University of Calgary

CPSC 457 - Principles of Operating Systems

Professor: Dr. Pavol Federl

Assignment #4

Deadlocks, Resource Allocation, and Banker's Algorithm

Evan Loughlin

Student ID: 00503393

Tutorial Section: T04

TA: Sina Keshvadi (AKA: John Cena)

March 16, 2018

Q1

Provide a simple example of a system in an unsafe state, where the processes could finish without entering a deadlock. Show the state, and two sequences of execution: one leading to a deadlock, and the other leading to the completion of all processes.

Safe State

A state is said to be safe if there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately. An unsafe state is not necessarily guaranteed to lead to a deadlock. Rather, an unsafe state is a state whereby no guarantee can be given that the processes can all finish without resulting in a deadlock.

	Has	**Max**
A	3	9
B	2	4
C	2	7

Free: 3

The above example is in a safe state, because there exists a scheduling order whereby it can be guaranteed that all processes can complete without leading to a deadlock. In this instance, allocating 2 resources to process B will allow it to finish. Then, 5 resources will be available for C to use. Process C can then finish, and return its 2 resources to the free stock, leaving 7, of which process A only needs 6. Then process A can finish, too. This demonstrates that the above is in a safe state.

Unsafe State Leading to Deadlock

	Has	**Max**
A	4	9
B	2	4
C	2	7

Free: 2

The above, however, is in an unsafe state. This is the same example as the safe state, except that process A has taken 1 additional resource. Now, it is still possible for Process B to complete, since there are 2 free resources. Once process B finishes, it can return 2 of its allocated resources to the free stock. However, only 4 resource instances will be free, which is not enough for either process A or process C to complete. This is a deadlock situation, shown below.

	Has	**Max**
A	4	9
B	-	-
C	2	7

Free: 4

Unsafe State Leading to Completion

Consider the same unsafe state:

	Has	**Max**
A	4	9
B	2	4
C	2	7

Free: 2

Although this is an unsafe state, it is not guaranteed to end in deadlock. For instance, say that the resource is preemptable. The scheduler can remove a single instance of the resource from Process A, in order to return it to a safe state, as shown:

	Has	**Max**
A	3	9
B	2	4
C	2	7

Free: 3

Next, process B can finish:

	Has	**Max**
A	3	9
B	4	4
C	2	7

Free: 1

Then, process B will return its resources.

	Has	**Max**
A	3	9
B	-	-
C	2	7

Free: 5

Next, process C can finish, and return its resources:

	Has	**Max**
A	3	9
B	-	-
C	7	7

Free: 0

	Has	**Max**
A	3	9
B	-	-
C	-	-

Free: 7

Then, process A can finish:

	Has	**Max**
A	9	9
B	-	-
C	-	-

Free: 1

	Has	**Max**
A	-	-
B	-	-
C	-	-

Free: 10

Q2

In a system consisting of four processes and five resource types, the current allocation and maximum needs are as follows:

Process	**Allocation**	**Maximum**	**Available**
P0	1 0 2 1 1	1 1 2 1 3	0 0 x 1 2
P1	2 0 1 1 0	2 2 2 1 0	
P2	1 1 0 1 0	2 1 3 1 0	
P3	1 1 1 1 0	1 1 2 2 1	

What is the smallest value of 'x' that can keep the system in a safe state? Once you find the 'x' value, find a safe execution order using the Safety Algorithm we discussed in the lecture. Include a side-by-side walk-through of the Safety Algorithm.

The Safety Algorithm (Banker's Algorithm) can be implemented in the following way:

1. Look for a row, R, whose unmet resource needs are all smaller than or equal to A. If no such row exists, the system will eventually deadlock.

2. Assume the process of the chosen row requests all of the resources it needs, and finishes. Mark that process as terminated and add all of its resources to the A vector.

3. Repeat steps 1 and 2 until either all processes are marked terminated (in which case the initial state was safe), or no process is left whose resource needs can be met (in which case the system was not safe).

In the above example, the algorithm can be followed as given below (and in this example, the smallest value of x to make the state safe will also be found.):

1. Start by making a new column, entitled the "required" column. Required = Maximum - Allocation. This shows how many resources a given process requires to be terminated.

