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 9 questions, everyone has its score specified.**

1. Using system calls for process creation and program execution, write the required code for getting the following output on the standard output device :

1. Listing of the working directory (same as command "ls -la")
2. The following text: ***Directory listing completed***

(1,0 point)

```
1  /*** C code ***/
2  #include "all required headers"
3
4  main() {
5    pid_t pid;
     int status;
     pid=fork();
     if (pid==0) {
         if (execl("/bin/ls", "ls" , "-la" , NULL)<0){
            fprintf(stderr,"error en el ls\n");
            exit(-1);
          }
      } else {
          wait(&status);
          printf("Directory listing completed\n");
      }
   }
```

2. Let's consider a timesharing system with short-term scheduler and a single ready queue. Consider processes A, B and C, that arrive at instants 0, 1 and 2 respectively :

A(t=0)	3CPU+2E/S+1CPU	B(t=1)	1CPU+1E/S+1CPU	C(t=2)	2CPU+1E/S+1CPU

Fill in the following diagrams for each of the scheduling policies indicated and calculate the values requested. I/O queue is FCFS. In case of simultaneous events the following occurrence order is considered: new process, process coming from I/O and quantum end. .

(1,2 points = 0,6+0,6)

a) SRTF (Shortest-Remaining-Time-First)

T	Ready	CPU	I/O queue	I/O	Event
0		A(2)			A arrives
1	A	B(0)			B arrives
2	C	A(1)		B	C arrives
3	B,C	A(0)			
4	C	B(0)		A	
5		C(1)		A	Fin B
6	A	C(0)			
7		A(0)		C	
8		C(0)			Fin A
9					Fin C
10					

	A	B	C	*Mean value*
Turnaround time	8	4	7	19/3 = 6,33 u.t.
Waiting time	2	1	3	6/3 = 2 u.t.
CPU utilization	9/9 = 100%			

b) RR (Round-Robin) q=1 u.t.

T	Ready	CPU	I/O queue	I/O	Event
0		A(2)			A arrives
1	A	B(0)			B arrives
2	C	A(1)		B	C arrives
3	A,B	C(1)			
4	C,A	B(0)			
5	C	A(0)			Fin B
6		C(0)		A	
7			C	A	
8		A(0)		C	
9		C(0)			Fin A
10					Fin C

	A	B	C	*Mean value*
Turnaround time	9	4	8	21/3 = 7 u.t.
Waiting time	3	1	3	7/3 = 2,33 u.t.
CPU utilization	9/10 = 90%			

3. A system has 16-bit logical addresses and pure paging (without virtual memory). Page size is 4096 bytes and physical addresses are 20-bit. .
A program X on this system contains 13004 bytes of code and 6900 bytes of variables. Along compiling and linking X the following elements are added: library B of 5004 Bytes, global variables of 1648 Bytes and constants of 796 Bytes. When executing X the system assigns to the process the maximum number of pages that allows the logical address format, allocates code, constants and variables on different pages, and it also separates the elements of program (X) and libraries (B). The compiler assigns to the stack 8 KBytes of the highest logical addresses and the remaining to the heap..

(1,0 point = 0,3+ 0,3 + 0,2 + 0,2)

1

a) In all, how many pages will have the execute permission (bit x) set to 1 on the memory map of process P created by running X?

6 pages: 4 from *X* and two from *B*

b) How many pages of P will have writing permission ?

16 – 6 (code) – 1 (constants from *B*) = 9

c) What range of logical addresses will occupy the stack of P ?

From E000 to FFFF

d) How many pages will be available for the heap *?*

16 - (6 code + 3 variables + 1 constants + 2 stack) = 4 pages

4. The following program (incomplete) is based on the famous "game of life", where **`m`** is an array of "cells" (characters) that can be "alive" ('*' character) or "dead" (blank character). On every iteration, every cell evolves (borns or dies), depending on the number of alive neighboring cells. There is a `Cell` thread for each element of the array, that keeps it evolving. It receives as a parameter a pointer to the array element that corresponds to it. As in the "matrix" program on the lab session about threads, here there is a thread `Refresher` that displays periodically the content of matrix `m` on the screen.

