Nombre alumno:
DNI:

Examen final de teoría de SO

Justifica todas tus respuestas de este examen. Cualquier respuesta sin justificar se considerará errónea.

Sistema de ficheros (*4 puntos*)

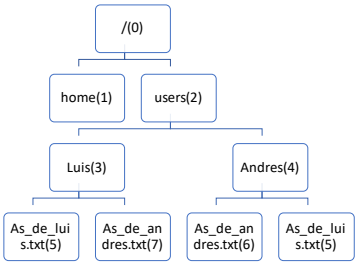
Deleted: *4*

Tenemos un sistema de ficheros basado en i-nodos tipo Linux. El tamaño de bloque es 512Bytes. Se dispone de una *buffer cache* en la que se guardan tanto i-nodos como bloques y que es de suficiente tamaño para guardar al mismo tiempo todo el contenido actual del sistema de ficheros. Las escrituras se gestionan de manera síncrona, es decir, si se escribe en un bloque se guarda en la *buffer cache* y al mismo tiempo se actualiza en el disco.

Tenemos un sistema de ficheros vacío, solo con el directorio / creado y ejecutamos la siguiente secuencia de comandos (el formato de ln es: ln [-s] file new_file).

```
mkdir home
mkdir users
cd users
mkdir Luis
mkdir Andres
echo “AAAA” > Luis/As_de_luis.txt
echo “AAAA” > Andres/As_de_andres.txt
ln /users/Luis/As_de_luis.txt Andres/As_de_luis.txt.
ln -s /users/Andres/As_de_andres.txt Luis/As_de_andres.txt
```

1. *(0,5 puntos)* Representa el árbol de directorios que genera esta secuencia de comandos. Indica junto al nombre del fichero el número de inodo que le has asignado (los inodos se asignan de manera secuencial).



2. *(1 punto)* Rellena la siguiente tabla que representa los inodos y bloques de datos de este sistema de ficheros.

Nombre alumno:
DNI:

Listado de Inodos

ID Inodo	0	1	2	3	4	5	6	7	
#enlaces	4	2	4	2	2	2	1	1	
Tipo	d	d	d	d	d	-	-	l	
BDs	0	1	2	3	4	5	6	7	

Listado de BDs

ID BD	0		1		2		3		4		5	6	7
Datos	.	0	.	1	.	2	.	3	.	4	"AAAA"	"AAAA"	"/users/Andres/ As_de_andres.txt"
	..	0	..	0	..	0	..	2	..	2			
	home	1			Luis	3	As_de_luis.txt	5	As_de_andres.txt	6			
	users	2			Andres	4	As_de_andres.txt	7	As_de_luis.txt	5			

3. Ejecutamos el siguiente código (correspondiente a p.c) desde el directorio "/". Ejecutamos el programa p de la siguiente forma:

p /users/Luis/As_de_luis.txt /users/Luis/tmp.txt

```
1.  int main(int argc,char *argv[])
2.  {
3.          int fdin,fdout,i;
4.          char c;
5.          fdin=open(argv[1],O_RDONLY);
6.          fdout=open(argv[2],O_WRONLY|O_CREAT|O_TRUNC,S_IRUSR|S_IWUSR);
7.          i=lseek(fdin,-1,SEEK_END);
8.          do{
9.                  lseek(fdin,i,SEEK_SET);
10.                 read(fdin,&c,sizeof(c));
11.                 write(fdout,&c,sizeof(c));
12.                 i--;
13.         }while(i>=0);
14.         return 0;
15. }
```

Contesta justificadamente las siguientes preguntas asumiendo que al ejecutar este programa la buffer cache está vacía y las tablas de E/S también. Asume que el superbloque está cargado en memoria.

a. *(0,5 puntos)* Describe la secuencia de accesos a superbloque, inodos y bloques (indica claramente los que son de lectura y de escritura) que hará la instrucción "`fdin=open(argv[1],O_RDONLY);`" Marca qué accesos llegarán finalmente a disco y cuáles no.

Inodo 0 lectura + BD 0 lectura + inodo 2 lectura+ DB 2+ inodo 3 lectura + DB 3 lectura + inodo 5 lectura

Nombre alumno:
DNI:

b. *(0,75 puntos)* Describe la secuencia de accesos a superbloque, inodos y bloques (indica claramente los que son de lectura y de escritura) que hará la instrucción "`fdout=open(argv[2],O_WRONLY|O_CREAT|O_TRUNC,S_IRUSR|S_IWUSR);`" Marca qué accesos llegarán finalmente a disco y cuáles no.

Inodo 0 lectura + BD 0 – lectura + inodo 2 lectura+ DB 2 lectura + inodo 3 lectura + DB 3 lectura -> modificación SB reserva inodo, reserva inodo 8, modificación BD 3 y modificación inodo 3

c. *(0,5 puntos)* Describe los accesos a superbloque, inodos y bloques (indica claramente los que son de lectura y de escritura) que harán las instrucciones de la línea 7-14. Indícalo por cada línea de código que incluya una llamada a sistema. Marca que accesos llegarán finalmente a disco y cuales no.

Este bucle lee el fichero de entrada y lo escribe en el nuevo fichero de salida. Modifica SB para reservar 1 bloque nuevo. Lee BD 5 , escribe en BD 8 y modifica inodo 8 con el nuevo tamaño. Los lseeks no generan ningún acceso a disco.

4. *(0,75 puntos)* Después de ejecutarlo, volvemos a ejecutar otra vez el programa con los siguientes parámetros: p /users/Luis/As_de_andres.txt /users/Luis/tmp.txt

   Describe la secuencia de accesos a supebloque, inodos y bloques (indica claramente los que son de lectura y de escritura) que hará la instrucción "`fdin=open(argv[1],O_RDONLY);`". Marca qué accesos llegarán finalmente a disco y cuáles no.

Inodo 0 lectura + BD 0 – lectura + inodo 2 lectura+ DB 2 lectura + inodo 3 lectura + DB 3 lectura -> inodo 7 (softlink) → Inodo 0 lectura + BD 0 – lectura + inodo 2 lectura+ DB 2 lectura + inodo 4 lectura + DB 4 + inodo 5

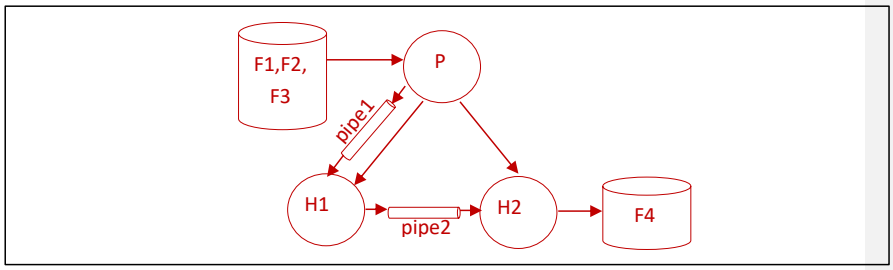
Nombre alumno:
DNI:

Procesos (*4 puntos*)

Nos proporcionan dos programas (sólo el ejecutable, no nos dan el código fuente) llamados filtro1 y filtro2. Cada uno de estos programas, ejecuta un bucle que lee de entrada estándar, hasta que detecta el final de datos, hace unas transformaciones, y escribe el resultado de las transformaciones en salida estándar (las únicas llamadas a sistema que hacen relevantes para este enunciado son read y write). Queremos hacer un programa que ejecute las dos transformaciones de filtro1 y filtro2 sobre una serie de ficheros que recibe como parámetro. El código que proponemos es el que muestra la figura 1 (al final de este enunciado). Contesta razonadamente a las siguientes preguntas, suponiendo que ninguna llamada a sistema devuelve error, y que ejecutamos este programa con el siguiente comando:

% cadena_filtrado f1 f2 f3 > f4

1. *(0,5 puntos)* Representa la jerarquía de procesos que crea este código, indicando claramente para cada proceso de qué dispositivo lee y en qué dispositivo escribe. Asigna a cada proceso un identificador para poder referirte a ellos.



2. *(0,5 puntos)* Representa el estado de las tablas de entrada/salida, suponiendo que se acaba de ejecutar la línea 26 del código (primer fork).

