SURNAME		NAME		Group
ID		Signatura		

- **Keep the exam sheets stapled.**
- **Write your answer inside the reserved space.**
- **Use clear and understandable writing. Answer briefly and precisely.**
- **The exam has 9 questions, everyone has its score specified.**

---

1. A paging system has pages of 4KB, 28-bit logical addresses and 4GB of main memory. In this system a process is being executed and the following table shows the **partial** content (in hexadecimal) of the process page table:

Page	Frame	Bv
0	521F5	1
1	112D2	1
2	43231	1
3		0
4	231F2	1
5	231F3	1
6	231F4	1
...	...	...

(1,0 points = 0,5 + 0,5)

1

a) Compute, in hexadecimal, the physical address generated if the process emits the logical address 0002F1F (in hexadecimal). Explain concisely your answer.

Logical address 0002F1F, has page id 0002 and offset F1F, so the frame id is 43231 and the resulting physical address is 43231F1F.

b) Compute, in hexadecimal, the logical address generated if the process does access to the physical address 231F2288 (in hexadecimal). Explain concisely your answer.

Physical address 231F2288, has frame id 211F2 and offset 288, so the page id is 4 and the logical address is 0004288.

2. An operating system has to manage efficiently the computer physical memory. Think about the problems that memory management has and answer the following items:

(1,0 points = 0,5 + 0,5)

2	**a)** What problem intends to solve the technique of compacting memory and on what type or types of contiguous memory allocation schemes can be applied? It solves the problem of external fragmentation. External fragmentation means that while having enough memory to locate a new process, this space is scattered in non-contiguous holes and therefore the new process can not be allocated. Compaction techniques allow shifting processes located in a position in memory to other positions, then free memory space can be grouped contiguously. These techniques can be applied whenever the processes are relocatable at runtime External fragmentation happens with variable partitions, where the system adjusts the assigned memory to a process to the process size.
	b) What is the utility of the MMU (Memory Unit Manager) and what problems has it to solve? The MMU is responsible for translating logical addresses to physical and solving the problem of relocation and providing protection *Translating logical addresses into physical addresses*. This device allows managing logical address spaces for every process independently of the physical address space, so processors emit logical addresses at runtime that are translated by the MMU (Memory Manager Unit) into physical addresses. *Relocation problem.* The processes must be able to run properly regardless of where they are located in main memory. This must be true even if this location changes during the process execution. To achieve this it is important to consider the logical spaces of each process separately, so that these spaces do not depend on the specifics of the machine or the actual state of main memory. *Providing protection.* The MMU is responsible for verifying that the addresses issued by processes belong to their own space as well as checking that the access type (R, W) is correct, otherwise it sends an exception to the processor.

3. Considering the memory map of a UNIX process, indicate whether the following statements are true(T) or false(F):

(0,6 points)

3	STATEMENT	T/F
	The uninitialized data region generated when running a program, is a region without support in any file.	T
	The code region with support on the executable file of a process, has read, write, and execute permissions.	F
	When a process maps a file into memory a new region is created in the memory map of this process that is accessed using input/output calls. .	F
	The use of dynamic libraries is based on the technique of mapping libraries in memory during execution of the process that uses them.	T

4. Consider an operating system that manages paged virtual memory with 4 KB pages. Free frames are assigned in ascending order of physical addresses using local replacement policy. The system assigns three

frames to a new process (frames 0 to 2) and during the process execution the following logical addresses are issued (in hexadecimal):

400000, 400014, 800000, 800024, 600030, 600034, 800280, 40060c, 200000, 400c10, 600f24

Indicate which pages stay in physical memory after the last access and what is the last physical address that is accessed in the following cases. (Indicate with '—' both the free frames or the busy frames with an invalid page).

(2,0 puntos = 0,5 + 0,5 + 0,5 + 0,5)

4

a) LRU

	400	800	600	800	400	200	400	600
0	***400***	400	400	400	*400*	400	*400*	400
1	—	***800***	800	*800*	800	800	800	***600***
2	—	—	***600***	600	600	***200***	200	200
	f	f	f			f		f

Last physical address: 1f24

b) Second chance

	400	800	600	800	400	200	400	600
0	***400***	400	400	400	*400*	***200***	200	200
1	—	***800***	800	*800*	800	800	***400***	400
2	—	—	***600***	600	600	600	600	*600*
	f	f	f			f	f	

Last physical address: 2f24

c) Optimal

	400	800	600	800	400	200	400	600
0	***400***	400	400	400	*400*	400	400	400
1	—	***800***	800	*800*	800	***200***	200	200
2	—	—	***600***	600	600	600	600	600
	f	f	f			f		

Last physical address: 2f24

d) Consider applying FIFO and after the last access there is a call to *exec* then the process does the following logical accesses:

400000, 400010, 600020

	Exec	400	600
0	—	***400***	400
1	—	—	***600***
2	—	—	—
		f	f

exec call invalidates the pages

Last physical address: 1020

5. In a two level paging system with 32-bit physical and logical addresses, page size is 4KB and the second level page tables are allocated in main memory occupying one frame each one.

