OS LAB 4

NAME: Aditya Anand

ROLL NO.: 20124009

BRANCH: IT

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Title | Date of Implementation | Remarks |
| 1 | Program to handle the Critical Section Problem using semaphore | 09-02-2022 |  |

CRITICAL SECTION PROBLEM SOLUTION USING SEMAPHORE

The Critical Section is the part of the program where shared resources are accessed by multiple processes.

When two or more process try to access the critical section at the same time, errors such as race condition may arise. This is known as the **critical section problem**.

**Semaphore** is an integer variable which is used in mutually exclusive manner by various concurrent cooperative processes in order to achieve process synchronization.

CODE:

#include<bits/stdc++.h>

using namespace :: std;

queue<int> ready;

class semaphore{

    int value;

    public:

        queue<int> blocked\_list, CS;

        semaphore(){

            this->value = 9; // Last digit of my roll number.

        }

        void down(int p){         // Entry code before entering Critical Section

            this->value--;

            if(this->value<0){

                // Put the process in the blocked list

                blocked\_list.push(p);

                cout<<"Process "<<p<<" has been put in the blocked list\n";

            }

            else{

                // Put the process in the Critical Section

                CS.push(p);

                cout<<"Process "<<p<<" has entered the Critical Section\n";

            }

        }

        void up(){

            this->value++;

            if(!CS.empty()){

                int p=CS.front();

                cout<<"Process "<<p<<" has exited the Critical Section\n";

                CS.pop();

            }

            if(this->value<=0){

                // Wake up a sleeping process from the blocked list

                int p=blocked\_list.front();

                cout<<"Process "<<p<<" has been removed from the blocked list\n";

                blocked\_list.pop();

                // Put the blocked process back in the ready queue

                ready.push(p);

            }

        }

        void inCS(){

            if(CS.empty()){

                cout<<"The Critical Section is empty\n";

            }

            else{

                queue<int> temp = CS;

                cout<<"Processes present in Critical Section are: ";

                while(!temp.empty()){

                    cout<<temp.front()<<" ";

                    temp.pop();

                }

                cout<<"\n";

            }

        }

        void inBlockedList(){

            if(blocked\_list.empty()){

                cout<<"The Blocked List is empty\n";

            }

            else{

                queue<int> temp = blocked\_list;

                cout<<"Processes present in Blocked List are: ";

                while(!temp.empty()){

                    cout<<temp.front()<<" ";

                    temp.pop();

                }

                cout<<"\n";

            }

        }

};

int main(){

    cout<<"PROCESS SYNCHRONISATION USING SEMAPHORE C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;

    cout<<"Enter number of processes: ";

    cin>>n;

    for(int i=1; i<=n; i++){

        ready.push(i);

    }

    semaphore S;

    int itr=0;

    while(!ready.empty()){

        itr++;

        while(!ready.empty()){

            int process = ready.front();

            ready.pop();

            S.down(process);       // Each process tries to enter the critical section

            if(itr!=1){

                break;

            }

        }

        S.inCS();

        cout<<"\n";

        S.inBlockedList();

        cout<<"\n\n";

        while(!S.CS.empty()){

            S.up();

        }

    }

    return 0;

}

RESULT:

