CSI3131 – Operating Systems
Tutorial 8 –Virtual Memory

1. Provide a short definition for each of the following terms:
   Page fault:
   Resident Set
   Working Set:

2. Describe how thrashing can occur in using the terms defined in question 1.

3. Consider the following page-replacement algorithms. Rank these algorithms from 1 to 4, where 1 is the best algorithm and 4 the worst, according to their page-fault rate.
   a. LRU replacement
   b. FIFO replacement
   c. Optimal replacement
   d. Clock (second-chance) replacement

4. Consider a demand-paged computer system where the degree of multiprogramming is currently fixed at four processes. The system was recently measured to determine utilization of CPU and the **paging** disk (which means that the regular I/O to the disk is not included in the statistics). The results are one of the following alternatives. For each case, what is happening? Can the degree of multiprogramming be increased to increase the CPU utilization?
   a. CPU utilization 13 percent; disk utilization 97 percent
   b. CPU utilization 87 percent; disk utilization 3 percent
   c. CPU utilization 13 percent; disk utilization 3 percent

5. An operating system supports a paged virtual memory, using a central processor with a memory access cycle time of 1 microsecond. It costs an additional 1 microsecond to access a page other than the current one. Pages have 1000 bytes, and the paging device is a drum that rotates at 3000 revolutions per minute and transfers 1 million bytes per second. The following statistical measurements were obtained from the system:
   - 1 percent memory accesses reference a page other than the current page.
   - Of the accesses that do not reference the current page, 80 percent reference a page already in memory.
   - When a new page was required, the replaced page was modified 50 percent of the time.

   Calculate the effective access time on this system, assuming that the system is running one process only and that the processor is idle during drum transfers.

6. Consider the following page reference string:
   1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
   How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, or seven frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.
   • LRU replacement
   • FIFO replacement
   • Optimal replacement

Number of frames	LRU	FIFO	Optimal
1			
2			
3			
4			
5			
6			
7			

7. Given a process that references virtual pages during its execution as indicated in the table below and an observation window size Δ of 5; complete the table below to show the process working set changes during its execution.

Page Reference	Working set
3	
2	
4	
3	
4	
2	
2	
3	
4	
5	
6	
7	
6	
7	

8. Consider a process running in a paged memory system with pages of size 256 bytes and a 16-bit logical address. The process image is composed of the following:
   - The matrix A defined as global data (an *int* occupies 4 bytes):
     ```
     int A[32][32];  // A row occupy contiguous memory
     ```
   - Code that occupies page 128 (locations 32768 to 33024) which manipulates the matrix; thus, every instruction fetch will be from page 128.
   - A PCB, 2 pages long, located at address 0.
   - The page table that follows the PCB contains 2 byte entries.
   - The stack starts at page 5 and grows upwards in memory.
   - The heap starts at the bottom of page 127 and grows downward.
   - Global data is stored starting at page 129.

   a. Sketch the virtual memory for the process showing where its various parts are located.

   b. If three physical frames (1, 2, 3) are allocated to the process for executing the program (for simplification, ignore the frames required for the stack, PCB, page table, stack and heap), how many page faults are generated by the following matrix-initialization loops, using LRU replacement, assuming frame 1 already contains the code, and the other two are initially empty?
   ```
   for (int j = 0; j < 32; j++)
      for (int i = 0; i < 32; i++)
         A[i][j] = 0;
   ```

   c. How many page faults are generated by the following matrix-initialization loops given the same conditions as in b?
   ```
   for (int i = 0; i < 32; i++)
      for (int j = 0; j < 32; j++)
        A[i][j] = 0;
   ```

9. Hierarchical Page Tables

Consider a "computer" with 10-bit physical and logical addresses, and with page/frame sizes of 16 bytes. Assume each page table entry takes 2 bytes, i.e. one page can contain up to 8 page table entries. Hierarchical page tables are used in order to be able to access the whole address space while allowing paging of the page table.

