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. We have studied some POSIX system call classified by the following categories:**P**rocesses, p**R**otection, **D**irectories, **S**ignals y **F**iles. Fill the table with the category to which the corresponding system call belongs to. Identify every category by the highlighted letter (**P**, **R**, **D**, **S** and **F)**.

(1,0 points: a hit scores 0,1 a failure -0,025)

System call	Category
fork	P
chmod	R
kill	S
open	F
exit	P
read	F
chown	R
sigaction	S
mkdir	D
opendir	D

2. In a timesharing system with short term scheduler and a single ready queue. Consider processes A, B and C, that arrive at times 0, 2 and 4 respectively :

A(t=0)	4 CPU+1 I/O+1 CPU	B(t=2)	3 CPU+1 I/O+2 CPU	C(t=4)	2 CPU+3 I/O+1 CPU

Fill in the following execution timeline tables corresponding to the schedulers indicated and obtain the performance indexes requested. The I/O queue is FCFS. In cases when several events occur the same time, queue allocation order is: new process, process coming from I/O and quantum end.

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

a) Preemptive priorities. Priority A < Priority B. Priority B = Priority C

T	Ready	CPU	I/O queue	I/O	Event
0		A			A arrives
1		A			
2	A	B			B arrives
3	A	B			
4	A,C	B			C arrives
5	A	C		B	
6	A,B	C			
7	A	B		C	
8	A	B		C	
9		A		C	B ends
10	A	C			
11		A			C ends
12				A	
13		A			
14					A ends

	A	B	C	*Mean*
Turnaround time	14	7	7	9,3
Waiting time	8	1	1	3,3

b) RR (Round-Robin) q = 2 u.t.

T	Ready	CPU	I/O queue	I/O	Event
0		A			A arrives
1		A			
2	A	B			B arrives
3	A	B			
4	B,C	A			C arrives
5	B,C	A			
6	B	C		A	
7	A,B	C			
8	A	B		C	
9		A	B	C	
10			B	C	A ends
11		C		B	
12		B			C ends
13		B			
14					B ends

	A	B	C	*Mean*
Turnaround time	10	12	8	10
Waiting time	4	4	2	3,3

3. The following program (incomplete), looks for the greatest number and position (row and column) in an array of integers. To do this, for every matrix row one thread is created that implements function **FindMaxNum** that seeks the greater value and its position in the row that is passed as a parameter, updating the corresponding variables: **RowMax**, **ColMax** y **NumMax**.

Explain your answer to the following questions: **(1,2 points = 0,5 + 0,2 + 0,50)**

```
1  #include <all_required>
2
3  #define NUMROWS 20
4  #define DIMROW 1000
5  int Nums[NUMROWS][DIMROW];
6  int RowMax, ColMax, NumMax;
7
8  void *FindMaxNum( void *p_row ) {
9      int k, row;
10     row  = (int)p_row;
11      for(k=0;k<DIMROW;k++) {
12         if (Nums[row][k] > NumMax) {
13            NumMax = Nums[row][k];
14            RowMax = row;
15            ColMax = k;
16         }
17      }
18      return NULL;
19  }
20
21 int main() {
22     int i,j;
23     NumMax = 0;
24     pthread_t  threads[NUMROWS];
25     pthread_attr_t attrib;
26
27     // Matrix elements are initialized randomly
28     for(i=0;i<NUMROWS;i++) {
29         for(j=0;j<DIMROW;j++) {
30             Nums[i][j] = rand();
31        }
32     }
33     // Thread attributes initialization
34     pthread_attr_init( &attrib );
35
36     for(i=0;i<NUMROWS;i++)
37     {
38        pthread_create(&threads[i], &attrib, FindMaxNum, (void*)i);
39     }
40
41     for(i=0;i<NUMROWS;i++)
42     {
43        pthread_join(threads[i], NULL);
44     }
45
46
47     printf("Max number is %d in pos %d,%d \n", NumMax, RowMax, ColMax);
48 }
49
50
```

a) Complete the code that creates one thread for every row, so they all can run concurrently and before the program executes the final printf all threads have finished. Make use only of variables and constants already defined in the program .

