Network Lab Exam Question 3-A

Albin Antony
Roll No:10
TVE16CS010

25 April 2019

1 Narrow Brigde

1.1 Problem Statement

A city is built on two islands connected by a narrow bridge. There are many cars driving throughout the city and occasionally crossing the bridge. The bridge is only wide enough for traffic in one direction at a time. Because the bridge is narrow the cars must also travel slowly while crossing the bridge (i.e. it should take some time). There is no traffic light.

When a car decides to cross the bridge, one of three situations can occur:

- i) the bridge is free, in which case the car may cross.
- ii) the bridge is occupied, and the traffic on the bridge is travelling in the right direction, so the car is allowed to cross.
- iii) the bridge is occupied, but the traffic is in the wrong direction. The car must either wait until the bridge is free, or come back later and try again.

Make sure that you have enough cars, and that they decide to cross the bridge often enough that all three of the traffic situations can occur. Use one process to represent one car.

1.2 Theory

1.2.1 Thread

A thread is a path of execution within a process. A process can contain multiple threads.A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads.A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes. Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space

1.2.2 Mutex

A Mutex is a lock that we set before using a shared resource and release after using it. When the lock is set, no other thread can access the locked region of code. So we see that even if thread 2 is scheduled while thread 1 was not done accessing the shared resource and the code is locked by thread 1 using mutexes then thread 2 cannot even access that region of code. So this ensures a synchronized access of shared resources in the code.

1.3 Algorithm

Algorithm 1 Algorithm for Narrow Server

```
Define mutex bridge_mutex
Define mutex Traffic_mutex

int traffic = 0

Function car_left:
    if traffic <=0:
        Car waiting for left....
    lock(Traffic_mutex)
    traffic++
    unlock(Traffic_mutex)
    lock(bridge_mutex)
    Car crossing....
    unlock(bridge_mutex)
    lock(Traffic_mutex)
    traffic--
    unlock(Traffic_mutex)

Function car_right:
    if traffic >=0:
        Car waiting for right....
    lock(Traffic_mutex)
    traffic--
    unlock(Traffic_mutex)
    lock(bridge_mutex)
    Car crossing....
    unlock(bridge_mutex)
    lock(Traffic_mutex)
    traffic++
    unlock(Traffic_mutex)

Create thread for cars randomly:
    if(car going left):
        car_left
    if(car going right):
        car_right
```

Each car calls a thread to cross a bridge.For a car going left carleft is used and for a car going right carright is used. A counter uses count of traffic to right and left.If counter is positive cars are going left and any car going left can access the lock.If counter is negative all cars on bridge are going right.All cars going right can enter the bridge.

1.4 Program

```
//AUTHOR : Albin Antony
//         Roll No:10

#include<iostream>
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>
#include<unistd.h>
#include<time.h>
using namespace std;

pthread_mutex_t b_mutex; //bridge mutex
pthread_mutex_t t_mutex; //trafiic mutex
int traffic=0;
//positive value means number of traffic to left.
//Negative means number of traffic to right

void *left(void *arg){
  long carid = (long)arg;
  printf("Car %d arrived for left\n",carid);//Car arrived
  if(traffic<0){
    printf("Car %d waiting for left\n",carid);//If car
  }                                            //if moving
     left

  pthread_mutex_lock(&t_mutex); //locking traffic
  traffic++;
  pthread_mutex_unlock(&t_mutex);//unlocking traffic

  pthread_mutex_lock(&b_mutex);
    printf("Car %d passing\n",carid);
    sleep(3);
  pthread_mutex_unlock(&b_mutex);

  pthread_mutex_lock(&t_mutex);//locking traffic
  traffic--;
  pthread_mutex_unlock(&t_mutex);//unlocking traffic
  pthread_exit(NULL);
}
```

