PROBLEM SESIONS

Operating Systems
Grau en Enginyeria Informàtica

Exercise 1. (T1,T2,T3)

1. (T1) A Linux user searches in Google the code of the system library for Linux (same architecture it works). He copies the code of the "write" system call in a separate program, he compiles it and executes it by passing valid arguments. What do you think it will happen? (The code will successfully execute or it will generate an "invalid operation" error code)
2. (T2) In an O.S. that applies a Round Robin policy, what it's the quantum? Is it possible to successfully execute a process with cpu burst that, in average, uses half of the quantum? The process will continue in the cpu once the cpu burst finishes or the process will leave the cpu? When the process receives the cpu after a blocking… (Passing through the ready state) … it will receive a complete quantum or only the remaining the last cpu burst?
3. (T3) Describe the benefits of using shared libraries.
4. (T3) Enumerates and summarizes the O.S. steps to load an executable file from disk to memory to implement a process mutation. Consider the case where the process continues in RUN state after the mutation.
5. (T3)We have a memory management system that implements virtual memory. We know it is suffering from thrashing. Describe briefly the thrashing problem and justify which metric will be clearly affected when having this problem: the user time or the system time.

Exercise 2. (T2,T3)

Let's consider the following code (we omit errors control in order to simplify the code and we can assume that there are not errors on any system call).

```
1. int *pids;
2. void usage()
3. {
4.   char  b[128];
5.   sprintf(b,"./Mtask processesLevel1(howMany) processesLevel2(0=no/1=yes)\n");
6.   write(1,b,strlen(b));
7.   exit(0);
8. }
9. void doTask(int i){
10.  // We omit this code to keep the code simple but we can suppose that
11.  // there are not system calls relevant for this exercise in this function
12.}
13.
14.void dataProcessing(int i, int multiprocess)
15.{
16.   int it;
17.   if (multiprocess>0){ it=0;  while((fork()>0) && (it<2)) it++;}
18.   doTask(i);
19.   exit(1);
20.}
21.void main(int argc,char *argv[])
22.{
23.  int i,ret,processes;
24.  char buff[128];
25.  if (argc!=3) usage();
26.  processes=atoi(argv[1]);
27.  pids=sbrk(processes*sizeof(int));
28.  for(i=0;i<processes;i++){
29.     ret=fork();
30.    if (ret==0) dataProcessing(i,atoi(argv[2]));
31.    pids[i]=ret;
32.  }
33.  while((ret=waitpid(-1,NULL,0))>0){
34.    for(i=0;i<processes;i++){
35.      if (pids[i]==ret){
36.        sprintf(buff,"Ends process number %d con pid %d \n" ,i,ret);
37.        write(1,buff,strlen(buff));
38.      }
39.    }
40.  }
        }
```

1. Represent the process hierarchy that it is created when executing this program in the two following ways (assign an identifier to each process to use in future references):
    a. ./Mtask 5 0
    b. ./Mtask 5 1
2. For each type of execution, will it be enough the size of the vector allocated in the initial process to keep the pids of the children? If the answer is not, then describe how to manage this issue. Justify your answers.
    a. ./Mtask 5 0
    b. ./Mtask 5 1
3. Which processes will execute the following code:
    a. Lines 36+37

- Launching ./Mtask 5 0
- Lauching ./Mtask 5 1

b. Function doTask

- Launching ./Mtask 5 0
- Lauching ./Mtask 5 1

4. When the initial process is loaded in memory it fills 2KB of code, 4Bytes of data, 4KB of stack and the heap size depends on the value of argv[1] (let's assume that heap size will be always be less than 4KB). Let's suppose that we execute this program on a Linux system, with a paging based system that has the following characteristics:
    - Page size is 4KB
    - Pages are not shared across different regions
    - There is COW optimization at page level

    Let's calculate the following, specifying each memory region (you can ignore all shared library regions):

    a. Size of the logical address space for each Mtask instance (number of pages)
    b. Size of the physical memory that we need to execute each execution case:
        i. ./Mtask 5 0
        ii. ./Mtask 5 1

Exercise 3. (T2)

Given the following code (for the sake of simplicity error control was omitted and we can assume that none of the calls returns an error).

```
1. int sigchld_recibido = 0;
2. int pid_h;
3. void trat_sigalrm(int signum) {
4. char buff[128];
5.    if (!sigchld_recibido) kill(pid_h, SIGKILL);
6.    strcpy(buff, “Timeout!”);
7.    write(1,buff,strlen(buff));
8.    exit(1);
9. }
10. void trat_sigchld(int signum) {
11.         sigchld_recibido = 1;
12. }
13. void main(int argc,char *argv[])
14. {
15.   int ret,n;
16.   int nhijos = 0;
17.   char buff[128];
18.   struct sigaction trat;
19.   trat.sa_flags = 0;
20.   sigempty(&trat.sa_mask);
21.   trat.sa_handler = trat_sigchld;
22.   sigaction(SIGCHLD, &trat, NULL);
23.
24.   n=atoi(argv[1]);
25.   if (n>0) {
26.         pid_h = fork();
27.         if (pid_h == 0){
28.           n--;
29.           trat.sa_flags = 0;
30.           sigempty(&trat.sa_mask);
31.           trat.sa_handler = trat_sigalrm;
32.           sigaction(SIGALRM, &trat, NULL);
33.           sprintf(buff, "%d", n);
34.           execlp("./ejercicio_exec", "ejercicio_exec", buff, (char *)0);
35.         }
36.         strcpy(buff,"Voy a trabajar \n");
37.         write(1,buff,strlen(buff));
38.         alarm (10);
39.         hago_algo_de_trabajo();/*no ejecuta nada relevante para el problema */
40.         alarm(0);
41.         while((ret=waitpid(-1,NULL,0))>0) {
42.             nhijos++;
43.         }
44.         sprintf(buff,"Fin de ejecución. Hijos esperados: %d\n",nhijos);
45.         write(1,buff,strlen(buff));
46.   } else {
47.         strcpy(buff,"Voy a trabajar \n");
48.         write(1,buff,strlen(buff));
49.         alarm(10);
50.         hago_algo_de_trabajo();/*no ejecuta nada relevante para el problema */
51.         alarm(0);
52.         strcpy(buff, “Fin de ejecución\n”);
53.         write(1,buff, strlen(buff));
54.   }
55. }
```

