Student Name: Tamasjit Ghoshal

Student ID: 11811520

Section: EE032

Roll no. A32

Sub.: CSE316 (Assignment)

Email Address: tamasjit789@gmail.com

GitHub Link: https://github.com/tamasjit/oscarbridge

Problem: (Q11) You have been hired by Coltrans Limited to automate the flow of traffic on a one-lane bridge that has been the site of numerous collisions. Coltrans Limited wants you to implement the following rules:

● Traffic can flow in only a single direction on the bridge at a time.

● Any number of cars can be on the bridge at the same time, as long as they are all traveling in the same direction.

● To avoid starvation, you must implement the

"five car rule": once 5 or more consecutive northbound cars have entered the bridge, if there are any southbound cars waiting then no more northbound cars may enter the bridge until some southbound cars have crossed. A similar rule also applies once 5 or more consecutive southbound cars have entered the bridge.

You must implement the traffic flow mechanism in C by defining a structure struct bridge, plus five functions described below.

When a northbound car arrives at the bridge, it invokes the function:

bridge_arrive_north(struct bridge *b)

This function must not return until it is safe for the car to cross the bridge, according to the rules above. Once a northbound car has finished crossing the bridge it will invoke the function: bridge_leave_north(struct bridge *b)

Southbound cars will invoke analogous functionsbridge_arrive_southand bridge_leave_south. Use the next pages to write a declaration forstruct bridge and the four functions above, plus the functionbridge_init, which will be invoked to initialize the bridge

You must write your solution in C using the functions for locks and condition variables: lock_init (struct lock *lock)

lock_acquire(struct lock *lock)

lock_release(struct lock *lock)

cond_init(struct condition *cond)

cond_wait(struct condition *cond, struct lock *lock) cond_signal(struct condition *cond, struct lock *lock)

cond_broadcast(struct condition *cond, struct lock *lock)

Use only these functions (e.g., no semaphores or other synchronization primitives).

You may not use more than one lock in each struct bridge.

Your solution must not use busy-waiting.

If your southbound functions are identical to the northbound functions except that north is replaced with south (and vice versa) in all identifiers, then you can omit the southbound functions and just circle this bullet point.

