**SLEEPING BARBER PROBLEM**

**PROBLEM DEFINITION:**

The Sleeping Barber problem is a classic computer science synchronization issue which illustrates the complexities that arise when multiple threads (or processes) interact with shared resources. This Go program simulates a barber shop with the following constraints:

* A finite number of barbers.
* A finite number of seats in the waiting room.
* A fixed length of time the barbershop is open.
* Clients (customers) arriving at random intervals.

The barbers either work (cut hair) if there are waiting customers, or sleep if there are none. Customers will leave if the waiting room is full or wait if there's an available seat. The shop stops accepting new customers at closing time but the barbers continue working until all waiting customers are served.

**PROGRAM STRUCTURE:**

**Types**

* BarberShop: Represents the barber shop.
* waitingRoom: A channel to simulate waiting room seats.
* BarberReady: A channel to indicate when a barber is ready.
* mutex: Mutex to protect shared resource (the state of the shop).
* isClosed: Indicates whether the shop is closed.
* wg: WaitGroup to wait for all barbers to finish.

**Functions**

1. **NewBarberShop() \*BarberShop**: Constructor for BarberShop. Initializes channels, mutex, and WaitGroup.
2. **(\*BarberShop) openShop()**: Begins the barber shop operation. Starts barber goroutines and simulates shop working hours. Initiates the close shop process after the working hours are over.
3. **(\*BarberShop) barber(id int)**: Goroutine function for barbers. Barbers wait for customers or sleep if there are no customers. They stop working when the shop closes, and all customers in the waiting room are served.
4. **(\*BarberShop) customer(id int)**: Goroutine function for customers. Customers enter the waiting room if there's an available seat; otherwise, they leave. They cannot enter if the shop is closed.
5. **(\*BarberShop) closeShop()**: Closes the barber shop. It sets **isClosed** to **true**, waits for all remaining customers to be served, and ensures all barber goroutines have finished before closing the waiting room channel.
6. **main()**: The entry point of the program. It initializes the barber shop, simulates the arrival of customers at random intervals, and ensures the shop opens and closes appropriately.

**Random Number Generation**

* **rand.Seed(time.Now().UnixNano())**: Initializes the random number generator to ensure different behavior (like customer arrival intervals and haircut durations) in each program run.

**Concurrency Control**

Mutex and channels are used to manage access to shared resources and synchronize between goroutines.

The sync.WaitGroup is crucial for waiting for all barber goroutines to finish before the program exits, ensuring no customer is left unattended.

**PROGRAM OUTPUT:**

**PS E:\Aravind\Companies\Altimetrik> go run .\SleepingBarber.go**

Customer 1 is waiting in the waiting room

Barber 0 is cutting hair of customer 1

Customer 2 is waiting in the waiting room

Barber 1 is cutting hair of customer 2

Barber 0 finished with customer 1

Barber 1 finished with customer 2

Customer 3 is waiting in the waiting room

Barber 0 is cutting hair of customer 3

Barber 0 finished with customer 3

Customer 4 is waiting in the waiting room

Barber 1 is cutting hair of customer 4

Barber 1 finished with customer 4

Customer 5 is waiting in the waiting room

Customer 6 is waiting in the waiting room

Customer 7 is waiting in the waiting room

Customer 8 is waiting in the waiting room

Customer 9 is waiting in the waiting room

Customer 10 found no empty chairs and is leaving

Customer 11 found no empty chairs and is leaving

Customer 12 found no empty chairs and is leaving

Customer 13 found no empty chairs and is leaving

Customer 14 found no empty chairs and is leaving

Customer 15 found no empty chairs and is leaving

Customer 16 found no empty chairs and is leaving

Customer 17 found no empty chairs and is leaving

Barber 1 is cutting hair of customer 5

Barber 0 is cutting hair of customer 6

Barber 1 finished with customer 5

Barber 1 is cutting hair of customer 7

Barber 1 finished with customer 7

Barber 1 is cutting hair of customer 8

Customer 18 found the shop closed and is leaving

Barber 0 finished with customer 6

Barber 0 is cutting hair of customer 9

Barber 0 finished with customer 9

Barber 1 finished with customer 8

**CONCLUSION:**

The sleeping barber problem is solved in go by simulating number of customer and time taken to cut the hair and goroutines and different synchronization primitives such as channels, mutex and waitgroup.