Answer the following questions **(justify all your answers**) assuming that the program is called as follows:

%./ejercicio_exec 4

1. Draw the resulting process hierarchy and assign to each process an identifier that allows you to refer to them in the following questions.
2. Assuming that the execution of the function *do_some_work()* takes always **less** than 10 seconds:
    a. For each process, indicate the messages that would be output to the screen
    b. For each process, tell what signals it would receive
    c. Assume that we move the sentences in lines from 29 to 32 to line 23. Would it affect somehow to the answers to questions a and b? How?

3. Assuming that the execution of the function *do_some_work()* takes always **more** than 10 seconds
    a) For each process, indicate the messages that would be output to the screen:
    b) For each process, tell what signals it would receive:
    c) Assume that we the sentences in lines from 29 to 32 to line 23. Would it affect somehow to the answers to questions a and b? How? Can we guarantee that the result will always be the same?

Exercise 4. (T3)

Consider the following code (simplified) belongs to program suma_vect.c

```
1.  int *x=0,*y,vector_int[10]={1,2,3,4,5,6,7,8,9,10};
2.  void main(int argc,char *argv[])
3.  {
4.  int i=0;
5.  char buffer[32];
6.  // POINT A
7.  x=malloc(sizeof(int)*10);
8.  y=&vector_int[0];
9.  fork();
10. Calcula(x,y);  // Performs a computation based on  x and y values . The output is saved at x.
11. free(x);
12. }
```

At **POINT A**, we see the following file "maps" (for simplicity we have removed some lines and some data that we have not studied during the course).

```
08048000-08049000   r-xp    /home/alumne/SO/test
08049000-0804a000   rw-     /home/alumne/SO/test
b7dbd000-b7ee2000   r-xp    /lib/tls/i686/cmov/libc-2.3.6.so
b7ee2000-b7ee9000   rw-p    /lib/tls/i686/cmov/libc-2.3.6.so
b7efa000-b7f0f000   r-xp    /lib/ld-2.3.6.so
b7f0f000-b7f10000   rw-p    /lib/ld-2.3.6.so
bfdf9000-bfe0f000   rw-p    [stack]
ffffe000-fffff000   ---p    [vdso]
```

1. Fill in the following table and explain your answer (POINT A)

Variable	Address range where it could be	Region name
x		
y		
vector_int		
i		
buffer		

Explanation (associate variable type with the region where it located).

2. Explain why the region of the heap does not appear at this point in the process execution (POINT A).
3. After malloc, a new region appears with the following definition:

```
0804a000-0806b000  rw-p [heap]
```

   a) What happens to the child process when it tries to access the x variable?
   b) Will it have access to the same address space?
   c) Explain the size that we see in the heap compared to the size allocated in line 7. What does C library want to optimize when it reserves more space than was requested?
4. Write again lines 7 and 11 if we wanted to do the same program using directly the corresponding system calls.

Exercise 5. (T2)

In a general purpose OS that uses Round Robin scheduling policy:

1. Can we assert that the OS is or not preemptive? Justify your answer explaining the meaning of preemptive and how have you reached your answer
2. When is it done a context switch? What is the criteria to select the next process to run in the CPU?
3. What is the data structure that the OS stores the process's information? Indicate the main fields we can find inside this structure.
4. What are the cases in which a process will run in the CPU less time than the Quantum? List the cases, justify, and show the process state at the time to leave the CPU

Exercise 6. (T2,T3)

We have the "prog" program that is generated when you compile the following code (for simplicity, the error handling code is omitted):

```
1.  int recibido = 0;
2.      void trat_sigusr1(int signum) {
3.          recibido = 1;
4.      }
5.
6.      main(int argc, char *argv[]) {
7.      int nhijos, mipid;
8.      int ret, i;
9.      char buf[80];
10.     struct sigaction trat;
11.
12.     trat.sa_handler = trat_sigusr1;
13.     trat.sa_flags = 0;
14.     sigemptyset(&trat.sa_mask);
15.     sigaction(SIGUSR1,&trat,NULL);
16.
17.     nhijos = atoi(argv[1]);
18.     mipid = getpid();
19.     for (i=0; i<nhijos; i++) {
20.         ret = fork();
21.         if (ret > 0){
22.             if (mipid != getpid()) {
23.                 while(!recibido) ;
24.             }
25.             kill(ret, SIGUSR1);
26.             waitpid(-1, NULL, 0);
27.             sprintf(buf, "Soy el proceso %d y acabo la ejecución\n",getpid());
28.             write(1,buf,strlen(buf));
29.             exit(0);
30.         }
31.     }
32.     sprintf(buf, "Soy el proceso %d y acabo la ejecución\n",getpid());
33.     write(1,buf,strlen(buf));
34.     }
35.
```

Answer the following questions, assuming it is run in the shell as follows and any system call results in error:

%./prog 3

1. (T2) Represent the hierarchy of processes this program creates when running. Assign each process an identifier to refer to them in the remaining questions.

2. (T2) What process(es) will execute code lines 32 and 33?

3. (T2) In what order will processes write on screen? Can we ensure that the order is always the same? Indicate the order in which processes write

4. (T3) Assume that you run this program in a paging based system that uses COW optimization in the process creation and whose page size is 4KB. Suppose also that the region code of this program occupies 1KB, data region occupies 1KB and the stack region ocuppies 1KB. That 3 regions do not share any page. How much physical memory is needed to load and simultaneously run all processes that are created in this program?

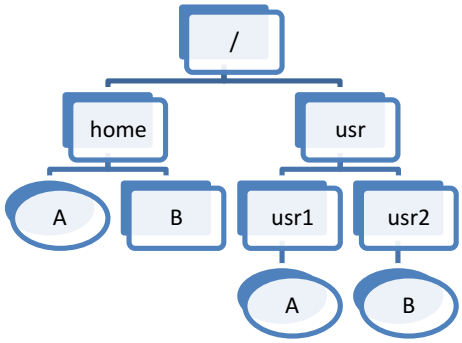
5. (T2) We want to modify this code in order to stop for three seconds before creating the next process. Indicate which lines of code you would add and their position in order to achieve this effect.

Exercise 7. (T4,T5)

1. (T4) What is the superblock of a filesystem? What kind of information can we find on it? What kind of information does it contain: data or metadata?
2. (T5) What is a race condition? What is a critical section? What is the relation between these two concepts?
3. (T4) Explain briefly the multi-level indexed allocation scheme used in UNIX-based systems, describing how many indexes (and their type) the inodes have. (You can draw a picture to show how it works).
4. (T5) List and explain briefly the advantages of using threads with respect to processes.

Exercise 8. (T4)

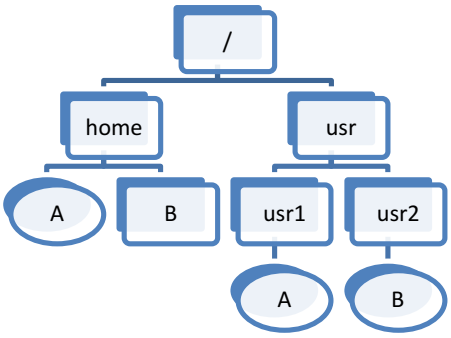
Given the following directory tree and the below information:



- Disk block size is 512 bytes and each inode takes 1 block.
- Both /home/A and /usr/usr1/A are hard-links and their sizes are 2 KB.
- /usr/usr2/B is a soft link to /home/usr/usr3 (a directory that does not exist anymore)

- Rectangle-shaped nodes stand for directories.
- The kernel implements a buffer cache (shared for both inodes and data blocks).

1. Assign inode numbers to all files and directories in the directory tree. Take into account the file type of each file.



2. Complete the following table with the inode attributes and the data-block contents. You must label file types as Linux does. For simplicity, there are only five data blocks; they are assigned in consecutive order.

Inode table (in disk)

inode number	0	1	2	3	4	5	6	7	8	9	10
#links											
File type											
Data blocks											

Data blocks

Block number	0	1	2	3	4	5	6	7	8

Data

Block number	9	10	11	12	13	14	15	16	17
Datas									

3. Assume that our current working directory is /home:
   a) Which is the absolute path to file /usr/usr1/A?
   b) Which is the relative path to file /usr/usr1/A?

4. Given the following program, executed in the described directory tree, answer the following questions (with a brief justification).

```
1. char c;
2. int fd,fd1,s,pos=0;
3. fd=open("/usr/usr1/A",O_RDONLY);
4. fd1=open("/usr/usr1/A",O_WRONLY);
5. s=lseek(fd1,0,SEEK_END);
6. while(pos!=s){
7.     read(fd,&c,sizeof(char));
8.     write(fd1,&c,sizeof(char));
9.     pos++;
10. }
```

   a) What is the functionality of this program?
   b) How many disk accesses performs sentence 3 assuming that the superblock is already loaded in memory. Describe which inodes and data blocks must be read.
   c) Does the sentence 4 add a new inode in the inode table (in memory)? Why?
   d) Assuming that no system call fails, how may iterations performs the loop from sentence 6 to 10?
   e) How many disc accesses performs the system call in sentence 7?
   f) Which is the value on the file pointer of file descriptor fd when loop exists.

Exercise 9. (T4)

We have the program "prog" generated after compiling the next code (for simplicity we have removed error code management):

```
1.  main(int argc char *argv[]) {
2.  int fd_pipe[2];
3.  int fd_file;
4.  int ret, pid1, pid2;
5.  char c;
6.  char buf[80];
7.  int size;
8.
9.  close(1);
10. fd_file = open (argv[1], O_WRONLY|O_TRUNC|O_CREAT, 0660);
11.
12. pipe(fd_pipe);
13.
14. pid1 = fork();
15. pid2 = fork();
16.
17. if (pid2 == 0) {
18.       while ((ret = read(fd_pipe[0],&c, sizeof(c))) > 0)
19.             write (1,&c, sizeof(c));
20. } else {
21.       while ((ret=read(0,&c,sizeof(c)) > 0)
22.             write(pipe_fd[1], &c, ret);
23.
24.       while (waitpid(-1, NULL, 0) > 0);
25.       sprintf(buf, "Fin ejecución\n");
26.       write(2,buf,strlen(buf));
27. }
28. }
```

JUSTIFY the following questions assuming the program is executed in the command line in the following way:

%./prog fichero_salida < fichero_entrada

we already know the content of "fichero_entrada" : "abcdefghijklmnñopqrstuvwxyz"

1. How many processes create this code? How many pipe's ?

2. Represents with a picture the process hierarchy that will create this program when running it. Assign a PID to each process to refer to them later. You must include in the picture the pipes showing which processes read/write from/to the pipe and which processes read/write from/to "fichero_entrada","fichero_salida".

