CN Assignment-3 Report

Name: Dhruv Deshmukh Roll No: 11940380 Email: dhruvr@iitbhilai.ac.in

Note: There are demo videos for every part. The individual video links are given in each question.
Here is the link for the folder with all videos. Please use the institute account to access the videos.
Link of all videos folder:
https://drive.google.com/drive/folders/1CxF5veDTt0cDM1iYnxFrS0kxIqowlgVz?usp=sharing

For each part the format in report is:

1. Server Code
2. Client Code
3. Explanation
4. Link for Demo video
5. Example Run

Part 1a:

The code for server:

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

#define MAXLINE 1024
#define PORT "3490"

int parseNum(char *message, char end)
```

```c
{
  int i=0;
  int id = 0;
  while(message[i] != end)
  {
    id = id*10+(message[i]-'0');
    i++;
  }
  return id;
}

void *cast_ipv(struct sockaddr* sa)
{
  if(sa->sa_family == AF_INET)
  {
    return &(((struct sockaddr_in*)sa)->sin_addr);
  }
  return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

int main(int argc, char* argv[])
{
  char* serverAddrString = NULL;
  int delay = 2;
  for(int i=1;i<argc;i++)
  {
    if(argv[i][0] == '-' && argv[i][1] == 'd')
    {
      i++;
      delay = parseNum(argv[i],'\0');
    }
    else
    {
      serverAddrString = (char*)malloc(sizeof(*argv[i]));
      strcpy(serverAddrString, argv[i]);
    }
  }

   printf("%s\n", serverAddrString==NULL?"localhost":serverAddrString);
  int socketDescriptor;
  int number;
  int addressLength;
  char message[MAXLINE];

  struct sockaddr_in  clientAddress;
  //Get IP from name
  int ev;
  struct addrinfo hints, *servinfo, *it;

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_DGRAM;

  if((ev = getaddrinfo(serverAddrString, PORT, &hints, &servinfo))!=0)
  {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ev));
    return 1;
  }

  for(it = servinfo; it != NULL; it = it->ai_next)
  {
    if((socketDescriptor = socket(AF_INET, it->ai_socktype,
it->ai_protocol))!=-1)
    {
      if(bind(socketDescriptor,it->ai_addr, it->ai_addrlen)==-1)
      {
        close(socketDescriptor);
        continue;
      }
      else
        break;
    }
  }

  if(it == NULL)
  {
    perror("Failed to acquire a socket!\n");
    return 2;
  }
  char sa[INET_ADDRSTRLEN];
  inet_ntop(it->ai_family,cast_ipv(it->ai_addr),sa, INET_ADDRSTRLEN);
  printf("\nServer Started ...%s\n",sa);

  freeaddrinfo(servinfo);
  int c=0;

  while(1){
    // printf("\n");
    addressLength = sizeof(clientAddress);

    number = recvfrom(socketDescriptor,message,MAXLINE,0,(struct
sockaddr*)&clientAddress,&addressLength);
    char ip[INET_ADDRSTRLEN];
    inet_ntop(clientAddress.sin_family,&(clientAddress.sin_addr),ip,
INET_ADDRSTRLEN);
    printf("\n Message from client %s \n ", ip);
    // printf("\n Client's Message: %s ",message);

    if(number<1)
      perror("send error");

    int n = rand();
    if(c>0 && n%c==0)
    {
      sleep(delay);
    }
    printf("n=%d c=%d\n",n, c);
      // printf("%s\n", message);
      sendto(socketDescriptor,message,number,0,(struct
sockaddr*)&clientAddress,addressLength);
      if(c>20)
        c=0;
      else
        c++;
  }
  close(socketDescriptor);
  return 0;
}
```

The code for client:

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

#define MAXLINE 1024
#define PORT "3490"
pthread_mutex_t lock;

typedef struct echosendkit{
  int interval;
  int packetSize;
  int numberOfMessages;
  struct sockaddr *to;
  int tolen;
  int socfd;
  clock_t *sendTimes;
} echoSendKit;

typedef struct echorcvkit{
  int interval;
  int packetSize;
  int numberOfMessages;
  struct sockaddr *from;
  int fromlen;
  int socfd;
  clock_t *sendTimes;
  clock_t *rcvTimes;
  int timeOut;
```

```
} echoRcvKit;

int parseNum(char *message, char end)
{
  int i=0;
  int id = 0;
  while(message[i] != end)
  {
    id = id*10+(message[i]-'0');
    i++;
  }
  return id;
}

void *sendEchoMessages(void *arg)
{
  echoSendKit *sendKit = (echoSendKit*)arg;
  for(int i=0;i<sendKit->numberOfMessages;i++)
  {
    //create message
    char* message = (char*)malloc(sizeof(char)*sendKit->packetSize);
    int len = sprintf(message, "%d", i);
    int paddingLen = sendKit->packetSize-len;
    char *padding = (char*)malloc(sizeof(char)*paddingLen);
    memset(padding, '$', paddingLen-1);
    strcat(message, padding);

    //send message
    pthread_mutex_lock(&lock);
    sendKit->sendTimes[i] = clock();
    int temp =
sendto(sendKit->socfd,message,sendKit->packetSize,0,sendKit->to,sendKit->t
olen);
    pthread_mutex_unlock(&lock);
    // printf("send: %d\n", temp);
    //delay
    sleep(sendKit->interval);
    if(message!=NULL)
      free(message);
    if(padding!=NULL)
```

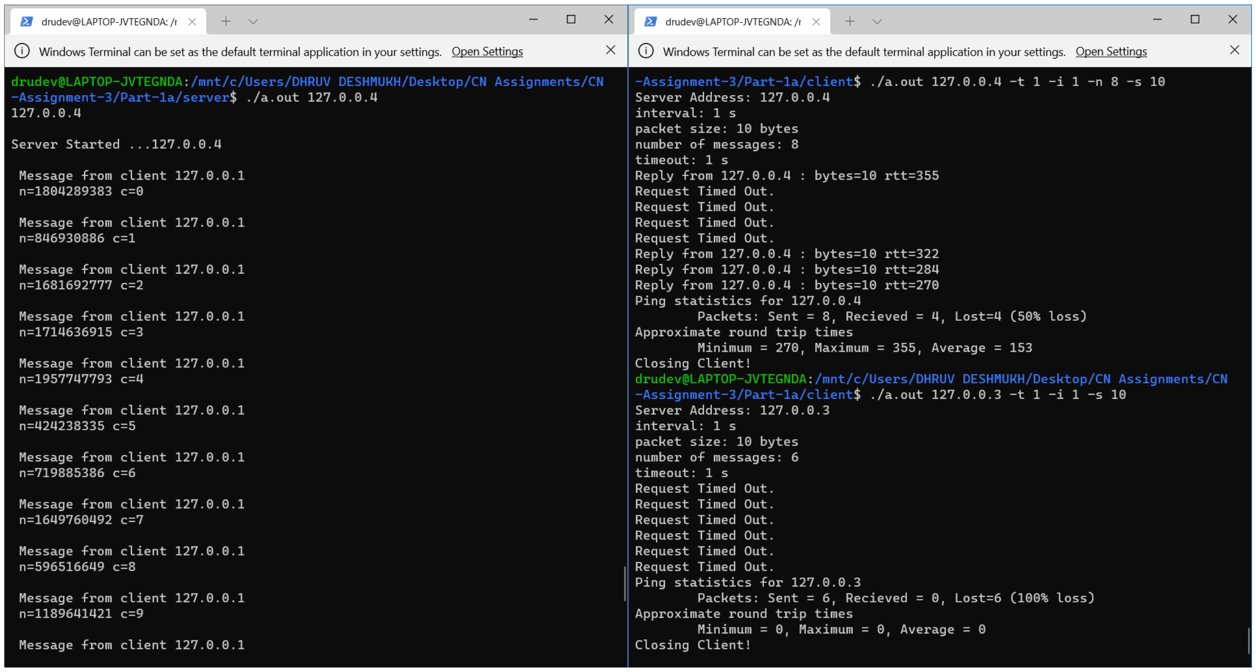
```
      free(padding);
    }
}

void *receiveEchoMessages(void *arg)
{
  echoRcvKit *rcvKit = (echoRcvKit*)arg;
  int expecting = rcvKit->numberOfMessages;
  while(expecting>0)
  {
    //check for timeout for each message
    pthread_mutex_lock(&lock);
    for(int i=0;i<rcvKit->numberOfMessages;i++)
    {
      if(rcvKit->sendTimes[i]!=-1)
      {
        if(rcvKit->rcvTimes[i]!=-2 && (rcvKit->rcvTimes[i]==-1 ||
(rcvKit->rcvTimes[i]==0 &&
clock()>rcvKit->sendTimes[i]+(1000000*rcvKit->timeOut))))
        {
          // printf("rcv: %ld ", rcvKit->rcvTimes[i]);
          rcvKit->rcvTimes[i] = -2;
          expecting--;
          printf("Request Timed Out.\n");
        }
      }

    }
    // printf("end2\n");
    pthread_mutex_unlock(&lock);
    // printf("d%d\n", expecting);
    //initialize message
    char* message = (char*)malloc(sizeof(char)*rcvKit->packetSize);

    //receive message
    int rcvlen =
recvfrom(rcvKit->socfd,message,rcvKit->packetSize,MSG_DONTWAIT,rcvKit->fro
m,&(rcvKit->fromlen));
    if(rcvlen!=-1)
    {
      clock_t rcvTime = clock();
      int id = parseNum(message, '$');
      // printf("outside\n");
      pthread_mutex_lock(&lock);
      // printf("inside\n");
      if(rcvKit->rcvTimes[id]!=-2)
      {
        rcvKit->rcvTimes[id] = rcvTime;
        if(rcvTime>rcvKit->sendTimes[id]+(1000000*rcvKit->timeOut))
          rcvKit->rcvTimes[id]=-1;
        else
        {
          clock_t rtt =
difftime(rcvKit->rcvTimes[id],rcvKit->sendTimes[id]);
          char ip4[INET_ADDRSTRLEN];
          inet_ntop(AF_INET,&(((struct
sockaddr_in*)rcvKit->from)->sin_addr), ip4,INET_ADDRSTRLEN);
          printf("Reply from %s : bytes=%d rtt=%ld\n", ip4, rcvlen, rtt);
          expecting--;
        }
      }
      // printf("end\n");
      pthread_mutex_unlock(&lock);
    }
    // printf("a%d\n", expecting);
  }
}

void initializeEchoSendKit(int interval, int packetSize, int
numberOfMessages, struct sockaddr *to, int tolen, int socfd, clock_t*
sendTimes, echoSendKit *sendKit)
{
  sendKit->interval = interval;
  sendKit->packetSize = packetSize;
  sendKit->numberOfMessages = numberOfMessages;
  sendKit->to = to;
  sendKit->tolen = tolen;
  sendKit->socfd = socfd;
  sendKit->sendTimes = sendTimes;
}
```

```c
void initializeEchoRcvKit(int interval, int packetSize, int
numberOfMessages, struct sockaddr *from, int fromlen, int socfd, clock_t*
sendTimes, clock_t* rcvTimes, int timeOut, echoRcvKit *rcvKit)
{
  rcvKit->interval = interval;
  rcvKit->packetSize = packetSize;
  rcvKit->numberOfMessages = numberOfMessages;
  rcvKit->from = from;
  rcvKit->fromlen = fromlen;
  rcvKit->socfd = socfd;
  rcvKit->sendTimes = sendTimes;
  rcvKit->rcvTimes = rcvTimes;
  rcvKit->timeOut = timeOut;
}

int main(int argc, char* argv[]){
  //params
  int interval = 2; //in seconds
  int packetSize = 8; //in bytes
  int numberOfMessages = 6;
  int timeOut = 4; //in seconds
  char* serverAddrString = NULL;
  int addressLength;
  // make a socket for the client
  int socketDescriptor = socket(AF_INET, SOCK_DGRAM, 0);

  //Parse all the arguments
  for(int i=1;i<argc;i++)
  {
    if(argv[i][0] == '-' && argv[i][1] == 'i')
    {
      i++;
      interval = parseNum(argv[i],'\0');
    }
    else if(argv[i][0] == '-' && argv[i][1] == 's')
    {
      i++;
      packetSize = parseNum(argv[i],'\0');
    }
    else if(argv[i][0] == '-' && argv[i][1] == 'n')
    {
      i++;
      numberOfMessages = parseNum(argv[i],'\0');
    }
    else if(argv[i][0] == '-' && argv[i][1] == 't')
    {
      i++;
      timeOut = parseNum(argv[i],'\0');
    }
    else
    {
      serverAddrString = (char*)malloc(sizeof(*argv[i]));
      strcpy(serverAddrString, argv[i]);
    }
  }

  printf("Server Address: %s\n",
serverAddrString==NULL?"localhost":serverAddrString);
  printf("interval: %d s\n", interval);
  printf("packet size: %d bytes\n", packetSize);
  printf("number of messages: %d\n", numberOfMessages);
  printf("timeout: %d s\n", timeOut);

  //Get IP from name
  int ev;
  struct addrinfo hints, *servinfo;

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_DGRAM;

  if((ev = getaddrinfo(serverAddrString, PORT, &hints, &servinfo))!=0)
  {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ev));
    return 1;
  }

  //Allocate and initialize the time arrays
  clock_t *sendTimes = (clock_t*)malloc(sizeof(clock_t)*numberOfMessages);
  clock_t *rcvTimes = (clock_t*)calloc(numberOfMessages,sizeof(clock_t));
  for(int i=0;i<numberOfMessages;i++)
  {
    sendTimes[i] = -1;
  }
  //Allocate and initialize the structures passed to the send and receive
functions
  echoSendKit *sendKit = (echoSendKit*)malloc(sizeof(echoSendKit));
  echoRcvKit *rcvKit = (echoRcvKit*)malloc(sizeof(echoRcvKit));
  initializeEchoSendKit(interval, packetSize, numberOfMessages,
servinfo->ai_addr, servinfo->ai_addrlen, socketDescriptor, sendTimes,
sendKit);

  initializeEchoRcvKit(interval, packetSize, numberOfMessages,
servinfo->ai_addr, servinfo->ai_addrlen, socketDescriptor, sendTimes,
rcvTimes, timeOut, rcvKit);

  //Declare the threads
  pthread_t sendThread, rcvThread;

  //Initialize a lock
  if (pthread_mutex_init(&lock, NULL) != 0)
  {
      printf("\n mutex init failed\n");
      return 1;
  }

  //Start the clock
  clock_t startTime = clock();

  //Create the threads for the send and receive functions
  if(pthread_create(&sendThread, NULL, sendEchoMessages, sendKit)!=0)
  {
    perror("Could not create Send Thread!\n");
    exit(0);
  }



  if(pthread_create(&rcvThread, NULL, receiveEchoMessages, rcvKit))
  {
    perror("Could not create Receive Thread!\n");
    exit(0);
  }

  //join the threads after completion
  if(pthread_join(sendThread, NULL)!=0)
  {
    perror("Could not join Send Thread!\n");
    exit(0);
  }
  if(pthread_join(rcvThread, NULL)!=0)
  {
    perror("Could not join Receive Thread!\n");
    exit(0);
  }

  //Get the Statistics
  char ip4[INET_ADDRSTRLEN];
  inet_ntop(AF_INET, &(((struct sockaddr_in
*)servinfo->ai_addr)->sin_addr), ip4, INET_ADDRSTRLEN);
  clock_t maxrtt = 0, avgrtt = 0;
  clock_t minrtt = INT_MAX;
  int lost = 0, rcved = 0, percentloss = 0;
  for(int i=0;i<numberOfMessages;i++)
  {
    clock_t rtt = difftime(rcvKit->rcvTimes[i], rcvKit->sendTimes[i]);
    if(rtt>=0)
    {
      if (rtt > maxrtt)
        maxrtt = rtt;
      if (rtt < minrtt)
        minrtt = rtt;
      avgrtt+=rtt;
    }
    else
      lost++;
  }
  if(minrtt == INT_MAX)
    minrtt = 0;
  avgrtt/=numberOfMessages;
  rcved = numberOfMessages - lost;
  percentloss = (lost*100)/numberOfMessages;
  printf("Ping statistics for %s\n", ip4);
  printf("\tPackets: Sent = %d, Recieved = %d, Lost=%d (%d%% loss)\n",
numberOfMessages, rcved, lost, percentloss);
  printf("Approximate round trip times\n");
  printf("\tMinimum = %ld, Maximum = %ld, Average = %ld\n", minrtt,
maxrtt, avgrtt);

  printf("Closing Client!\n");

  freeaddrinfo(servinfo);
  if(serverAddrString!=NULL)
    free(serverAddrString);
  if(sendKit!=NULL)
    free(sendKit);
  if(rcvKit!=NULL)
    free(rcvKit);
  if(sendTimes!=NULL)
    free(sendTimes);
  if(rcvTimes!=NULL)
    free(rcvTimes);


  return 0;
}
```

