CSI3131 – Operating Systems
Tutorial 8 – Virtual Memory – Solution

1. Provide a short definition for each of the following terms:

   Page fault:

   A page fault occurs when the CPU tries to use a logical address for which a corresponding physical address does not exist, that is, the process page that contains the logical address had not been loaded into physical memory. The OS must load the logical page into physical memory to allow the process to continue its execution.

   Resident Set

   The set of physical frames occupied by a process (note that the resident set cannot exceed the physical frames allocated to the process).Another definition: the set of logical pages of a process loaded into physical memory. Both definitions are essentially equivalent – just two different perspectives.

   Working Set:

   The set of pages required by a process to execute, that is, according to locality of reference, the set of pages required by the process in memory to keep the number of page faults to a minimum.

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

   If a process resident set is smaller than its working set, it will generate a many page faults. This means that the process can spend considerable time waiting for I/O due to these page faults. The OS when it sees that the system is under utilized, can add more processes to the system which can suffer the same fate (since pages must be share among more processes), i.e. wait for I/O because of page faults. This causes a cascading effect that leads to thrashing: a system that spends most of its time doing I/O for page faults and not much time executing code for the processes.

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

Rank	Algorithm
1	Optimal
2	LRU
3	Clock (second chance)
4	FIFO

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

   a. Thrashing is occurring, since the disk utilization is very high while the CPU is underutilized. Processes are spending most of their time paging. The degree of multiprogramming cannot be increased. On the contrary, one or more processes should be suspended to allow increase in the utilization of the CPU.
   b. CPU utilization is sufficiently high to leave things alone. Any increase in the degree of multiprogramming could lead to thrashing.
   c. Increase the degree of multiprogramming. The CPU is available for executing additional processes.

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.

   Timing: **Time to access current page: (1μs)**
   Time to access another page in memory: (2μs)
   Time for one revolution of the drum: $\frac{60{,}000{,}000\,\mu s / \min}{3000\,rev / \min} = 20{,}000\,\mu s / rev$
   Time to find a page on the drum (average ½ revolution) = 10,000 μs
   Time to read or write a page from/to the drum: $\frac{1000\,bytes}{1\,bytes / \mu s} = 1000\,\mu s$

   Effective access time = 0.99 × (1 μs) **- 99% of accesses in current page**
   + 0.008 × (2 μs) **- .008% of access in other page in memory**
   + 0.002 × (10,000 μs + 1,000 μs) **- .002% of access to page not in memory**
   + 0.001 × (10,000 μs + 1,000 μs) **- .001% (50% of .002%) of pages modified**
   = (0.99 + 0.016 + 22.0 + 11.0) μs
   = 34.0 μs

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	20	20	20
2	18	18	15
3	15	16	11
4	10	14	8
5	8	10	8
6	7	10	7
7	7	7	7

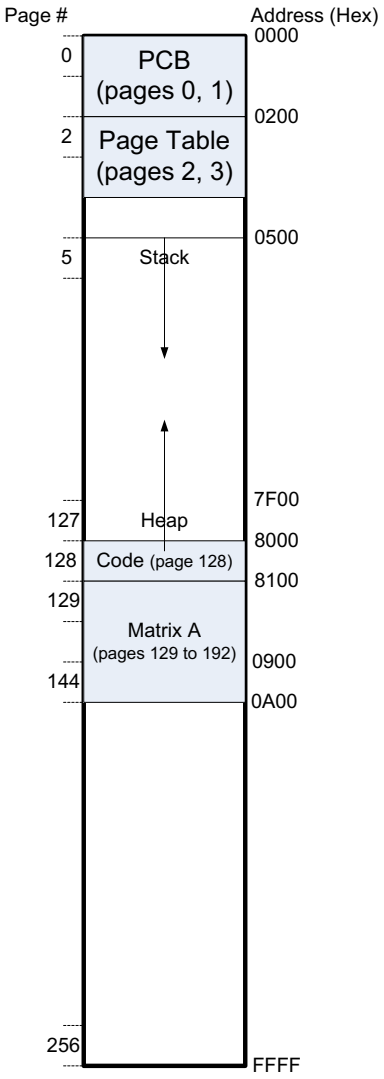
7. Given a process that references virtual pages during its execution as indicated in the table below and an observation window size Δ of 5; complete the table below to show the process working set changes during its execution.

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

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.