3. Complete the following tables to represent the initial values for the File Descriptors table, the Open File's table and the i-Node Table (at the beginning of the execution of the program).

File Descriptors Table

	OFT Entry
0	
1	
2	

Open File's Table

	#refs	Mod	offset	t. inodes Entry
0		rw	--	0
1		r	0	

i-Nodes Table

	#refs	inode
0	1	i-tty
1		

1. Complete now the tables to represent the values for the different tables and for all the processes. Complete them for the case that all the processes are at line 16.

File descriptor Table

	OFT Entry
0	
1	
2	
3	
4	

Open File's Table

	#refs	Mod	offset	t. inodes Entry
0		rw	--	0
1		r	0	
2				
3				
4				

i-Nodes Table

	#refs	inode
0	1	i-tty
1	1	i-fichEnt
2		
3		

2. Which processes will finish the execution?

3. Which lines of code, and in which position (use line numbers as reference) you will add to make all the processes finish their execution.

4. Which processes read from "fichero_entrada"? Is it possible to know which part of the file reads each process? If we repeat the execution, can we guarantee that all the processes will read exactly the same portion of the file?

5. At the end of the execution, which will be the content of the "fichero_salida" file? If we repeat several times the execution of this program, can we guarantee the file content will be always the same?

Exercise 10. (T4,T5)

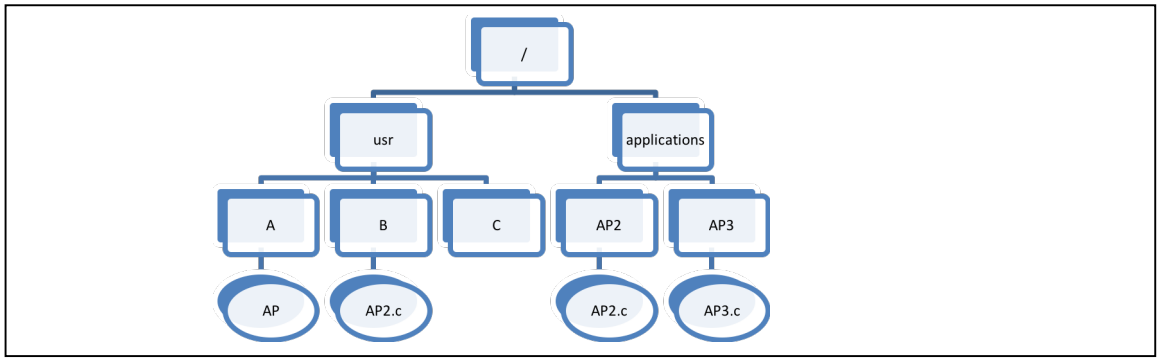
1. Briefly explain the impact of executing fork and exit on the input/output management tables.
2. List the input/output system calls that can create new entries in the file descriptor table.
3. The following code runs two non-linked processes that want to exchange their PIDs using two named-pipes: one per communication direction. Every process properly open the pipes and both of them firstly execute a read and then a write operation. Say whether it is correct or not, and justify your answer.

```
int su_pid,mi_pid;
mi_pid=getpid();
read(fd_lect,&su_pid,sizeof(int)); close(fd_lect);
write(fd_esc,&mi_pid,sizeof(int)); close(fd_esc);
```

4. In an Inode based filesystem, what are the two available types of links? Briefly explain their implementation (special file type (yes/no), information, etc).

Exercise 11. (T4)

Given the following directory scheme in a file system based on i-nodes:



We know that /usr/A/AP is a soft-link to /applications/AP2/AP2.c and that /usr/B/AP2.c and /applications/AP2/AP2.c are hard-links. Considering that:

- File type can be dir = directort, dat= regular file, link=softlink.
- The root directory has i-node 0.
- Squares in the figure represent directories.
- File AP3.c fills 3 blocks of data.

1. Fill up the following i-nodes and blocks of data in the following way:
    a) First, assign an i-node to each element in the directory scheme and write it down next to each name in the figure.
    b) Assign a block of data to each element in the directory scheme
    c) Complete the following figure filling up the fields in the i-nodes table and filling up the content that you can infer for each block of data.

I-nodes table in disk

i-node number	0	1	2	3	4	5	6	7	8	9	10
#links											
File type											
Data blocks											

Data blocks

Block number	0	1	2	3	4	5	6	7	8
Content									

Block number	9	10	11	12	13	14	15	16	17
Content									

2. Given the following code, describe the disk accesses performed by each system call (indicate how many disk accesses and what information are accessing). Let's assume that the system started just before executing this code and that during this execution this is the only process that is accessing files in the file system. We know that block size is 4096 Bytes, that there is a buffer cache in the system that is able to keep 1024 blocks at the same time and that the i-node is always written to disk after closing a channel.

```
    char buff[4096];
1.  fd=open("/applications/AP3/AP3.c",O_RDONLY);
2.  fd1=open("/usr/C/AP3.c",O_WRONLY|O_CREAT,0660);
3.  ret= read(fd,buff,sizeof(buff));
4.  lseek(fd,4096, SEEK_CUR);
5.  write(fd1,buff,ret);
6.  close(fd1);
7.  close(fd);
```

