"AUTOMATIC PARKING SYSTEM USING SEMAPHORE"

PROJECT REPORT

Submitted for the course: Operating Systems (CSE2005)

By

Sahil Jain	**16BCE0372**
Siddharth Rajesh Goradia	**17BCE0287**
Bhavya Kansal	**16BCE0453**

Slot: F2

Name of faculty: VIJAYA KUMAR K

(SCHOOL OF COMPUTER SCIENCE AND ENGINEERING)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

October, 2018

CERTIFICATE

This is to certify that the project work entitled **“Automatic Parking System Using Semaphore”** that is being submitted by “Sahil Jain and Team” for Operating Systems (CSE2005) is a record of bonafide work done under my supervision. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted for any other CAL course.

Place: Vellore

Date:

Signature of Student:

Signature of Faculty:

<u>ACKNOWLEDGEMENTS</u>

I take immense pleasure in thanking **Dr. G. Viswanathan**, my beloved Chancellor, VIT University and respected Dean, **School of Computer Science and Engineering**, for having permitted me to carry out the project.

I express gratitude to my guide, **Prof. VIJAYA KUMR K**, for guidance and suggestions that helped me to complete the project on time. Words are inadequate to express my gratitude to the faculty and staff members who encouraged and supported me during the project. Finally, I would like to thank my ever-loving parents for their blessings and my friends for their timely help and support.

Signature of Student

Table of Contents

List of Figures

Abstract

With the increasing number of cars and traffic, it has become a major problem to find place for parking cars in public places such as market places or malls or any famous place. To deal with this problem we have come up with an idea for automatic parking system. This is an efficient way of parking cars using semaphore and first come first service for four-wheeler and two-wheeler separately. : With this project we will have an efficient way of parking the cars and this will help to reduce the car parking traffic on streets. This is a much cheaper and convenient way of parking.

1. INTRODUCTION

We are incorporating two main concepts of operating systems in our project which are:

1. **Semaphore:** A semaphore, in its most basic form, is a protected integer variable that can facilitate and restrict access to shared resources in a multi-processing environment. The two most common kinds of semaphores are counting semaphores and binary semaphores. Counting semaphores represent multiple resources, while binary semaphores, as the name implies, represents two possible states (generally 0 or 1; locked or unlocked).

wait() was called (meaning *to decrement*) and signal() was called V (meaning *to increment*).

In our project we will use semaphore to ensure that only one vehicle gets to be parked at one time and all other have to wait until their turns come. As getting two vehichle (process) in the automatic machine will cause a deadlock that's why to avoid deadlock we are using semaphore.

2. **FCFS Synchronization:** To avoid deadlock other process have to wait until the vehicle in the automatic machine gets parked. So, to synchronize the order and avoid any confusion we are using First Come First Serve Synchronization which will ensure that the vehicles align in the queue according to the order they came.

3.

2. Literature Survey

Semaphore is a visual method of communication that involves signaling the alphabet or numbers by the hand holding of 2 flags in specific positions. It has been described as ’**Optical telegraph**’. The flags are colored differently, depending on whether the signal is sent over the land or across the sea. Red and yellow flags (the Oscar flags) are used at sea and are similar to our Surf Lifesaving flags.

The system was developed in France, in 1790 by Claude Chappe and his brothers. This was the time of the French Revolution and there was a great need for the government to be able to quickly communicate orders and to receive information. Their first message, on March 2, 1791 was sent a distance of 10 miles and read: ”If you succeed, you will soon bask in glory”. They used black and white flags initially, as well as clocks, codebooks and telescopes.

Many automatic parking systems have been designed in the present scenario. Now we are going to take a look the work conducted by Tuan Le-Anh and M.B.M. De Koster. They have come up with an automated vehicle guiding system which can simplify the task of parking. Initially they have fixed a parking area, and they decide where the next vehicle should be parked using scheduling and semaphore.

The parking allotment is done on first come first served basis. They have used the vehicle position system to determine the free slots in the parking area. Vehicle locations have to loaded as soon as possible to get quick responses for the new vehicles approaching the parking which has also been handled by their static vehicle positioning strategy. This system can also be handled using semaphore which we have used.

3. METHODOLOGY: EXPERIMENTAL/SIMULATION

Our project is based on the concept of semaphores and scheduling. The vehicles enter the parking system in two queues. Only one vehicle can enter the parking system/machine at a time.

P and V Semaphores are used to avoid deadlock whenever a vehicle enter into machine function P is called which decrements the value of S (constant) to 0 and until S equal’s one no other vehicle can enter the machine and when vehicle get parked it calls function V which increments S to 1 so now other vehicle can also enter the machine.

We have done this for two-wheelers and four-wheeler’s separately. Then the vehicles reach the automatic parking machines and are left in the machine to start the process of automatic parking. The machine automatically parks the vehicles. This can be explained as : The parking machine is a critical section and the vehicles are processes.

The parking entrance of 2 rows acts as a lock Only one process can enter the critical section/shared memory at a time in each row one four-wheeler and one two-wheeler.

The processes(here vehicles) are then parked(scheduled) on the basis of first come first serve and semaphore. The processes the leave the shared memory and next vehicle can enter the parking area.

Logical Diagram

For Entrance

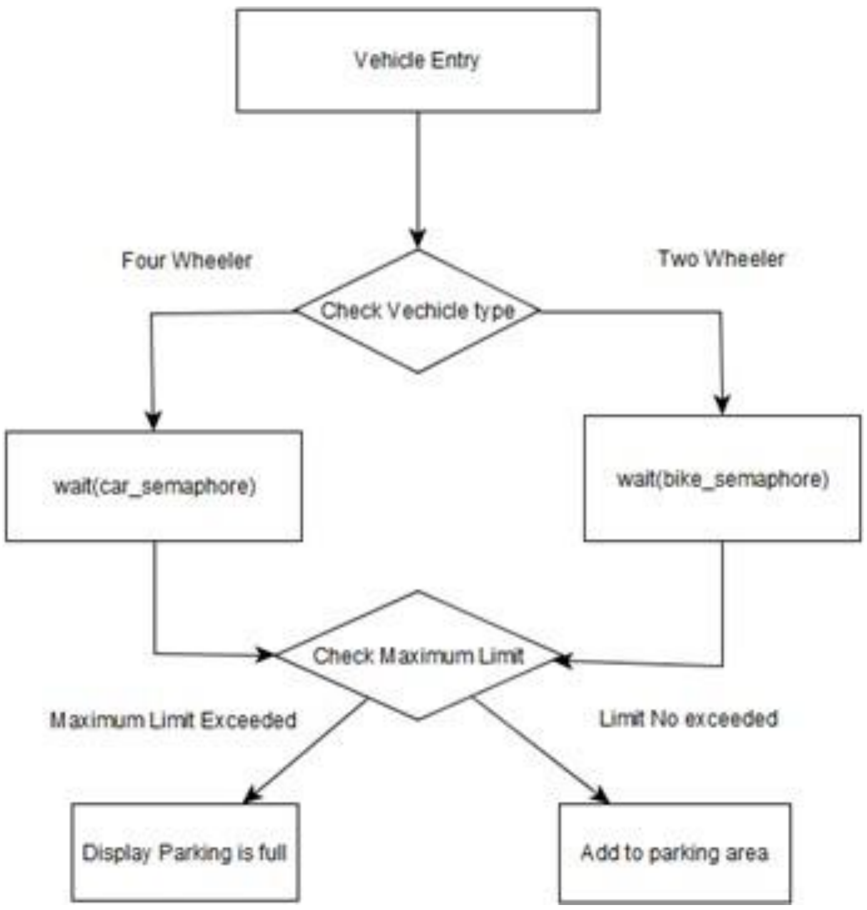


Figure 3.1: Block Diagram for Vehicles Entrance

For Exit

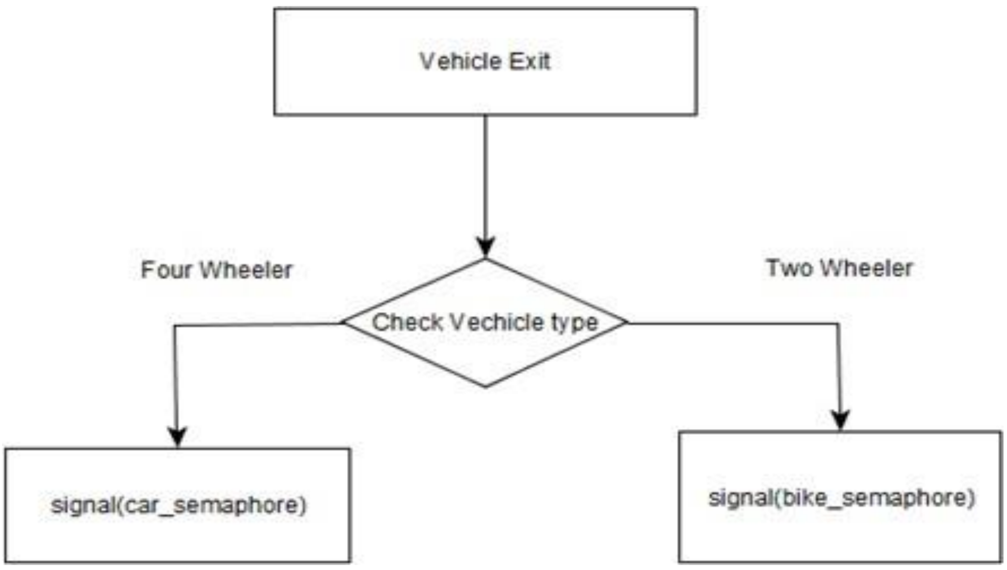


Figure 3.2: Block Diagram for Vehicles Exit

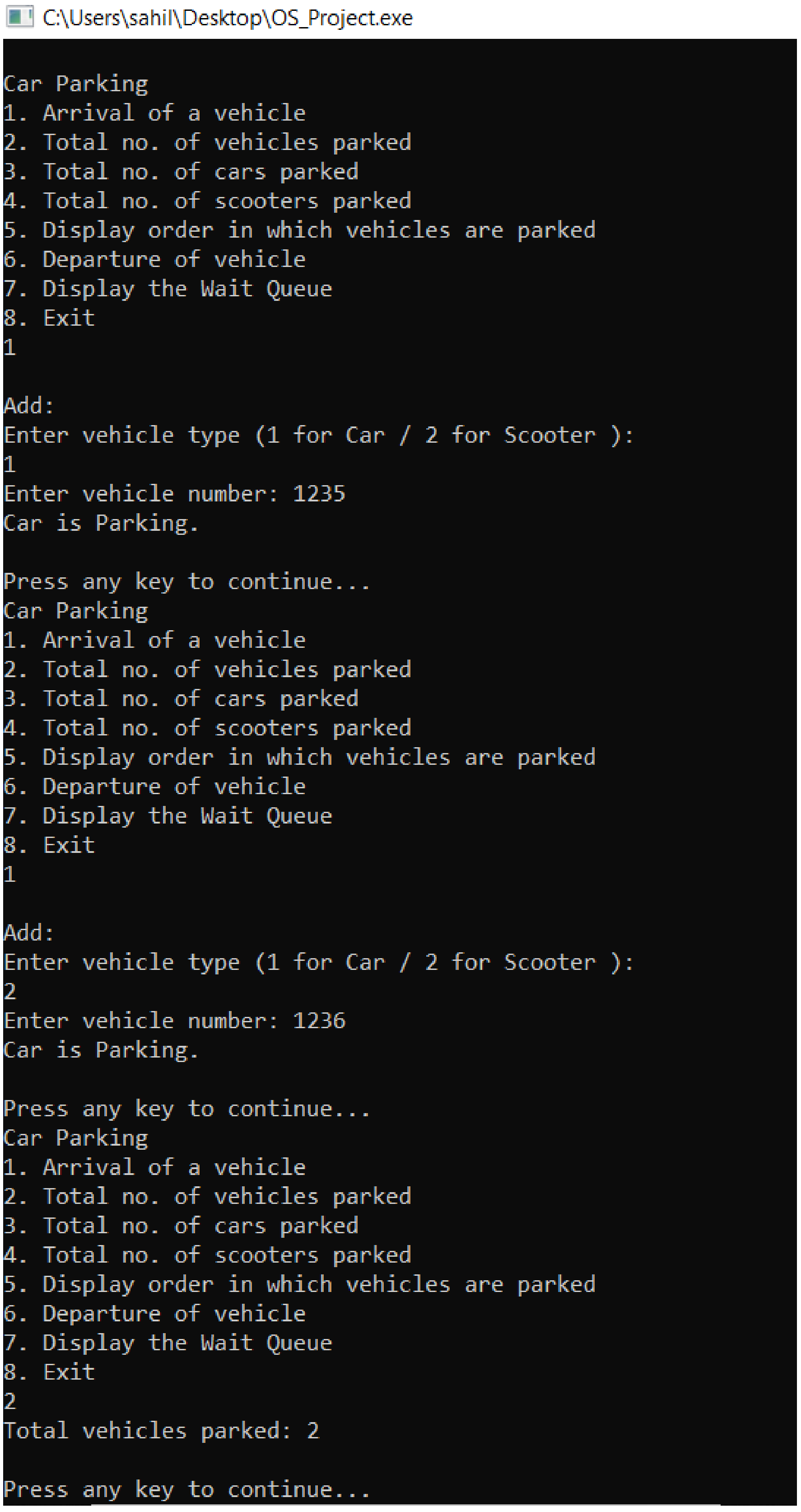
4. RESULTS AND DISCUSSION

It has become a major problem to find place for parking cars in public places such as market places or malls or any famous place. Based on the concept of semaphore and scheduling we have solved this problem.

Using P and V Semaphores, P semaphore will decrease the value of S and V will increase the value of S. Every vehicle will be given priority on the basis of first come first service.

