Nombre alumno:
DNI:

Segundo control de teoría de SO

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

Preguntas Cortas (*2,5 puntos*)

1. (*0,5 puntos*) Un proceso está bloqueado en la siguiente instrucción:

```
ret = open ("/midispositivo",O_RDONLY);
```

La llamada se desbloquea devolviendo error y la variable `errno` contiene el valor `EINTR`

- ¿A qué puede deberse el bloqueo?
  /midispositivo es una pipe con nombre que no se ha abierto todavía por el extremo de escritura

- ¿Cuál es la causa del desbloqueo?
  El proceso que invoca open recibe un signal capturado.

2. *(0,5puntos)* Indica qué efectos tiene la ejecución de la llamada `exit()` sobre las tablas de gestión de entrada/salida.

- Exit() cierra todos los canales de un proceso
- Eso implica decrementar en 1 todas las referencias correspondientes en la TFA. Si en alguna entrada de la TFA el resultado de decrementar las referencias es 0, entonces se elimina la entrada de la TFA.
- Si se han eliminado entradas en la TFA, entonces en las entradas correspondientes de la TI se decrementan las referencias desde la TFA. Se eliminarán de la TI las entradas que después de decrementar tengan 0 como cantidad de referencias.

Nombre alumno:
DNI:

3. (*0,5 puntos*) ¿Es correcto el siguiente inodo?

N° Inodo	**34**
Tipo	**directorio**
#links	**4**
Bloques de datos	***(vacio)***

No. Un directorio siempre tiene al menos un bloque de datos, que contiene los links "." y ".."

4. *(0,5 puntos)* Tenemos el programa lector y el programa escritor que pretendemos que intercambien datos usando una pipe sin nombre. El código propuesto es el siguiente:

```
1. /* código de lector */
2. main() {
3. int fd[2];
4. int ret;
5. char buf[80];
6.      pipe(fd);
7.      ret=read(fd[0],buf,sizeof(buf));
8.      write(1,buf,ret);
9.}
```

```
1. /* código de escritor */
2. main() {
3. int fd[2];
4.      pipe(fd);
5.      write(fd[1],"hola",4);
6.}
```

Se ejecutan los dos códigos al mismo tiempo en dos ventanas diferentes sin hacer ninguna redirección de entrada/salida. ¿Funcionará el intercambio de datos? ¿Acabarán los dos procesos la ejecución? ¿Veremos algún mensaje en la ventana del lector?

No funcionará, cada proceso creará una pipe ordinaria diferente. El proceso escritor dejará el mensaje en su pipe y acabará. El proceso lector intentará leer de su propia pipe y se quedará bloqueado para siempre. No aparecerá ningún mensaje en pantalla.

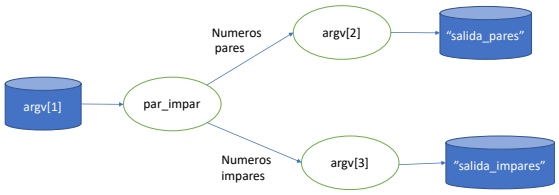
5. (*0,5 puntos*) Indica la diferencia entre el campo "`#referencias`" en una entrada de la tabla de inodos y el campo "`#enlaces`" de un inodo.

- #referencias en una entrada de la tabla de inodos indica el la cantidad de entradas que hacen referencia a ese inodo desde la Tabla de Ficheros Abiertos
- #enlaces en un inodo indica el número de nombres diferentes (hard-links) que tiene ese inodo.

Nombre alumno:
DNI:

Par_impar (*4 puntos*)

Queremos hacer un programa que llamaremos “par_impar” que hará el siguiente esquema de procesos y comunicaciones.



El programa recibe tres parámetros, el nombre de un fichero con enteros, y dos nombres de programas. El primero (en argv[2]) solo trabaja con **números pares** y el segundo solo con **números impares.** Ambos trabajan con los canales estándar de entrada/salida y con los números en formato binario.Queremos que la salida de procesar los números pares/impares vaya a unos nombres de fichero fijo, como tenéis en el esquema, y que ha de contener solamente la salida de cada ejecución. La comunicación entre el programa principal (par_impar) y los otros dos se hará utilizando pipes con nombre que ya estarán creadas previamente y que se llaman “pares” y “impares”. Nos dan el código de la figura 1 (al final de este enunciado) que está incompleto y no funciona como se desea.

1. *(1,5 puntos)* Analiza el código y modifica las siguientes tablas de E/S para reflejar el estado que tendrían cuando los procesos creados en este programa (incluido en inicial par_impar) estaría, respectivamente en las líneas 18 y 24, y el inicial en la línea 34. Os damos el estado de las tablas al inicio del main del programa par_impar.