	Disk accesses (which ones)	Tables modification requirements (YES/NO)		
		Channels	Open files	I-nodes
1				
2				
3				
4				
5				
6				
7				

Exercise 12. (T4)

We have a program "prog" that results from compiling the following code (for the sake of simplicity error handling was omitted):

```
1.  #include <stdio.h>
2.  #include <unistd.h>
3.  #include <string.h>
4.
5.  main() {
6.
7.  int fd[2];
8.  int ret, pid1, pid2;
9.  char c;
10. char buf[80];
11.
12.
13. pid1 = fork();
14. pipe(fd);
15. pid2 = fork();
16.
17. if (pid2 == 0) {
18.       while ((ret = read(fd[0],&c, sizeof(c))) > 0)
19.             write (1,&c, sizeof(c));
20. } else {
21.       sprintf(buf, "This is my PID: %d\n",getpid());
22.       write(fd[1], buf, strlen(buf));
23.       while (waitpid(-1, NULL, 0) > 0);
24. }
25.
26. }
27.
```

Answer the following questions (justify your answers).

1. Assuming that the program is executed as follows:

%./prog

a) Analysis of the code

i. How many processes does this program create? How many pipes?

ii. Depict the process hierarchy that this process will create when it is executed. Assign to each process a pid so you can refer to each of them in the remaining questions. Depict also the pipes that will be created indicating which process(es) read(s) from each pipe and which process(es) write(s) to each of them.

iii. What process(es) will execute the lines 18 and 19 of the code? And those between lines 21 and 23?

iv. Is it necessary to add any code to synchronize the processes when accessing the pipe(s)?

v. What messages will be shown in the screen?

vi. What processes will finish their execution? Why? What changes would you introduce to the code to achieve that all processes finish their execution without changing the functionality of the code? It will be valued that the minimum possible number of changes are made.

b) Accessing kernel data stuctures

The figure below depicts the state of the file descriptors table, the open files table and the inode table at the beginning of the execution of this program. Complete the figure with the state of the file descriptors table of each process, the open files table and the inode table, assuming that all processes are in line 16.

Canal	Ent. TFA
0	
1	
2	
3	
4	
5	

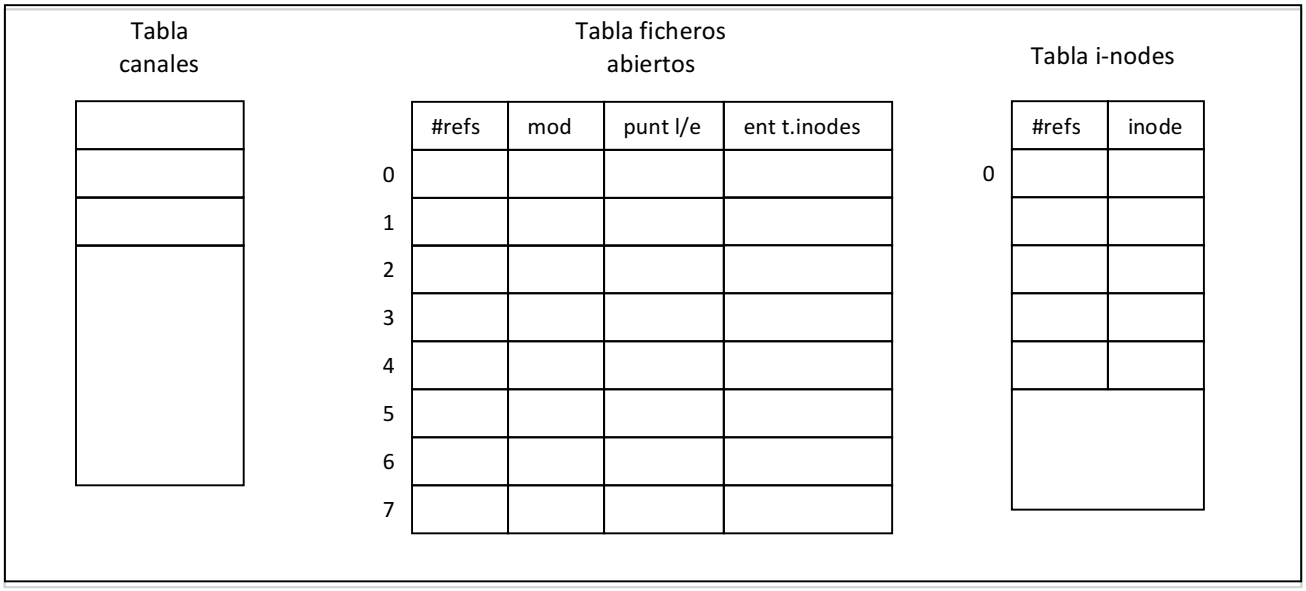
TFA	#ref	modo	Puntl/e	Ent.T. inodo
0	3	rw		0
1				
2				
3				
4				
5				

T.inodo	#ref	inodo
0	1	tty
1		
2		
3		
4		
5		

2. If we now assume that the program is executed as follows:

%./prog > f1

a) Would the state depicted in the figure above change in any way? Write in the figure below the initial state that we would have when executing the program as indicated.



b) What messages will appear to the screen?

c) What processes will finish their execution? Why? What changes would you introduce to the code to achieve that all processes finish their execution without changing the functionality of the code? It will be valued that the minimum possible number of changes are made.

Exercise 13. (T3)

1. What are the two main features of the Memory Management Unit (MMU)? (indicating what they are and what they consist of)
2. Explain what provides the virtual memory optimization, just about having paging and load on demand.

