**Ex. No: 8 Date: 16-10-2023**

**CLASSIC SYNCHRONIZATION USING SEMAPHORE**

**Aim:**

To simulate classic synchronization problems like Dining Philosophers and Producer Consumer in C language using Semaphore.

1. **Dining Philosopher’s Problem:**

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

typedef struct{

int position;

int count;

sem\_t \*forks;

sem\_t \*lock;

} params\_t;

void initialize\_semaphores(sem\_t \*lock, sem\_t \*forks, int num\_forks);

void run\_all\_threads(pthread\_t \*threads, sem\_t \*forks, sem\_t \*lock, int num\_philosophers);

void \*philosopher(void \*params);

void think(int position);

void eat(int position);

int main(int argc, char \*args[]){

int num\_philosophers = 5;

sem\_t lock;

sem\_t forks[num\_philosophers];

pthread\_t philosophers[num\_philosophers];

initialize\_semaphores(&lock, forks, num\_philosophers);

run\_all\_threads(philosophers, forks, &lock, num\_philosophers);

pthread\_exit(NULL);

}

void initialize\_semaphores(sem\_t \*lock, sem\_t \*forks, int num\_forks){

int i;

for (i = 0; i < num\_forks; i++){

sem\_init(&forks[i], 0, 1);

}

sem\_init(lock, 0, num\_forks - 1);

}

void run\_all\_threads(pthread\_t \*threads, sem\_t \*forks, sem\_t \*lock, int num\_philosophers){

int i;

for (i = 0; i < num\_philosophers; i++){

params\_t \*arg = malloc(sizeof(params\_t));

arg->position = i;

arg->count = num\_philosophers;

arg->lock = lock;

arg->forks = forks;

pthread\_create(&threads[i], NULL, philosopher, (void \*)arg);

}

}

void \*philosopher(void \*params){

int i;

params\_t self = \*(params\_t \*)params;

for (i = 0; i < 3; i++){

think(self.position);

sem\_wait(self.lock);

sem\_wait(&self.forks[self.position]);

sem\_wait(&self.forks[(self.position + 1) % self.count]);

eat(self.position);

sem\_post(&self.forks[self.position]);

sem\_post(&self.forks[(self.position + 1) % self.count]);

sem\_post(self.lock);

}

think(self.position);

pthread\_exit(NULL);

}

void think(int position){

printf("Philosopher %d thinking...\n", position);

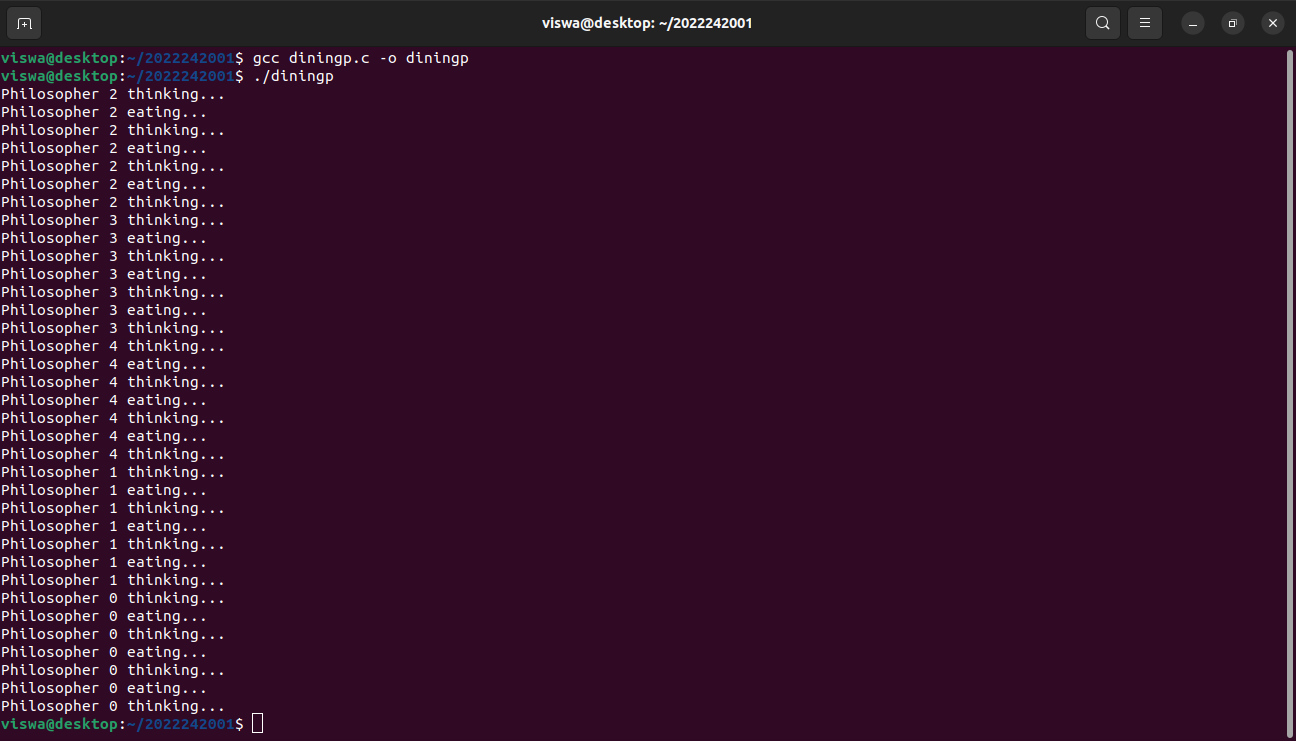
}

void eat(int position){

printf("Philosopher %d eating...\n", position);

}

**Output:**



1. **Producer Consumer Problem:**

**Program:**

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>

#define MaxItems 5

#define BufferSize 5

sem\_t empty;

sem\_t full;

int in = 0;

int out = 0;

int buffer[BufferSize];

pthread\_mutex\_t mutex;

void \*producer(void \*pno){

int item;

for (int i = 0; i < MaxItems; i++){

item = rand();

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[in] = item;

printf("Producer %d: Insert Item %d at %d\n", \*((int \*)pno), buffer[in], in);

in = (in + 1) % BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

}

void \*consumer(void \*cno){

for (int i = 0; i < MaxItems; i++){

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int item = buffer[out];

printf("Consumer %d: Remove Item %d from %d\n", \*((int \*)cno), item, out);

out = (out + 1) % BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

}

int main(){

pthread\_t pro[5], con[5];

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&empty, 0, BufferSize);

sem\_init(&full, 0, 0);

int a[5] = {1, 2, 3, 4, 5};

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

pthread\_create(&pro[i], NULL, (void \*)producer, (void \*)&a[i]);

}

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

pthread\_create(&con[i], NULL, (void \*)consumer, (void \*)&a[i]);

}

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

pthread\_join(pro[i], NULL);

}

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

pthread\_join(con[i], NULL);

}

pthread\_mutex\_destroy(&mutex);

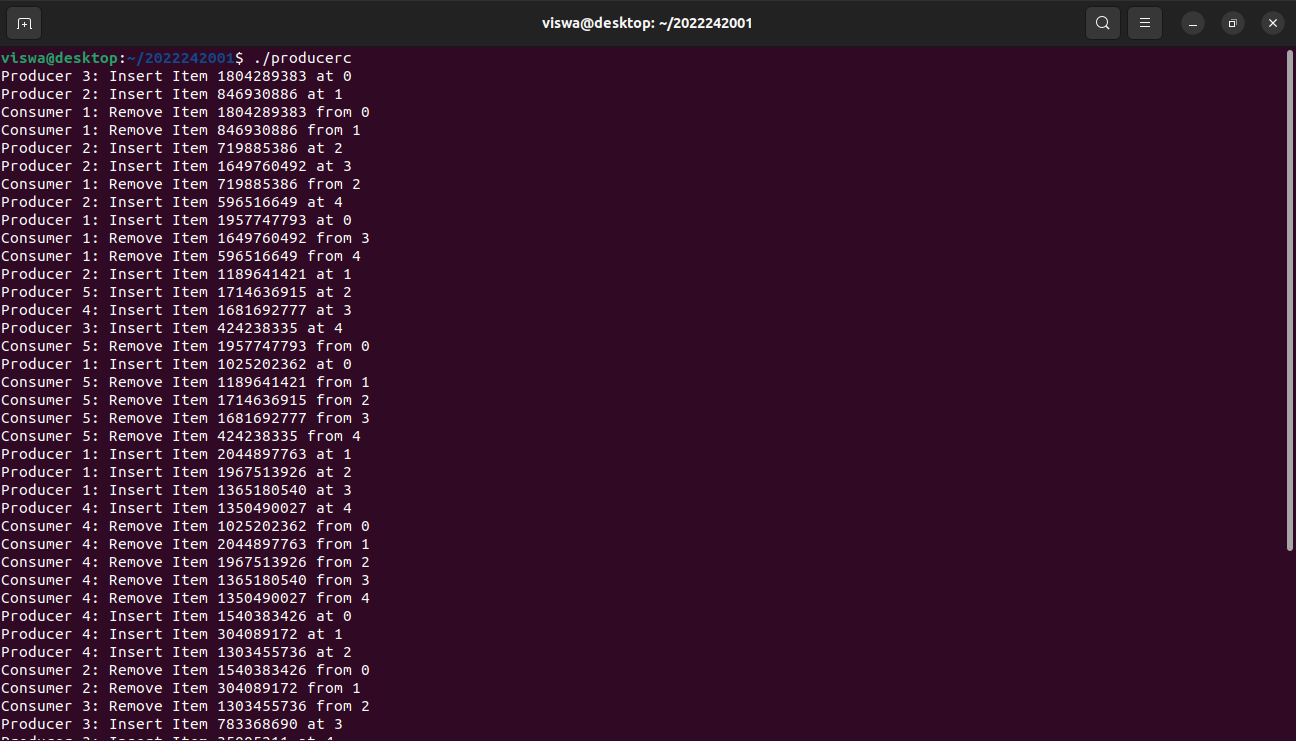
sem\_destroy(&empty);

