University Mohamed Khider Biskra Faculty of exact sciences and natural and life sciences. Department of Computer Science		
Level : 2LMD	**Date: March 2024**	**Module** : Operating Systems 1

Series 2: Non-contiguous memory management (solution)

Exercise 1

I)

a)

1) Size of the logical address space (virtual memory space) is:
2^{24} = 16MB (address on 24 bits therefore the number of combinations or addresses =2^{24})

2) Offset= 9 bits (since 2^9=512 the size of a page, so 9 bits are needed to address the 512 slots)

3) 24 – 9 = 15 bits (logical address=[p,offset], offset=9, then p=24-9=15bits)

4) 20 bits (M.M size=1MB=2^{20} bits So the number of bits for addressing this space 2^{20} is 20 bits)

5) 20 – 9 = 11 bits (since total bits for a physical address =20 then offset=9 So 20-9=11bits for the frame)

6) 2^{15} = 32KB (the number of bits to address the pages is 15 then the number of all page addresses = 2^{15} which represents the number of elements in the page table = the entries of Page Table)

b) Yes. The program needs 2000 bytes of data (vector) and 1024 bytes of code, so a total of 3024 bytes in memory.
The number of occupied pages: [3024 / 512] = 6 pages. In the last page, there remain 512 – 464 = 48 free bytes (464=3024-(512*5)), which causes an internal fragmentation.

II) 32/4=8 bits (The format PT1(8bits) PT2(8bits) PT3(8bits) offset(8bits)
The number of tables: 1 + 256 + (256*256).
The number of entries: 256

Exercise 2

Program size=460 Byte
Page size=100 Byte → program size with pages= 5 pages
Main Memory size= 200 Byte → M.M frames number = 2 frames

Waste calculation formula: page size- (pgm size **mod** page size)

Number of wasted space =100 -mod (460/100)=40 Byte (internal fragmentation at last page)

The reference chain: 10 11 104 170 73 309 185 245 246 434 458 364 corresponding to the following page numbers:

0 0 1 1 0 3 1 2 2 4 4 3

NB: to find the number of the corresponding page, you have to make the integer division of the reference number by the size of the page, for example:
For the reference 10: 10/100=0

A) FIFO

	0	0	1	1	0	3	1	2	2	4	4	3
Frame 0	0	0	0	0	0	3	3	3	3	4	4	4
Frame 1			1	1	1	1	1	2	2	2	2	3

Number of page fault is: 6
Page fault rate = 6/12 *100= 50%

B) LRU

	0	0	1	1	0	3	1	2	2	4	4	3
Frame 0	0	0	0	0	0	0	1	1	1	4	4	4
Frame 1			1	1	1	3	3	2	2	2	2	3

Number of page fault is = 7
Page fault rate =7/12 *100= 58%

C) OPTIMAL

	0	0	1	1	0	3	1	2	2	4	4	3
Frame 0	0	0	0	0	0	3	3	3	3	3	3	3
Frame 1			1	1	1	1	1	2	2	4	4	4

Number of page fault is =5
Page fault rate =5/12 *100= 42%

Exercise 3

1 and 2)

-Several algorithms have been proposed:

-Counter method;

-Stack method;

-Mask method

1) Counter (Index) method:

Let **capacity** be the number of pages that memory can hold. Let **set** be the current set of pages in memory.

```
1- Start traversing the pages.

 i) If set holds less pages than capacity.
   a) Insert page into the set one by one until
      the size  of set reaches capacity or all
      page requests are processed.
   b) Simultaneously maintain the recent occurred
      index of each page in a map called indexes.
   c) Increment page fault
 ii) Else
   If current page is present in set, do nothing.
   Else
     a) Find the page in the set that was least
     recently used. We find it using index array.
     We basically need to replace the page with
     minimum index.
     b) Replace the found page with current page.
     c) Increment page faults.
     d) Update index of current page.

2. Return page faults.
```

2. Mask method

It is an implementation method of the LRU replacement algorithm according to the mask method.

1. Each page is associated with a byte.

2. The initial value of the mask is one byte (00000000). The most significant bit is set to 1 each time this page is used.

3. At each period (N ms), a right shift is made of the byte associated with each page.

4. The victim page is the page with the lowest mask value when the algorithm is launched.

```
LRU algorithm;

Var found: boolean;

Min: integer;

Min← byte-min(number-page-load-mm);

Found← unique-test(min);

If (found) Then {page is unique}

        Victim-num ←Selection(min,list-page-load);

Else {page is not unique}

Victim-num ← Selection (min,list-page-load, FIFO);

Finish;
```

3)

Stack method	Counter method	Mask method
It is difficult to manage	**it is easy to update**	**it is easy but requires a shift (update)**
Stack size increases	**the number of counters increases with the size of the central memory**	**by increasing the size of the main memory we will increase the number of the required bytes**