Explain your answer to the following questions : **(1,1 points = 0,25 + 0,25 + 0,2 + 0,2 + 0,2)**

```
1  #include "all required headers"
2  #define FILS 10
3  #define COLS 10
4  #define ITERATIONS 20
5
6  char m[FILS][COLS];
7  pthread_attr_t atrib;
8  pthread_t  cell_thread[FILS][COLS];
9  pthread_t  refresh_thread;
10
11 int NumNeighbors(char *ptr){
12  /* Reads 8 neighbor cells, on matrix
13 m and counts how many are '*'. Not
14 shown to simplify */
15 }
16
17 void *Cell(void *ptr){
18   char* pchar= (char*) ptr;
19   int i, vec;
20   for(i=0; i<ITERATIONS; i++){
21     vec= NumNeighbors(pchar);
22     if(*pchar=='*'){ // Cell alive
23        if ((vec!=2)&&(vec!=3))
24          *pchar=' '; // Dies
25        }else // Cell dead
26          if (vec==3)
27            *pchar='*'; // Borns
28     usleep(1000000); // Wait 1seg
29   }
30 }
31 void *Refresher(void *ptr){
32   // Not shown to simplify
33 }
34
35 int main() {
36  int fil, col;
37
38  // Random cells start
39  for(fil=0; fil<FILS; fil++)
40   for(col=0; col<COLS; col++)
41    m[fil][col]=(rand()%10>4)?'*':' ';
42
43  pthread_attr_init(&atrib);
44
45  // Create one cell thread per element
46
47  // Create refresher thread
48  // (not include it to simplify)
49
50  // Wait cell threads ending
51
52 }
```

a) Complete the code on line 45 to create a cell thread for every of element of matrix m, so that all of them operate concurrently. Use only the variables and functions that have already been defined in the program .

```
for(fil=0; fil<FILS; fil++)
 for(col=0; col<COLS; col++)
  pthread_create(&cell_thread[fil][col],
                 &atrib, Cell, &m[fil][col]);
```

b) Complete the code on line 50 to do the waiting for all cell threads .

```
for(fil=0; fil<FILS; fil++)
  for(col=0; col<COLS; col++)
    pthread_join(cell_thread[fil][col],NULL);
```

c) What is the maximum number of threads that can run concurrently? Explain your answer.

102 threads: 100 Cell (1 per each m's element) + Refresher + main tread

d)The program doesn't wait for Refresher thread. Explain what happens to this thread when the program ends reaching line 52.

When the main thread ends without calling pthread_exit, then the OS ends the execution of all the process threads, so Refresher thread ends

e) Define the concept of "race condition". Can a race condition happen in this program?

A race condition happens when multiple threads or processes access concurrently the same data and the result of the execution depends on the access order. In this program a race condition can happen because Cell threads access concurrently the same m array elements without any synchronization, so the final result may be different after every execution

NOTE: Functions definition:

int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);

int pthread_join(pthread_t thread, void **retval);

5. Three processes are running concurrently containing three functions f1(), f2() and f3() as shown in the table. Before and after the execution of these functions there have to be a input protocol (PEX) and an output protocol (PSX), relying **EXCLUSIVELY** on operations on two semaphores A and B, to comply with the following requirements for every case:

Case a) f1(),f2() y f3() They have to be executed in mutual exclusion, but without any particular order.

Case b) The sequence of execution of these functions has to be the following f3(), f2() y f1().

Case c) A maximum of two functions can run concurrently (without mutual exclusion) .

Case d) f1() and f2() can run concurrently (without mutual exclusion), but f3() has to wait for f1() and f2() endings .

