Bronson Edralin

EE 468 Operating Systems

Homework 6

Total Points = 11

10/23/14

**Textbook Problem 7.16 (3 pts)**

In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods (months). Resources break or are replaced, new processes come and go, and new resources are bought and added to the system. If deadlock is controlled by the banker’s algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and under what circumstances?

1. Increase Available (new resources added).
2. Decrease Available (resource permanently removed from system).
3. Increase Max for one process (the process needs or wants more resources than allowed).
4. Decrease Max for one process (the process decides it does not need that many resources).
5. Increase the number of processes.
6. Decrease the number of processes.
7. This is safe at any time. A deadlock can’t be created by increasing number of resources available because it can only decrease number of processes that has to wait for resources.
8. This is safe only when Max is less or equal to Available. The case when deadlock can occur for decreasing number of resources is when system decides to use old resource data for the next state. If change in resources number occurs after the evaluation, the system can proceed with allocation while the next state becomes unsafe.
9. It is safe only when Max is less or equal to Available Error condition could happen for two reasons. One possible reason may be from Banker’s algorithm point of view and process exceeds his maximum claim. Another way could be when the condition that decided it to be safe right before it became unsafe because changes occurred right after safety evaluation.
10. This is safe at any time.
11. With increase of the number of processes, deadlock may occur since the decision factor whether the next state will be safe may be based on data that is only of processes present in system.
12. This is safe at any time.

**Textbook Problem 7.17 (1 pt)**

Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock free.

Yes, this system will be deadlock free. A system has only four resources. There are three processes where each process requires a maximum of two resources. Here is the reasoning: Every process holds one single resource while any process may hold two resources. Therefore, when a job of the first process is finished, all the resources are free so this means the remaining process will acquire the resources.

**Textbook Problem 7.18 (1 pt)**

Consider a system consisting of m resources of the same type being shared by n processes. A process can request or release only one resource at a time. Show that the system is deadlock free if the following two conditions hold:

1. The maximum need of each process is between one resource and m resources.
2. The sum of all maximum needs is less than m + n.
3. This relationship may be expressed by the following expressions: and . The maximum need for each process is . Since Maxi is greater or equal to 1, this means each process Pi has at least one resource that it can release. Obviously this system is not in a deadlock state.
4. This may be expressed by the following equation: where , , and These equations show that it is possible to have a process Pi where Needi=0. Since Maxi is greater or equal to 1, this means each process Pi has at least one resource that it can release. Obviously this system is not in a deadlock state too.

**Textbook Problem 7.19 (1 pt)**

Consider the version of the dining-philosophers problem in which the chopsticks are placed at the center of the table and any two of them can be used by a philosopher. Assume that requests for chopsticks are made one at a time. Describe a simple rule for determining whether a particular request can be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

do {

wait(chopstick[i]);

wait(chopstick[(i+1)%5]);

-----------

// eat

-----------

signal(chopstick[i]);

signal(chopstick[(i+1)%5]);

------------

// think

------------

} while(true);

This will prevent deadlock. A philosopher’s request for the first chopstick can only be satisified if there is no other philosopher with two chopsticks and if there is only one chopstick remaining.

**Textbook Problem 7.22 (2 pts)**

Consider the following snapshot of a system:

|  |  |  |
| --- | --- | --- |
|  | ***Allocation*** | ***Max*** |
|  | A B C D | A B C D |
| P0 | 3 0 1 4 | 5 1 1 7 |
| P1 | 2 2 1 0 | 3 2 1 1 |
| P2 | 3 1 2 1 | 3 3 2 1 |
| P3 | 0 5 1 0 | 4 6 1 2 |
| P4 | 4 2 1 2 | 6 3 2 5 |

Using the banker’s algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the processes may complete. Otherwise, illustrate why the state is unsafe.

1. ***Available*** = (0, 3, 0, 1)
2. ***Available*** = (1, 0, 0, 2)

Need = Max – Allocation

|  |  |
| --- | --- |
|  | ***Need*** |
|  | A B C D |
| P0 | 2 1 0 3 |
| P1 | 1 0 0 1 |
| P2 | 0 2 0 0 |
| P3 | 4 1 0 2 |
| P4 | 2 1 1 3 |

1. **This is not safe.** Processes P2, P1 and P3 but no remaining processes are able to finish so deadlock occurs.

*RESULT*

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Need*** | ***Allocated*** | ***Res Avail.*** |
|  | A B C D | A B C D | A B C D |
| P2 | 0 2 0 0 | 3 1 2 1 | 0 3 0 1 |
| P1 | 1 0 0 1 | 2 2 1 0 | 3 4 2 1 |
| P3 | 4 1 0 2 | 0 5 1 0 | 5 6 3 1 |
|  |  |  | 5 11 4 1 |
|  |  |  |  |

1. **This is safe.** Processes P1, P2 and P3 are able to finish. Then processes P0 and P4 were also able to finish.

