**Comp 4735 Winter 2015**

## Lab Instructor: Mirela Gutica SET : 4D

**Name: Jens Christiansen Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Lab 6

Solve the following exercises. Work in pairs. Discuss each exercise with your lab instructor.

1. Describe the paths 1- 6 from Figure 6.2.

**Path 1**

* 1. Process Q obtains resource B
  2. Process Q obtains resource A
  3. Process Q releases resource B
  4. Process Q releases resources A

**Path 2**

1. Q obtains B
2. Q obtains A
3. Process P tries to obtain A; blocked
4. Q releases B
5. Q releases A

**Path 3**

1. Q obtains B
2. P obtains A
3. Deadlock; both process are waiting for a resource from the other.

**Path 4**

1. P obtains A
2. Q obtains B
3. Deadlock; both process are waiting for a resource from the other.

**Path 5**

1. P obtains A
2. P obtains B
3. Q tries to obtain B; blocked
4. P releases A
5. P releases B

**Path 6**

1. P obtains A
2. P obtains B
3. P releases A
4. P releases B
5. Discuss Table 6.1.

The conditions for a deadlock are mutual exclusion; hold and wait; no preemption; and circular wait. It is through circular wait, however, that makes a deadlock occur.

**Mutual exclusion**

Only one process may use a resource at a time. No process

may access a resource unit that has been allocated to another process.

**Hold and wait**

A process may hold allocated resources while awaiting assignment

of other resources.

**No preemption**

No resource can be forcibly removed from a process holding it.

**Circular wait**

A closed chain of processes exists, such that each process holds

at least one resource needed by the next process in the chain

There are three approaches for deadlock prevention.

**Prevention**

To prevent a deadlock, the system is designed such that it prevents the occurrence of one of the first three conditions above, in order to avoid the fourth condition, circular wait. It is quite conservative in allocation of resources and inefficient execution of processes.

**Avoidance**

The system dynamically predicts if the acquisition of a resource will potentially produce a deadlock. It requires knowledge of future process resource requests.

**Detection**

An algorithm is run periodically to determine if a circular wait condition has occurred. Once a deadlock has been detected, a number of remedies can be used: Abort all deadlocked processes; back up each deadlocked process to some previously defined checkpoint, and restart all processes; successively abort deadlocked processes until dealock no longer exists; successively preempt resources until deadlock no longer exists.

1. Write the pseudo-code for the dinning philosophers’ problem.

semaphore fork [5] = {1};

int i;

void philosopher (int i)

{

while (true) {

think();

wait (fork[i]);

wait (fork [(i+1) mod 5]);

eat();

signal(fork [(i+1) mod 5]);

signal(fork[i]);

}

}

void main()

{

begin (

philosopher (0), philosopher (1),

philosopher (2), philosopher (3),

philosopher (4)

);

}

1. Solve a problem with the Banker’s algorithm.

**Initial State**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **R1** | **R2** | **R3** |  |  | **R1** | **R2** | **R3** |  |  | **R1** | **R2** | **R3** |
| **P1** | **2** | **4** | **1** |  | **P1** | **1** | **2** | **0** |  | **P1** | **1** | **2** | **1** |
| **P2** | **2** | **3** | **3** |  | **P2** | **1** | **1** | **2** |  | **P2** | **1** | **2** | **1** |
| **P3** | **0** | **1** | **4** |  | **P3** | **0** | **1** | **2** |  | **P3** | **0** | **0** | **2** |
| **Claim Matrix C** | | | |  | **Allocation Matrix A** | | | |  | **C - A** | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **R1** | **R2** | **R3** |  |  |  | **R1** | **R2** | **R3** |  |  |  |
|  |  | **3** | **4** | **6** |  |  |  | **1** | **0** | **2** |  |  |  |
|  |  | **Resource Vector R** | | |  |  |  | **Available vector V** | | |  |  |  |

**P3 Runs to completion**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **R1** | **R2** | **R3** |  |  | **R1** | **R2** | **R3** |  |  | **R1** | **R2** | **R3** |
| **P1** | **2** | **4** | **1** |  | **P1** | **1** | **2** | **0** |  | **P1** | **1** | **2** | **1** |
| **P2** | **2** | **3** | **3** |  | **P2** | **1** | **1** | **2** |  | **P2** | **1** | **2** | **1** |
| **P3** | **0** | **0** | **0** |  | **P3** | **0** | **0** | **0** |  | **P3** | **0** | **0** | **2** |
| **Claim Matrix C** | | | |  | **Allocation Matrix A** | | | |  | **C - A** | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **R1** | **R2** | **R3** |  |  |  | **R1** | **R2** | **R3** |  |  |  |
|  |  | **3** | **4** | **6** |  |  |  | **0** | **1** | **2** |  |  |  |
|  |  | **Resource Vector R** | | |  |  |  | **Available vector V** | | |  |  |  |