   A 16-bit address limits the logical address space (i.e. virtual memory) to 64 Kbytes. Given a page size of 256 bytes the logical address space is divided into 256 pages. The page table occupies 256 entries X 2 bytes/entry = 512 bytes or 2 pages. The matrix A occupies 32 X 32 X 4=4096 bytes, that is, 16 pages (pages 129 to 144, 16 pages). The following sketch shows where all components of the process 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;
```

Consider the execution of the second *for* that varies the variable *i*, keeping *j* constant. Each row occupies a one half page (32*4 = 128 bytes), and when a page of the matrix is loaded, two adjacent rows are in memory. Each time an element of the array is accessed, a different row is accessed (i.e. *i* is assigned a different value), and thus at each second access to a row a different page (not yet in physical memory) is accesses when the index *i* is varied from 0 to 31. LRU will not select frame 1 for replacement since it is accessed whenever an instruction is loaded. Frames 2 and 3 shall contain a part of the matrix and thus shall be selected for replacement each time a different page is referenced. The following table illustrates how the contents of frames 2 and 3 change as the elements in the matrix are accessed.

Matrix Element	Contents of Frame 2	Contents of Frame 3	Page Fault
A[0][0]	Rows 0 and 1	Not loaded	*
A[1][0]	Rows 0 and 1	Not loaded	
A[2][0]	Rows 0 and 1	Rows 2 and 3	*
A[3][0]	Rows 0 and 1	Rows 2 and 3	
A[4][0]	Rows 4 and 5 (replaces rows 0,1)	Rows 2 and 3	*
A[5][0]	Rows 4 and 5	Rows 2 and 3	
A[6][0]	Rows 4 and 5	Rows 6 and 7 (replaces rows 2,3)	*
A[7][0]	Rows 4 and 5	Rows 6 and 7	
.	.	.	.
.	.	.	.
.	.	.	.

Thus for processing a single column, 32/2 = 16 page faults shall be generated. This is repeated for each column; which means a total of 32*16 = 512 page faults.

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;
```

Since the rows are now varied in the outer loop (i.e. *i* is varied in the outer loop), all elements in the each page are updated before moving to the next page; which means that only 16 page faults are required, one for each page containing the matrix.

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 decomposed into 3 bits as the outer page index (ooo), 3 bits for the page table index (ppp), and 4 bits for the offset (ssss): ooo ppp ssss

Logical address	**Physical address (or page fault)**
0010010010	000101 0010
1000100100	Page fault (valid bit 0 for 4 in outer table)
0110110101	100000 0101
0111101100	Page fault (valid bit 0 in Page 3 of page table).
1101101101	101101 1101

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?

Content of the outer page table: **(frame 1000000 is loaded with page 4 of the page table)**

Position	0	1	2	3	4	5	6	7
Frame #		000011	110011	011011	**100000**	010101	110001	
Valid	0	1	1	1	**1**	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 **(in frame 100000)**					100001			
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)								

The logical page is loaded into frame 100001. The corresponding physical address will be 100001 1000.

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.

Page Number 5 bits	**Offset 11 bits**

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)	**01011 00000000011 (frame 11)**
P2: *00000* 00000000000 (page 0)	**01001 00000000000 (frame 9)**
P3: *11111* 11111111111 (page 31)	**Page fault**
P4: *00001* 10010010001 (page 1)	**01010 10010010001 (frame 10)**
P1: *00111* 01100110011 (page 7)	**01110 01100110011 (frame 14)**

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.

	P1	P2	P3	P4
0	7,1	9,1	4,1	6,1
1	-,0	-,0	-	10,1
2	1,1	-,0	13,1	0,1
3	3,1	12,1	8,1	2,1
4	-,0	-,0	-,0	-,0
5	5,1	15,1	-,0	-,0
6	11,1	-,0	.	.
7	14,1	-,0	.	.
8	-,0	.	.	.
.	.	.	.	.
.	.	.		
.	.	.		
32	-,0			

The page table entry shows the frame number and valid/invalid bit

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

Given a byte for the page table entry (5 bits for physical address space and 3 control bits), the page table will be 32 bytes long (max 32 logical pages – see (b)). No need for hierarchical page tables, the size of the page tables is very small relative to the page size (and the process virtual memory).

e) Give the **resident set size** (in terms of pages) of each process in the system.
P1: 6 P2:3 P3: 3 P4: 4

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

P2 incurs 7 page faults, 5 during its first execution cycle and 2 during its second execution cycle. P2 will have in its resident set the pages: 3, 2, 5

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	Y	*(P2,1)***(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)
P2,0		
P2,3		
P2,0		
P2,1		
P2,5		
P2,1		
P2,2	Y	*(P2,1)*(P2,2)**(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)*
P3,3		
P3,6	Y	*(P2,1)*(P2,2)(P3,6)**(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)*
P3,3		
P1,1	Y	*(P2,1)*(P2,2)(P3,6)*(P1,1)***(P3,0)**(P1,5)(P4,0)(P1,0)*(P3,3)*(P2,0)(P4,1)(P1,6)*(P2,3)*(P3,2)(P1,7)*(P2,5)*
P1,0		
P1,3	Y	*(P2,1)*(P2,2)(P3,6)*(P1,1)(P1,3)***(P1,5)**(P4,0)(P1,0)*(P3,3)*(P2,0)(P4,1)(P1,6)*(P2,3)*(P3,2)(P1,7)*(P2,5)*
P1,0		
P1,3		
P2,1		
P2,3		
P2,2		
P2,5		
Final:		*(P2,1)*(P2,2)(P3,6)*(P1,1)(P1,3)***(P1,5)**(P4,0)(P1,0)*(P3,3)*(P2,0)(P4,1)(P1,6)*(P2,3)*(P3,2)(P1,7)*(P2,5)*

P1 resident set: (0,1,3,5,6,7) Total number of page faults: 5

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.

(P2,2)(P3,6)(P1,5)(P4,0)(P1,0) (P2,0)(P4,1)(P1,6) (P3,2)(P1,7)