7.6: 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).

This could safely be changed without any problems.

1. Decrease Available (resource permanently removed from system).

This could have an effect on the system and introduce the possibility of deadlock as the safety of the system assumed there were a certain number of available resources.

1. Increase Max for one process (the process needs or wants more resources than allowed).

This could have an effect on the system and introduce the possibility of deadlock.

1. Decrease Max for one process (the process decides that it does not need that many resources).

This could safely be changed without any problems.

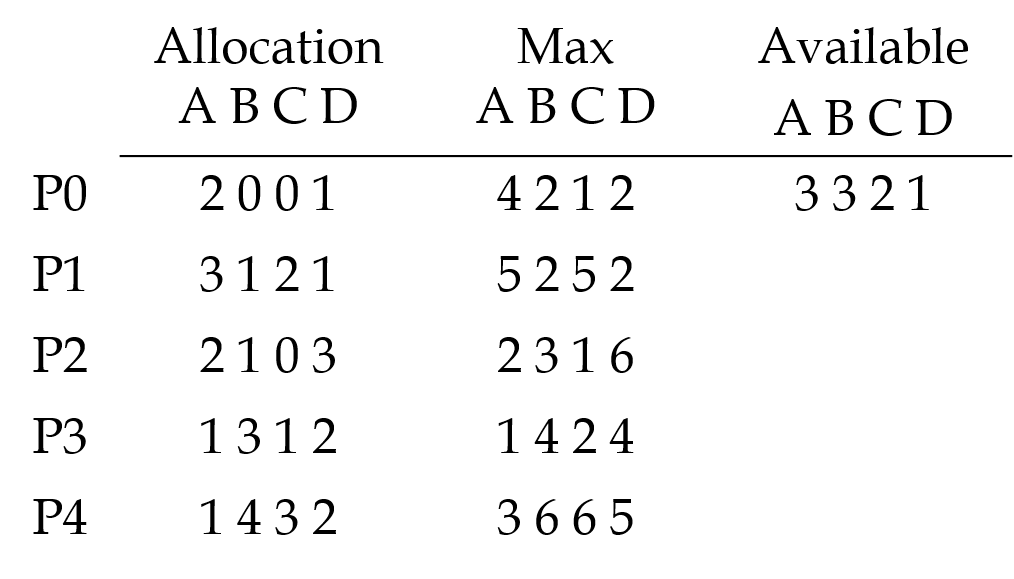
1. Increase the number of processes.

This could be allowed assuming that resources were allocated to the new process such that the system does not enter an unsafe state.

1. Decrease the number of processes.

This could safely be changed without any problems.

7.13: Consider the following snapshot of a system:



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.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Need | | | |  | Available | | | |
|  | A | B | C | D | 3 3 2 1 | A | B | C | D |
| P0 | 2 | 2 | 1 | 1 | init+P0 | 5 | 3 | 2 | 2 |
| P1 | 2 | 1 | 3 | 1 | init+P0+P3+P4+P1 | 10 | 11 | 8 | 7 |
| P2 | 0 | 2 | 1 | 3 | init+P0+P3+P4+P1+P2 | 12 | 12 | 8 | 10 |
| P3 | 0 | 1 | 1 | 2 | init+P0+P3 | 6 | 6 | 3 | 4 |
| P4 | 2 | 2 | 3 | 3 | init+P0+P3+P4 | 7 | 10 | 6 | 6 |

P0->P3->P4->P1->P2

1. If a request from Process P1 arrives for (1,1,0,0), can the request be granted immediately?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Need | | | |  | Available | | | |
|  | A | B | C | D | 2 2 2 1 | A | B | C | D |
| P0 | 2 | 2 | 1 | 1 | init+P0 | 4 | 2 | 2 | 2 |
| P1 | 1 | 0 | 3 | 1 | init+P0+P3+P4+P1 | 9 | 10 | 8 | 7 |
| P2 | 0 | 2 | 1 | 3 | init+P0+P3+P4+P1+P2 | 11 | 11 | 8 | 10 |
| P3 | 0 | 1 | 1 | 2 | init+P0+P3 | 5 | 5 | 3 | 4 |
| P4 | 2 | 2 | 3 | 3 | init+P0+P3+P4 | 6 | 9 | 6 | 6 |

Yes, it can be granted by sequence P0->P3->P4->P1->P2

(c) If a request from Process P4 arrives for (0,0,2,0), can the request be granted immediately?

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

No, the initial available cannot fulfill any process in the sequence, the system will go to unsafe state and all the process will be in deadlock.

7.15: 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 neighbor 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 ok\_to\_cross = 1;

Enter\_bridge() {

ok\_to\_cross.wait();

}

Exit\_bridge() {

ok\_to\_cross.signal();

}

8.1: Explain the difference between internal and external fragmentation.

Internal fragmentation happens when memory is split into mounted-sized blocks. When block size is larger than the memory request, the difference between allotted and requested memory is called internal fragmentation.

External fragmentation happens when there’s a sufficient quantity of area within the memory to satisfy the memory request of a method. The process memory request cannot be fulfilled because the memory offered is in a non-contiguous manner. Whether you apply a first-fit or best-fit memory allocation strategy it will cause external fragmentation.

8.9: Compare paging with segmentation with respect to how much memory the address translation structures require to convert virtual addresses to physical addresses.

Paging requires more memory overhead to maintain the translation structures. Segmentation requires just one (or more) entry for code and one entry for data. Paging on the other hand requires multiple entries for code and data depending on page size. So paging almost requires more memory for page table than the memory space required to segment table.

8.16: Consider a computer system with a 32-bit logical address and 4KB page size. The system supports up to 512MB of physical memory. How many entries are there in each of the following?

(a) A conventional single-level page table

2^32/2^12 = 2^20

1. An inverted page table

2^29/2^12 = 2^17

9.8: Consider the following page reference string:  
7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1.  
Assume demand paging with three frames, how many page faults would occur for the following replacement algorithms?  
(a) LRU replacement

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 2 | 3 | 1 | 2 | 5 | 3 | 4 | 6 | 7 | 7 | 1 | 0 | 5 | 4 | 6 | 2 | 3 | 0 | 1 |
| 7 | 7 | 7 | 1 | 1 | 1 | 3 | 3 | 3 | 7 | 7 | 7 | 7 | 5 | 5 | 5 | 2 | 2 | 2 | 1 |
|  | 2 | 2 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 4 | 4 | 4 | 3 | 3 | 3 |
|  |  | 3 | 3 | 3 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 0 | 0 | 0 | 6 | 6 | 6 | 0 | 0 |
| m | m | m | m | h | m | m | m | m | m | h | m | m | m | m | m | m | m | m | m |

Page faults = 18  
(b) FIFO replacement

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 2 | 3 | 1 | 2 | 5 | 3 | 4 | 6 | 7 | 7 | 1 | 0 | 5 | 4 | 6 | 2 | 3 | 0 | 1 |
| 7 | 7 | 7 | 1 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 0 | 0 | 0 | 6 | 6 | 6 | 0 | 0 |
|  | 2 | 2 | 2 | 2 | 5 | 5 | 5 | 5 | 7 | 7 | 7 | 7 | 5 | 5 | 5 | 2 | 2 | 2 | 1 |
|  |  | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 4 | 4 | 4 | 3 | 3 | 3 |
| m | m | m | m | h | m | h | m | m | m | h | m | m | m | m | m | m | m | m | m |

Page faults = 17  
(c) Optimal replacement

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 2 | 3 | 1 | 2 | 5 | 3 | 4 | 6 | 7 | 7 | 1 | 0 | 5 | 4 | 6 | 2 | 3 | 0 | 1 |
| 7 | 7 | 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  | 2 | 2 | 2 | 2 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 4 | 6 | 2 | 3 | 3 | 3 |
|  |  | 3 | 3 | 3 | 3 | 3 | 4 | 6 | 7 | 7 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| m | m | m | m | h | m | h | m | m | m | h | h | m | h | m | m | m | m | h | h |

Page faults = 13

9.11: Discuss situations in which the least frequently used (LFU) page-replacement algorithm generates fewer page faults than the least recently used (LRU) page-replacement algorithm. Also discuss under which circumstances the opposite holds.

In a small system like files system or application files, users use more frequently of these pages so the system can save more time. However, if a page is heavily used during the initial phase of a process and is never used again. Since the count of the page is high, it will not be eliminated even though it is no longer needed.

9.17: A page-replacement algorithm should minimize the number of page faults. We can achieve this minimization by distributing heavily used pages evenly over all of memory, rather than having them compete for a small number of page frames. We can associate with each page frame a counter of the number of pages associated with that frame. Then, to replace a page, we can search for the page frame with the smallest counter.

(a) Define a page-replacement algorithm using this basic idea. Specifically address these problems:

(i) What is the initial value of the counters?

0

(ii) When are counters increased?

Whenever a page is associated with that frame.

(iii) When are counters decreased?

Whenever one of the pages associated with that frame is no longer required.

(iv) How is the page to be replaced selected?

Find a frame with the smallest counter. Use FIFO for breaking ties.

(b) How many page faults occur for your algorithm for the following reference string with four page frames?

1,2,3,4,5,3,4,1,6,7,8,7,8,9,7,8,9,5,4,5,4,2.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 3 | 4 | 1 | 6 | 7 | 8 | 7 | 8 | 9 | 7 | 8 | 9 | 5 | 4 | 5 | 4 | 2 |
| 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 6 | 6 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
|  | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
|  |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 4 | 4 |
|  |  |  | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 2 |
| m | m | m | m | m | h | h | m | m | m | m | h | h | m | h | h | h | m | m | h | m | m |

Page faults = 14

(c) What is the minimum number of page faults for an optimal page-replacement strategy for the reference string in part(b) with four page frames?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 3 | 4 | 1 | 6 | 7 | 8 | 7 | 8 | 9 | 7 | 8 | 9 | 5 | 4 | 5 | 4 | 2 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 | 6 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 2 |
|  | 2 | 2 | 2 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
|  |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 4 | 4 | 4 | 4 |
|  |  |  | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |
| m | m | m | m | m | h | h | h | m | m | m | h | h | m | h | h | h | h | m | h | h | m |

Page faults = 11

9.19: What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?

Thrashing is caused by under allocation of the minimum number of pages required by a process, forcing it to continuously page fault. The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.