(1,0 points = 0,5 + 0,5)

5

a) Determine the number of entries that will have every second level page table. Consider that each table entry or page descriptor includes 12 control bits like Bv, Br, Bm, R, W, X, etc.

Every second level table entry occupies: 20 bits for the frame id (there are $2^{32}/2^{12} = 2^{20}$ frames) + 12 control bits (Bv, Br, etc.) = 32 bits -> 4 bytes per entry

So there are 4KB/4B = 1K entries (1024).

b) Describe the structure of the logical addresses and the physical addresses in this system, and the size in bits of every address field .

Logical addresses:

- 12 bits for the offset (page size is 4KB)
- 10 bits for the second level table index (1K = 2^{10} entries per table)
- 10 bits = 32-12-10 for the first level table index

```
-------------------------------------------------------------
|   p1 (10 bits)   |  p2 (10 bits)   |   offset (12 bits)      |
-------------------------------------------------------------
```

Physical addresses:

- 12 bits for the offset
- 20 bits for the frame id

```
-------------------------------------------------------------
|            frame (20 bits)       |       offset (12 bits) |
-------------------------------------------------------------
```

6. The content of the file descriptor tables corresponding to three processes at a given time are the following:

Initial page table	
0	STDIN
1	STDOUT
2	STDERR
3	
4	
5	

P1 page table	
0	STDIN
1	STDOUT
2	STDERR
3	"result"
4	"fd_pipe[0]"
5	"fd_pipe[1]"

P2 page table	
0	STDIN
1	"fd_pipe[1]"
2	STDERR
3	
4	
5	

P3 page table	
0	"fd_pipe[0]"
1	"result"
2	STDERR
3	
4	
5	

Complete the following C code fragment with POSIX primitives required to create the three processes and to match the content shown of their descriptor tables, at points marked as /*** Table ...***/

(1.0 points)

6

```
int fd_pipe[2]; /* pipe descriptor */
int fd;     /* regular file descriptor */

/*** Initial descriptor table ***/
  fd=open("result",O_WRONLY |O_CREAT|O_TRUNC,0666);
  pipe(fd_pipe);

 /*** P1 descriptor table ***/
  if (!(pid=fork())) {
    dup2(fd_pipe[1],1);
    close(fd_pipe[0]);
    close(fd_pipe[1]);
    close (fd);

 /*** P2 descriptor table ***/
    execlp("/bin/cat", "cat", "fich1", NULL);
  }
  if(!(pid=fork())){
    dup2(fd_pipe[0],0);
    dup2(fd,1);
    close(fd_pipe[0]);
    close(fd_pipe[1]);
    close(fd);

  /*** P3 descriptor table ***/
    execlp("/usr/bin/wc", "wc", "-l",NULL);
  }
  close(fd_pipe[0]); close(fd_pipe[1]);
  close (fd);
  while(pid != wait(&status));
}
```

7. Given the following directory listing on a POSIX system:

```
i-node permissions  links   user      group size       date         name
2021662 drwxr-xr-x   2      sterrasa  fso  4096        dec  9 2015  .
2021611 drwxr-xr-x   8      sterrasa  fso  4096        sep 10 2015  ..
2021663 -rwxr-sr-x   1      sterrasa  fso  1139706     dec  9 2015  copia
2021664 -rw-rw-r--   1      sterrasa  fso  9706        dec  9 2015  f1
2021665 -rw-rw-r--   2      sterrasa  fso  4157        dec  9 2015  f2
2021665 -r--r--rw-   2      sterrasa  fso  4157        dec  9 2015  f3
2021666 lrwxrwxrwx   1      sterrasa  fso     2        dec  9 2015  f4->f1
```

(1.25 points = 0.75 + 0.5)

7

a) The program copy copies the file contents received as the first argument into another file whose name is indicated as the second argument. That is, "copy a b" copies the contents of file *a* to file *b*, if b does not exist it has to create it and then perform the copy. Fill the following table, indicating in case of success what permissions are checked and, in case of failure, which is the permission missed and why.

