Computer Science & Information Systems

**Lab Sheet-Deadlocks**

1. **Objective:**

At the end of this lab session student will be able to:

* Use two methods of resolving a deadlock condition.
* Use three methods of preventing a deadlock condition.

1. **Theory:**

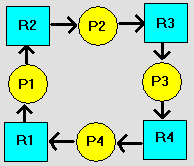
Processes share computer systems resources. In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. A **deadlock** is a situation in which two programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function. For necessary conditions for deadlock to happen are: Mutual exclusion, hold and wait, No preemption and circular wait. Apart from providing mechanisms for handling process creation, scheduling, synchronization, operating system is also responsible for deadlock handling for processes.

Deadlocks can be described more precisely in terms of a directed graph called a **system resource allocation graph**. This graph consists of a set of vertices *V* and a set of edges *E.* The set of vertices V is partitioned into two different types of nodes: *P* == { P1, P2, ... , Pn}, the set consisting of all the active processes in the system, and *R* == {R1, R2, ... , Rm }the set consisting of all resource types in the system. A directed edge from process Pi to resource type Rjis denoted by Pi 🡪 Rj ; it signifies that process Pi has requested an instance of resource type Rjand is currently waiting for that resource. A directed edge from resource type Rjto process Piis denoted by Rj 🡪 Pj; it signifies that an instance of resource

type Rjhas been allocated to process Pi.

1. **Steps to perform :**
   1. Enter the source code in the compiler using compiler’s PROGRAM SOURCE[INPUT] editor window and compile the source code to generate the executable Binary Code.
   2. Load the executable code in memory.
   3. Create a single process from program in the OS simulator.
   4. Select Round Robin scheduling algorithm, set RR Time slice to required ticks ansimulation speed is set at maximum.
   5. Start executing the process.
   6. Use different techniques to overcome deadlock condition and use visualization tools to analyse the performance.
2. **Examples :**

**Example 1 : This exercise demonstrates the deadlock situation amongst 4 processes**



Consider thefollowing 4 programs:

**DeadlockP1.txt :**

program P1

resource(1, allocate)

wait(3)

resource(2, allocate)

for n = 1 to 20

next

end

**DeadlockP2.txt :**

program P2

resource(2, allocate)

wait(3)

resource(3, allocate)

for n = 1 to 20

next

end

**DeadlockP3.txt :**

program P3

resource(3, allocate)

wait(3)

resource(4, allocate)

for n = 1 to 20

next

end

**DeadlockP4.txt :**

program P4

resource(4, allocate)

wait(3)

resource(1, allocate)

for n = 1 to 20

next

end

**Procedure:**

**Step 1**: Compile all the four source codes and load in in main memory at starting addresses 100, 200, 300 and 400 respectively.

**Step 2:** Create 4 processes P1, P2, P3 and P4

**Step 3:** Select round robin scheduling algorithm and set time slice to 10 ticks

**Step 4:** Start the process.

**Step 5:** Open the system resources window by pressing **VIEW RESOURCES..** button under views tab. Then press **SHOW DEADLOCKED PROCESSES..** button. Observe the resource allocation graph.

**Example 2 : Two methods of recovering from a deadlock condition after it happens.**

**Method 1: Release resource method**

**Procedure:**

**Step 1**: Compile all the four source codes and load in in main memory at starting addresses 100, 200, 300 and 400 respectively.

**Step 2:** Create 4 processes P1, P2, P3 and P4

**Step 3:** Select round robin scheduling algorithm and set time slice to 10 ticks

**Step 4:** Start the process.

**Step 5:** Open the system resources window by pressing **VIEW RESOURCES..** button under views tab. Then press **SHOW DEADLOCKED PROCESSES..** button. Observe the resource allocation graph. If four resource shapes are in red color then it indicates the deadlock situation.

**Step 6:** Select one of these resources and click on the **Release** button next to it. Observe the resource allocation graph.

**Method 2: Remove process method**

**Procedure:**

**Step 1**: Compile all the four source codes and load in in main memory at starting addresses 100, 200, 300 and 400 respectively.

**Step 2:** Create 4 processes P1, P2, P3 and P4

**Step 3:** Select round robin scheduling algorithm and set time slice to 10 ticks

**Step 4:** Start the process.

**Step 5:** Open the system resources window by pressing **VIEW RESOURCES..** button under views tab. Then press **SHOW DEADLOCKED PROCESSES..** button. Observe the resource allocation graph. If four resource shapes are in red color then it indicates the deadlock situation.

**Step 6:** In the OS simulator window, select a process in the waiting queue in the **WAITING PROCESSES** frame. Click on the REMOVE button and observe the processes. Observe the resource allocation graph.

Note: Before removing a process, CPU should be suspended.

**Example 3 : Three methods of preventing a deadlock condition before it happens**

**Method 1: Disallow hold and wait**

**Procedure:**

**Step 1**: Compile all the four source codes and load in in main memory at starting addresses 100, 200, 300 and 400 respectively.

**Step 2:** Create 4 processes P1, P2, P3 and P4

**Step 3:** Select round robin scheduling algorithm and set time slice to 10 ticks

**Step 4:** In the **System Resources** window select the **Disallow hold and wait** check box in the **Prevent** frame.

**Step 5:** Start the process.

**Step 6:** Observe the resource allocation graph.

**Method 2: Disallow circular hold**

**Procedure:**

**Step 1**: Compile all the four source codes and load in in main memory at starting addresses 100, 200, 300 and 400 respectively.

**Step 2:** Create 4 processes P1, P2, P3 and P4

**Step 3:** Select round robin scheduling algorithm and set time slice to 10 ticks

**Step 4:** In the **System Resources** window select the **Disallow Circular wait** check box in the **Prevent** frame.

**Step 5:** Start the process.

**Step 6:** Observe the resource allocation graph.

**Method 3: Total Ordering**

Here the resources are allocated in increasing resource id numbers only. So, for example, resource R3 must be allocated after resources R0 to R2 and resource R1 cannot be allocated after resource R2 is allocated.

**Procedure:**

**Step 1**: Compile all the four source codes and load in in main memory at starting addresses 100, 200, 300 and 400 respectively.

**Step 2:** Create 4 processes P1, P2, P3 and P4

**Step 3:** Select round robin scheduling algorithm and set time slice to 10 ticks

**Step 4:** In the **System Resources** window select the **Use total ordering** check box in the **Prevent** frame.

**Step 5:** Start the process.

**Step 6:** Observe the resource allocation graph.