SURNAME		NAME		Group
ID		Signature		

- **Keep the exam sheets stapled.**
- **Write your answer inside the reserved space.**
- **Use clear and understandable writing. Answer briefly and precisely.**
- **The exam has 8 questions, everyone has its score specified.**

1. A file system organized in 512-byte blocks, with 2-byte block pointers, has an average block access time of 2 msec. We intend to access a 32 KByte file, assuming that the information contained in the pointer to the first block of data is found in Memory. Memory access time is negligible compared to disk access time. Justify the average time required to access reading byte 4650 of that file for each of the following block allocation methods:

(1,0 point = 0,4 + 0,3 + 0,3)

1

a) Linked allocation

Each block includes the pointer to the next block so that of the 512 bytes of each block only 510 are available for data. The block number that contains byte 4650 is :

Floor(4650 / 510) = 9

You have to read 10 blocks sequentially, from 0 to 9, so the time required is :
10 * 2 = 20 mseg

b) Two level indexed allocation

In two level indexed allocation there is, for each file, a block of pointers that point each of them to another block of pointers that finally point to data blocks. Therefore, two pointer block readings have to be done before the data block can be read, so the time required is :
2 + 2 + 2 = 6 mseg

c) FAT (*File allocation Table*). Consider that the table with all its pointers to blocks is in Main Memory and that the access time to Memory is negligible

S If the table of pointers is in memory then finding the location of the block containing byte 4650 is done without access to disk, in an amount of time much lower than disk access time, so the time required is 2 msec approximately .

2. A process has to write "parent message" string on file "messages.txt" and then create a child that writes "child message" string on the same file. The parent process has to wait for the completion of the child process to read ALL the contents of file "messages.txt" file through its standard input and write it on its standard output. In addition, the parent process has to end by closing all opened file descriptors. In order to perform this actions complete the C program in section a) with the necessary POSIX calls, one on each line with an underlined number.

NOTE: Use open(), read(), write(), close() and dup2() when required. Using lseek() call is not allowed.

(1,2 points = 0,8 + 0,4)

a)

```
1   #include <all_needed>
2   #define SIZE 50
3   int main( int argc, char **argv ){
4     int fd1, fd2, nbytes;
5     char buffer[SIZE];
6     mode_t fd_mode = S_IRWXU;     // file permissions
7     char *parent_message = "parent message \n";
8     char *child_message = "child messages \n";
9     fd1 = open("messages.txt", O_TRUNC|O_CREAT|O_RDWR, fd_mode); //complete open()
10    write( fd1 , parent_message, strlen(parent_message));      //complete write()
11    if (fork() == 0) { // child
12      write(fd1, child_message, strlen(child_message));
13      close(fd1);
14      exit(0);  }
15    wait(NULL);
16    fd2 = open("messages.txt", O_RDONLY);
17    dup2(fd2, STDIN_FILENO);
18    while(1) {
19      nbytes = read( 0 , buffer, strlen(child_message));       //complete read()
20      if (nbytes > 0)
21        write(1, buffer, nbytes);
22      else
23        break;
24    }
25    close(fd1);
26    close(fd2);
27  }
```

b) Fill in the file descriptor table of the child process after executing line 12 and the parent process after line 20. The tables have to comply with the requirements and implementation on section a)

Child's file descriptor table after line 12	
0	STDIN
1	STDOUT
2	STDERR
3	"messages.txt"
4	
5	

Parent's file descriptor table after line 20	
0	"messages.txt"
1	STDOUT
2	STDERR
3	"messages.txt"
4	"messages.txt"
5	

3. Program */usr/bin/passwd* allows changing user passwords. For its execution this program needs to read and write files */etc/passwd* and */etc/shadow*. Program */usr/bin/chage* allows changing the expiration time of a password saved in */etc/shadow*. This implies reading file */etc/passwd* file and reading and writing file */etc/shadow*. Consider the following (partial) content of these two directories: **(1,2 points = 0,8 + 0,4)**

directory *./usr/bin*:

```
i-node permissions links  user   group   size    date       name
655364 drwxr-xr-x  2     root    root   36864 nov 18 14:02 .
655363 drwxr-xr-x 12     root    root    4096 jul 20  2018 ..
656249 -rwxr-sr-x  1     root    shadow 71816 mar 22  2019 chage
658048 lrwxrwxrwx  1     root    root       5 may 20  2019 gcc -> gcc-7
657101 lrwxrwxrwx  1     root    root      22 may  8  2019 gcc-7 -> x86_64-linux-gnu-gcc-7
657839 -rwsr-xr-x  1     root    root   59640 mar 22  2019 passwd
655397 -rwxr-xr-x  2     root    root 2097720 nov 19  2018 perl
655397 -rwxr-xr-x  2     root    root 2097720 nov 19  2018 perl5.26.1
657867 -rwxr-xr-x  1     root    root 1010624 may  8  2019 x86_64-linux-gnu-gcc-7
```

directory *./etc:*

```
808275 -rw-r--r--  1     root    root         1812 jul 16  2018 passwd
800878 -rw-r-----  1     root    shadow       1041 jul 20  2018 shadow
```

3 **a)** Indicate whether the execution by the specified user of the following commands would work without error. In case of success justify what are the permissions that are being checked and, in case of error, what is the permission that fails and why .

(UID, GID)	COMMAND	SUCCESS	EXPLANATION
(eva, fso)	`/usr/bin/passwd`	Yes	/user/bin/passwd has x permission for others and as SETUID bit is active, the process runs as (root, fso). /etc/passwd and /etc/shadow files can be read and written since they have read and write permissions for root.
(eva, fso)	`chage -M7 eva` (set expiration to 7 days for user eva)	No	SETGID bit is active, so the process runs as (eva, shadow). /etc/passwd file can be read because it has read permission in the others field. But /etc/shadow file cannot be written, because it only has write permission for root.
(eva, fso)	`gcc --help` (get help for gcc)	Yes	gcc goes through a double symbolic link to executable x86_64-linux-gnu-gcc-7. The permissions that are verified are those of the latter, which give others permission to execute, so eva user can execute it .

b) For directory */usr/bin* justify the number of links in its files "." and "perl"

NAME	LINKS	EXPLANATION
.	2	This directory has 2 links, "bin" from his parent and the self reference ".". It has no subdirectory that references it with ".."
perl	2	This file has 2 links: “perl” entry and “perl5.26.1” entry that can be seen as having the same i-node .

4. A disk with 64 MByte capacity, is formatted in MINIX with the following specifications:

- The boot block and the superblock occupy 1 block each
- 32-byte i-node size with 7 direct data zone pointers,1 indirect and 1 double indirect .
- 16-bit data zone pointers
- 16-byte directory entries: 2 bytes for i-node, 14 bytes for name
- 1 zone = 1 block = 1 KByte

(1,6 points = 0,8 + 0,8)

4 **a)** It is formatted by reserving space in the header for a total of 16384 i-nodes (16K i-nodes). Calculate the number of blocks that each header element occupies and the number of block on the data area.

Boot block	Superblock	i-node bit map	Zone bit map	i-nodes	Data zones

1 Block to boot block → block 0
1 Block to superblock → block 1
2 Blocks to i-node bit map → blocks 2 and 3
16 K i-nodes need 16 K bits; 16 Kbits / 1 KByte = 16 Kbits / 8 KBits = 2 blocks
8 Blocks to zone bit map → blocks 4, 5, 6, 7, 8, 9, 10 y 11.
Partition size = 64 MBytes; Nº zones = 64 MBytes / 1 KByte = 64K zonas →
64 Kbits / 8 Kbits = 8 blocks
512 Blocks to i-nodes → blocks from 12 to 523
Nº of i-nodes = 16 K; 16 Ki-nodes * 32 Bytes = 512 KBytes to store the i-nodes
512 KBytes / 1 KByte = 512 blocks
Data area starts at block 524
The header takes: 1 + 1 + 2 + 8 + 512 = 524 blocks
Partition size is 64 MBytes → 65536 Blocks; 65536 - 524 = 65012 blocks to data area

b) Assume that starting from the empty disk, the disk is occupied according to the following sequence of actions :

1. Creation of root directory (/)
2. Creation of regular file ***/fso*** with 514 KByte size
3. Creation of directory ***/Exam***
4. Creation of directory ***/Exam/Final***
5. Creation of regular file ***/Exam/Final/Actualcourse*** with 800 KByte size
6. Creation of hard link ***/EFinal*** to regular file ***/Exam/Final/Actualcourse***

After these actions, explain the following file system values :

- Root directory size (/) in bytes:

Direcotory / has 5 directory entries: "." , ".." , /fso, /EFinal and /Examen
Cada entrada ocupa 16 Bytes, por tanto el directorio ocupa 80 Bytes

- Number of i-nodes occupied in the file system:

5 i-nodes are occupied, one for each of the files or directories created, except the physical link that shares i-node.

- Number of blocks occupied on the data area with references to other blocks:

In this file system, the only files that use blocks with references to other blocks are:
1. File /fso with size of 514 KBytes → 514 data zones are referenced. Therefore, it uses the 7 direct pointers and the SI pointer, which points to **a block** that references 507 data zones.
2. File /Exam/Final/Current Course with size of 800KBytes → 800 data zones required. It uses the 7 direct pointers, the SI pointer that points to a block with 512 references to data zones, and the DI pointer that will use two more blocks of pointers. The first level block will have a single reference to a second level block that will contain 800 - (512 + 7) = 281 references to data zones. Therefore, **3 blocks** of pointers. .
In all 1 + 3 = **4 blocks** of pointers

5. A system manages its 1000 Kilobyte main memory with contiguous allocation with variable partitions and NO compaction. The initial state of main memory if the following:

0 1000KB - 1

OS 100KB	HOLE 150KB	P1 200KB	HOLE 100KB	P2 100KB	HOLE 350KB

The allocation algorithm always allocates processes within a hole by adjusting to the lower addresses (left), leaving the higher hole addresses free.

a) Indicate the **base address** for processes P3 (100KBytes), P4 (400KBytes) and P5 (200KBytes) applying the allocation algorithms: Best Fit, First Fit and Worst Fit. Consider the proposed sequence of events and the former initial memory state on each case. If a process cannot be allocated write DOESN'T FIT and continue with the following process.

(1,2 points = 0,9 + 0,3)

EVENT	**BEST FIT**	**FIRST FIT**	**WORST FIT**
P3 ARRIVES P3 BASE:	450K	100K	650K
P1 ENDS	----	----	----
P4 ARRIVES P4 BASE:	NO CABE	NO CABE	100K
P2 ENDS	----	----	----
P5 ARRIVES P5 BASE:	100K	200K	750K

b) Indicate for Worst Fit algorithm, the remaining gaps (**start address** and **size**) at the end of the previous sequence of events, and the type of fragmentation generated. :

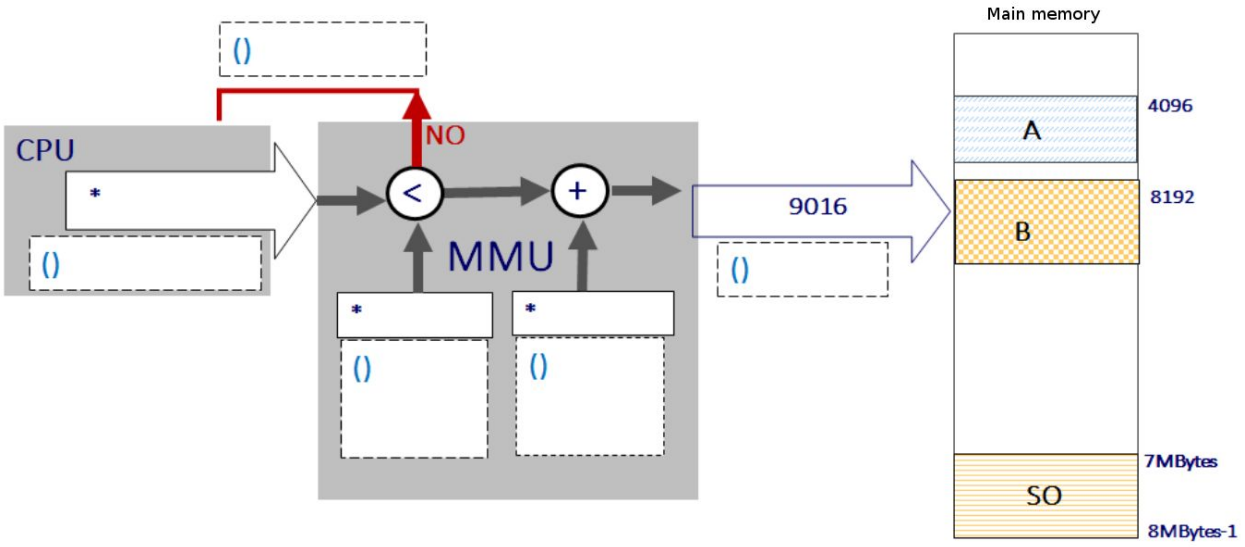
150Kb hole in 500K position and 50KB hole in 950K position .

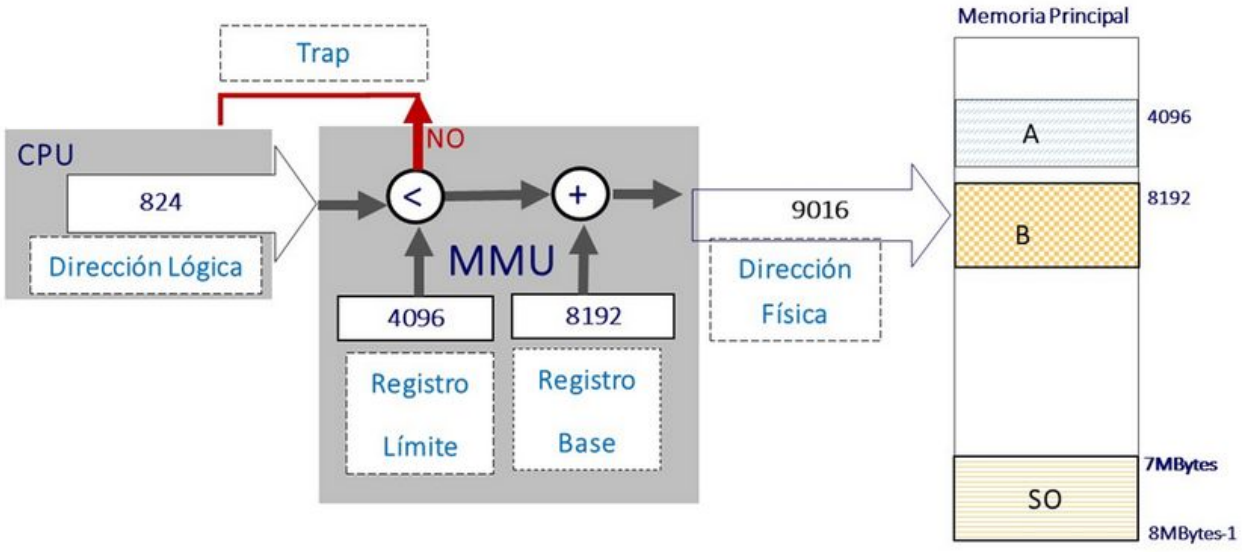
External fragmentation, since the hole sizes are defined by the size of the leaving processes, they may not be large enough to contiguously allocate a new process. This can be solved with compaction.

6. Consider a system with 8 MByte Main Memory in which two processes A (size 2 KByte) and B (size 4 KByte) are located from the addresses indicated on the figure.

(0,8 points)

6 Assume that process B is currently running and that main memory address 9016 is accessed. The system has a basic MMU and manges memory by contiguous allocation, fill in the values in the boxes marked with * for every element, so that such access can be carried out. Also indicate the name of each element in the boxes next to them marked with **()**.





Lógical address = Physical address - Base register = 9016 - 8192 = 824
Limit register corresponds to the process size: B → 4096
Base register corresponds to the first physical address of the process: B → 8192

7. An 32 bit Intel processor family, works with a paged memory architecture with two level paging and 4 KByte page size. For both paging levels, each page table entry (descriptor) occupies 4 bytes and each page table occupies 4 KBytes. Answer the following questions about this system:

(1.2 points = 0,4 + 0,4 + 0,4)

7

a) Maximum number of pages a process can have.

Maximum number of pages is **2^{20} → 1 Mpages**

32 bit logical addresses:

- 4 KByte pages → 2^2 2^{10} → 12 bits to offset
- 32 - 12 = 20 bits to page id

b) Size in bytes of the first level page table and maximum number of entries (descriptors) that this table can have.

A single First Level Page with 1024 descriptors and a size of 4 KBytes

32 bit logical addresses:

- 4 KByte pages → 2^2 2^{10} → 12 bits to offset
- Each page table occupies 4 KBytes and descriptors of 4 Bytes → 4 KBytes / 4 Bytes = 1024 descriptors on each page
- 1024 descriptors → 10 bits to first level page id

c) For a program that occupies 100 MBytes, obtain the number of descriptors it needs at each paging level and the amount of memory consumed by its page tables.

Program size = 100 MBytes → 100 MBytes / 4 KBytes = 25 Kpages → 25 Kdescriptors

Each page table occupies 4KBytes and contains 1024 descriptors → 25 second level page tables with 1024 descriptors each + a first level page table with only 25 descriptors

8. Consider a system with demand paging, 4 KByte pages and 24-bit logical and physical addresses. In this system two processes A and B are in execution. The system assigns to them 5 frames that they share with global scope second chance replacement policy. Both processes have a size of 4 pages (from 0 to 3). The relationship of the assigned frames with the process pages is shown in the following table:

Frame	Process:page	Load time	Last access time	Reference bit	Valid bit
0	A:0	1	6	1	1
1	B:1	2	15	1	1
2	A:1	7	7	1	1
3	B:2	8	12	1	1
4	A:3	12	14	1	1

(1,8 points = 0,4 + 0,4 + 1,0)

8

a) Knowing that up to instant t = 15 only page faults have occurred without replacement, indicate the content of the entries in the page table of process B, for each of its descriptors, it is not necessary to set the permission bits, neither the times .

N º de Página	Marco	R	V
0	-	-	0
1	1	1	1
2	3	1	1
3	-	-	0

b) From instant t = 16 the CPU issues the following logical addresses:

A:0x002345, A:0x002346, B:0x001B72, B:0x000B32, B:0x000B33,
A:0x000111, A:0x001222, A:0x000111, B:0x002ABC, B:0x002ABD

Assume that a working set scheme is applied, with a window size of 4. Obtain the working sets for processes A and B after completing their last access.

WS(A)= {0, 1, 2}

WS(B)= {0, 2}

Can thrashing happen? Explain your answer

The addition of the working set sizes is 5 that matches the available frames so thrashing will not happen.

c) Calculate the reference string corresponding to the logical address sequence in section b):

A:2, B:1,B:0,A:0,A:1,A:0,B:2

Apply **the second chance algorithm with global scope** for this reference string. Fill in the following table (using the required number of columns, maximum 8), with the evolution of memory frames allocated for processes A and B, from instant t = 16. The first column corresponds to t = 15, when process B accessed its page 1. Indicate in each table cell the process, the page and the R bit value after making the access and when there is a page replacement highlight the box chosen as victim.

marco	B:1 (R)	A:2	B:1	B:0	A:0	A:1	A:0	B:2	
0	A:0 (1)	A:2 (1)	A:2 (1)	A:2 (1)	A:2 (1)	A:2 (1)	A:2 (1)	A:2 (0)	
1	B:1 (1)	B:1 (0)	B:1 (1)	B:1 (0)	B:1 (0)	B:1 (0)	B:1 (0)	B:2 (1)	
2	A:1 (1)	A:1 (0)	A:1 (0)	B:0 (1)	B:0 (1)	B:0 (1)	B:0 (1)	B:0 (1)	
3	B:2 (1)	B:2 (0)	B:2 (0)	B:2 (0)	A:0 (1)	A:0 (1)	A:0 (1)	A:0 (1)	
4	A:3 (1)	A:3 (0)	A:3 (0)	A:3 (0)	A:3 (0)	A:1 (1)	A:1 (1)	A:1 (1)	

Number of page replacements: 5