Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 1

Sea un sistema de archivos similar al de UNIX con un tamaño de bloque de 4KB y un nodo-i con 10 punteros directos, 1 indirecto simple, 1 doble y 1 triple.
Sin embargo, a diferencia del sistema de archivos de UNIX, este sistema usa *write-through* en todas las operaciones de escritura en el disco, tanto para la metainformación como para los propios datos.
Se pretende analizar en este sistema qué zonas de una partición son actualizadas por las distintas llamadas al sistema.
Para ello se considerarán las siguientes zonas:

a. Mapa de bloques libres (MB).
b. Mapa de nodos-i libres (MN).
c. Bloques con nodos-i (BN).
d. Bloques de datos (BD).

Dado el siguiente fragmento de programa:

```
a) mkdir("/dir", 0755);              /* llamada 1 */
b) fd=creat("/dir/fl", 0666);        /* llamada 2 */
c) write(fd, buf, 4096);             /* llamada 3 */
d) lseek(fd, 40960, SEEK_SET);       /* llamada 4 */
e) write(fd, buf, 4096);             /* llamada 5 */
f) close(fd);                        /* llamada 6 */
g) symlink("/dir/fl","/dir/f2");     /* llamada 7 */
h) unlink("/dir/fl"};                /* llamada 8 */
```

Suponiendo que no se produce ningún error en la ejecución de dicho fragmento, se pide especificar qué zonas son actualizadas en cada llamada explicando razonadamente el motivo de dicha actualización.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Solución

a) Llamada **mkdir**:

Suponiendo que hay espacio suficiente en el directorio raíz para incluir esta nueva entrada, esta llamada implica las siguientes operaciones:

1. Reservar un nodo-i para el nuevo directorio (actualizar **MN** para marcar un nodo-i libre como ocupado).
2. Reservar un bloque de datos (actualizar **MB** para marcar un bloque libre como ocupado) para que contenga las entradas del nuevo directorio (inicialmente "." y ". . ").
3. Escribir sobre el nuevo nodo-i reservado (**BN**) toda la información del directorio creado (uid, gid, permisos, tamaño, dirección del primer bloque, etc.).
4. Escribir en el bloque de datos reservado (**BD**) las entradas iniciales del nuevo directorio (" . " y " . . ")
5. Escribir en el bloque de datos del directorio raíz (**BD**) la nueva entrada (dir + número del nuevo nodo-i reservado).
6. Actualizar el nodo-i del directorio raíz (**BN**), puesto que después de incluir la nueva entrada se ha modificado parte de la información de este nodo-i (por ejemplo, la fecha de última modificación).

En el caso de que no hubiese espacio en el directorio raíz, habría que asignarle un nuevo bloque al directorio. Dependiendo de qué tipo de puntero se viera afectado por la expansión, se presentarían distintas posibilidades:

- Si se trata de un puntero directo, únicamente habría que reservar un nuevo bloque para almacenar la nueva entrada (MB). La escritura correspondiente a la quinta operación anteriormente especificada se realizará sobre este nuevo bloque. Asimismo, durante la actualización del nodo-i raíz realizada en el paso sexto, se incluirá en dicho nodo-i una referencia al nuevo bloque. Nótese que, en caso de necesitarse la expansión, lo más probable es que se trate de este caso, o sea, que sólo se vean afectados los punteros directos, ya que normalmente el tamaño de un directorio no será mayor de 40 KB.
- Si se ven implicados los punteros indirectos, sería necesario reservar, además del bloque de datos, los bloques de índices necesarios (MB) dependiendo del nivel de indirección alcanzado y escribir en esos bloques la información de direccionamiento correspondiente (BD). En la quinta llamada planteada en el problema se analizará un caso similar a éste.

b) Llamada **creat**:

En este caso estamos seguros de que hay espacio en el directorio dir ya que se acaba de crear. Por lo tanto, esta llamada implica las siguientes operaciones:

7. Reservar un nodo-i para el nuevo archivo (**MN**).
8. Escribir sobre el nuevo nodo-i reservado (**BN**) toda la información del archivo creado (uid, gid, permisos, tamaño, etc.).
9. Escribir en el bloque de datos del directorio dir (BD) la nueva entrada (f1 + número del nuevo nodo-i reservado).
10. Actualizar el nodo-i del directorio dir (**BN**), puesto que después de incluir la nueva entrada se ha modificado parte de la información de este nodo-i (por ejemplo, la fecha de última modificación).

c) Llamada **write**:

Esta llamada implica las siguientes operaciones:

1. Como el archivo está inicialmente vacío, hay que reservar espacio para satisfacer esta petición. Dado que los datos a escribir ocupan 4 KB, sólo es necesario reservar un bloque (MB) para copiar en él los datos.
2. Escribir los datos en el bloque reservado (**BD**).
3. Actualizar el nodo-i del archivo (**BN**) para reflejar los cambios producidos en el mismo (tamaño, dirección del primer bloque, fecha de última modificación, etc.).

d) Llamada **lseek**:

Esta llamada no produce la actualización de ninguna de las zonas del disco, ya que lo único que hace es fijar un nuevo valor para el puntero de posición del archivo (lo coloca en el byte 40960) y esta información no está almacenada en disco sino en la tabla de archivos intermedia que reside en memoria. Nótese que el archivo no sufre ningún cambio debido a esta llamada. Solamente se verá afectado cuando realmente se produzca una escritura sobre el mismo.

e) Llamada **write**:

Como el puntero de posición del archivo está colocado en el byte 40.960, esta escritura afecta al undécimo bloque del archivo. Dado que sólo existen 10 punteros directos, se verá implicado el puntero indirecto simple. Nótese que en UNIX no se asigna espacio a los huecos intermedios que pueden quedar en un archivo (en este caso desde el segundo hasta el noveno). Una lectura de uno de estos bloques devolvería ceros. La asignación de espacio se posterga hasta que se produzca una escritura sobre el bloque. Por lo tanto, esta llamada implicaría las siguientes operaciones:

1. Hay que reservar espacio para satisfacer esta petición. Dado que los datos a escribir ocupan 4 KB y se ve implicado por primera vez el puntero indirecto simple, será necesario reservar dos bloques (**MB**): uno que corresponderá con el bloque de índices y otro para copiar en él los datos.
2. Escribir en la primera posición del bloque de índices la dirección del bloque reservado para almacenar los datos (**BD**).
3. Escribir los datos en el bloque reservado (**BD**).
4. Actualizar el nodo-i del archivo (**BN**) para reflejar los cambios producidos en el mismo (tamaño, dirección del bloque indirecto simple, fecha de última modificación, etc.).

f) Llamada **close**:

Esta llamada no produce la actualización de ninguna de las zonas del disco ya que, como se usa write-through, la copia del nodo-i en disco estará actualizada. Sólo, en el caso de que no haya ninguna apertura del archivo activa, se liberará la copia en memoria del nodo-i.

g) Llamada **symlink**:

Un enlace simbólico es un tipo de archivo especial que almacena como datos la ruta de acceso para llegar a otro archivo (que puede no existir). Nótese que no implica ninguna actualización (ni siquiera consulta) de la información relacionada con el archivo f1. Por lo tanto, el número de enlaces asociado al archivo f1 sigue valiendo 1 (el enlace original). Esta llamada implicaría las siguientes actualizaciones:

1. Reservar un nodo-i para el nuevo archivo (MN).
2. Reservar un bloque de datos (MB) para almacenar en él la ruta dir/f 1.
3. Escribir dicha ruta en el bloque de datos reservado (BD).
4. Escribir sobre el nuevo nodo-i reservado (BN) toda la información del archivo creado (tipo enlace simbólico, uid, gid, permisos, dirección del primer bloque, tamaño, etc.).
5. Escribir en el bloque de datos del directorio dir (BD) la nueva entrada (f2 + número del nuevo nodo-i reservado).
1. Actualizar el nodo-i del directorio dir (**BN**), puesto que después de incluir la nueva entrada se ha modificado parte de la información de este nodo-i (por ejemplo, la fecha de última modificación).

Grupo: …… NIA: …… Nombre y apellidos: ……..

h) Llamada **unlink**:

Esta llamada decrementa el número de enlaces del archivo f1 y, si el valor llega a 0, libera el archivo. Si suponemos que ningún otro programa ha creado un enlace físico al archivo, el contador llegará a 0 produciéndose el borrado del archivo que implicaría las siguientes operaciones:

6. Escribir en el bloque de datos del directorio dir (BD) para eliminar la entrada correspondiente a f1.
7. Actualizar el nodo-i del directorio dir (BN), puesto que después de eliminar la entrada se ha modificado parte de la información de este nodo-i (por ejemplo, la fecha de última modificación).
8. Liberar el nodo-i de f1 (MN).
9. Liberar los tres bloques del archivo (MB): dos bloques de datos y el indirecto simple.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 2

Se pretende imprimir el tamaño de de un fichero

```
printf("El tamaño de %s es %d\n", argv[l], size);
```

¿Cuál de los siguientes códigos es la más eficaz? ¿Por qué?