Exercise 4

Access	Segment	Page	Offset	Result
1. Load data	0	1	1	6145
2. Load data	1	1	10	10
3. Load data	3	3	2047	Page Fault
4. Save data	0	1	4	Protection fault
5. Data storage	3	1	2	2050
6. Save data	3	0	14	28686
7. Jump to the address	1	3	100	16484
8. Load data	0	2	50	Page Fault
9. Read data	2	0	5	Page table fault
10. Jump to the address	3	0	60	Protection Fault

The details :

We have the page size= 2kb= 2*1024=2048 B

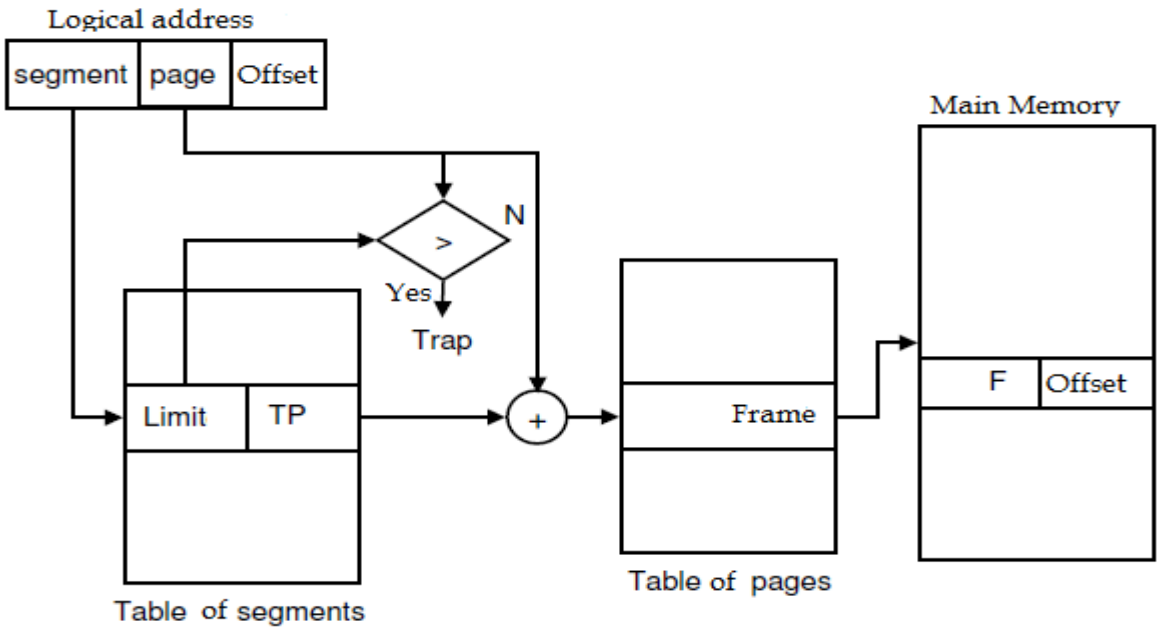
Physical address= frame*page size + offset

The access rights must be checked for each case.

1) Access: Load data with: segment=0, page=1 and d=1
From the segment table: the actual page (box)=3
So the physical address= 3*2048+1=6145
2) Access: Load data with: segment=1, page=1 and d=10
From segment table: actual page (frame)=0
So the physical address= 0*2048+10=10
3) Access: Load data with: segment=3, page=3 and d=2047
From segment table: real page (frame)=virtual page is on disk
So: page fault
4) Access: save data with: segment=0 or access is read-only => protection fault
5)) Access: data storage with: segment=3, page=1 and d=2
From the segment table: the actual page (box)=1
So the physical address= 1*2048+2=2050
6)) Access: save data with: segment=3, page=0 and d=14
From the segment table: the actual page (box)=14
So the physical address= 14*2048+14=28686
7)) Access: Jump to the address with: segment=1, page=3 and d=100
From the segment table: the actual page (box)=8
So the physical address= 8*2048+100=16484
8)) Access: Load data with: segment=0, page=2 and d=50
From the segment table: the actual page (frame)=on disk
So: Page Fault
9)) Access: Read data with: segment=2, page=0 and d=5
From the segment table: the page table is not in MC
So: Page table fault
10)) Access: Jump to the address with: segment=3, page=0 and d=60
From the segment table: access is read and write only
So: Protection fault

Exercise 5

a) The translation of a logical address (s,p,d) to a physical address:



Segments	Pages	Cadres
S1	2	2
	3	0
S2	2	9
S3	1	12

Page size=4kb=1024*4=4096 byte

logical@: 8212 = 2*4096 + 20

SO :

1. segment = S1 (The third page is in segment 1)

2. the page number in the segment = 3

3. Offset in the page = 20 (the rest)

S=1- P=3 - Offset=20

4. frame=0 (from array)

5. the displacement in the frame = 20 (the same displacement as the logical address)

C=0 - offset=20

6. The physical address is therefore 20 in decimal (since the frame number is 0). The physical address is expressed in 16 bits (physical space=64 kB = 2^{16}), including 4 bits for the frame number and 12 bits for the displacement in the frame (4 kB = 2^{12} =2^{10}*2^{2}), which gives us in binary .

C= 0000 and Offset= 0000 0001 0100