HW3

Written Exercise

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 a 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?

(a) Increase Available (new resources added).

This could safely be changed without any problems.

(b) 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.

(c) 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.

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

This could safely be changed without any problems.

(e) Increase the number of processes.

This could be allowed assum-ing that resources were allocated to the new process(es) such that the system does not enter an unsafe state.

(f) Decrease the number of processes.

This could safely be changed without any problems.

7.13: Consider the following snapshot of a system:

	Allocation A B C D	Max A B C D	Available 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:

(a) Illustrate that the system is in a safe state by demonstrating an order in which the processes may complete.

Need = Max – Allocation.
Now, let us see which of the above needs may be met by the Available resources.
P1's Need are satisfied. So assume we have allocated these resources to P1. P1 completes execution and then releases its resources.
Now Available= <3 3 3 1 2> + <0 1 2 1 1> = <3 4 5 2 3>.
Now, P2's Need are satisfied. So assume we have allocated these resources to P2. P2 completes execution and then releases its resources.
Now Available= <3 4 5 2 3> + <2 1 0 3 1> = <5 5 5 5 4>.
Now, P3's Need are satisfied. So assume we have allocated these resources to P3. P3 completes execution and then releases its resources.
Now Available= <5 5 5 5 4> + <1 3 1 2 2> = <6 8 6 7 6>.
Now, P4's Need are satisfied. So assume we have allocated these resources to P4. P4 completes execution and then releases its resources.
Now Available= <6 8 6 7 6> + <1 4 3 2 2> = <7 12 9 9 8>.
Now, P1, P2, P3 or P4's Need may be satisfied. Assume we have allocated resources to P0. P0 completes execution and then releases its resources.
Now Available= <7 12 9 9 8> + < 2 0 0 1 0>= <9 12 9 10 8>.
Since there is at least one sequence <P1, P2, P3, P4, P0> that can successfully complete execution, there is no deadlock.

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

Let us assume that the request (1, 1, 0, 0, 1) is granted.
Then Available=<3 3 2 1 2> – <1 1 0 0 1> = < 2 2 2 1 1>.
P0 can still be completed with the Available. After it completes,
Available = < 2 2 2 1 1> + <2 0 0 1 1> = <4 2 2 2 2>.
P3 can still finish with the Available. After it completes,

Available = <4 2 2 2 2> + <1 3 1 2 2 > = <5 5 3 4 2>.
P1, P2, or P4 can finish in any order.
Since there is a possible sequence in which all processes can finish even after granting P1’s (1 1 0 0 1) request, the request may be granted immediately.

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

Let us assume that the request (0, 0, 2, 0 1) is granted.
Then Available=<3 3 2 1 2> – <0 0 2 0 1> = < 3 3 0 1 3>.
None of the processes P0-P4 can finish with the new Available. So we should defer granting resources to this request.

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 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 ok_to_cross = 1;
void enter_bridge() {
        P(ok_to_cross);
}
void exit_bridge() {
        V(ok_ to_cross);
}
```

8.1: Explain the difference between internal and external fragmentation.

Internal fragmentation occurs when memory is divided into fixed-sized partitions.
External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.

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 two registers per segment:one to maintain the base of the segment and the other to maintain the extent of the segment. Paging on the other hand requires one entry per page, and this entry provides the physical address in which the page is located.

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

Conventional single-level page table: 2^32 / 2^12 (4000) = 2^20 = 1,048,576

(b) An inverted page table

Inverted page table: 2^29 (512mb)/ 2^12 (4000) = 2^17 = 131,072

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

LRU (18 page faults): 7; 7 2; 7 2 3; 1 2 3; 1 2 5; 3 2 5; 3 4 5; 3 4 6; 7 4 6; 7 1 6; 7 1 0; 5 1 0; 5 4 0; 5 4 6; 2 4 6; 2 3 6; 2 3 0; 1 3 0

(b) FIFO replacement

FIFO (17 page faults): 7; 7 2; 7 2 3; 1 2 3; 1 5 3; 1 5 4; 6 5 4; 6 7 4; 6 7 1; 0 7 1; 0 5 1; 0 5 4; 6 5 4; 6 2 4; 6 2 3; 0 2 3; 0 1 3

(c) Optimal replacement

OPT (13 page fautls): 7; 7 2; 7 2 3; 1 2 3; 1 5 3; 1 5 4; 1 5 6; 1 5 7; 1 5 0; 1 4 0; 1 6 0; 1 2 0; 1 3 0

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.

Consider the following sequence of memory accesses in a system that can hold four pages in memory: 1 1 2 3 4 5 1. When page 5 is accessed,the least frequently used page-replacement algorithm would replace a page other than 1, and therefore would not incur a page fault when page1 is accessed again. On the other hand, for the sequence “1 2 3 4 5 2,” the least recently used algorithm performs better.

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?

initial value of the counters - 0

(ii) When are counters increased?

Counters are increased - whenever a new page is associated with that frame.

(iii) When are counters decreased?

Counters are decreased - whenever one of the pages associated with that frame is no longer required.

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

How the page to be replaced is 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.

14 page faults

(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?

11 page faults

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.