LOVELY PROFESSIONAL UNIVERSITY

Transforming Education Transforming India

# Jurassic problem using Multithreaded Programming:

**Student Name:  Shaik Rahamtulla**

**Student ID: 11705940**

**Section: E1705**

**Roll no: 41**

**Email Address:** shaikrahamtulla1999@gmail.com

**GitHub Link:**

**Code :  Jurrasic park  problem using Multithreaded Programming.**

**Submitted to :**

**Prabhdeep  Kaur mam**

**QUESTION:17 Jurassic Park consists of a dinosaur museum and a park for safari riding. There are m passengers and nsingle-passenger cars. Passengers wander around the museum for a while, then line up to take a ride in a safari car. When a car is available, it loads the one passenger it can hold and rides**

**around the park for a random amount of time. If the n cars are all out riding passengers around, then a passenger who wants to ride waits; if a car is ready to load but there are no waiting passengers, then the car waits.**

**Using synchronization tools like locks, semaphores and monitorssynchronize the m passenger processes and the n car processes**

## Multithreading in Operating System:

A **thread** is a path which is followed during a program's execution. Majority of programs written now a days run as a single thread.Lets say, for example a program is not capable of reading keystrokes while making drawings. These tasks cannot be executed by the program at the same time. This problem can be solved through multitasking so that two or more tasks can be executed simultaneously.

Multitasking is of two types: Processor based and thread based. Processor based multitasking is totally managed by the OS, however multitasking through multithreading can be controlled by the programmer to some extent.

The concept of **multi-threading** needs proper understanding of these two terms – **a process and a thread**. A process is a program being executed. A process can be further divided into independent units known as threads.

A thread is like a small light-weight process within a process. Or we can say a collection of threads is what is known as a process.

Applications –
Threading is used widely in almost every field. Most widely it is seen over the internet now days where we are using transaction processing of every type like recharges, online transfer, banking etc. Threading is a segment which divide the code into small parts that are of very light weight and has less burden on CPU memory so that it can be easily worked out and can achieve goal in desired field. The concept of threading is designed due to the problem of fast and regular changes in technology and less the work in different areas due to less application. Then as says "need is the generation of creation or innovation" hence by following this approach human mind develop the concept of thread to enhance the capability of programming.

**We can find this problem by using Semiphores as a Synchronization** Mechanism.

## Semaphores in Process Synchronization

Prerequisite: process-synchronization, Mutex vs Semaphore

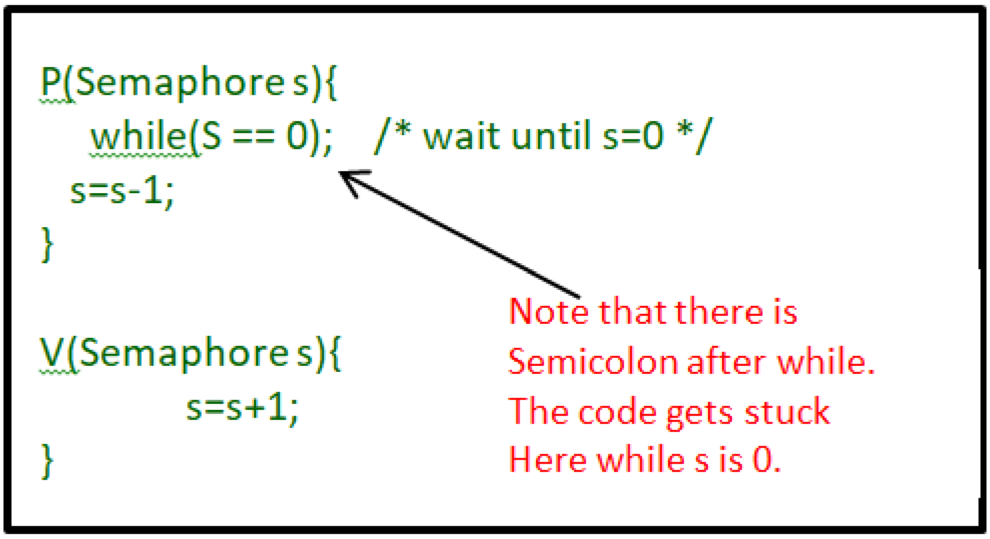
Semaphore was proposed by Dijkstra in 1965 which is a very significant technique to manage concurrent processes by using a simple integer value, which is known as a semaphore. Semaphore is simply a variable which is non-negative and shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.

Semaphores are of two types:

1. Binary Semaphore – This is also known as mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problem with multiple processes.
2. Counting Semaphore – Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

Now let us see how it do so.

First, look at two operations which can be used to access and change the value of the semaphore variable.