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);*

b) What is the maximum number of threads that can run simultaneously? Explain your answer considering the number of processor cores and the number of threads created by the program .

It will be the lower value between the number processor's cores and the number of created threads (in this case 20 + 1). For example, in a typical Intel Core i5 4-core processor it will be 4, and in an Intel Xeon Skylake (28 cores) it will be 20 + 1.

c) Can a race condition happen in this program? Explain your answer. If it is affirmative, write the code that has to be protected:

Yes, it can happen. Variables NumMax, RowMax and ColMax are global and they can be changed concurrently by the running threads, so race condition can happen.

```
Code to protect:   if (Nums[row][k] > NumMax) {
                        NumMax = Nums[row][k]; RowMax = row; ColMax = k;
                   }
```

4. Given the following code, use semaphores with the notation proposed by Dijkstra (P and V operations) to ensure that different threads execute the functions whose name begins with "fX" according to the order suggesting the numbers employed in such names. If there are several functions with the same name, none of them should start before the end of the function with the previous name and all must be completed before the start of the function with the next name. In addition, it must be ensured that "sc()" function calls are executed in mutual exclusion. Indicate the initial values required for the semaphores used. . **(1,0 points)**

4

Semaphores (initial value) :

mutex = 1
S1 = 0
S2 = 0
S3 = 0

Process 1	Process 2	Process 3

f1():
V(S1)
 	
P(mutex)
sc();
V(mutex)
 	
P(S3);
f4()
 	
P(mutex)
sc()
V(mutex)
	P(mutex)
	sc()
	V(mutex)
	
	P(S1); P(S1)
	f2()
	V(S2)
	
	P(mutex)
	sc();
	V(mutex)
	
	P(S3);
	f4();
		f1()
		V(S1)
		
		P(mutex)
		sc()
		V(mutex)
		
		P(S2);
		f3()
		V(S3)
		V(S3)

5. In memory management system based on two level paging (without virtual memory) both logical and physical addressing spaces are 4GB. Pages are 4KB. Every second level table has 4K entries (page descriptors). Answer the following questions :

(1,0 puntos = 0,4 + 0,4 + 0,2)

5

a) Logical and physical address formats with name and number of bits of each field:

Logical address (name and number of bits of every field)

P1 (8 bits)	*P2 (12 bits)*	*Offset (12 bits)*

b31 *b0*

Physical address (name and number of bits of every field)

Frame ID (20 bits)	*Offset (12 bits)*

b31 *b0*

b) A program X on this system contains 4200 Bytes of code, 100 Bytes of global variables and a maximum stack space of 4KB. The code begins in logical address 0x02000000 and pages for code, variables and stack are assigned in sequence. Frames are also allocated in sequence numbered from frame 0x0004b. Obtain the content of the second level page table in use, detailing the values of validity bit (V) and permission bits read, write and execute (R, W and X). Specify also in hexadecimal the index for every entry in this table. For all hexadecimal values, express them with the number of hexadecimal digits corresponding to the formats obtained in the previous section .

Page ID 2nd level	***Frame ID***	***V***	***R***	***W***	***X***
0x000	*0x0004b (code)*	*1*	*1*	*0*	*1*
0x001	*0x0004c (code)*	*1*	*1*	*0*	*1*
0x002	*0x0004d (global variables)*	*1*	*1*	*1*	*0*
0x003	*0x0004e (stack)*	*1*	*1*	*1*	*0*

c) According to the former section, what logical address corresponds to physical address 0x0004c00f?

Offset = 0x00f, Frame ID = 0x0004c ⇒ P2=0x001, P1=0x02 ⇒ DL= 0x0200100f

6. A system with virtual memory based on demand paging, uses LRU replacement algorithm with GLOBAL scope. The logical address space per process is 1GB and it has 4GB of main memory (physical memory) installed. Page size is 64KB. At any given time the execution of two processes starts: A and B, and the system assigns a total of four frames for both processes, frames that are initially empty.

