OSLAB\_PROJECT

THE BARBER SHOP

**TEAM MEMBERS:**

* **DEEKSHANT BANSAL(21JE0285)**
* **DEEPIKA TANUVI(21JE0293)**
* **DEBRAJ MAITY(21JE0283)**
* **DEV RADADIA(21JE0298)**

**Introduction:**

* **This project is based on a real life Barber Shop problem which is classic inter-process communication and synchronization problem between multiple processes/threads.**
* **This Problem can be solved by using the concept of semaphores or mutex locks. We are solving this problem using semaphores.**

**What is Semaphore?**

Semaphore *S* is an *integer* variable which apart from *initialization* is accessed only through two standard *atomic* operations: *wait ()* and *signal ().*

Wait () was originally termed as P (proberen: to test)

Signal () was originally termed as V (verhogen: to increment)

The operations wait () and signal () are atomic which means that if a process P is executing either wait() or signal() then no other process can preempt P until it finishes wait()/signal().

Definition of wait ()

|  |
| --- |
| wait(S) {  while S <= 0 ; //no-operation  S--;  } |

Definition of signal ()

signal(S) {

S++;

}

*Working of wait ()*

The initial value of the Semaphore variable is ‘1’. 1 means that the resource is free and the process can enter the critical section. If the value of S is ‘0’ this means that some other process is in its critical section and thus, the current process should wait.

Wait () is called before the critical section. When a process calls wait (), it checks the value of S. If the value is less than or equal to ‘0’, then the process performs no operations. Hence, the process gets stuck in the while loop and is not able to come out of the wait () function. So, it is not able to enter critical section. But if the value of S is ‘1’, then it comes out of the while loop, decrements the value of S to 0 and enters the critical section.

*Working of signal()*

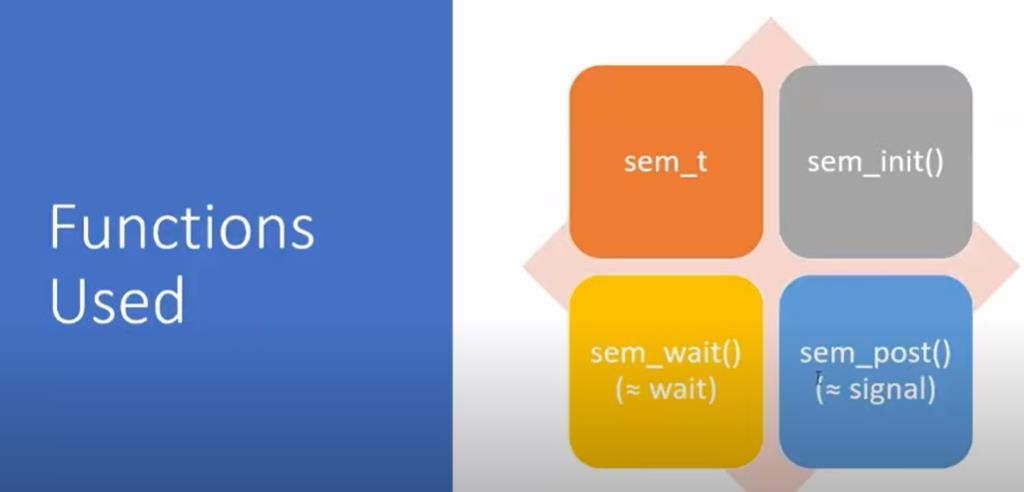
Once the process has finished the critical section part, it calls signal(). Within the signal() function, the process increments the value of S. Finally, giving a signal to a waiting process to enter in its critical section.