Explanation: The server and client are adapted from the one Professor Anand used in his lectures. The server just receives the message and sends it back. Also, a delay mechanism is implemented to simulate an actual delay in networks. A random number is generated every time and if it is divisible by a counter that is incremented periodically then delay is added before sending the packet back to the client. Now the timeout set for the client is by default 4 s and the delay at the server is 2s so there will be no loss in general. But if the timeout is made 2s then there will be a timeout. I have shown this in the explanation video. The client takes in the inputs from the user using command line arguments. There are two threads one for send and the other for receive. The send and receive times are recorded to calculate the RTT and finally, all the statistics are printed. Everything is explained in the video in detail.

Link to Demo Video(7.5 mins):
https://drive.google.com/file/d/1tLnmbXUoCHRpLCgXjc74W_QiggxiTWqh/view?usp=sharing

Example Run and output:



```
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-1a/server$ ./a.out 127.0.0.4
127.0.0.4

Server Started ...127.0.0.4

 Message from client 127.0.0.1
 n=1804289383 c=0

 Message from client 127.0.0.1
 n=846930886 c=1

 Message from client 127.0.0.1
 n=1681692777 c=2

 Message from client 127.0.0.1
 n=1714636915 c=3

 Message from client 127.0.0.1
 n=1957747793 c=4

 Message from client 127.0.0.1
 n=424238335 c=5

 Message from client 127.0.0.1
 n=719885386 c=6

 Message from client 127.0.0.1
 n=1649760492 c=7

 Message from client 127.0.0.1
 n=596516649 c=8

 Message from client 127.0.0.1
 n=1189641421 c=9

 Message from client 127.0.0.1
```

```
-Assignment-3/Part-1a/client$ ./a.out 127.0.0.4 -t 1 -i 1 -n 8 -s 10
Server Address: 127.0.0.4
interval: 1 s
packet size: 10 bytes
number of messages: 8
timeout: 1 s
Reply from 127.0.0.4 : bytes=10 rtt=355
Request Timed Out.
Request Timed Out.
Request Timed Out.
Request Timed Out.
Reply from 127.0.0.4 : bytes=10 rtt=322
Reply from 127.0.0.4 : bytes=10 rtt=284
Reply from 127.0.0.4 : bytes=10 rtt=270
Ping statistics for 127.0.0.4
        Packets: Sent = 8, Recieved = 4, Lost=4 (50% loss)
Approximate round trip times
        Minimum = 270, Maximum = 355, Average = 153
Closing Client!
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-1a/client$ ./a.out 127.0.0.3 -t 1 -i 1 -s 10
Server Address: 127.0.0.3
interval: 1 s
packet size: 10 bytes
number of messages: 6
timeout: 1 s
Request Timed Out.
Request Timed Out.
Request Timed Out.
Request Timed Out.
Request Timed Out.
Request Timed Out.
Ping statistics for 127.0.0.3
        Packets: Sent = 6, Recieved = 0, Lost=6 (100% loss)
Approximate round trip times
        Minimum = 0, Maximum = 0, Average = 0
Closing Client!
```

The left is the server and the right is the client. In the first run of the client the correct address is provided correctly so we get a response with occasional dropping of packets due to the delay mechanism implemented.
In the second run the wrong address is given and hence we get time out on all messages as there is no server at that ip address.

Part 1b:

The code for server:

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

```c
#define MAXLINE 1024
#define PORT "3490"



int parseNum(char *message, char end)
{
  int i=0;
  int id = 0;
  while(message[i] != end)
  {
    id = id*10+(message[i]-'0');
    i++;
  }
  return id;
}

void *cast_ipv(struct sockaddr* sa)
{
  if(sa->sa_family == AF_INET)
  {
    return &(((struct sockaddr_in*)sa)->sin_addr);
  }
  return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

int main(int argc, char* argv[])
{
  char* serverAddrString = NULL;
  int delay = 2;
  for(int i=1;i<argc;i++)
  {
    if(argv[i][0] == '-' && argv[i][1] == 'd')
    {
      i++;
      delay = parseNum(argv[i],'\0');
    }
    else
    {
      serverAddrString = (char*)malloc(sizeof(*argv[i]));
      strcpy(serverAddrString, argv[i]);
    }
  }

  printf("%s\n", serverAddrString==NULL?"localhost":serverAddrString);
  int socketDescriptor;
  int number;
  int addressLength;
  char message[MAXLINE];

  struct sockaddr_in  clientAddress;
  //Get IP from name
  int ev;
  struct addrinfo hints, *servinfo, *it;

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_DGRAM;

  if((ev = getaddrinfo(serverAddrString, PORT, &hints, &servinfo))!=0)
  {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ev));
    return 1;
  }

  for(it = servinfo; it != NULL; it = it->ai_next)
  {
    if((socketDescriptor = socket(AF_INET, it->ai_socktype,
it->ai_protocol))!=-1)
    {
      if(bind(socketDescriptor,it->ai_addr, it->ai_addrlen)==-1)
      {
        close(socketDescriptor);
        continue;
      }
      else
        break;
    }
  }
  if(it == NULL)
  {
    perror("Failed to acquire a socket!\n");
    return 2;
  }

  char sa[INET_ADDRSTRLEN];
  inet_ntop(it->ai_family,cast_ipv(it->ai_addr),sa, INET_ADDRSTRLEN);
  printf("\nServer Started ...%s\n",sa);

  freeaddrinfo(servinfo);
  int c=0;

  while(1){
    // printf("\n");
    addressLength = sizeof(clientAddress);

    number = recvfrom(socketDescriptor,message,MAXLINE,0,(struct
sockaddr*)&clientAddress,&addressLength);
    char ip[INET_ADDRSTRLEN];
    inet_ntop(clientAddress.sin_family,&(clientAddress.sin_addr),ip,
INET_ADDRSTRLEN);
    // printf("\n Message from client %s \n ", ip);
    // printf("\n Client's Message: %s ",message);

    if(number<1)
      perror("send error");

    int n = rand();
    if(c%23!=0)//(c==0 || n%c!=0)
    {
    // printf("n=%d c=%d\n",n, c);
      // printf("%s\n", message);
      sendto(socketDescriptor,message,number,0,(struct
sockaddr*)&clientAddress,addressLength);
    }
      if(c>20)
        c=0;
      else
        c++;
  }
  close(socketDescriptor);
  return 0;
}
```

The code for client:

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

#define MAXLINE 1024
#define PORT "3490"
pthread_mutex_t lock;
int sent[10]={0},receive[10]={0};
int sthroughput[10]={0}, rthroughput[10]={0};
clock_t pdelay[10]={0};
int totalSent=0,totalRcv=0;
int s=0;

typedef struct echosendkit{
  int interval;
  int packetSize;
  struct sockaddr *to;
  int tolen;
  int socfd;
  clock_t *sendTimes;
} echoSendKit;

typedef struct echorcvkit{
```

```c
  int interval;
  int packetSize;
  struct sockaddr *from;
  int fromlen;
  int socfd;
  clock_t *sendTimes;
  clock_t *rcvTimes;
} echoRcvKit;



void delay(int seconds)
{
    int milliSeconds = 50 * seconds;

    clock_t startTime = clock();

    while (clock() < startTime + milliSeconds)
        ;
}

int parseNum(char *message, char end)
{
  int i=0;
  int id = 0;
  while(message[i] != end)
  {
    id = id*10+(message[i]-'0');
    i++;
  }
  return id;
}

void *sendEchoMessages(void *arg)
{
  echoSendKit *sendKit = (echoSendKit*)arg;
  for(int k=0;k<10;k++)
  {
    clock_t curTime = clock();
    int c=0;
    while(clock()<curTime+1000000)//&&c<=left)
    {
    // create message
        char *message = (char *)malloc(sizeof(char) * 100);
        int len = sprintf(message, "%d", totalSent);
        int paddingLen = 100 - len;
        char *padding = (char *)malloc(sizeof(char) * paddingLen);
        memset(padding, '$', paddingLen - 1);
        padding[paddingLen-1]='\0';
        strcat(message, padding);

        // send message
        pthread_mutex_lock(&lock);
        sendKit->sendTimes[totalSent] = clock();
        int temp = sendto(sendKit->socfd, message, sendKit->packetSize, 0,
sendKit->to, sendKit->tolen);
        c++,totalSent++;
        pthread_mutex_unlock(&lock);
        // printf("send: %d\n", temp);
        // delay
        delay(sendKit->interval);
        if (message != NULL)
          free(message);
        if (padding != NULL)
          free(padding);
    }
    sent[k]=c;
  }
  printf("Sent Packets = %d\n", totalSent);
  s=1;
}

void *receiveEchoMessages(void *arg)
{
  echoRcvKit *rcvKit = (echoRcvKit*)arg;
  for(int i=0;i<10&&s==0;i++)
  {
    int c=0;
    clock_t startTime = clock();
    while(clock()< 1000000 + startTime)
```

```
    {
      //initialize message
      char* message = (char*)malloc(sizeof(char)*rcvKit->packetSize);

      //receive message
      int rcvlen =
recvfrom(rcvKit->socfd,message,rcvKit->packetSize,MSG_DONTWAIT,rcvKit->fro
m,&(rcvKit->fromlen));
      if(rcvlen!=-1)
      {
        c++,totalRcv++;
        clock_t rcvTime = clock();
        int id = parseNum(message, '$');
        pthread_mutex_lock(&lock);
        rcvKit->rcvTimes[id] = rcvTime;
        pthread_mutex_unlock(&lock);
      }
    }
    receive[i]=c;
  }
  printf("Received packets = %d\n",totalRcv);
}

void initializeEchoSendKit(int interval, int packetSize, struct sockaddr
*to, int tolen, int socfd, clock_t* sendTimes, echoSendKit *sendKit)
{
  sendKit->interval = interval;
  sendKit->packetSize = packetSize;
  sendKit->to = to;
  sendKit->tolen = tolen;
  sendKit->socfd = socfd;
  sendKit->sendTimes = sendTimes;
}

void initializeEchoRcvKit(int interval, int packetSize, struct sockaddr
*from, int fromlen, int socfd, clock_t* sendTimes, clock_t* rcvTimes,
echoRcvKit *rcvKit)
{
  rcvKit->interval = interval;
  rcvKit->packetSize = packetSize;
  rcvKit->from = from;
  rcvKit->fromlen = fromlen;
  rcvKit->socfd = socfd;
  rcvKit->sendTimes = sendTimes;
  rcvKit->rcvTimes = rcvTimes;
}

int main(int argc, char* argv[]){
  //params
  int interval = 1; //in seconds
  int packetSize = 100; //in bytes
  char* serverAddrString = NULL;
  int addressLength;
  // make a socket for the client
  int socketDescriptor = socket(AF_INET, SOCK_DGRAM, 0);

  //Parse all the arguments
  for(int i=1;i<argc;i++)
  {
    if(argv[i][0] == '-' && argv[i][1] == 's')
    {
      i++;
      packetSize = parseNum(argv[i],'\0');
    }
    else
    {
      serverAddrString = (char*)malloc(sizeof(*argv[i]));
      strcpy(serverAddrString, argv[i]);
    }
  }

  printf("Server Address: %s\n",
serverAddrString==NULL?"localhost":serverAddrString);
  printf("packet size: %d bytes\n", packetSize);

  //Get IP from name
  int ev;
  struct addrinfo hints, *servinfo;

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_DGRAM;

  if((ev = getaddrinfo(serverAddrString, PORT, &hints, &servinfo))!=0)
  {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ev));
    return 1;
  }

  //Allocate and initialize the time arrays
  clock_t *sendTimes = (clock_t*)malloc(sizeof(clock_t)*200000);
  clock_t *rcvTimes = (clock_t*)calloc(200000,sizeof(clock_t));
  for(int i=0;i<200000;i++)
  {
    sendTimes[i] = -1;
  }

  //Allocate and initialize the structures passed to the send and receive
functions
  echoSendKit *sendKit = (echoSendKit*)malloc(sizeof(echoSendKit));
  echoRcvKit *rcvKit = (echoRcvKit*)malloc(sizeof(echoRcvKit));
  initializeEchoSendKit(interval, packetSize, servinfo->ai_addr,
servinfo->ai_addrlen, socketDescriptor, sendTimes, sendKit);

  initializeEchoRcvKit(interval, packetSize, servinfo->ai_addr,
servinfo->ai_addrlen, socketDescriptor, sendTimes, rcvTimes, rcvKit);

  //Declare the threads
  pthread_t sendThread, rcvThread;

  //Initialize a lock
  if (pthread_mutex_init(&lock, NULL) != 0)
  {
      printf("\n mutex init failed\n");
      return 1;
  }

  //Start the clock
  printf("Sending Packets\n");
  clock_t startTime = clock();
  //Create the threads for the send and receive functions
  if(pthread_create(&sendThread, NULL, sendEchoMessages, sendKit)!=0)
  {
    perror("Could not create Send Thread!\n");
    exit(0);
  }

// sleep(1);
  if(pthread_create(&rcvThread, NULL, receiveEchoMessages, rcvKit))
  {
    perror("Could not create Receive Thread!\n");
    exit(0);
  }

  //join the threads after completion
  if(pthread_join(sendThread, NULL)!=0)
  {
    perror("Could not join Send Thread!\n");
    exit(0);
  }
  if(pthread_join(rcvThread, NULL)!=0)
  {
    perror("Could not join Receive Thread!\n");
    exit(0);
  }

  //Get the Statistics
  for(int i=0;i<10;i++)
  {
    sthroughput[i]=8*packetSize*sent[i];
    rthroughput[i]=8*packetSize*receive[i];
  }
  int savgtpt = (8*packetSize*totalSent)/10;
  int ravgtpt = (8*packetSize*totalRcv)/10;

  int avgpack = totalRcv/10;
  int j=0;
  clock_t avgdelay=0;
  for(int i=0;i<10;i++)
  {
    int c=0;
    for(int k=0;k<avgpack;k++)
    {
      if(rcvTimes[j*avgpack+k]!=0)
      {
        pdelay[i] +=
(difftime(rcvTimes[j*avgpack+k],sendTimes[j*avgpack+k])/2);
        c++;
      }
    }
    j++;
    avgdelay+=pdelay[i];
    pdelay[i]/=c;
    // printf("%ld\n", pdelay[i]);
  }
  avgdelay/=totalRcv;
  char ip4[INET_ADDRSTRLEN];
  inet_ntop(AF_INET, &(((struct sockaddr_in
*)servinfo->ai_addr)->sin_addr), ip4, INET_ADDRSTRLEN);
  printf("ThroughPut and Delay Statistics for %s\n", ip4);
  printf("Avg Send Throughput = %d bps Avg Recieve Throughput = %d bps\n",
savgtpt, ravgtpt);
  printf("Avg delay = %ld microseconds\n",avgdelay);

  printf("Storing the values in plot.txt\n");
  FILE *fptr = fopen("plot.txt", "w");
  for(int i=0;i<10;i++)
    fprintf(fptr, "%d %d %d %ld\n", i+1,sthroughput[i], rthroughput[i],
pdelay[i]);

  fclose(fptr);
  printf("Plotting the Graphs\n");
  int p1 = fork();

  if (p1 == 0)
  {
    /* This is the child process.  Execute the shell command. */
    char *args[3];
    args[0] = strdup("gnuplot");
```

```c
  args[1] = strdup("plot_throughput.plt");
  args[2] = NULL;
  int p2 = fork();
  if (p2 == 0)
  {
    /* This is the child process.  Execute the shell command. */
    char *args1[3];
    args1[0] = strdup("gnuplot");
    args1[1] = strdup("plot_delay.plt");
    args1[2] = NULL;

    execvp(args1[0],args1);
  }
  else if (p2 < 0)
    /* The fork failed.  Report failure.  */
    exit(1);
  else
  {
    /* This is the parent process.  Wait for the child to complete.  */
    execvp(args[0],args);
  }
}
else if (p1 < 0)
  /* The fork failed.  Report failure.  */
  exit(1);
else
  /* This is the parent process.  Wait for the child to complete.  */
  wait(NULL);

printf("Graphs Plotted. Closing Client!\n");

freeaddrinfo(servinfo);
if(serverAddrString!=NULL)
  free(serverAddrString);
if(sendKit!=NULL)
  free(sendKit);
if(rcvKit!=NULL)
  free(rcvKit);
if(sendTimes!=NULL)
  free(sendTimes);
  if(rcvTimes!=NULL)
    free(rcvTimes);


