Nombre alumno:
DNI:

Examen de teoría de SO

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

Preguntas Cortas (*2 puntos*)

1. *(0,5 puntos)* Ejecutamos el siguiente comando:

```
$> ln A B
```

Sabiendo que "A" existe, "B" no existe, que disponemos permisos para crear elementos en el directorio actual y que hay espacio libre en el disco, el comando falla. ¿A qué puede ser debido?

Porque "A" es un directorio y no podemos crear hardlinks entre directorios.

2. *(0,5 puntos)* Define el concepto de fragmentación externa de memoria, indicando en qué modelo de gestión de memoria se puede producer.

Es memoria libre y no asignada pero no se puede asignar a un proceso por no estar contigua. No está reservada pero no sirve. Se produce en el modelo de segmentación.

3. *(0,5 puntos).* En un programa se ejecuta la siguiente línea:

```
ret = read(fd, buffer,100);
```

Asumiendo que **fd** apunta a un dispositivo virtual válido y que **buffer** está bien declarado y tiene tamaño suficiente, indica al menos un caso en que tras finalizar **read** la variable **ret** tenga el siguiente intervalo de valores: **1 ≤ ret ≤ 99**

- Cuando el dispositivo asociado a fd solo dispone de entre 1 y 99 bytes disponibles para lectura.
- Cuando, mientras se ejecuta la transferencia de datos al buffer el proceso recibe un signal capturado donde el flag SA_RESTART está desactivado.

Nombre alumno:
DNI:

4. *(0,5 puntos)* Supón que conoces la dirección de entrada de la rutina de kernel que implementa el servicio de la llamada a sistema write. ¿Se podría hacer un *call* directamente a esa rutina desde una aplicación de usuario, asumiendo que pasamos correctamente los parámetros?

- No es posible. Desde el espacio lógico de un proceso de usuario no es accesible el espacio de direcciones del kernel.

Gestión de memoria *(1 Punto)*

Tenemos el siguiente código del programa "memoria". Este programa se ejecuta en un sistema Linux con tamaño de página 4096 bytes y nuestro sistema implementa la optimización de COW.

```
1. #define M_SIZE 4096
2. void print_limit()
3. {
4.   char buffer[256];
5.   void* limit;
6.   limit = sbrk(0);
7.   sprintf(buffer,"Limite %p\n", limit);
8.   write(1,buffer,strlen(buffer));
9. }
10.void main(int argc,char *argv[])
11.{
12.  int  ret, i , *p1;
13.  print_limit();
14.  p1 = sbrk(M_SIZE * sizeof(int));
15.  print_limit();
16.  for (i = 0; i < M_SIZE; i++) p1[i] = 0;
17.  ret = fork();
18.  /* A */
19.  sbrk(-1 * M_SIZE * sizeof(int));
20.  print_limit();
21.}
```

Al ejecutarlo tenemos la siguiente salida:

```
[user@login]$ ./memoria
Limite 0x1971000
Limite 0x1975000
Limite XXXXXXXXX
Limite YYYYYYYYY
```

Contesta a las siguientes preguntas:

1. Rellena las líneas punteadas con la respuesta correcta.
   a. La variable p1 está en la región de memoria ……pila/stack……., la variable limit está en la región de memoria ……pila/stack…….

Nombre alumno:
DNI:

b. El valor que veremos en las XXXXXXXX es ……0x1971000…… y en las YYYYYYYYY es ……0x1971000……

2. Si nos dan la siguiente información: el tamaño de la pila es 4096 bytes, el código de este programa ocupa 1000 bytes, la variable p1 ocupa 8 bytes y el tamaño de un int son 4 bytes:
   a. ¿Cuántos marcos de página (páginas físicas) tendrán reservados para las regiones de pila en el punto A incluyendo los dos procesos?

