# OPERATING SYSTEMS

## EXPERIMENT – 3

**Ayush Jain| 60004200132 | SE – B2**

**AIM-** Building multi-threaded and multi-process applications.

**PROBLEM STATEMENT:**

1. Build an instance of bus ticket reservation system using multithreading for the following scenario.
   ABC Bus service has only two seats left for reservations. Two users are trying to book the ticket at the same time.
2. Create separate thread per user. Show how this code is leading to inconsistency.
3. Improve the code by applying synchronization.
4. Conclude the experiment by stating the importance of synchronization in multiprocess and multithread application.

**THEORY:**

- **Multi-threading:** Multi-threading is a process of executing multiple threads simultaneously. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.
- **Runnable:** Java runnable is an interface used to execute code on a concurrent thread. It is an interface which is implemented by any class if we want that the instances of that class should be executed by a thread. The runnable interface has an undefined method run () with void as return type, and it takes in no arguments.
- **Thread:** Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface. Commonly used Constructors of Thread class: Thread (), Thread (String name), Thread (Runnable r), Thread (Runnable r, String name).

Thread class

**CODE:**

- **Using multi-threading (No Synchronization):**

```
class Main{

    public static void main(String[] args){ Reservation

        reserve = new Reservation(); Person thread1 = new

        Person(reserve, 2);

        thread1.start();

        Person thread2 = new Person(reserve, 2);

        thread2.start();

    }
}

class Reservation{

    static int availableSeats = 2; void

    reserveSeat(int requestedSeats) {

        System.out.println(Thread.currentThread().getName() + " entered.");

        System.out.println("Availableseats : " + availableSeats + "\n
Requestedseats : " + requestedSeats);

        if (availableSeats >= requestedSeats){ System.out.println("Seat

            Available. Reserve now :-)"); try{

                Thread.sleep(100);


            }

            catch (InterruptedException e){

                System.out.println("Thread interrupted");

            }
```

```
            System.out.println(requestedSeats + " seats reserved.");
            availableSeats = availableSeats - requestedSeats;
        }
        else{
            System.out.println("Requested seats not available :-(");
        }
        System.out.println(Thread.currentThread().getName() + " leaving.");
    }
}
class Person extends Thread{
    Reservation reserve;
    int requestedSeats;
    public Person(Reservation reserve, int requestedSeats){ this.reserve
        = reserve;
        this.requestedSeats = requestedSeats;
    }
    public void run() { reserve.reserveSeat(requestedSeats);
    }
}
```

```
Thread-1entered.
Thread-0entered.
Availableseats : 2
 Requestedseats : 2
Seat Available. Reserve now :-)
Availableseats : 2
 Requestedseats : 2
Seat Available. Reserve now :-)
2 seats reserved.
2 seats reserved.
Thread-0 leaving.
Thread-1 leaving.


...Program finished with exit code 0
Press ENTER to exit console.
```

**Using multi-threading (With Synchronization):**

```
class Main{

    public static void main(String[] args){ Reservation

        reserve = new Reservation(); Person thread1 =

        new Person(reserve, 2); thread1.start(); Person

        thread2 = new Person(reserve, 2);

        thread2.start();

    }

}

class Reservation{

    static int availableSeats = 2; synchronized void

    reserveSeat(int requestedSeats) {

        System.out.println(Thread.currentThread().getName() + " entered.");

        System.out.println("Availableseats : " + availableSeats + "\n
Requestedseats : " + requestedSeats);

        if (availableSeats >= requestedSeats){ System.out.println("Seat

            Available. Reserve now :-)"); try{

                Thread.sleep(100);


            }

            catch (InterruptedException e){

                System.out.println("Thread interrupted");

            }
            System.out.println(requestedSeats + " seats reserved.");

                availableSeats = availableSeats - requestedSeats;

        }
```

```
            else{
                System.out.println("Requested seats not available :-(");
            }
            System.out.println(Thread.currentThread().getName() + " leaving.");
        }
    }
    class Person extends Thread{
        Reservation reserve;
        int requestedSeats;
        public Person(Reservation reserve, int requestedSeats){ this.reserve
            = reserve;
            this.requestedSeats = requestedSeats;
        }
        public void run() { reserve.reserveSeat(requestedSeats);
        }
    }
```