  return 0;
}
```

The code given to gnuplot for throughput graph plotting:

set terminal png
set output "Send_Rcv_Throughput.png"
set title "Throughput Plot"
set xlabel "Time (Seconds)"
set ylabel "Throughput (bits per second)"
plot "plot.txt" using 1:2 with linespoints title "Send Throughput", "plot.txt" using 1:3 with linespoints title "Receive Throughput"

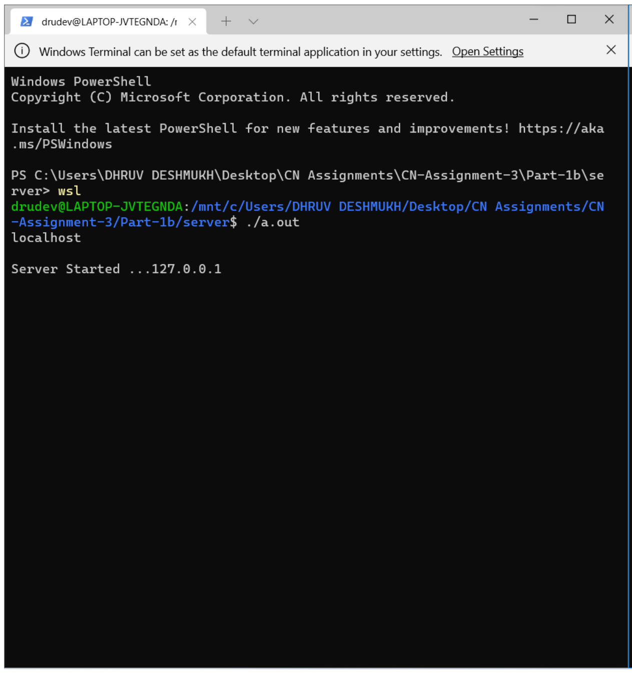
The code given to gnuplot for average delay graph plotting:

set terminal png
set output "Average_Delay.png"
set title "Average Delay Plot"
set xlabel "Time (Seconds)"
set ylabel "Average Delay (microseconds)"
plot "plot.txt" using 1:4 with linespoints title "Average Delay"

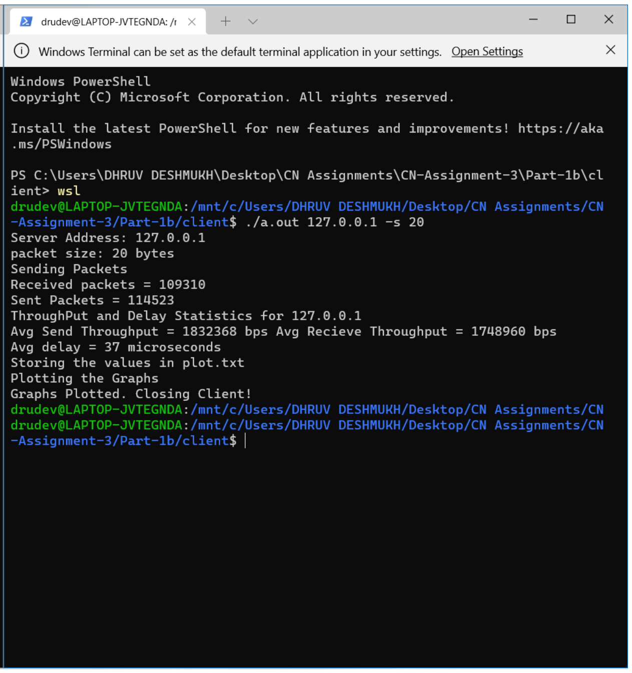
Explanation: The server is the same as part1a only every time a message is received from the client it does not print out that message is received to avoid lag in terminal as lots of messages are sent for throughput calculation. A packet drop mechanism is implemented to drop some packets all together to simulate loss in a real network. The client has two threads send and receive which run for 10 seconds and send and receive messages continuously. The send and receive times are recorded to calculate the RTT which is used to estimate the average end to end delay. The number of packets sent and received each second is also stored to plot the throughput graphs for every second. I use gnuplot to plot these graphs. The detailed explanation of code is provided in the explanation video.

Link for Demo Video(10 mins)-
https://drive.google.com/file/d/1rviDVT2LhEAmHXD-Xypm9SeCYcI_AKUX/view?usp=sharing

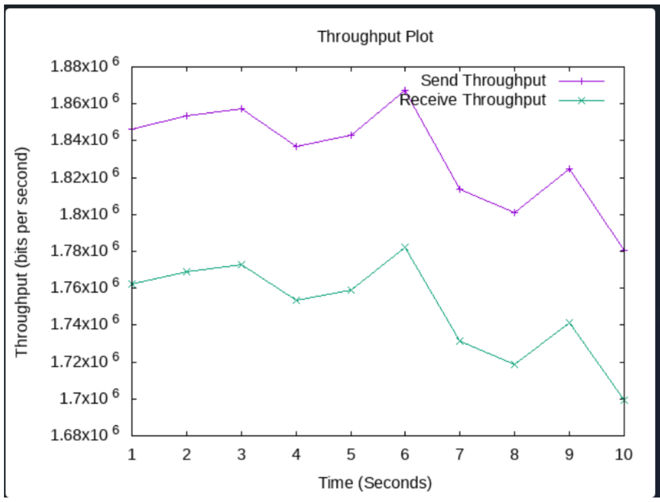
Example:

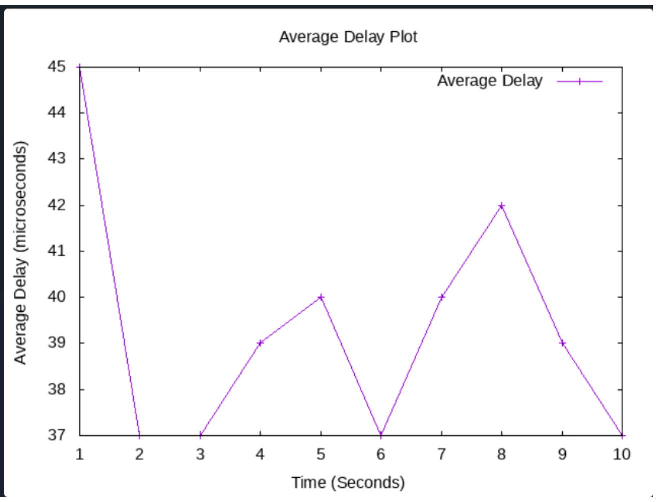

drudev@LAPTOP-JVTEGNDA: /r
Windows Terminal can be set as the default terminal application in your settings. Open Settings

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka
.ms/PSWindows

PS C:\Users\DHRUV DESHMUKH\Desktop\CN Assignments\CN-Assignment-3\Part-1b\se
rver> wsl
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-1b/server$ ./a.out
localhost

Server Started ...127.0.0.1
```


drudev@LAPTOP-JVTEGNDA: /r
Windows Terminal can be set as the default terminal application in your settings. Open Settings

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka
.ms/PSWindows

PS C:\Users\DHRUV DESHMUKH\Desktop\CN Assignments\CN-Assignment-3\Part-1b\cl
ient> wsl
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-1b/client$ ./a.out 127.0.0.1 -s 20
Server Address: 127.0.0.1
packet size: 20 bytes
Sending Packets
Received packets = 109310
Sent Packets = 114523
ThroughPut and Delay Statistics for 127.0.0.1
Avg Send Throughput = 1832368 bps Avg Recieve Throughput = 1748960 bps
Avg delay = 37 microseconds
Storing the values in plot.txt
Plotting the Graphs
Graphs Plotted. Closing Client!
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-1b/client$ |
```





The example is shown and also the plotted graphs. Whenever the delay is high throughput generally goes down. Also average values of throughput and delay are printed.

Part 2:

The code for server:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
```

```
#include<unistd.h>
#include <arpa/inet.h>
#include <time.h>
#include <string.h>
#include <netdb.h>
#include <pthread.h>

#define MAXLINE 1024
#define PORT "3490"
int socketDescriptor;
struct sockaddr_in  clientAddress;
int delay=2;
clock_t processingTimeTable[5] = {2, 5, 10, 15, 20}; //seconds
int tables[10] = {0};
pthread_mutex_t lock;
struct order{
  int tableNum;
  clock_t arrived;
  clock_t timeToProcess;
  clock_t timeToComplete;
  struct sockaddr_in  clientAddress;
  int sent;
  int contents[5];
}orderQueue[10];
int front = -1;
int rear = -1;
clock_t queuingDelay = 0;

int parseNum(char *message, int* it, char end)
{
  int i=*it;
  int id = 0;
  while(message[i] != end)
  {
    id = id*10+(message[i]-'0');
    i++;
  }
  *it=i;
  return id;
}
```

```
void *cast_ipv(struct sockaddr* sa)
{
    return &(((struct sockaddr_in*)sa)->sin_addr);
}

void* rcvOrders(void *arg)
{
  int c=0;
  while(1){
    // printf("\n");
    int addressLength = sizeof(clientAddress);
    char receipt[1024], message[1024];
    int number;
    number = recvfrom(socketDescriptor,message,MAXLINE,0,(struct
sockaddr*)&clientAddress,&addressLength);
    clock_t arrived = clock();
    char ip[INET_ADDRSTRLEN];
    inet_ntop(clientAddress.sin_family,&(clientAddress.sin_addr),ip,
INET_ADDRSTRLEN);
    printf("\n Order received. %s\n ", message);
    int i = 0;
    int tableNum = parseNum(message, &i, ',');
    i++;
    if(tables[tableNum]==0)
    {
      if((rear-front+1)%1000!=0)
      {
        if(front==-1)
          front++;
        rear=(rear+1)%1000;
        printf("Printing order\n%d\n",tableNum);
        orderQueue[rear].clientAddress = clientAddress;
        orderQueue[rear].tableNum = tableNum;
        orderQueue[rear].arrived = arrived;
        orderQueue[rear].timeToProcess = 0;
        // printf("sent %d %d\n",rear,orderQueue[rear].sent);
        while(message[i]!='\0')
        {
          // printf("%d\n",i);
          int item = parseNum(message, &i, '-');
          // printf("%d\n",i);
          i++;
          // printf("%d\n",i);
          int qty = parseNum(message, &i, ',');
          i++;
          // break;
          orderQueue[rear].contents[item] = qty;
          orderQueue[rear].timeToProcess += qty*processingTimeTable[item];
          printf("%d %d\n",item, qty);
        }
        pthread_mutex_lock(&lock);
        orderQueue[rear].timeToComplete = orderQueue[rear].timeToProcess +
queuingDelay;
        tables[tableNum] = orderQueue[rear].timeToComplete;
        queuingDelay += orderQueue[rear].timeToProcess;
        pthread_mutex_unlock(&lock);
      }

    }

    sprintf(receipt,"%d", tables[tableNum]);
    // printf("\n Client's Message: %s ",message);

    if(number<1)
      perror("send error");

    int n = rand();
    if(c==0 || n%c==0)
    {
      sleep(delay);
    }
    // printf("n=%d c=%d\n",n, c);
      // printf("%s\n", message);
    pthread_mutex_lock(&lock);
    sendto(socketDescriptor,receipt,strlen(receipt)+1,0,(struct
sockaddr*)&(orderQueue[rear].clientAddress),addressLength);
    orderQueue[rear].sent = 1;
    pthread_mutex_unlock(&lock);
    // printf("Hello");
    if(c>20)
      c=0;
    else
      c++;
  }
}

void* processOrders(void *arg)
{
  while(1)
  {
    while(front !=-1 && rear !=-1)
    {
      char reply[1024];
      pthread_mutex_lock(&lock);
      int sent = orderQueue[front].sent;
      if(sent == 1) orderQueue[front].sent = 0;
      // printf("sent1 %d %d\n",front,sent);
      pthread_mutex_unlock(&lock);
      if(sent==1)
      {
        struct order curOrder = orderQueue[front];
        sprintf(reply,"%d",curOrder.tableNum);
        printf("Starting to process order for table %d\n",
curOrder.tableNum);
        strcat(reply, "OIP");
        int len = strlen(reply);
        sendto(socketDescriptor,reply,len+1,0,(struct
sockaddr*)&(curOrder.clientAddress),sizeof(clientAddress));
        if(front == rear)
        {
          front=-1;
          rear=-1;
        }
        else
        {
          front = (front+1)%1000;
        }
        for(int i=0;i<curOrder.timeToProcess;i++)
        {
          sleep(1);
          pthread_mutex_lock(&lock);
          queuingDelay--;
          pthread_mutex_unlock(&lock);
          // printf("%d\n",i);
        }
        reply[len-1]='C';
        printf("Order Complete for table %d\n", curOrder.tableNum);
        sendto(socketDescriptor,reply,len+1,0,(struct 
sockaddr*)&(curOrder.clientAddress),sizeof(clientAddress));
        tables[curOrder.tableNum] = 0;

      }
    }
  }
}

void initializeOrderQueue()
{
  for(int i=0;i<1000;i++)
  {
    orderQueue[i].tableNum=0;
    orderQueue[i].arrived = -1;
    orderQueue[i].timeToProcess = 0;
    orderQueue[i].timeToComplete = 0;
    orderQueue[i].sent=0;
    for(int j=0;j<5;j++)
    {
      orderQueue[i].contents[j] = 0;
    }
  }
}

int main(int argc, char* argv[])
{
  char* serverAddrString = NULL;
  int delay = 2;
  for(int i=1;i<argc;i++)
  {
    if(argv[i][0] == '-' && argv[i][1] == 'd')
    {
      i++;
      delay = parseNum(argv[i], NULL,'\0');
    }
    else
    {
      serverAddrString = (char*)malloc(sizeof(*argv[i]));
      strcpy(serverAddrString, argv[i]);
    }
  }

  printf("%s\n", serverAddrString==NULL?"localhost":serverAddrString);
  int number;
  char message[MAXLINE];

  //Get IP from name
  int ev;
  struct addrinfo hints, *servinfo, *it;

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_DGRAM;

  if((ev = getaddrinfo(serverAddrString, PORT, &hints, &servinfo))!=0)
  {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ev));
    return 1;
  }

  for(it = servinfo; it != NULL; it = it->ai_next)
  {
    if((socketDescriptor = socket(AF_INET, it->ai_socktype,
it->ai_protocol))!=-1)
    {
      if(bind(socketDescriptor,it->ai_addr, it->ai_addrlen)==-1)
      {
        close(socketDescriptor);
        continue;
      }
      else
```

```
        break;
    }
  }

  if(it == NULL)
  {
    perror("Failed to acquire a socket!\n");
    return 2;
  }

  initializeOrderQueue();

  char sa[INET_ADDRSTRLEN];
  inet_ntop(it->ai_family,cast_ipv(it->ai_addr),sa, INET_ADDRSTRLEN);
  printf("\nServer Started ...%s\n",sa);

  freeaddrinfo(servinfo);
  pthread_t processThread, rcvThread;

  if (pthread_mutex_init(&lock, NULL) != 0)
  {
    printf("\n mutex init failed\n");
    return 1;
  }

  if(pthread_create(&rcvThread, NULL, rcvOrders, NULL)!=0)
  {
    perror("Could not create Send Thread!\n");
    exit(0);
  }


  if(pthread_create(&processThread, NULL,processOrders, NULL))
  {
    perror("Could not create Receive Thread!\n");
    exit(0);
  }

  //join the threads after completion
  if(pthread_join(rcvThread, NULL)!=0)
  {
    perror("Could not join Send Thread!\n");
    exit(0);
  }
  if(pthread_join(processThread, NULL)!=0)
  {
    perror("Could not join Receive Thread!\n");
    exit(0);
  }
  close(socketDescriptor);
  return 0;
}
```

