****

**OPERATING SYSTEM PROJECT ON**

**Semaphore and Critical Section**

Submitted in the partial fulfillment for the academic requirements

**lV th Semester B-tech**

**In**

**Computer Science Engineering (CSE)**

**Submitted By: -**

**Name:** Sahil Sonu Chandel

**Reg .no:** 11813235

**Roll No:** 20

**Email:** sonuchandel900@gmail.com

**GitHub Link:**

**Guided By: Dr. Aarti Ma’am**

**Problem Statement**

Implementation of Semaphore and Critical Section problem.

**INTRODUCTION**

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.

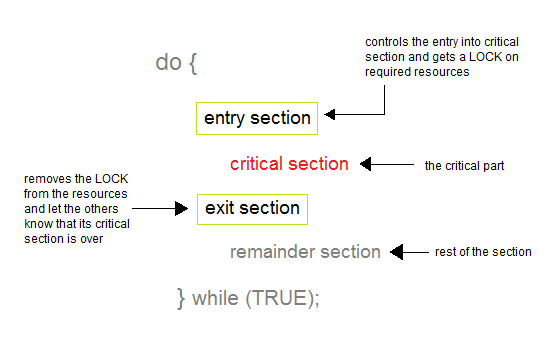
Semaphore is simply a variable that is non-negative and shared between threads. A semaphore is a signalling mechanism, and a thread that is waiting on a semaphore can be signalled by another thread. It uses two atomic operations, 1) wait, and 2) signal for the process synchronization

**Process Synchronization**

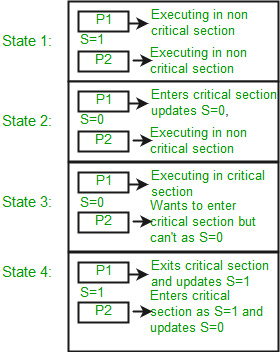
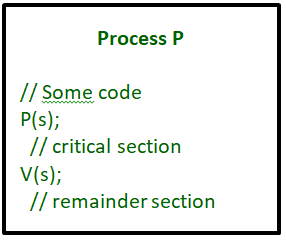
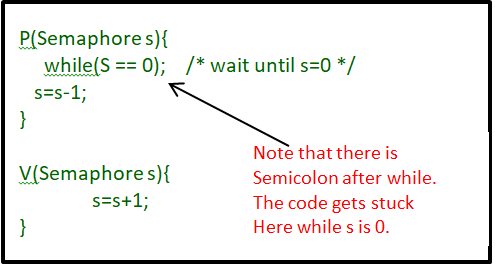
Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process executions.

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



**Algorithm**

****

**Program**

#include <stdio.h>

#include <semaphore.h>

#include <string.h>

sem\_t left;

sem\_t right;

sem\_t waitingForRight;

sem\_t waitingForLeft;

void LeaveShop (char s[])

{

 if (strcmp (s, "LEFT") == 0)

  {

   sem\_post (&left);

  }

 else if (strcmp (s, "RIGHT") == 0)

  {

   sem\_post (&right);

  }

  return;

}

void ShopForAWhile()

{

 //some statements

 return;

}

void EnterShop (char s[])

{

 if (strcmp (s, "LEFT") == 0)

  {

   sem\_wait (&waitingForRight);

   int i = 0;

   int j = 0;

   int k = 0;

   while ((i > 0) && (j == 1) && (k == 1))

{

 sem\_getvalue (&waitingForLeft, &i);

 sem\_getvalue (&left, &j);

 sem\_getvalue (&right, &k);

};

   sem\_wait (&left);

   sem\_post (&waitingForRight);

 ShopForAWhile ();

   LeaveShop ("LEFT");

  }

 else if (strcmp (s, "RIGHT") == 0)

  {

   sem\_wait (&waitingForLeft);

   int l = 0;

   int m = 0;

   int n = 0;

   while ((l > 0) && (m == 1) && (n == 1))

{

 sem\_getvalue (&waitingForRight, &l);

 sem\_getvalue (&left, &m);

 sem\_getvalue (&right, &n);

};

   sem\_wait (&right);

   sem\_post (&waitingForLeft);

   ShopForAWhile ();

   LeaveShop ("RIGHT");

  }

  return;

}

int main ()

{

 sem\_init (&left, 0, 0);

 sem\_init (&right, 0, 0);

 sem\_init (&waitingForRight, 0, 0);

 sem\_init (&waitingForLeft, 0, 0);

 return 0;

}

**Calculate complexity of implemented algorithm**

* Overall Time complexity: n
* Overall Space Complexity: 1

**Explain all the constraints given in the problem**

* one of the biggest constraints in 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

**If you have implemented any additional algorithm to support the solution, explain the need and usage of the same. Description:**

* No, I have not implemented any additional algorithm to support the solution.

**Explain the boundary conditions of the implemented code**

* Boundary condition for Enter the shop :2

Enter shop for all customer is in a pair, because we assume that all the customer should enter in a shop only in a pair if not then wait until you find a pair.

* Boundary condition for customer in the shop: pair (1)

There should one pair of customers inside the shop at the same time**.**

* Boundary condition for leave the shop :1

Customer can leave shop 1 at a time, but both customers must leave before the next pair can enter.

**Have you made minimum 5 revisions of solution on GitHub?**

* Yes

GitHub Link: