**“Car/Bike pooling using socket programming”**

***A***

***Project Report***

*submitted in partial fulfillment of the*

*requirements for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE & ENGINEERING**

**by**

|  |  |
| --- | --- |
| **Name** | **Roll No.** |
| **Ayushman Saini** | **R164217012** |
| **Anuj Pandey** | **R164217009** |
| **Dev Pratap Singh** | **R164217014** |
|  |  |

***under the guidance of***

**Mr Vidyanand Mishra**

**SCHOOL OF COMPUTER SCIENCE**

**UNIVERSITY OF PETROLEUM & ENERGY STUDIES**

**Bidholi, Via Prem Nagar, Dehradun, UK**

**November – 2019**

**CANDIDATE’S DECLARATION**

I/We hereby certify that the project work entitled **“Car/Bike pooling using socket programming”** in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in <SPECIALIZATION) and submitted to the Department of Systemics at School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of my/ our work carried out during a period from **August** **2019** to **November** **2019** under the supervision of **Mr Vidyanand Mishra, Assistant Professor,SOCS>**.

The matter presented in this project has not been submitted by me/ us for the award of any other degree of this or any other University.

**Ayushman Saini**

**R164217012**

**Anuj Pandey**

**R164217009**

**Dev Pratap Singh**

**R164217014**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 06/11/2019

**Mrs Neelu Ahuja Mr Vidyanand Mishra**

Department of Systemics Project Guide

ACKNOWLEDGEMENT

We wish to express our deep gratitude to our guide **Mr Vidyanand Mishra**, for all advice, encouragement and constant support he has given us through out our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thank to our respected Program Head of the Department, **Mrs Neelu Ahuja**, for his great support in doing our project in **Area (like Iot, Cloud Computing etc.)** at **CIT**.

We are also grateful to **Dr. Manish Prateek (Dean SoCS)**, UPES for giving us the necessary facilities to carry out our project work successfully.

We would like to thank all our **friends** for their help and constructive criticism during our project work. Finally we have no words to express our sincere gratitude to our **parents** who have shown us this world and for every support they have given us.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Ayushman Saini** | **Anuj Pandey** | **Dev Pratap Singh** |
| **Roll No.** | **R164217012** | **R164217009** | **R164217014** |

ABSTRACT

Carpooling (in like manner vehicle sharing, ride-sharing, lift-sharing), is the sharing of vehicle ventures with the objective that more than one individual goes in a vehicle. By having more people utilizing one vehicle, carpooling reduces each individual's development costs, for instance, fuel costs, tolls, and the stress of driving. Carpooling is seen as an even more normally cordial and affordable way to deal with go as sharing journeys decreases carbon releases, traffic stop up in the city, and the necessity for parking spaces. Specialists much of the time stimulate carpooling, especially during high pollution periods and high fuel costs. Drivers and travelers offer and quest for ventures through one of the few mediums accessible. In the wake of finding a match they reach each other to organize any subtleties for the journey(s). Costs, meeting focuses and different subtleties like space for baggage are conceded to. They at that point meet and do their common vehicle journey(s) as arranged. Carpooling is usually executed for driving yet is progressively prevalent for longer erratic adventures, with the convention and normality of courses of action shifting among plans and voyages. Carpooling isn't constantly arranged for the entire length of an adventure. Particularly on long adventures, it is basic for travelers to join for parts of the voyage, and give a contribution dependent on the separation that they travel. This gives carpooling additional adaptability and enables more individuals to share journeys and set aside cash.[9]

TABLE OF CONTENTS

1. Introduction
2. Motivation
3. Problem statement
4. Literature Review
5. Objective
6. Proposed Method
7. Design Approach
   1. Flow Chart
   2. Algorithm
   3. Pseudo-code
8. Methodology
9. System Requirement
10. Output Snapshots
11. Plan To Work
12. Schedule
13. REFERENCES
14. APPENDIX I PROJECT CODE

**LIST OF FIGURES**

1. Figure 1.1: Green living through Carpooling
2. Figure 1.2:Carpooling System
3. Figure 1.3:Road Map
4. Figure 1.4: Socket Programming
5. Figure 1.5: Flow Chart
6. Figure 1.6:Pert Chart

INTRODUCTION

There are numerous individuals who are not monetarily rich or the ones who can't bear to book a vehicle of their own for an adventure or the ones who don't get themselves able at a moment to encourage the ride. What's more, there is likewise a lot of individuals who are as of now encouraging such an extravagance. There are numerous cases wherein the individual who is encouraging can help the other party for a separation in their adventure. [9]Utilizing the above actualities the assistance required for an individual can be given. Different parameters to be considered can be the arranged pair of beginning stage and goal of the impermanent taxi the same facilitator. Taking into account that the necessary requested pair is accessible, the facilitator can give the office to number of solicitations relying on the seats accessible. So as to oversee among facilitator and the mentioning customer, attachment programming can be utilized. Attachment programming will guarantee that ensure that proper, sorted and feasible requests are sent to the facilitator.

