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. Let's consider a system that manages 1200 KBytes of main memory relying on contiguous allocation with variable partitions. The hole allocation policy is Worst Fit with compaction. The compaction algorithm minimizes shifts in memory until finding enough space to allocate the process that requests memory at every moment, always choosing to move processes towards the highest addresses. Consider the initial allocation state shown in the following table:

0 1200KB-1

SO 80KB	P1 160KB	HOLE 400KB	P2 120KB	HOLE 440KB

(1,0 points = 0,6 + 0,4)

a) Manage main memory, applying compaction if necessary, and indicate the base address assigned to each process when the indicated sequence of process arrivals happens, from left to right:

	Initial state	P3 arrives Requests 280K	P4 arrives Requests 200K	P5 arrives Requests 280K	P6 arrives Requests 80K
Base address of P1	80K	80K	80K	80K	80K
Base address of P2	640K	640K	640K	1040K	1080K
Base address of P3	----	760K	760K	760K	800K
Base address of P4	----	---	240K	240K	240K
Base address of P5	----	----	----	440K	440K
Base address of P6	----	----	----	.....	720K

b) Explain what type of fragmentation appears in variable partitions. If it happens at some point in section a) then estimate its amount.

External fragmentation, since the partition size is adjusted to the process size.So, when processes end they leave memory holes that may not be large enough to contiguously allocate a new process, then compaction is required. .
When trying to locate P5 in memory there are two holes, none is big enough:
200-Byte hole in byte 440 and 160-Byte hole in byte 1040 → Total external fragmentation = 360 Bytes

When trying to locate P6 in memory there are two holes, none is big enough:
40-Byte hole in byte 720 and 40-Byte hole in byte 1160 → Total external fragmentación = 80 Bytes.

2. In relation to page sharing in a paging system, indicate for every one of the following statements if it is true (T) or false (F) :

Note. A wrong answer voids a correct answer. **(0,8 points)**

2	STATEMENT	T / F
	There are page descriptors belonging to different processes that contain the same frame id corresponding to shared pages	T
	The copy-on-write technique allows to share pages with writing permission	F
	Processes that share pages allocated in memory, also share the page table	F
	A page loaded in memory in a region with permissions r-xp can be shared by several processes	T
	The probability to reach thrashing increases when increasing the number of shared pages between processes	F

3. A system has 24-bit both physical and logical memory addresses. Page size is 4-KByte and it manages virtual memory through paging. Free frames are assigned in increasing order, the replacement algorithm applied is LRU with LOCAL scope .

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

3

a) Obtain the reference string corresponding to the following sequence of logical addresses (in hexadecimal) emitted by processes A and B during their execution: A:40000, A:40014, B:80000, B:80024, B:60030, A:60034, B:80280, B:4060c, A:20000, A:40c10, B:60f24

A:40 B:80 B:60 A:60 B:80 B:40 A:20 A:40 B:60

b) The system assigns frames 0 and 1 to process A and frames 2, 3, 4 to process B, all of them are initially are empty. Complete the following table with the content evolution of main memory for the reference string obtained in the previous section. Indicate the number of page faults produced.

Frame	A:40	B:80	B:60	A:60	B:80	B:40	A:20	A:40	B:60
0 (A)	A:40	A:40	A:40	A:40	A:40	A:40	A:20	A:20	A:20
1 (A)				A:60	A:60	A:60	A:60	A:40	A:40
2 (B)		B:80	B:80	B:80	B:80	B:80	B:80	B:80	B:80
3 (B)			B:60	B:60	B:60	B:60	B:60	B:60	B:60
4 (B)						B:40	B:40	B:40	B:40

Number of page faults: 7 (2 with replacement)

4. A system with demand paging has two levels paging. The first level table has 16 page descriptors, logical and physical addresses are both 24-bit wide, and page size is 4-Kbyte. This system assigns 3 frames to each process and it is executing process A. At time t = 10, process A has frames 0xf00, 0xf01 and 0xf02 occupied by pages 0x001, 0xf2f and 0x11a respectively .

(1,6 points = 0,4 + 0,4 + 0,8)

4

a) Describe the structure of the logical addresses and physical addresses in this system, as well as the size in bits of each one of their fields.

Logical addresses:

- 16 page descriptors 1st level -> Page id first level 4 bits
- 4 KByte page size -> Offset 12 bits
- #bits Page id 2nd = 24 - 12 - 4 = 8 bits

```
-b23-------------------b20-b19--------b12-b11-----------------b0-
|  P1 1er nivel (4 bits) |  p2 (8 bits) | offset (12 bits)      |
-----------------------------------------------------------------
```

Physical addresses: Frame id 12 bits + Offset 12 bits

```
-b23---N° marco (12 bits)---b12-b11---Desplazam. (12 bits)---b0-
```

b) Explain the content of the page descriptors for process A that have the valid bit set to 1 at time t=10.

