Sleeping Barber Problem

Problem statement and find that solution.

Source: Wikipedia

Problem statement:

The multiple-sleeping barber problem is a classical synchronization problem that models a scenario where multiple barbers work at a barbershop with limited chairs for waiting for customers. The problem is as follows:

There is a barbershop with M chairs for waiting for customers and N barbers who cut hair. When a customer enters the barbershop, if there is an available barber, the customer sits in the barber's chair and gets a haircut. If there is no available barber, the customer sits in one of the M chairs to wait for a barber to become available. If all chairs are occupied when a new customer arrives, the customer leaves and does not get a haircut.

Solution:

There are several possible solutions, but all solutions require a mutex, which ensures that only one of the participants can change state at once. Implement the answer like that the barber must acquire the room status mutex before checking for customers and release it when they begin to sleep or cut hair. Customers must acquire it before entering the shop and release it when they sit in a waiting room or barber chair and leave the shop because no seats are available. This would take care of both problems mentioned above.

This means when a barber thread wakes up, they will try to acquire the mutex and check if there is any customer waiting. If there is no customer waiting, they will release the mutex and go back to sleep. Otherwise, they will remove the customer from the queue, update the number of waiting for customers, and start cutting their hair. Once they are done, they will release the mutex and return to sleep.

How program run?

This multi-threaded program simulates a barbershop with multiple barbers and customers. The program uses the pthread library to create and manage threads.

The program defines three global variables: maxcustomer, maxbarber, and sizequeue. These variables specify the maximum number of customers, barbers, and the size of the waiting room.

The program defines mutex variables and two conditional variables. The mutex variable mtx, is used to protect shared resources from simultaneous access by multiple threads. The conditional variables cuscondition and barcondition are used to signal waiting threads when a condition is met.

The program defines two functions, barber and customer which represent the barber and customer threads

The barber function takes a pointer to an integer argument representing the barber ID. The function enters an infinite loop and locks the mutex mtx. It then waits on the conditional variable cuscondition while the waitroom queue is empty. When the queue is not empty, the barber pops the first customer ID from the waitroom queue, decrements the waitcustomer count, and releases the mutex.

The customer function takes a pointer to an integer argument representing the customer ID. The function locks the mutex mtx and checks if the waiting room is full, the function prints a message that the customer is leaving and exits the function. Otherwise, the function pushes the customer ID onto the waitroom queue, increments the waitcustomer count, and signals the cuscondition variable. The function then releases the mutex and exits.

In the main function, the program creates an array of maxbarber, barber threads, and maxcustomer customer threads. The program initializes the barber and customer ID arrays and creates each thread using the pthread_create function.