1
SISTEMA DE FICHEROS

Problema 1.1

Dado un sistema con agrupaciones de 4 KiB.

Algunas de las estructuras del sistema son las mostradas en las tablas 1.1 y 1.2.

Nodo-i	Dueño (UID)	Permisos	Tipo	Agrupaciones	Tamaño	SUID, SGID
2	Root (0)	rwx r-x r-x	directorio	7644	4 KiB	
23	Pepa (34)	rwx r-x r-x	directorio	173	4 KiB	
87	Root (0)	rws --x --x	archivo	79752 y 79753	7,5 KiB	Activo SUID
273	Root (0)	rwx r-x r-x	directorio	7635	4 KiB	
324	Tere (622)	rwx --- ---	directorio	23	4 KiB	
753	Juana (62)	rwx --- ---	archivo	4546, 3874 y 22993	9,2 KiB	
823	Root (0)	rw- --- ---	archivo	12764	3 KiB	
876	Pepa (34)	rwx --- ---	directorio	453	4 KiB	

Tabla 1.1 Contenido de algunos nodos-i

Agrupación	Contenido
56	1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 .....
443	qwei oier oeiru qoieruower qpoweioru qpwoeir....
453	. 876
	.. 2
453	iue 8234
569	1234567890...r
4413	abcdefghijklme.....
7635	. 273
	.. 2
	pri1.txt 823
	edit.exe 87
7644	. 2
	.. 2
	doc 273
	Proa 876
	Prob 23
12764	ieqo ieoqo eir93o3hnf ‘wqn34209e rn349 wheew.....
79752	ABCDEFGHIJKLM.....

Tabla 1.2 Contenido de algunas agrupaciones

Se pide:

Establecer el esquema de directorios, sabiendo que el directorio raíz utiliza el nodo_i número 2.

El usuario con UID = 34 intenta ejecutar desde el shell el programa `/doc/edit.exe` para editar el fichero `/doc/pri1.txt`. Indicar si está autorizado a ejecutar dicho programa y si el programa podrá abrir el fichero a editar.

Recordando que el shell crea un proceso hijo para ejecutar el programa que le solicita el usuario, indicar las estructuras de datos relativas al servidor de ficheros que se verán involucradas en la ejecución anterior.

Problema 1.2

Dado el siguiente código:

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <unistd.h>
4  #include <sys/stat.h>
5  #include <fcntl.h>

6  #define BUFFSIZE 27

7  int main (int argc, char *argv[]) {
8     int miFd, miFd2, miFd3, pid;
9     char buff[BUFFSIZE]="abcdefghijklmnopqrstuvwxyz";
10    char buff2[10], buff3[10];
11    int pos1, pos2, pos3;

12    if (argc != 3) {
13       fprintf(stderr,"Error al introducir argumentos");
14       exit(1);  }

15    miFd= open(argv[1],O_CREAT|O_TRUNC|O_RDWR, 0700);