Tabla de Canales

	Entrada TFA
0	0
1	1
2	0
3	2
4	3
5	4
6	5

Tabla de Ficheros abiertos

	refs	modo	Posición l/e	Entrada T.inodo
0	4	RW	-	0
1	2	W	0	1
2	2	R	-	2
3	2	W	-	2
4	2	R	-	3
5	2	W	-	3
6				
7				

Tabla de iNodo

	refs	inodo
0	1	/dev/tty
1	2	F4
2	2	Pipe1
3	2	Pipe2
4		
5		

Nombre alumno:
DNI:

Tabla de Canales Hijo 1 Entrada TFA

0	**0**
1	**1**
2	**0**
3	**2**
4	**3**
5	**4**
6	**5**

3. *(0,5 puntos*) Suponiendo que todos los signals que reciben los procesos creados por cadena_filtrado son enviados por él mismo, indica para cada proceso qué signals recibe. ¿Se puede saber en qué línea(s) de código estará(n) cuando ejecute(n) el tratamiento? ¿Por qué?

El padre recibe el sigalrm programado por el alarm de la línea 53 y se tratará en la línea 54, ya que ese signal sólo está desbloquedado cuando estamos en el sigsupend

4. *(0,5 puntos*) ¿Aparece algún mensaje en pantalla?

Ninguno. Las escrituras en salida estándar se harán sobre F4, y la única escritura por el canal de errores se hace como tratamiento del sigpipe que en ningún momento se recibe.

5. *(0,5 puntos)* ¿Acabarán todos los procesos la ejecución?

Si. El padre después de leer todos los ficheros y de escribirlos en la pipe cierra el extremo de escritura en pipe1, lo cuál hace que H1 detecte el final de entrada de datos y salga de su bucle de lectura y acabe. Al acabar H1 también se cierra el único extremo de escritura en pipe2 con lo cual H2 también acaba la ejecución. El padre saldrá del waitpid y también acabará la ejecución.

6. *(0,75 puntos)* Supongamos que los execlp de las líneas 30 y 39 se ejecutan antes de que pasen 3 segundos de ejecución y que devuelven error. ¿Afectará a los mensajes que veremos en pantalla?

Nombre alumno:
DNI:

Si. En este caso los hijos salen por los exit de las líneas 31 y 40 dejando a pipe1 sin lectores. Por lo tanto, cuando el padre intente escribir en la pipe recibirá el SIGPIPE que tratará en el sigsuspend (cuando esté desbloqueado)

7. *(0,75 puntos)* Supongamos que queremos limitar el tiempo de ejecución de los programas filtro1 y filtro2: si pasados 10 segundos el proceso no ha acabado queremos que aborte la ejecución. Se nos ocurre añadir un alarm(10) antes de la ejecución de execlp (entre las líneas 29 y 30, y entre las líneas 38 y 39) .¿Conseguiremos nuestro objetivo?

No. Porque el SIGALRM se bloquea en la línea 17, los hijos heredan la máscara de signals bloqueados y no se pierde en el execlp. Aunque recibirán el signal no lo tratará y por tanto no abortará la ejecución

Nombre alumno:
DNI:

```
1.  trat_sigpipe(int s) {
2.  char msg[50];
3.          sprintf(msg,"Signal %d\n",s);
4.          write(2,msg,strlen(msg));
5.  }
6.  trat_alrm(int s) {}
7.
8.  main(int argc, char *argv[]){
9.  int nfichs = argc - 1;
10. int pipe1[2], pipe2[2];
11. int pidh,i,fd,ret;
12. char buff[256];
13. struct sigaction trat;
14. sigset_t mask;
15.
16. sigfillset(&mask);
17. sigprocmask(SIG_BLOCK,&mask,NULL);
18. sigemptyset(&trat.sa_mask);
19. trat.sa_flags=0;
20. trat.sa_handler=trat_alrm;
21. sigaction(SIGALRM, &trat, NULL);
22. trat.sa_handler=trat_sigpipe;
23. sigaction(SIGPIPE, &trat, NULL);
24. pipe(pipe1);
25. pipe(pipe2);
26. pidh=fork();
27. if (pidh == 0) {
28.         dup2(pipe1[0],0); close(pipe1[0]);close(pipe1[1]);
29.         dup2(pipe2[1],1); close(pipe2[1]);close(pipe2[0]);
30.         execlp("./filtro1", "filtro1", NULL);
31.         exit(1);
32. }
33. if (pidh>0) {
34.         close(pipe1[0]); close(pipe2[1]);
35.         pidh=fork();
36.         if (pidh==0){
37.                 close(pipe1[1]);
38.                 dup2(pipe2[0],0); close(pipe2[0]);
39.                 execlp("./filtro2", "filtro2", NULL);
40.                 exit(1);
41.         }
42. }
43. close(pipe2[0]);
44. sigfillset(&mask);
45. sigdelset(&mask,SIGALRM);
46. sigdelset(&mask,SIGPIPE);
47. for (i=1;i<argc;i++) {
48.         fd=open(argv[i], O_RDONLY);
49.         while((ret=read(fd,buff,sizeof(buff)))>0){
50.                 write(pipe1[1], buff, ret);
51.         }
52.         close(fd);
53.         if (i<(argc-1)){
54.                 alarm(3);
55.                 sigsuspend(&mask);
56.         }
57.
58. }
59. close(pipe1[1]);
60. while(waitpid(-1,NULL,0) >0);
61. }
```

