Experiment No. 7

Aim: Implementation of Deadlock.

Theory:

- Deadlocks in distributed systems are similar to deadlocks in single processor systems, only worse.
  - They are harder to avoid, prevent or even detect.
  - They are hard to cure when tracked down because all relevant information is scattered over many machines.
- People sometimes might classify deadlock into the following types:
  - Communication deadlocks -- competing with buffers for send/receive
  - Resources deadlocks -- exclusive access on I/O devices, files, locks, and other resources.
- We treat everything as resources, there we only have resources deadlocks.
- Four best-known strategies to handle deadlocks:
  - The ostrich algorithm (ignore the problem)
  - Detection (let deadlocks occur, detect them, and try to recover)
  - Prevention (statically make deadlocks structurally impossible)
  - Avoidance (avoid deadlocks by allocating resources carefully) Distributed Deadlock Detection:
- Since preventing and avoiding deadlocks to happen is difficult, researchers works on detecting the occurrence of deadlocks in distributed system.
- The presence of atomic transaction in some distributed systems makes a major conceptual difference.
  - When a deadlock is detected in a conventional system, we kill one or more processes to break the deadlock --- one or more unhappy users.
  - When deadlock is detected in a system based on atomic transaction, it is resolved by aborting one or more transactions.
  - But transactions have been designed to withstand being aborted.
  - When a transaction is aborted, the system is first restored to the state it had before the transaction began, at which point the transaction can start again.
  - With a bit of luck, it will succeed the second time.
  - Thus, the difference is that the consequences of killing off a process are much less severe when transactions are used.

Algorithm:

1. Let the number of resource classes be m, with E1 resources of class 1, E2 resources of class 2, and generally, Ei resources of class i ($1 < i < m$). E is the existing resource vector. It gives the total number of instances of each resource in existence. For example, if class 1 is tape drives, then $E1 = 2$ means the system has two tape drives.
2. At any instant, some of the resources are assigned and are not available. Let A be the available resource vector, with Ai giving the number of instances of resource i that are currently available (i.e., unassigned). If both of our two tape drives are assigned, A1 will be 0.
3. Let C, the current allocation matrix, and R, the request matrix. The i-th row of C tells how many instances of each resource class Pi currently holds. Thus Cij is the number of instances of resource j that are held by process i. Similarly, Rij is the number of instances of resource j that Pi wants.
4. Define the relation $A \leq B$ on two vectors A and B to mean that each element of A is less than or equal to the corresponding element of B. Mathematically, $A \leq B$ holds if and only if $Ai \leq Bi$ for $1 < i < m$.
5. Each process is initially said to be unmarked. As the algorithm progresses, processes will be marked, indicating that they are able to complete and are thus not deadlocked.

When the algorithm terminates, any unmarked processes are known to be deadlocked.

The deadlock detection algorithm can now be given, as follows.

1. Look for an unmarked process, Pi, for which the i-th row of R is less than or equal to A.
2. If such a process is found, add the i-th row of C to A, mark the process, and go back to step 1.
3. If no such process exists, the algorithm terminates.

When the algorithm finishes, all the unmarked processes, if any, are deadlocked.

Code:

```java
import java.io.*;
public class ChandyMisraHaas {
    public static int flag = 0;
    public static void main(String args[]) throws Exception {
        BufferedReader ob = new BufferedReader(new InputStreamReader(System.in));
        int init, aa, bb, x = 0, end = 5;
        File input = new File("Dependencies.txt");
        BufferedReader in = new BufferedReader(new InputStreamReader(new FileInputStream(input)));
        String line;
        int[][] a = new int[end][end];
        line = in.readLine();
        line = in.readLine();
        while ((line = in.readLine()) != null) {
            aa = 3;
            bb = 4;
            for (int y = 0; y < end; y++) {
                a[x][y] = Integer.parseInt(line.substring(aa, bb));
                aa += 2;
                bb += 2;
            }
            x++;
        }
        System.out.println("_____________________________________________________________");
        System.out.println();
        System.out.println(" CHANDY-MISRA-HAAS DISTRIBUTED DEADLOCK DETECTION ALGORITHM");
        System.out.println();
        System.out.println("\tS1\tS2\tS3\tS4\tS5");
        for (int i = 0; i < end; i++) {
            System.out.print("S" + (i + 1) + "\t");
            for (int j = 0; j < end; j++) {
                System.out.print(a[i][j] + "\t");
            }
            System.out.println();
        }
        System.out.println();
        System.out.print("Enter Initiator Site No. : ");
        init = Integer.parseInt(ob.readLine());
        int j = init - 1;
        System.out.println();
        System.out.println();
        System.out.println(" DIRECTION\tPROBE");
        System.out.println();
        for (int k = 0; k < end; k++) {
            if (a[j][k] == 1) {
                System.out.println(
                        " S" + (j + 1) + " --> S" + (k + 1) + "     (" + init + "," + (j + 1) + "," + (k + 1) + ")");
                aman(a, j, k);
            }
        }
        if (flag == 0) {
            System.out.println();
            System.out.println(" NO DEADLOCK DETECTED");
        }
        System.out.println("_____________________________________________________________");
        ob.readLine();

    }
    public static void aman(int[][] a, int init, int k) {
        int end = 5;
        for (int x = 0; x < end; x++) {
            if (a[k][x] == 1) {
                if (init == x) {
                    System.out.println(" S" + (k + 1) + " --> S" + (x + 1) + "     (" + (init + 1) + "," + (k + 1) + ","
                            + (x + 1) + ")" + " --------> DEADLOCK DETECTED");
                    flag = 1;
                    break;
                }
                System.out.println(" S" + (k + 1) + " --> S" + (x + 1) + "     (" + (init + 1) + "," + (k + 1) + ","
                        + (x + 1) + ")");
                aman(a, init, x);
            }
        }
    }
}
```

Output:

```
D:\Practical>java ChandyMisraHaas
-----------------------------------------------------------

 CHANDY-MISRA-HAAS DISTRIBUTED DEADLOCK DETECTION ALGORITHM

        S1      S2      S3      S4      S5
S1      0       1       0       0       1
S2      0       0       1       0       0
S3      0       0       0       1       1
S4      1       0       0       0       0
S5      0       0       0       0       0

Enter Initiator Site No. : 1


 DIRECTION      PROBE

 S1 --> S2     (1,1,2)
 S2 --> S3     (1,2,3)
 S3 --> S4     (1,3,4)
 S4 --> S1     (1,4,1) --------> DEADLOCK DETECTED
 S3 --> S5     (1,3,5)
 S1 --> S5     (1,1,5)
-----------------------------------------------------------
```

Conclusion:

When multiple copies of some of the resources exist, a different approach is needed to detect deadlocks. Above method present a matrix-based algorithm for detecting deadlock among n processes. Deadlocks is not recovered but instead are allowed to occur and then the situation is analysed and later some action might be taken to recover the occurred deadlock.