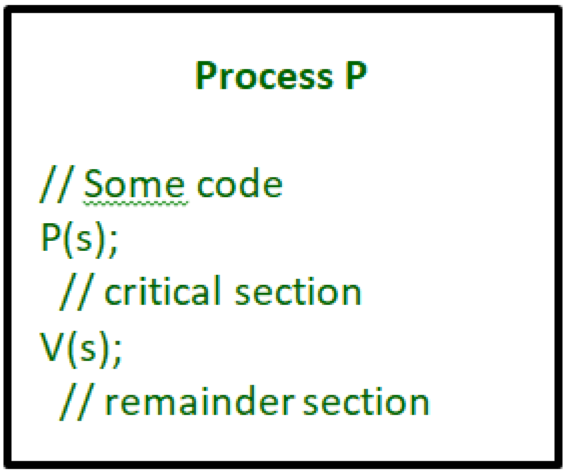


```
P(Semaphore s){
    while(S == 0);   /* wait until s=0 */
   s=s-1;
}

V(Semaphore s){
            s=s+1;
}
```

Note that there is
Semicolon after while.
The code gets stuck
Here while s is 0.

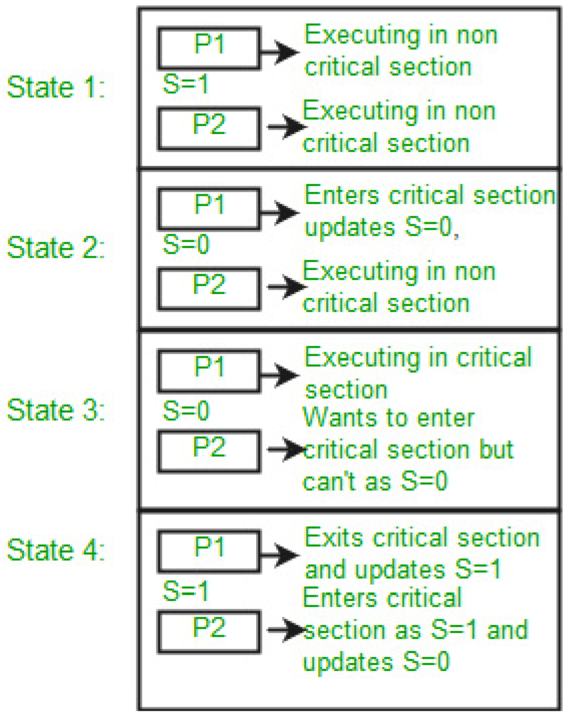
## Some point regarding P and V operation:

1. P operation is also called wait, sleep or down operation and V operation is also called signal, wake-up or up operation.
2. Both operations are atomic and semaphore(s) is always initialized to one.Here atomic means that variable on which read, modify and update happens at the same time/moment with no pre-emption i.e. in between read, modify and update no other operation is performed that may change the variable.
3. A critical section is surrounded by both operations to implement process synchronization.See below image.critical section of Process P is in between P and V operation.



```
Process P

// Some code
P(s);
  // critical section
V(s);
  // remainder section
```

Now, let us see how it implements mutual exclusion. Let there be two processes P1 and P2 and a semaphore s is initialized as 1. Now if suppose P1 enters in its critical section then the value of semaphore s becomes 0. Now if P2 wants to enter its critical section then it will wait until s > 0, this can only happen when P1 finishes its critical section and calls V operation on semaphore s. This way mutual exclusion is achieved. Look at the below image for details which is Binary semaphore.


State 1:
P1
Executing in non critical section
S=1
P2
Executing in non critical section
State 2:
P1
Enters critical section updates S=0,
S=0
P2
Executing in non critical section
State 3:
P1
Executing in critical section
S=0
P2
Wants to enter critical section but can't as S=0
State 4:
P1
Exits critical section and updates S=1
S=1
P2
Enters critical section as S=1 and updates S=0

The description above is for binary semaphore which can take only two values 0 and 1 and ensure the mutual exclusion. There is one other type of semaphore called counting semaphore which can take values greater than one.

Now suppose there is a resource whose number of instance is 4. Now we initialize S = 4 and rest is same as for binary semaphore. Whenever process wants that resource it calls P or wait function and when it is done it calls V or signal function. If the value of S becomes zero then a process has to wait until S becomes positive. For example, Suppose there are 4 process P1, P2, P3, P4 and they all call wait operation on S(initialized with 4). If another process P5 wants the resource then it should wait until one of the four processes calls signal function and value of semaphore becomes positive.

Limitations:

One of the biggest limitation of semaphore is priority inversion.