The code for client:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <netdb.h>
#include <string.h>
#include <pthread.h>
#include <limits.h>
#define MAXLINE 1024
#define PORT "3490"
pthread_mutex_t lock;
char order[MAXLINE];

typedef struct echosendkit{
  struct sockaddr *to;
  int tolen;
  int socfd;
  clock_t* sendTime;
} echoSendKit;
```

```
typedef struct echorcvkit{
  struct sockaddr *from;
  int fromlen;
  int socfd;
  clock_t* sendTime;
  clock_t* rcvTime;
  int timeOut;
} echoRcvKit;



// void delay(int seconds)
// {
//     int milliSeconds = 1000000 * seconds;

//     clock_t startTime = clock();

//     while (clock() < startTime + milliSeconds)
//         ;
// }

int parseNum(char *message, char end)
{
  int i=0;
  int id = 0;
  while(message[i] != end)
  {
    id = id*10+(message[i]-'0');
    i++;
  }
  return id;
}

void *sendEchoMessages(void *arg)
{
  echoSendKit *sendKit = (echoSendKit*)arg;
  char cont = 'y';
  while(cont == 'y')
  {
    //Take order
    int orderLen = 0;
    printf("Place your order. Enter the dish table number number-quantity
separated by comma(eg:1,1-2,2-3. here first 1 is the table number followed
by the items ordered)\n");
    do{
    scanf("%s", order);
    orderLen = strlen(order);
    if(orderLen == 0)
      printf("Order cannot be empty!Please place a valid order\n");
    }while(orderLen == 0);
    order[orderLen++] = ',';
    order[orderLen] = '\0';
    printf("%s\n", order);

    //send message
    pthread_mutex_lock(&lock);
    *(sendKit->sendTime) = clock();
    int temp =
sendto(sendKit->socfd,order,orderLen+1,0,sendKit->to,sendKit->tolen);
    pthread_mutex_unlock(&lock);
    // printf("send: %d\n", temp);
    //delay
    // sleep(sendKit->interval);
    cont = 'n';
  }
}

void *receiveEchoMessages(void *arg)
{
  echoRcvKit *rcvKit = (echoRcvKit*)arg;
  int expecting = 1;
  while(expecting)
  {
    //check for timeout for each message
    pthread_mutex_lock(&lock);
    if(*(rcvKit->sendTime)!=-1)
    {
      if(*(rcvKit->rcvTime)==0 &&
clock()>*(rcvKit->sendTime)+(1000000*rcvKit->timeOut))
```

```
      {
        // printf("rcv: %ld ", rcvKit->rcvTime);
        printf("Resending order\n");
        *(rcvKit->sendTime) = clock();
        int temp =
sendto(rcvKit->socfd,order,strlen(order)+1,0,rcvKit->from,rcvKit->fromlen)
;
      }
    }
    pthread_mutex_unlock(&lock);
    // printf("d%d\n", expecting);
    //initialize message
    char message[1000];

    //receive message
    int rcvlen =
recvfrom(rcvKit->socfd,message,10,MSG_DONTWAIT,rcvKit->from,&(rcvKit->from
len));
    if(rcvlen!=-1)
    {
      clock_t rcvTime = clock();
      // int id = parseNum(message, '$');
      // printf("outside\n");
      pthread_mutex_lock(&lock);
      // printf("inside\n");
      *(rcvKit->rcvTime) = rcvTime;
      printf("Please wait for %s s\n", message);
      int timer = parseNum(message, '\0');
      do{
      rcvlen =
recvfrom(rcvKit->socfd,message,4,0,rcvKit->from,&(rcvKit->fromlen));
      }while(message[rcvlen-2]>='0' && message[rcvlen-2]<='9');
      if(rcvlen!=-1)
      {
        printf("Order is being processed\n");
      }
      do{
      rcvlen =
recvfrom(rcvKit->socfd,message,4,0,rcvKit->from,&(rcvKit->fromlen));
      }while(message[rcvlen-2]>='0'&&message[rcvlen-2]<='9');
      if(rcvlen!=-1)
      {
        printf("Order is completed\n");
      }
      expecting=0;
      // printf("end\n");
      pthread_mutex_unlock(&lock);
    }
    // printf("a%d\n", expecting);
  }
}

void initializeEchoSendKit(struct sockaddr *to, int tolen, int socfd,
clock_t* sendTime, echoSendKit *sendKit)
{
  sendKit->to = to;
  sendKit->tolen = tolen;
  sendKit->socfd = socfd;
  sendKit->sendTime = sendTime;
}

void initializeEchoRcvKit(struct sockaddr *from, int fromlen, int socfd,
clock_t* sendTime, clock_t* rcvTime, int timeOut, echoRcvKit *rcvKit)
{
  rcvKit->from = from;
  rcvKit->fromlen = fromlen;
  rcvKit->socfd = socfd;
  rcvKit->sendTime = sendTime;
  rcvKit->rcvTime = rcvTime;
  rcvKit->timeOut = timeOut;
}

int main(int argc, char* argv[]){
  //params
  int timeOut = 2; //in seconds
  char* serverAddrString = NULL;
  int addressLength;
  // make a socket for the client
  int socketDescriptor = socket(AF_INET, SOCK_DGRAM, 0);
  //Parse all the arguments
  for(int i=1;i<argc;i++)
  {
    if(argv[i][0] == '-' && argv[i][1] == 't')
    {
      i++;
      timeOut = parseNum(argv[i],'\0');
    }
    else
    {
      serverAddrString = (char*)malloc(sizeof(*argv[i]));
      strcpy(serverAddrString, argv[i]);
    }
  }

  printf("Server Address: %s\n",
serverAddrString==NULL?"localhost":serverAddrString);
  printf("timeout: %d s\n", timeOut);

  //Get IP from name
  int ev;
  struct addrinfo hints, *servinfo;

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_DGRAM;

  if((ev = getaddrinfo(serverAddrString, PORT, &hints, &servinfo))!=0)
  {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ev));
    return 1;
  }

  //Allocate and initialize the time arrays
  clock_t sendTime = -1;
  clock_t rcvTime = 0;

  // set the server info
  // struct sockaddr_in serverAddress;
  // serverAddress.sin_family = AF_INET;
  // serverAddress.sin_addr.s_addr = servinfo[0].; //localhost
  // serverAddress.sin_port = htons(PORT);

  // addressLength = sizeof(serverAddress);

  //Allocate and initialize the structures passed to the send and receive
functions
  echoSendKit *sendKit = (echoSendKit*)malloc(sizeof(echoSendKit));
  echoRcvKit *rcvKit = (echoRcvKit*)malloc(sizeof(echoRcvKit));
  initializeEchoSendKit(servinfo->ai_addr, servinfo->ai_addrlen,
socketDescriptor, &sendTime, sendKit);

  initializeEchoRcvKit(servinfo->ai_addr, servinfo->ai_addrlen,
socketDescriptor, &sendTime, &rcvTime, timeOut, rcvKit);

  //Declare the threads
  pthread_t sendThread, rcvThread;

  //Initialize a lock
  if (pthread_mutex_init(&lock, NULL) != 0)
  {
      printf("\n mutex init failed\n");
      return 1;
  }

  //Start the clock
  clock_t startTime = clock();

  //Create the threads for the send and receive functions
  if(pthread_create(&sendThread, NULL, sendEchoMessages, sendKit)!=0)
  {
    perror("Could not create Send Thread!\n");
    exit(0);
  }


  if(pthread_create(&rcvThread, NULL, receiveEchoMessages, rcvKit))
  {
    perror("Could not create Receive Thread!\n");
    exit(0);
  }

  //join the threads after completion
  if(pthread_join(sendThread, NULL)!=0)
  {
    perror("Could not join Send Thread!\n");
    exit(0);
  }
  if(pthread_join(rcvThread, NULL)!=0)
  {
    perror("Could not join Receive Thread!\n");
    exit(0);
  }

  //Get the Statistics
  // char ip4[INET_ADDRSTRLEN];
  // inet_ntop(AF_INET, &(((struct sockaddr_in
*)servinfo->ai_addr)->sin_addr), ip4, INET_ADDRSTRLEN);
  // clock_t maxrtt = 0, avgrtt = 0;
  // clock_t minrtt = INT_MAX;
  // int lost = 0, rcved = 0, percentloss = 0;
  // for(int i=0;i<numberOfMessages;i++)
  // {
  //   clock_t rtt = difftime(rcvKit->rcvTimes[i], rcvKit->sendTimes[i]);
  //   if(rtt>=0)
  //   {
  //     if (rtt > maxrtt)
  //       maxrtt = rtt;
  //     if (rtt < minrtt)
  //       minrtt = rtt;
  //     avgrtt+=rtt;
  //   }
  //   else
  //     lost++;
  // }
  // if(minrtt == INT_MAX)
  //   minrtt = 0;
  // avgrtt/=numberOfMessages;
  // rcved = numberOfMessages - lost;
  // percentloss = (lost*100)/numberOfMessages;
  // printf("Ping statistics for %s\n", ip4);
  // printf("\tPackets: Sent = %d, Recieved = %d, Lost=%d (%d%% loss)\n",
numberOfMessages, rcved, lost, percentloss);
  // printf("Approximate round trip times\n");
  // printf("\tMinimum = %ld, Maximum = %ld, Average = %ld\n", minrtt,
maxrtt, avgrtt);

  printf("Closing Client!\n");
  // sendto(socketDescriptor,sendMessage,MAXLINE,0,(struct
sockaddr*)&serverAddress,addressLength);
  // recvfrom(socketDescriptor,recvMessage,MAXLINE,0,NULL,NULL);

