LOVELY PROFESSIONAL UNIVERSITY

Class Assignment-2

Simulation Based Assignment Report

Subject:- Operating systems

Term: Max. Marks:-30 **Date of submission:- 02-04-2020**

By

Thamminaina Venkatesh

Registration No:- 11704920

Roll No:- 06

Section:- EE033

Group:- 01

Stream:- Electronics and Communication Engineering

Email Id:- venkatesh4973771@gmail.com

Github link:- https://github.com/venkateshthamminaina/Thamminaina-Venkatesh

Submitted to:- Nahida Nazir

Transforming Education Transforming India

School of Electrical and Electronics Engineering

Lovely Professional University, Phagwara, Punjab

The question is:-

- Indian Rail has decided to improve its efficiency by automating not just its trains but also its passengers. Each passenger and each train is controlled by a thread. You have been hired to write synchronization functions that will guarantee orderly loading of trains. You must define a structure struct station, plus several functions described below. When a train arrives in the station and has opened its doors, it invokes the function station_load_train(struct station *station, int count) where count indicates how many seats are available on the train. The function must not return until the train is satisfactorily loaded (all passengers are in their seats, and either the train is full or all waiting passengers have boarded). When a passenger arrives in a station, it first invokes the function station_wait_for_train (struct station *station). This function must not return until a train is in the station (i.e., a call to station_load_train is in progress) and there are enough free seats on the train for this passenger to sit down. Once this function returns, the passenger robot will move the passenger on board the train and into a seat (you do not need to worry about how this mechanism works). Once the passenger is seated, it will call the function station_on_board(struct station *station) to let the train know that it's on board. Create a file IndianRail.c that contains a declaration for struct station and defines the three functions above, plus the function station_init, which will be invoked to initialize the station object when IndianRail boots. In addition:

  You must write your solution in C using 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 a single lock in each struct station.

● You may assume that there is never more than one train in the station at once, and that all trains (and all passengers) are going to the same destination (i.e. any passenger can board any train).

● Your code must allow multiple passengers to board simultaneously (it must be possible for several passengers to have called station_wait_for_train, and for that function to have returned for each of the passengers, before any of the passengers calls station_on_board).

● Your code must not result in busy-waiting.

Semaphores

It's simple and always have a non-negative Integer value. Works with many processes. Can have many different critical sections with different semaphores. Each critical section has unique access semaphores. Can permit multiple processes into the critical section at once, if desirable

Semaphores are mainly of two types:

- Binary Semaphore
- Counting Semaphore

Binary Semaphore:-

It is a special form of semaphore used for implementing mutual exclusion, hence it is often called a Mutex. A binary semaphore is initialized to 1 and only takes the values 0 and 1during execution of a program.

Counting Semaphore:-

These are used to implement bounded concurrency.

Thread

A thread is a path of execution with in a process. A process can contain multiple threads. A thread is also known as light weight process. The idea is to achieve parallelism by dividing a process in to multiple threads. For example, in a browser, multiple tabs can be different threads. MSWord uses multiple threads: one thread to format the text, another thread to process inputs, etc.

There are two types of threads:-

- User level Thread
- Kernel level Thread

User level Thread:-

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

Kernel level Thread:-

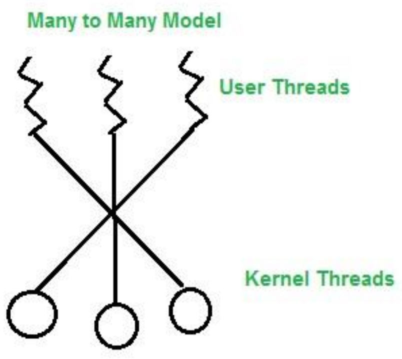
In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

