report

tangpeng: 517020910038

The main purpose of this lab is to use the mininet to simulate the server/client model in the network.

Server implemented

We need to write a server program to accept from clients' requests. Because the server need serve many clients simultaneously, we need create a new thread to do the service when a new client comes. The code of server is shown as follows.

```
//// multserver.cc


#include <pthread.h>
#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <unistd.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

pthread_mutex_t mutex;
// the function for every thread
int BufSize = 1024;
int count = 100000;
int clinentNumber = 1024; //the max number of clients server can accept.


//the function for thread
void *client_thread(void *client)
{
    int clientC = *(int*)client;//get the file description
    char buffer[BufSize];

    memset(&buffer,'0', BufSize);

    for(int i=0; i<count; i++)
    {
        snprintf(buffer, sizeof(buffer), "I'm server!\r\n");
        write(clientC, buffer, strlen(buffer));
    }

   close(clientC);

}
```

```c


int main(){

    pthread_t client[clinentNumber]; //to store the threads
    int clientnum=0; //to recode the number of clients

    pthread_attr_t attr;
    pthread_attr_init(&attr);
    struct sockaddr_in server, client_addr;
    socklen_t leng = sizeof(client_addr);
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    memset(&server, '0', sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(2020);
    bind(sock, (struct sockaddr*)&server, sizeof(server));
    listen(sock,20);
    while (1)
    {

        memset(&client_addr,'0',sizeof(client_addr));
        int clientcount = accept(sock,(struct sockaddr*)&client_addr, &leng); //get the client

        //print the information about the client
        char ip[16];
        char name[256];
         snprintf(name,sizeof(name),"client ip: %s, port: %d. \n", inet_ntop(AF_INET,&client_addr.sin_addr,ip,sizeof(ip)),
                ntohs(client_addr.sin_port));
        fputs(name,stdout);

        //creat the threads
         if(pthread_create(&client[clientnum], &attr,client_thread, (void*)&clientcount) ){
            printf("Error creating thread\n");
            return 1;
            }

        clientnum++;
        sleep(1);
    }
    close(sock);
    return 0;
}
```

- In above code, every thread will execute the **client_thread()** function, which can send "I'm server!" to clients for 100000 times.
- In the **main()** function, we create a socket for the server and wait for connecting with clients. When it accepts a request from a client, it will create a new thread for the client., then the thread will run the **client_thread()** function to send data to the client.

Client implemented

We need to write a client program to connect with server and receive the data coming from the server. The code of client is shown as follows.

```
//// client.cc


#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 <arpa/inet.h>
#include <time.h>
#include <string>
int BufSize = 1024;
int main(int argc, char *argv[]){

    if(argc!=2)
        return 0;
    int clientIndex=atoi(argv[1]);  //to get the client index


    time_t clock;
    int CreateSocket = 0,n = 0;
    char dataReceived[BufSize];
    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(2020);
    ipOfServer.sin_addr.s_addr = inet_addr("10.0.0.11");
    if(connect(CreateSocket, (struct sockaddr *)&ipOfServer, sizeof(ipOfServer))<0)
    {
        printf("Connection failed due to port and ip problems\n");
        return 1;
    }


    //print the starting time

    snprintf(dataReceived, sizeof(dataReceived), "%d", clientIndex);
    if(fputs(dataReceived, stdout) == EOF)
    {
    printf("\nStandard output error");
    }
    snprintf(dataReceived, sizeof(dataReceived), " starting time: ");
    if(fputs(dataReceived, stdout) == EOF)
    {
    printf("\nStandard output error");
    }
    clock = time(NULL);
    snprintf(dataReceived, sizeof(dataReceived), "%.24s\r\n", ctime(&clock));
    if(fputs(dataReceived, stdout) == EOF)
    {
        printf("\nStandard output error");
    }


    //creat the file index.txt file to store the data coming from the server
    char filename[20]="./clients/";
    strcat(filename, argv[1]);
    char tt[10] = ".txt";
    strcat(filename, tt);

    FILE *outfile;
    outfile = fopen(filename,"a+");

    //receive the data
    while((n = read(CreateSocket, dataReceived, sizeof(dataReceived)-1)) > 0)
    {
        dataReceived[n] = 0;
        if(fputs(dataReceived, outfile) == EOF)     //write the data to the file
        {
            printf("\nStandard output error");
        }

    }
    if( n < 0)
    {
        printf("Standard input error \n");

    }


    //print the ending time of this client
    snprintf(dataReceived, sizeof(dataReceived),"%d", clientIndex);
    if(fputs(dataReceived, stdout) == EOF)
    {
    printf("\nStandard output error");
    }
    snprintf(dataReceived, sizeof(dataReceived), " ending time:   ");
    if(fputs(dataReceived, stdout) == EOF)
    {
    printf("\nStandard output error");
    }
    clock = time(NULL);

    snprintf(dataReceived, sizeof(dataReceived), "%.24s\r\n", ctime(&clock));
    if(fputs(dataReceived, stdout) == EOF)
    {
        printf("\nStandard output error");
    }


    return 0;
}
```