16    pid = fork();
17    switch (pid) {
18       case -1: perror("fork");
19           exit(2);
20       case 0:      /* Proceso hijo */
21          write(miFd,buff,BUFFSIZE);
22          link(argv[1],argv[2]);
23          miFd2=open(argv[2],O_RDWR);
24          miFd3=dup(miFd);
25          pos1=lseek(miFd,4,SEEK_SET);
26          read(miFd2,buff2,10);
27          read(miFd3,buff3,10);
28          pos2=lseek(miFd2,0,SEEK_END);
29          write(miFd2,buff2,10);
30          pos3=lseek(miFd3,0,SEEK_CUR);
31          write(miFd3,buff3,10);
32          close(miFd);
33          close(miFd2);
34          close(miFd3);
35          return 0;
36       default:     /* Proceso padre */
37          write(miFd,buff,BUFFSIZE);
38          close(miFd);
39          return 0;
40    }
41 }
```

Resolver las siguientes cuestiones:

a) *Explicar el código, dibujando la relación existente entre los distintos descriptores y ficheros para los procesos involucrados. Dibujar los nodos-i de los ficheros con los campos más significativos al completarse la ejecución de ambos procesos.*

b) *¿En qué consisten los argumentos que recibe el programa principal? Teniendo en cuenta las llamadas al sistema que se realizan en el código, ¿qué restricciones deben cumplir dichos argumentos para que la ejecución sea correcta?*

c) *¿Cuál es el contenido final de los ficheros involucrados? ¿Depende del orden de ejecución de los procesos padre e hijo? Si es así, indicar el contenido de los ficheros suponiendo que el proceso padre se ejecuta de forma completa antes que el proceso hijo.*

d) *¿Cuáles son los valores que reciben las variables* `pos1`, `pos2` *y* `pos3` *en las líneas 25, 28 y 31? Suponer que el proceso padre se ha ejecutado de forma completa antes de que comience a ejecutar el proceso hijo.*

e) *Dibujar un diagrama indicando cómo se modifican las entradas de directorios y nodos-i correspondientes cuando se lleva a cabo la llamada al sistema link() (línea 22).*

f) *¿Sería equivalente sustituir la línea 15 por:* `int miFd=creat(argv[1],0700);` *?*

Si no lo es, explicar cómo se ve modificada la ejecución del programa.

Problema 1.3

Sea un microsistema de ficheros tipo UNIX, que está simulado sobre memoria, con bloque de 1 KiB. Los componentes del sistema de ficheros son los siguientes:

- *Bloque de BOOT: Ocupa el bloque 1. No es utilizado en el problema.*
- *Superbloque: Ocupa el bloque 2. No es utilizado en el problema.*
- *Mapa de bits para los nodos-i del sistema: Ocupa el bloque 3. Cada bit representa el estado de un nodo-i del sistema: 0 significa que el nodo-i está libre y 1 implica que el nodo-i está ocupado.*
- *Mapa de bits para los bloques de datos: Ocupa el bloque 4. Cada bit representa el estado de un bloque de datos: 0 significa que el bloque está libre y 1 implica que el bloque está ocupado.*
- *Nodos-i del sistema: En este área se almacenan los nodos-i del sistema. Cada nodo-i ocupa 128 B.*
- *Bloques de datos y directorios: en este área se almacenan los bloques de datos y directorios.*

Responder razonadamente a las siguientes preguntas:

a) *Calcular el número máximo de ficheros del sistema.*

b) *Calcular el tamaño máximo que puede tener el sistema de ficheros.*

c) *Calcular el tamaño máximo que puede tener un fichero.*

d) *¿Por qué los bloques de datos y los directorios aparecen mezclados en un sistema de ficheros tipo UNIX?*

e) *Calcular el número máximo de caracteres que puede tener el nombre de un fichero de directorio en este sistema, suponiendo que cada bloque contiene 4 entradas de directorio.*

f) *Rellenar una tabla que represente los bloques del sistema de ficheros en el caso de que se tenga la estructura de ficheros de la figura 1.1. Los ficheros ordinarios aparecen encuadrados. El fichero* otro *contiene un texto de 3.745 caracteres.*

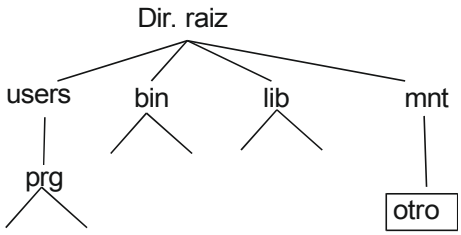

Figura 1.1

Problema 1.4

a) Implementar el programa `GenerarClave` *con los siguientes requisitos:*

- *El programa recibe como único parámetro el nombre de un fichero o directorio.*
- *En caso de haber recibido un directorio el programa emite –1 por su salida estándar y finaliza.*
- *Si el parámetro recibido es un fichero el programa deberá emitir por su salida estándar un número entero calculado de la siguiente manera: tamaño en bytes del fichero más la suma de todos los bytes del fichero.*

b) Se desea realizar el programa `InsertarClave` *con los siguientes requisitos:*

- *El programa recibe como argumentos el nombre de un directorio y el nombre de un fichero. Se debe suponer que tanto el directorio como el fichero existen y son válidos.*
- *Calcula la clave del fichero indicado como segundo argumento, usando para su cálculo el programa* `GenerarClave` *realizado en el anterior apartado.*
- *Una vez calculada la clave, crea en el directorio especificado como primer argumento un enlace simbólico con nombre el valor de la clave calculada, apuntando al fichero recibido como entrada.*

Por ejemplo, en la siguiente invocación del programa:

InsertarClave /var/claves/ /tmp/fichDatos.txt

si la clave calculada es 67534, se debería crear la siguiente entrada en el directorio `/var/claves/`*:*

lrwxr-xr-x 1 luis luis ***13*** *Jun 2 16:36 67534 -> /tmp/****fichDatos.txt***

c) Realizar el programa `ComprobarClaves` *con los siguientes requisitos:*

- *Recibe como parámetro de entrada el nombre de un directorio. Se debe suponer que este directorio solamente contiene entradas creadas por el programa* `InsertarClave`*.*
- *Recorre uno por uno todos los ficheros del directorio, comprobando que su clave es correcta.*
- *En caso de encontrar una clave incorrecta, imprime por su salida estándar el nombre del fichero correspondiente.*

d) Responda razonadamente. ¿Habría alguna diferencia si en vez de enlaces simbólicos el programa `InsertarClave` *creara enlaces físicos?, ¿qué ventajas e inconvenientes podría suponer esta situación?*

Problema 1.5

a) Dado el siguiente código, en el cual no se ha realizado ninguna comprobación de errores:

```
1  int main(int argc, char *argv[]) {
2     int fd, fd2, fd3;
3     pid_t pid1, pid2;
4     char buffer[1024]="12345678901234567890123456789012345";
5
6     fd=open(argv[1],O_RDWR);
7     pid1=fork();
```