Exercise 14. (T2)

Given the following codes

Código 1

```
int recibido=0;
void f(int s)
{ recibido=1; }
void main()
{
struct sigaction trat;
trat.sa_flags = 0;
trat.sa_handler = f;
sigemptyset(&trat.sa_mask);
sigaction(SIGUSR1,&trat, NULL);
…
while(recibido==0);
recibido=0;
…
}
```

Código 2

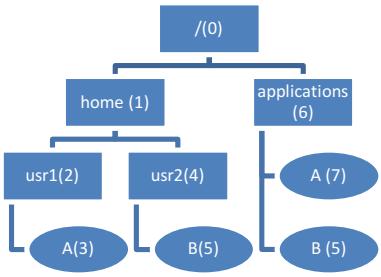
```
void f(int s)
{ }
void main()
{
struct sigaction trat;
sigset_t mask;
trat.sa_flags = 0;
trat.sa_handler = f;
sigemptyset(&trat.sa_mask);
sigaction(SIGUSR1,&trat, NULL);
…
sigfillset(&mask);
sigdelset(&mask, SIGUSR1);
sigsuspend();
…
}
```

Answer the following questions:

1. What is the code that performs an active waiting?
2. Is it required to reprogram the signal SIGUSR1 in the Code 2? What does it happen if it is not done?
3. In what memory section the variable “recibido” can be found?
4. Would it be any difference in the behavior of both codes if we know that, during their execution, they will receive a single SIGUSR1 event? Justify your answer and, if so, indicate behavior would be the most appropriate and why.

Exercise 15.(T4)

Assume a file system based on i-nodes that has the following structure (squares represent directories, i-node number is in brackets):



We know that:

- /home/usr1/A is a soft link to /applications/A
- /home/usr2/B is a hard link to /applications/B
- /applications/B is an empty file
- /applications/A uses 1000 bytes
- Each i-node and each directory uses 1 disk bloc
- Block size is 512bytes

And we execute the following code sequence:

```
1. char c[100];int i,fd_in,fd_out,ret;
2. fd_in=open("/home/usr1/A",O_RDONLY);
3. fd_out=open("/home/usr2/B",O_WRONLY);
4. ret=read(fd_in,c,sizeof(c));
5. while(ret>0){
6.    for(i=0;i<ret;i++) write(fd_out,&c[i],1);
7.    ret=read(fd_in,c,sizeof(c));
8. }
```

Answer the following questions (explain them all):

1. What i-node(s) is loaded in the i-node table in memory after running the system call at line 2?
2. How many disk accesses and which ones we have to do for running the system call at line 2? Show which ones correspond to i-nodes and which ones to data blocks (although not telling the number of data blocks). Assume there is no buffer cache.
3. Assume that a system call takes 10 ms, how long will take this code (total) just in system calls? (Show the cost (in ms) next to each line of code and the total amount at the end).

```
1.  char c[100];
2.  fd_in=open("/home/usr1/A",O_RDONLY); →
3.  fd_out=open("/home/usr2/B",O_WRONLY); →
4.  ret=read(fd_in,c,sizeof(c)); →
5.  while(ret>0){
6.     for(i=0;i<ret;i++) write(fd_out,&c[i],1); →
7.     ret=read(fd_in,c,sizeof(c));→
8.  }

TOTAL=
```

4. How many disk accesses (in total) takes the loop at lines 5-8 in case of …
    a) No buffer cache
    b) A buffer cache of 1000 blocks, assuming that the disk writings are synchronized when the file is closed.

Exercise 16. (T4)

Given the following codes prog1.c y prog2.c, without error checking to easy readability:

```c
/ * codigo de prog1.c */
main() {

int pid_h;
int pipe_fd[2];
char buf [80];

pipe(pipe_fd);

pid_h =fork();

dup2(pipe_fd[0], 0);
dup2(pipe_fd[1], 1);

close(pipe_fd[0]);
close(pipe_fd[1]);

sprintf(buf, "%d", pid_h);

execlp("./prog2", "prog2", buf, (char *) NULL);


}
```

```c
/ * codigo de prog2.c */
int turno_escr;

void trat_sigusr1(int signum) {
    turno_escr = 1;
}

main (int argc, char *argv[]) {

char buf [80];
int pid_dest;
int i,ret,valor_rec;
int valor  = getpid();
struct sigaction trat;
trat.sa_flags = 0;
trat.sa_handler=trat_sigusr1;
sigemptyset(&trat.sa_mask);

sigaction(SIGUSR1, &trat, NULL);

pid_dest = atoi(argv[1]);

if (pid_dest == 0) {
    pid_dest = getppid();
    write(1, &valor,sizeof(valor));
    turno_escr = 0;
    kill(pid_dest,SIGUSR1);
    valor ++;

} else {
    turno_escr = 0;
}

for (i = 0; i< 5; i++) {
    while (!turno_escr);
    ret=read (0,&valor_rec,sizeof(valor_rec));
    sprintf(buf,"%d",valor_rec);
    write(2,buf,ret);
    write(2,"\n",1);
    write(1,&valor,sizeof(valor));
    turno_escr = 0;
    kill(pid_dest,SIGUSR1);
    valor ++;
```

Assume that the current working directory holds both executables, prog1 and prog2, and we execute the following command line:

%./prog1

Briefly justify your answers to the following questions

1. How many processes will be created? How many pipes?
2. Depict the process hierarchy from the execution of this program. Provide a PID to every process to be able to identify them in the subsequent questions. Draw also the pipes showing what process(es) read/write from/to them.
3. What process(es) run prog2?
4. What messages are shown in the terminal?
5. Briefly describe the behavior of this code. Why are signals being used?