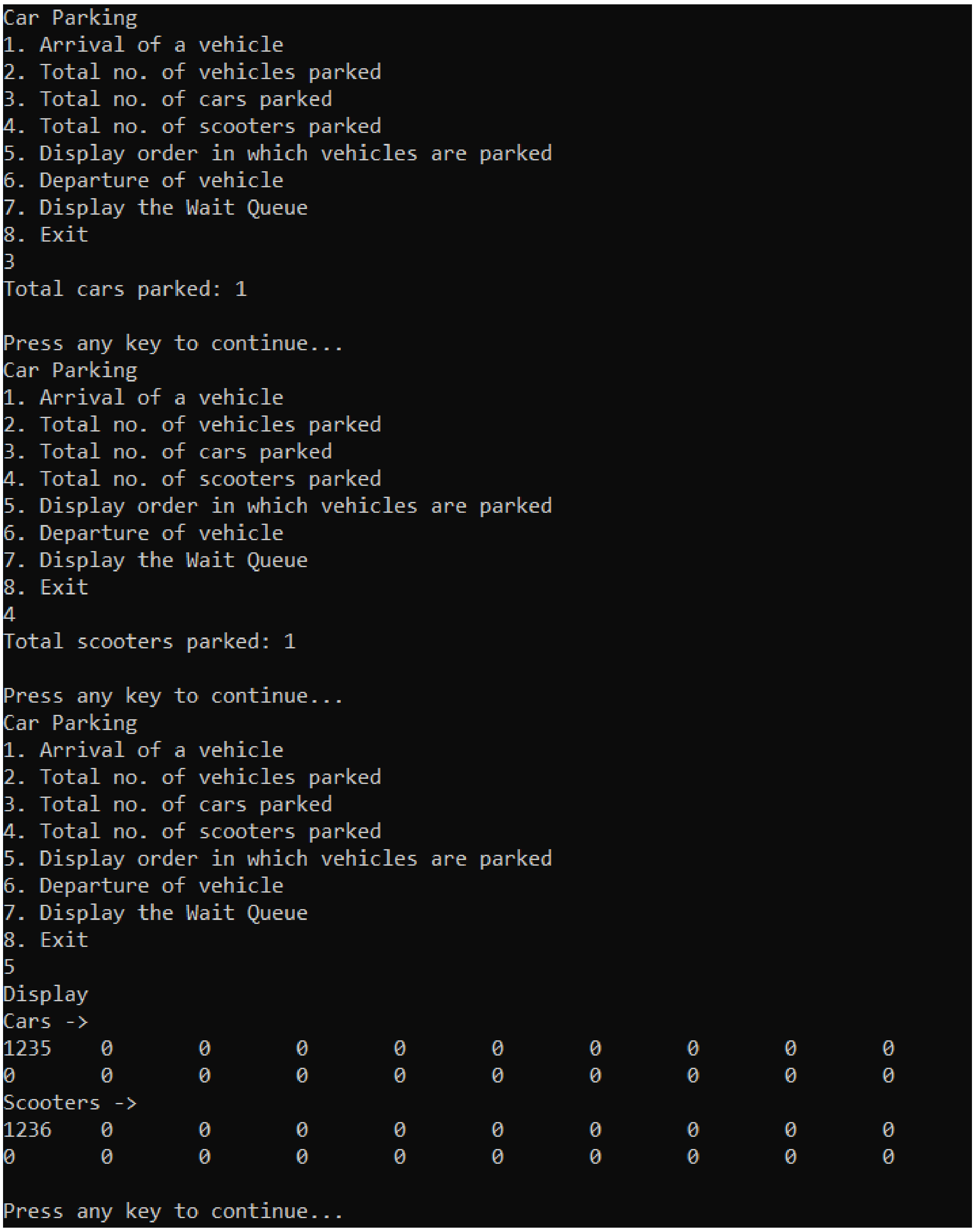
This way the traffic problem will be solved everybody will get place for parking vehicle both two-wheeler and four-wheeler at proper places according to their sizes.

Snapshot of OUTPUT


C:\Users\sahil\Desktop\OS_Project.exe

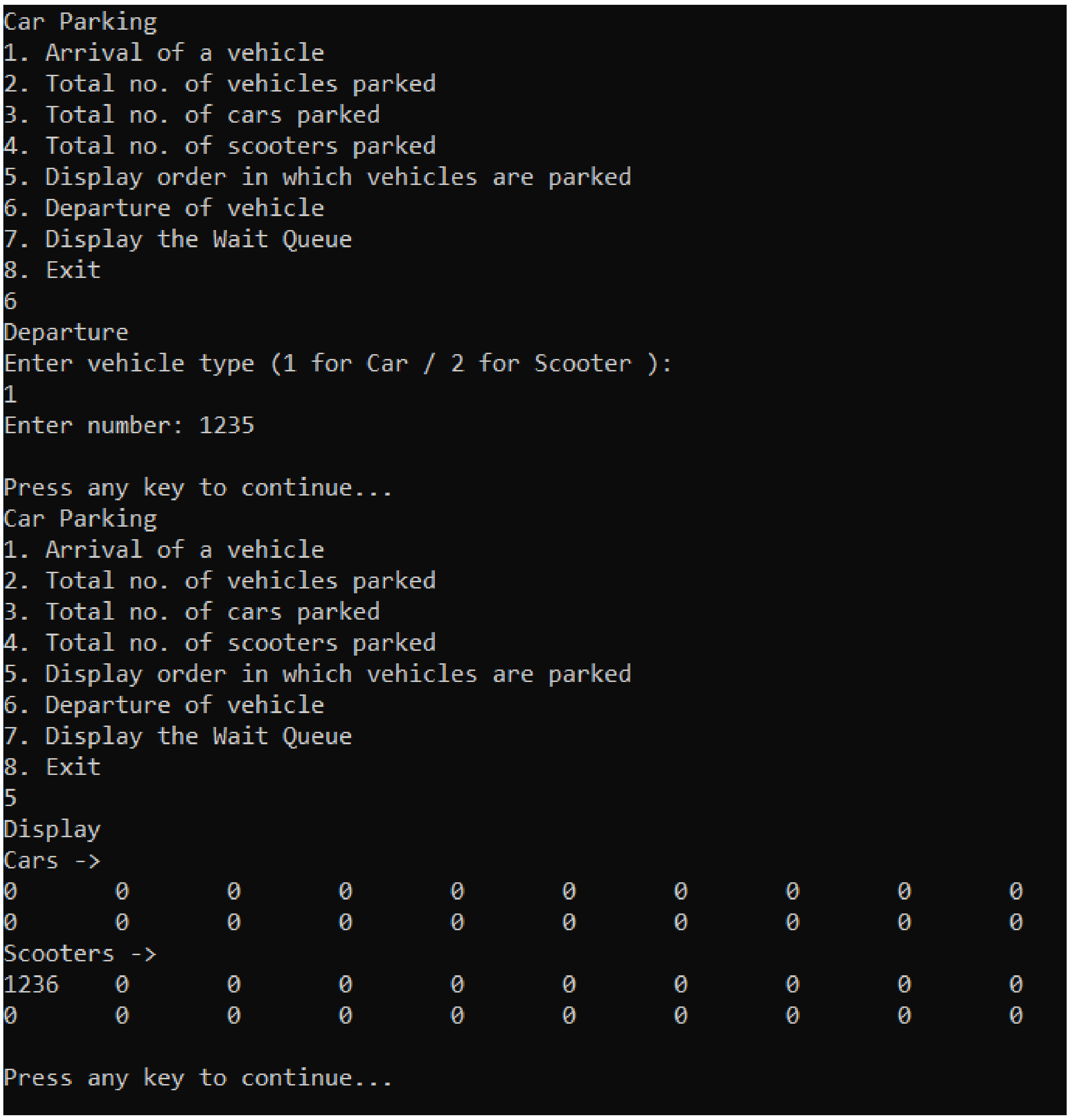
```
Car Parking
1. Arrival of a vehicle
2. Total no. of vehicles parked
3. Total no. of cars parked
4. Total no. of scooters parked
5. Display order in which vehicles are parked
6. Departure of vehicle
7. Display the Wait Queue
8. Exit
1

Add:
Enter vehicle type (1 for Car / 2 for Scooter ):
1
Enter vehicle number: 1235
Car is Parking.

Press any key to continue...
Car Parking
1. Arrival of a vehicle
2. Total no. of vehicles parked
3. Total no. of cars parked
4. Total no. of scooters parked
5. Display order in which vehicles are parked
6. Departure of vehicle
7. Display the Wait Queue
8. Exit
1

Add:
Enter vehicle type (1 for Car / 2 for Scooter ):
2
Enter vehicle number: 1236
Car is Parking.

Press any key to continue...
Car Parking
1. Arrival of a vehicle
2. Total no. of vehicles parked
3. Total no. of cars parked
4. Total no. of scooters parked
5. Display order in which vehicles are parked
6. Departure of vehicle
7. Display the Wait Queue
8. Exit
2
Total vehicles parked: 2

Press any key to continue...
```

Figure 4.1: Screenshot of Adding Vehicles



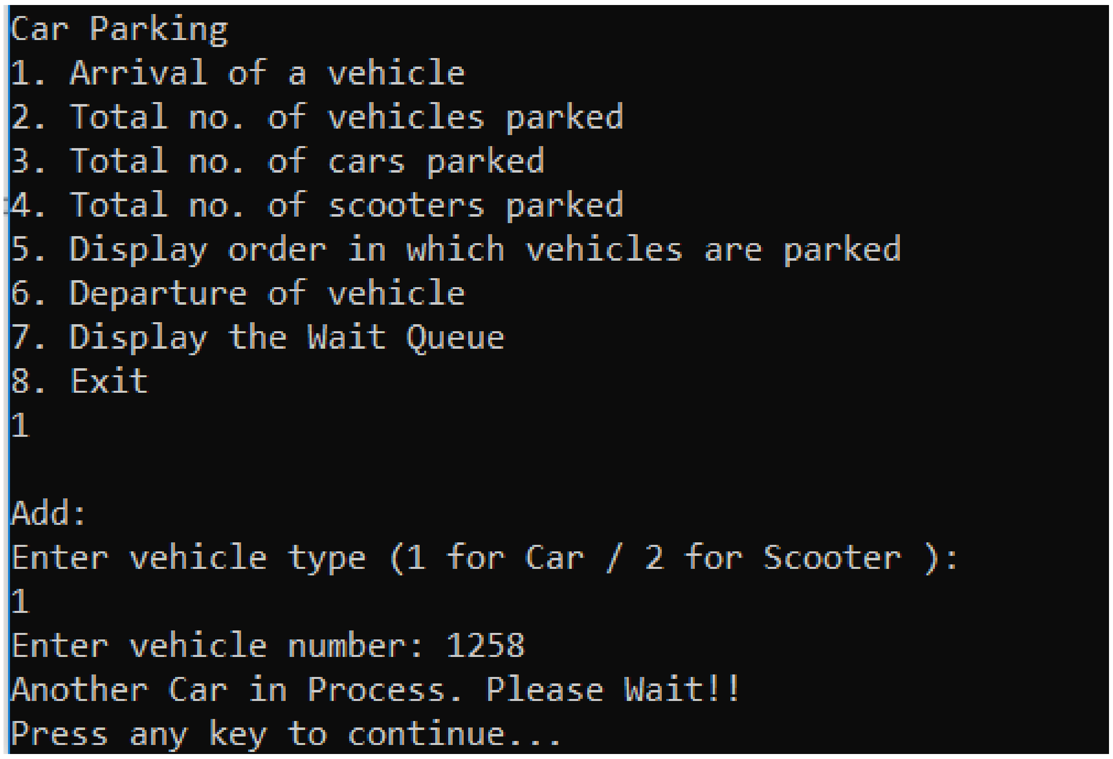
```
Car Parking
1. Arrival of a vehicle
2. Total no. of vehicles parked
3. Total no. of cars parked
4. Total no. of scooters parked
5. Display order in which vehicles are parked
6. Departure of vehicle
7. Display the Wait Queue
8. Exit
3
Total cars parked: 1

Press any key to continue...
Car Parking
1. Arrival of a vehicle
2. Total no. of vehicles parked
3. Total no. of cars parked
4. Total no. of scooters parked
5. Display order in which vehicles are parked
6. Departure of vehicle
7. Display the Wait Queue
8. Exit
4
Total scooters parked: 1

Press any key to continue...
Car Parking
1. Arrival of a vehicle
2. Total no. of vehicles parked
3. Total no. of cars parked
4. Total no. of scooters parked
5. Display order in which vehicles are parked
6. Departure of vehicle
7. Display the Wait Queue
8. Exit
5
Display
Cars ->
1235    0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0
Scooters ->
1236    0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0

Press any key to continue...
```