Tabla de Canales

	Entrada TFA
0	0
1	0
2	0
3	5
4	6
5	7

Tabla de Ficheros abiertos

	refs	modo	Posición l/e	Entrada T.inodo
0	9	RW	-	0
1	1	R	-	1
2	1	W	0	2
3	1	R	-	3
4	1	W	0	4
5	1	R	File_size	5
6	1	W	-	1
7	1	W	-	3

Tabla de iNodo

	refs	inodo
0	1	/dev/tty
1	2	Pares
2	1	Salida_pares
3	2	Impares
4	1	Salida_impares
5	1	Fichero argv[1]

	Tabla de Canales Hijo 1 Entrada TFA
0	0
1	0
2	0
3	1
4	2

	Tabla de Canales Hijo 2 Entrada TFA
0	0
1	0
2	0
3	3
4	4

Nombre alumno:
DNI:

2. *(1 punto)* Para cada uno de los procesos creados (para números pares e impares), indica si habría que hacer alguna modificación para conseguir el comportamiento especificado en el enunciado respecto a los dispositivos de entrada/salida de datos (lectura entrada estándar y escritura salida estándar). **Justifícalo** e **incluye el código** necesario para que así sea en caso negativo.

Programa para pares: No . solamente hemos abierto los ficheros pero no accederemos a ellos. Hay diferentes soluciones pero una posible es hacer
close(0); p1=open("pares"…);
close(1); fd=open("salida…");

Programa para impares: Exactamente igual, no estamos accediendo a los ficheros ya que no están en los canales standard. Habría que hacer el equivalente:
close(0); p1=open("impares"…)
close(1);fd=open("salida.."…)
Otras posibles soluciones incluyen usar dup o dup2. Por ejemplo:
p1=open("impares"…); dup2(p1,0); close(p1)

3. *(0,5 puntos*) ¿Conseguiremos que la salida de los programas para números pares e impares se guarde en el fichero especificado? ¿Contendrá únicamente la salida correspondiente a la ejecución actual? **Justifícalo** e **incluye el código** necesario para que así sea en caso negativo.

No, ya que estos flags solo son correctos al abrir un fichero que ya existe. Para crear el fichero hemos de usar los flags de creación, incluido el O_TRUNC y los permisos que queremos asignar.

fd=open("salida_pares",O_WRONLY|O_CREAT|O_TRUNC,S_IRUSR|S_IWUSR)

4. *(0,5 puntos)* Analiza el código del proceso padre e indica si realiza correctamente la funcionalidad descrita en el enunciado y si terminaría correctamente. **Justifícalo** e **incluye el código** necesario para que así sea en caso negativo.

No, aún sería necesario cerrar el canal de escritura de las pipes después del bucle de lectura de datos ya que sino se quedarán los procesos hijos bloqueados en lectura y el padre en el waitpid. Habría que incluir
close(p1);close(p2);

Nombre alumno:
DNI:

5. (0,5 puntos) ¿Qué efecto tendría en el código mover las líneas 27-33 justo después de la línea 13 (antes del primer fork)?

No seria correcto ya que entonces el proceso inicial se quedaría bloqueado al hacer el open de la pipe en escritura ya que no habría ningún lector todavía.

```
1./* par_impar */
2.void usage(char *app)
3.{
4.  char buffer[256];
5.  sprintf(buffer,"usage: %s fichero app1 app2\n",app);
6.  write(1,buffer,strlen(buffer));
7.  exit(0);
8.}
9.int main(int argc,char *argv[])
10. {
11.     int ret,fd,mi_num;
12.     int pid1,pid2,p1,p2;
13.     if (argc!=4) usage(argv[0]);
14.     pid1=fork();
15.     if (pid1==0){ /* Proceso números pares */
16.           p1=open("pares",O_RDONLY);
17.           fd=open("salida_pares",O_WRONLY);
18.           execlp(argv[2],argv[2],NULL);
19.     }
20.     pid2=fork();
21.     if (pid2==0){ /* Proceso números impares */
22.           p1=open("impares",O_RDONLY);
23.           fd=open("salida_impares",O_WRONLY);
24.           execlp(argv[3],argv[3],NULL);
25.
26.     }
27.     fd=open(argv[1],O_RDONLY);
28.     p1=open("pares",O_WRONLY);
29.     p2=open("impares",O_WRONLY);
30.     while((ret=read(fd,&mi_num,sizeof(int)))>0){
31.           if ((mi_num%2)==0) write(p1,&mi_num,sizeof(int));
32.           else write(p2,&mi_num,sizeof(int));
33.     }
34.     while(waitpid(-1,NULL,0)>0);
35.     return 0;
36. }
```

figura 1 Codigo par_impar

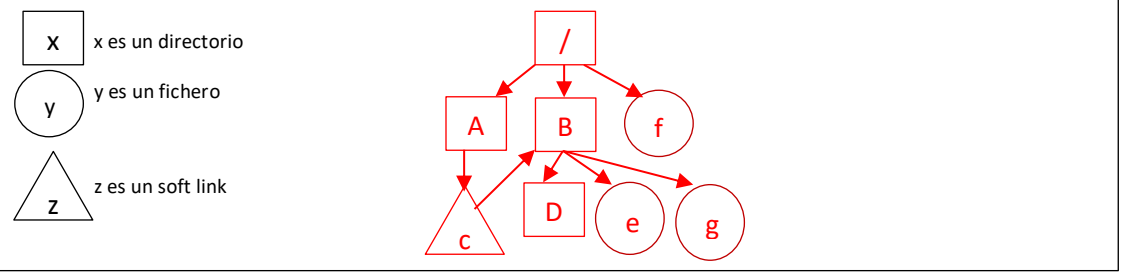
Nombre alumno:
DNI:

Sistema de ficheros (*3,5 puntos*)

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. El contenido actual del sistema de ficheros es el descrito por la siguiente figura.