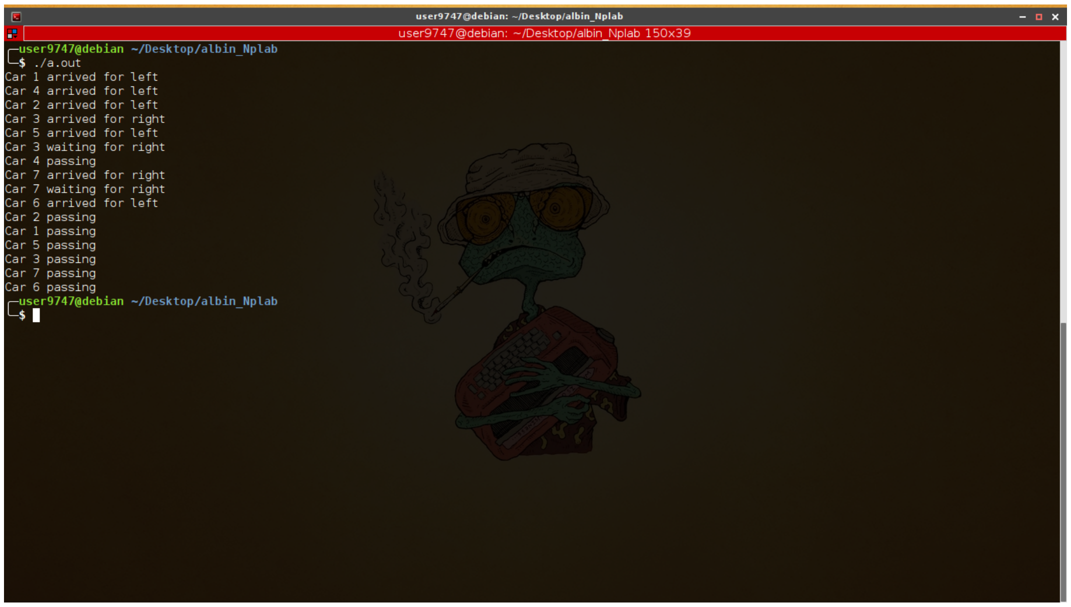
```
void *right(void *arg){
 long carid = (long)arg;
  printf("Car %d arrived for right\n",carid);//Car arrived
  if(traffic >0){
    printf("Car %d waiting for right\n",carid);//If car is
  }                                             //moving
      right

  pthread_mutex_lock(&t_mutex); //locking traffic
  traffic--;
  pthread_mutex_unlock(&t_mutex);//unlocking traffic

  pthread_mutex_lock(&b_mutex);
    printf("Car %d passing\n",carid);
    sleep(3);
  pthread_mutex_unlock(&b_mutex);

  pthread_mutex_lock(&t_mutex); //locking traffic
  traffic++;
  pthread_mutex_unlock(&t_mutex);        //unlocking traffic
  pthread_exit(NULL);

}
int main(){
  pthread_t thread;

  pthread_create(&thread,NULL,left,(void *)1);
  pthread_create(&thread,NULL,left,(void *)2);
  pthread_create(&thread,NULL,right,(void *)3);
  pthread_create(&thread,NULL,left,(void *)4);
  pthread_create(&thread,NULL,left,(void *)5);
  pthread_create(&thread,NULL,left,(void *)6);
  pthread_create(&thread,NULL,right,(void *)7);

  pthread_exit(NULL);
  return 0;
}
```

1.4.1 To run the program:

```
g++ bridge.cpp -lpthread
```

1.5 Output

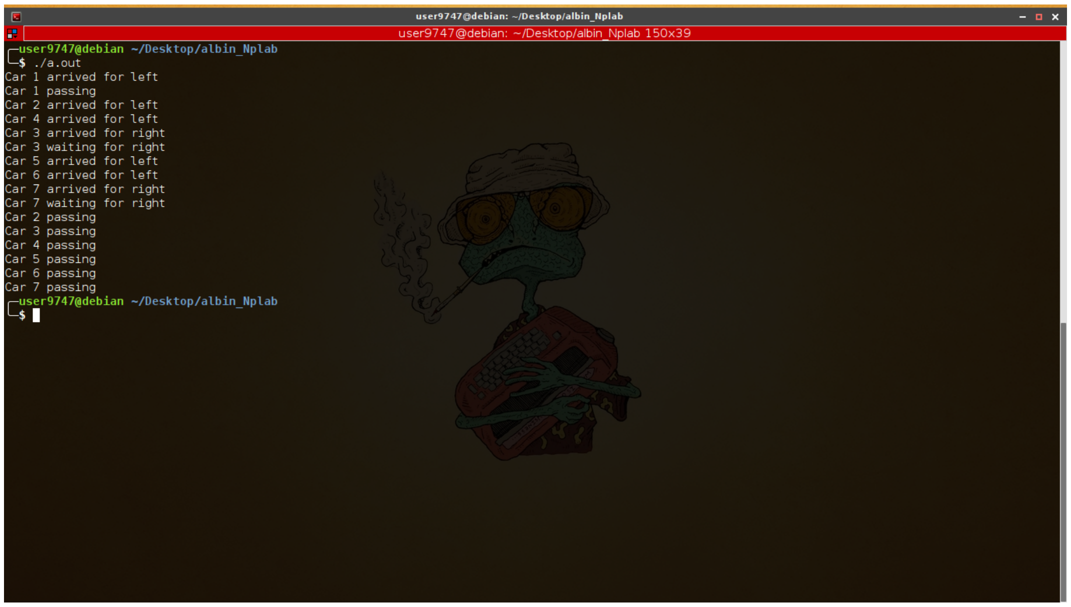
1.5.1 Test case 1



user9747@debian: ~/Desktop/albin_Nplab
user9747@debian: ~/Desktop/albin_Nplab 150x39