- In this code, the **main()** function need a parameter to certify the index of clients. Then it will connect with the server and receive the data. The data received will be stored in the "*clientIndex.txt*" file in the "*clients*" folder.
- What's more, in order to know how much time this data transport will spend, we output the starting time and ending time of the client process.
- We need to modify the server ip in the code for different environments.

Network implemented

We use the mininet to create the network that one server and N clients model. The code is shown as follows.

```
### net.py

import sys

from functools import partial

from mininet.net import Mininet
from mininet.node import UserSwitch, OVSKernelSwitch, Controller
from mininet.topo import Topo
from mininet.log import lg, info
from mininet.util import irange, quietRun
from mininet.link import TCLink
from mininet.node import CPULimitedHost
from mininet.util import dumpNodeConnections
from mininet.cli import CLI

class netTopo(Topo):
    #define the star network
    def build(self, N, **params):
        #N clients and one server and N switches
        server = self.addHost('server')
        switch = self.addSwitch('s1')
        clients = [self.addHost('client%s' %h) for h in irange(1,N)]

        #add links
        self.addLink(server,switch,bw=1, delay='5ms', loss=0, use_htb=True)
        for client in clients:
            self.addLink(client, switch,bw=1, delay='5ms', loss=0, use_htb=True)





def netTest(N): # N is the clients number
    topo = netTopo(N)
    net = Mininet( topo=topo,
                    host=CPULimitedHost, link=TCLink,
                    autoStaticArp=True )
    net.start()
    info( "Dumping host connections\n" )
    dumpNodeConnections(net.hosts)

    server = net.getNodeByName('server')
    clients = [net.getNodeByName('client%s' %h) for h in irange(1,N)]


    #set the server and clients
    server.cmd("./multserver &")
    for i in range(1,N+1):
        clients[i-1].cmd("./client %s >> cilent.txt &" %i)

    CLI(net)
    net.stop()

if __name__=='__main__':
    #define N clients
    N=10
    netTest(N)
```

In above code, we create N(here we let $N = 10$) hosts as clients and one client as server. Because the host can not connect with each other, we use one switch to connect them. After creating the network, we use the following commands in the code to run the server program and client program defined before.

```
    server.cmd("./multserver &")
    for i in range(1,N+1):
        clients[i-1].cmd("./client %s >> cilent.txt &" %i)
```

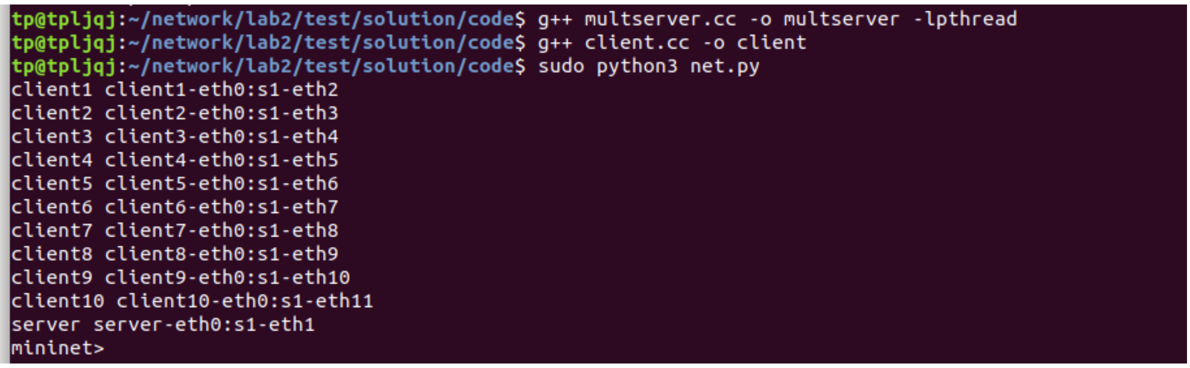
What's more, we output the starting time and ending time of every client to the *"cilent.txt"*.