Process 1	Process 2	Process 3
sleep(5) . PE1 . f1() . PS1	sleep(5) . PE2 . f2() . PS2	sleep(10) . PE3 . f3() . PS3

(1,0 points = 0,25 + 0,25 + 0,25 + 0,25)

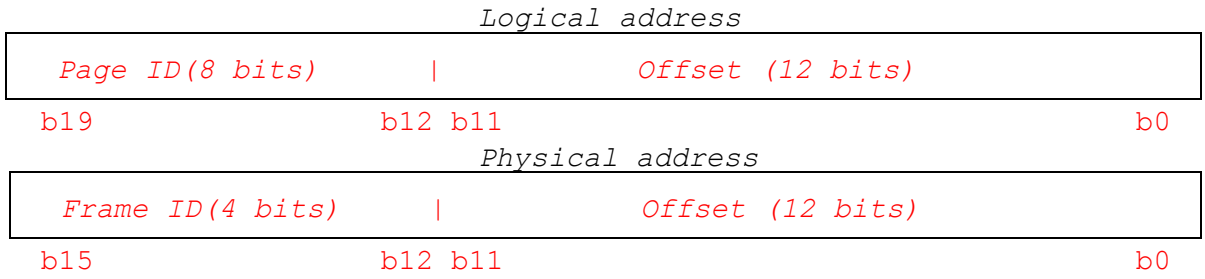
5 **Complete the table.** Indicate the initial value of the semaphores for every case.
The "EXAMPLE" column shows how to complete the table and does not correspond to any logical sequence. If code for input and output protocols is not required indicate it with an hyphen (as in EXAMPLE for entry PE3).

	EXAMPLE	Case a)	Case b)	Case c)	Case d)
Initial value A	0	1	0	2	0
Initial value B	0	---	0	---	0
PE1	P(A)	P(A)	P(A)	P(A)	---
PS1	V(B)	V(A)	---	V(A)	V(A)
PE2	P(B)	P(A)	P(B)	P(A)	---
PS2	P(A)	V(A)	V(A)	V(A)	V(B)
PE3	---	P(A)	---	P(A)	P(A);P(B)
PS3	V(B)	V(A)	V(B)	V(A)	----

6. Let's consider a virtual memory system with demand paging that uses LRU replacement algorithm with LOCAL scope. In this system, the logical addresses are 20-bit and physical addresses are 16-bit with 4KB page size. Main memory size is 20 KB (5 frames) and initially main memory is empty. There are two processes A and B in the system, to which the operating system allocates frames 1, 2 and 3 to process A, and frames 4 and 5 to process B .

(1,3 points = 0,4 + 0,2 + 0,4 + 0,3)

6 **a)** Obtain the formats for logical and physical addresses, with the name of the fields and their sizes, indicating starting and ending bits. .



b) Obtain from the sequence of logical addresses, the reference string:
(A,2D4B8), (A,2D4B9), (A,2D4EB), (B,2D4EB), (B,2D86F), (B,0B621), (B,0B815), (A,2C4B8), (A,2CA23), (B,946C3), (B,2D1A7), (B,2D1A1), (A,2D4C7), (B,0BA31), (B,0BA32), (A,2CA24), (A,2CA25)

(A,2D) (B,2D) (B,0B) (A,2C) (B,94) (B,2D) (A,2D) (B,0B) (A,2C)

c) Complete the table with the evolution of main memory content with reference string obtained on the previous section. Also indicate when the page faults happen.

Frame	A,2D	B,2D	B,0B	A,2C	B,94	B,2D	A,2D	B,0B	A,2C
1	**2D**	2D	2D	2D	2D	2D	2D	2D	2D
2				**2C**	2C	2C	2C	2C	2C
3									
4		**2D**	2D	2D	**94**	94	94	**0B**	0B
5			**0B**	0B	0B	**2D**	2D	2D	2D

TOTAL NUMBER OF PAGE FAULTS :

d) Compute the working sets for the two processes, with a window size of 6, at the end of the sequence of section b). Would you change the number of frames allocated to each process? and if so, what would be your proposal?