**P1 Runs to Completion**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **R1** | **R2** | **R3** |  |  | **R1** | **R2** | **R3** |  |  | **R1** | **R2** | **R3** |
| **P1** | **0** | **0** | **0** |  | **P1** | **0** | **0** | **0** |  | **P1** | **0** | **0** | **0** |
| **P2** | **2** | **3** | **3** |  | **P2** | **1** | **1** | **2** |  | **P2** | **1** | **2** | **1** |
| **P3** | **0** | **0** | **0** |  | **P3** | **0** | **0** | **0** |  | **P3** | **0** | **0** | **0** |
| **Claim Matrix C** | | | |  | **Allocation Matrix A** | | | |  | **C - A** | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **R1** | **R2** | **R3** |  |  |  | **R1** | **R2** | **R3** |  |  |  |
|  |  | **3** | **4** | **6** |  |  |  | **2** | **3** | **3** |  |  |  |
|  |  | **Resource Vector R** | | |  |  |  | **Available vector V** | | |  |  |  |

**P2 Runs to Completion**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **R1** | **R2** | **R3** |  |  | **R1** | **R2** | **R3** |  |  | **R1** | **R2** | **R3** |
| **P1** | **0** | **0** | **0** |  | **P1** | **0** | **0** | **0** |  | **P1** | **0** | **0** | **0** |
| **P2** | **0** | **0** | **0** |  | **P2** | **0** | **0** | **0** |  | **P2** | **0** | **0** | **0** |
| **P3** | **0** | **0** | **0** |  | **P3** | **0** | **0** | **0** |  | **P3** | **0** | **0** | **0** |
| **Claim Matrix C** | | | |  | **Allocation Matrix A** | | | |  | **C - A** | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **R1** | **R2** | **R3** |  |  |  | **R1** | **R2** | **R3** |  |  |  |
|  |  | **3** | **4** | **6** |  |  |  | **3** | **4** | **6** |  |  |  |
|  |  | **Resource Vector R** | | |  |  |  | **Available vector V** | | |  |  |  |

1. Solve a problem with the deadlock detection algorithm.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **R1** | **R2** | **R3** |  |  | **R1** | **R2** | **R3** | |
| **P1** | **0** | **0** | **0** |  | **P1** | **0** | **0** | **0** | |
| **P2** | **2** | **0** | **0** |  | **P2** | **0** | **0** | **2** | |
| **P3** | **0** | **0** | **2** |  | **P3** | **2** | **0** | **0** | |
| **Request Matrix Q** | | | |  | **Allocation Matrix A** | | | | |
|  |  |  |  |  |  |  |  |  | |
|  |  | **R1** | **R2** | **R3** |  | **R1** | **R2** | **R3** | |
|  |  | **2** | **1** | **2** |  | **0** | **1** | **0** | |
|  |  | **Resource Vector R** | | |  | **Available vector W** | | |

Each cell in P1’s Request matrix Q is less than its corresponding cell in available vector W; therefore, the process is marked as OK.

Cell P2/R1 in Q exceeds Cell R1 in W; there is a deadlock in process 2.

Cell P3/R3 in Q exceeds Cell R3 in W; there is a deadlock in process 3.

1. Solve problem 6.3 in textbook.

**Path 1:**

1. Q obtains B
2. Q obtains A
3. Q releases B
4. Q releases A

**Path 2:**

1. Q obtains B
2. Q obtains A
3. P tries to obtain A; blocked
4. Q releases B
5. Q releases A

**Path 3:**

1. Q obtains B
2. P obtains A
3. P releases A
4. Q obtains A
5. Q releases B
6. Q releases A

**Path 4:**

1. P obtains A
2. Q obtians B
3. P releases A
4. Q obtains A
5. Q releases B
6. P obtains B
7. P releases B

**Path 5:**

1. P obtains A
2. P releases A
3. P obtains B
4. Q tries to obtain B; blocked
5. P releases B

**Path 6:**

1. P obtains A
2. P releases A
3. P obtains B
4. P releases B
5. Solve problems 6.6; 6.12; 6.14.

6.12)

The Banker’s algorithm is an avoidance algorithm, and as such is more inefficient with resource allocation and program execution; know how much of each resource a process could possibly request. In most systems, this information is unavailable, making it impossible to implement the Banker's algorithm.

6.14)

a) Yes, it can create a deadlock. If one process gets blocked, the other process gets blocked as well.

1. foo() semWait (S);

2. bar() semWait(R);

3. foo() semWait(R);

4. bar() semWait(S);

5. DEADLOCK

b) No.