Run the whole program

In the *"code"* folder, we can run the following commands in the terminal in ubantu 18.04 to execute these programs.

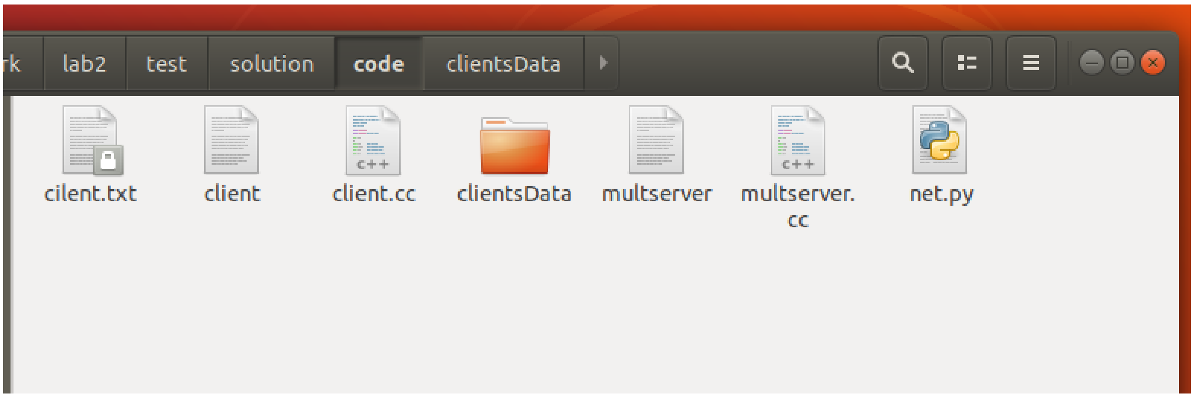
```
g++ multserver.cc -o multserver -lpthread
g++ multserver.cc -o multserver -lpthread
sudo python3 net.py
```

The following picture shows these commands running in my computer.

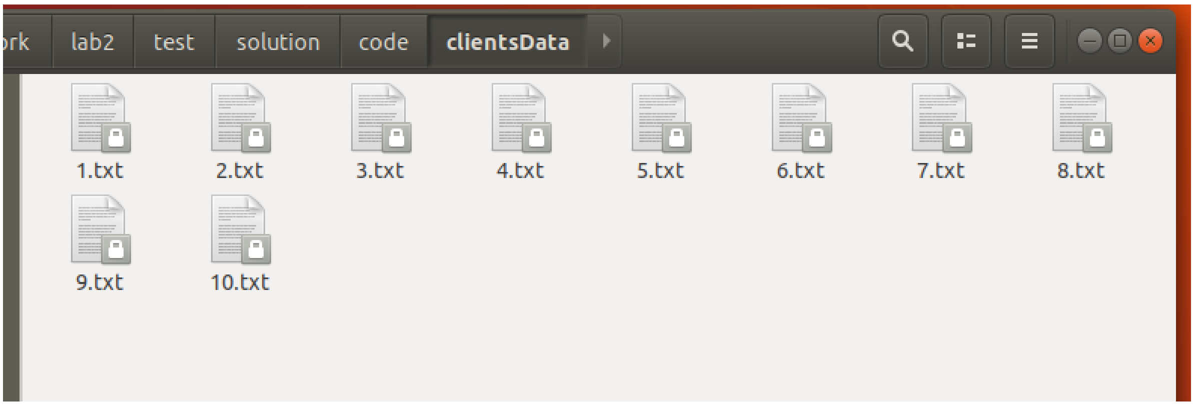


```
tp@tpljqj:~/network/lab2/test/solution/code$ g++ multserver.cc -o multserver -lpthread
tp@tpljqj:~/network/lab2/test/solution/code$ g++ client.cc -o client
tp@tpljqj:~/network/lab2/test/solution/code$ sudo python3 net.py
client1 client1-eth0:s1-eth2
client2 client2-eth0:s1-eth3
client3 client3-eth0:s1-eth4
client4 client4-eth0:s1-eth5
client5 client5-eth0:s1-eth6
client6 client6-eth0:s1-eth7
client7 client7-eth0:s1-eth8
client8 client8-eth0:s1-eth9
client9 client9-eth0:s1-eth10
client10 client10-eth0:s1-eth11
server server-eth0:s1-eth1
mininet>
```

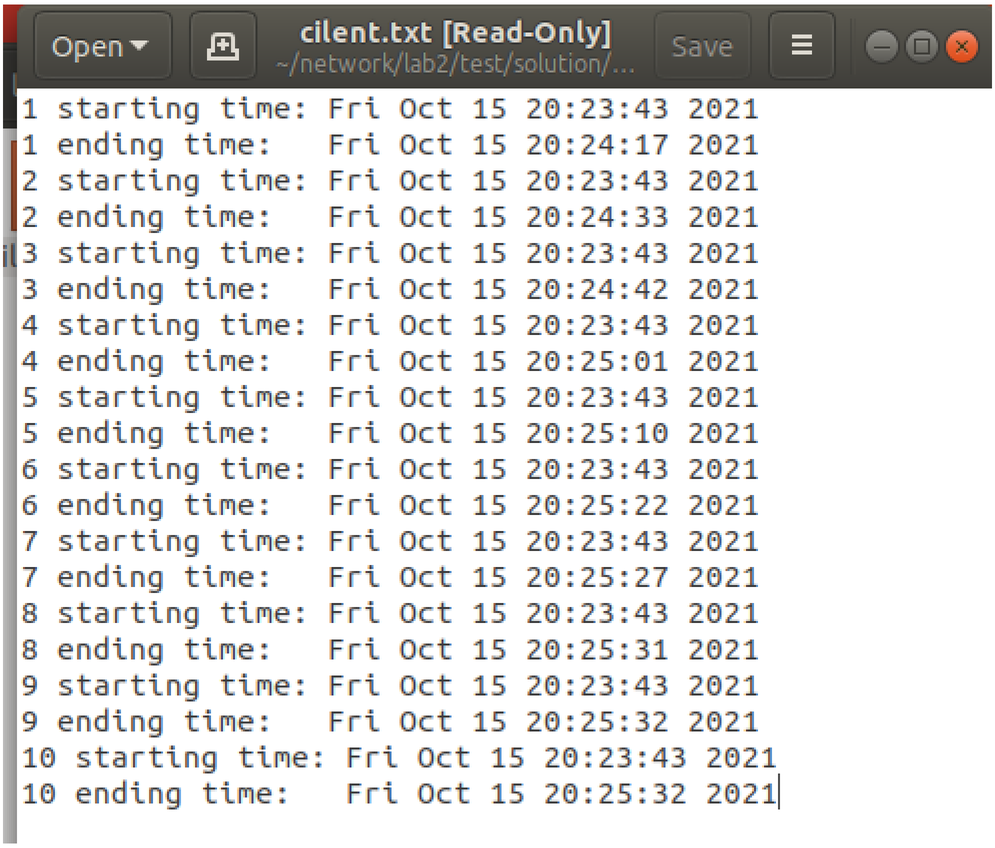
After running above commands, the "code" folder is like the following figure.



In the *"clientsData"* folder, there are 10(we create 10 clients) files which store the data clients received.



The *client.txt* file stores the starting time and ending time of all clients. We can compute the whole time the server spends to transport the data according the time in the file. We can just use the max time minus the min time, we can obtain the spending time. In this case, the time is 109 seconds.

