**Assignment 5**

**Name : Yadnesh Patil**

**Div : C**

**Roll No : 23**

**Problems Statement:** Solve synchronization problems – Reader writer problem, Producer consumer problem & dinning philosopher problem using mutex & semaphore.

**Dinning Philosopher ProblemCode:**

*#include*<stdio.h>

*#include*<stdlib.h>

*#include*<pthread.h>

*#include*<semaphore.h>

*#include*<unistd.h>

sem\_troom;

sem\_tchopstick[5];

void\*philosopher(void\*);

voideat(int);

intmain()

{

    inti,a[5];

    pthread\_ttid[5];

    sem\_init(&room,0,4);

*for*(i=0;i<5;i++)

        sem\_init(&chopstick[i],0,1);

*for*(i=0;i<5;i++){

        a[i]=i;

        pthread\_create(&tid[i],NULL,philosopher,(void\*)&a[i]);

    }

*for*(i=0;i<5;i++)

        pthread\_join(tid[i],NULL);

}

void\*philosopher(void\*num)

{

    intphil=\*(int\*)num;

    sem\_wait(&room);

    printf("\nPhilosopher%d has entered room",phil);

    sem\_wait(&chopstick[phil]);

    sem\_wait(&chopstick[(phil+1)%5]);

    eat(phil);

    sleep(2);

    printf("\nPhilosopher%d has finished eating",phil);

    sem\_post(&chopstick[(phil+1)%5]);

    sem\_post(&chopstick[phil]);

    sem\_post(&room);

}

voideat(intphil)

{

    printf("\nPhilosopher%d is eating",phil);

}

**Output:**

**Graphical user interface, text

Description automatically generated**

**Producer Consumer Problem Code:**

*#include*<stdio.h>

*#include*<stdlib.h>

*// Initialize a mutex to 1*

int mutex =1;

*// Number of full slots as 0*

int full =0;

*// Number of empty slots as size*

*// of buffer*

int empty =10, x =0;

*// Function to produce an item and*

*// add it to the buffer*

voidproducer()

{

*// Decrease mutex value by 1*

    --mutex;

*// Increase the number of full*

*// slots by 1*

    ++full;

*// Decrease the number of empty*

*// slots by 1*

    --empty;

*// Item produced*

    x++;

    printf("\nProducer produces"

        "item %d",

        x);

*// Increase mutex value by 1*

    ++mutex;

}

*// Function to consume an item and*

*// remove it from buffer*

voidconsumer()

{

*// Decrease mutex value by 1*

    --mutex;

*// Decrease the number of full*

*// slots by 1*

    --full;

*// Increase the number of empty*

*// slots by 1*

    ++empty;

    printf("\nConsumer consumes "

        "item %d",

        x);

    x--;

*// Increase mutex value by 1*

    ++mutex;

}

*// Driver Code*

intmain()

{

    int n,i;

    printf("\n1. Press 1 for Producer"

        "\n2. Press 2 for Consumer"

        "\n3. Press 3 for Exit");

*#pragmaompcritical*

*for*(i=1;i>0;i++){

        printf("\nEnter your choice:");

        scanf("%d",&n);

*// Switch Cases*

*switch*(n){

*case*1:

*// If mutex is 1 and empty*

*// is non-zero, then it is*

*// possible to produce*

*if*((mutex ==1)

                &&(empty !=0)){

                producer();

            }

*// Otherwise, print buffer*

*// is full*

*else*{

                printf("Buffer is full!");

            }

*break*;

*case*2:

*// If mutex is 1 and full*

*// is non-zero, then it is*

*// possible to consume*

*if*((mutex ==1)

                &&(full !=0)){

                consumer();

            }

*// Otherwise, print Buffer*

*// is empty*

*else*{

                printf("Buffer is empty!");

            }

*break*;

*// Exit Condition*

*case*3:

            exit(0);

*break*;

        }

    }

}

**Output:**

**Text

Description automatically generated**

**Reader WriterProblem Code:**

*#include*<iostream>

*#include*<condition\_variable>

*#include*<thread>

*#include*<vector>

*#include*<random>

*#include*<mutex>

intreaders\_count=0;

int counter =0;

constint X =5;

std::mutex m;

std::condition\_variablereader\_cond;

std::condition\_variablewriter\_cond;

voidread(){

  std::mt19937rng;

  rng.seed(std::random\_device()());

  std::uniform\_int\_distribution<std::mt19937::result\_type>dist(0,20);

  std::unique\_lock<std::mutex>lk(m,std::defer\_lock);

*for*(inti=0;i< X;i++){

    std::this\_thread::sleep\_for(std::chrono::milliseconds(dist(rng)));

    lk.lock();

*if*(readers\_count==-1){

      reader\_cond.wait(lk,[](){*return*readers\_count!=-1;});

    }

    readers\_count++;

    lk.unlock();

    std::cout<<"read value: "<< counter <<", number of readers: "<<readers\_count<<std::endl;

    lk.lock();

    readers\_count--;

*if*(readers\_count==0){

      writer\_cond.notify\_all();

    }

    lk.unlock();

  }

}

voidwrite(){

  std::mt19937rng;

  rng.seed(std::random\_device()());

  std::uniform\_int\_distribution<std::mt19937::result\_type>dist(0,20);

  std::unique\_lock<std::mutex>lk(m,std::defer\_lock);

*for*(inti=0;i< X;i++){

    std::this\_thread::sleep\_for(std::chrono::milliseconds(dist(rng)));

    lk.lock();

*if*(readers\_count>0){

      writer\_cond.wait(lk,[](){*return*readers\_count==0;});

    }

    readers\_count=-1;

    lk.unlock();

    counter++;

    std::cout<<"written value: "<< counter <<", number of readers: "<<readers\_count<<std::endl;

    lk.lock();

    readers\_count=0;

    reader\_cond.notify\_all();

    writer\_cond.notify\_all();

    lk.unlock();

  }

}

intmain(){

  constint NUM\_READERS =5;

  constint NUM\_WRITERS =5;

  std::cout<<std::thread::hardware\_concurrency()<<std::endl;

  std::vector<std::thread> threads;

*for*(inti=0;i< NUM\_READERS;i++){

    threads.push\_back(std::thread{read});

  }

*for*(inti=0;i< NUM\_WRITERS;i++){

    threads.push\_back(std::thread{write});

  }

*for*(inti=0;i< NUM\_READERS + NUM\_WRITERS;i++){

    threads[i].join();

  }

*return*0;

}

**Output:**

**Text

Description automatically generated**

**Text

Description automatically generated**