**GARVIL JAIN 2022UCP1302 OSLAB**

**PRODUCER -CONSUMER PROBLEM**

#include <iostream>

#include <thread>

#include <semaphore.h>

#include <queue>

std::*queue*<*int*> buffer;

const *int* bufferSize = 5;

*sem\_t* emptySlots;

*sem\_t* fullSlots;

*sem\_t* mutex;

*void* producer(*int* *id*) {

*int* item = 0;

while (true) {

item++; // Produce

sem\_wait(&emptySlots); // Wait

sem\_wait(&mutex); // Lock

buffer.push(item);

std::cout << "Producer " << *id* << " produced item " << item << "\n";

sem\_post(&mutex); // Unlock

sem\_post(&fullSlots); // Signal

std::this\_thread::sleep\_for(std::chrono::*seconds*(1));

}

}

*void* consumer(*int* *id*) {

while (true) {

sem\_wait(&fullSlots); // Wait

sem\_wait(&mutex); // Lock

*int* item = buffer.front();

buffer.pop();

std::cout << "Consumer " << *id* << " consumed item " << item << "\n";

sem\_post(&mutex); // Unlock

sem\_post(&emptySlots); // Signal

std::this\_thread::sleep\_for(std::chrono::*seconds*(1));

}

}

*int* main() {

sem\_init(&emptySlots, 0, bufferSize);

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

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

std::*thread* producers[2], consumers[2];

for (*int* i = 0; i < 2; ++i) {

producers[i] = std::*thread*(producer, i+1);

consumers[i] = std::*thread*(consumer, i+1);

}

for (*int* i = 0; i < 2; ++i) {

producers[i].join();

consumers[i].join();

}

sem\_destroy(&emptySlots);

sem\_destroy(&fullSlots);

sem\_destroy(&mutex);

return 0;

}

**READERS-WRITERS problem**

#include <iostream>

#include <thread>

#include <semaphore.h>

using *namespace* std;

*sem\_t* readLock;

*sem\_t* writeLock;

*int* readCount = 0;

*void* reader(*int* *id*) {

while (true) {

sem\_wait(&readLock); // Lock

readCount++;

if (readCount == 1) {

sem\_wait(&writeLock);

}

sem\_post(&readLock); // Unlock

// Reading section

std::cout << "Reader " << *id* << " is reading\n";

std::this\_thread::sleep\_for(std::chrono::*seconds*(1));

sem\_wait(&readLock); // Lock

readCount--;

if (readCount == 0) {

sem\_post(&writeLock);

}

sem\_post(&readLock); // Unlock

std::this\_thread::sleep\_for(std::chrono::*seconds*(1));

}

}

*void* writer(*int* *id*) {

while (true) {

sem\_wait(&writeLock); // Lock

// Writing section

std::cout << "Writer " << *id* << " is writing\n";

std::this\_thread::sleep\_for(std::chrono::*seconds*(2));

sem\_post(&writeLock); // Unlock

std::this\_thread::sleep\_for(std::chrono::*seconds*(1));

}

}

*int* main() {

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

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

std::*thread* readers[3], writers[2];

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

readers[i] = std::*thread*(reader, i+1);

}

for (*int* i = 0; i < 2; ++i) {

writers[i] = std::*thread*(writer, i+1);

}

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

readers[i].join();

}

for (*int* i = 0; i < 2; ++i) {

writers[i].join();

}

sem\_destroy(&readLock);

sem\_destroy(&writeLock);

return 0;

}

**SLEEPING BARBER PROBLEM**

#include <iostream>

#include <thread>

#include <semaphore.h>

#include <queue>

#include <chrono>

*sem\_t* barberReady;

*sem\_t* accessSeats;

*sem\_t* customerReady;

*int* freeSeats = 3;

*void* barber() {

while (true) {

sem\_wait(&customerReady);

sem\_wait(&accessSeats);

freeSeats++;

sem\_post(&barberReady);

sem\_post(&accessSeats);

std::cout << "Barber is cutting hair\n";

std::this\_thread::sleep\_for(std::chrono::*seconds*(3));

}

}

*void* customer(*int* *id*) {

sem\_wait(&accessSeats);

if (freeSeats > 0) {

freeSeats--;

std::cout << "Customer " << *id* << " is waiting.\n";

sem\_post(&customerReady);

sem\_post(&accessSeats);

sem\_wait(&barberReady);

std::cout << "Customer " << *id* << " is getting a haircut.\n";

} else {

std::cout << "Customer " << *id* << " found no available seats and left.\n";

sem\_post(&accessSeats);

}

}

*int* main() {

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

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

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

std::*thread* barberThread(barber);

std::*thread* customers[5];

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

customers[i] = std::*thread*(customer, i+1);

std::this\_thread::sleep\_for(std::chrono::*seconds*(1));

}

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

customers[i].join();

}

barberThread.join();

sem\_destroy(&barberReady);

sem\_destroy(&accessSeats);

sem\_destroy(&customerReady);

return 0;

}