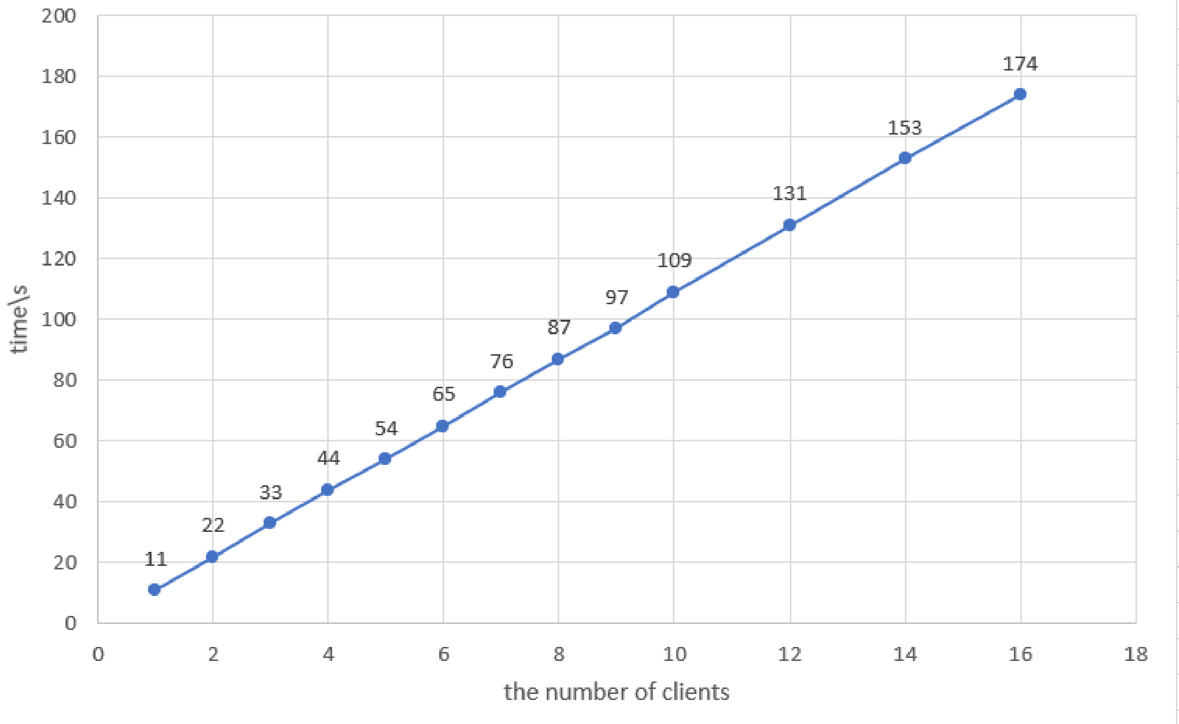


```
1 starting time: Fri Oct 15 20:23:43 2021
1 ending time:   Fri Oct 15 20:24:17 2021
2 starting time: Fri Oct 15 20:23:43 2021
2 ending time:   Fri Oct 15 20:24:33 2021
3 starting time: Fri Oct 15 20:23:43 2021
3 ending time:   Fri Oct 15 20:24:42 2021
4 starting time: Fri Oct 15 20:23:43 2021
4 ending time:   Fri Oct 15 20:25:01 2021
5 starting time: Fri Oct 15 20:23:43 2021
5 ending time:   Fri Oct 15 20:25:10 2021
6 starting time: Fri Oct 15 20:23:43 2021
6 ending time:   Fri Oct 15 20:25:22 2021
7 starting time: Fri Oct 15 20:23:43 2021
7 ending time:   Fri Oct 15 20:25:27 2021
8 starting time: Fri Oct 15 20:23:43 2021
8 ending time:   Fri Oct 15 20:25:31 2021
9 starting time: Fri Oct 15 20:23:43 2021
9 ending time:   Fri Oct 15 20:25:32 2021
10 starting time: Fri Oct 15 20:23:43 2021
10 ending time:   Fri Oct 15 20:25:32 2021
```

If we want modify the number of clients, we need to modify the value of N in the *net.py*. What's more, may we also need to modify the server ip in the *client.cc*.

Compare the downloading time

We can change the clients number to get different time. The following figure shows the change of time when we change the number of clients. From the figure, we can see the relationship between time and the number of clients is linear.


time\s
the number of clients