Need of customers are chosen by first come first serve.

.



Figure 1.1: Green living through Carpooling

While travelling from source to destination, one might not have required resources or ability to

travel, many a times using a paid service is not feasible economically. The other solution for a

client is to stand at the source address and wait for a facilitator(one who can road-lift the client to

their destination) for an unknown period of time, the drawback is that the waiting time is

unknown thus increasing turn-around time, and in a condition with multiple clients, some with

multiple and some with different set of ordered pair(starting point and ending point), the

knowledge about the client is unknown until their meeting to the facilitator, thus eliminating the

chance to use any sort of feasible algorithm. With socket programming and and first come first

serve priority the road-lifting can be done systematically.

Carpooling is such a need in the present age since the fuel costs are ascending and places to go in a day are multiple.

Individuals who are rich and can manage the price of a vehicle on a regular

schedule need not reconsider before spending it. Be that as it may, everyday citizens like us have

to save money on the movement endeavors. Carpooling will help on fuel costs to tumble down

and a lot more preferences.[9]

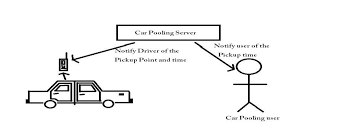


Figure 1.2: Carpooling System



Figure 1.3: Road Map

Sockets permit correspondence between two unique nodes on the equivalent or various machines. To be increasingly exact, it's a method to converse with different PCs utilizing standard Unix document descriptors. In Unix, each I/O activity is finished by composing or perusing a record descriptor. A record descriptor is only a whole number related with an open document and it very well may be a network connection, a content document, a terminal, or something different. [4]

To a developer, a socket looks and carries on a lot of like a low-level record descriptor. This is on the grounds that directions, for example, read() and compose() work with sockets similarly they do with records and pipes.

Sockets were first presented in 2.1BSD and accordingly refined into their present structure with 4.2BSD. The sockets highlight is presently accessible with most current UNIX framework discharges.[4]

Socket programming is a method for interfacing two hubs on a system to speak with one another. One socket (node) tunes in on a specific port at an IP, while other socket contacts the other to frame an association. Server frames the audience socket while customer contacts the server.

MOTIVATION

The application viable, the Car Pooling application, is an original thought which has never been executed. This turned into our wellspring of inspiration for proceeding with this undertaking. All the present car pooling strategies are

1. Time consuming.
2. Require a great deal of before-hand arranging.
3. Require a few rounds of interchanges as arrangement of messages or a progression of telephonic discussions.

An alternate source of inspiration driving the improvement of this application is that of the individuals experiencing one issue that they need to do struggle for lift so we offered a Car Pool for some relief.This truly gave us the energy to pursue the fulfillment of this venture. The initiator of the occasion could choose proposed beneficiaries from the contact list. The application at that point checks for the overall setting of the area of all the chose beneficiaries and advances the occasion to just those beneficiaries that fulfill the unique situation.

PROBLEM STATEMENT

There is intense issue of traffic on roads these days and the extending fuel costs add to the wretchedness of step by step clients of individual vehicles. Also utilization of vehicles causes pollution which has its ominous impacts. Vehicle sharing is an answer anyway issues like security and trust come into picture. Would this be able to issue be clarified? Answer for this issue is Car pooling utilizing socket programming. The Carpool system would enable its client a protected and secure way to deal with share card. This could fuse both short each day ventures, for instance, going to workplace inside the city and besides long between city trips.

LITERATURE REVIEW

Objective: The objective of the literature review is to derive a solution from the research previously done. Literature analysis will provide numerous methods to tackle the car pooling problem.

Car Pooling- The problem mentioned in the problem statement describes the necessity of the

solution. Various methods researched are as follows:

1. Estimating Scalability Issues While Finding an Optimal Assignment for Carpooling[1]. Candidate carpoolers register their personal profile and a set of periodically recurring trips. Planned periodic trips correspond to nodes in a graph; the edges are labeled with the probability for negotiation success while trying to merge planned trips by carpooling.
2. Clustering Algorithm for Urban Taxi Carpooling Vehicle Based on Data Field Energy[2]. A clustering algorithm for urban taxi carpooling based on data field energy and point spacing is proposed to solve the clustering problem of taxi carpooling on urban roads. To realize the clustering of taxis, the central point, outlier, and data points of each cluster subset are discriminated according to the threshold value determined by the product of each data point field values and point spacing. The clustering results of the proposed algorithm are better than those of the classical clustering algorithm. Various other methods such as sequential methods, cluster based methods, construction based methods are also used which are related to constraint method but they don’t connect with the real time and person.