Figure 4.2: Screenshot of Order in which vehicles are Parked



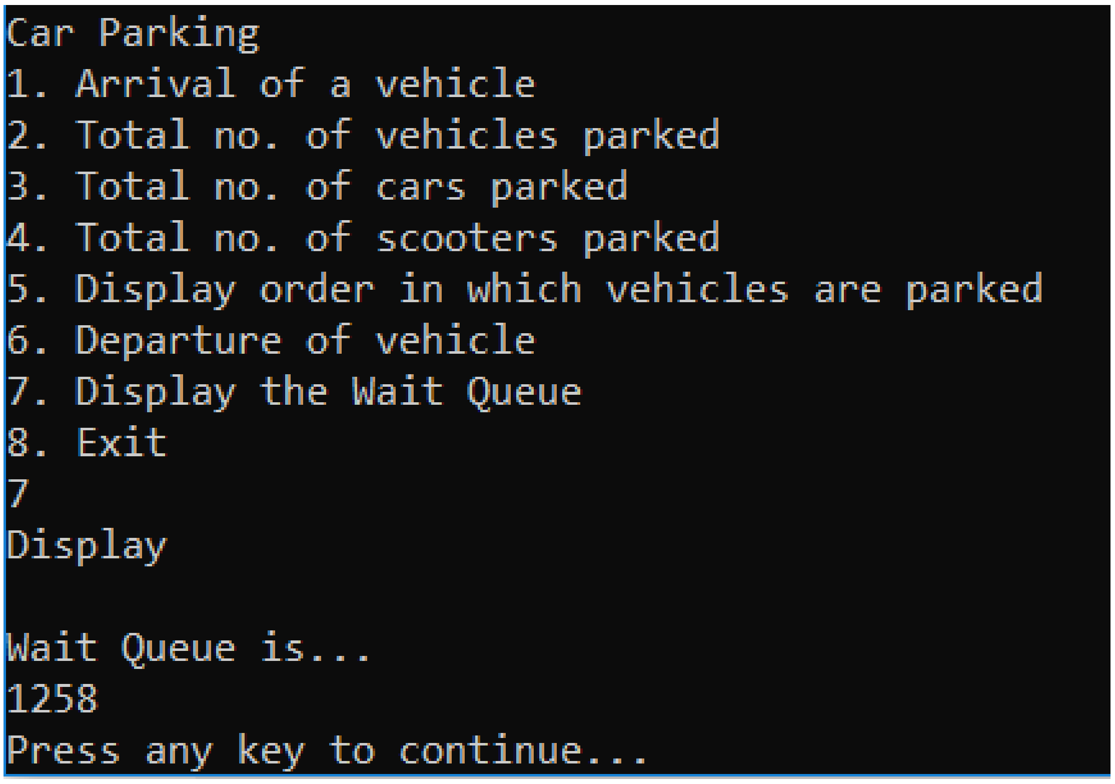
```
Car Parking
1. Arrival of a vehicle
2. Total no. of vehicles parked
3. Total no. of cars parked
4. Total no. of scooters parked
5. Display order in which vehicles are parked
6. Departure of vehicle
7. Display the Wait Queue
8. Exit
6
Departure
Enter vehicle type (1 for Car / 2 for Scooter ):
1
Enter number: 1235

Press any key to continue...
Car Parking
1. Arrival of a vehicle
2. Total no. of vehicles parked
3. Total no. of cars parked
4. Total no. of scooters parked
5. Display order in which vehicles are parked
6. Departure of vehicle
7. Display the Wait Queue
8. Exit
5
Display
Cars ->
0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0
Scooters ->
1236    0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0

Press any key to continue...
```

Figure 4.3: Screenshot of Departure of Vehicle



```
Car Parking
1. Arrival of a vehicle
2. Total no. of vehicles parked
3. Total no. of cars parked
4. Total no. of scooters parked
5. Display order in which vehicles are parked
6. Departure of vehicle
7. Display the Wait Queue
8. Exit
1

Add:
Enter vehicle type (1 for Car / 2 for Scooter ):
1
Enter vehicle number: 1258
Another Car in Process. Please Wait!!
Press any key to continue...
```

Figure 4.4: Screenshot of avoiding Deadlock



```
Car Parking
1. Arrival of a vehicle
2. Total no. of vehicles parked
3. Total no. of cars parked
4. Total no. of scooters parked
5. Display order in which vehicles are parked
6. Departure of vehicle
7. Display the Wait Queue
8. Exit
7
Display

Wait Queue is...
1258
Press any key to continue...
```

Figure 4.5: Screenshot of Displaying the Wait Queue

5. Conclusion

We have handled the parking problem using semaphore. Practical implementation of parking system can also be implemented using semaphores. Some more improvement in our project because it is a busy waiting solution so every vehicle has to wait until there space is available. If the problems are handled this can be a very good solution for the actual parking scenarios.

