10) Inter-process communication (IPC) and Deadlock

1. Inter process communication (produced consumer problem)

⇒ class Q {

```
int n;
boolean valueSet = false;
Synchronized int get() {
    while (!valueSet) {
        try {
            System.out.println("In consumer
                waiting\n");
            wait();
        }
        catch (InterruptedException e) {
        System.out.println("InterruptedException
            caught");
        }
    }
```

```
System.out.println("Got : " + n);
valueSet = false;
System.out.println("In Intimate Producer \n");
notify();
return n;
}
Synchronized void put(int n).
{
while(valueSet) {
try {
System.out.println("In produce waiting \n");
wait();
}
catch (InterruptedException e) {
system.out.println("Interrupted.
Exception caught");
} }
this.n = n;
valueSet = true;
System.out.println("Put : " + n);
System.out.println("In Intimate consum
er \n");
notify();
}
}
class producer Implements Runnable {
Q q;
producer(Q q) {
this.q = q;
new Thread(this, "Producer").start();
}
public void run() {
int i = 0;
while (i < 15) {
```

```
        q.put(i++);
    }
}
}

class Consumer implements Runnable{
    Q q;
    Consumer(Q q){
        this.q = q;
        new Thread(this, "consumer").start();
    }
    public void run(){
        int i = 0;
        while(i<15){
            int x = q.get();
            System.out.println("Consumed : "+x);
            i++;
        }
    }
}

public class PCFixed{
    public static void main(String[] args)
    {
        Q q = new Q();
        new Producer(q);
        new Consumer(q);
        System.out.println("Press Control-c
                            to stop");
    }
}
```

o/p:
```
put : 0
Interrupted Consumer
get : 0
Consumed : 0
Interrupted Producer
...
```

2. Deadlock

```
class A {
    synchronized void foo(B b) {
        String name = Thread.currentThread().getName();
        System.out.println(name + " entered A.foo");
        try{
            Thread.sleep(1000);
        }
        catch (Exception e) {
            System.out.println(name + " A Interrupted");
        }
        System.out.println(name + " trying to call B.last()");
        b.last();
    }
    synchronized void last() {
        System.out.println("Inside A.last");
    }
}
class B {
    synchronized void bar(A a) {
        String name = Thread.currentThread().getName();
        System.out.println(name + " entered B.bar");
        try {
            Thread.sleep(1000);
        }
        catch (Exception e) {
            System.out.println("B Interrupted");
        }
```

```
System.out.println(name + "trying to
        call A.last()");
    a.last();
}
synchronized void last() {
    System.out.println("Inside B.last");
}
}
class Deadlock implements Runnable {
    A a = new A();
    B b = new B();

    Deadlock() {
        Thread.currentThread().setName
            ("mainthread");
        Thread t = new Thread(this, "Rac
            -ngthread");
        t.start();
        a.foo(b);
        System.out.println("Back in main
            thread");
    }
    public void run() {
        b.bar(a);
        System.out.println("Back in other
            thread");
    }
    public static void main(String[] arg
        new Deadlock();
    }
}
```

o/p : Main thread entered A.foo.
Racing thread entered B.bar
Main thread trying to call B.last()
Racing thread trying to call A.last()