1 Exercise 1 - Static Partitioning

Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB, and 250 KB.

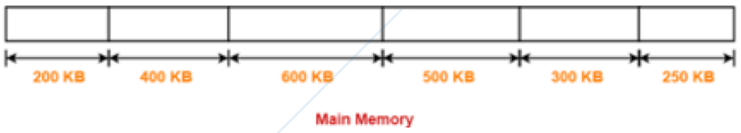


Figure 1: Memory partitioning

THERE ARE FIXED-SIZE PARTITIONS, SO IT IS CALLED STATIC PARTITIONING. ONCE ONE PARTITION IS ALLOCATED FOR A SPECIFIC PROCESS WE CANNOT STORE NOTHING ELSE INTO IT. SO THE REMAINING SPACE WITHIN THE PARTITION (DUE TO THE DIFFERENT SIZE BETWEEN THE PROCESS AND THE PARTITION IS INTERNAL FRAGMENTATION

These partitions need to be allocated to four processes of sizes 357 KB, 210 KB, 468 KB, and 491 KB in that order.

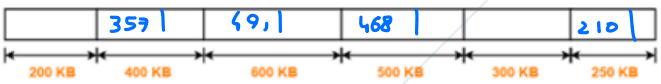
Perform static memory allocation using the first-fit and best-fit strategy. Compute the internal fragmentation.

(1) FIRST FIT

SELECT THE FIRST PARTITION THAT IS BIGGER ENOUGH

- THE LAST PROCESS (491 KB) CANNOT BE STORED INTO THE MEMORY AS THERE IS NOT A PARTITION THAT IS BIGGER ENOUGH
- INTERNAL FRAGMENTATION IS 465 KB

(2) BEST FIT

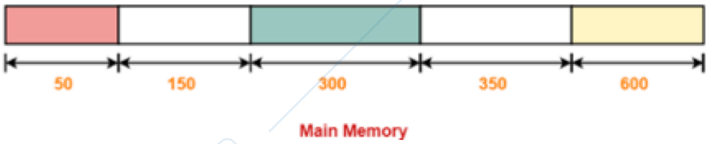


SELECT THE BEST PARTITION) SO THE SMALLEST PARTITION THAT IS BIGGER ENOUGH TO STORE THE GIVEN PROCESS

- HERE WE CAN STORE ALL THE GIVEN PROCESSES
- INTERNAL FRAGMENTATION IS 224 KB

2 Exercise 2 - Dynamic Partitioning

Consider the following heap (figure) in which blank regions are not in use and hatched regions are in use.



IN DYNAMIC PARTITIONING WE SIMPLY RESIZE THE PARTITION WHERE WE'LL STORE THE PROCESS. THE REMAINING FREE SPACE MUST BE ADDED INTO A FREE HOLE SET THAT CONTAINS ALL THE HOLES OF THE MEMORY.

Figure 2: Memory partitioning

A set of processes requests memory blocks of sizes 300, 25, 125, 50. Perform dynamic memory allocation using: (i) the best fit strategy, (ii) the first fit strategy, (iii) the worst fit strategy

1) BEST FIT

- THE LAST PROCESS CANNOT BE STORED INTO THE MEMORY SINCE THERE IS NOT ENOUGHT FREE SPACE

2) FIRST FIT

- ALL PROCESSES CAN BE STORED INTO THE MEMORY

3) WORST FIT

- SAME LIKE FIRST FIT IN THIS CASE

3 Exercise 3 - Buddy System

Consider a memory allocator that uses the buddy system to manage a block of memory with a size of 2048 bytes. The allocator uses a binary splitting algorithm to divide the block into smaller blocks as needed.

Suppose that the allocator receives the following sequence of requests:

1. Allocate a block of memory with a size of 512 bytes
2. Allocate a block of memory with a size of 256 bytes
3. Allocate a block of memory with a size of 128 bytes
4. Allocate a block of memory with a size of 64 bytes
5. Allocate a block of memory with a size of 32 bytes
6. Allocate a block of memory with a size of 16 bytes
7. Free the block of memory allocated in step 4
8. Free the block of memory allocated in step 2
9. Allocate a block of memory with a size of 512 bytes
10. Free the block of memory allocated in step 1