From this project we learn a real life scenario in which semaphores can be used. There are many such scenarios which can be efficiently handled by semaphores. Synchronization is required in many places around us, which tells us about the importance of semaphores.

Similarly, many concepts of operating systems can be applied to get solution for problems around us.

6. Appendix

Code

```
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <semaphore.h>
#define CAR 1
#define SCOOTER 2
void delay(int n);
struct vehicle
{
int num ;
int row ;
int col ;
int type ;
} ;
int k=1;
void V();
void P();
#define MAX 10
int top =-1;
int waitqueue[MAX];
int parkinfo[4][10] ;  /* a 2-D array to
store number of vehicle parked */
int vehcount ;  /* to store total count of
vehicles */
```

```
int carcount ;      /* stores total count of
cars */
int scootercount ;  /* stores total count of
scooters */
void display( ) ;
void dispaly_wq();
void changecol ( struct vehicle * ) ;
struct vehicle * add ( int, int, int, int ) ;
void del ( struct vehicle * ) ;
void getfreerowcol ( int, int * ) ;
void getrcbyinfo ( int, int, int * ) ;
void show( ) ;

/* decrements the col. number by one
this fun. is called when the data is
shifted one place to left */
void changecol ( struct vehicle *v )
{
v -> col = v -> col - 1 ;
}

/* adds a data of vehicle */
struct vehicle * add ( int t, int num, int
row, int col )
{
struct vehicle *v ;

v = ( struct vehicle * ) malloc ( sizeof (
struct vehicle ) ) ;

v -> type = t ;
v -> row = row ;
v -> col = col ;
if(k==0){
printf("Another Car in Process. Please
Wait!!");
if(top == MAX-1)
{
printf("\n Wait Queue is full!!");
}
else
{
top=top+1;
waitqueue[top]=num;
}
}
if ( t == CAR && k==1)
{
P();
carcount++ ;
//delay(10000);
parkinfo[row][col] = num ;
vehcount++ ;
```

```
//V();
}
else if(t== SCOOTER && k==1)

{
P();
scootercount++ ;
parkinfo[row][col] = num ;
//V();
vehcount++ ;
}

// parkinfo[row][col] = num ;

return v ;
}

/* deletes the data of the specified
car from the array, if found */
void del ( struct vehicle *v )
{
int c ;

for ( c = v -> col ; c < 9 ; c++ )
parkinfo[v  ->  row][c]  =  parkinfo[v  ->
row][c+1] ;

parkinfo[v -> row][c] = 0 ;

if ( v -> type == CAR )
carcount-- ;
else
scootercount-- ;

vehcount-- ;
}

/* get the row-col position for the vehicle
to be parked */
void getfreerowcol ( int type, int *arr )
{
int r, c, fromrow = 0, torow = 2 ;

if ( type == SCOOTER )
{
fromrow += 2 ;
torow += 2 ;
}

for ( r = fromrow ; r < torow ; r++ )
{
for ( c = 0 ; c < 10 ; c++ )
{
```

```
if ( parkinfo[r][c] == 0 )
{
arr[0] = r ;
arr[1] = c ;
return ;
}
}
}

if ( r == 2 || r == 4 )
{
arr[0] = -1 ;
arr[1] = -1 ;
}
}

/* get the row-col position for the vehicle
with specified number */
void getrcbyinfo ( int type, int num, int
*arr )
{
int r, c, fromrow = 0, torow = 2 ;

if ( type == SCOOTER )
{
fromrow += 2 ;
torow += 2 ;
}

for ( r = fromrow ; r < torow ; r++ )
{
for ( c = 0 ; c < 10 ; c++ )
{
if ( parkinfo[r][c] == num )
{
arr[0] = r ;
arr[1] = c ;
return ;
}
}
}

if ( r == 2 || r == 4 )
{
arr[0] = -1 ;
arr[1] = -1 ;
}
}

/* displays list of vehicles parked */
void display( )
{
int r, c ;
```

```
printf ( "Cars ->\n" ) ;

for ( r = 0 ; r < 4 ; r++ )
{
if ( r == 2 )
printf ( "Scooters ->\n" ) ;

for ( c = 0 ; c < 10 ; c++ )
printf ( "%d\t", parkinfo[r][c] ) ;
printf ( "\n" ) ;
}
}
void display_wq()
{
int i;
if(top==-1)
{
printf("\nWait Queue is empty!!");
}
else
{
printf("\nWait Queue is...\n");
for(i=top;i>=0;--i)
printf("%d\t",waitqueue[i]);
}
}
void main( )
{
int choice, type, number, row = 0, col = 0
;
int i, tarr[2] ;
int finish = 1 ;
struct vehicle *v ;

/* creates a 2-D array of car and scooter
class */
struct vehicle *car[2][10] ;
struct vehicle *scooter[2][10] ;


/* displays menu and calls corresponding
functions */
while ( finish )
{

printf ( "\nCar Parking\n" ) ;
printf ( "1. Arrival of a vehicle\n" ) ;
printf ( "2. Total no. of vehicles
parked\n" ) ;
printf ( "3. Total no. of cars parked\n" ) ;
printf ( "4. Total no. of scooters
parked\n" ) ;
```

```
printf ( "5. Display order in which
vehicles are parked\n" ) ;
printf ( "6. Departure of vehicle\n" ) ;
printf ( "7. Display the Wait Queue\n" ) ;
printf ( "8. Exit\n" ) ;
scanf ( "%d", &choice ) ;

switch ( choice )
{
case  1 :

printf ( "\nAdd: \n" ) ;

type = 0 ;

/* check for vehicle type */
while ( type != CAR && type !=
SCOOTER )
{
printf ( "Enter vehicle type (1 for Car / 2
for Scooter ): \n" ) ;
scanf ( "%d", &type ) ;
if ( type != CAR && type != SCOOTER
)
   printf ( "\nInvalid vehicle type.\n" ) ;
}

printf ( "Enter vehicle number: " ) ;
scanf ( "%d", &number ) ;

/* add cars' data */



if ( type == CAR || type == SCOOTER )
{
getfreerowcol ( type, tarr ) ;

if ( tarr[0] != -1 && tarr[1] != -1 )
{
   row = tarr[0] ;
   col = tarr[1] ;

   if ( type == CAR )
      car[row][col] =  add ( type, number,
row, col ) ;
   else
      scooter[row - 2][col] = add ( type,
number, row, col ) ;
}
else
{
   if ( type == CAR )
```

```
        printf ( "\nNo parking slot free to
park a car\n" ) ;
    else
        printf ( "\nNo parking slot free to
park a scooter\n" ) ;
}
}
else
{
printf ( "Invalid type\n" ) ;
break ;
}

printf ( "\nPress any key to continue..." )
;
getch( ) ;
break ;

case  2 :
printf ( "Total vehicles parked: %d\n",
vehcount ) ;
printf ( "\nPress any key to continue..." )
;
getch( ) ;
break ;

case  3 :

printf ( "Total cars parked: %d\n",
carcount ) ;
printf ( "\nPress any key to continue..." )
;
getch( ) ;
break ;

case  4 :

printf ( "Total scooters parked: %d\n",
scootercount ) ;
printf ( "\nPress any key to continue..." )
;
getch( ) ;
break ;

case  5 :

printf ( "Display\n" ) ;
display( ) ;

printf ( "\nPress any key to continue..." )
;
getch( ) ;
break ;
```

```
case  6 :
printf ( "Departure\n" ) ;
type = 0 ;
/* check for vehicle type */
while  (  type  !=  CAR  &&  type  !=
SCOOTER )
{
  printf ( "Enter vehicle type (1 for Car / 2
for Scooter ): \n" ) ;
scanf ( "%d", &type ) ;
if ( type != CAR && type != SCOOTER
)
      printf ( "\nInvalid vehicle type.\n" ) ;
}
printf ( "Enter number: "  ) ;
scanf ( "%d", &number ) ;

if ( type == CAR || type == SCOOTER )
{
getrcbyinfo ( type, number, tarr ) ;
if ( tarr[0] != -1 && tarr[1] != -1 )
{
col = tarr [1] ;
/* if the vehicle is car */
if ( type == CAR )
{
row = tarr [0] ;
del ( car [row][col] ) ;

                for ( i = col ; i < 9 ; i++ )
          {
              car[row][i] = car[row][i + 1] ;
             // changecol ( car[row][col] ) ;
          }
free ( car[row][i] ) ;

  car[row][i] = NULL ;
}
      /* if a vehicle is scooter */
      else
{
row = tarr[0] - 2 ;
          if ( ! ( row < 0 ) )
          {
              del ( scooter[row][col] ) ;
              for ( i = col ; i < 9 ; i++ )
              {
                  scooter[row][i]                    =
scooter[row][i + 1] ;
                  //changecol                          (
scooter[row][col] ) ;
              }
```

```
       scooter[row][i] = NULL ;
}
}
}
  else
{
if ( type == CAR )
     printf ( "\nInvalid car number, or a
car with such number has not been
parked here.\n" ) ;
else
printf ( "\nInvalid scooter number, or a
scooter with such number has not been
parked here.\n" ) ;
}
}
printf ( "\nPress any key to continue..." )
;
getch( ) ;
break ;

case  7 :

printf ( "Display\n" ) ;
display_wq( ) ;

printf ( "\nPress any key to continue..." )
;
getch( ) ;
break ;

case  8 :

for ( row = 0 ; row < 2 ; row++ )
{
for ( col = 0 ; col < 10 ; col++ )
{
   if ( car[row][col] -> num != 0 )
      free ( car[row][col] ) ;
   if ( scooter[row][col] -> num != 0 )
      free ( scooter[row+2][col] ) ;
}
}
finish = 0 ;
break ;
}
}
}

void delay(int n)
{
int ms=1000*n;
clock_t start_time=clock();
```

```
printf("Another car in process,wait\n");
while(clock()<start_time+ms);
//printf("Now you go\n");

}

void P()
{
printf("Car is Parking.\n");
if (k == 0) {

printf("Wait!!     Another     Car     In
Process!!");
// add process to queue
delay(10000);
}
else{
k = k - 1;
}
}
void V()
{
k = k + 1;

}
```