6. Assume that the code “prog1” takes up 1KB, whereas prog2 requires 4KB. The programs run on a computer with a memory System based on paging, the page size is 4KB and the Copy-on-Write optimization is available. How much memory do we need to sustain the execution of both processes at the same time, assumint that every process in the last instruction before the end of the code?

7. The following figure depicts the State of the file descriptor’s table, OFT, and I-node table at the beginning of the execution of this program. Properly fill the figure showing the state of the tables, assumint all processes are running the last instruction before the end of the execution.

FD Table

0	0
1	0
2	0

Open File Table

	#refs	Mode	offset	Ent I-node T.
0	3	rw	--	0

I-node Table

	#refs	inode
0	1	i-tty

Exercise 17. (T1,T3,T4)

1. (T1) A system that offers four execution modes, three with user privileges and one with system privileges… Can implement secure system calls?
2. (T3) We observe that two processes have the same logical and physical address space, which optimization offers this Operating System to make possible this situation? There must be some relation ship between the two processes?
3. (T3) Briefly describe the purpose of the swap area when applying the replacement algorithm (virtual memory optimization)

4. (T4) Which is the type of files . and .. we can find in directories. Which is the ¿De qué tipo son los ficheros . y .. que encontramos en los directorios y a que hacen referencia?

Exercise 18.(T2,T3,T5)

CASE 1 PROCESSES: If we know that a process occupies (4KB of code, 4KB of data and 4KB of stack) at user space and that the kernel uses PCBs of 4KB, how much space would be needed (in total, user+system, including the initial process) in the following cases:

1. The process makes the following system calls and the system doesn't implement COW

```
1. fork();
2. fork();
3. fork();
```

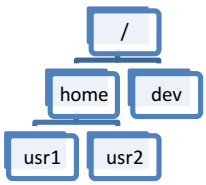
2. The process makes the same system calls, implements COW, and all the processes are in the line after the last fork

CASE 2 THREADS: The process makes the following calls, assuming that everything is correct and that the system doesn't offer COW. How much space would have been reserved in total (including the initial process) at the beginning of line 4?

```
1. pthread_create(&th1,función_th,null, null);
2. pthread_create(&th2,función_th,null, null);
3. pthread_create(&th3,función_th,null, null);
4.
```

Exercise 19. (T4)

We have the following filesystem (all elements are directories):



And we have the following code:

```
1. void f(int i){
2.   char c,buff[64];
3.   int fd;
4.   sprintf(buff,"/home/usr1/%d",i);
5.   fd=open(buff,O_WRONLY|O_CREAT,0777);
6.   while(read(0,&c,1)>0) write(fd,&c,1);
7. }
8. void main(int argv,char *argv[]){
9.    int i,pid;
10.    for(i=0;i<4;i++){
11.     pid=fork();
12.     if(pid==0){
13.      f(i);
14.      exit(0);
15.     }
16.    }
17.    f(i);
18.    while(waitpid(-1,null,0)>0);
19. }
```

Briefly describe what does this code (processes that are created, what each one does, etc.)

1. Given that at the beginning of the program, only 0, 1 and 2 channels are created, what will variable fd value for each process
2. How many entries will generate this code in the open files table?
3. Draw how will the file system be at the end of the program
4. We know that std input is the console, and we know that the user types the following characters 1234567890, and then presses ctrD, what will be the contents of each file?
5. If we know that the system does not offer buffer cache, and disk writes are done immediately, How many disk accesses will occur as a result in loop line 6?
6. Before you run this code we have inodes and data blocks with the following contents. (table below). Modify the table to represent how it will be, assuming that the processes are executed in the order of creation and that each process has read 1 character.

Inode list – stored in disk

Inode num	0	1	2	3	4	5	6	7	8	9	10
#links	4	4	2	2	2						
File type	d	d	d	d	d						
Data blocks	0	1	2	3	4						

Datablocks

Block num.	0	1	2	3	4	5	6	7	8
Data	. 0 .. 0 home 1 dev 2	. 1 .. 0 usr1 3 usr2 4	. 2 .. 0	. 3 .. 1	. 4 .. 1				

Block num.	9	10	11	12	13	14	15	16	17
Data									

Exercise 20. (T2,T4)

Consider the program “prog” generated after compiling the following code (in order to simplicity we omit errors control code):

```
1.  main() {
2.  int fd_pipe1[2];
3.  int fd_pipe2[2];
4.  int ret, i;
5.  char buf[80];
6.      pipe(fd_pipe1);
7.      ret = fork();
8.      if (ret > 0){
9.          dup2(fd_pipe1[1],1); close(fd_pipe1[1]); close(fd_pipe1[0]);
10.         execlp("cat","cat",(char *) 0);
11.     } else {
12.         close(fd_pipe1[1]);
13.         pipe(fd_pipe2);
14.         ret = fork();
15.
16.         if (ret > 0) {
17.             dup2(fd_pipe1[0],0); close(fd_pipe1[0]);
18.             dup2(fd_pipe2[1],1); close(fd_pipe2[1]); close(fd_pipe2[0]);
19.             execlp("cat","cat",(char *) 0);
20.
21.         } else {
22.             close(fd_pipe1[0]);
23.             dup2(fd_pipe2[0],0), close (fd_pipe2[0]); close(fd_pipe2[1]);
24.             execlp("cat", "cat", (char *) 0);
25.
26.         }
27.     }
28.
29.     write (2, "Fin", 3);
30. }
```

Answer the following questions. Suppose we executed the program with the following command and no syscall returns errors. Explain your answers:

%./prog < input_file > output_file

And suppose input_file exists and its content is “abcdefghijklmnñopqrstuvwxyz”

1. How many processes creates this program? How many pipes? Construct a process’s hierarchy diagram of the program in execution. Give to each process an identifier to refer them in the next questions. Also show the created pipes indicating which process(es) read(s) from them and which one(s) write(s) on each. On the diagram, mark (using arrows for the file access mode: read/write) which processes access to the file “input_file” and “output_file”.