OBJECTIVE

The problem to be noted is that the lack of mode of transportation at any time. Thus the solution

is focused on real time availability of the means. The current solution is based on the

identification by chance of the person who requires the facility with no previous knowledge of

the clients and their ordered pair of origin and destination whatsoever. Thus our solution’s

parameter is the of the client mapped with their dataset of origin and destination. The boundation

is the exactness of the pickup and drop, thus reducing the waiting time and overall turn-around

time of the process.

1. To implement a system which can communication medium between car pooler and passenger.
2. To provide real time location of car pooler to passenger and vice versa.To provide exactness of the pickup and drop, thus reducing the waiting time and overall turn-around time of the process

**PROPOSED METHOD**

For car pooling system, We require a real time system, which can process requests of passengers and drivers. We are using Socket Programming for making this system a real time system using C programming. Socket programming uses client server model. Here client and server can communicate with server using custom made protocol for a particular application. In our system server will handle the requests of clients and clients here can be a driver or passenger. According to the requests of clients drivers’ and clients’ routes are matched and their rides are fixed. Details of each client is sent to the other client.

DESIGN APPROACH

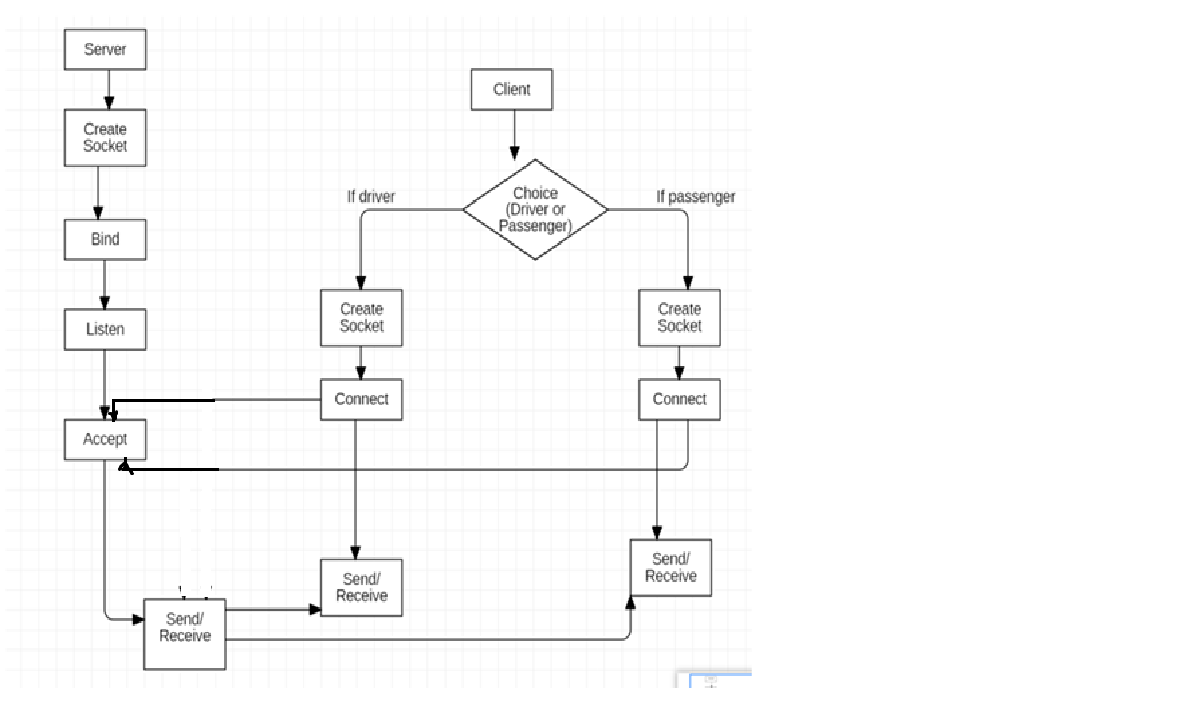
FLOW CHART

Figure 1.4: Socket programming

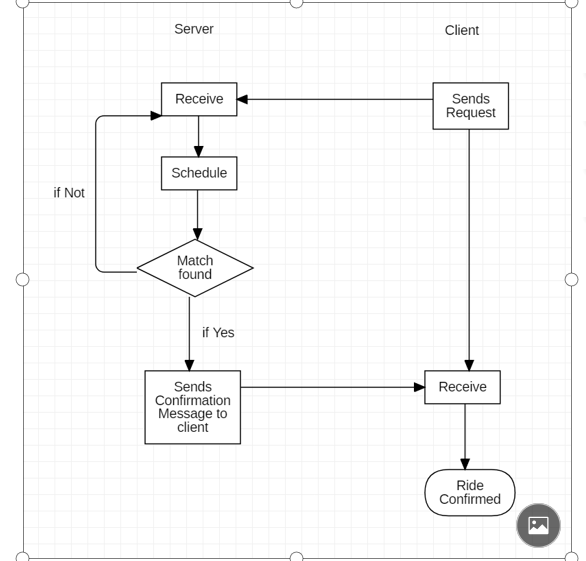


Figure 1.5: Flow chart

ALGORITHM

STEP 1: START Server

STEP 2: START Clients

STEP 3: Enter Clients(Driver and Passenger) details

While no error

CONNECTION SUCCESS

else

CONNECTION ERROR

goto: step 9

STEP 4: Enter passenger’s start point and end point

if driver’s start point < P’s start point AND driver’s end point> P’s end point

Match Found

Ride Confirmed

else

No driver is available on the same route

STEP 9: END

PSEUDO-CODE

On Client side:

* Enter 1 for Driver and 2 for Passenger
* Enter start point and end point for Driver or Passenger
* Create Socket
* Connect()
* Sends request for ride or lift

On Server Side:

* Create Socket
* Bind()
* Listen()
* Accept()
* Receives Client’s request
* If driver\_startpt<=passenger\_strartpt && driver\_endpt>=passenger\_endpt

Prints Match Found

Ride Confirms

* Else if driver\_list==NULL

Prints Not enough drivers

* Else if passenger\_list==NULL

Prints Not enough Passengers

* Sends Confirmation to Client
* Client Receives Confirmation
* Terminates

METHODOLOGY

Using Socket programming, Clients’ request can be fulfilled on real time basis. In this approach client can either be a passenger or a driver.

