NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

(KARACHI CAMPUS)

FAST School of Computing Fall 2024

PROJECT REPORT

MEMBERS:

- Mishaal Bheraiya (22K-4254)
- Ayesha Ansari (22K-4453)

COURSE INSTRUCTOR:

- Miss Rabia Ansari
- Sir Monis

Project Topic:

Sleeping Barber Problem

1. Introduction

The sleeping barber problem is a classic synchronization problem involving a barber who sleeps when there are no customers and is woken up by a customer if he is asleep when the customer arrives. This problem is used to demonstrate process synchronization using semaphores.

This project implements a solution to the sleeping barber problem in the form of a Linux model. The implementation uses semaphores for managing access to the barber's resources, like the waiting room and the barber chair.

2. Tools and Environment Setup

Step 1: Installing Required Tools

Ensure you have the necessary tools installed on your Ubuntu system, primarily a C compiler like GCC and the pthread library. Run these commands on your terminal:

- sudo apt-get update
- sudo apt-get install build-essential

Step 2: Writing the Code

Create a new file named OSproject.c using a text editor like nano:

- nano OSproject.c

Step 3: Compile and Run the Code Compile:

- gcc OSproject.c -o project

Run:

- ./project

3. Code Overview

- Semaphores: Used to control access and synchronize the barber and customers.
- Customer and Barber Threads: Functions that simulate the behavior of customers and the barber.

4. Detailed Implementation

Semaphore Initialization: Four
semaphores are used:

- waitingRoom: Controls access to the waiting room.

- barberChair: Ensures exclusive access to the barber chair.
- barberPillow: Used by customers to wake the barber.
- seatBelt: Keeps the customer in the chair until the haircut is done.

Barber and Customer Threads

Customer Function:

- Arrive at the barber shop and try to enter the waiting room.
- If entry is successful, attempt to acquire the barber chair.
- Wake the barber if he is asleep and wait for the haircut to finish.
- Leave the barber shop.

Barber Function:

- Continuously check for customer presence.
- Sleep if no customers are in the waiting room.
- Perform the haircut and release the customer.

5. Compilation and Running Run the following commands:

- gcc OSproject.c -o project
- ./project

6. Code:

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <time.h>

#define MAX_CUSTOMERS 10

typedef struct Queue {
    int items[MAX_CUSTOMERS];
    int front, rear;
} Queue;

Queue waitingQueue;
pthread_mutex_t queueLock;        // Mutex for queue access
pthread_mutex_t barberStateLock; // Mutex for barber state

int isBarberAwake = 0; // 0 = asleep, 1 = awake

void initQueue(Queue *q) {
    q->front = -1;
    q->rear = -1;
}

int isEmpty(Queue *q) {
    return q->front == -1;
}
```

```
int isFull(Queue *q) {
    return q->rear == MAX_CUSTOMERS - 1;
}

void enqueue(Queue *q, int value) {
    if (isFull(q)) {
        printf("Queue is full! Cannot enqueue %d\n", value);
        return;
    }
    if (q->front == -1) {
        q->front = 0;
    }
    q->items[++q->rear] = value;
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        return -1;
    }
    int value = q->items[q->front];
    if (q->front >= q->rear) {
        q->front = q->rear = -1;
    } else {
        q->front++;
    }
    return value;
}

void printQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Waiting queue: ");
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d ", q->items[i]);
    }
    printf("\n");
}

// Semaphores
sem_t waitingRoom;
sem_t barberChair;
sem_t barberPillow;
sem_t seatBelt;

int flag = 0; // Flag to close the barber shop

void *customer(void *num) {
    int c = *(int *)num;
    printf("Customer %d leaving for barber shop.\n", c);
    sleep(1); // Simulate travel time
    printf("Customer %d reached at barber shop.\n", c);

    if (sem_trywait(&waitingRoom) == 0) { // Try to enter waiting room
        pthread_mutex_lock(&queueLock);
        enqueue(&waitingQueue, c);
        printf("Customer %d entered waiting room. ", c);
        printQueue(&waitingQueue);
        pthread_mutex_unlock(&queueLock);

        sem_wait(&barberChair); // Wait for barber chair
        pthread_mutex_lock(&barberStateLock);
        if (!isBarberAwake) {
            printf("\n\t\tCustomer %d waking the barber.\n", c);
            isBarberAwake = 1; // Mark barber as awake
            sem_post(&barberPillow); // Wake up the barber
        }
        pthread_mutex_unlock(&barberStateLock);
```