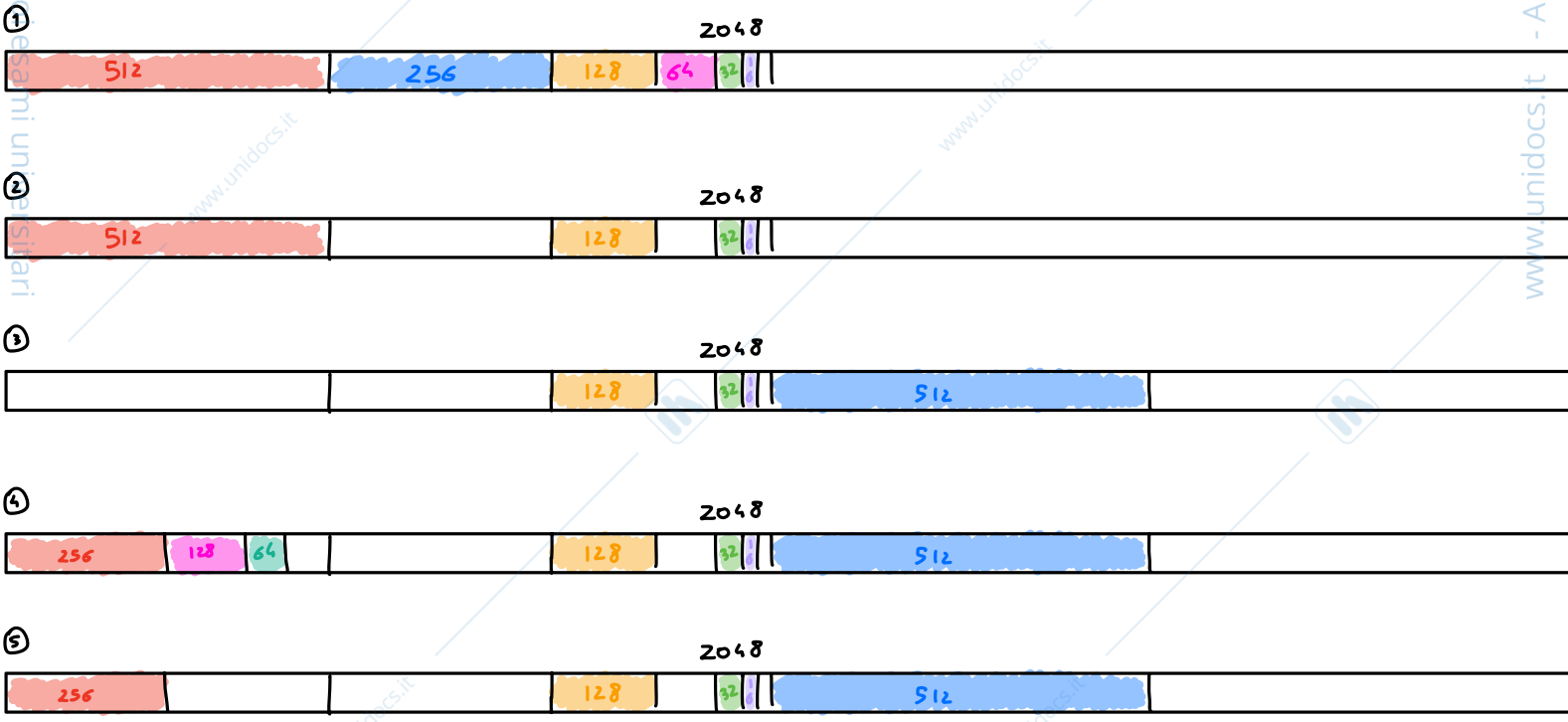
- BE CAREFULL WHEN YOU MERGE TWO BUDDIES IN ONLY ONE BUDDY
- WHEN YOU NEED TO STORE SOMETHING USE BEST FIT

Show the state of the memory after each request is handled, including the size and address of each allocated and free block.

Now suppose that the allocator receives the following additional requests:

11. Allocate a block of memory with a size of 256 bytes
12. Allocate a block of memory with a size of 128 bytes
13. Allocate a block of memory with a size of 64 bytes
14. Free the block of memory allocated in step 10
15. Free the block of memory allocated in step 9

Show the final state of the memory after all requests are handled, including the size and address of each allocated and free block.