Each client (passenger or driver) has different functions. Passenger can request for a ride on a particular route. Driver can offer a ride to passenger in between its route.

This system aims at providing rides to passengers on real time basis. A passenger can request a ride at any time. Simultaneously Server searches for available rides for the passenger on a same route.

Drivers can offer rides at any time or at any point on map. Server monitors the position of driver and find the passengers on the same route and every point that driver reaches. So even if the driver has started his ride, even then driver can find any passenger or vice versa.

Once server finds a match, it sends the details of passenger and driver to each other, to identify them.

This system tries to solve the real world problem on procedural language(C). As the number of clients increases this system will fail because socket programming uses the ports available to perating system, which can interfere with the system.

SYSTEM REQUIREMENTS

•Hardware Interface:

1. Required a person with different type of probe vehicle for case study

2. Required 64-bit processor architecture supported by Linux.

3. Minimum Ram requirement for proper functioning is 4GB

• Software Interface:

1. This system has to be developed in C programming language.

2. Linux Operating System required.

3. It requires C compiler.

OUTPUT SNAPSHOT

PLAN TO WORK

Schedule: Starting from Aug 2019

|  |  |
| --- | --- |
| 1 st – 2 nd week  Assigned to: Ayushman,Anuj,Dev | Study Period |
| 3 rd week  Assigned to: Ayushman,Anuj,Dev | Requirement gathering |
| 4 th week  Assigned to: Ayushman,Anuj,Dev | Design |
| 5 th – 7 th week  Assigned to: Ayushman,Anuj,Dev | Pseudo code |
| 8 th – 9 th week  Assigned to: Ayushman,Anuj,Dev | Prepare Prototype |
| 10 th – 13 th week  Assigned to: Ayushman,Anuj,Dev | Coding |
| 11 th -12 th week  Assigned to: Ayushman,Anuj,Dev | Database Implementation |
| 13 th week  Assigned to: Ayushman,Anuj,Dev | Testing |
| 14 th week  Assigned to: Ayushman,Anuj,Dev, | Debugging |
| 14 th week  Assigned to: Ayushman,Anuj,Dev | Feedback |

SCHEDULE

****

Figure 1.6: Pert Chart

**References**

**(As per their appearance in the chapters)**

1) Estimating Scalability Issues While Finding an Optimal Assignment for Carpooling

https://www.sciencedirect.com/science/article/pii/S1877050913006595

2) Clustering Algorithm for Urban Taxi Carpooling Vehicle Based on Data Field Energy

https://www.hindawi.com/journals/jat/2018/3853012/

3) https://www.toolsqa.com/software-testing/waterfall-model/

4) <https://www.geeksforgeeks.org/socket-programming-cc/>

5) UNIX Network Programming, Volume 1: Interprocess Communications (The Unix Networking Reference Series , Vol 1)

6)UNIX Network Programming, Volume 2: Interprocess Communications (The Unix Networking Reference Series , Vol 2)

