

```

```

161         int len = sizeof(sa);
162
163         if(!getpeername(clintConnt, (struct sockaddr *)&sa,
reinterpret_cast<socklen_t *>(&len)))
164         {
165             printf("New connection from %s:%d
!\n", inet_ntoa(sa.sin_addr), ntohs(sa.sin_port));
166         }
167
168
169         if(pthread_create(&thread_id, NULL, reinterpret_cast<void *(*)(void
*)>(sendfile), (void *)&clintConnt)) {
170             fprintf(stderr, "pthread create error");
171             break;
172         }
173     }
174
175
176 }
177
178 int main (int argc, char *argv[]){
179     sub_downloader(TORRENT_NAME, TRACKER_IP, TRACKER_PORT);
180     stringstream strValue;
181     strValue << argv[1];
182     unsigned int portValue;
183     strValue >> portValue;
184     cout<<portValue;
185     thread upload(share, portValue);
186     upload.join();
187 };

```

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

file to requests

```

1  #! /usr/bin/bash
2  make clean
3  make all
4  make copy

```

code for peerdownloader

```

1  //
2  // Created by austinguish on 2020/10/12.
3  //
4  #include <iostream>
5  #include <string.h>
6  #include <vector>
7  #include <stdio.h> // standard input and output library
8  #include <stdlib.h> // this includes functions regarding memory allocation
9  #include <string.h> // contains string functions
10 #include <errno.h> //It defines macros for reporting and retrieving error
    conditions through error codes

```

```

11 #include <time.h> //contains various functions for manipulating date and
    time
12 #include <unistd.h> //contains various constants
13 #include <sys/types.h> //contains a number of basic derived types that
    should be used whenever appropriate
14 #include <arpa/inet.h> // defines in_addr structure
15 #include <sys/socket.h> // for socket creation
16 #include <netinet/in.h> //contains constants and structures needed for
    internet domain addresses
17 #include <sys/file.h>
18 #include <fstream>
19 #include <sstream>
20 #include <sys/stat.h>
21 #include <thread>
22 using namespace std;
23 const string TORRENT_NAME = "seed.torrent";
24 const string TRACKER_IP = "10.0.0.7";
25 const string TRACKER_PORT = "2017";
26 const int BUFFER_SIZE=1024;
27 inline bool is_exist (const std::string& name) {
28     struct stat buffer;
29     return (stat (name.c_str(), &buffer) == 0);
30 }
31 void sub_downloader(const string fn, string dest_ip, string dest_port){
32     cout<<dest_ip<<endl;
33     cout<<dest_port<<endl;
34     clock_t start = clock();
35     int CreateSocket = 0;
36     char dataReceived[1024];
37     struct sockaddr_in ipOfServer;
38     stringstream strValue;
39     strValue << dest_port;
40     unsigned int portValue;
41     strValue >> portValue;
42     memset(dataReceived, '0' ,sizeof(dataReceived));
43
44     if((CreateSocket = socket(AF_INET, SOCK_STREAM, 0))< 0)
45     {
46         printf("Socket not created \n");
47         exit(0);
48     }
49     ipOfServer.sin_family = AF_INET;
50     ipOfServer.sin_port = htons(portValue);
51     ipOfServer.sin_addr.s_addr = inet_addr(dest_ip.c_str());
52
53     if(connect(CreateSocket, (struct sockaddr *)&ipOfServer,
54 sizeof(ipOfServer))<0)
55     {
56         printf("Connection failed due to port and ip problems\n");
57     }
58     const char *file_name = fn.c_str();
59     char buffer[BUFFER_SIZE];
60     bzero(buffer,BUFFER_SIZE);
61     strncpy(buffer,file_name,strlen(file_name)>BUFFER_SIZE?
    strlen(file_name):BUFFER_SIZE);
62     if (send(CreateSocket,buffer,BUFFER_SIZE,0)<0) {
63         perror("Send File Name Failed:");

```

```

64         exit(1);
65     }
66     FILE *fp = fopen(file_name, "wb+");
67     if (fp == NULL) {
68         printf("File open error");
69         return;
70     }
71     bzero(buffer, BUFFER_SIZE);
72     int length = 0;
73     long recvlength = 0;
74     while((length = recv(CreateSocket, buffer, BUFFER_SIZE, 0)) > 0) {
75         if(fwrite(buffer, sizeof(char), length, fp) < length) {
76             printf("Failed\n");
77             break;
78         }
79         recvlength += BUFFER_SIZE;
80         bzero(buffer, BUFFER_SIZE);
81     }
82     fclose(fp);
83     cout << "rev" << recvlength << endl;
84     printf("Receive done\n");
85     close(CreateSocket);
86     cout << "Time cost:" << clock() - start << endl;
87 }
88 void downloader(){
89     int download_cnt = 0;
90     ifstream inFile("seed.torrent", ios::in);
91     string lineStr;
92     vector<vector<string>> strArray;
93     while (getline(inFile, lineStr))
94     {
95         stringstream ss(lineStr);
96         string str;
97         vector<string> lineArray;
98         // 按照逗号分隔
99         while (getline(ss, str, ','))
100             lineArray.push_back(str);
101         strArray.push_back(lineArray);
102     }
103     thread downloadThreads[5];
104     for (auto i : strArray)
105     {
106         if(is_exist(i[0])) continue; else {
107             downloadThreads[download_cnt] =
108             thread(sub_downloader, i[0], i[1], i[2]);
109             downloadThreads[download_cnt].join();
110             ++download_cnt;
111         }
112     }
113 }
114 }
115
116 int main (int argc, char *argv[]){
117     sub_downloader(TORRENT_NAME, TRACKER_IP, TRACKER_PORT);
118     thread download(downloader);
119     download.join();
120 };

```

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.