Code:

```
#include<stdio.h>
unsigned int northwait[20];
unsigned int soutwait[20];
unsigned int nx,sx;
struct lock
{
   unsigned int lka;
};
void lock_init(struct lock lko){
   lko.lka = 0;
}
//initilization as per
struct bridge
{

   unsigned int north_timewaiting;
   unsigned int north_timecrossing;
   unsigned int north_timeconsecutive;

   unsigned int south_timewaiting;
   unsigned int south_timecrossing;
   unsigned int south_timeconsecutive;

   struct lock llk;
}operatingbridge;




//lock acquire
void acquireing_lock_north()
{
   operatingbridge.llk.lka = 2;
}
void bridge_init(struct bridge operetingsys)
{
   operetingsys.north_timewaiting = 0;
   operetingsys.north_timecrossing = 0;
   operetingsys.north_timeconsecutive = 0;
   operetingsys.south_timeconsecutive = 0;
   operetingsys.south_timecrossing = 0;
```

```
    operetingsys.south_timewaiting = 0;
    lock_init(operetingsys.llk);
}
void acquireing_lock_south()
{
    operatingbridge.llk.lka = 1;
}
void south_empty_wait(){

    unsigned int loopvo;
    for(loopvo = 0; loopvo < sx; loopvo++)
    {
        printf("\tSouth Car %d Process gets executed::::: \n",soutwait[loopvo]);
    }
    puts("");
    sx = 0;

}
//North empyty wait
void north_empty_wait()
{
    unsigned int loopvo;
    for(loopvo = 0; loopvo < nx; loopvo++)
    {
        printf("\tNorth Car %d Process gets executed::::: \n",northwait[loopvo]);
    }
    puts("");
    nx = 0;
}
//check
void bridge_arrive_south(unsigned int call_ver_t)
{
    if(operatingbridge.llk.lka == 0 || operatingbridge.llk.lka == 1)
            {
                printf("\tSouth Car %d process gets executed:::::\n",call_ver_t);
                acquireing_lock_south();
            }
            else
            {
                printf("\tSouth Car %d process goes to waiting.....\n",call_ver_t);
                soutwait[sx] = call_ver_t;
                sx++;
                if(sx >= 5)
                {
                    acquireing_lock_south();
                    south_empty_wait();
                }
            }
}
void bridge_arrive_north(unsigned int call_ver_o)
```

```
{
   if(operatingbridge.llk.lka == 0 || operatingbridge.llk.lka == 2)
         {
            printf("\tNorth Car %d process gets executed:::::\n",call_ver_o);
            acquireing_lock_north();
         }
         else
         {
            printf("\tNouth Car %d process goes to waiting.....\n",call_ver_o);
            northwait[nx] = call_ver_o;
            nx++;
            if(nx >= 5)//“five car rule” test case
            {
               acquireing_lock_north();
               north_empty_wait();
            }
         }
}

unsigned int partition(unsigned int a[], unsigned int low, unsigned int high);

unsigned int partition (unsigned int a[], unsigned int low, unsigned int high)
{
   unsigned int pivot = a[high];
   unsigned int i = (low-1);
   unsigned int j;
   for ( j = low; j <= high-1; j++)
   {

      if (a[j] <= pivot)
      {
         i++;
         unsigned int t = a[i];
         a[i] = a[j];
         a[j] = t;
      }
   }
   unsigned int t = a[i + 1];
   a[i+1] = a[high];
   a[high] = t;
   return (i + 1);
}
void sort_arival(unsigned int a[], unsigned int low, unsigned int high)

{
         if (low < high)
   {

      unsigned int pi=partition(a, low, high);
            sort_arival(a, low, pi - 1);
```

```
        sort_arival(a, pi + 1, high);
    }
}
unsigned int south_cars, north_cars;
unsigned int main()
{
        unsigned int mainver;

    printf("~~~~~~~~~~~Let all the cars are inputed according ascending order arrival
time~~~~~~~~~~~\n\n");


    printf("\tNumber of total north car:\t");
    scanf("%d",&north_cars);
    unsigned int north_arival[north_cars];
    for( mainver = 0; mainver < north_cars; mainver++)
    {
        printf("\tTime of north arrival of the car %d:\t",mainver+1);
        scanf("%d",&north_arival[mainver]);
        }

        printf("\tNumber of total south car:\t");
    scanf("%d",&south_cars);
    unsigned int south_arival[south_cars];
    for( mainver = 0; mainver < south_cars; mainver++)
    {
        printf("\tTime of south arrival of the car %d:\t",mainver+1);
        scanf("%d",&south_arival[mainver]);
        }
    printf("~~~~~~~~~~Cars of North~~~~~~~~~~\n");
    printf("\tNo.\tArrival\tLeave\n");
    for(mainver = 0; mainver < north_cars; mainver++)
    {
        printf("\t%d\t%d\n",mainver,north_arival[mainver]);
    }
    printf("~~~~~~~~~~Cars of South~~~~~~~~~~\n");
    printf("\tNo.\tArrival\tLeave\n");
    for(mainver = 0; mainver < south_cars; mainver++)
    {
        printf("\t%d\t%d\n",mainver,south_arival[mainver]);
    }
    printf("~~~~~~~~~~~Start Whole System Processing and Execution~~~~~~~~~~~\n\n");
    // Input SetUp done;
    bridge_init(operatingbridge);
    mainver = 0;
    unsigned int call_ver_o = 0;
    unsigned int call_ver_t = 0;
    do{
        if(north_arival[call_ver_o] <= south_arival[call_ver_t])
        {
```

```
            bridge_arrive_north(call_ver_o);
            call_ver_o++;
        }
        else
        {
            bridge_arrive_south(call_ver_t);
            call_ver_t++;
        }
    }
    while(call_ver_o < north_cars && call_ver_t < south_cars);
 south_empty_wait();
 north_empty_wait();
    if(call_ver_o == north_cars)
    {
        for(call_ver_t; call_ver_t <south_cars; call_ver_t++)
        {
            printf("\tSouth Car %d process gets executed:::::\n",call_ver_t);
        }
    }
    else
    {
        for(call_ver_o; call_ver_o <north_cars; call_ver_o++)
        {
            printf("\tNorth Car %d process gets executed:::::\n",call_ver_o);
        }
    }
}
```