```
8     if (pid1==0) {
9        fd2 = creat(argv[2], 0640);
10       fd3 = dup(fd);
11       read(fd3,buffer,10);
12       read(fd,buffer,10);
13       write(fd2,buffer,20);
14       close(fd);
15       close(fd2);
16       close(fd3);
17       return 0;
18    }
19    else {
20       pid2=fork();
21       if (pid2==0) {
22          fd2=open(argv[1],O_RDONLY);
23          fd3=open(argv[2],O_RDWR);
24          write(fd3, buffer, 10);
25          read(fd,buffer,10);
26          write(fd,buffer+10,10);
27          close(fd);
28          close(fd2);
29          close(fd3);
30          return 0;
31       }
32       else {
33          fd2=creat(argv[2],0755);
34          fd3=dup(fd2);
35          read(fd,buffer,10);
36          write(fd2,buffer,10);
37          write(fd3,buffer,10);
38          close(fd);
39          close(fd2);
40          close(fd3);
41          return 0;
42       }
43    }
44 }
```

Si se ejecuta este programa con los argumentos fich1 fich2 y el fichero fich1 tiene el contenido "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ", responder razonadamente a las siguientes cuestiones:

1. Si el orden en el que se ejecutan los procesos de forma completa es: proceso padre, primer proceso hijo, segundo proceso hijo, ¿cuál es el contenido del fichero fich2 al final de la ejecución?

2. Realizar el mismo ejercicio para el caso de que el orden de ejecución sea el inverso (segundo proceso hijo, primer proceso hijo, proceso padre).

3. Para el orden de ejecución del apartado 1, rellenar la siguiente tabla, con los valores correspondientes de la tabla en la línea ejecutada:

Num. línea	Puntero de posición del fichero fd1	Puntero de posición del fichero fd2	Puntero de posición del fichero fd3	Contenido de buffer

4. Una vez que el proceso correspondiente ejecuta el código de la línea 10, ¿qué modo de acceso tiene sobre el fichero cuyo descriptor es fd3? Responder a la misma cuestión para el caso del proceso que ejecuta el código de la línea 34.

5. El código correspondiente a la línea 26, ¿produciría un error de compilación?, ¿produciría un error de ejecución? Explicar gráficamente en qué consiste esta operación, dibujando el buffer y el fichero cuyo descriptor es fd.

6. ¿Qué diferencias habría en la ejecución del proceso que ejecuta el código de la línea 36, si dicha línea se sustituyera por `write(fd3,buffer,10);`*?*

Problema 1.6

En un sistema UNIX existen los siguientes usuarios:

- *alumno1 (UID = 101)*
- *alumno2 (UID = 102)*
- *alumno3 (UID = 103)*
- *alumno4 (UID = 104)*
- *root (UID = 0)*

El grupo alumnosSO, cuyo identificador es el 300, incluye los siguientes usuarios: alumno1, alumno2. Por su parte, el grupo alumnosDSO, cuyo identificador es el 400, incluye los siguientes usuarios: alumno3, alumno4. El usuario root pertenece al grupo root (GID = 0).

Si se lleva a cabo un listado del directorio /fichs en formato largo e incluyendo información sobre los nodos-i (ls –lai) se obtiene la siguiente información:

Nodo-i	Permisos	Enlaces	Usuario	Grupo	Tamaño	Fecha	Nombre
41190	drwxrwxrwx	2	root	root	4096	2004-05-11	.
2	drwxr-xr-x	26	root	root	4096	2004-05-11	..
36890	-rws--x--x	1	root	root	28004	2004-08-12	cambiarClaves
37820	-rwxr-s--x	1	alumno2	alumnosSO	28004	2004-03-16	cambiarClaves2
54321	-rw-------	1	root	root	80	2004-04-10	claves
46230	-rw-r--r--	1	alumno2	alumnosSO	10	2004-07-22	datos
13357	-rw-r--r--	1	alumno3	alumnosDSO	334	2004-09-23	pre.dat
12356	-rw-r--r--	1	root	root	16170	2004-05-11	variables.pdf

Los permisos `rws------` implican que el propietario del fichero tiene permiso de lectura, escritura y ejecución y además que el fichero tiene activado el bit SETUID. Los permisos `---r-s---` implican que el grupo tiene permiso de lectura y ejecución y además que el fichero tiene activado el bit SETGID.

Se va a suponer que el sistema en caso de asignar nuevos nodos-i, lo hará utilizando valores crecientes a partir de 58743 inclusive.

Los ficheros cambiarClaves y cambiarClaves2 son dos ejecutables con el mismo código. A continuación se describe el código de ambos:

```
int main (int argc, char *argv[])
{  int fd, acceso, pos;
   struct stat estado;
   char buffer[1024]="ABCDEFGHIJKLMNOPQRSTUVWXYZ";

   if (argc!=3) {
      fprintf(stderr,"Error\n");
      exit(1);
   }
   link (argv[1], argv[2]);

fd = open(argv[2], O_RDWR | O_APPEND);
if (fd>=0)
{
      write(fd, buffer, 10);
pos = lseek(fd, 20, SEEK_CUR);
      fstat(fd, &estado);
      printf("%d\n",estado.st_ino);
   }
   acceso = access(argv[2], R_OK);
   if (acceso ==0)
      unlink(argv[1]);
    return 0;
}
```

Responder a las siguientes cuestiones:

Si el usuario alumno1 lleva a cabo la siguiente ejecución:

```
$ /fichs/cambiarClaves /fichs/claves /fichs/claves2
```

a) *¿Cuál es el valor de la variable* `fd`*?*
b) *Después de ejecutar el servicio* `lseek`*, ¿cuál es el valor de la variable* `pos`*?*
c) *¿Cuál es el valor impreso por la salida estándar?*
d) *Al final de la ejecución, ¿Cuántos enlaces físicos tiene el fichero cuyo nodo-i es 54321?*

Si el usuario alumno3 lleva a cabo la siguiente ejecución:

```
$ /fichs/cambiarClaves2 /fichs/datos /fichs/datos2
```

e) *¿Cuál es el valor de la variable* `fd`*?*
f) *Al final de la ejecución, ¿cuántos enlaces físicos tiene el fichero cuyo nodo-i es 46230?*

Problema 1.7

En un sistema tipo UNIX se dispone de dos sistemas de ficheros A y B, soportados respectivamente en las particiones A y B. El sistema de ficheros B está montado en `/usr` *del sistema A, según se muestra en la figura 1.2. Dicha figura muestra, para cada fichero, su nombre después del montaje, si es un fichero directorio o normal, el nombre del dueño y de su grupo, así como los permisos.*

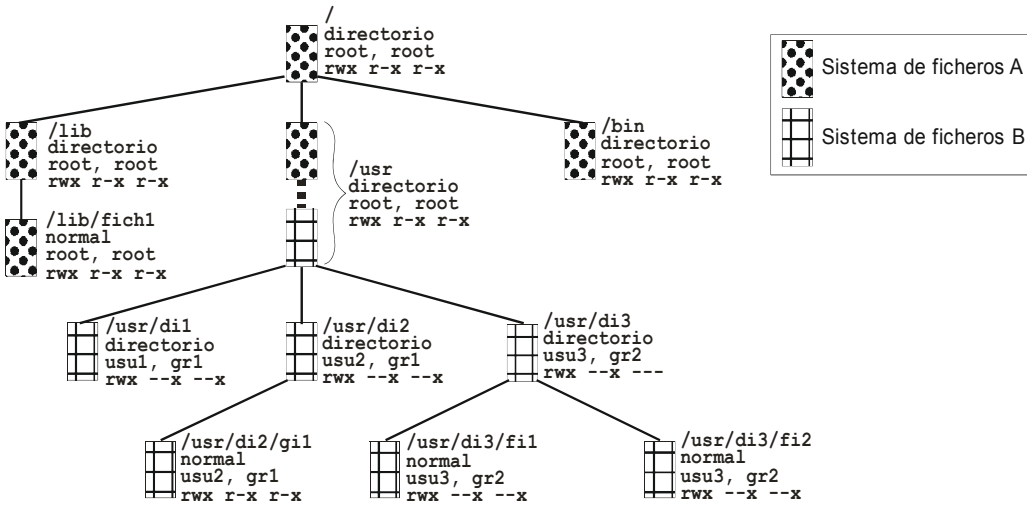

Figura 1.2

Consideraremos tres procesos: el X del usuario usu3 y grupo gr2, el Y del usuario usu1 y grupo gr1, y el Z del usuario root y grupo root. Inicialmente los procesos solamente tienen ocupadas las tres primeras entradas de sus tablas de descriptores fd.

Para cada paso de ejecución indicar todos y cada uno de los puntos siguientes, en caso de no modificarse alguno indicarlo expresamente:

1.- El valor devuelto por el servicio (por ejemplo, 0).

2.- Campos modificados en la tabla de descriptores (fd) del o de los procesos involucrados (por ejemplo, se rellena la entrada número 0 de la tabla fd del proceso M).

3.- Campos modificados en la tabla intermedia residente en memoria (por ejemplo, la tabla intermedia no se modifica).

4.- Modificaciones en nodos-i de ficheros de usuario en la tabla de nodos-i residente en memoria (por ejemplo, se incrementa el campo xxx del nodo-i del fichero yyy).

5.- Modificaciones en la partición A (por ejemplo, en la partición A se rellena un nuevo nodo-i y se añade una entrada en el directorio xxx con el valor yyy).

6.- Modificaciones en la partición B (por ejemplo, la partición B no se modifica).

Los pasos son los siguientes, que se ejecutan en el orden dado:

a) Proceso X: `fd1 = open ("/usr/di2/gi1", O-RDONLY);`
b) ProcesoY: `fd1 = open ("/usr/di2/fi1", O-RDONLY);`
c) Proceso X: `pid = fork ( );` *,ejecuta con éxito, generando el proceso XH*
d) Proceso XH: `fd1 = open("/lib/fich1", O-RDWR);`
e) Proceso Y: `li = symlink ("/lib/fich1", "/usr/di1/fich1");`
f) Proceso XH: `li = link ("/usr/di2/gi1", "/usr/di3/fich1");`
g) Proceso XH: `fd2 = dup(3);`
h) Proceso Y: `fd2 = open ("/usr/di1/fich1", O-RDONLY);`
i) Proceso Z: `lf = unlink ("/lib/fich1");`
j) Proceso Y: `n = read(fd2, buf, 128);`

Problema 1.8

Se desea implementar un programa que cifre los contenidos de todos los ficheros de un directorio. El programa recibirá dos argumentos: (1) el nombre de un ejecutable que hará de cifrador y (2) el directorio a cifrar. El programa no tiene que cifrar subdirectorios, sólo los ficheros que se encuentre en el directorio pasado como argumento. Las características del funcionamiento del programa son las siguientes:

- *Se recorre el directorio dado con la función recorrer_directorio.*
- *Durante el recorrido del directorio, por cada fichero que se localice, se llamará a la función cifrar_fichero.*
- *Para cifrar el fichero se ejecutará el cifrador, este ejecutable recibe por la entrada estándar los contenidos a cifrar y genera por la salida estándar dichos contenidos cifrados. Esta tarea la realiza la función cifrar_fichero.*
- *Al final de la ejecución, el directorio debe contener los mismos nombres de ficheros, pero sus contenidos deben de estar cifrados.*

a) Implementar la función recorrer_directorio.

b) El proceso de cifrado manipula dos ficheros, el original y el cifrado. Potencialmente, que estos dos ficheros tengan el mismo nombre puede ser un problema. Considerando que no se puede dar un fallo o caída en el programa durante el proceso de cifrado, existen tres alternativas para hacerlo:

1. Renombrar el fichero original, crear el fichero cifrado con el antiguo nombre del original, cifrar y finalmente borrar el fichero original renombrado. (Usando rename).
2. Abrir el fichero original, borrar el nombre de fichero manteniéndolo abierto y crear el fichero cifrado con dicho nombre. Luego cifrar y cerrar los ficheros. (Usando unlink)
3. Abrir el fichero original, enganchar la salida del cifrador a una tubería (pipe), cifrar, borrar el fichero original y crear un fichero con el mismo nombre y volcar el contenido del pipe a dicho fichero. (Usando pipe).
¿Alguna de estas alternativas no funcionaría correctamente? Indique para cada alternativa sus ventajas e inconvenientes.

c) Implementar la función cifrar_fichero.

d) Si reconsideramos el apartado b), planteándonos la posibilidad de que el proceso falle bajo las siguientes condiciones:

d.i) Que el proceso de cifrado y/o el que recorre el directorio se caiga en cualquier punto de la ejecución.
d.ii) Que el espacio en disco sea muy escaso y pueda agotarse, sobretodo al cifrar ficheros de gran tamaño.
Bajo estos escenarios, comente las alternativas 1, 2 y 3, del apartado b) bajo estas dos posibles causas de error ¿Cómo calificaría la fragilidad de cada una de las soluciones?

Problema 1.9

En el directorio `/home` *de un sistema de ficheros A se ha montado el sistema de ficheros B, sin enmascarar ningún permiso.* ***Parte del árbol*** *de nombres resultante se muestra en la figura 1.3, en base a una salida parcial de varios mandatos* `ls` *(se recuerda que el segundo dato del* `ls` *es el número de enlaces).*

Las máscaras de creación de ficheros y los grupos de algunos usuarios son los siguientes:

- *Usuario* `pepe`*, máscara = 022 y grupo div1*
- *Usuario* `macu`*, máscara = 027 y grupo div1*
- *Usuario* `mari`*, máscara = 077 y grupo div2*

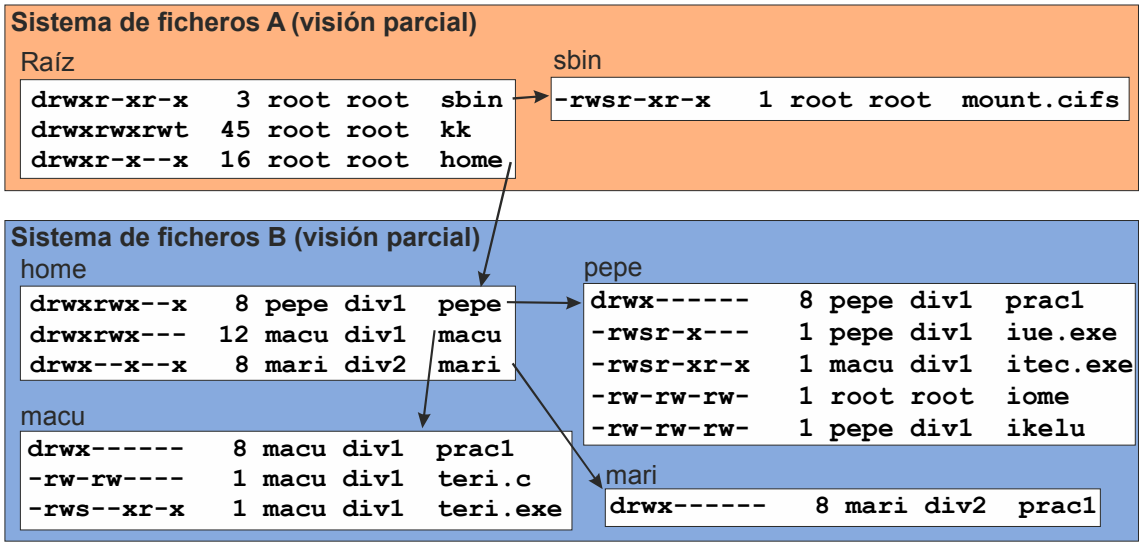

Figura 1.3

El programa `itec.exe` *contiene el siguiente esquema de código*