  // printf("\nServer's Echo : %s\n",recvMessage);

  freeaddrinfo(servinfo);
  if(serverAddrString!=NULL)
    free(serverAddrString);
  if(sendKit!=NULL)
    free(sendKit);
  if(rcvKit!=NULL)
    free(rcvKit);

  return 0;
}
```

Explanation:

The current ordering system using waiters in a restaurant has following problems:

- Sometimes, wrong order is received.
- Don’t get a correct estimate of how much time it will take.
- Experience delay and don’t get order in expected time.

My application tries to tackle these problems.

Basic overview of my application:

- Online ordering and notification service.
- Specifically implemented for scenario of restaurants.
- Can be easily adapted for similar use in different sectors which involves queuing of orders/requests.
- Clients send their order to server.

- The server will process these orders one by one and give proper notifications to the clients.

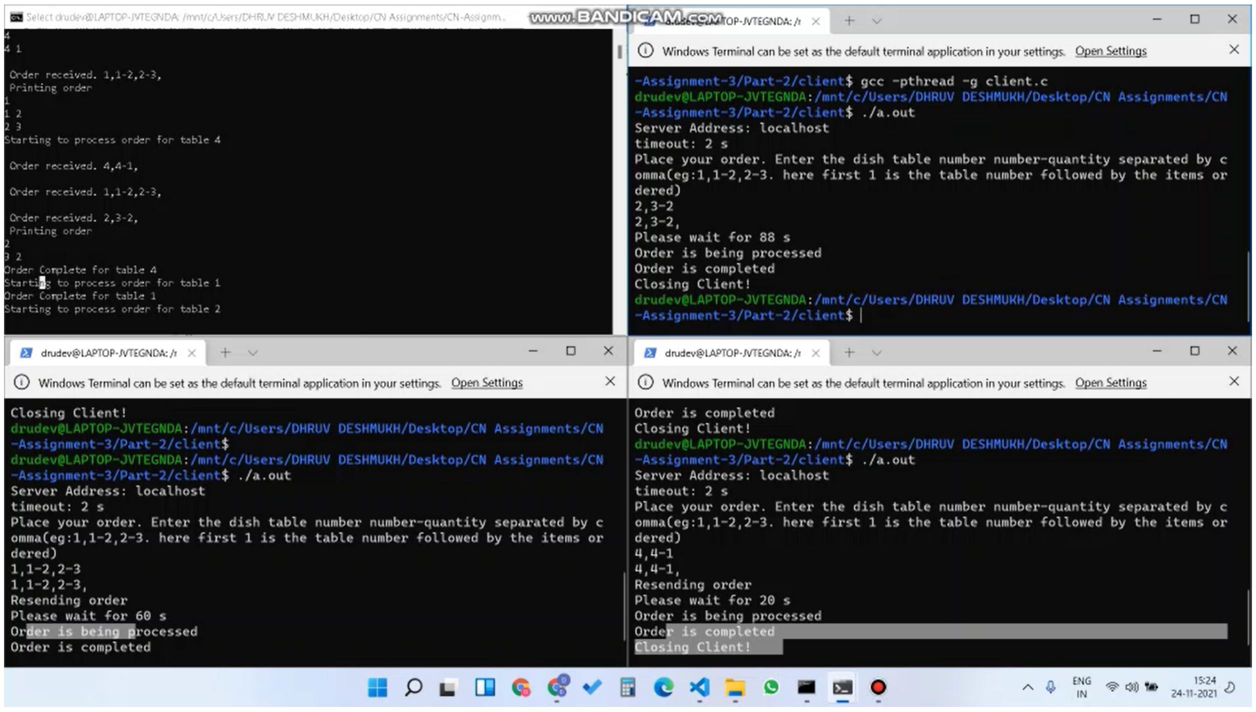
The two features implemented:

1. Reliability:
    a. If on ordering acknowledgement is not received within timeout then order is sent again till we get a reply.
    b. The server on receiving the order initially will calculate an estimated time to complete it and send that to the client.
    c. If the order is received again then it will just send back this time.
2. Server that does some processing:(Server is not dumb)
    a. The echo server was dumb and just sent back the message again.
    b. This server parses the message, calculates a time estimate and sends it to client
    c. Queues the order
    d. Process the order
    e. Notifies the client when order processing starts and is completed

The detailed code explanation and example run are shown in demo video.

Link for the Demo Video(16.5 mins):
https://drive.google.com/file/d/1wLJFJtEYQmgRa_D0fjwHYqN0jDAagpf0/view?usp=sharing

Example Run:



```
4
4 1

 Order received. 1,1-2,2-3,
 Printing order
1
1 2
2 3
Starting to process order for table 4

 Order received. 4,4-1,

 Order received. 1,1-2,2-3,

 Order received. 2,3-2,
 Printing order
2
3 2
Order Complete for table 4
Starting to process order for table 1
Order Complete for table 1
Starting to process order for table 2
```

```
-Assignment-3/Part-2/client$ gcc -pthread -g client.c
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-2/client$ ./a.out
Server Address: localhost
timeout: 2 s
Place your order. Enter the dish table number number-quantity separated by c
omma(eg:1,1-2,2-3. here first 1 is the table number followed by the items or
dered)
2,3-2
2,3-2,
Please wait for 88 s
Order is being processed
Order is completed
Closing Client!
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-2/client$
```

```
Closing Client!
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-2/client$
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-2/client$ ./a.out
Server Address: localhost
timeout: 2 s
Place your order. Enter the dish table number number-quantity separated by c
omma(eg:1,1-2,2-3. here first 1 is the table number followed by the items or
dered)
1,1-2,2-3
1,1-2,2-3,
Resending order
Please wait for 60 s
Order is being processed
Order is completed
```

```
Order is completed
Closing Client!
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-2/client$ ./a.out
Server Address: localhost
timeout: 2 s
Place your order. Enter the dish table number number-quantity separated by c
omma(eg:1,1-2,2-3. here first 1 is the table number followed by the items or
dered)
4,4-1
4,4-1,
Resending order
Please wait for 20 s
Order is being processed
Order is completed
Closing Client!
```

Part 3:

The code for server:

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

#define MAXLINE 1024
#define PORT "3495"

// void delay(int seconds)
// {
//     int milliSeconds = 100000 * seconds;

//     clock_t startTime = clock();

//     while (clock() < startTime + milliSeconds)
//         ;
// }

int parseNum(char *message, char end)
{
  int i=0;
  int id = 0;
  while(message[i] != end)
  {
    id = id*10+(message[i]-'0');
    i++;
  }
  return id;
}
```

```
void *cast_ipv(struct sockaddr* sa)
{
  if(sa->sa_family == AF_INET)
  {
    return &(((struct sockaddr_in*)sa)->sin_addr);
  }
  return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

int main(int argc, char* argv[])
{
  char* serverAddrString = NULL;
  int delay = 2;
  for(int i=1;i<argc;i++)
  {
    if(argv[i][0] == '-' && argv[i][1] == 'd')
    {
      i++;
      delay = parseNum(argv[i],'\0');
    }
    else
    {
      serverAddrString = (char*)malloc(sizeof(*argv[i]));
      strcpy(serverAddrString, argv[i]);
    }
  }
  int socketDescriptor;
  int number;
  int addressLength;
  char message[MAXLINE];

  struct sockaddr_storage  clientAddress;
  //Get IP from name
  int ev;
  struct addrinfo hints, *servinfo, *it;

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_DGRAM;
  if((ev = getaddrinfo(serverAddrString, PORT, &hints, &servinfo))!=0)
  {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ev));
    return 1;
  }

  for(it = servinfo; it != NULL; it = it->ai_next)
  {
    if((socketDescriptor = socket(it->ai_family, it->ai_socktype,
it->ai_protocol))!=-1)
    {
      if(bind(socketDescriptor,it->ai_addr, it->ai_addrlen)==-1)
      {
        close(socketDescriptor);
        continue;
      }
      else
        break;
    }
  }

  if(it == NULL)
  {
    perror("Failed to acquire a socket!\n");
    return 2;
  }
  char sa[INET6_ADDRSTRLEN];
  inet_ntop(it->ai_family,cast_ipv(it->ai_addr),sa, INET6_ADDRSTRLEN);
  printf("\nServer Started ...%s\n",sa);

  freeaddrinfo(servinfo);

  int c=0;

  while(1){
    // printf("\n");
    addressLength = sizeof(clientAddress);

    number = recvfrom(socketDescriptor,message,MAXLINE,0,(struct
sockaddr*)&clientAddress,&addressLength);
```

```
    char ip[INET6_ADDRSTRLEN];
    inet_ntop(clientAddress.ss_family,cast_ipv((struct
sockaddr*)&clientAddress),ip, INET6_ADDRSTRLEN);
    printf("\n Message from client %s \n ", ip);
    // printf("\n Client's Message: %s ",message);

    if(number<1)
      perror("send error");

    int n = rand();
    if(c>0 && n%c==0)
    {
      sleep(delay);
    }
    printf("n=%d c=%d\n",n, c);
      // printf("%s\n", message);
      sendto(socketDescriptor,message,number,0,(struct
sockaddr*)&clientAddress,addressLength);
      if(c>20)
        c=0;
      else
        c++;
  }
  close(socketDescriptor);
  return 0;
}
```

The code for client:

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

```c
#include <limits.h>

#define MAXLINE 1024
#define PORT "3495"
pthread_mutex_t lock;

typedef struct echosendkit{
  int interval;
  int packetSize;
  int numberOfMessages;
  struct sockaddr *to;
  int tolen;
  int socfd;
  clock_t *sendTimes;
} echoSendKit;

typedef struct echorcvkit{
  int interval;
  int packetSize;
  int numberOfMessages;
  struct sockaddr *from;
  int fromlen;
  int socfd;
  clock_t *sendTimes;
  clock_t *rcvTimes;
  int timeOut;
} echoRcvKit;



// void delay(int seconds)
// {
//     int milliSeconds = 1000000 * seconds;

//     clock_t startTime = clock();

//     while (clock() < startTime + milliSeconds)
//         ;
// }
```

```
int parseNum(char *message, char end)
{
  int i=0;
  int id = 0;
  while(message[i] != end)
  {
    id = id*10+(message[i]-'0');
    i++;
  }
  return id;
}

void *cast_ipv(struct sockaddr* sa)
{
  if(sa->sa_family == AF_INET)
  {
    return &(((struct sockaddr_in*)sa)->sin_addr);
  }
  return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

void *sendEchoMessages(void *arg)
{
  echoSendKit *sendKit = (echoSendKit*)arg;
  for(int i=0;i<sendKit->numberOfMessages;i++)
  {
    //create message
    char* message = (char*)malloc(sizeof(char)*sendKit->packetSize);
    int len = sprintf(message, "%d", i);
    int paddingLen = sendKit->packetSize-len;
    char *padding = (char*)malloc(sizeof(char)*paddingLen);
    memset(padding, '$', paddingLen-1);
    strcat(message, padding);

    //send message
    pthread_mutex_lock(&lock);
    sendKit->sendTimes[i] = clock();
    int temp =
sendto(sendKit->socfd,message,sendKit->packetSize,0,sendKit->to,sendKit->t
olen);
```