7)https://www.geeksforgeeks.org/program-for-priority-cpu-scheduling-set-1/

8)<https://www.sciencedirect.com/science/article/pii/S1877050913006595>

9) <https://en.wikipedia.org/wiki/Carpool>

APPENDIX I PROJECT CODE

**Server.c**

#include <stdio.h>

#include <string.h> //strlen

#include <stdlib.h>

#include <errno.h>

#include <unistd.h> //close

#include <arpa/inet.h> //close

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <sys/time.h> //FD\_SET, FD\_ISSET, FD\_ZERO macros

#define TRUE 1

#define FALSE 0

#define PORT 8888

struct driver

{

struct driver \*prev;

char d\_name[20], type[10];

int d\_id,dstartpt,dendpt,capacity;

int d\_socket;

struct driver \*next;

}\*h,\*temp,\*temp1,\*temp2,\*temp3,\*temp4;

struct passenger

{

struct passenger \*prev;

char p\_name[20];

int p\_id,pstartpt,pendpt;

int p\_socket;

struct passenger \*next;

}\*p,\*p\_temp,\*p\_temp1,\*p\_temp2,\*p\_temp3,\*p\_temp4;

int d\_count = 0;

int p\_count = 0;

/\* TO d\_create an empty driver \*/

void d\_create(char name[20], char type[10], int id, int startpt, int endpt, int capacity,int s)

{

temp =(struct driver \*)malloc(1\*sizeof(struct driver));

temp->prev = NULL;

temp->next = NULL;

strcpy(temp->d\_name,name);

temp->d\_id=id;

strcpy(temp->type,type);

temp->dstartpt=startpt;

temp->dendpt=endpt;

temp->capacity=capacity;

temp->d\_socket=s;

d\_count++;

}

/\* TO p\_create an empty passenger \*/

void p\_create(char name[20],int id,int startpt, int endpt, int s)

{

p\_temp =(struct passenger \*)malloc(1\*sizeof(struct passenger));

p\_temp->prev = NULL;

p\_temp->next = NULL;

strcpy(p\_temp->p\_name,name);

p\_temp->p\_id=id;

p\_temp->pstartpt=startpt;

p\_temp->pendpt=endpt;

p\_temp->p\_socket=s;

p\_count++;

}

/\* TO insert at end (Driver) \*/

void d\_insert(char name[20], char type[10], int id, int startpt, int endpt, int capacity,int s)

{

if (h == NULL)

{

d\_create(name,type,id,startpt,endpt,capacity,s);

h= temp;

temp1 = h;

}

else

{

d\_create(name,type,id,startpt,endpt,capacity,s);

temp1->next = temp;

temp->prev = temp1;

temp1 = temp;

}

}

/\* TO insert at end (Passenger)\*/

void p\_insert(char name[20],int id,int startpt, int endpt, int s)

{

if (p == NULL)

{

p\_create(name,id,startpt,endpt,s);

p = p\_temp;

p\_temp1 = p;

}

else

{

p\_create(name,id,startpt,endpt,s);

p\_temp1->next = p\_temp;

p\_temp->prev = p\_temp1;

p\_temp1 = p\_temp;

}

}

//For printing driver's list

void d\_print()

{

temp2 = h;

if (temp2 == NULL)

{

printf("List empty to display \n");

return;

}

printf("\n Linked list elements from begining : ");

while (temp2->next != NULL)

{

printf("\nName of driver: %s ", temp2->d\_name);

printf("\nID of driver: %d ", temp2->d\_id);

printf("\nType of vehicle: %s ", temp2->type);

printf("\nStarting point: %d ", temp2->dstartpt);

printf("\nEnd point: %d ", temp2->dendpt);

printf("\nCapacity: %d ", temp2->capacity);

printf("\n......................");

temp2 = temp2->next;

}

printf("\nName of driver: %s ", temp2->d\_name);

printf("\nID of driver: %d ", temp2->d\_id);

printf("\nType of vehicle: %s ", temp2->type);

printf("\nStarting point: %d ", temp2->dstartpt);

printf("\nEnd point: %d ", temp2->dendpt);

printf("\n......................");

}

// for printing passenger's list

void p\_print()

{

p\_temp2 = p;

if (p\_temp2 == NULL)

{

printf("\nList empty to display \n");

return;

}

printf("\n Linked list elements from begining : ");

while (p\_temp2->next != NULL)

{

printf("\nName of Passenger: %s ", p\_temp2->p\_name);

printf("\nID of Passenger: %d ", p\_temp2->p\_id);

printf("\nStarting point: %d ", p\_temp2->pstartpt);

printf("\nEnd point: %d ", p\_temp2->pendpt);

printf("\n......................");

p\_temp2 = p\_temp2->next;