En ese punto la pila ya no se comparte ya que tanto i como ret se han modificado. En total 1 página por proceso = 2 páginas.

   b. ¿Cuántos marcos de página (páginas físicas) tendrán reservados para las regiones de heap en el punto A incluyendo los dos procesos?

Se ha reservado 4 páginas (4096 x 4). El padre inicializa los datos, por lo que se reservan 4 páginas. Al hacer fork, como se aplica COW y ni padre ni hijo modifican los datos, entre los dos tendrán 4 páginas.

Procesos y Signals *(3 Puntos)*

Analiza el código que te mostramos a continuación y responde a las preguntas de la manera más detallada posible dentro del espacio de que dispones. Supón que ejecutamos el programa desde el

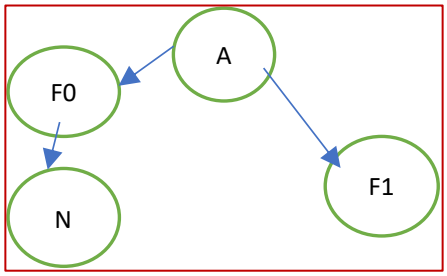
```
1. int beep = 0;
2. void ras(int s) {
3.   sigset_t nores, m;
4.   sigfillset(&nores);
5.   if (s == SIGALRM) {
6.     write(1, " FINAL!\n", 8);
7.     beep++;
8.   }
9.   if (beep < 1) {
10.    sigprocmask(SIG_SETMASK, &nores, &m);
11.    sigdelset(&m, SIGALRM);
12.    sigsuspend(&m);
13.  }
14.}
15.
```

```
16.int main(int argc, char *argv[]) {
17.    int i;
18.    struct sigaction sa;
19.    int n = atoi(argv[1]);
20.    sa.sa_handler = ras;
21.    sigfillset(&sa.sa_mask);
22.    sa.sa_flags = SA_RESTART;
23.    for (i = 0; i < 32; i++) {
24.         sigaction(i, &sa, NULL);
25.         alarm(i);
26.    }
27.    for (i = 0; i < n; i++) fork();
28.    while (waitpid(-1, NULL, 0) > 0);
29.    write(1, "\tEXAMEN!", 8);
30.    exit(0);
31.}
```

terminal con la siguiente línea de comandos: `$ ./a.out 2`

a) ***(0,5 puntos)*** Dibuja la jerarquía de procesos creada. Etiquétalos para poder referirte a ellos durante el resto de tu respuesta.

Nombre alumno:
DNI:



b) ***(0,5 puntos)*** ¿Qué signals están bloqueados cuando comenzamos a ejecutar la función `ras()`?

Todos. En la línea 21, llenamos la máscara sa.sa_mask y, en la línea 24, la fijamos como máscara de signals del proceso mientras se ejecute la rutina de atención al signal. En la línea 10, inicializamos la m con la máscara de signals actual del proceso.

c) ***(1,5 puntos)*** ¿Qué procesos escriben "EXAMEN!"? ¿Y "FINAL!"?

EXAMEN lo escriben, enseguida, los procesos N y F1 (ie, las hojas del árbol, que no tienen hijos). Esto provoca que F0 y A (los padres respectivos) reciban un SIGCHLD y queden suspendidos en línea 12 esperando un SIGALRM. Sólo A tiene programado un temporizador y, pasados 32 segundos, lo recibirá y escribirá FINAL. Entonces A queda esperando, en la línea 28, la muerte de F0. Pero F0 nunca morirá (a menos que reciba un SIGALRM desde otro proceso).