Deadlock, suppose a process is trying to wake up another process which is not in sleep state.Therefore a deadlock may block indefinitely.

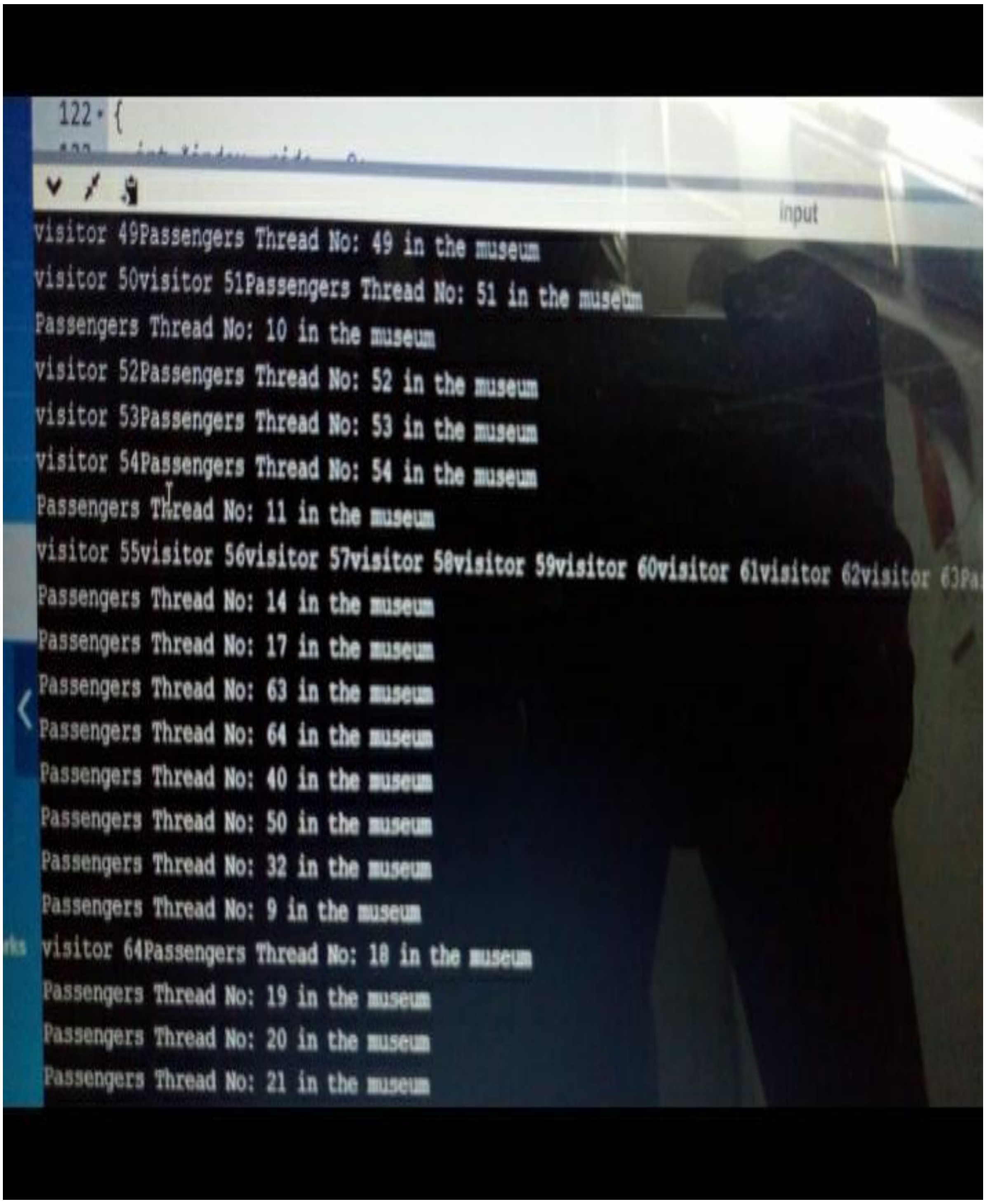
The operating system has to keep track of all calls to wait and to signal the semaphore.Problem in this implementation of semaphoreWhenever any process waits then it continuously checks for semaphore value (look at this line while (s==0); in P operation) and waste CPU cycle.