With two levels paging two page tables are required, one of first level and another of 2nd level, in order to be able to translate logical addresses into physical ones. All pages will have their validity bit at 0 except those in memory at t = 10.

Page 0x001→ 1st level descriptor 0, 2nd level descriptor 1, frame f00, v=1

Page 0xf2f → 1st level descriptor f , 2nd level descriptor 2f, frame f01, v=1

Page 0x11a → 1st level descriptor 1, 2nd level descriptor 1a, frame f02, v=1

c) Complete the following table with the corresponding logical or physical addresses, which could have been issued or accessed for process A at time t = 10, explaining if the given values are possible.

Logical address	Physical address	Is it possible? Explain why
f2ff02	0xf01f02	It is possible, it corresponds to frame id f01 and offset f02
11af01	0xf02f01	It is pòssible, it corresponds to frame id f02 and offset f01
0x11a13b	f0213b	The logical address 11a13b corresponds to pages 11a which is located in frame f02
0x13b11a	Page fault	Page 13b is not in memory and therefore when it is referenced a page fault happens. All available frames are taken so a victim has to be identified.

5. Complete the C program in section a) with the necessary POSIX primitives (one on each line with underlined number) for a parent process to read the contents of the "message.txt" file and send 2 concatenated copies of it to its child process through a pipe. The child process will forward the content that it receives from the pipe to the standard output relying on cat program.
Note. cat, without arguments, writes everything it reads from the standard input into the standard output).

(1,4 points= 0,8 + 0,6)

5 **a)**

```
1   #include <all_needed>
2   #define SIZE 80
3   #define MODE O_RDONLY
4   int main( int argc, char **argv ){
5       int fd, fdp[2], readbytes, ncopy;
6       char buffer[SIZE];
7       pipe(fdp);
8       if (fork() == 0) { // child
9           dup2(fdp[0],0);
10          close(fdp[0]);
11          close(fdp[1]);
12          execlp("cat","cat", NULL);
13      }
14      else { // parent
15          close(fdp[0]);
16          fd= open("message.txt", MODE);
17          for(ncopy=0; ncopy<2; ncopy++) {
18              while((readbytes= read(fd, buffer, SIZE)) > 0){
19                  write(fdp[1], buffer, readbytes        ); /*complete*/
20              }
21              lseek(fd, 0, 0     ); /*complete*/
22          }
23          close(fdp[1]);
24          close(fd);
25      }
26      wait(NULL);
27      exit(0);
28  }
```

b) Fill the child's file descriptor table, at the time when line 12 is executed. Do the same for the parent process in line 17. The tables have to be compliant with the requirements and the implementation of section a).

Child process	
0	"fdp[0]"
1	STDOUT
2	STDERR
3	
4	
5	

Parent process	
0	STDIN
1	STDOUT
2	STDERR
3	"mensaje.txt"
4	"fdp[1]"
5	

6. Given the following listing of a directory in a POSIX system :

```
i-node  permissions links    user   group   size     date       name
37093377 drwxr-xr-x  3       marta   disca    4096 dic 11 11:57 .
32448485 drwxr-xr-x  3       marta   disca    4096 dic 11 12:02 ..
32448767 -rwsr-xr-x  1       marta   disca  141528 dic 11 10:31 cp2
33373385 dr-xrwxr-x  2       marta   disca    4096 dic 11 12:02 dir1
32448804 lrwxrwxrwx  1       marta   disca       4 dic 11 10:35 dir2 -> dir1
32448793 -r--r--r--  1       marta   disca     337 dic 11 10:33 f1
32448802 -rw-r--r--  3       marta   disca     402 dic 11 10:33 f2
32448802 -rw-r--r--  3       marta   disca     402 dic 11 10:33 f3
32448803 lrwxrwxrwx  1       marta   disca       2 dic 11 10:34 f4 -> f3
```

(1,4 points = 0,8 + 0,6)

6

a) Program cp2 is similar to Linux command cp. When the last parameter of cp2 is a directory it copies the indicated files in this directory (creating the files if they do not exist). Assume that directory dir1 is empty, containing only "." and "..". In case of success indicate what are the permissions that are checked and, in case of error, what is the permission that fails and why.

(UID,GID)	COMMAND	DOES IT WORK?	EXPLANATION
(pepe, disca)	cp2 f1 dir1	No	When "pepe" executes cp2 the process effective user changes to "marta" who is the owner of f1 and dir1. Checking permissions we find that the process can read f1 (ok) but it can not create a file (write) to dir1 (error)
(juan, fso)	cp2 f1 dir2	No	dir2 is a symbolic link to dir1 so you need write permissions on dir1 to copy the file inside. When "juan" executes cp2 the process effective user changes to"marta" who is the owner of f1 and dir1, checking permissions we find that the process can read from f1 (ok) but it can not write to dir1 (error)
(ana, alum)	cp2 f1 f2	Yes	When "ana" executes cp2 the process effective user becomes "marta" who is the owner of f1 and f2, as such the process can read from f1 (ok) and can write in f2 (ok)

b) Explain the value on the links column for every entry, indicating what is or may be the reason for that entry to have the given number of links.

