APELLIDOS		**NOMBRE**		Grupo
DNI		**Firma**		

- **Keep the exam sheets stapled.**
- **Answer only in the reserved area.**
- **Use clear and readable writing. Answer briefly and precisely.**
- **The exam has 8 questions, question 1 scores 1.5 points and question 3 scores 2.5 (0.75+ 1+ 0.5+0.25) points, the remaining questions score 1 point each.**
- **Exam time: 2 hours**

Let a system with variable partitions and the following allocation state:

Libre 20K	Ocupado	Libre 10K	Ocupado	Libre 5K	Ocupado	Libre 15K	Ocupado	Libre 25K

Libre = Free, Ocupado = Busy

Where the free gap list is kept ordered towards increasing addresses:

Gap list →20K→10K→5K→15K→25K

Then four processes ask for memory allocation in the following order P1, P2, P3 y P4. Considering that their sizes are P1(10K), P2(15K), P3(7K) y P4(14K), indicate the memory state (free and busy gaps) as well as the gap list when the following techniques are applied:

a) Best fit
b) Worst fit
c) First fit

1.5 points

a)Best fit and gap list

Gap list 13K→5K→11K

P3/Free 13K	Busy	P1	Busy	Free 5K	Ocupado	P2	Ocupado	P4/Free 11K

b)Worst fit and gap list

Gap list 5K→10K→5K→8K→1K

P2/Free 5K	Ocupado	Libre	Ocupado	Free 5K	Ocupado	P3/Free 8K	Ocupado	P1/P4/Free 1K

c) First fit and gap list

Gap list 3K→10K→5K→11K

P1/P3/Free 3K	Ocupado	Free 10K	Ocupado	Free 5K	Ocupado	P2	Ocupado	P4/Free 11K

Considering a paged memory system with 22 bit logical addresses and 2 Kbyte page size, answer to the following questions:

a) What is the maximum number of pages that a process can allocate.
b) If the system uses multilevel paging with two levels, and the first level has 8 entries, how many second level tables requieres a process with 1024 pages? Explain your answer.

1.0 points

2

a)
2K -> 11 bits so there are 11 bits for page index -> 2K pages

b)
8 entries -> 3 bits in the first level so it remain 8 bits in the second level then it has 256 entries.
1000/256 = 4 tables.

Let a virtual memory system based on segmentation with paging and the following features:

- 16 bit logical addresses
- Physical memory with 1024 frames
- Every process has a máximum of 4 frames
- 256 words pages
- Maximum of 16 segments per process

Currently in memory there is a single process P that occupies frames from 0 to 3, as shown in figure (format segment/page):

Frame	Segment, Page
0	S0,4
1	S1,9
2	S0,5
3	S2,1

a) Describe the logical address and physical address formats explaining its structure, bit distribution and sizes

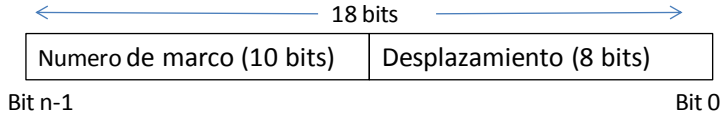
0.75 points

b) Using OPTIMAL page replacement algorithm with LOCAL context, show the evolution of main memory along the following references:
0x1358 0x056D 0x1901 0x216E 0x2178 0x3BD8 0x0567 0x1345 0x1367
0x0590 0x0500 0x0D33 0x0D23 0x1340 0x3B34 0x3B21 0x21AB

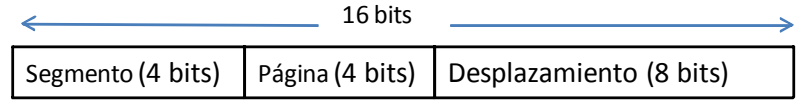
1.0 point

3 a)

a)
Page size is 256 words = 2^8, the offset length is 8 (for both frames and pages)
Because there are 1024 frames 10 bits are required for frame index. So physical addresses are 18 bit, 10 for frame (número de marco) and 8 for offset (desplazamiento).