figura 1 Código cadena_filtrado

Nombre alumno:
DNI:

Memoria (2 *puntos*)

A continuación tienes el código del programa mem.c. Se ha eliminado el control de errores para facilitar la lectura.

```
1.int i;
2.
3.void main() {
4.
5.    int *buffint1;
6.    int buffint2[500];
7.
8.    buffint1=sbrk(500*sizeof(int));
9.
10.   for (i=0;i<500;i++) buffint2[i]=i;
11.   fork();
12.
13.   for (i=0;i<500;i++) buffint1[i] =buffint2[i]+i;
14.
15.    waitpid(-1,NULL,0);
16. }
```

El programa "mem.c" se ejecuta en una máquina con 1 única CPU, con SO con gestión paginada de la memoria. Un entero ocupa 4 Bytes, un puntero ocupa 4 bytes, el tamaño de página es de 2048 bytes. El código de mem.c ocupa 3000 bytes.

1. *(0,5 puntos)* Indica en la siguiente tabla el consumo de memoria física por región del programa mem.c en ejecución en la línea 7.

Región	Tamaño en páginas	Justificación
Código	2 páginas	El código ocupa 3000 bytes que se deben acomodar en dos páginas de 2048 bytes cada una
Datos	1 página	La variable i, esta declarada globalmente. Se aloja en la sección de datos
Pila	1 página	En la pila se almacena el puntero buffint1 (4 bytes) y el vector buffint2 (2000 bytes). Juntos caben en una página
Heap	0 páginas	En la línea 7 todavía no se ha reservado memoria dinamica

Nombre alumno:
DNI:

2. Indica en las siguientes tablas el consumo de memoria física por región del programa mem.c en ejecución suponiendo que los dos procesos están en la línea 14.

   a. *(0,75 puntos)* El sistema no ofrece la optimización COW

Comentarios generales que no te quepan en la tabla:

Si el sistema no ofrece COW no se compartirán páginas entre los procesos padre e hijo (creado en la línea 11)

Región	Tamaño en páginas	Justificación
Código	4 páginas	Las dos páginas del padre y dos páginas del hijo
Datos	2 páginas	Que incluyen la copia de la variable i del padre y la del hijo
Pila	2 páginas	1 para el padre y otra para el hijo, con buffint1 y buffint2
heap	2 páginas	En la línea el padre reserva 2000 bytes (1 página). Por la línea 11 (fork sin COW) y la línea 13 se debe copiar esta página al hijo

Nombre alumno:
DNI:

b. *(0,75 puntos)* El sistema sí ofrece la optimización COW

Comentarios generales que no te quepan en la tabla:

Al haber COW hay que analizar si se comparte alguna región

Región	Tamaño en páginas	Justificación
Código	2 páginas	Al haber COW y como el código no se modifica, se puede compartir.
Datos	2 páginas	La variable global "i" se modifica en los dos procesos, por tanto debe hacer dos copias, en dos páginas
Pila	1 página	El puntero buffint1 y el vector buffint2 no se modifican después del fork
Heap	2 páginas	El contenido del puntero buffint1 se va modificando para cada proceso. Cada uno debe tener su copia.