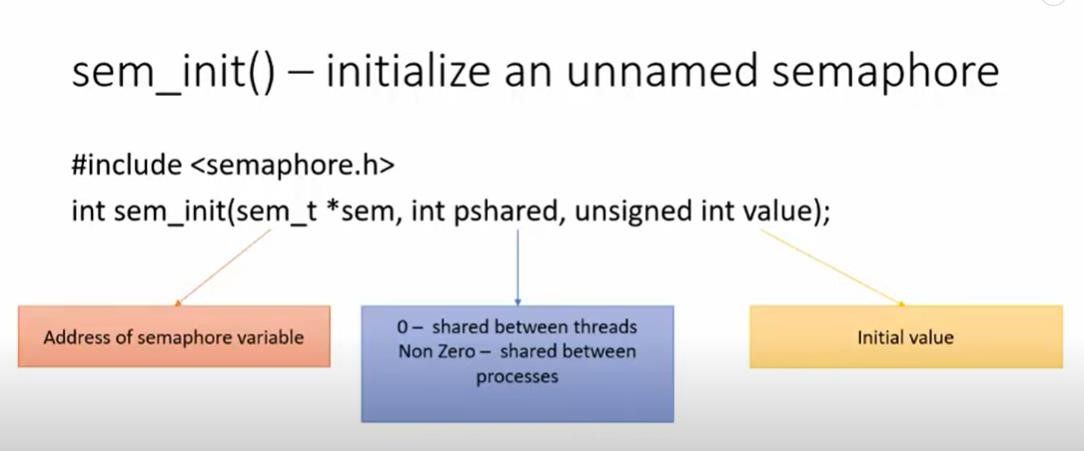
# Usage of Semaphore

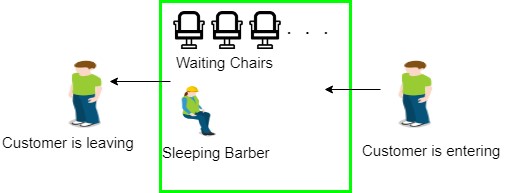
Semaphore are of two types: binary and counting. In Binary semaphore, the value of S can be 0 or 1 only. An alternate name for Binary semaphores is **mutex locks** because they provide mutual exclusion.

**Binary Semaphore** is used when there is only one instance of a resource. Hence the semaphore can have two value: 1 for free and 0 for busy.

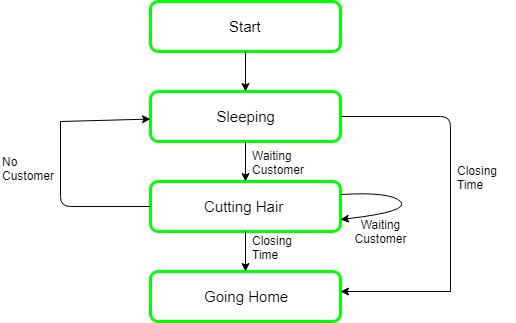
**Counting Semaphore** is used when there are multiple instances of the same resource. For example, there are 5 Printers. This means there are 5 instances of the resource Printer. Now 5 process can simultaneously print. So, the initial value of the semaphore variable should be 5 i.e., equal to the number of instances available.







**FLOW CHART:**



CODE-

#include <unistd.h>

#include <bits/stdc++.h>

#include <pthread.h>

#include <semaphore.h>

using namespace std;

// The maximum number of customer threads.

#define MAX\_Customers 20

// Function prototypes...

void \*customer (void \*num);

void \*barber (void \*);

void random\_wait (int secs);

// waitingRoom Limits the no of allowed customers

// to enter the waiting room at one time.

sem\_t waitingRoom;

// barChair ensures mutually exclusive access to

// the barber chair.

sem\_t barChair;

// barPillow is used to allow the barber to sleep

// until a customer arrives.

sem\_t barPillow;

// seatBelt is used to make the customer to wait until

// the barber is done styling the customer.

sem\_t seatBelt;

// Flag to stop the barber thread when all barber

// have been serviced.

int allDone = 0;

int Number[MAX\_Customers];

void change\_priority(){

    int new\_priority\_arr[MAX\_Customers];

    unordered\_set<int> used\_priorities;

    for (int j=0;j<MAX\_Customers;j++) {

        if (used\_priorities.count(Number[j]) == 0) {

            new\_priority\_arr[j]=(Number[j]);

            used\_priorities.insert(Number[j]);

        } else {

            int k = Number[j] + 1;

            while (used\_priorities.count(k) > 0) {

                k++;

            }

            new\_priority\_arr[j]=k;

            used\_priorities.insert(k);

        }

    }

    for(int j=0;j<MAX\_Customers;j++){

        Number[j]=new\_priority\_arr[j];

    }

}

int main (int argc, char \*argv[])

{

//argc is the number of arguments being passed into our program from the command line and argv is the

//array of arguments.