(UID,GUI)	COMMAND	DOES IT WORK?	EXPLANATION
(pepe, fso)	copia f1 f5	NO	In order to succeed, execution permission is required on "copia", reading on f1 and writing on the directory because f5 is a new file The command fails writing on the directory, because accessing the directory as (pepe, FSO) the group domain applies and it hasn't writing permission.
(sterrasa, fso)	copia f1 f3	NO	In order to succeed execution permission is required on "copia", reading on f1 and writing on f3 The command fails while trying to write on f3, because not eve the owner (sterrasa) has writing permission on f3
(ana, etc)	copia f3 f4	YES	In order to succeed execution permission is required on "copia", reading on f3 and writing on f4. (ana, etc) can execute "copia" (x permission on the third triplet) and that changes their eGID, becoming (ana, fso). With this new eGID, the process has reading permission on f3 and writing on f1, that is the file that is finally accessed by the symbolic link f4

b) Indicate what values from column `links` on the former listing will change if directory entry *f3* is removed (`rm f3`). What values of the same column will change if removing *f4*?. Explain your answers.

When executing command "rm f3", the corresponding directory entry is removed. Also f3 and f2 are hard links to the same content, we know it because they share the inode and also the number of links field is 2. If we eliminate f3, apart from removing the number of entries field of f2 it would appear a 1 indicating that there is only one directory entry to access the contents of the file with inode 2021665.

However when removing directory entry f4, nothing would change in the number of links column, as f4 is a symbolic link to file f1, and the i-node field number of links, symbolic links are not accounted, only physical links.

8. A 32MByte partition has 2MBytes reserved to file system control structures and it is organized into 512-byte blocks, the size of a pointer to block is 32 bits. Obtain:

(1,0 points = 0,5 + 0,5)

8	**a)** Maximum file size using indexed allocation with a one block of pointers per file. Pointers to blocks are 32 bits --> 4 bytes With 32-bit pointers the maximum number of blocks that can be addressed is: 2^{32}= 4 x 1024 x 1024 x 1024 blocks, so the maximum file size has to be equal or less this number of blocks. Every block has 512 bytes so it contains 512/4 = 128 pointers to blocks. So 128 x 512= 2^7 x 2^9 = 2^{16} = 64 KBytes is the maximum file size
	b) Maximum file size using linked allocation. The free space available to create the file is: 32 MBytes - 2 MBytes = 30 MBytes Every block is 512 bytes, so the total number of disk blocks is: 30 MBytes / 512 Bytes = 30 2^{20} /2^9 = 30 * 2^{11} = 60 x 1024 blocks On every block there are: 512 bytes = 4 bytes for pointer-to-next + 508 bytes file_content → Maximum file content size is: 508 x 60 x 1024 = 30480 KBytes

9. A disk with 6GB capacity, is formatted with a version of MINIX whose sizes are as follows :

- The boot block and superblock occupy one block each.
- The i-node size is 32 bytes (7 direct pointers, 1 indirect, 1 double indirect).
- With 16-bit zone pointers.
- Every directory entry is 16 bytes.
- 1 zone = 1 block = 1 KByte
- The number of reserved i-nodes is 4.096

The scheme of the partition elements is the following:

Boot	Superblock	i-node bi map	Zone bitmap	i-nodes	Data area

Obtain:

a) Number of blocks occupied by the following elements: the i-node bitmap, the zone bitmap and the inodes.

b) The content of the existing directories is shown, assuming that every regular files occupies 10 KBytes, how many inodes are busy? how many data zones are occupied?

1	.
1	..
4	usr
10	bin

4	.
1	..
20	list
12	alumno

12	.
4	..
26	prac

10	.
1	..
35	calc

(1,15 points = 0,5 + 0,65)

9

a)

Nº blocks for i-node map = 4096 / (1024*8) = 0,5 -> 1 blocks

Nº blocks for zone map = (6G / 1024) / (1024*8) = 768 blocks

Nº blocks for i-nodes = (4096*32) / 1024 = 128 blocks

b)

There are 4 directories with i-nodes: 1, 4, 12 and 10, and 3 regular files with i-nodes 20, 26 and 35. In all there are 7 i-nodes busy.

The busy zones can be classified as:

- Directory content: 4 zones
- Regula file content: 30 zones
- Pointers: The 7 direct pointers address 7 KBytes, in order to address the remaining 3KBytes one simple-indirect pointer is required, so one zone of pointers is required for every regular file then 3 zones.

The total number of busy zones is 37.