A picture containing symbol, logo, emblem, font

Description automatically generated

**Operating System Project**

**Sleeping Barber Problem(System Call)**

**Group Members:**

**Aahil Ashiq Ali (K21-4549)**

**Khuzaima Ahsan (K21-3328)**

**Khubaib Lodhi (K21-4596)**

**Instructor name: Dr Ghufran Ahmed**

**Objective :**

The objective of the Sleeping Barber Problem is to model and simulate a scenario in which a limited number of resources (barber chairs) must be shared among a larger number of clients (customers who want a haircut). The problem arises when there are more customers than there are available barbers or chairs, leading to congestion and long wait times.

The problem is usually formulated as follows: In a barber shop, there is one barber and a certain number of chairs for customers to wait in. When the barber is idle, he sleeps in his chair until a customer wakes him up. If there are no customers waiting, the barber goes back to sleep. If a customer arrives and finds the barber sleeping, he wakes him up and waits in one of the chairs if they are available, or leaves if all the chairs are occupied. When the barber finishes cutting a customer's hair, he either goes back to sleep (if no other customers are waiting) or calls in the next customer waiting in the queue.

The objective is to implement a solution to this problem that ensures the correct and efficient use of resources while avoiding deadlocks, livelocks, and other synchronization problems. The solution should also be fair, meaning that all customers eventually get their haircuts in the order they arrived, without anyone waiting too long or getting skipped over.

Usage in Modern OS:

The Sleeping Barber Problem aims to demonstrate the challenges of managing shared resources in a multi-process system, particularly in the face of concurrency and race conditions. It is a classic synchronization problem in computer science that can be used to illustrate various concepts such as mutual exclusion, critical section, deadlock, starvation, and fairness.

The motivation behind this project is to understand and solve this problem to compreehensively understand the synchronizations used in race condition.

The biggest problem in this project was that only one of barber or customer should be allowed to change positions. For that we have used mutex locks.

First challenge in this project was to do it with system call. Using linux libraries was a bit tricky and it took forever to compile. Plus, we wanted the barber function to run indefinitely and the customers to come so we conciously have made an indefinite loop in this project because it is mant this way. So in order to end the program, you have to kill it.

Platform:

The Sleeping Barber problem is implemented through the below illustrated system,

Operating System:

Linux (Ubuntu 16.04)

Locks:

Semaphore Locks

Threads:

POSIX threads

Languages:

The language used in this project is solely (C).

Methodology :

Rules:

If there are no customers, the barber falls asleep in the chair

A customer must wake the barber if he is asleep

If a customer arrives while the barber is working, the customer leaves if all chairs are occupied and sits in an empty chair if it's available

When the barber finishes a haircut, he inspects the waiting room to see if there are any waiting customers and falls asleep if there are none.

Only customer or barber can change position at a time. For this we will use mutex (which will freeze the state of one process if the other process is changing state). Plus, semaphores will indicate state of system (using wait and signal).

Methadology:

One mutex is used by both barber and customer. Max people waiting is 5.

First, in the barber function, mutex is locked so only barber is allowed to change position. Then it checks if any customer is waiting. If not, it goes to sleep of 100 milliseconds. If he finds a customer waiting, then he cuts his hair(sleep of 500 milliseconds). The people waiting would be decremented.

Second, in the customer function, mutex is locked so customer can check if he has space to wait. If there is space, then customers waiting is incremented, else the person has to leave with no effect on variable.

Then in asm\_linkage functin, it acts like a main function. Mutex is initialized. Then threads are created and started executing. The threads donot stop themselves until kernel decides to do so if it wants to unload module or process is terminated.

**Results:**

Below are the attached snap shot of our project