NAME	LINKS	EXPLANATION
.	3	Directory "." has 3 links that are: the reference from its parent directory, its self-reference of its entry "." and reference ".." in dir1 directory
dir1	2	This directory does not have any subdirectory since it only has two links, that of its father and its own entry "."
f2	3	This file has 3 links: the entry itself f2, f3 which can be seen to have the same i-node and there should be a third entry (hard link) in a directory not shown

7. A file system is organized in blocks of 512 bytes, with block pointers of 16 bits. In this system there is a 32 KByte file named "example.txt. For every of the indicated allocation methods, obtain the number of accesses to data blocks needed to read Byte 16900 in the file considered. Explain your answer.

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

7	**a)** Linked allocation In linked allocation the pointers to blocks are together with the data of the file itself. So each data block contains in its last 2 bytes a pointer to the next block, then only 510 bytes per block are available for data, therefore the block index where byte 16900 is located at block: Floor(16900 / 510) = 33 As in linked allocation in each block there is a pointer to the next, 34 accesses to blocks (from 0 to 33) will be required to read the byte 16900
	b) Indexed allocation In indexed allocation pointers to data blocks are contained inside a block. A block of 512 bytes with 2-byte pointers, can contain up to 256 pointers to data blocks. The block index where byte 16900 is located is: Floor(16900 / 512) = 33 In case of indexed allocation, it is only necessary to make 2 accesses, one to the index block, that containing the pointers, and another to access block 33.

8. A 64-MByte disk is formatted with a Minix file system, with the following specs:

- Boot block and superblock: 1 block each one
- 32-Byte i-nodes with 7 direct pointers, 1 indirect,1 double indirect
- 16-bit pointers to zone
- 16-byte directory entries: 2 Bytes for i-node, 14 Bytes for name
- 1 zone = 1 block = 1 KByte

(1,6 points = 0,3 + 0,8 + 0,5)

8	**a)** Obtain the maximum number of i-nodes that this system can have. Explain your answer. The maximum number of i-nodes is limited by the size of the i-node on a directory entry, which is 16 bits. Therefore the maximum number of i-nodes allowed is $2^{16} = 65536$.

b) Suppose the disk is formatted by reserving space in the header for a total of 32768 i-nodes (32 K i-nodes). Calculate the number of blocks occupied by each element of the file system, indicated below.

Boot	Super block	i-nodes bit map	Zones bit map	i-nodes	Data zones

1 Boot block → bloque 0
1 Superblock → bloque 1
4 Blocks for the i-node bit map → blocks 2, 3, 4 and 5
32K i-nodes require 32K bits, so 32Kbits / 1KByte = 32Kbits / 8KBits = 4 blocks
8 Blocks for the zone bit map → blocks 6, 7, 8, 9, 10, 11, 12, 13 and 14
64 MBytes partition size = 64 MBytes / 1 KByte = 64K zones → 64Kbits required
64 Kbits / 8 Kbits= 8 Blocks
1024 Bloques para los Nodos-i → block 16 to block 1040
32 K i-nodes * 32 Bytes = 1 MByte to store the i-nodes
1 MByte / 1 KByte = 1024 blocks to store the i-nodes
The data area starts at block 1041
The header occupies: 1 + 1 + 4 + 8 + 1024 = 1038 blocks
Partition size is 64 MBytes → 64 MBytes / 1 KByte = 64K Blocks = 65536 Blocks
65536 - 1038 = 64498 data zones

c) Suppose that, at any given moment, there are a total of 32768 nodes-i (32K nodes-i) occupied on the disk with a single directory, the root directory, and regular files all of 1KByte in size.Obtain the number of data zones that will be occupied.
i-node 0 is reserved for deleting directory entries
i-node 1 corresponds to the root directory, so there are 32766-nodes available for regular files.
Each regular file has a directory entry associated with it, in addition to "." and ".." directory entries.
So this implies that the root directory will be a file with 32768 directory entries.
Each directory entry occupies 16 Bytes, so the root directory occupies:
32 K Directory entries * 16 Bytes = 512K Bytes
The addressing capability of direct pointer is 7 KBytes, so the single indirect pointer is used and consequently a block of pointers in the data area is used.
Each file occupies 1 zone, that is 1 KByte, these files do not need blocks of pointers since they are addressed with the first direct pointer of the node-i, the occupation of the files is thus: 32766 * 1K

The number of occupied zones is:
512 root directory + 32766 regular files + 1 pointers = 33279 zones