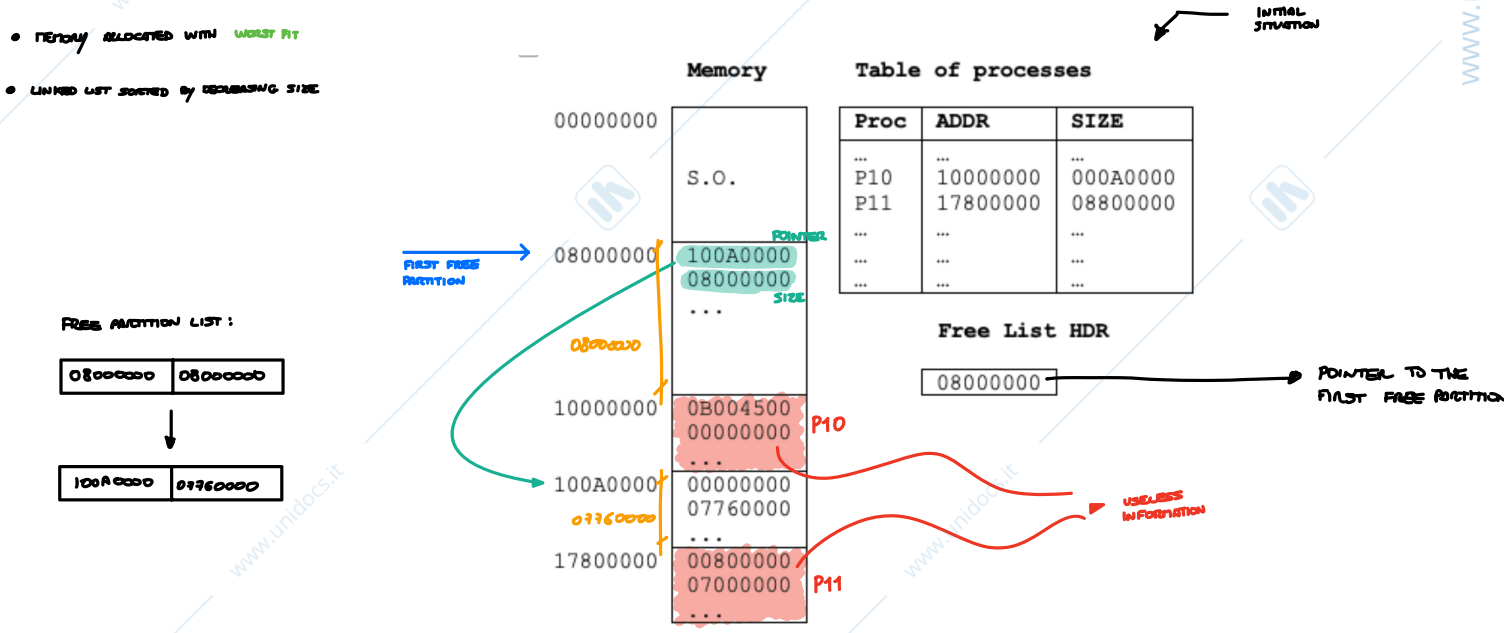


4 Exercise 4 - Contiguous allocation

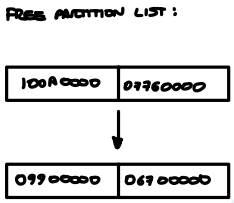
Consider a system with physical memory of 512MB, in which a management scheme with (contiguous) variable partitions is used with a minimum memory allocation unit of 64B (i.e. the memory space is allocated in multiples of 64 bytes). The first 128MB of memory are permanently allocated to the Operating System.

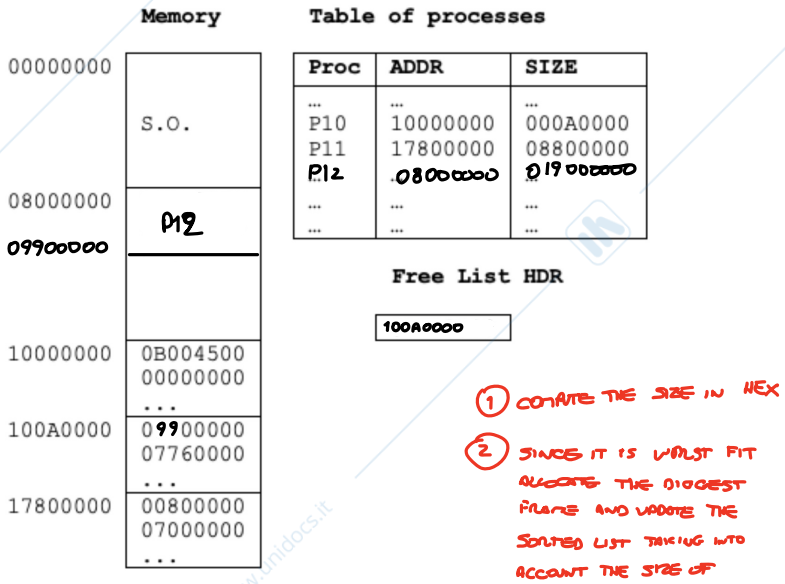
The process table contains, for each active process, the starting address (ADDR) and the size (SIZE) of the relative partition in memory. The memory is allocated with the Worst-Fit strategy. Free partitions are managed through a linked list sorted by decreasing size, in which each node represents a free partition; the nodes of the list consist of two fields: (pointer to the next partition, and size of the partition, both represented on 4 bytes, size, and addresses represented in Byte, with the value 0 used as a null pointer) and are stored in the first bytes of the partition which represent.

Suppose that at a given instant, the process table and the pointer to the first free partition contain the information shown in Figure 2. Represent the changes to the partitions in memory, the process table, and the Free List, following the activation of 2 new processes, P12 and P13, which require respectively 25MB and 150MB of memory, followed by the termination of the P11 process.



① ACTIVATION OF P12 → 25 MB (19 MB)