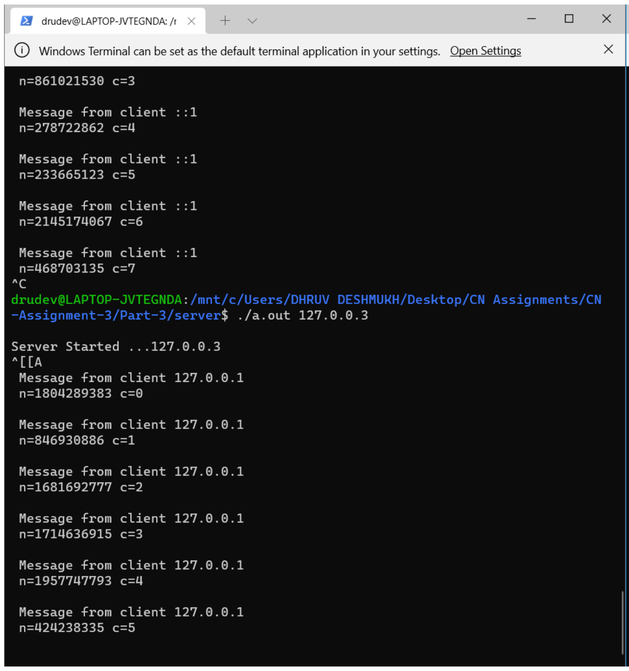
```
    pthread_mutex_unlock(&lock);
    // printf("send: %d\n", temp);
    //delay
    sleep(sendKit->interval);
    if(message!=NULL)
      free(message);
    if(padding!=NULL)
      free(padding);
  }
}

void *receiveEchoMessages(void *arg)
{
  echoRcvKit *rcvKit = (echoRcvKit*)arg;
  int expecting = rcvKit->numberOfMessages;
  while(expecting>0)
  {
    //check for timeout for each message
    pthread_mutex_lock(&lock);
    for(int i=0;i<rcvKit->numberOfMessages;i++)
    {
      if(rcvKit->sendTimes[i]!=-1)
      {
        if(rcvKit->rcvTimes[i]!=-2 && (rcvKit->rcvTimes[i]==-1 ||
(rcvKit->rcvTimes[i]==0 &&
clock()>rcvKit->sendTimes[i]+(1000000*rcvKit->timeOut))))
        {
          // printf("rcv: %ld ", rcvKit->rcvTimes[i]);
          rcvKit->rcvTimes[i] = -2;
          expecting--;
          printf("Request Timed Out.\n");
        }
      }

    }
    // printf("end2\n");
    pthread_mutex_unlock(&lock);
    // printf("d%d\n", expecting);
    //initialize message
    char* message = (char*)malloc(sizeof(char)*rcvKit->packetSize);
    //receive message
    int rcvlen =
recvfrom(rcvKit->socfd,message,rcvKit->packetSize,MSG_DONTWAIT,rcvKit->fro
m,&(rcvKit->fromlen));
    if(rcvlen!=-1)
    {
      clock_t rcvTime = clock();
      int id = parseNum(message, '$');
      // printf("outside\n");
      pthread_mutex_lock(&lock);
      // printf("inside\n");
      if(rcvKit->rcvTimes[id]!=-2)
      {
        rcvKit->rcvTimes[id] = rcvTime;
        if(rcvTime>rcvKit->sendTimes[id]+(1000000*rcvKit->timeOut))
          rcvKit->rcvTimes[id]=-1;
        else
        {
          clock_t rtt =
difftime(rcvKit->rcvTimes[id],rcvKit->sendTimes[id]);
          char ip[INET6_ADDRSTRLEN];
          inet_ntop(rcvKit->from->sa_family, cast_ipv(rcvKit->from),
ip,INET6_ADDRSTRLEN);
          printf("Reply from %s : bytes=%d rtt=%ld\n", ip, rcvlen, rtt);
          expecting--;
        }
      }
      // printf("end\n");
      pthread_mutex_unlock(&lock);
    }
    // printf("a%d\n", expecting);
  }
}

void initializeEchoSendKit(int interval, int packetSize, int
numberOfMessages, struct sockaddr *to, int tolen, int socfd, clock_t*
sendTimes, echoSendKit *sendKit)
{
  sendKit->interval = interval;
```

```c
  sendKit->packetSize = packetSize;
  sendKit->numberOfMessages = numberOfMessages;
  sendKit->to = to;
  sendKit->tolen = tolen;
  sendKit->socfd = socfd;
  sendKit->sendTimes = sendTimes;
}

void initializeEchoRcvKit(int interval, int packetSize, int
numberOfMessages, struct sockaddr *from, int fromlen, int socfd, clock_t*
sendTimes, clock_t* rcvTimes, int timeOut, echoRcvKit *rcvKit)
{
  rcvKit->interval = interval;
  rcvKit->packetSize = packetSize;
  rcvKit->numberOfMessages = numberOfMessages;
  rcvKit->from = from;
  rcvKit->fromlen = fromlen;
  rcvKit->socfd = socfd;
  rcvKit->sendTimes = sendTimes;
  rcvKit->rcvTimes = rcvTimes;
  rcvKit->timeOut = timeOut;
}


int main(int argc, char* argv[]){
  //params
  int interval = 2; //in seconds
  int packetSize = 8; //in bytes
  int numberOfMessages = 6;
  int timeOut = 4; //in seconds
  char* serverAddrString = NULL;
  int addressLength;
  // make a socket for the client
  int socketDescriptor;

  //Parse all the arguments
  for(int i=1;i<argc;i++)
  {
    if(argv[i][0] == '-' && argv[i][1] == 'i')
    {
      i++;
      interval = parseNum(argv[i],'\0');
    }
    else if(argv[i][0] == '-' && argv[i][1] == 's')
    {
      i++;
      packetSize = parseNum(argv[i],'\0');
    }
    else if(argv[i][0] == '-' && argv[i][1] == 'n')
    {
      i++;
      numberOfMessages = parseNum(argv[i],'\0');
    }
    else if(argv[i][0] == '-' && argv[i][1] == 't')
    {
      i++;
      timeOut = parseNum(argv[i],'\0');
    }
    else
    {
      serverAddrString = (char*)malloc(sizeof(*argv[i]));
      strcpy(serverAddrString, argv[i]);
    }
  }

  printf("Server Address: %s\n",
serverAddrString==NULL?"localhost":serverAddrString);
  printf("interval: %d s\n", interval);
  printf("packet size: %d bytes\n", packetSize);
  printf("number of messages: %d\n", numberOfMessages);
  printf("timeout: %d s\n", timeOut);

  //Get IP from name
  int ev;
  struct addrinfo hints, *servinfo, *it;

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_DGRAM;
  if((ev = getaddrinfo(serverAddrString, PORT, &hints, &servinfo))!=0)
  {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ev));
    return 1;
  }

  for(it = servinfo; it != NULL; it = it->ai_next)
  {
    if((socketDescriptor = socket(it->ai_family, it->ai_socktype,
it->ai_protocol))!=-1)
    {
      break;
    }
  }

  if(it == NULL)
  {
    perror("Failed to acquire a socket!\n");
    return 2;
  }

  //Allocate and initialize the time arrays
  clock_t *sendTimes = (clock_t*)malloc(sizeof(clock_t)*numberOfMessages);
  clock_t *rcvTimes = (clock_t*)calloc(numberOfMessages,sizeof(clock_t));
  for(int i=0;i<numberOfMessages;i++)
  {
    sendTimes[i] = -1;
  }

  // set the server info
  // struct sockaddr_in serverAddress;
  // serverAddress.sin_family = AF_INET;
  // serverAddress.sin_addr.s_addr = servinfo[0].; //localhost
  // serverAddress.sin_port = htons(PORT);

  // addressLength = sizeof(serverAddress);

  //Allocate and initialize the structures passed to the send and receive
functions
  echoSendKit *sendKit = (echoSendKit*)malloc(sizeof(echoSendKit));
  echoRcvKit *rcvKit = (echoRcvKit*)malloc(sizeof(echoRcvKit));
  initializeEchoSendKit(interval, packetSize, numberOfMessages,
it->ai_addr, it->ai_addrlen, socketDescriptor, sendTimes, sendKit);

  initializeEchoRcvKit(interval, packetSize, numberOfMessages,
it->ai_addr, it->ai_addrlen, socketDescriptor, sendTimes, rcvTimes,
timeOut, rcvKit);

  //Declare the threads
  pthread_t sendThread, rcvThread;

  //Initialize a lock
  if (pthread_mutex_init(&lock, NULL) != 0)
  {
      printf("\n mutex init failed\n");
      return 1;
  }

  //Start the clock
  clock_t startTime = clock();

  //Create the threads for the send and receive functions
  if(pthread_create(&sendThread, NULL, sendEchoMessages, sendKit)!=0)
  {
    perror("Could not create Send Thread!\n");
    exit(0);
  }


  if(pthread_create(&rcvThread, NULL, receiveEchoMessages, rcvKit))
  {
    perror("Could not create Receive Thread!\n");
    exit(0);
  }

  //join the threads after completion
  if(pthread_join(sendThread, NULL)!=0)
  {
    perror("Could not join Send Thread!\n");
    exit(0);
  }
  if(pthread_join(rcvThread, NULL)!=0)
  {
    perror("Could not join Receive Thread!\n");
    exit(0);
  }

  //Get the Statistics
  char ip[INET6_ADDRSTRLEN];
  inet_ntop(it->ai_family, it->ai_addr, ip, INET6_ADDRSTRLEN);
  clock_t maxrtt = 0, avgrtt = 0;
  clock_t minrtt = INT_MAX;
  int lost = 0, rcved = 0, percentloss = 0;
  for(int i=0;i<numberOfMessages;i++)
  {
    clock_t rtt = difftime(rcvKit->rcvTimes[i], rcvKit->sendTimes[i]);
    if(rtt>=0)
    {
      if (rtt > maxrtt)
        maxrtt = rtt;
      if (rtt < minrtt)
        minrtt = rtt;
      avgrtt+=rtt;
    }
    else
      lost++;
  }
  if(minrtt == INT_MAX)
    minrtt = 0;
  avgrtt/=numberOfMessages;
  rcved = numberOfMessages - lost;
  percentloss = (lost*100)/numberOfMessages;
  printf("Ping statistics for %s\n", ip);
  printf("\tPackets: Sent = %d, Recieved = %d, Lost=%d (%d%% loss)\n",
numberOfMessages, rcved, lost, percentloss);
  printf("Approximate round trip times\n");
  printf("\tMinimum = %ld, Maximum = %ld, Average = %ld\n", minrtt,
maxrtt, avgrtt);

  printf("Closing Client!\n");
  // sendto(socketDescriptor,sendMessage,MAXLINE,0,(struct
sockaddr*)&serverAddress,addressLength);
  // recvfrom(socketDescriptor,recvMessage,MAXLINE,0,NULL,NULL);

  // printf("\nServer's Echo : %s\n",recvMessage);

  freeaddrinfo(servinfo);
  if(serverAddrString!=NULL)
    free(serverAddrString);
  if(sendKit!=NULL)
    free(sendKit);
  if(rcvKit!=NULL)
    free(rcvKit);
  if(sendTimes!=NULL)
    free(sendTimes);
  if(rcvTimes!=NULL)
    free(rcvTimes);


  return 0;
}
```

Explanation: The code is similar to Part-1a only changes as given in hints have been made to make the client server work for ipv4 and ipv6 addresses both. The details of changes made are in the explanation video.