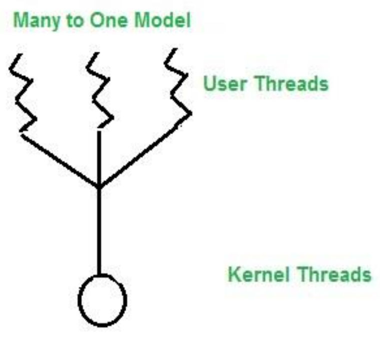
Multi Threading Models

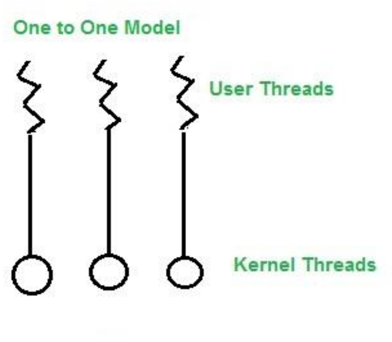
Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process.

Multi threading models are three types:-

- Many to Many Model
- Many to one model
- One to One Model







Syntax:-

```
#include<pthread.h>
pthread_mutex_t mp=PTHREAD_MUTEX_INTIALIZER;
pthread_mutexattr_t mattr;
int ret;
/*initialize a mutex to its default value*/
ret=pthread_mutex_init(&mp,NULL);
/*initialize a mutex*/
ret=pthread_mutex_init(&mp,&mattr);
```

Mutex Locks

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released. As the resource is locked while a process executes its critical section hence no other process can access it.

Syntax to initiate a mutex:-

```
#include<synch.h>

#include<thread.h>

int mutex_init(mutex_t*mp,int type,void*arg);
```

Syntax to destroy a mutex:-

```
#include<thread.h>

int mutex_destroy(mutex_t*mp);
```

C language code

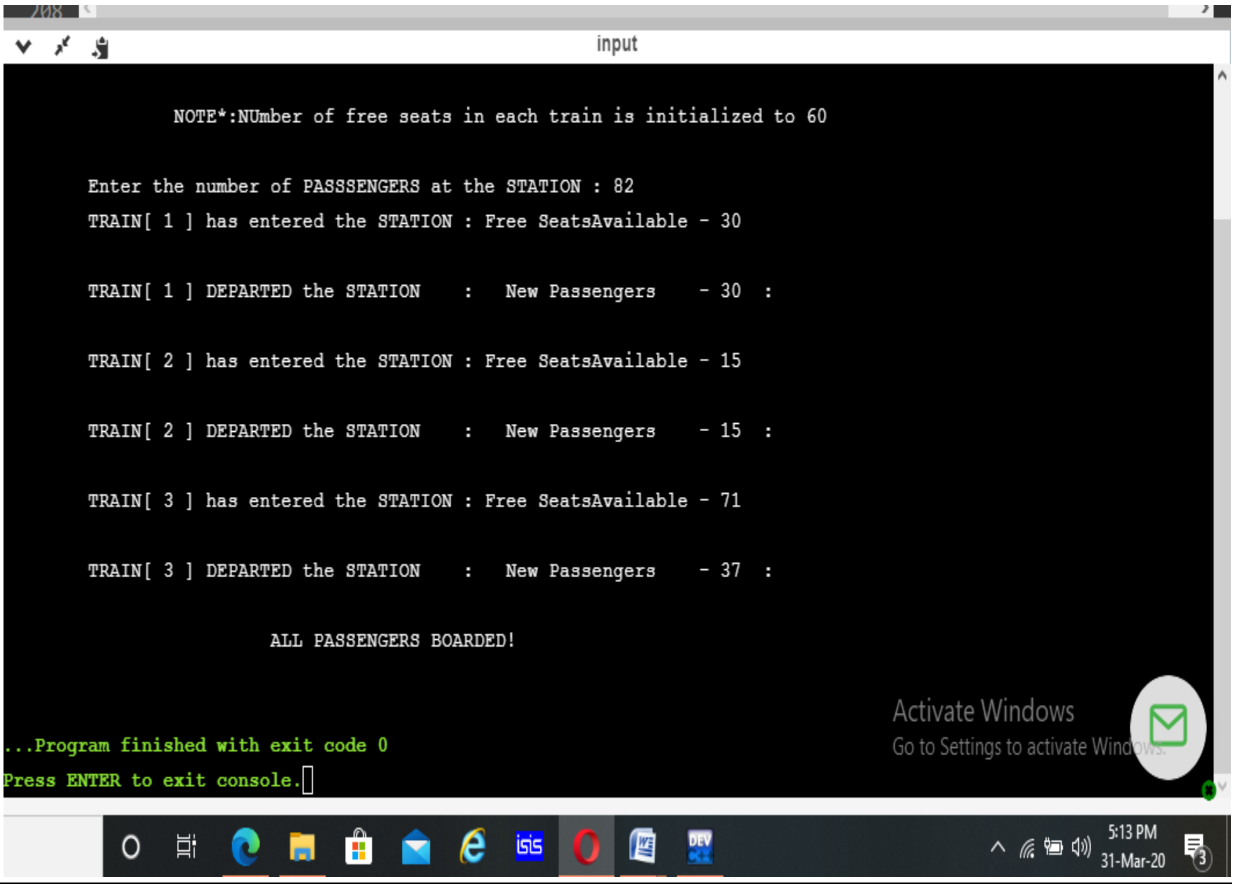
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
int num;                              //number of passengers
struct station
{
   pthread_mutex_t tpLock;
   pthread_cond_t trainArrived;
   pthread_cond_t passengerSettled;
   int boarded_passengers;         //passengers boarded inside the train
   int passengers_inStation;       //passengers waiting in the station
   int seats_vacant;               //vacant seats in the train
};
//FunctionDeclaration
int min (int x, int y);
void station_init (struct station *station);
void station_load_train (struct station *station, int count);
void station_wait_for_train (struct station *station);
void station_on_board (struct station *station);
//FunctionDefinition
void
station_init (struct station *station)
{
   pthread_mutex_init (&station->tpLock, NULL);           //initialize mutex locks
   pthread_cond_init (&station->trainArrived, NULL);     //thread condition variable
   pthread_cond_init (&station->passengerSettled, NULL);
```