Description:

The concept of lock is used to solve the problem. Locks are method of synchronization used to prevent multiple threads from accessing a resource at the same time. A lock variable provides the simplest synchronization mechanism for processes. Here in the solution, lock variable is used. In the solution, traffic can flow in only one direction. So, for the northbound cars first lock is acquired, meanwhile the southbound cars can pass the bridge. Then the lock for northbound cars are released and lock for southbound cars are acquired. So, the northbound cars then pass the bridge. In this case, when there 5 cars enter into the bridge then the lock is acquired for both the bounds. The concept of releasing and acquiring lock of operating system is used here. Basically, the lock ensure that there is no starvation case occurred.

Algorithm:

1.Lock is acquired for the northbound cars; then southbound cars are passed through the bridge.

2.Then the northbound lock is released, and southbound cars acquire lock.

3.Then the northbound cars pass through the bridge.

Purpose of use:

This algo can be used in the purpose of the memory read and write. In memory management we can assume north bounded car as the operation which are need to read from and the south bound cars as which are write on the memory. We know there are single bus in the CPU and memory interfacing according architecture and two singles are used to tell what read/write process to be done in the memory. And this algo is also helpful to stop high number of processes whole process until another important process successfully executed.

Test cases:

North Bridge >=5 or <=5

South Bridge >=5 or <=5

Boundaries:

Any input from unsigned integer range:

Negative value is prohibited

0 to 4,294,967,295

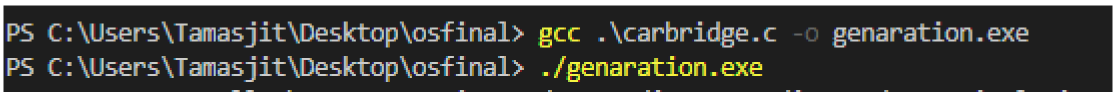
Constraints:

North Car Number> 0

South Car Number> 0

Output:

Code Compilation:

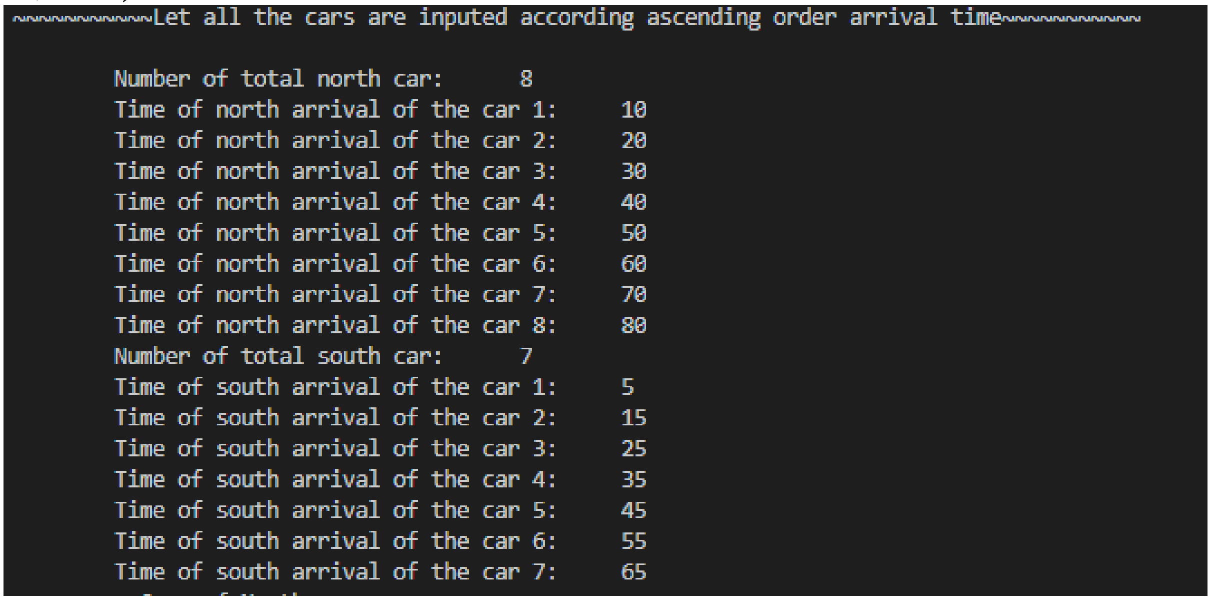


```
PS C:\Users\Tamasjit\Desktop\osfinal> gcc .\carbridge.c -o genaration.exe
PS C:\Users\Tamasjit\Desktop\osfinal> ./genaration.exe
```