```
fd = open("/home/macu/teri.c", O_RDWR);
n = fork();
```

a) El usuario `mari` *crea un nuevo programa* `xx` *que incluye la siguiente línea:*

```
m = execl("/home/pepe/itec.exe", "itec.exe", NULL);
```

Teniendo en cuenta que `m` *es una variable de* `xx` *y* `fd` *de* `itec.exe`*, y que no hay errores inesperados, indicar el valor de* `m` *y el de* `fd` *cuando* `mari` *ejecuta el programa* `xx`*.*

b) Los usuarios `pepe` *y* `macu` *lanzan al tiempo la ejecución del programa* `itec.exe`*. ¿Cuál sería el máximo número de referencias (también llamados* `ndups`*) y de* `nopens` *que podrían llegar a alcanzarse?*

c) El programa `iue.exe` *contiene la siguiente línea de código:*

```
n = creat("/home/pepe/prac1/siet.c", 0666);
```

El usuario `macu` *intenta ejecutar dicho programa desde su home, indicar si se crea el fichero y, en su caso, indicar el dueño y los derechos.*

Ahora, el programa que ha lanzado el usuario `pepe` *ejecuta la siguiente línea de código:*

```
fd = open("/home/pepe/ikelu", O_RDWR|O_CREAT|O_TRUNC,0640);
```

obteniendo `fd= 5`*. Seguidamente, el programa crea dos hijos H1 y H2. Inmediatamente, H2 hace un* `exec`*.*

Al mismo tiempo, pero cuando `pepe` *ya ha ejecutado el* `open`*, el usuario* `mari` *ejecuta otro programa (proceso M) que incluye:*

```
fd = open("/home/pepe/ikelu", O_RDWR);
```

recibiendo fd = 3.

A partir de este punto se produce la siguiente secuencia de ejecuciones

	Proceso	***Servicio***
1	*H1*	`write(5, "11111111111111111111111111111", 23);`
2	*H2*	`write(5, "22222222222222222222222222222", 26);`
3	*M*	`write(1, "3333", 4);`
4	*H1*	`lseek(5, 17, SEEK_END);`
5	*H2*	`write(5, "9999", 4);`
6	*M*	`lseek(3, 2, SEEK_SET);`
7	*H1*	`write(5, "4444", 4);`
8	*M*	`write(3, "45", 2);`
9	*H2*	`lseek(5, 4, SEEK_SET);`
10	*H1*	`write(5, "8901", 4);`
11	*H1*	`close(5);`
12	*H2*	`close(5);`
13	*M*	`Close(3);`

Aplicando la política de coutilización de UNIX:

d) Indicar el tamaño real del fichero al final de esta secuencia.

e) Indicar el contenido del fichero.

Problema 1.10

Sea un sistema de ficheros UNIX con las siguientes características:

- *Bloque de 1 KiB y agrupación de 1 bloque.*
- *Nodos i ocupados del 2 al 29, el resto libres.*
- *Bloques ocupados del 1 al 79, el resto libres.*
- *Para asignar un bloque o un nodo i se selecciona el primero libre.*
- *Entradas libres de la tabla intermedia: a partir de la 11.*
- *El directorio /"(raíz), pertenece al usuario y grupo root y tiene los permisos: rwx rwx --x*
- *El directorio /"home,"pertenece al usuario y grupo root, tiene los permisos: rwx rwx rwx y su nodo i es el 14.*
- *El fichero /"home/user2/prog"tiene: UID = 400, GID = 7 y permisos = 06755.*
- *Enteros de 4 bytes.*

En el instante inicial existen dos procesos PA y PB cuyas identidades y descriptores de fichero se indican en la figura 1.4.

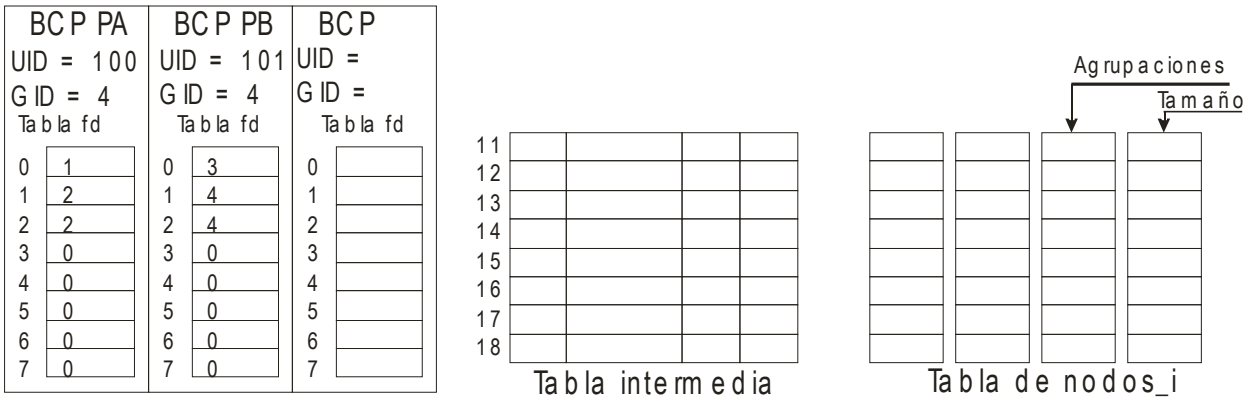

Figura 1.4

Se ejecuta la secuencia siguiente:

```
PA      umask(0023);
        mkdir("/home/user1", 0753);   // El directorio no existe inicialmente.
        fd=creat("/home/user1/file1.txt", 0666);
        for (i=0;i<100;i++)
           n=write(fd, "0123456789ABCDEFGHJK", 20);
        pid=fork();     // Suponer que no falla. Llamar PC al hijo.
PC      lseek(fd, 3, SEEK_SET);
        n=write(fd, "xter",4);
PB      fd2=open("/home/user1/file1.txt", O_RDWR);
```

Punto 1 de la secuencia de ejecución.