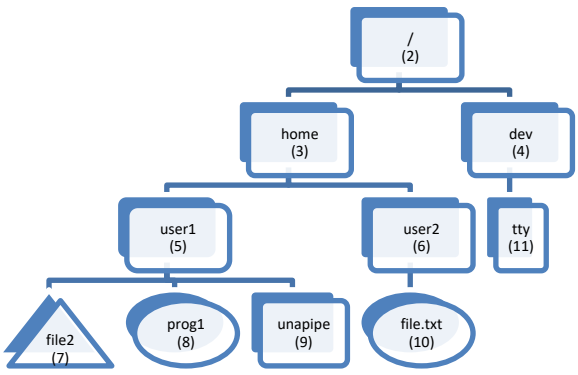
a) ***(0,5 puntos)***. ¿Qué línea de comandos tendrías que ejecutar pata que todos los procesos llegasen a la línea 29, escribiendo por pantalla?

```
$ kill -ALRM pidF0
```

Cuando F0 reciba un SIGALRM, saldrá del sigsuspend y escribirá FINAL. Como su hijo ya está muerto y beep vale 1, no se bloqueará en la línea 28, ni en la 12. Escribirá EXAMEN y morirá. Entonces su padre, A, saldrá del waitpid de la línea 28. Tampoco entrará en el if de la línea 9, porque su variable beep vale 1. Escribirá EXAMEN y morirá.

Entrada/Salida *(4 Puntos)*

Tenemos el siguiente SF basado en I-Nodos, con un tamaño de bloque de 1KB.



Nombre alumno:
DNI:

El fichero “file.txt” es un fichero de caracteres que contiene 2KB de datos. El fichero “file2” es un soft-link que apunta /home/user2/file.txt mediante un path absoluto. El fichero “prog1” es un ejecutable que ocupa menos de un bloque y es el resultado de compilar el siguiente código fuente:

```
1. main() {
2.    char c;
3.    int ret,fd;
4.    ret = fork();
5.    if (ret == 0) {
6.      fd=open(“/home/user1/unapipe”,O_WRONLY);
7.      while (read(0,&c,sizeof(c))>0)
8.          write(fd,&c,sizeof(c));
9.      exit(0);
10.     }
11.   fd=open(“/home/user1/unapipe”,O_RDONLY);
12.   while(read(fd,&c,sizeof(c)) > 0)
13.         write(1,&c,sizeof(c));
14.   waitpid(-1,NULL,0);
15. }
```

d) ***(0,75 puntos)*** Completa las siguientes tablas con la información que falta para representar esta jerarquía. El campo “path” sólo se utiliza para los soft links y contiene el path del fichero referenciado.

ID Inodo	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
#enlaces	4	4	2	2	2	1	1	1	1	1
Tipo	d	d	d	d	d	l	-	p	-	c
Path	-	-	-	-	-	/home/user2/file.txt	-	-	-	-
Tabla de índices a BD	0	1	2	3	4		5		6,7	

ID BD	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Contenido	. 2 .. 2 home 3 dev 4	. 3 .. 2 user1 5 user2 6	. 4 .. 2 tty 11	. 5 .. 3 file2 7 prog1 8 unapipe 9	. 6 .. 3 file.txt 10	codigo	datos	datos		

e) ***(3,25 puntos)*** Supón que el directorio actual de trabajo es /home/user1 y que ejecutamos el siguiente comando:

%./prog1 < /home/user1/file2 > /home/user1/file3.txt

Nombre alumno:
DNI:

1) ***(0,5 puntos)*** Completa el estado de las siguientes estructuras de datos suponiendo que los procesos se encuentran justo después del fork (entre la línea 4 y 5).

Tabla de Canales

	Entrada TFA
0	1
1	2
2	0
3	
4	

Tabla de Ficheros abiertos

	refs	modo	Posición l/e	Entrada T.inodo
0	2	rw	-	0
1	2	r	0	1
2	2	w	0	2
3				
4				

Tabla de iNodo

	refs	inodo
0	1	l tty1
1	1	l file.txt
2	1	l file3.txt
3		
4		

0	1
1	2
2	0
3	
4	

2) ***(0,5 puntos)*** Describe brevemente lo que hace este comando.¿Acabará la ejecución? Justifica tu respuesta

La ejecución del comando provocará la creación de /home/user1/file3.txt con una copia de /home/user2/file.txt.