Logical addresses are 16 bits: 4 for segment (at most), 4 for page (página) and 8 for offset (desplazamiento).



3 b)

Total number of page faults = 3 page faults with replacement

Frame		**S1,3**	S0,5	S1,9	S2,1	**S3,B**	S0,5	S1,3	S0,5	**S0,D**	S1,3	S3,B
0	S0,4	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3
1	S1,9	S1,9	S1,9	S1,9	S1,9	S3,B	S3,B	S3,B	S3,B	S3,B	S3,B	S3,B
2	S0,5	S0,5	S0,5	S0,5	S0,5	S0,5	S0,5	S0,5	S0,5	S0,D	S0,D	S0,D
3	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1

c) Obtain the physical address that will be generated from logical address 0x21AB, if the physical memory allocation is as follows:

Marco	Seg, Pág
0	S0,4
1	S1,9
2	S0,5
3	S2,1

0.5 points

3 c)

Address 0x21AB corresponds to segment 2, page 1 and offset AB. As page from segment has been allocated in frame 3 the physical address in hexadecimal is 0x003AB

d) Explain if, for the same references, the page replacement algorithms FIFO, LRU and 2nd chance will produce a greater or lesser number of page faults

0.25 points

3 d)

The optimal algorithm is based on choosing as a victim the page that will take longer to be referenced, and so is the one that produces the fewest number of faults. This algorithm can not be implemented in reality since we cannot predict what will be the following references.
.

Given the following page reference set done by processes A, B y C (described as process/page number):

A1,A2,A1,A3,B1,B2,B2,B2,B3,B4,B3,B5,C2,C3,C4,A1,C5,C6,C4,
C3,B5,B4,B3,A2,A3,A1,B4,B5, B3,C2,C3,C4,B2,B1,A5,A6,A7

a) Get the working set at the last reference for a working set estimation window of 6.
b) From the previous result explain if thrashing can happen with 8 main memory frames.

1.0 point

4

a) Working set for a window of 6

First we get the references for every process
A= A1,A2,A1,A3,A1,A2,A3,A1,A5,A6,A7
B=B1,B2,B2,B2,B3,B4,B3,B5,B5,B4,B3 B4,B5, B3,B2,B1
C= C2,C3,C4,C5,C6,C4,C3,C2,C3,C4

WS(A) = A1,A2,A3,A5,A6,A7 -> WSS(A) = 6

	WS(B) = B1,B2,B3,B4,B5 -> WSS(B) = 5 *WS(C) = C2,C3,C4,C6 -> WSS(C) = 4*
	b) *WSS = 6 + 5 + 4 = 15 > Frame number = 8* *All the working pages do not fit in memory so thrashing can happen*

Considering the memory map of a process in UNIX, mark the following sentences as TRUE(T) or FALSE(F):

1.0 points

5	T/F	SENTENCE
	F	All the memory map regions have support on a file
	T	The code region with support on the process executable file has read and execution permissions but not write permission
	T	In order a process to map a file into memory it has to call previously to "open" on that file
	F	When a process P maps successfuly a file f into its memory map it gets a file descriptor associated to f that allows P t oread/write from/to f using system calls read() and write()
	T	When a process calls to exec() then in its memory map changes the code región that will have support on the new executable file
	T	When a process creates a child process calling to fork() the child memory map is identical to the one of the parent
	F	An executable file of a program that uses library functions occupies more disk space if libraries are linked dynamicaly than if they are linked statically

Given the following C code fragment with POSIX primitives, that corresponds to a process that when running inherits from his father a table of file descriptors with the three standard file descriptors

```
.....
 int tubo[2];
 int fd;

  pipe(tubo);
 /*** Fill descriptor table for P1

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

   if(!(pid=fork())){
        dup2(tubo[0],0);
        fd=open("result",O_WRONLY |O_CREAT|O_TRUNC,0666);
        dup2(fd,1);
        close(tubo[0]);
        close(tubo[1]);
        close(fd);
  /*** Fill descriptor table for P3
        execlp("/usr/bin/wc", "wc", "-l",NULL);
   }
  close(tubo[0]);
```

```
   close(tubo[1]);
   while(pid != wait(&status));
}
```

a) Fill the file descriptors table for every process involved in the moments marked with the comment "Fill descriptor table …"

b) Indicate what would be the equivalent command line (commands, pipes, redirections, etc.) that will be executed when the former code runs.

6

a)

P1 table

0	STDIN
1	STDOUT
2	STDERR
3	tubo[0]
4	tubo[1]
5	
6	

P2 table

0	STDIN
1	tubo[1]
2	STDERR
3	
4	
5	
6	

P3 table

0	tubo[0]
1	result
2	STDERR
3	
4	
5	
6	

b)

```
$ cat fich1 | wc -l >result
```

A partition of 32MBytes where 2Mbytes are busy with the file system structures, is organized into blocks of 512 bytes, the pointer to block is 32-bit. Calculate:

a) Maximum size of a file if using direct indexing, with a single block of index pointers per file.

b) Maximum size of a file if you are using linked allocation

no había puntuación en el examen. Asumo 1 point.

7

a)

Every block contains 512/4 = 128 pointers, so the máximum size of a file is:

128x512=64 KBytes

b)

*The máximum size is the lower value between the disk partition and the pointer addressing capability. A 32 bit block pointer has an addressing capability of 2^{32} * 512 bytes that is much higher tan the partition data size 30 Mbytes. So the máximum size in this case is the partition size removing the pointers space:*

30MBytes -> $30x2^{10}$Kbytes -> $30x2^{11}$ blocks

*Maximum file size = 30 Mbytes – 15 * 2^{11} * 4 bytes =*

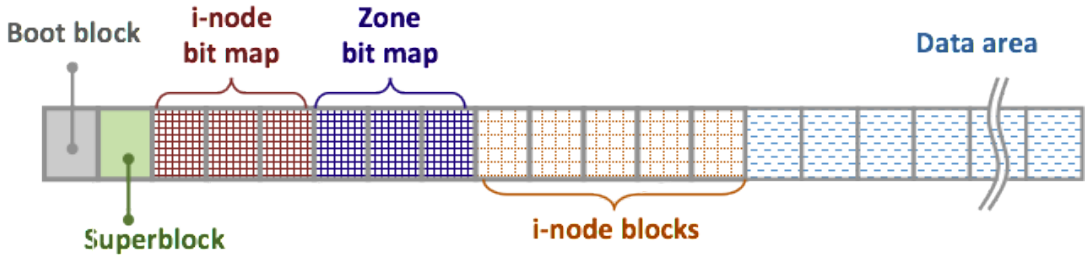
30720 Kbytes – 120 Kbytes =30600 Kbytes

A 16 MByte partition has been formatted with Minix, using standard sizes and creating a i-node by each 2Kbytes Minix standard sizes are as follows: nodos-i of 32bytes (7 direct pointers, 1 indirect pointer and 1 double indirect pointer), directory entries of 16bytes, 1 zone = 1 block = 1024bytes, pointers to zone have 16-bit

Determine the partition structure indicating each of the elements that compose it and the size of every one in blocks or zones, as required.

1.0 point

8



1 Boot block

1 Superblock

Number of i-nodes = 16Mbyte / 2Kbyte = 8Kbyte= 8192 i-nodes

*i-nodes map >= 8192 bits = 8*1024 = 1Kbyte = 1 block*

Number of zones = 16Mbytes / 1Kbyte = 16K zones

Zone map = 16384 bits = 2Kbyte

*i-nodes area= 8192 i-nodes * 32 bytes = 32 Kbytes = 32 blocks*

Data zone = 16384 zones - (1 boot + 1 superblock + 1 i-node map + 2 zone map + 32 i-nodes) = 16384 - 37= 16347 zones (or blocks: 1 zone = 1 block)