}

printf("\nName of Passenger: %s ", p\_temp2->p\_name);

printf("\nID of Passenger: %d ", p\_temp2->p\_id);

printf("\nStarting point: %d ", p\_temp2->pstartpt);

printf("\nEnd point: %d ", p\_temp2->pendpt);

printf("\n......................");

}

void d\_delete()

{

if (temp1 == NULL)

{

printf("\nList empty to delete \n");

}

else{

temp3=temp1;

temp1=temp1->prev;

temp1->next=NULL;

free(temp3);

}

}

void p\_delete()

{

if (p\_temp1 == NULL)

{

printf("\nList empty to delete \n");

}

else{

p\_temp3=p\_temp1;

p\_temp1=p\_temp1->prev;

p\_temp1->next=NULL;

free(p\_temp3);

}

}

// For matching/ scheduling driver and passenger

int schedule( int dp\_arr[],int ch )

{

printf("Inside schedule func\n");

temp4=h;

p\_temp4=p;

printf("drivers list\n");

d\_print();

printf("passengers list\n");

p\_print();

if (ch==2){

if (temp4 == NULL)

{

return -2;

}

}

else if(ch==1){

if (p\_temp4 == NULL)

{

return -1;

}

}

while(1){

if((temp4->dstartpt<=p\_temp4->pstartpt) && (temp4->dendpt>=p\_temp4->pendpt))

{

dp\_arr[0]=temp4->d\_socket;

printf("\nDriver socket %d ",dp\_arr[0]);

dp\_arr[1]=p\_temp4->p\_socket;

printf("\nPassenger socket %d ",dp\_arr[1]);

return 0;

}

p\_temp4 = p\_temp4->next;

temp4 = temp4->next;

if((temp4->next==NULL)||(p\_temp4->next==NULL)){

if (ch==2){

return -2;

}

else if(ch==1){

return -1;

}

}

}

}

int main()

{

printf("\*\*\*\*\*\*\*\*\*Server\*\*\*\*\*\*\*\*\*\*\*\n");

char name[20], type[10];

int ch,id,startpt,endpt,capacity;

int dp\_arr[2];

int x;

int opt = TRUE;

int master\_socket , addrlen , new\_socket , client\_socket[30] , max\_clients = 30 , activity, i , valread , sd;

int max\_sd;

struct sockaddr\_in address;

char buffer[100];

//set of socket descriptors

fd\_set readfds;

//a message

char \*message = "ECHO Daemon v1.0 \r\n";

//initialise all client\_socket[] to 0 so not checked

for (i = 0; i < max\_clients; i++)

{

client\_socket[i] = 0;

}

//create a master socket

if( (master\_socket = socket(AF\_INET , SOCK\_STREAM , 0)) == 0)

{

perror("socket failed");

exit(EXIT\_FAILURE);

}

//set master socket to allow multiple connections ,

//this is just a good habit, it will work without this

if( setsockopt(master\_socket, SOL\_SOCKET, SO\_REUSEADDR, (char \*)&opt, sizeof(opt)) < 0 )

{

perror("setsockopt");

exit(EXIT\_FAILURE);

}

//type of socket created

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons( PORT );

//bind the socket to localhost port 8888

if (bind(master\_socket, (struct sockaddr \*)&address, sizeof(address))<0)

{

perror("bind failed");

exit(EXIT\_FAILURE);

}

printf("Listener on port %d \n", PORT);

//try to specify maximum of 3 pending connections for the master socket

if (listen(master\_socket, 3) < 0)

{

perror("listen");

exit(EXIT\_FAILURE);

}

//accept the incoming connection

addrlen = sizeof(address);

puts("Waiting for connections ...");

while(TRUE)

{

//clear the socket set

FD\_ZERO(&readfds);

//add master socket to set

FD\_SET(master\_socket, &readfds);

max\_sd = master\_socket;

//add child sockets to set

for ( i = 0 ; i < max\_clients ; i++)

{

//socket descriptor

sd = client\_socket[i];

//if valid socket descriptor then add to read list

if(sd > 0)

FD\_SET( sd , &readfds);

//highest file descriptor number, need it for the select function

if(sd > max\_sd)

max\_sd = sd;

}

//wait for an activity on one of the sockets , timeout is NULL ,

//so wait indefinitely

activity = select( max\_sd + 1 , &readfds , NULL , NULL , NULL);

if ((activity < 0) && (errno!=EINTR))

{

printf("select error");

}

//If something happened on the master socket ,

//then its an incoming connection

if (FD\_ISSET(master\_socket, &readfds))

{

if ((new\_socket = accept(master\_socket,

(struct sockaddr \*)&address, (socklen\_t\*)&addrlen))<0)

{

perror("accept");

exit(EXIT\_FAILURE);