Content of the outer page table:

Position	0	1	2	3	4	5	6	7
Frame #		000011	110011	011011		010101	110001	
Valid	0	1	1	1	0	1	1	0

Content of the pages of the page table (- means valid bit set to 0):

Entry #	0	1	2	3	4	5	6	7
Page 0 (not in memory)								
Page 1 (in frame 000011)	000100	000101	-	000110	000111	111000	111001	-
Page 2 (in frame 110011)	001000	-	001001	001010	001011	001100	-	-
Page 3 (in frame 011011)	-	-	011100	100000	-	-	-	-
Page 4 (not in memory								
Page 5 (in frame 010101)	111111	111110	-	-	111101	111100	111011	111010
Page 6 (in frame 110001)	101010	101001	101000	-	101100	-	101101	101111
Page 7 (not in memory)								

a) For each of the following logical addresses compute the corresponding physical address, or mark it as causing page fault (consider them separately):

Logical address	**Physical address (or page fault)**
0010010010	
1000100100	
0110110101	
0111101100	
1101101101	

b) Assume that the frames 100000 to 100111 are free and will be the ones into which the new pages (causing page faults) will be loaded. Update the content of the outer page table and page table entries when the logical address 1001001000 is referenced. What would be the physical address?

10. Virtual Memory Management

Consider a computer with 16 bit logical address, 2 Kbyte pages, with each page table entry taking 2 bytes and the following Inverted Page Table:

Frame #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(PID:log page num)	P4.2	P1,2	P4,3	P1,3	P3,0	P1,5	P4,0	P1,0	P3,3	P2,0	P4,1	P1,6	P2,3	P3,2	P1,7	P2,5

Describe the format of logical address for this computer, by answering the following questions:

a) Draw a bar, indicating how many bits define the offset and how many bits define the logical page number.

b) Translate the following logical addresses into corresponding physical addresses or indicate that the access will cause a page fault.

Logical Address	Physical Address
P1: *00110* 00000000011 (page 6)	
P2: *00000* 00000000000 (page 0)	
P3: *11111* 11111111111 (page 31)	
P4: *00001* 10010010001 (page 1)	
P1: *00111* 01100110011 (page 7)	

c) Assume that you want to represent the same situation using per-process page tables, instead of the Inverted Page Table. Draw the page tables of the processes.

d) How big (in bytes) can a page table of a process be? Do you need hierarchical page tables? Why?

e) Give the **resident set size** (in terms of pages) of each process in the system.

f) Consider a page replacement strategy with **local replacement scope**, using the **LRU** algorithm. What is the final resident set of process P2, and how many page faults has P2 incurred? (You might need to do full simulation to figure that out).

(0,3,5) P2: 1 (PF, repl 0: 3,5,1), 0, 3, 0,1, 5, 1, 2
(0,3,2) P3: 3 (NOPF: 0,2,3), 6, 4
(2, 3, 5, 0, 6, 7) P1: 1, 0, 3, 0, 3
(5,1,2) P2: 1, 3, 2, 5

Legend:
The above lists at the start provides the list of pages in memory for the process according their access time, starting with the least recently used page
For each page accessed, one of the following describes if and how a page is replaced:
No page fault: NOPF: *list* where *list* is the list of pages in memory according their access time, starting with the least recently used page
Page fault: PF, repl *x*: *list* where *x* is the page being replaced and *list* is the list of pages in memory according their access time, starting with the least recently used pages

g) Consider a page replacement strategy with **global replacement scope** using the **FIFO** algorithm. What is the final resident set of process P1? How many page faults happened in total (generated by all processes).

Mem Access	**Pg fault**	**Memory contents (Bolded entry indicates current pointer position, dirty pages in italics)** **(P4,2)**(P1,2)(P4,3)(P1,3)(P3,0)(P1,5)(P4,0)(P1,0)(P3,3)(P2,0)(P4,1)(P1,6)(P2,3)(P3,2)(P1,7)(P2,5)
P2,1		
P2,0		
P2,3		
P2,0		
P2,1		
P2,5		
P2,1		
P2,2		
P3,3		
P3,6		
P3,3		
P1,1		
P1,0		
P1,3		
P1,0		
P1,3		
P2,1		
P2,3		
P2,2		
P2,5		
Final:		

h) List the **clean** pages of **all** processes at the end of these memory requests in g, assuming that initially all pages were clean, and each request to an odd logical page was write and each request to an even logical page was read.