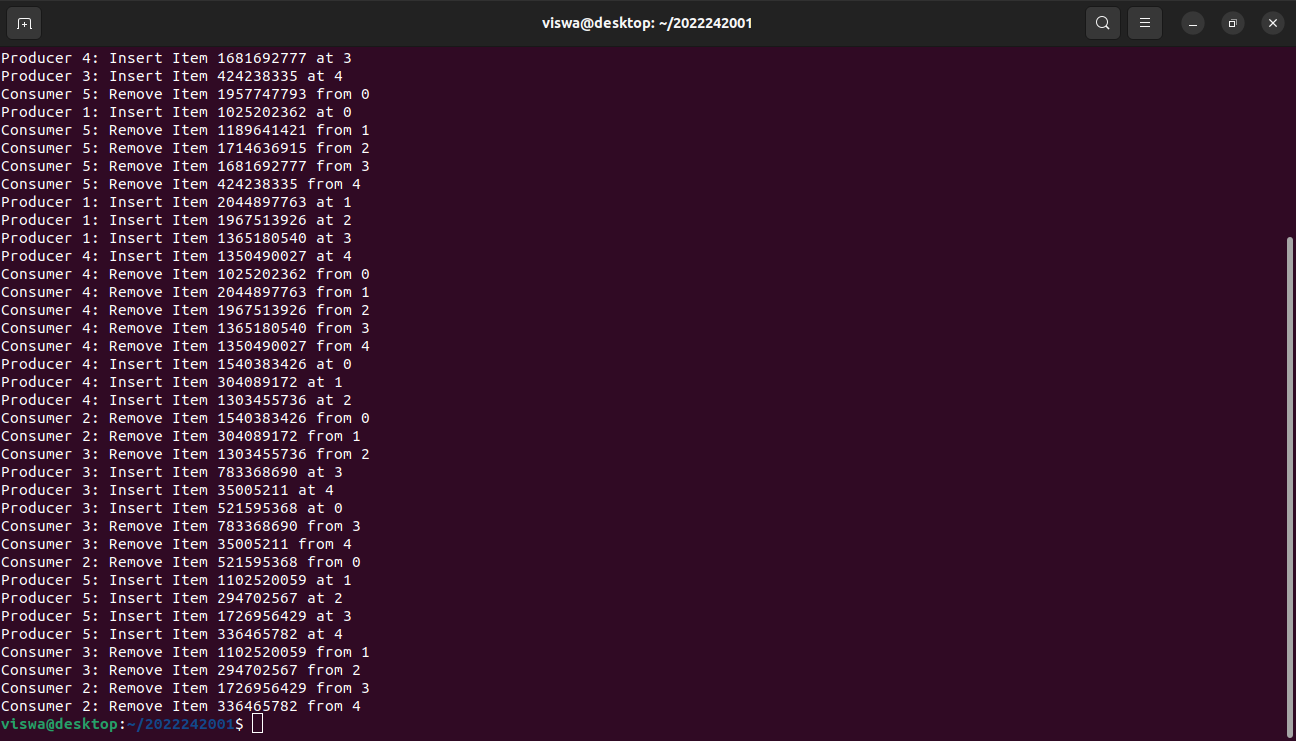
sem\_destroy(&full);

return 0;

}

**Output:**





**Result:**

Thus, the classic synchronization problems like Dining Philosophers and Producer

Consumers are successfully simulated in C Program using semaphores