```
        sem_wait(&seatBelt);    // Wait until haircut is finished
        sem_post(&barberChair); // Leave the barber chair
        printf("Customer %d leaving barber shop.\n", c);
        sem_post(&waitingRoom); // Free up space in the waiting room
    } else {
        printf("Customer %d found no room in the waiting area and is leaving.\n", c);
    }
    return NULL;
}

void *barber(void *data) {
    while (!flag) {
        pthread_mutex_lock(&queueLock);
        if (isEmpty(&waitingQueue)) {
            pthread_mutex_unlock(&queueLock);
            pthread_mutex_lock(&barberStateLock);
            isBarberAwake = 0; // Mark barber as asleep
            pthread_mutex_unlock(&barberStateLock);

            printf("\n\t\tBarber is sleeping\n");
            sem_wait(&barberPillow); // Wait to be woken up
        } else {
            int nextCustomer = dequeue(&waitingQueue); // Serve the first customer
            pthread_mutex_unlock(&queueLock);

            printf("\t\tBarber is cutting hair for customer %d\n", nextCustomer);
            sleep(2); // Simulate haircut duration
            printf("\t\tBarber has finished cutting hair for customer %d.\n", nextCustomer);

            sem_post(&seatBelt); // Signal the haircut is done
        }
    }
    printf("Barber is closing shop and going home.\n");
    return NULL;
}

int main(void) {
    srand(time(NULL)); // Seed for random number generation

    pthread_t barber_id;
    pthread_t customer_id[MAX_CUSTOMERS];
    int numCustomers = (rand() % 6) + 5; // Random number between 5 and 10
    int numChairs = 4;    // Number of waiting room chairs
    int i;
    int cus[MAX_CUSTOMERS];

    sem_init(&waitingRoom, 0, numChairs);
    sem_init(&barberChair, 0, 1);
    sem_init(&barberPillow, 0, 0);
    sem_init(&seatBelt, 0, 0);

    pthread_mutex_init(&queueLock, NULL);        // Initialize queue lock
    pthread_mutex_init(&barberStateLock, NULL); // Initialize barber state lock
    initQueue(&waitingQueue);                    // Initialize the queue

    printf("Number of customers: %d\n", numCustomers);

    pthread_create(&barber_id, NULL, barber, NULL);
    for (i = 0; i < numCustomers; i++) {
        cus[i] = i + 1;
        pthread_create(&customer_id[i], NULL, customer, (void *)&cus[i]);
        sleep(1); // Stagger customer arrivals
    }

    for (i = 0; i < numCustomers; i++) {
        pthread_join(customer_id[i], NULL);
    }
```

```
    flag = 1; // Signal barber to close the shop
    sem_post(&barberPillow); // Wake up the barber if sleeping

    pthread_join(barber_id, NULL);

    sem_destroy(&waitingRoom);
    sem_destroy(&barberChair);
    sem_destroy(&barberPillow);
    sem_destroy(&seatBelt);
    pthread_mutex_destroy(&queueLock);
    pthread_mutex_destroy(&barberStateLock);

    return 0;
}
```

7. Output

```
Ayesha ayesha4453@LAPTOP-GQ40IDRT:~$ ./sleep5
Number of customers: 10

                Barber is sleeping
Customer 1 leaving for barber shop.
Customer 1 reached at barber shop.
Customer 1 entered waiting room. Waiting queue: 1

                Customer 1 waking the barber.
Customer 2 leaving for barber shop.
                Barber is cutting hair for customer 1
Customer 2 reached at barber shop.
Customer 2 entered waiting room. Waiting queue: 2
Customer 3 leaving for barber shop.
                Barber has finished cutting hair for customer 1.
                Barber is cutting hair for customer 2
Customer 3 reached at barber shop.
Customer 3 entered waiting room. Waiting queue: 3
Customer 4 leaving for barber shop.
Customer 1 leaving barber shop.
Customer 4 reached at barber shop.
Customer 4 entered waiting room. Waiting queue: 3 4
Customer 5 leaving for barber shop.
                Barber has finished cutting hair for customer 2.
                Barber is cutting hair for customer 3
Customer 2 leaving barber shop.
Customer 6 leaving for barber shop.
Customer 5 reached at barber shop.
Customer 5 entered waiting room. Waiting queue: 4 5
Customer 6 reached at barber shop.
Customer 6 entered waiting room. Waiting queue: 4 5 6
Customer 7 leaving for barber shop.
                Barber has finished cutting hair for customer 3.
                Barber is cutting hair for customer 4
Customer 3 leaving barber shop.
Customer 7 reached at barber shop.
Customer 7 entered waiting room. Waiting queue: 5 6 7
```

```
Customer 8 leaving for barber shop.
Customer 8 reached at barber shop.
Customer 8 found no room in the waiting area and is leaving.
Customer 9 leaving for barber shop.
                Barber has finished cutting hair for customer 4.
                Barber is cutting hair for customer 5
Customer 4 leaving barber shop.
Customer 9 reached at barber shop.
Customer 9 entered waiting room. Waiting queue: 6 7 9
Customer 10 leaving for barber shop.
Customer 10 reached at barber shop.
Customer 10 found no room in the waiting area and is leaving.
                Barber has finished cutting hair for customer 5.
                Barber is cutting hair for customer 6
Customer 5 leaving barber shop.
                Barber has finished cutting hair for customer 6.
                Barber is cutting hair for customer 7
Customer 6 leaving barber shop.
                Barber has finished cutting hair for customer 7.
                Barber is cutting hair for customer 9
Customer 7 leaving barber shop.
                Barber has finished cutting hair for customer 9.

                Barber is sleeping
Customer 9 leaving barber shop.
Barber is closing shop and going home.
```

8. Conclusion

This project effectively demonstrates the use of semaphores in the Linux environment to solve the sleeping barber problem, showcasing process synchronization and thread management.