CODE: #include <pthread.h>

```
#include <semaphore.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
sem_t sem_visitor, sem_car;
//Variable to terminate the program after
serving no of passengers after a given
amount of time
int CYCLES;

int main (){
  printf("\t\t------------Operating
System Project------------\n\n");
  printf("\t\tJurrassic Park Problem
Using Semaphores\n\n");
  int VISITORS;
  int CARS;
 // int CYCLES;

 //Input Validation

  printf("Enter the No of
passengers:\n");
  scanf("%d",&VISITORS);
  if(VISITORS<0){
    printf("Enter Positive no passengers
only\n Try again\n");
    exit(0);
  }
  printf("Enter the No of cars:\n");
  scanf("%d",&CARS);
  if(CARS<0){
    printf("Enter Positive no cars only\n
Try again\n");
    exit(0);
  }
  printf("Enter the No of cycles:\n");
  scanf("%d",&CYCLES);
  if(CYCLES<0){
    printf("Enter Positive no cycles
only\n Try again\n");
    exit(0);
  }

  pthread_t visi[VISITORS], car[CARS];
  void *Car (void *), *Visitor (void *);
  int i, v[VISITORS];
  extern sem_t sem_car, sem_visitor;
 //Initlializing the sem_car unnamed
semaphore with pshared value=0
  sem_init (&sem_car, 0, 0);
 //Initlializing the sem_visitor unnamed
semaphore with pshared value=0
  sem_init (&sem_visitor, 0, 0);

  srand (time (NULL));
 //Threads creation for passengers
  for (i=0; i<VISITORS; i++)
  {
    v[i] = i;
printf("visitor %d",i) ;
    if (pthread_create (&visi[i], NULL,
Visitor, &v[i]))
    {
    printf("Thread created Vis %d ",i);
    exit (1);
    }
  }
 //Thread creation for cars
  for (i = 0; i < CARS; i++)
  {
      v[i] = i;
    printf("Car %d",i);
      if (pthread_create (&car[i], NULL,
Car, &v[i]))
      {
    printf("Car Thread created %d",i);
    exit (1);
    }
  }
   //Waiting for the passenger thread to
terminate
  for (i=0; i<VISITORS; i++)
  {
    printf("visitors waiting %d",i);
      pthread_join (visi[i], NULL);
    }

  printf("\n\t---- All Passengers are
done with all cycles completed---- \n");
}

//Passenger Processing
void *Visitor (void *p)
{
  extern sem_t sem_car, sem_visitor;
  int i, *index;

  index = (int *)p;
  for (i = 0; i < CYCLES; i++)
    {
      // Passengers Waiting at the
museum.
      fprintf (stdout, "Passengers Thread
No:% d in the museum\n", *index+1);
      sleep (rand()%4);

      // Waiting for a car
     //Unlocking the sem_car semaphore
      sem_post (&sem_car);

     //Locking the sem_visitor semaphore
      sem_wait (&sem_visitor);

      // Passengers in car
      fprintf (stdout, "Passengers Thread
No:% d is in the car\n", *index+1);
      sleep (rand()%3);

      /* Free the car. */
    }
    //For the program termination we are
using cycles for eg afer serving
passengers in 3 cycles the program will
terminate

    fprintf (stdout, "Passengers Thread
No:% d has finished his% d CYCLES.\n",
*index+1, CYCLES);
}

//Car Processing
void *Car (void *p)
```

```
{
  int *index, ride = 0;
  extern sem_t sem_car, sem_visitor;
  index = (int *) p;

  printf("cr %d  waiting for passengers
%d",sem_car,*index+1);
  /*The cars Waiting for Passengers*/
  while(1)
    {
      // Waiting untill the passengers
are available
      //Locking the sem_car variable
untill passengers are available
      sem_wait (&sem_car);

      //Take Safari
      sleep (rand()%3);

      //Go back to museum and turn down
the passenger
      //Unlocking the sem_visitor
semaphore
      sem_post (&sem_visitor);

    }
printf("Car %d  finish with museum
%d",sem_car,*index+1);
}
```

**OUTPUT:**



```
visitor 49Passengers Thread No: 49 in the museum
visitor 50visitor 51Passengers Thread No: 51 in the museum
Passengers Thread No: 10 in the museum
visitor 52Passengers Thread No: 52 in the museum
visitor 53Passengers Thread No: 53 in the museum
visitor 54Passengers Thread No: 54 in the museum
Passengers Thread No: 11 in the museum
visitor 55visitor 56visitor 57visitor 58visitor 59visitor 60visitor 61visitor 62visitor 63Pa
Passengers Thread No: 14 in the museum
Passengers Thread No: 17 in the museum
Passengers Thread No: 63 in the museum
Passengers Thread No: 64 in the museum
Passengers Thread No: 40 in the museum
Passengers Thread No: 50 in the museum
Passengers Thread No: 32 in the museum
Passengers Thread No: 9 in the museum
visitor 64Passengers Thread No: 18 in the museum
Passengers Thread No: 19 in the museum
Passengers Thread No: 20 in the museum
Passengers Thread No: 21 in the museum
```