The working set size for process A is 2 (2D, 2C) and for B is 3 (94,0B, 2D). Therefore process A has been assigned more frames than needed. Process B however requires three frames, to match its working set. So optimal allocation would be: process A with two frames and process B with three frames

7. Program Descrip.c generates two processes (parent and child) and before they start running, files A.txt and B.txt contain 10 and 16 characters, respectively; as shown below:

Descrip.c

```
int main() {
   int fd1, fd2, fd3, p[2];
   char buffer[50];

   fd1 = open( "A.txt", O_RDONLY );
   read(fd1, buffer, 5);
   fd2 = open( "B.txt", O_WRONLY );
   write(fd2, buffer, 5);
   dup2(fd1,5);
   read(5, buffer, 3);
   fork();
   write(fd2, buffer, 3);
   read(fd1, buffer, 1);
   /*write(fd1, buffer, 1);*/ /*comment*/
   exit(1);
}
```

A.txt

```
1234567890
```

B.txt

```
AAAABBBBCCCCDDDD
```

(1.1 points = 0.5+0.4+0.2)

7

a) For every generated process indicate, the content of its file descriptor table after running its last instruction and before exit()

	Parent process table		Child process table
0	*STDIN*	0	*STDIN*
1	*STDOUT*	1	*STDOUT*
2	*STDERR*	2	*STDERR*
3	*A.txt*	3	*A.txt*
4	*B.txt*	4	*B.txt*
5	*A.txt*	5	*A.txt*
6		6	
7		7	

b) Indicate the content of files A.txt and B.txt after finishing the execution of those processes

A.txt	B.txt
1234567890	*12345678678CDDDD*

c) Indicate the result of running the above code if we uncomment line /* comment */

Since in this process fd1 descriptor is obtained only with read permission -> This system call for writing will have no effect

8. Given the following content listing of a directory on a POSIX system :

```
drwxr-xr-x   2 pep     alumne         4096 ene  8     2013  .
drwxr-xr-x  11 pep     alumne         4096 ene 10     14:39 ..
-rwsr--r-x   1 pep     alumne      1139706 ene  9     2013  intercanvi
-rw-------   1 pep     alumne       634310 ene  9     2013  f1
-rw-r--r--   1 ana     profes       104157 ene  9     2013  f2
-rw-rw-r--   1 pep     alumne       634310 ene  9     2013  f3
lrwxrwxrwx   1 pep     alumne       634310 ene  9     2013  f4 ->f1
```

Where `intercanvi` program requires the name of two files as arguments. The `intercanvi` program swaps the contents of two files, accessing only to the files and memory buffers, ie, the command `intercanvi file1 file2` results that the initial content of `file1` is copied to `file2` and the initial content of `file2` is copied to `file1`. Fill the table indicating in case of success the permissions that are checked and in case of error which is the permission that fails and why

(1,0 points = 0,25 + 0,25 + 0,25 + 0,25)

8

User	Group	Command	Does it work?
ana	**profes**	**./intercanvi f1 f2**	**No**
Explain To execute this command the permissions required are: execution on "intercanvi", read and write on f1 and read and write on f2. The command fails because, despite being able to run intercanvi for (ana, profes) and becoming (pep, profes) after starting up (setuid bit is set), when the file permissions f2 are checked (f2 is owned by (ana, profes)), we see that we have read permissions only on f2 because it would comply with the second permissions triplet .			
ana	**profes**	**./intercanvi f3 f4**	**Yes**
Explain To execute this command the permissions required are: execution on intercanvi, read and write on f3 and read and write on f4, which as a symbolic link to f1, actually the permissions that apply are those from f1. (Ana, profes) can execute "intercanvi" becoming (pep, profes), changing the effective owner to the one that owns files f1 and f3 (pep), the process can read and write on both f1 and f3			
pau	**alumne**	**./intercanvi f1 f3**	**No**
Explain **To execute this command the permissions required are: execution on intercanvi, read and write on f1 and read and write on f3.** **Intercanvi cannot be executed by group "alumne" because the group triplet has no set execution permission**			
pep	**alumne**	**./intercanvi f2 f1**	**No**
Explain **To execute this command the permissions required are: execution on intercanvi, read and write on f2 and read and write on f1.** **Fails to write on f2, since the matched triplet is "others" and it doesn't have the write bit set**			