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

6

a) Indicate the logical and physical addresses formats, with the field names, their sizes and bounding bits of each field.

Logical address

30 bits (29-0) -> 14 bits(29-16) Page ID + 16 bits(0-15) Offset

Physical address

32 bits (21-0) -> 16 bits(31-16) Frame ID + 16 bits(0-15) Offset

b) Get from the following sequence of logical addresses (in hexadecimal), the reference string:
(A:3A00FFF8) (A:3A00FFFC) (A:3A010000) (B:00400000) (A:080012FB) (B:1000458F) (A:08001AFF) (A:3A00FFF0) (B:00400004) (B:10012ABC)
A:3A00; A:3A01; B:0040; A:0800; B:1000; A:0800; A:3A00; B:0040; B:1001

c) Complete the table with the evolution of main memory content from the reference string obtained in the previous section. The frames are assigned in ascending order following the demand sequence. Also indicate the page faults produced.

Frame	A:3A00	A:3A01	B:0040	A:0800	B:1000	A:0800	A:3A00	B:0040	B:1001	
1	A:3A00*	A:3A00	A:3A00	A:3A00	B:1000*	B:1000	B:1000	B:1000	B:1001*	
2		A:3A01*	A:3A01	A:3A01	A:3A01	A:3A01	A:3A00*	A:3A00	A:3A00	
3			B:0040*	B:0040	B:0040	B:0040	B:0040	B:0040	B:0040	
4				A:0800*	A:0800	A:0800	A:0800	A:0800	A:0800	

TOTAL PAGE FAULTS: 7

d) After the last access on section c) the following new accesses are generated ,

(A:3A00FFFF) (B:00400008) (B:FFFFFFFF)

What physical address generates the MMU for every access in case of being correct?
(A:3A00FFFF) -> 0002FFFF
(B:0040008) -> 00030008
(B:FFFFFFFF) - > address out of range from 1GB logical space

7. Given the process inheritance mechanisms in Unix and the POSIX system calls, answer the following paragraphs :

(1.1 points = 0.4+0.7)

7 **a)** Write the file descriptor tables content indicating the non-empty descriptors, for every process involved in the following command:

```
$ /bin/ls | /bin/grep ".c" > my_programs
```

the command has to end correctly

Nota. Name the pipe as "pipeX"

Proceso "\bin\ls" table		Process "\bin\grep" table	
0	STDIN	0	pipeX[0]
1	pipeX[1]	1	"my_programs"
2	STDERR	2	STDERR

b) Complete the following C program with the required sentences and system calls in order to behave like the following command : `$cat < proc.c | grep "for" > loop.txt`
Assume no errors occur in system calls.

```
#include <unistd.h>
#include <fcntl.h>
#define readfile O_RDONLY
#define newfile (O_RDWR | O_CREAT | O_TRUNC)
#define mode644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

int main() {
    int pipeX[2];
    int fd;
    pipe(pipeX);

    if (fork()) {
        fd = open("proc.c", readfile);
        dup2(fd,STDIN_FILENO);
        dup2(pipeX[1], STDOUT_FILENO);
        close(pipeX[0]); close(pipeX[1]);
        close (fd);

        execl("/bin/cat", "cat", NULL);
     } else {
        fd = open("loop.txt", newfile, mode644);
        dup2(pipeX[0], STDIN_FILENO);
        dup2(fd, STDOUT_FILENO);
        close(pipeX[0]); close(pipeX[1]);
        close(fd);
        execl("/bin/grep", "grep", "for", NULL);


    }
    return 0;
}
```