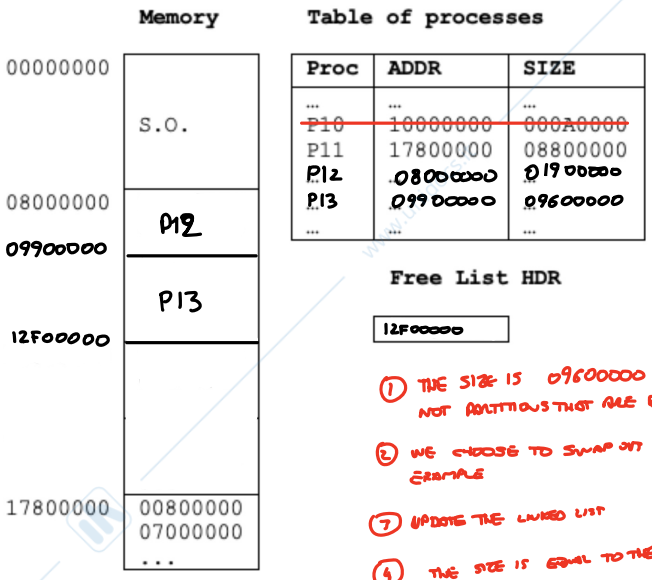


① COMPUTE THE SIZE IN HEX

② SINCE IT IS WORST FIT ALLOCATE THE BIGGEST FREE AND UPDATE THE SORTED LIST TAKING INTO ACCOUNT THE SIZE OF PARTITIONS

① ACTIVATION OF P13 → 150 MB

DEC : $150 \cdot 2^{20}$ B → 09600000 BYTES IN EXADECI



① THE SIZE IS 09600000 BUT THERE ARE NOT PARTITIONS THAT ARE BIG ENOUGHT

② WE CHOOSE TO SWAP OUT P10 FOR EXAMPLE

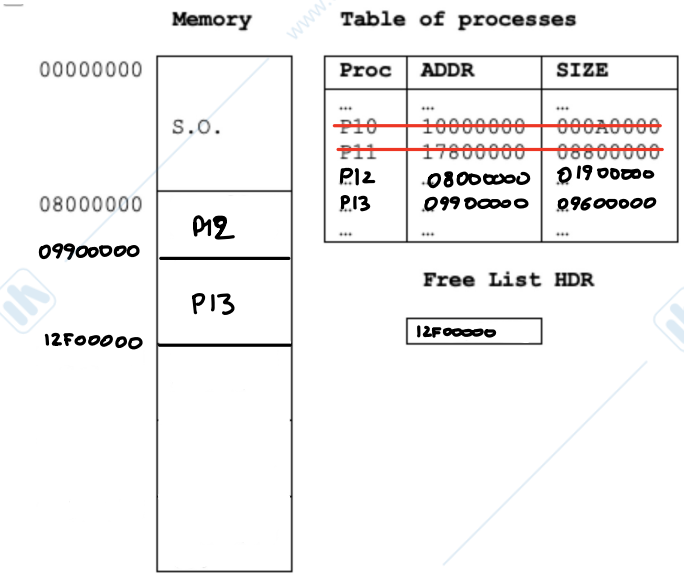
③ UPDATE THE LINKED LIST

④ THE SIZE IS EQUAL TO THE SUM OF THE THREE PARTITIONS
067000000 + 000A0000 + 09760000

⑤ ALLOCATE P13

⑥ UPDATE EVERYTHING AGAIN

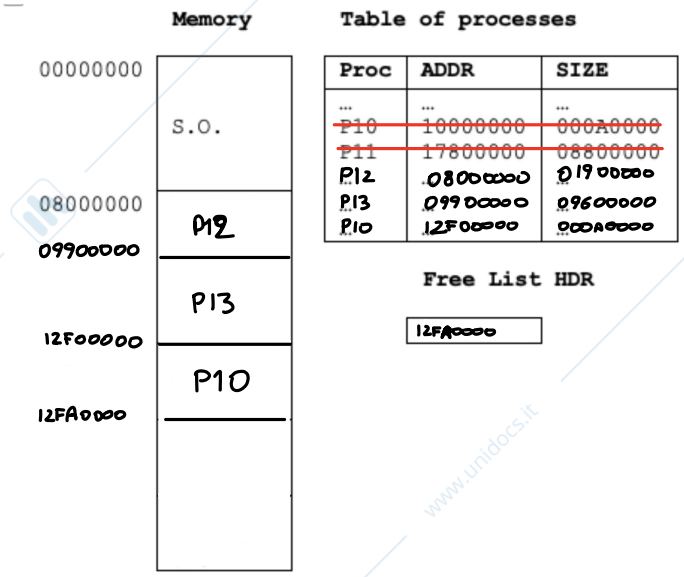
① P11 TERMINATO



FREE PARTITION LIST:

00000000	00100000

① P10 SWAP IN AGAIN



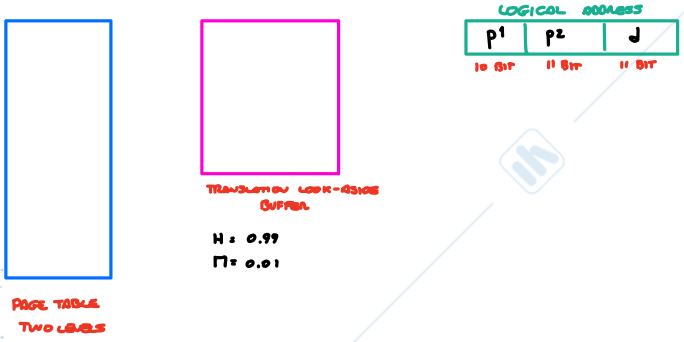
FREE PARTITION LIST:

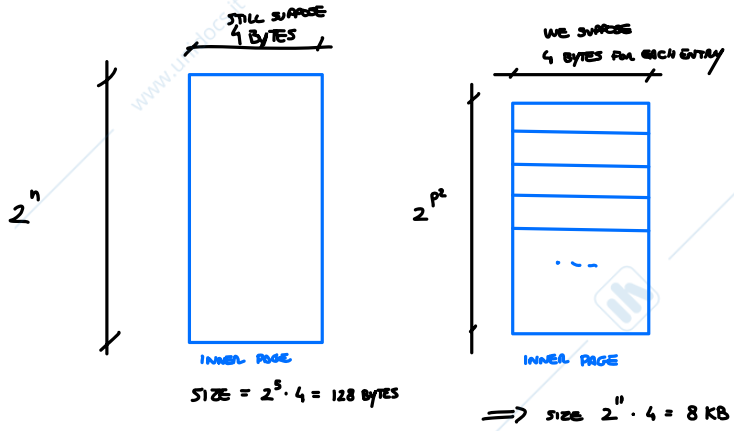
00000000	00050000

5 Exercise 5 - Paging

Consider a virtual memory system with paging, in which the Bytes are addressed. The system has a TLB (Translation Look-aside Buffer), on which a "hit ratio" of 99% is experimentally measured. The page table ("page-table") is created with a two-level scheme, in which a 32-bit logical address is divided (from MSB to LSB) into 3 parts: p1, p2, d, respectively of 10-bit, 11-bit, and 11-bit. No additional data structures (such as hash tables or inverted page tables) are used to speed up access.

- Tell us what is meant by "hit ratio"
- Illustrate the layout of the page table and its overall size for a P1 process having a virtual address space of 100MB
- Calculate external and internal fragmentation for process P1 (see the previous point).
- Assuming that the RAM memory has an access time of 300 ns, calculate the effective access time (EAT) for the proposed case (hit ratio = 99%)





- EXTERNAL FRAGMENTATION IS Ø WITH PAGING
- THE PAGE SIZE IS 2^{11} BYTES = 2 KB SO THE INTERNAL FRAGMENTATION MAY BE
  - AVG: ½ 2KB
  - WORST CASE: 2KB − 1 BYTE

!!! FOR P1 IS Ø BECAUSE 2KB IS A MULTIPLE OF THE VIRTUAL SPACE FOR P1 (100 MB) !!!