Listado de Inodos									
ID Inodo	0	1	2	3	4	5	6		
#enlaces	4	2	3	1	2	2	1		
Tipo	d	d	d	l	d	-	-		
BDs	0	1	2	3	4	5	6		
Listado de BDs									
ID BD	0	1	2	3	4	5	6		
Datos	. 0 .. 0 A 1 B 2 f 5	. 1 .. 0 c 3	. 2 .. 0 D 4 e 5 g 6	/B	. 4 .. 2	Contiene 256 letras 'A'	Contiene 512 letras 'B'		

1. *(1 punto)* Representa el grafo de directorios para este contenido de i-nodos y bloques de datos. Sigue la notación para los nodos que se especifica en la figura. Para los soft-link añade una flecha indicando a qué otro fichero apunta.



2. Supongamos que el superbloque está cargado en memoria y que la buffer cache está vacía. Se ejecuta el siguiente el programa *prog* que muestra la siguiente figura:

```
1. /* código de prog */
2. char buf[512];
3. int fdr, fdw, ret;
4. fdr=open("/A/c/e", O_RDONLY);
5. fdw=open("/B/g", O_WRONLY);
6. ret=read(fdr,buf,sizeof(buf));
7. while(ret > 0){
8.    write(fdw,buf,ret);
9.    ret=read(fdr,buf,sizeof(buf));
10.  }
11. close(fdr);
12. close(fdw);
```

Nombre alumno:
DNI:

a) *(1 punto)* Indica para las siguientes sentencias cuántos **accesos a disco** serán necesarios. Para justificar tu respuesta indica, **para las sentencias que acceden a disco**, qué bloques y/o i-nodos accede la sentencia, y **para las sentencias que no acceden a disco**, por qué no es necesario hacerlo. Para las sentencias que forman parte del bucle indica el número de accesos totales teniendo en cuenta todas las iteraciones.

Sentencia	**Accesos a disco**
`4. fdr=open("/A/c/e", O_RDONLY);`	Acceso i-nodo /, acceso bloque /, acceso i-nodo A, acceso bloque A, acceso i-nodo c, acceso bloque c, acceso i-nodo B, acceso bloque B, acceso i-nodo e
`6. ret=read(fdr,buf,sizeof(buf));`	Acceso al bloque de e
`9. ret=read(fdr,buf,sizeof(buf));`	0 accesos (se compara el puntero l/e de la entrada de la TFA con el tamaño del i-nodo que está en la TI)

b) *(1,5 puntos)* Supongamos que desde la línea de comandos se ejecutan las siguientes sentencias (asume que es la primera vez que se ejecuta prog):

```
1.rm /f
2.prog
3.mkdir /B/D/H
```

Representa en la siguiente figura los i-nodos y bloques que hayan cambiado (considerando sólo los campos representados en la figura) con respecto a la situación inicial de este (sólo los que hayan cambiado, el resto no hace falta). Si es necesario eliminar algún bloque o algún i-nodo indícalo en la justificación. Si es necesario añadir algún bloque o i-nodo, asígnale un número que no esté en uso y representa su contenido. Si es necesario modificar el contenido de algún bloque o i-nodo, representa de nuevo todo su contenido.

Nombre alumno:

DNI:

Listado de Inodos modificados									
ID Inodo	5	4	7						
#enlaces	1	3	2						
Tipo	-	d	d						
BDs	5	4	7						
Listado de BDs modificados									
ID BD	0	6	4	7					
Datos	. 0 .. 0 A 1 B 2	Contiene 256 letras 'A' y 256 letras 'B'	. 4 .. 2 H 7	. 7 .. 4					

Justificación:

rm /f: Al borrar el fichero /f, tenemos que actualizar el directorio / quitando la entrada para f. Además hay que decrementar el número de referencias del inodo de f (el 5). Como no ha llegado a 0, el fichero no se elimina.

./prog: Al ejecutar el código de prog, el único cambio es que se sobreescriben los 256 caracteres iniciales de g (bloque 6).

mkdir /B/D/H: Al crear un subdirectorio en D, tenemos que actualizar el número de enlaces del i-nodo de D (pasan a ser 3 porque en H tendremos una referencia a D), tenemos que actualizar el bloque de D, con la entrada para H, tenemos que inicializar un nuevo i-nodo y un nuevo bloque.