}

//inform user of socket number - used in send and receive commands

printf("New connection , socket fd is %d , ip is : %s , port : %d \n" , new\_socket , inet\_ntoa(address.sin\_addr) , ntohs (address.sin\_port));

printf("\nReceiving Request");

if(recv(new\_socket, name, 20,0)<0)

printf("[-]Error in receiving data.\n");

//printf("\n%s",name);

recv(new\_socket, &id, sizeof(int),0);

recv(new\_socket, &ch, sizeof(int),0);

if(ch==1){

//printf("\n Driver");

recv(new\_socket, type, 10,0);

recv(new\_socket, &capacity, sizeof(int),0);

}

//printf("\n Driver1");

recv(new\_socket, &startpt, sizeof(int),0);

recv(new\_socket, &endpt, sizeof(int),0);

if(ch==1){

d\_insert(name,type,id,startpt,endpt,capacity,new\_socket);

d\_print();

}

else if(ch==2){

p\_insert(name,id,startpt,endpt,new\_socket);

p\_print();

}

printf("\nConnection Accepted from Client %d",id);

x=schedule(dp\_arr,ch);

if (x==-1){

bzero(buffer,100);

strcpy(buffer," There are no passengers yet !!");

send(new\_socket, buffer, 100,0);

d\_delete();

}

else if(x==-2){

bzero(buffer,100);

strcpy(buffer," There are no drivers yet !!");

send(new\_socket, buffer, 100,0);

p\_delete();

}

else if(x==0){

bzero(buffer,100);

strcpy(buffer,"Match found !!");

send(dp\_arr[0], buffer, 100,0);

bzero(buffer,100);

strcpy(buffer,"Match found !!");

send(dp\_arr[1], buffer, 100,0);

temp2=h;

p\_temp2=p;

while (temp2->next != NULL){

if(dp\_arr[0]==temp2->d\_socket){

send(dp\_arr[0], temp2->d\_name, 20,0);

send(dp\_arr[0], &temp2->d\_id,sizeof(int),0);

}

temp2 = temp2->next;

}

while(p\_temp2->next!=NULL){

if(dp\_arr[1]==p\_temp2->p\_socket)

{

send(dp\_arr[1], p\_temp2->p\_name, 20,0);

send(dp\_arr[1], &p\_temp2->p\_id,sizeof(int),0); }

p\_temp2 = p\_temp2->next;

}

d\_delete();

p\_delete();

}

//add new socket to array of sockets

for (i = 0; i < max\_clients; i++)

{

//if position is empty

if( client\_socket[i] == 0 )

{

client\_socket[i] = new\_socket;

printf("Adding to list of sockets as %d\n" , i);

break;

}

}

}

close(new\_socket);

//else its some IO operation on some other socket

for (i = 0; i < max\_clients; i++)

{

sd = client\_socket[i];

if (FD\_ISSET( sd , &readfds))

{

//Check if it was for closing , and also read the

//incoming message

if ((valread = read( sd , buffer, 100)) == 0)

{

//Somebody disconnected , get his details and print

getpeername(sd , (struct sockaddr\*)&address , (socklen\_t\*)&addrlen);

printf("Host disconnected , ip %s , port %d \n" , inet\_ntoa(address.sin\_addr) , ntohs(address.sin\_port));

//Close the socket and mark as 0 in list for reuse

close( sd );

client\_socket[i] = 0;

}

//Echo back the message that came in

/\*else

{

//set the string terminating NULL byte on the end

//of the data read

buffer[valread] = '\0';

send(sd , buffer , strlen(buffer) , 0 );

} \*/} } }

return 0;

}

**Client.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define PORT 8888

void send\_request(char name[40], int id, int ch, char type[10], int capacity, int startpt, int endpt , int clientSocket)

{

printf("\nSending Request");

if(send(clientSocket, name,20,0)<0)

printf("[-]Error in sending data.\n");

send(clientSocket, &id, sizeof(int),0);

send(clientSocket, &ch, sizeof(int),0);

send(clientSocket, type,10,0);

send(clientSocket, &capacity, sizeof(int),0);

send(clientSocket, &startpt, sizeof(int),0);

send(clientSocket, &endpt, sizeof(int),0);

printf("\nRequest Sent");

}

void send\_request1(char name[40], int id, int ch, int startpt, int endpt , int clientSocket)

{

printf("\nSending Request");

if(send(clientSocket, name,20,0)<0)

printf("[-]Error in sending data.\n");

send(clientSocket, &id, sizeof(int),0);

send(clientSocket, &ch, sizeof(int),0);

send(clientSocket, &startpt, sizeof(int),0);

send(clientSocket, &endpt, sizeof(int),0);

printf("\nRequest Sent");