- EAT = $(0.99 \cdot 300) + (0.01 \cdot 3 \cdot 300)$ ns
  ↓
  OUTER + INNER + RAM

IN THIS CASE WE KNOW THAT THE VIRTUAL ADDRESS SPACE IS ON 27 BITS (2^{27} = 127 MB)
SINCE WE HAVE 32 BITS AVAILABLE WE CONSIDER
d = 11, p1 = 11 ... SO p2 HAS ONLY 5 BITS THAT ARE NEEDED (32 − 27 = 5)
⟹ n = 5

6 Exercise 6 - Paging

Describe the advantages and disadvantages of an inverted page table (IPT) compared to a standard (possibly hierarchical) page table.

Consider a process having a virtual addressing space of 32 GB, equipped with 8GB of RAM, on a 64-bit architecture (in which the Byte is addressed), with the management of paged memory (1KB pages/frames). You want to compare a standard page table-based solution (one table for each process) and an IPT-based solution.

Calculate the page table size (at one level only) for the process and the IPT. Let's assume that the pid of a process can be represented on 16 bits. Use 32 bits for page and/or frame indexes. Finally, using the proposed IPT (32 bits for a page/frame index), what is the maximum possible size for the virtual address space of a process?

VIRTUAL ADDRESS SPACE : 32 GB

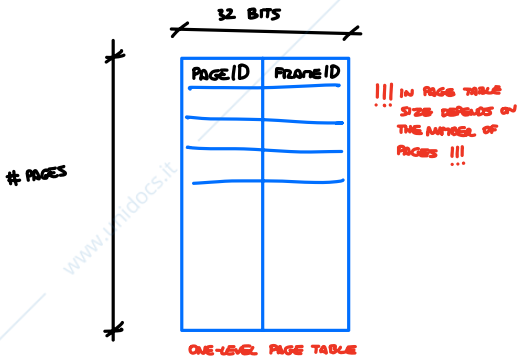
RAM (main physical memory) size : 8 GB

PAGE + FRAME ON 32 BITS

FRAME STILL ON 32 BITS

PID ON 16 BITS

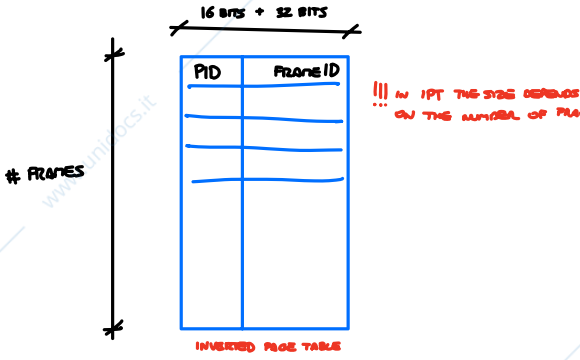
PAGE / FRAME SIZE : 1 KB



!!! IN PAGE TABLE SIZE DEPENDS ON THE NUMBER OF PAGES !!!

$$\#\text{PAGES} = \frac{\text{VIRTUAL ADDRESS SPACE}}{\text{PAGE SIZE}} = \frac{32\text{ GB}}{1\text{ KB}} = 32\text{ M}$$

$$\text{SIZE} = \#\text{PAGES} \cdot 32\text{ BITS} = 128\text{ MB}$$



!!! IN IPT THE SIZE DEPENDS ON THE NUMBER OF FRAMES !!!

$$\#\text{FRAMES} = \frac{\text{PHYSICAL ADDRESS SPACE}}{\text{PAGE SIZE}} = \frac{8\text{ GB}}{1\text{ KB}} = 8\text{ M}$$

$$\text{SIZE} = \#\text{FRAMES} \cdot 48\text{ BITS} = 48\text{ MB}$$

TO COMPUTE THE MAXIMUM SIZE OF VIRTUAL SPACE WE COMPUTE THE MAXIMUM NUMBER OF PAGES, THAT IS 2^{32} AND WE MULTIPLY IT BY THE SIZE OF ONE PAGE (1 KB)

$\Rightarrow$ MAXIMUM SIZE = 4TB

7 Exercise 7 - Page replacement

Consider the following sequence of references in memory in the case of a 1000-word program: 261, 409, 985, 311, 584, 746, 632, 323, 470, 915, 858.

Determine the string of page references, assuming their size is 200 words. Use a second-chance set page replacement algorithm (with a limit of 3 available frames). Suppose the references began when the program started, with the 3 frames allocated to pages n. 4, 3, and 1, and the FIFO queue containing, in order, 40,31,10 (The subscript represents the reference bit). Determine which and how many page faults (accesses to pages not present in the resident set) will occur (the display of the resident set is required after each reference). Suppose the reference bit of a page is initialized to 0 at a page fault.