//pthread\_t is the data type used to uniquely identify a thread. It is returned by pthread\_create() and used

//by the application in function calls that require a thread identifier.

  pthread\_t dtid;

  pthread\_t tid[MAX\_Customers];

  int i, numCustomers = 0, numChairs = 0;

  int option;

e:cout <<

    ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>"

    << endl;

  cout << "' '" << endl;

  cout << "' WELCOME TO '" << endl;

  cout << "' BARBER'S SHOP '" << endl;

  cout << "' '" << endl;

  cout << "' '" << endl;

  cout << "' Press Your Option :- '" << endl;

  cout << "' 1.press for Info '" << endl;

  cout << "' 2.Enter number of Customers '" << endl;

  cout << "' 3.Enter number of Chairs '" << endl;

  cout << "' 4.Enter the priority of the customers: '" << endl;

  cout << "' '" << endl;

  cout << "' 5.Exit '" << endl;

  cout << "' Enter your choice \_\_ '" << endl;

  cout << "' '" << endl;

  cout <<

    ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>"

    << endl;

  cout << "choice : ";

  cin >> option;

  cout << endl;

  switch (option)

    {

    case 1:

      {

cout <<

 "This project is based on a real life Barber-Customer problem (in a barber shop)";

cout <<

 " which is classic inter-process communication and synchronization problem between multiple processes/threads.\n"

 << endl;

cout <<

 "This Problem can be solved by using the concept of semaphores or mutex locks.";

cout << " We are solving this problem using semaphores.\n";

goto e;

      }

    case 2:

      {

cin >> numCustomers;

for (i = 0; i < numCustomers; i++)

    {

      Number[i] = i;

    }

goto e;

      }

    case 3:

      {

cin >> numChairs;

goto e;

      }

    case 4:

      {

          for (i = 0; i < numCustomers; i++)

    {

      cin>>Number[i];

    }

change\_priority();

goto e;

      }

    case 5:

      {

cout << "Terminal closed\n";

break;

      }

    default:

      cout <<

"Option you entered is currently unvailable!!\nplease Enter option Valid option! : \n";

      goto e;

      break;

    }

  // Make sure the number of threads is less than the number of

  // Customers we can support.

  if (numCustomers > MAX\_Customers)

    {

      printf ("\nThe maximum number of Customers is %d.\n", MAX\_Customers);

      exit (-1);

    }

  // Initialize the numbers array.

//   for (i = 0; i < numCustomers; i++)

//     {

//       Number[i] = numCustomers-i-1;

//     }

  // Initialize the semaphores with initial values...

  sem\_init (&waitingRoom, 0, numChairs);

  sem\_init (&barChair, 0, 1);

  sem\_init (&barPillow, 0, 0);

  sem\_init (&seatBelt, 0, 0);

  // Create the barber.

  pthread\_create (&dtid, nullptr, barber, nullptr);

  // Create the Customers.

  for (i = 0; i < numCustomers; i++)

    {

      pthread\_create (&tid[i], nullptr, customer, (void \*) &Number[i]);

      sleep (1);

    }

  // Join each of the threads to wait for them to finish.

  for (i=0; i<numCustomers; i++) {

      pthread\_join(tid[i],nullptr);

      sleep(1);

  }

  // When all of the Customers are styled, kill the

  // barber thread.

  allDone = 1;

  sem\_post(&barPillow);

  // Wake the barber so he will exit.

  pthread\_join(dtid,nullptr);}

  void \*customer(void \*number) {

      int num = \*(int \*)number;

      // Leave for the shop and take some random amount of

      // time to arrive.

      printf("customer %d leaving for barber shop.\n", num);

      random\_wait(2);

      printf ("customer %d arrived at barber shop.\n", num);

      // Wait for space to open up in the waiting room...

      sem\_wait(&waitingRoom);

      printf("customer %d entering waiting room.\n", num);

      // Wait for the barber chair to become free.

      sem\_wait(&barChair);

      // The chair is free so give up your spot in the

      // waiting room.

      sem\_post(&waitingRoom);

      // Wake up the barber...

      printf("customer %d getting ready for the cut.\n", num);

      sem\_post(&barPillow);

      // Wait for the barber to finish cutting your hair.

      sem\_wait(&seatBelt);

      // Give up the chair.

      sem\_post(&barChair);

      printf("customer %d leaving barber's shop.\n", num);

  }

  void \*barber(void \*) {

      // While there are still Customers to be serviced...

      // Our barber can tell if there are

      // Customers still on the way to his shop

      while (!allDone) {

          // Sleep until someone arrives and wakes you..

          printf("The barber is calling the next customer\n");sem\_wait(&barPillow);

          //if all work completed

          if (!allDone) {

              // Take a random amount of time to style the customers

              cout<<"The barber is styling the customer\n\n";

              random\_wait(2);

              cout<<"The barber has finished styling.\n\n";

              // Release the customer when styling is finished...

              sem\_post(&seatBelt);

          }

              else {cout<<"No more customers left for the day!\n";

              cout<<"The barber is going home for the day.\n";

              }}}

  void random\_wait (int secs)

  {

    int len; // Generate a random number...

    len = (int) ((1 \* secs) + 1);

    sleep(len);

  }