For the coding purpose I used Visual Studio Code and compiled it through gcc in Windows Power Shell terminal.

Input Giving:

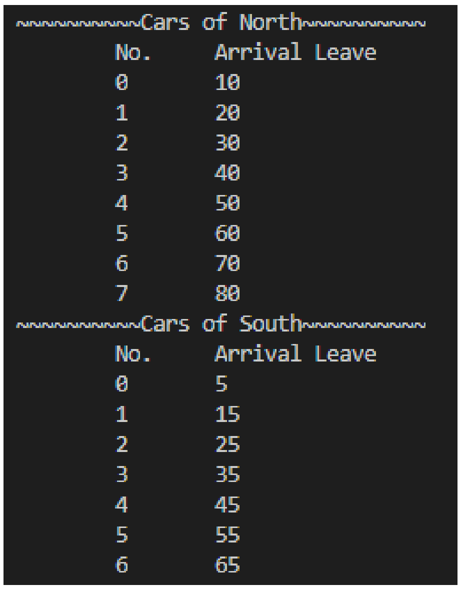
1.The number of northbound cars and southbound cars are taken from user. Here the number of northbound car is 8 (10, 20, 30, 40, 50, 60, 70, 80) and southbound car is 7(5, 15, 25, 35, 45, 55, 65) . Their time of arrival is also taken from then user.



```
~~~~~~~~~~~~~Let all the cars are inputed according ascending order arrival time~~~~~~~~~~~~~