```
PB      lseek(fd2, 5*1024, SEEK_CUR);
        n=write(fd2, "ProcesoB",8);
```

Punto 2 de la secuencia de ejecución.

```
PB      execl("/home/user2/prog", "/home/user2/prog", NULL);
        exit (1);
PB      (nuevo código)
        fd3=creat("/home/user1/file2.txt", 0666);   // El fichero no existe inicialmente.
        n3=access("/home/user1", R_OK);
```

Punto 3 de la secuencia de ejecución.

Se pide:

a) Determinar los permisos con los que se crea el fichero "home/user1/file1.txt."

b) Completar la figura 1.4 para el punto 1 de la secuencia de ejecución. Deberá indicar los títulos de las columnas de la tabla intermedia y de nodos i que faltan.

c) Indicar el contenido de los bloques que se modifican por la secuencia anterior en su ejecución hasta el punto 1.Usar para los bloques de directorio el siguiente formato: nombre – nodo i; nombre – nodo i; ... Limitar el contenido presentado a los primeros caracteres de cada bloque seguido de algunos puntos, si éste tiene un contenido mayor. Indicar el comienzo de la basura poniendo interrogaciones ???.

d) En el punto 2 de la secuencia de ejecución indicar las agrupaciones asignadas al fichero "home/user1/file1.txt"

e) En el punto 3 de la secuencia de ejecución indicar los valores devueltos en las variables fd3 y n3.

Problema 1.11

*Sea el programa `TT.c`, al que le faltan las líneas 21 a 51 y al que se le han tachado ciertos textos con la notación XXX**##**XXX, siendo ## es el número de línea donde están.*

```
___1 /* TT.c */
___2 #include <fcntl.h>
___3 #include <stdio.h>
___4 #include <stdlib.h>
___5 #include <unistd.h>
___6 #define SIZE 1024
___7
___8 int main(int argc, char *argv[])
___9 {
__10     int pp[2], fd;
__11     char buff[SIZE];
__12     int ret;
__13
__14     /* Comprobar Argumentos */
__15     if (!argv[1]) {
__16         fprintf(stderr, "USO: %s mandato [args...]\n", argv[0]);
__17         exit(1);
__18     }
__19
__20     /* Almacenar Copia de Entrada Estándar */
....
....
__52
__53     /* Almacenar Copia de Salida Estándar */
__54     ret = pipe(pp);
__55     if (ret < 0) {
__56         perror("pipe");
__57         exit(1);
__58     }
__59     switch(fork()) {
__60     case -1:
__61         perror("fork");
__62         exit(1);
__63     case  0:
__64         close(XXX64XXX);
__65         fd = creat("salida", 0666);
__66         if (fd < 0) {
__67             perror("salida");
__68             exit(1);
__69         }
__70         while((ret = read(pp[0], buff, XXX70XXX)) > 0) {
__71             write(fd, buff, XXX71XXX);
__72             write(1, buff, XXX72XXX);
__73         }
__74         if (ret < 0) {
__75             perror("salida");
__76             exit(1);
__77         }
__78         return 0;
__79     default:
__80         close(XXX80XXX);
__81         dup(XXX81XXX);
__82         close(pp[0]);
__83         close(pp[1]);
```

```
__84      }
__85
__86      /* Ejecutar Mandato */
__87      execvp(argv[1], &argv[1]);
__88      perror(argv[1]);
__89      exit(1);
__90
__91      return 0;
__92  }
```

Dado el código presentado (sin considerar las líneas 21 a 51 que faltan) se pide que deduzca cuál es la utilidad del programa. Para ello:

a) *Suponga que el fichero* `dias.txt` *contiene, uno por línea, los nombres de los días de la semana. Estudie el siguiente ejemplo de uso del mandato* `TT`*.*

```
$ ./TT sort < dias.txt
```

Identifique claramente qué información llegará por la entrada estándar, qué información producirá por la salida estándar y cuál será la información finalmente contenida en los ficheros auxiliares que se hayan creado.

b) *Dibuje un diagrama de procesos comunicados, donde quede clara la jerarquía de los procesos, los descriptores que utilizan y para qué los utilizan.*

c) *Explique con sus propias palabras para qué sirve el mandato* `TT`*.*

d) *Razone y explique cuál sería el valor correcto del texto tachado en las líneas 70 a 72.*

e) *Razone y explique cuál sería el valor correcto del texto tachado en las líneas 80 y 81.*

f) *Razone y explique cuál sería el valor correcto del texto tachado en la línea 64. ¿Qué le sucedería a la ejecución del mandato si se hubiese omitido esta llamada a* `close`*?*

Dado el código expuesto, no es difícil imaginar el código de las líneas 21 a 51 que faltan, ya que su misión es simétrica al de las líneas 54 a 84.

g) *Codifique las líneas que faltan. Preste especial atención a las diferencias necesarias y subráyelas.*

h) *En vista al nuevo código que ha escrito, ¿sería posible eliminar las líneas 54 a 58?, es decir, ¿serían necesarias una o dos llamadas a* `pipe`*?, y en este último caso, ¿bastaría con la declaración de* `pp` *en la línea 10 o haría falta una segunda?*

i) *Conteste nuevamente al apartado* ***a)****.*

j) *Conteste nuevamente al apartado* ***b)****.*

Problema 1.12

Debe usted implementar en lenguaje C y para UNIX el código fundamental del mandato `cyp` *(CopiaYPega) cuya utilidad es poder copiar una parte de un fichero, dadas su posición y longitud (de bytes a gigabytes), a otro fichero en otra posición dada. Realice el código más correcto y claro que pueda (NO se admitirá pseudocódigo), realice un tratamiento correcto de los errores y libere todos los recursos previamente reservados, cuando dejen de usarse.*

a) Siga estrictamente las siguientes instrucciones de diseño:

- *Implemente la función* `copia` *que copia* `size` *bytes del fichero abierto* `fdorg` *al fichero abierto* `fddst`*:*

```
copia(fdorg, size, fddst)
```

- *Haciendo uso de la función anterior, implemente la función* `copiaypega`*, que añade a la anterior la*

posición absoluta de origen y destino, `offorg` *y* `offdst`*:*

```
copiaypega(fdorg, offorg, size, fddst, offdst)
```

- *Haciendo uso de la función anterior, implemente la función* `cyp` *que realiza la operación de* `copiaypega` *entre los ficheros* `origen` *y* `destino` *de nombre dado:*

```
cyp(origen, offorg, size, destino, offdst)
```

b) Considere la siguiente secuencia de mandatos y analice el comportamiento de `cyp`*:*

```
(1)  echo –n 0123456789 > A
(2)  echo –n abcdefghij > B
(3)  cyp A 7 5 B 2
(4)  cyp A 1 5 B 8
```

Razone cuál sería el contenido del fichero `B` *al final de los pasos (3) y (4) y porqué.*

c) Ahora, piense y rehaga sólo la parte imprescindible del código anterior, para realizar una implementación equivalente de `cyp` *pero mediante ficheros proyectados en memoria. Observe que conseguir el mismo comportamiento anterior implica una solución un poco más completa que la trivial.*

d) Razone:

- *¿Cuál de estas implementaciones será más rápida (en bytes copiados por unidad de tiempo)?*
- *¿Cuál de estas implementaciones será más apta para copiar cantidades muy grandes de datos (decenas de gigas) (considere un sistema de 32 bits)?*

e) Considere ahora la necesidad de poder ejecutar varios mandatos `cyp` *concurrentes haciendo copias cruzadas de información entre un conjunto de ficheros.*

- *Implemente el código necesario para asegurar que cada cyp sucede sin verse afectado por otros en ejecución simultánea.*

¿Podría suceder interbloqueo?, ¿cómo podría evitarse?

Problema 1.13

Sea un programa que ejecuta sobre una máquina UNIX y que realiza las siguientes operaciones sobre un determinado fichero:

```
int fd;
...
fd = creat( "shared", 0755 );
...
fork();
...
fork();...
/* Padre */
     write( fd, "Soy P\n", 6 );
     write( fd, "Adiós\n", 6 );
     exit(1);
...
/* Hijo 1 */
     write( fd, "Soy H1\n", 7 );
     lseek( fdm 1024, SEEK_CUR ); /* Desde posición actual */
     exit(1);
...
/* Hijo 2 */
     write( fd, "Soy H2\n", 7 );
     exit(1);
```

Conteste a las siguientes preguntas:

*1.- ¿Cuál sería el tamaño final del fichero? ¿Sería distinto si el **lseek** se realizara en último lugar?*

2.- ¿Cuál sería el contenido final del fichero (examine las diferentes posibilidades)?

*3.- Como habrá observado, en este ejemplo el **lseek** puede plantear un problema de confidencialidad de información, ¿cuál es?, ¿cómo se resuelve?*

4.- ¿Qué estructuras de datos permiten relacionar los descriptores de fichero de los procesos con los inodos? Dibuje claramente estas estructuras reflejando los tres procesos y el fichero del ejemplo.

5.- Suponga que en el esquema anterior el proceso Padre quiere que sus dos operaciones sobre el fichero se realicen de forma atómica. ¿Cómo lo puede conseguir?

Problema 1.14

Dado un disco de 4 GiBytes con tamaño de bloque de 1 KiB se quieren analizar los dos siguientes sistemas de ficheros:

1.- Sistema de ficheros tipo UNIX con las siguientes características:

Representación del fichero mediante nodos-i con 10 direcciones directas a bloque, un indirecto simple, un indirecto doble y un indirecto triple y direcciones de bloque de 4 bytes. El sistema utiliza un mapa de bits para la gestión del espacio vacío.

2.- Sistema de ficheros tipo MS-DOS (FAT) con las siguientes características:

Entradas de 4 bytes y tamaño de agrupaciones de 4 bloques.

Se pide:

a) ¿Cuál es el tamaño máximo de los ficheros en cada sistema de ficheros?

b) ¿Qué tamaño ocupan la FAT y el mapa de bits en cada caso?

c) Se desea abrir un fichero llamado datos.txt que se encuentra en el directorio user y acceder a los bytes 273.780.000 y 281.450.500. Si nos encontramos en el directorio raíz, ¿cuál será el número de accesos al disco para realizar la anterior operación en cada sistema de ficheros?

d) ¿Dónde se almacenan los atributos del fichero en cada sistema de ficheros?. ¿Qué problemas puede presentar este sistema de atributos en MS-DOS?