FIFO

	1	2	4	1	2	3	3	1	2	4	4
	261,	409,	985,	311,	584,	746,	632,	323,	470,	915,	858.
4_0	4_0	2_0	4_0	4_0	4_0	3_0	3_1	3_1	3_1	3_0	3_0
3_1	3_1	3_1	3_0	3_0	2_0	2_0	2_0	2_0	2_1	4_0	4_1
1_0	1_1	1_1	1_0	1_1	1_1	1_0	1_0	1_1	1_1	1_0	1_0
		X	X		X	X				X	

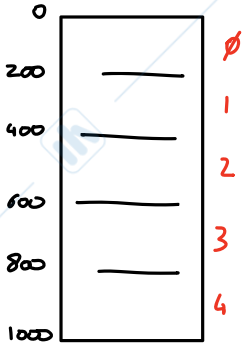
PAGE REFERENCE STRING

INITIAL SITUATION IS GIVEN

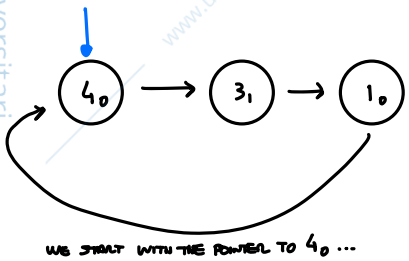
PAGE FAULT... PUT IMMEDIATELY THE 'X' AND THEN SEARCH FOR THE VICTIM PAGE

5 PAGE FAULTS

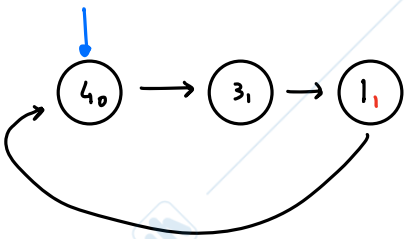
PAGES NUMBER: $\emptyset, 1, 2, 3, 4$

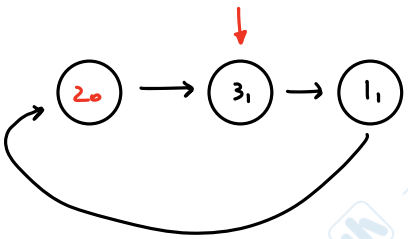


• WHEN WE SWAP-IN A PAGE THE REFERENCE NUMBER IS = 0

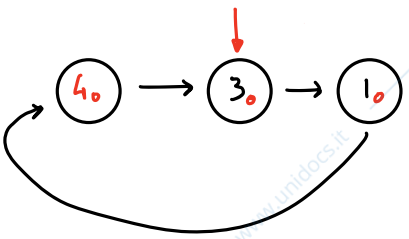


WE START WITH THE POINTER TO 4_0 ... WRITTEN IN THE TEXT

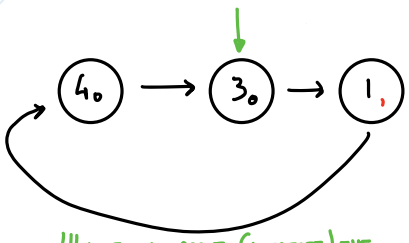




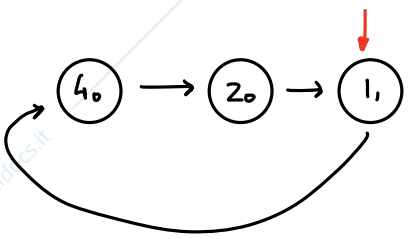
!!! UPDATE POINTER AFTER SWAP IN/OUT !!!

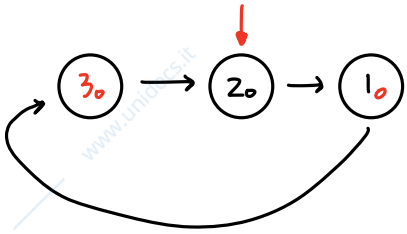


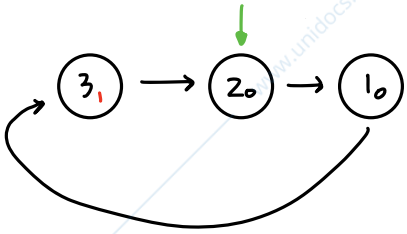
!!! UPDATE POINTER AFTER SWAP IN/OUT !!! ... EVEN WHEN WE MODIFY REFERENCE BIT

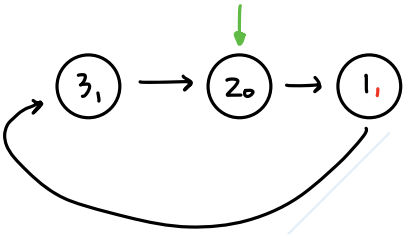


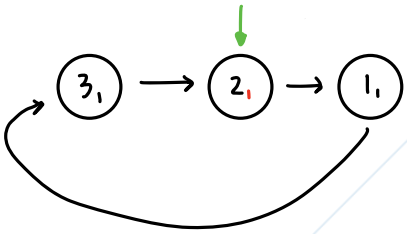
!!! WHEN I UPDATE (INCREASE) THE REFERENCE BIT THE POINTER REMAINS THERE !!!

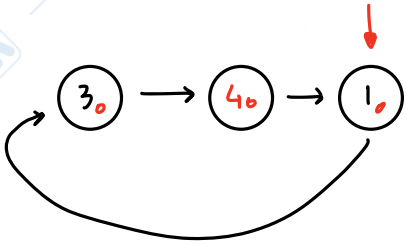


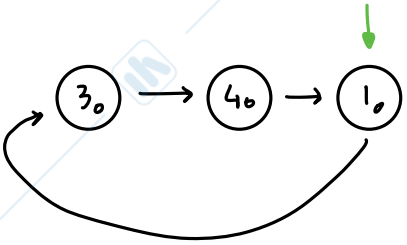












8 Exercise 8 - Page replacement

Consider the following sequence of references in memory in the case of a program in which, for each access (addresses in hexadecimal, the Byte is addressed), it is indicated whether it is reading (R) or writing (W): R 3F5, R 364, W 4D3, W 47E, R 4C8, W 2D1, R 465, W 2A0, R 3BA, W 4E6, R 480, R 294, R 0B8, R 14E.