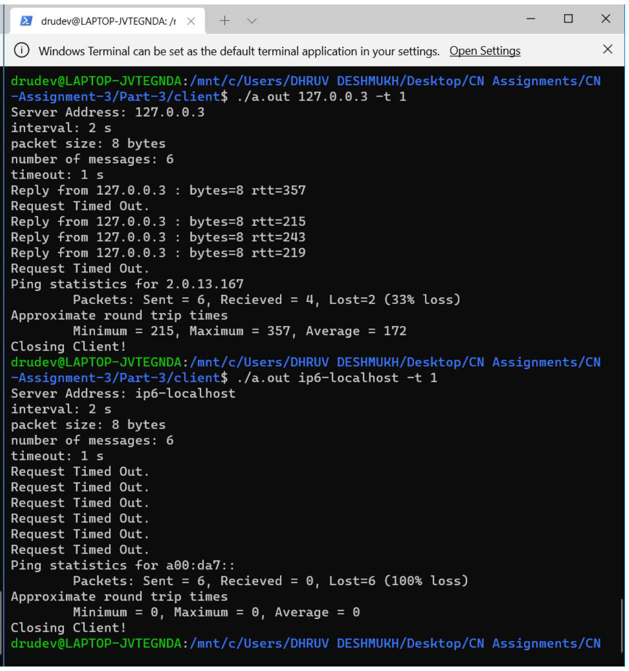
Link of Demo Video(7 min):
https://drive.google.com/file/d/1dB3t8Mgt2skerHIkU4o9S_YfpAthBUd1/view?usp=sharing

Example run and output:



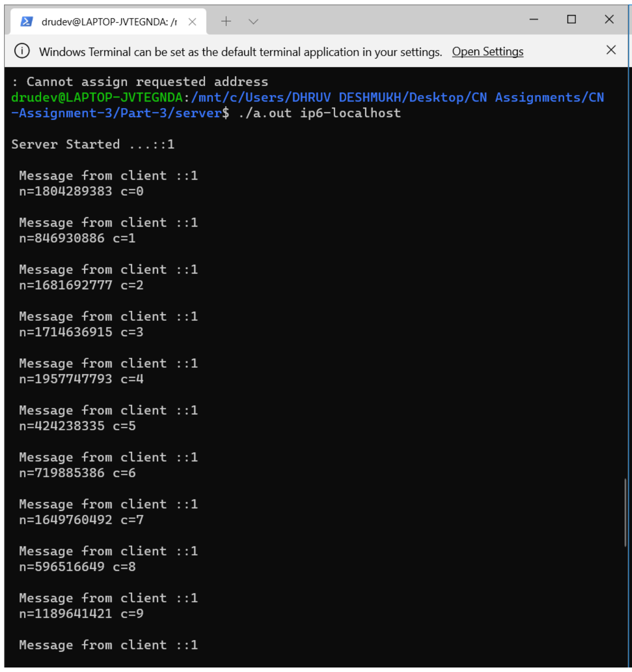
Windows Terminal can be set as the default terminal application in your settings. Open Settings

```
 n=861021530 c=3

 Message from client ::1
 n=278722862 c=4

 Message from client ::1
 n=233665123 c=5

 Message from client ::1
 n=2145174067 c=6

 Message from client ::1
 n=468703135 c=7
^C
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-3/server$ ./a.out 127.0.0.3

Server Started ...127.0.0.3
^[[A
 Message from client 127.0.0.1
 n=1804289383 c=0

 Message from client 127.0.0.1
 n=846930886 c=1

 Message from client 127.0.0.1
 n=1681692777 c=2

 Message from client 127.0.0.1
 n=1714636915 c=3

 Message from client 127.0.0.1
 n=1957747793 c=4

 Message from client 127.0.0.1
 n=424238335 c=5
```



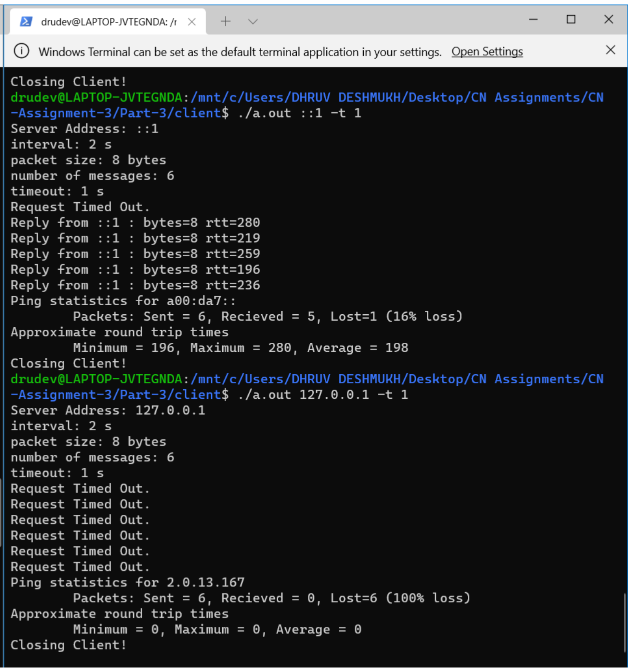
Windows Terminal can be set as the default terminal application in your settings. Open Settings

```
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-3/client$ ./a.out 127.0.0.3 -t 1
Server Address: 127.0.0.3
interval: 2 s
packet size: 8 bytes
number of messages: 6
timeout: 1 s
Reply from 127.0.0.3 : bytes=8 rtt=357
Request Timed Out.
Reply from 127.0.0.3 : bytes=8 rtt=215
Reply from 127.0.0.3 : bytes=8 rtt=243
Reply from 127.0.0.3 : bytes=8 rtt=219
Request Timed Out.
Ping statistics for 2.0.13.167
        Packets: Sent = 6, Recieved = 4, Lost=2 (33% loss)
Approximate round trip times
        Minimum = 215, Maximum = 357, Average = 172
Closing Client!
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-3/client$ ./a.out ip6-localhost -t 1
Server Address: ip6-localhost
interval: 2 s
packet size: 8 bytes
number of messages: 6
timeout: 1 s
Request Timed Out.
Request Timed Out.
Request Timed Out.
Request Timed Out.
Request Timed Out.
Request Timed Out.
Ping statistics for a00:da7::
        Packets: Sent = 6, Recieved = 0, Lost=6 (100% loss)
Approximate round trip times
        Minimum = 0, Maximum = 0, Average = 0
Closing Client!
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
```



Windows Terminal can be set as the default terminal application in your settings. Open Settings

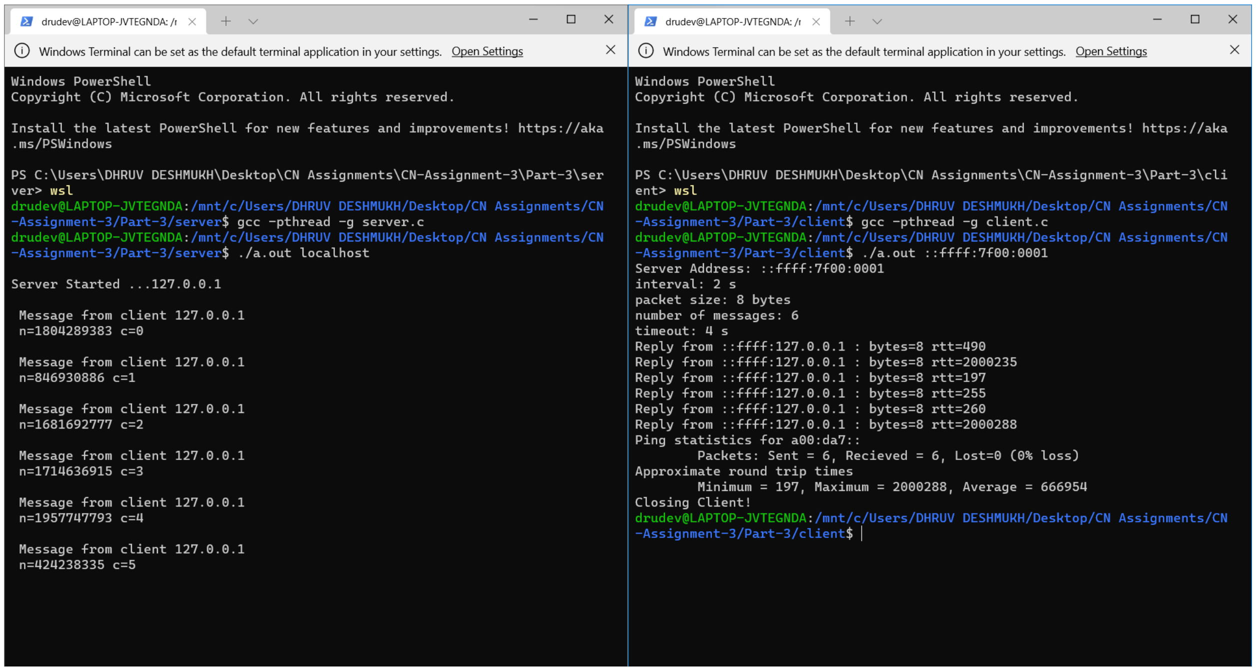
```
: Cannot assign requested address
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-3/server$ ./a.out ip6-localhost

Server Started ...::1

 Message from client ::1
 n=1804289383 c=0

 Message from client ::1
 n=846930886 c=1

 Message from client ::1
 n=1681692777 c=2

 Message from client ::1
 n=1714636915 c=3

 Message from client ::1
 n=1957747793 c=4

 Message from client ::1
 n=424238335 c=5

 Message from client ::1
 n=719885386 c=6

 Message from client ::1
 n=1649760492 c=7

 Message from client ::1
 n=596516649 c=8

 Message from client ::1
 n=1189641421 c=9

 Message from client ::1
```



Windows Terminal can be set as the default terminal application in your settings. Open Settings

```
Closing Client!
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-3/client$ ./a.out ::1 -t 1
Server Address: ::1
interval: 2 s
packet size: 8 bytes
number of messages: 6
timeout: 1 s
Request Timed Out.
Reply from ::1 : bytes=8 rtt=280
Reply from ::1 : bytes=8 rtt=219
Reply from ::1 : bytes=8 rtt=259
Reply from ::1 : bytes=8 rtt=196
Reply from ::1 : bytes=8 rtt=236
Ping statistics for a00:da7::
        Packets: Sent = 6, Recieved = 5, Lost=1 (16% loss)
Approximate round trip times
        Minimum = 196, Maximum = 280, Average = 198
Closing Client!
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-3/client$ ./a.out 127.0.0.1 -t 1
Server Address: 127.0.0.1
interval: 2 s
packet size: 8 bytes
number of messages: 6
timeout: 1 s
Request Timed Out.
Request Timed Out.
Request Timed Out.
Request Timed Out.
Request Timed Out.
Request Timed Out.
Ping statistics for 2.0.13.167
        Packets: Sent = 6, Recieved = 0, Lost=6 (100% loss)
Approximate round trip times
        Minimum = 0, Maximum = 0, Average = 0
Closing Client!
```

In the first example the server is set up at ipv4 address so only when ipv4 address is given we get response else there is timeout. In the second example ipv6 address is used for the server and hence the request to that address works. This shows that both ipv6 and ipv4 are supported.


drudev@LAPTOP-JVTEGNDA: /r
Windows Terminal can be set as the default terminal application in your settings. Open Settings
drudev@LAPTOP-JVTEGNDA: /r
Windows Terminal can be set as the default terminal application in your settings. Open Settings

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka
.ms/PSWindows

PS C:\Users\DHRUV DESHMUKH\Desktop\CN Assignments\CN-Assignment-3\Part-3\ser
ver> wsl
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-3/server$ gcc -pthread -g server.c
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-3/server$ ./a.out localhost

Server Started ...127.0.0.1

 Message from client 127.0.0.1
 n=1804289383 c=0

 Message from client 127.0.0.1
 n=846930886 c=1

 Message from client 127.0.0.1
 n=1681692777 c=2

 Message from client 127.0.0.1
 n=1714636915 c=3

 Message from client 127.0.0.1
 n=1957747793 c=4

 Message from client 127.0.0.1
 n=424238335 c=5
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka
.ms/PSWindows

PS C:\Users\DHRUV DESHMUKH\Desktop\CN Assignments\CN-Assignment-3\Part-3\cli
ent> wsl
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-3/client$ gcc -pthread -g client.c
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-3/client$ ./a.out ::ffff:7f00:0001
Server Address: ::ffff:7f00:0001
interval: 2 s
packet size: 8 bytes
number of messages: 6
timeout: 4 s
Reply from ::ffff:127.0.0.1 : bytes=8 rtt=490
Reply from ::ffff:127.0.0.1 : bytes=8 rtt=2000235
Reply from ::ffff:127.0.0.1 : bytes=8 rtt=197
Reply from ::ffff:127.0.0.1 : bytes=8 rtt=255
Reply from ::ffff:127.0.0.1 : bytes=8 rtt=260
Reply from ::ffff:127.0.0.1 : bytes=8 rtt=2000288
Ping statistics for a00:da7::
        Packets: Sent = 6, Recieved = 6, Lost=0 (0% loss)
Approximate round trip times
        Minimum = 197, Maximum = 2000288, Average = 666954
Closing Client!
drudev@LAPTOP-JVTEGNDA:/mnt/c/Users/DHRUV DESHMUKH/Desktop/CN Assignments/CN
-Assignment-3/Part-3/client$ |
```

Here you can see that the server is at ipv4 address and client is given the ipv6 address for that ipv4 address and still it works.