```
    station->boarded_passengers = 0;
    station->passengers_inStation = 0;
    station->seats_vacant = 0;
}
/*train arrives*/
void
station_load_train (struct station *station, int count)
{
  //returns when there are no passengers or train is full
     pthread_mutex_lock (&station->tpLock);
     station->seats_vacant = count;
     while (station->seats_vacant > 0 && station->passengers_inStation > 0)
       {
          pthread_cond_broadcast (&station->trainArrived);
            pthread_cond_wait (&station->passengerSettled, &station->tpLock);
         }
       station->seats_vacant = 0;
       pthread_mutex_unlock (&station->tpLock);
}
//passenger arrives
void
station_wait_for_train (struct station *station)
{
  //return when there are enough available seats and train is in the station
    pthread_mutex_lock (&station->tpLock);
      station->passengers_inStation++;
      while (station->boarded_passengers == station->seats_vacant)
        {
            pthread_cond_wait (&station->trainArrived, &station->tpLock);
          }
        station->boarded_passengers++;
        station->passengers_inStation--;
        pthread_mutex_unlock (&station->tpLock);
 }
//passenger boarded
void
station_on_board (struct station *station)
{
  //to inform the train that it is on board
   pthread_mutex_lock (&station->tpLock);
   station->boarded_passengers--;
   station->seats_vacant--;
       if ((station->seats_vacant == 0) || (station->boarded_passengers == 0))
         {
              pthread_cond_signal (&station->passengerSettled);
          }
     pthread_mutex_unlock (&station->tpLock);
  }
 volatile int threads_completed = 0;
void *
passenger_thread (void *arg)
{
        struct station *station = (struct station *) arg;
        station_wait_for_train (station);
        threads_completed++;
        return NULL;
  }
struct TrainLoaded_Para
{
    struct station *station;
    int free_seats;
};
volatile int return_LoadTrain = 0;
void *
load_train_thread (void *args)
{
    struct TrainLoaded_Para *temp = (struct TrainLoaded_Para *) args;
    station_load_train (temp->station, temp->free_seats);
    return_LoadTrain = 1;
    return NULL;
}
//finds the minimum value among x and y
#ifndef MIN
#define MIN(_x,_y) ((_x) < (_y)) ? (_x) : (_y)
#endif
//main function starts from here
Int
main ()
{
        struct station station;
        station_init (&station);
        srandom (getpid () ^ time (NULL));            //generates random numbers
        int i;
        printf ("\n\n\n\t\t\tINDIAN RAILWAYS\n\n");
```

```
      printf("\n\t\tNOTE*:NUmber of free seats in each train is initialized to 60");
      printf ("\n\n\tEnter the number of PASSSENGERS at the STATION : ");
      scanf ("%d", &num);
      if (num < 0)
     {
      printf(" \tYou have entered number of passengers as %d which is not possible.\n",num);
      printf (" \tPlease enter a valid number!!\n");
      scanf ("%d", &num);
     }
        if (num == 0)
          {
              printf (" \t NO PASSENGERS in the STATION!!\n\n");
              return 0;
            }
   const int total_Passngrs = num;
   int remaining_Passngrs = total_Passngrs;
   for (i = 0; i < total_Passngrs; i++)
  {
    pthread_t tid;
    int ret = pthread_create (&tid, NULL, passenger_thread, &station);
  }
int total_Passngrs_boarded = 0;
const int tot_FreeSeats_PerTrain = 100;
int pass = 0;
int j = 1, p = 1;
while (remaining_Passngrs > 0)
  {
       int free_seats = random () % tot_FreeSeats_PerTrain;
```

```
          printf("  \tTRAIN[ %d ] has entered the STATION : Free SeatsAvailable - %d\n\n",j,
free_seats);

      j++;

      return_LoadTrain = 0;

      struct TrainLoaded_Para args = { &station, free_seats };

      pthread_t lt_tid;

      int ret = pthread_create (&lt_tid, NULL, load_train_thread, &args);

      if (ret != 0)

              {

                perror ("pthread_create");

                exit (1);

              }

      int threads_to_reap = MIN (remaining_Passngrs, free_seats);

      int threads_reaped = 0;

      while (threads_reaped < threads_to_reap)

              {

                 if (return_LoadTrain)

                   {

                     exit (1);

                   }

                if (threads_completed > 0)

                   {

                     if ((pass % 2) == 0)

                            usleep (random () % 2);

                     threads_reaped++;

                     station_on_board (&station);

                     threads_completed++;

                     }

              }
      remaining_Passngrs -= threads_reaped;

      total_Passngrs_boarded += threads_reaped;

      printf("  \tTRAIN[ %d ] DEPARTED the STATION    :   New Passengers    - %d  :\n\n",p,
threads_to_reap);

       pass++;

       p++;

    }

    if (total_Passngrs_boarded == total_Passngrs)

    {

      printf ("\t\t\t ALL PASSENGERS BOARDED!\n");

      return 0;

      }

}
```

Output of the code:-



```
        NOTE*:NUmber of free seats in each train is initialized to 60

    Enter the number of PASSSENGERS at the STATION : 82
    TRAIN[ 1 ] has entered the STATION : Free SeatsAvailable - 30

    TRAIN[ 1 ] DEPARTED the STATION    :   New Passengers    - 30  :

    TRAIN[ 2 ] has entered the STATION : Free SeatsAvailable - 15

    TRAIN[ 2 ] DEPARTED the STATION    :   New Passengers    - 15  :

    TRAIN[ 3 ] has entered the STATION : Free SeatsAvailable - 71

    TRAIN[ 3 ] DEPARTED the STATION    :   New Passengers    - 37  :

                      ALL PASSENGERS BOARDED!


...Program finished with exit code 0
Press ENTER to exit console.
```

Github link:-
https://github.com/venkateshthamminaina/Thamminaina-Venkatesh

Thank You