Process	**Allocation**	**Maximum**	**Required**	**Available**
P0	1 0 2 1 1	1 1 2 1 3	0 1 0 0 2	0 0 x 1 2
P1	2 0 1 1 0	2 2 2 1 0	0 2 1 0 0	
P2	1 1 0 1 0	2 1 3 1 0	1 0 3 0 0	
P3	1 1 1 1 0	1 1 2 2 1	0 0 1 1 1	

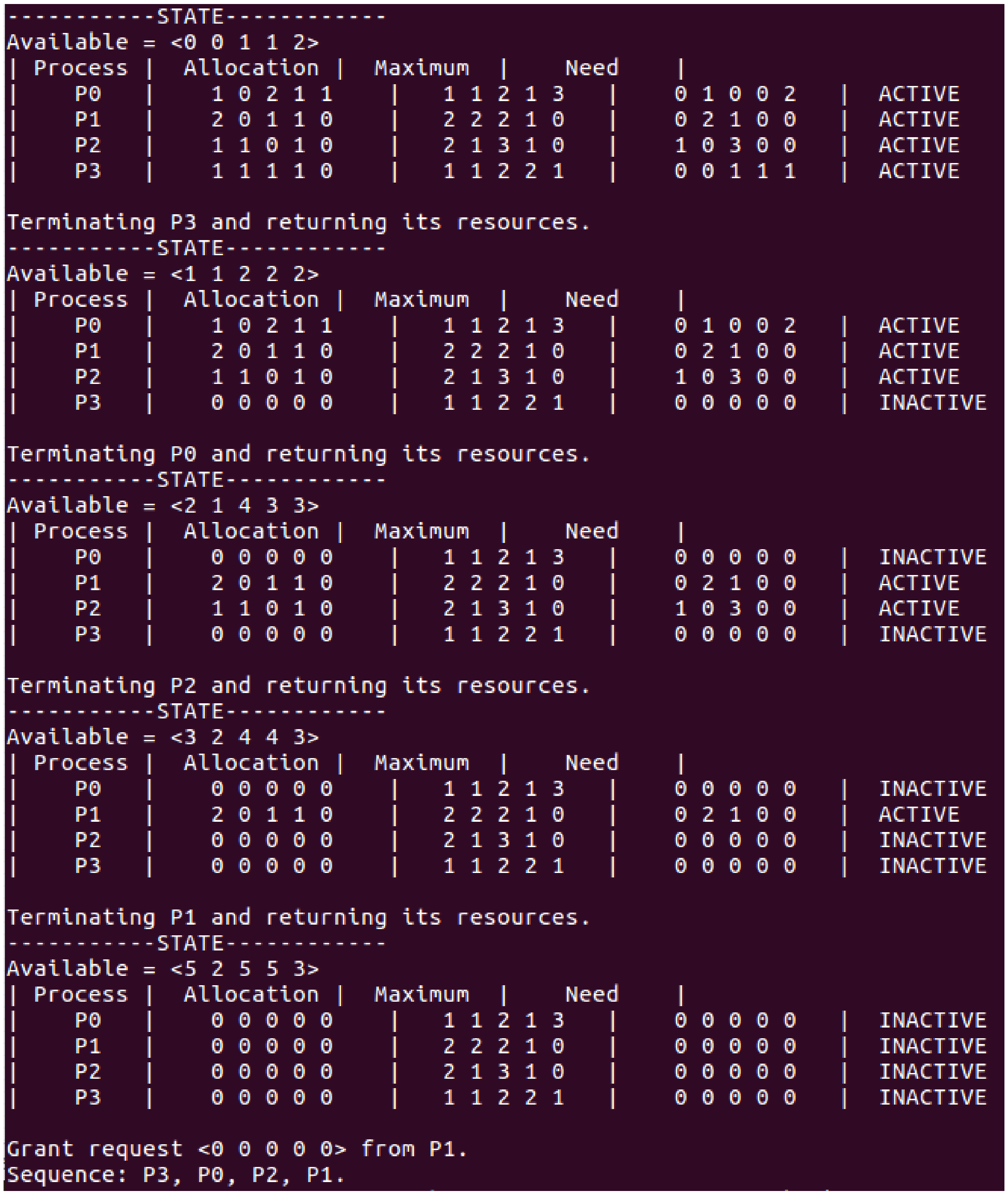
2. P0 Required = (0 1 0 0 2) > (0 0 x 1 2), therefore FALSE for all positive values of x.
3. P1 Required = (0 2 1 0 0) > (0 0 x 1 2), therefore FALSE for all positive values of x.
4. P2 Required = (1 0 3 0 0) > (0 0 x 1 2), therefore FALSE for all positive values of x.
5. P3 Required = (0 0 1 1 1) < (0 0 x 1 2), TRUE for x $\geq$ 1.
6. Terminate P3, return its allocated resources to available.
   Recalculate available = (0 0 x 1 2) + (1 1 1 1 0) = (1 1 (1+x) 2 2)
7. P0 Required = (0 1 0 0 2) < (1 1 (1+x) 2 2), TRUE for x $\geq$ 0.
8. Terminate P0, return its allocated resources to available.
   Recalculate available = (1 1 (1+x) 2 2) + (1 0 2 1 1) = (2 1 (3+x) 3 3)
9. P1 Required = (0 2 1 0 2) > (2 1 (3+x) 3 3), therefore FALSE for all positive values of x.
10. P2 Required = (1 0 3 0 0) < (2 1 (3+x) 3 3), TRUE for x $\geq$ 0.
11. Terminate P2, return its allocated resources to available.
    Recalculate available = (2 1 (3+x) 3 3) + (1 1 0 1 0) = (3 2 (3+x) 4 3)
12. P1 Required = (0 2 1 0 2) < (3 2 (3+x) 4 3), TRUE for x $\geq$ 0.
13. All processes are terminated. Therefore the state is safe for x $\geq$ 1.

