**Name: Atharva Salitri Division: CSAI B**

**Roll No.: 37 PRN: 12310120**

**Subject: OS Lab Assignment 3**

**Title: Synchronisation Problems**

1. **Producer – Consumer:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define MaxItems 5

#define BufferSize 5

sem\_t empty;

sem\_t full;

sem\_t S;

int in = 0;

int out = 0;

int buffer[BufferSize];

void \*producer(void\* arg)

{

int item;

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

item = rand()%100;

sem\_wait(&empty);

sem\_wait(&S);

buffer[in] = item;

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

in = (in+1)%BufferSize;

sem\_post(&S);

sem\_post(&full);

}

}

void \*consumers(void\* arg)

{

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

sem\_wait(&full);

sem\_wait(&S);

int item = buffer[out];

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

out = (out+1)%BufferSize;

sem\_post(&S);

sem\_post(&empty);

}

}

int main()

{

pthread\_t prod, cons;

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

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

sem\_init(&S,0,1);

pthread\_create(&prod, NULL, (void \*)producer, NULL);

pthread\_create(&cons, NULL, (void \*)consumers, NULL);

pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

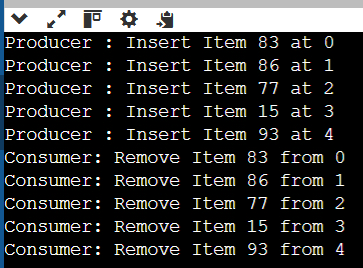
sem\_destroy(&empty);

sem\_destroy(&full);

sem\_destroy(&S);

return 0;

}



1. **Reader – Writer:**

#include <stdio.h>

#include <string.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <unistd.h>

#define BUFFER\_SIZE 16

int buffer[BUFFER\_SIZE];

sem\_t database, mutex;

int counter, readerCount;

pthread\_t readerThread[50], writerThread[50];

void init(){

sem\_init(&mutex, 0, 1);

sem\_init(&database, 0, 1);

counter = 0;

readerCount = 0;

}

void \*writer(void \*param){

sem\_wait(&database);

int item;

item = rand() % 5;

buffer[counter] = item;

printf("Data writen by the writer%d is %d\n", (\*(int \*)param),

buffer[counter]);

counter++;

sleep(1);

sem\_post(&database);

}

void \*reader(void \*param){

sem\_wait(&mutex);

readerCount++;

if (readerCount == 1){

sem\_wait(&database);

}

sem\_post(&mutex);

counter--;

printf("Data read by the reader%d is %d\n", (\*(int \*)param),buffer[counter]);

sleep(1);

sem\_wait(&mutex);

readerCount--;

if (readerCount == 0){

sem\_post(&database);

}

sem\_post(&mutex);

}

int main(){

init();

int no\_of\_writers, no\_of\_readers;

printf("Enter number of readers: ");

scanf("%d", &no\_of\_readers);

printf("Enter number of writers: ");

scanf("%d", &no\_of\_writers);

int i;

for (i = 0; i < no\_of\_writers; i++){

pthread\_create(&writerThread[i], NULL, writer, &i);

}

for (i = 0; i < no\_of\_readers; i++){

pthread\_create(&readerThread[i], NULL, reader, &i);

}

for (i = 0; i < no\_of\_writers; i++){

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

}

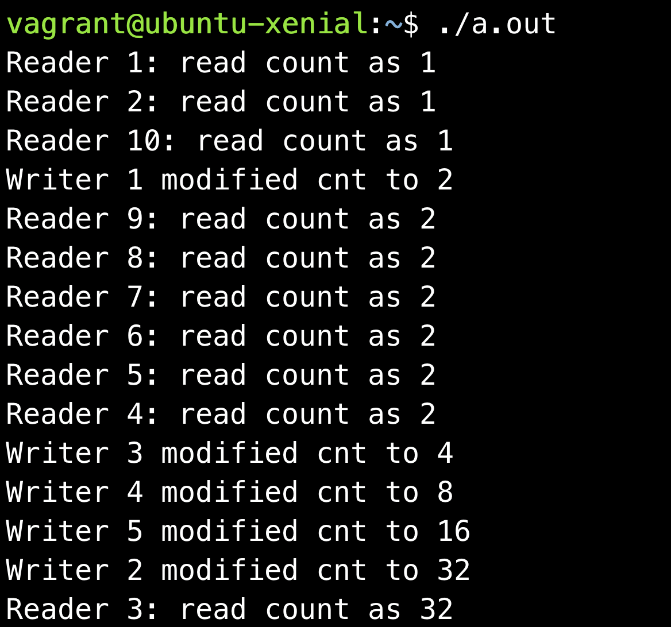
for (i = 0; i < no\_of\_readers; i++){

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

}

return 0;

}



1. **Dining – Philosopher:**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <zconf.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (Philosopher\_ID + 4) % N

#define RIGHT (Philosopher\_ID + 1) % N

int state[N];

int phil[N] = {0, 1, 2, 3, 4};

sem\_t mutex;

sem\_t S[N];

void Test\_Dining(int Philosopher\_ID) {

if (state[Philosopher\_ID] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING) {

// Eating State

state[Philosopher\_ID] = EATING;

sleep(2);

printf("Philosopher %d takes chopsticks %d and %d\n",

Philosopher\_ID + 1, LEFT + 1, Philosopher\_ID + 1);

printf("Philosopher %d is eating\n", Philosopher\_ID + 1);

sem\_post(&S[Philosopher\_ID]);

}

}

// Pick up the chopsticks

void Pickup\_Forks(int Philosopher\_ID) {

sem\_wait(&mutex);

// Hungry State

state[Philosopher\_ID] = HUNGRY;

printf("Philosopher %d is hungry\n", Philosopher\_ID + 1);

// Eat if adjacent neighbours are not eating

Test\_Dining(Philosopher\_ID);

sem\_post(&mutex);

// Wait to be signalled if unable to eat

sem\_wait(&S[Philosopher\_ID]);

sleep(1);

}

// Put down the chopsticks

void Return\_Forks(int Philosopher\_ID) {

sem\_wait(&mutex);

// Thinking State

state[Philosopher\_ID] = THINKING;

printf("Philosopher %d putting chopsticks %d and %d down\n",

Philosopher\_ID + 1, LEFT + 1, Philosopher\_ID + 1);

printf("Philosopher %d is thinking\n", Philosopher\_ID + 1);

Test\_Dining(LEFT);

Test\_Dining(RIGHT);

sem\_post(&mutex);

}

void \*philospher(void \*num) {

while (1) {

int \*i = num;

sleep(1);

Pickup\_Forks(\*i);

sleep(1);

Return\_Forks(\*i);

}

}

int main() {

int i;

pthread\_t thread\_id[N];

// Initialize semaphores

sem\_init(&mutex, 0, 1);

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

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

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

// Create philosopher threads

pthread\_create(&thread\_id[i], NULL,

philospher, &phil[i]);

printf("Philosopher %d is thinking\n", i + 1);

}

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

pthread\_join(thread\_id[i], NULL);

}

