**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Topic** | **Page No.** |
| 01 | Introduction | 1-2 |
| 02 | Theoretical Background | 3-4 |
| 03 | Screenshots | 5 |
| 04 | Code | 6-7 |
| 05 | Resources Used | 8 |
| 06 | Hardware/Software Requirements | 9 |

**Introduction**

In computer science, the Railway Ticket Counter problem is a classic inter-process communication and synchronization problem between multiple operating system processes. The problem is analogous to that of keeping a ticket distributing staff working when there are customers, resting when there are none, and doing so in an orderly manner.

The analogy is based upon a hypothetical Ticket Booking Counter with one ticket distributing employee. Each employee has one counter and a waiting room with a number of chairs in it. When the employee finishes providing tickets, he dismisses the customer and then waits to see if there are other customers are waiting. If there are, he handle’s there queries. If there are no other customers waiting, he returns to his chair and sleeps in it.

Each customer, when he arrives, looks to see what the employee is doing. If he is sleeping, then the customer wakes him up and sits in the chair. If the employee is busy, then the customer goes to the waiting room. If there is a free chair in the waiting room, the customer sits in it and waits his turn. If there is no free chair, then the customer leaves.

Based on a naïve analysis, the above description should ensure that the counter functions correctly, with the tickets getting booked of anyone who arrives until there are no more customers, and then sleeping until the next customer arrives. In practice, there are a number of problems that can occur that are illustrative of general scheduling problems.

**Theoretical Background**

**Problems to be faced:**

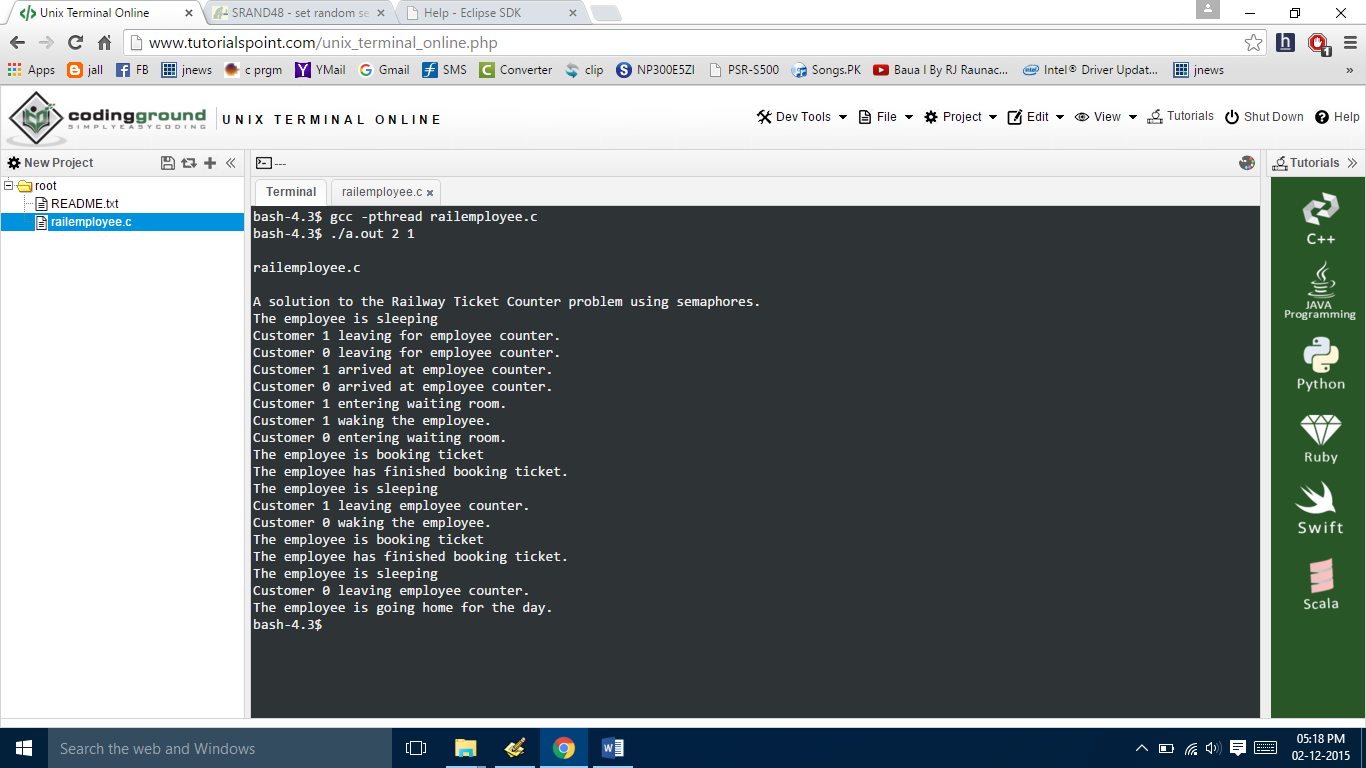
The problems are all related to the fact that the actions by both the employee and the customer (checking the waiting room, entering the counter, taking a waiting room chair, etc.) all take an unknown amount of time. For example, a customer may arrive and observe that employee is busy, so he goes to the waiting room. While he is on his way, a employee finishes the job he is doing and goes to check the waiting room. Since there is no one there (the customer not having arrived yet), he goes back to his chair and sleeps. The employee is now waiting for a customer and the customer is waiting for the employee. In another example, two customers may arrive at the same time when there happens to be a single seat in the waiting room. They observe that the employee is booking ticket, go to the waiting room, and both attempt to occupy the single chair.

**Possible Solution:**

Many possible solutions are available. The key element of each is a mutex, which ensures that only one of the participants can change state at once. The employee must acquire this mutual exclusion before checking for customers and release it when he begins either to sleep or books tickets. A customer must acquire it before entering the counter and release it once he is sitting in either a waiting room chair or the employee chair, and also when he leaves the counter because no seats were available. This eliminates both of the problems mentioned in the previous section. A number of semaphores is also required to indicate the state of the system. For example, one might store the number of people in the waiting room.

A multiple Railway Ticket Counters’ problem has the additional complexity of coordinating several employees among the waiting customers.

**Screenshots:**



**Resources Used**

The various resources used are:

1. **Kernel of KUbuntu**
2. **GCC Compiler**
3. **G++ Compiler**
4. **Text Editor**

**Hardware Requirements**

**Hardware Specification (Minimum)**:

**Disc Space:** 5 GB

**Processor:**  Pentium 3

**Memory:** 512 MB RAM

**File System:** 32/64 Bit

**Software Specification:**

**Operating System (Client Side):** KUbuntu.

**Language:** C, C++