        Number of total north car:      8
        Time of north arrival of the car 1:     10
        Time of north arrival of the car 2:     20
        Time of north arrival of the car 3:     30
        Time of north arrival of the car 4:     40
        Time of north arrival of the car 5:     50
        Time of north arrival of the car 6:     60
        Time of north arrival of the car 7:     70
        Time of north arrival of the car 8:     80
        Number of total south car:      7
        Time of south arrival of the car 1:     5
        Time of south arrival of the car 2:     15
        Time of south arrival of the car 3:     25
        Time of south arrival of the car 4:     35
        Time of south arrival of the car 5:     45
        Time of south arrival of the car 6:     55
        Time of south arrival of the car 7:     65
```

2. Displaying user what are the input in given to each car as per the number of the car and the input time.

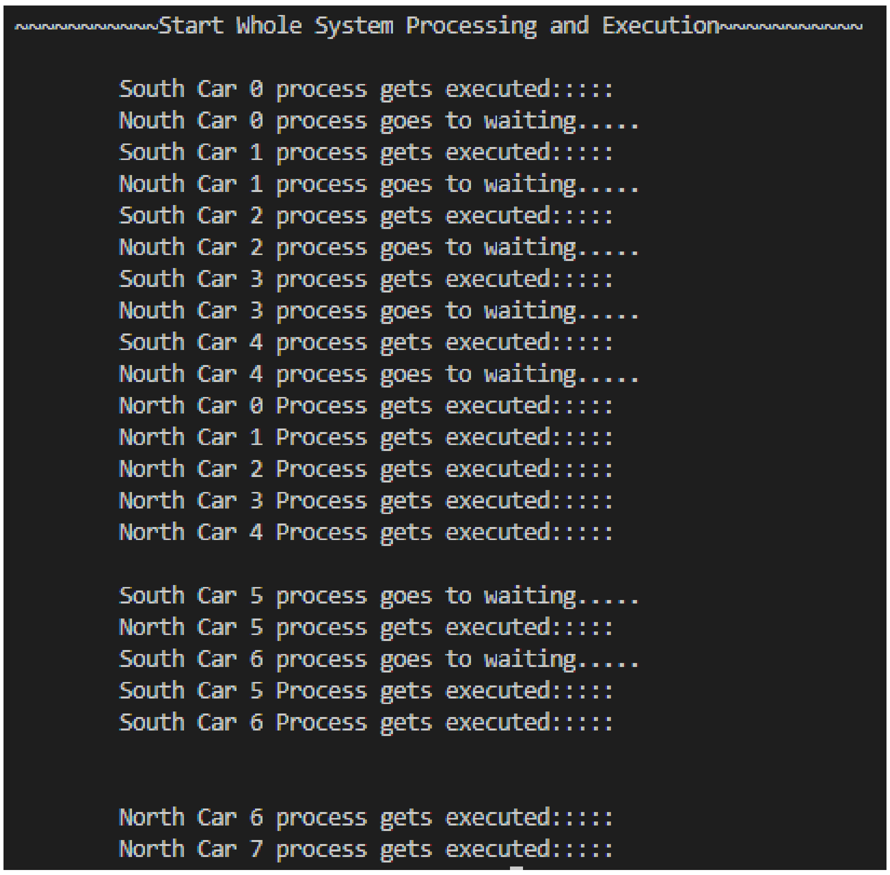


```
~~~~~~~~~~~Cars of North~~~~~~~~~~~
        No.     Arrival Leave
        0       10
        1       20
        2       30
        3       40
        4       50
        5       60
        6       70
        7       80
~~~~~~~~~~~Cars of South~~~~~~~~~~~
        No.     Arrival Leave
        0       5
        1       15
        2       25
        3       35
        4       45
        5       55
        6       65
```

3.As northbound cars are greater than the 5 so they are go to waiting state until south bound cars got executed to make that bridge one way all time.

4.Here the north cars are executed or passed through the bridge. This is how the starvation case is avoided using locks.

5. After successful execution of the of all south bound cars north bound cars executed one by one.



```
~~~~~~~~~~~~~Start Whole System Processing and Execution~~~~~~~~~~~~~

        South Car 0 process gets executed:::::
        Nouth Car 0 process goes to waiting.....
        South Car 1 process gets executed:::::
        Nouth Car 1 process goes to waiting.....
        South Car 2 process gets executed:::::
        Nouth Car 2 process goes to waiting.....
        South Car 3 process gets executed:::::
        Nouth Car 3 process goes to waiting.....
        South Car 4 process gets executed:::::
        Nouth Car 4 process goes to waiting.....
        North Car 0 Process gets executed:::::
        North Car 1 Process gets executed:::::
        North Car 2 Process gets executed:::::
        North Car 3 Process gets executed:::::
        North Car 4 Process gets executed:::::

        South Car 5 process goes to waiting.....
        North Car 5 process gets executed:::::
        South Car 6 process goes to waiting.....
        South Car 5 Process gets executed:::::
        South Car 6 Process gets executed:::::


        North Car 6 process gets executed:::::
        North Car 7 process gets executed:::::
```

GitHub Link: https://github.com/tamasjit/oscarbridge