}

int main(){

int clientSocket, ret;

struct sockaddr\_in serverAddr;

char buffer[100];

int ch,id,startpt,endpt;

int capacity=0;

char name[20];

char type[10]="nill";

char t\_name[20];

int t\_id;

printf(" Welcome to Carpooling System");

printf("\n[1]-Driver [2]-Passenger");

printf("\nEnter your choice: ");

scanf("%d",&ch);

printf("\n Enter your name : ");

scanf("%s", &name[0]);

printf("\n Enter ID: ");

scanf("%d", &id);

if(ch==1){

printf("\nEnter Type of vehicle: ");

scanf("%s",&type[0]);

printf("\nEnter vehicle capacity: ");

scanf("%d",&capacity);

printf("\n[1]-Ballupur [2]-FRI [3]-IMA [4]-Prem Nagar [5]-Nanda Ki Chowki [6]-Pondha [7]-Kandoli [8]-Bhidoli");

printf("\n Enter Sarting point : ");

scanf("%d", &startpt);

printf("\n Enter End point : ");

scanf("%d", &endpt);

printf("\nRequest is processing");

clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if(clientSocket < 0){

printf("[-]Error in connection.\n");

exit(1);

}

printf("[+]Client Socket is created.\n");

memset(&serverAddr, '\0', sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(PORT);

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

ret = connect(clientSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

if(ret < 0){

printf("[-]Error in connection.\n");

exit(1);

}

printf("[+]Connected to Server.\n");

send\_request(name,id,ch,type,capacity,startpt,endpt, clientSocket);

while(1)

{

bzero(buffer,100);

recv(clientSocket, buffer, 100,0);

printf("\n%s",buffer);

if(strcmp(buffer," There are no passengers yet !!") == 0){

printf("\n Start your ride without passenger. Will check for passenger on next point");

while (startpt!=endpt){

sleep(10);

startpt= startpt+1;

send\_request(name,id,ch,type,capacity,startpt,endpt,clientSocket);

bzero(buffer,100);

recv(clientSocket, buffer, 100,0);

if(strcmp(buffer,"Match found !!") == 0)

recv(clientSocket, t\_name, 20,0);

recv(clientSocket, &t\_id, sizeof(int),0);

if(ch==1)

printf("\nDriver's name and id are %s and %d",t\_name,t\_id);

else if(ch==2)

printf("\nPassenger's name and id are %s and %d",t\_name,t\_id);

close(clientSocket);

printf("[-]Disconnected from server.\n");

exit(1);

}

printf("\nNot found any passengers in ride. \nRide Complete\n");

close(clientSocket);

printf("[-]Disconnected from server.\n");

exit(1);

}

else if(strcmp(buffer,"Match found !!") == 0){

recv(clientSocket, t\_name, 20,0);

recv(clientSocket, &t\_id, sizeof(int),0);

if(ch==1)

printf("\nDriver's name and id are %s and %d",t\_name,t\_id);

else if(ch==2)

printf("\nPassenger's name and id are %s and %d",t\_name,t\_id);

close(clientSocket);

printf("[-]Disconnected from server.\n");

exit(1);

}}}

else if(ch==2)

{

pintf("\n[1]-Ballupur [2]-FRI [3]-IMA [4]-Prem Nagar [5]-Nanda Ki Chowki [6]-Pondha [7]-Kandoli [8]-Bhidoli");

printf("\n Enter Sarting point : ");

scanf("%d", &startpt);

printf("\n Enter End point : ");

scanf("%d", &endpt);

printf("\nRequest is processing");

clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if(clientSocket < 0){

printf("[-]Error in connection.\n");

exit(1);

}

printf("[+]Client Socket is created.\n");

memset(&serverAddr, '\0', sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(PORT);

serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

ret = connect(clientSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

if(ret < 0){

printf("[-]Error in connection.\n");

exit(1);

}

printf("[+]Connected to Server.\n");

send\_request1(name,id,ch,startpt,endpt, clientSocket);

bzero(buffer,100);

recv(clientSocket, buffer, 100,0);

printf("\n%s",buffer);

if(strcmp(buffer," There are no drivers yet !!") == 0){

bzero(buffer,100);

recv(clientSocket, buffer, 100,0);

}

if(strcmp(buffer,"Match found !!") == 0){

recv(clientSocket, t\_name, 20,0);

recv(clientSocket, &t\_id, sizeof(int),0);

if(ch==1)

printf("\nDriver's name and id are %s and %d",t\_name,t\_id);

else if(ch==2)

printf("\nPassenger's name and id are %s and %d",t\_name,t\_id);

close(clientSocket);

printf("[-]Disconnected from server.\n");

exit(1);

}

}

close(clientSocket);

return 0;

}