2. Complete the following figure with the information needed to show the channels (user file descriptors) table for each process, the opened file table and the inode table. Suppose that all the processes are just before the end of their executions.

Channel table

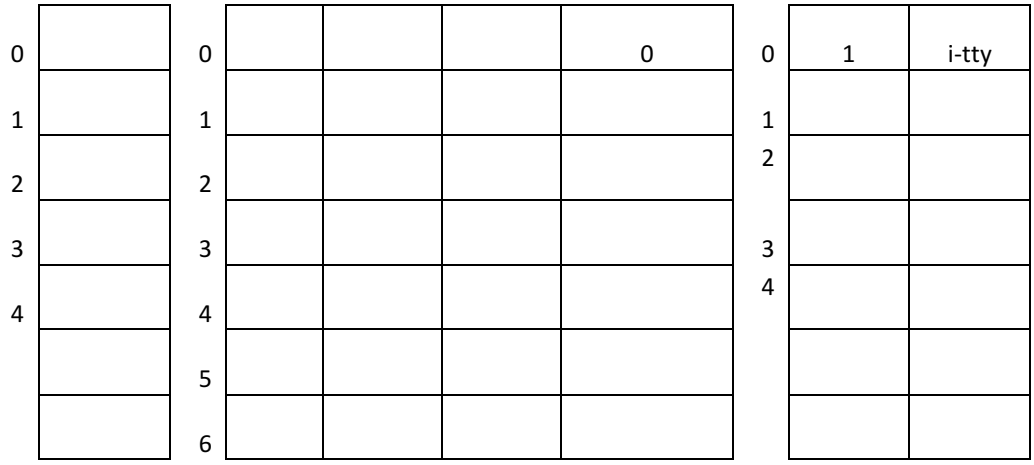
Ent. TFA

Opened file table

#refs	Mod	Punt l/e	Ent t. inodes

i-node table

#refs	inode



3. What's the content of output_file at the end of execution? What messages appear on screen?

4. All running process will end up?

5. We want to change the code so that from time to time (eg every second) it displays a message on the screen. We can think of the following code modification (changes are in bold):

```
1.  int trat_sigarlm(int signum) {
2.  char buf[80];
3.      strcpy(buf, "Ha llegado alarma!");
4.      write(2,buf,strlen(buf));
5.      alarm(1);
6.  }
7.  main() {
8.  int fd_pipe1[2];
9.  int fd_pipe2[2];
10. int ret, i;
11. char buf[80];
12. struct sigaction trat;
13.    trat.sa_flags = 0;
14.     trat.sa_handler = trat_sigalrm;
15.     sigemtpyset(&trat.sa_mask);
16.     sigaction(SIGARLM ,trat_sigalrm,NULL);
17.     alarm(1);
18.     pipe(fd_pipe1);
19.     ret = fork();
20.     // There are no more changes from this point on
21.    //…
22. }
```

Assuming the initial process takes more than 1 second to finish, will it work? What messages will now appear on screen? What processes will write them? Will it change the execution result at all?

Exercise 21.(T2)

1. Which is the difference between a blocked signal and an ignored signal? Which system call we need to use to block a signal? And to ignore it?

2. Describe the atomicity of system call sigsuspend. Use an example to support your explanation.
3. Given the following code, we ask for the signals that are blocked in the points marked as A, B, C, D, E, F, G, H and I.

```
1.  void sigusr1(int signum)
2.  { sigset_t mascara;
3.    /* C */
4.    sigemptyset(&mascara);
5.    sigaddset(&mascara, SIGINT); sigaddset(&mascara, SIGUSR1);
6.    sigprocmask(SIG_BLOCK, &mascara, NULL);
7.    /* D*/
8.  }
9.  void sigusr2(int signum)
10. { /* B */
11.   kill(getpid(), SIGUSR1);
12. }
13. void sigalrm(int signum)
14. { /* H*/
15. }
16. main()
17. { sigset_t mascara;
18.   struct sigaction new;
19.
20.   new.sa_handler = sigusr1; new.sa_flags = 0;
21.   sigemptyset(&new.sa_mask); sigaddset(&new.sa_mask, SIGALRM);
22.   sigaction(SIGUSR1, &new, NULL);
23.
24.   new.sa_handler = sigalrm; sigemptyset(&new.sa_mask);
25.   sigaction(SIGALRM, &new, NULL);
26.
27.   new.sa_handler = sigusr2;
28.   sigemptyset(&new.sa_mask); sigaddset(&new.sa_mask, SIGPIPE);
29.   sigaction(SIGUSR2, &new, NULL);
30.
31.   /* A */
32.   kill(getpid(), SIGUSR2);
33.   /* E */
34.   sigemptyset(&mascara); sigaddset(&mascara, SIGALRM);
35.   sigprocmask(SIG_BLOCK, &mascara, NULL);
36.   /* F */
37.   sigfillset(&mascara); sigdelset(&mascara, SIGALRM);
38.   alarm(2);
39.   /* G */
40.   sigsuspend(&mascara);
41.   /* I */
42. }
43.
```