8. Given the following listing that corresponds to a directory content on a POSIX system :

```
drwxrwxr-x   2 pep     alumne         4096 ene  8     2013  .
drwxr-xr-x  11 pep     alumne         4096 ene 10     14:39 ..
-rw-r-srwx   1 pep     alumne      1139706 ene  9     2013  append
-rw-------   1 pep     alumne       634310 ene  9     2013  f1
lrwxrwxrwx   1 ana     profes       104157 ene  9     2013  f2 ->f1
-rw-rw-r--   1 pep     alumne       634310 ene  9     2013  f3
-rwxrwxrwx   1 pep     alumne       634310 ene  9     2013  f4
-rwxrwxrwx   1 pep     alumne       634824 ene  9     2013  f5
```

Where the `append` program requires the name of two files as arguments. Command `append file1 file2` makes `file1` content to be written after the actual content of `file2`. If `file2` does not exist then it is created. Fill the table indicating, for every case, if the command is executed successfully or not. Also indicate what permissions are checked and, in case of failure, which is the permission that fails and why. .

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

8

User	Group	Command	Dos it work?
ana	**alumne**	**./append f1 f4**	**NO**
Explain **(ana, alumne) has execution permission for ./append but it doesn't have read permission on f1 because (ana,alumne) checks the second permission triplet and it doesn't give reading permission**			
pep	**alumne**	**./append f3 f4**	**NO**
Explain **It can not even execute ./append because (pep, alumne) checks the first permission triplet and it doesn't give execution permission**			
ana	**profes**	**./append f3 f6**	**SI**
Explain **(ana,profes) checks permissions on the third triplet that gives execution permission on ./append, becoming (ana,alumne) as effective user. So it has reading permission on f3 (second triplet) and writing permission on the directory to create file f6 (second triplet)**			
mar	**alumne**	**./append f5 f2**	**NO**
Explain **(mar,alumne) it can execute ./append because it checks permissions for ./append on the second triplet that has execution permission. It keeps identity (mar,alumne). So it checks the second triplet for f5 that has reading permission. But when trying to write on f2 permissions for f2 are checked on the second triplet and there is no writing premission.**			

9. Consider a Minix file system with the following features:

- i-nodes size 32 Byte, with 7 direct pointers to zones, 1 indirect pointer and 1 double indirect pointer
- Pointers to zone of 16 bits (2Bytes)
- Directory entries of 16 Bytes
- 1 block = 1 zone = 4KBytes

(1,2 points = 0.5+0.4+0.3)

9

a) Obtain the sizes for every header field considering a 128 MByte partition with 8192 as the maximum number of i-nodes.

1 boot block, 1 super block, 1 block i-nodes map, 1 block zones map and 64 blocks i-nodes

i-nodes map → 8192 / (4K*8) = $2^3\ 2^{10} / 2^2\ 2^{10}\ 2^3 < 1$ → 1 Block
Maximum number of zones = 128MB / 4KB = 32K zones
Zones map → Nº of zones / Block size in bits = 32K / (4K*8) < 1 → 1 Block
i-nodes → 8192 * 32 Bytes / 4KBytes = 64 Blocks

b) Explain what is the minimum number of busy zones, if the partition contains a file system with 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

It has 7 directories: root, app, tmp, media, home, luis and marta.
It has 5 files: doom3 (play), trash, emacs, p.pdf and tfg.pdf. Files "play" and "doom3" have the same i-node so it's the same file.

Every directory has assigned 1 zone, and it's enough to contain their respective directory entries → 7 zones
Every file has at least 1 zone → 5 zones at minimum
The minimum number of occupied zones is 12

c) Describe what advantages and disadvantages might involve using i-node pointers to zones of 20-bit, but only having seven direct pointers. Justify your answer obtaining the maximum partition size and the maximum file size that can be managed. .

ADVANTAGE: With 20 bit pointers to zone the addressable partition size is bigger than using 16 bit pointers. Particularly, with 20 bit pointers the maximum addresable size is 1 MBlocks, while with 16 bit pointers it is 64 KBlocks.
DISADVANTAGE: If i-nodes only have 7 direct pointers then the maximum file size is 7 zones, so we lose the capability of addressing much bigger files relying on single and double indirect pointers.