Q3

Write a program that implements the Banker's Algorithm. Your program will determine whether there is a safe execution sequence for a given set of processes, and a request. The process information will be provided in a configuration file, specified as a command-line argument. The configuration file will contain the number of processes (numProc), the number of resource types (numResourceTypes), currently available resources (available), current resource allocation for all processes, the maximum resources required by each process, and a request (request) by process 'i'.

Refer to banker.cpp.

The following screenshot of the output is shown, demonstrating the working program with the input from Q2:



```
-----------STATE------------
Available = <0 0 1 1 2>
| Process |  Allocation |  Maximum  |    Need    |
|    P0   |    1 0 2 1 1    |   1 1 2 1 3   |   0 1 0 0 2   |  ACTIVE
|    P1   |    2 0 1 1 0    |   2 2 2 1 0   |   0 2 1 0 0   |  ACTIVE
|    P2   |    1 1 0 1 0    |   2 1 3 1 0   |   1 0 3 0 0   |  ACTIVE
|    P3   |    1 1 1 1 0    |   1 1 2 2 1   |   0 0 1 1 1   |  ACTIVE

Terminating P3 and returning its resources.
-----------STATE------------
Available = <1 1 2 2 2>
| Process |  Allocation |  Maximum  |    Need    |
|    P0   |    1 0 2 1 1    |   1 1 2 1 3   |   0 1 0 0 2   |  ACTIVE
|    P1   |    2 0 1 1 0    |   2 2 2 1 0   |   0 2 1 0 0   |  ACTIVE
|    P2   |    1 1 0 1 0    |   2 1 3 1 0   |   1 0 3 0 0   |  ACTIVE
|    P3   |    0 0 0 0 0    |   1 1 2 2 1   |   0 0 0 0 0   |  INACTIVE

Terminating P0 and returning its resources.
-----------STATE------------
Available = <2 1 4 3 3>
| Process |  Allocation |  Maximum  |    Need    |
|    P0   |    0 0 0 0 0    |   1 1 2 1 3   |   0 0 0 0 0   |  INACTIVE
|    P1   |    2 0 1 1 0    |   2 2 2 1 0   |   0 2 1 0 0   |  ACTIVE
|    P2   |    1 1 0 1 0    |   2 1 3 1 0   |   1 0 3 0 0   |  ACTIVE
|    P3   |    0 0 0 0 0    |   1 1 2 2 1   |   0 0 0 0 0   |  INACTIVE

Terminating P2 and returning its resources.
-----------STATE------------
Available = <3 2 4 4 3>
| Process |  Allocation |  Maximum  |    Need    |
|    P0   |    0 0 0 0 0    |   1 1 2 1 3   |   0 0 0 0 0   |  INACTIVE
|    P1   |    2 0 1 1 0    |   2 2 2 1 0   |   0 2 1 0 0   |  ACTIVE
|    P2   |    0 0 0 0 0    |   2 1 3 1 0   |   0 0 0 0 0   |  INACTIVE
|    P3   |    0 0 0 0 0    |   1 1 2 2 1   |   0 0 0 0 0   |  INACTIVE

Terminating P1 and returning its resources.
-----------STATE------------
Available = <5 2 5 5 3>
| Process |  Allocation |  Maximum  |    Need    |
|    P0   |    0 0 0 0 0    |   1 1 2 1 3   |   0 0 0 0 0   |  INACTIVE
|    P1   |    0 0 0 0 0    |   2 2 2 1 0   |   0 0 0 0 0   |  INACTIVE
|    P2   |    0 0 0 0 0    |   2 1 3 1 0   |   0 0 0 0 0   |  INACTIVE
|    P3   |    0 0 0 0 0    |   1 1 2 2 1   |   0 0 0 0 0   |  INACTIVE

Grant request <0 0 0 0 0> from P1.
Sequence: P3, P0, P2, P1.
```