Acaba la ejecución. El proceso hijo acabará cuando haya completado la lectura de file1.txt. En ese momento, desparecerá el único canal de escritura abierto para la pipe. El padre cuando vacíe la pipe, detectará que no quedan escritores y también saldrá de su bucle y acabará la ejecución.

3) ***(0,5 puntos)***. Indica cómo quedarán las estructuras de datos del apartado a) cuando el bucle de la línea 11 haya completado 2048 iteraciones. Si necesitas utilizar nuevos inodos o nuevos bloques asígnalos secuencialmente.

ID Inodo	2	3	4	5	6	7	8	9	10	11	12
#enlaces	4	4	2	2	2	1	1	1	1	1	1
Tipo	d	d	d	d	d	l	-	p	-	c	-

Nombre alumno:
DNI:

Path	-	-	-	-	-	/home/user2/file.txt	-	-	-	-	-
Tabla de índices a BD	0	1	2	3	4		5		6,7		8,9

ID BD	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Contenido	. 2 .. 2 home 3 dev 4	. 3 .. 2 user1 5 user2 6	. 4 .. 2	. 5 .. 3 file2 7 prog1 8 unapipe 9 file3.txt 12	. 6 .. 3 file.txt 10	codi go	datos	datos	COPIA DE DATOS DE 6	COPIA DE DATOS DE 7

Justificación: La creación de un nuevo fichero y la copia del contenido implica reservar un nuevo inodo, reservar dos nuevos bloques que contendrán lo mismo que los bloques de file.txt y añadir una nueva entrada en el directorio /home/user1

4) ***(0,5 puntos)*** Rellena la siguiente tabla indicando qué bloques de datos y qué inodos accederán los procesos (independientemente de si están en disco o es una copia en alguna estructura de datos en memoria) durante la ejecución de las siguientes sentencias del código.

Sentencia	**Bloques e inodos accedidos**	**Justificación**
7. read(0, &c, 1)	Bloque 6 y 7, Inodo 10	Lee todos los bloques de file.txt. La información sobre cuáles son esos bloques está en el inodo del fichero.
12. write(1,&c,1)	Bloque 8,9, Inodo 12	Escribe en los bloques asignados a file3.txt. La información sobre qué bloques son está en el inodo del fichero

5) ***(0,5 puntos)*** La línea 6 (open("/home/user1/unapipe",…);), ¿necesita acceder a algún bloque de datos? ¿Y a algún inodo? De ser así, indica la secuencia de accesos que realizará. En cualquier caso, justifica tu respuesta.

Nombre alumno:
DNI:

(superbloque) I2,B0,I3,B1,I5,B3,I9

6) ***(0,75 puntos)*** Indica cuáles de las siguientes sentencias modificarán la tabla de ficheros abiertos y cuáles modificarán la tabla de inodos (modificar puede ser añadir nueva entrada o cambiar el contenido de una ya existente). En la justificación describe los cambios que harán.

Sentencia	**Modifica TFA (si/no)**	**Modifica T. Inodos (si/no)**
6. fd=open("/home/user1/unapipe",O_WRONLY);	Si	Si
8. write(fd,&c,sizeof(c));	No	Si
11. fd=open("/home/user1/unapipe",O_RDONLY);	Si	Si
13. write(1,&c,sizeof(c));	Si	Si

Justificación

6 y 11: Cada open crea una nueva entrada en la TFA. Si el dispositivo no estaba en uso, crea una nueva entrada en la T. Inodos, si ya estaba en uso incrementa el número de referencias del inodo.

8. La escritura en una pipe no afecta a la entrada de la TFA pero deberemos actualizar en el inodo la fecha de último acceso

13. La escritura en un fichero modifica el puntero de lectura/escritura en la TFA. Además estamos haciendo crecer el fichero, así que en el inodo hay que actualizar tamaño, bloques (cuando se añaden) y fecha de último acceso.