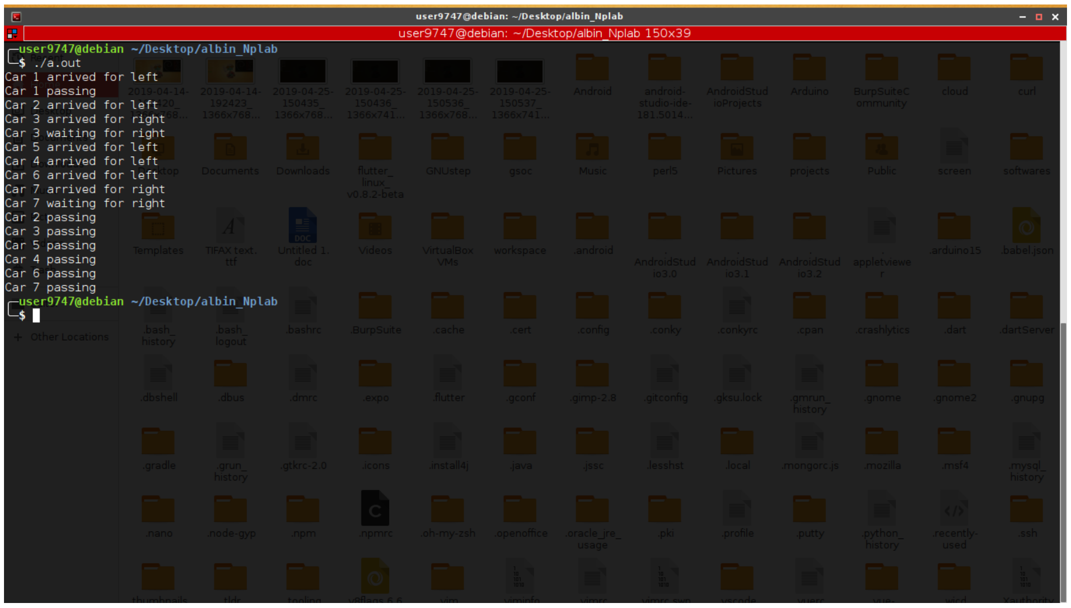
```
┌─user9747@debian ~/Desktop/albin_Nplab
└─$ ./a.out
Car 1 arrived for left
Car 4 arrived for left
Car 2 arrived for left
Car 3 arrived for right
Car 5 arrived for left
Car 3 waiting for right
Car 4 passing
Car 7 arrived for right
Car 7 waiting for right
Car 6 arrived for left
Car 2 passing
Car 1 passing
Car 5 passing
Car 3 passing
Car 7 passing
Car 6 passing
┌─user9747@debian ~/Desktop/albin_Nplab
└─$
```

1.5.2 Test case 2


user9747@debian: ~/Desktop/albin_Nplab
user9747@debian: ~/Desktop/albin_Nplab 150x39

```
┌─user9747@debian ~/Desktop/albin_Nplab
└─$ ./a.out
Car 1 arrived for left
Car 1 passing
Car 2 arrived for left
Car 4 arrived for left
Car 3 arrived for right
Car 3 waiting for right
Car 5 arrived for left
Car 6 arrived for left
Car 7 arrived for right
Car 7 waiting for right
Car 2 passing
Car 3 passing
Car 4 passing
Car 5 passing
Car 6 passing
Car 7 passing
┌─user9747@debian ~/Desktop/albin_Nplab
└─$ 
```

1.5.3 Test case 3


user9747@debian: ~/Desktop/albin_Nplab
user9747@debian: ~/Desktop/albin_Nplab 150x39

```
┌─user9747@debian ~/Desktop/albin_Nplab
└─$ ./a.out
Car 1 arrived for left
Car 1 passing
Car 2 arrived for left
Car 3 arrived for right
Car 3 waiting for right
Car 5 arrived for left
Car 4 arrived for left
Car 6 arrived for left
Car 7 arrived for right
Car 7 waiting for right
Car 2 passing
Car 3 passing
Car 5 passing
Car 4 passing
Car 6 passing
Car 7 passing
┌─user9747@debian ~/Desktop/albin_Nplab
└─$ 
```