A. Assuming that both physical and logical addresses are on 12 bits, that paging with pages of 128 Byte is used, and that the maximum address the program can use is C10, compute how many pages are present in the address space of the program and calculate the internal fragmentation.

B. Determine the string of the page references (Switching from hexadecimal to binary is recommended to determine the page number correctly and, if necessary, the displacement/offset). An LRU (Least Recently Used) page replacement algorithm is used. Assume that 3 frames are available at physical addresses (expressed in hexadecimal) 780, A00, B00. The computation (after each access) of the resident set (the physical frames containing logical pages) is required.

C. Determine which and how many page faults (accesses to pages not present in the resident set) will occur. Finally, tell which physical addresses the accesses are made to (among those listed above) R 3F5, W 4D3, R 3BA

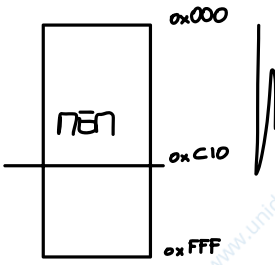
THEY REFERS TO THE LOGICAL ADDRESS SPACE OF OUR PROGRAM

(A)

PHYSICAL AND LOGICAL HAS 12 BITS... SIZE OF 2^{12} B (12 ADDRESS BITS + BYTE ADDRESSED) $\Rightarrow$ SIZE = 4 KB

SIZE OF PAGE : 128 B

!!! ADDRESSES [0, C10] !!!



SIZE = (0xC11) · 1 B = 3089 B

SO WE CONSIDER ALSO THE PAGE THAT IS OUTSIDE THE GIVEN RANGE

NUM. PAGES = $\frac{\text{SIZE OF THE POSSIBLE MEMORY}}{\text{SIZE OF ONE PAGE}} = \frac{3089\ B}{128\ B} = 25$ PAGES (!!! TAKE THE UPPER INTEGER... !!!)

INTERNAL FRAGMENTATION : WE CAN USE 25 PAGES

= (25 · 128) − 3089 = 111 B OF INT. FRAGMENTATION

(B)

THE REFERENCE STRING CONTAINS THE NUMBER OF THE PAGE

or writing (W): R 3F5, R 364, W 4D3, W 47E, R 4C8, W 2D1, R 465, W 2A0, R 3BA, W 4E6, R 480, R 294, R 0B8, R 14E.

① WE KNOW THAT 1 PAGE IS 128 BYTES. SO THE OFFSET IS ON 7 BITS

② WE HAVE 12 BIT ADDRESS

$\Rightarrow$ 5 MSBs FOR THE PAGE NUMBER

e.g. 3F5 → 0011 1111 0101

LRU

00111 → PAGE 7

	7	6	9	8	9	5	8	5	7	9	9	5	1	2
(780) 10	7	7	7	8	8	8	8	8	8	9	9	9	9	2
(A00) 20		6	6	6	6	5	5	5	5	5	5	5	5	5
(B00) 22			9	9	9	9	9	9	7	7	7	7	1	1
	X	X	X	X		X			X	X			X	X

RESIDENT SET

9 PAGE FAULTS

7 NOT IN

PUT 8 INSTEAD OF 7 BECAUSE 7 WAS THE LEAST RECENTLY USED

... EVERY TIME CHECK FOR THE LEAST RECENTLY USED LOOKING THE RECENT PAST ...

(c)

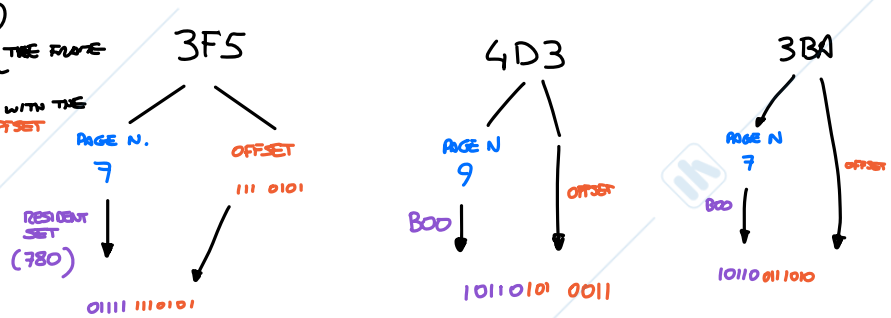
GIVEN THESE LOGICAL ADDRESSES R 3F5, W 4D3, R 3BA

WE CAN COMPUTE THE PHYSICAL ADDRESS

SIMPLY :

1. LOOKING IN WHICH RESIDENT SET HAVE BEEN STORED (IS IN PHYSICAL MEMORY)
2. COMPUTE THE FRAME NUMBER
3. COMBINE WITH THE SAME OFFSET

COMPUTE PHYSICAL ADDRESSES



9 Exercise 9 - Page replacement

Consider the following sequence of references in memory in the case of a 4K word program in which, for each access (addresses in hexadecimal), it is indicated whether it is a read (R) or a write (W):
W 3A1, R 3F5, R A64, W BD3, W 57E, R A08, R B85, W 3A0, R A1A, W A36, R B20, R 734, R AB8, R C4E, W B64.

Determine the string of page references, assuming their size is 512 words. An Enhanced Second-Chance page replacement algorithm is used. The modification bit (to be initialized to 0 at the first access to a new page after the relative page fault) is added to the reference bit (modify bit). Assume that a page is always modified in correspondence with a write, that 3 frames are available and that the algorithm operates with the following criterion: given the pointer to the current page (according to the FIFO strategy), a first turn is made , without changing the reference bit on the pages to locate the victim (the order of priority is (reference, modify): (0,0), (0,1), (1,0), (1,1)) ; once the victim has been determined, a second turn is made to reset the reference bits of the "saved" pages (between the starting position and the victim). (1) (2)

Determine which and how many page faults (accesses to pages not present in the resident set) will occur. The resident set is requested to be displayed (after each access), indicating the reference and modification bits for each frame. Number the pages starting from 0.

For this question, use the following scheme to carry out the exercise, indicating in the first line the string of references to pages (represented by choice in hexadecimal or decimal), in the second Read or Write, in the following three (representing the 3 frames of the resident set), the pages allocated in the corresponding frames, indicating the bits for each (reference, modify). Also indicate (by underlining, circling or placing an arrow) which page is at the top of the FIFO. In the last line, indicate the presence or absence of a Page Fault.

• PAGE SIZE = 512 WORDS ⇒ 2^9 ⟹ 23 BITS NEEDED

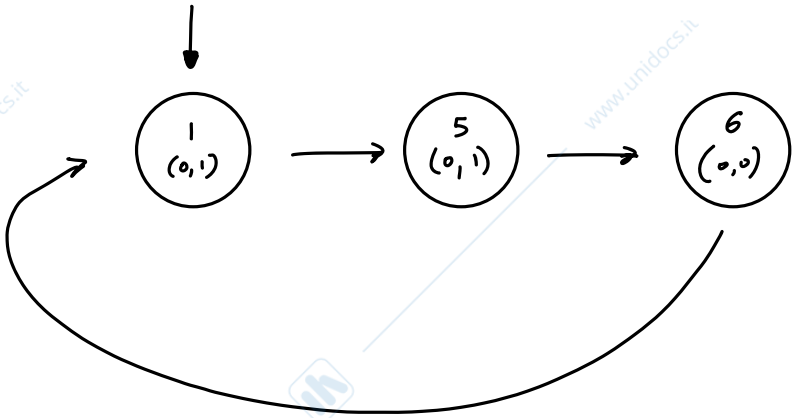
YOU CAN SIMPLY CONSIDER THE FIRST 3 MSBs SINCE XXXX XXXX YYYY = PAGE + OFFSET

1 (001) 1 (001) 5 (101) 5 (101) 2 (010) 5 (101) 5 (101) 1 (001) 5 (101) 5 (101) 5 (101) 3 (011)

W 3A1, R 3F5, R A64, W BD3, W 57E, R A08, R B85, W 3A0, R A1A, W A36, R B20, R 734,
R AB8, R C4E, W B64.
5 (101) 6 (110) 5 (101)

REFERENCES	1	1	5	5	2	5	5	1	5	5	5	3	5	6	5
R/W	W	R	R	W	W	R	R	W	R	W	R	R	R	R	W
	1 (0,1)	1 (1,1)	1 (1,1)	1 (1,1)	1 (1,1)	1 (1,1)	1 (1,1)	1 (1,1)	1 (1,1)	1 (1,1)	1 (1,1)	1 (0,1)	1 (0,1)	1 (0,1)	1 (0,1)
			5 (0,0)	5 (1,1)	5 (1,1)	5 (1,1)	5 (1,1)	5 (1,1)	5 (1,1)	5 (1,1)	5 (1,1)	5 (0,1)	5 (1,1)	5 (0,1)	5 (1,1)
					2 (0,1)	2 (0,1)	2 (0,1)	2 (0,1)	2 (0,1)	2 (0,1)	2 (0,1)	3 (0,0)	3 (0,0)	6 (0,0)	6 (0,0)
PAGE FAULT	X		X		X							X		X	

(0,0)(0,1)(1,0)(1,1)



5 PAGE FAULTS

10 Exercise 10 - Page replacement

Given the string of references to pages 3, 4, 1, (3, 1, 4, 4, 3, 1, 1) *10, where the syntax (...) *n indicates that the string in brackets is repeated/iterated n times (the string can, for example, derive from an iterative construct).

Use a working-set page replacement algorithm (exact version) with window $\Delta = 3$. Suppose you call page-out the removal of a page from the resident set (as it leaves the working set). The references and the resident set are displayed in the following diagram after each reference, indicating the page faults (accesses to pages not present in the resident set) and the page-outs. Analyze the first two iterations of the substring repeated 10 times.

ATTENTION: each row of the resident set represents a frame. Therefore a page present in a frame cannot change the row when it remains in the resident set. In the case of page fault without page-out, use the first free frame from above. In the case of page-out and (simultaneous) page-fault, the frame just freed (from page-out) is reused. If page-out and page-fault are related to the same page, the replacement algorithm takes them into account, avoiding both page-out and page-fault.

$\Delta = 3$

EVERY TIME CHECK THE WINDOW CONSIDERING THE CURRENT REFERENCE ALSO

TIME	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16				
REFEREN.	3	4	1	3	1	4	4	3	1	1	3	1	4	4	3	1	1				
RESIDENT SET	3	3	3	3	3	3		3	3	3	3	3	3		3	3					
		4	4	4		1	1		1	1	1	1	1	1		1	1				
			1	1	1	4	4	4	4				4	4	4	4	4				
PAGE FAULT	X	X	X			X		X	X				X		X	X					
PAGE OUT					4		3	1		4				3	1		4				

I KEEP IN MEMORY ONLY THE WORKING SET THAT I NEED

7 SWAP OUT

9 PAGE FAULT