9. A Minix file system contains the following directory entries :

1	.
1	..
3	app
4	tmp
6	media

3	.
1	..
17	doom3

4	.
1	..
13	trash
41	emacs

6	.
1	..
15	home

15	.
6	..
20	luis
23	marta

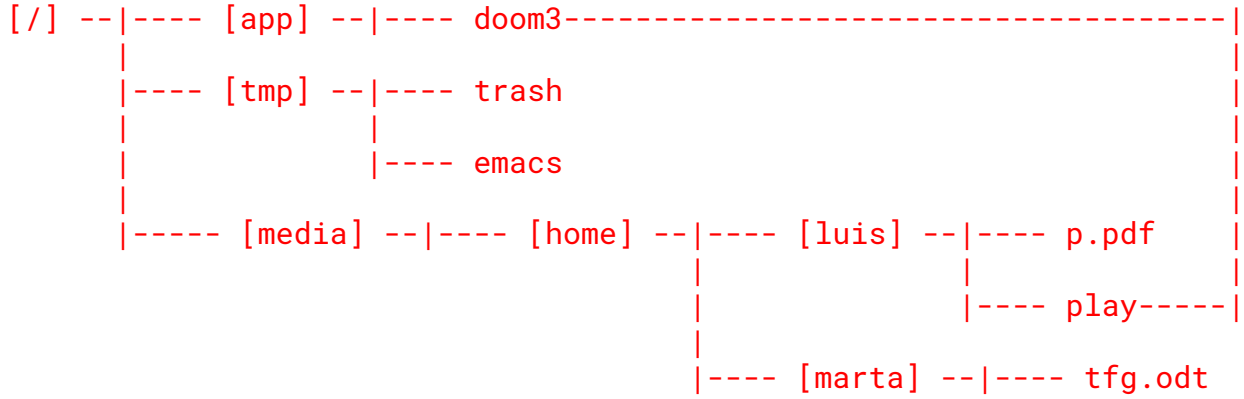
20	.
15	..
50	p.pdf
17	play

23	.
15	..
64	tfg.odt

(1,3 points = 0.4+0.2+0.2+0.5)

9

a) Draw the file system tree, indicating for every file its type.



[*xxx*] directory, *xxx* regular file

b) Explain if there is any hard link between the regular files.

Yes, files "/app/doom3" and "/media/home/luis" are physical links to the same regular file, both point to the same i-node (17)

c) Explain if there are physical links between directory files, if so tell the cases.

Yes, there are physical links to directories. All directory files have the following physical links: one from itself (directory entry "."), one from its parent directory (except the root directory), and one from every subdirectory (directory entry "..")

d) If the file system is implemented on a partition formatted as 1 zone = 1 block = 1KB with i-node size of 32-bytes, 16K inodes and a total size of 64K blocks. Indicate (in KB) the size of the partition header, clearly stating the size for every header element.

The header takes: [1 (boot) + 1 (superblock) + n (i-node map) + m (zone map) + N (i-nodes)] blocks
1 block = 1KB -> 1Kx8 bits = 8K bits
n = 16K / 8K = 2 blocks
m = 64K / 8K = 8 blocks

i-node takes 32B, in 1 block they fit 1KB/32B = 32 i-nodes
N = 16K/32 = 512 blocks
So, we have
Header takes: 1 + 1 + 2 + 8 + 512 = 524 blocks = 524KB