*RESULT*

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Need*** | ***Allocated*** | ***Res Avail.*** |
|  | A B C D | A B C D | A B C D |
| P2 | 0 2 0 0 | 3 1 2 1 | 1 0 0 2 |
| P1 | 1 0 0 1 | 2 2 1 0 | 4 1 2 3 |
| P3 | 4 1 0 2 | 0 5 1 0 | 6 3 3 3 |
| P4 | 2 1 1 3 | 4 2 1 2 | 6 8 4 3 |
| P0 | 2 1 0 3 | 3 0 1 4 | 8 10 5 5 |

**Textbook Problem 7.23 (2 pts = 1pt for (a) and 1 pt for (b,c))**

Consider the following snapshot of a system:

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Allocation*** | ***Max*** | ***Available*** |
|  | A B C D | A B C D | A B C D |
| P0 | 2 0 0 1 | 4 2 1 2 | 3 3 2 1 |
| P1 | 3 1 2 1 | 5 2 5 2 |  |
| P2 | 2 1 0 3 | 2 3 1 6 |  |
| P3 | 1 3 1 2 | 1 4 2 4 |  |
| P4 | 1 4 3 2 | 3 6 6 5 |  |

Answer the following questions using the banker’s algorithm:

1. Illustrate that the system is in a safe state by demonstrating an order in which the processes may complete.
2. If a request from process P1 arrives for (1, 1, 0, 0), can the request be granted immediately?
3. If a request from process P4 arrives for (0, 0, 2, 0), can the request be granted immediately?

|  |  |
| --- | --- |
|  | ***Need*** |
|  | A B C D |
| P0 | 2 2 1 1 |
| P1 | 2 1 3 1 |
| P2 | 0 2 1 3 |
| P3 | 0 1 1 2 |
| P4 | 2 2 3 3 |

1. **This is safe.** With Processes finishing in order P1, P2, P3, P4, and P0.

*RESULT*

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Need*** | ***Allocated*** | ***Res Avail.*** |
|  | A B C D | A B C D | A B C D |
| P1 | 2 1 3 1 | 3 1 2 1 | 3 3 2 1 |
| P2 | 0 2 1 3 | 2 1 0 3 | 6 4 4 2 |
| P3 | 0 1 1 2 | 1 3 1 2 | 8 5 4 5 |
| P4 | 2 1 1 3 | 1 4 3 2 | 9 8 5 7 |
| P0 | 2 2 1 1 | 2 0 0 1 | 10 12 8 9 |

1. **This is safe** with processes finishing in order P0, P3, P4, P1, and P2.

*GIVEN*:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Allocation*** | ***Max*** | ***Available*** | ***Need*** |
|  | A B C D | A B C D | A B C D | A B C D |
| P0 | 2 0 0 1 | 4 2 1 2 | 2 2 2 1 | 2 2 1 1 |
| P1 | 4 2 2 1 | 5 2 5 2 |  | 1 0 3 1 |
| P2 | 2 1 0 3 | 2 3 1 6 |  | 0 2 1 3 |
| P3 | 1 3 1 2 | 1 4 2 4 |  | 0 1 1 2 |
| P4 | 1 4 3 2 | 3 6 6 5 |  | 2 2 3 3 |

*RESULT:*

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Need*** | ***Allocated*** | ***Res Avail.*** |
|  | A B C D | A B C D | A B C D |
| P0 | 2 2 1 1 | 2 0 0 1 | 2 2 2 1 |
| P3 | 0 1 1 2 | 1 3 1 2 | 4 2 2 2 |
| P4 | 2 2 3 3 | 1 4 3 2 | 5 5 3 4 |
| P1 | 1 0 3 1 | 4 2 2 1 | 6 9 6 6 |
| P2 | 0 2 1 3 | 2 1 0 3 | 10 11 8 7 |

1. **This is not safe.** Deadlock definitely occurs here with all processes failing because there is not enough available resources.

*GIVEN*:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Allocation*** | ***Max*** | ***Available*** | ***Need*** |
|  | A B C D | A B C D | A B C D | A B C D |
| P0 | 2 0 0 1 | 4 2 1 2 | 3 3 0 1 | 2 2 1 1 |
| P1 | 3 1 2 1 | 5 2 5 2 |  | 2 1 3 1 |
| P2 | 2 1 0 3 | 2 3 1 6 |  | 0 2 1 3 |
| P3 | 1 3 1 2 | 1 4 2 4 |  | 0 1 1 2 |
| P4 | 1 4 5 2 | 3 6 6 5 |  | 2 2 1 3 |

**Textbook Problem 7.25 (1 pt)**

A single-lane bridge connects the two Vermont villages of North Tunbridge and South Tunbridge. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if a northbound and a southbound farmer get on the bridge at the same time. (Vermont farmers are stubborn and are unable to back up.) Using semaphores and/or mutex locks, design an algorithm in pseudocode that prevents deadlock. Initially, do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa).

Semaphore okay\_to\_cross = 1;

ENTER\_BRIDGE()

P(okay\_to\_cross);

EXIT\_BRIDGE()

V(okay\_to\_cross);