a)
```
fd = open(argv[l] , O_RDONLY)
size = 0
while ((leidos = read(fd, buf, BUF_SIZE) ) > 0)
      size = size + leidos;
close (fd);
```

b)
```
fd = open(argv[l], O_RDONLY)
size = lseek(fd, O, SEEK__END)
close (fd)
```

c)
```
stat(argv[l], &attr)
size = attr.st_size
```

d)
```
fd = open(argv[l], O_RDONLY)
fstat(fd, &attr)
close (fd)
size = attr.st_size
```

Solución

La más ineficaz es claramente la primera, ya que requiere leer el contenido de todo el archivo. De las otras tres, la segunda y la cuarta implican una apertura del archivo. Esta apertura requiere:

1. Traer el nodo-i del archivo a memoria (a la tabla de nodos-i).
2. Reservar un nuevo descriptor en la tabla de descriptores de archivos del proceso.
3. Reservar una entrada en la tabla de archivos intermedia.
4. Enlazar todas las entradas anteriores.

La tercera forma, que usa la llamada stat (), no requiere todas las operaciones anteriores, solo acceder al nodo-i del archivo. Por tanto, es la más eficaz.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 3

Se dispone de un dispositivo de almacenamiento secundario de datos industrial con las siguientes características:

- No volátil.
- Dispositivo de acceso directo con bloques de 1 Kbyte.
- Permite una única escritura (sin sesiones y no se puede borrar lo escrito).
- Se puede leer el contenido infinitas veces.
- Tiempo de búsqueda de bloques muy alto y proporcional a la distancia al último bloque accedido.
- Tiempo de transferencia de datos bajo.

Se pretende construir un sistema de fichero para el sistema operativo que utilice este dispositivo. Por tanto, se pide:

a) Construir el esquema del volumen del sistema de ficheros para el dispositivo anterior (indicando sus componentes y los datos que los componen) que cumpla lo siguiente:
   - Debe incluir archivos y directorios.
   - Debe permitir enlaces duros y simbólicos.
   - Debe optimizar el rendimiento del dispositivo (reducir al máximo el tiempo de búsqueda y de transferencia en todas las operaciones) y el espacio de almacenamiento desperdiciado (fragmentación interna y externa).

b) Indique cual es, en su opinión, el tamaño de agrupación de bloques óptimo para el sistema de ficheros diseñado en el apartado a).

c) Implemente en pseudocódigo la operación `open` para el sistema de ficheros diseñado en el apartado a).

SOLUCIONES

a) El sistema propuesto es muy similar a un CD-ROM, así que el diseño propuesto se asemejara mucho al utilizado en los CD-ROM (no igual por que los CD-ROM reales son algo más complejos y arcaicos).

Por facilidad partiremos de un sistema de ficheros UNIX típico modificándolo donde se considere oportuno:

- *Como solo se escribe una vez y no se puede borrar y además el tiempo de búsqueda es alto y proporcional a la distancia lo ideal es utilizar asignación contigua para almacenar los ficheros (todos los bloques del fichero almacenados en secuencia del primero al último).*
- *Para mejorar el tiempo de búsqueda situaremos los inodos cerca de los bloques de datos, 2 posibilidades:*
  - *Poner los datos del fichero a continuación de su inodo. (Está es la solución más parecida a lo que hacen los CD-ROMS).*

- *Agrupar un conjunto de ficheros en un bloque más grande como lo hace el S.F. fast file system (estos bloques tendrán un grupo de inodos seguidos y a continuación sus datos)*

Componentes necesarios:

- *Sector de arranque y superbloque: pueden ser dos bloques o solo uno. Contiene datos relativos al sistema de ficheros como número de inodos, inodo del directorio raíz, etc.*
- *Inodos: contiene los atributos del fichero salvo el nombre. Incluye tamaño, fechas del fichero, etc. Incluye dirección del bloque inicial de datos y del final (este último es opcional). Para enlaces duros no hace falta contador de enlaces porque no se puede borrar los ficheros. Hace falta un atributo para indicar si es enlace simbólico, directorio o archivo.*
- *Bloques de datos.*

NOTA: No se incluye mapa de bits porque no tiene sentido, (solo se escribe una vez, y si se hicieran sesiones como en los CDROM basta con buscar donde termina el último fichero).

La disposición propuesta de los elementos es la siguiente:

Sector de Arranque	*Superbloque*	*Inodo A*	*Datos A*	*Datos A*	*Inodo B*	*Datos B*	*Datos B*	*...*

b) Agrupación: Numero de bloques con la misma dirección dentro del sistema de ficheros. Suele ser un número de bloques potencia de 2.

Ventajas de una agrupación grande:

- *En sistemas donde los ficheros no tienen los bloques contiguos y utilizan tablas con bloques indirectos para localizar sus datos, una agrupación grande reduce el tamaño de la tabla y el número de bloques directos,*
- *Además en el caso anterior, si hay tiempo de búsqueda grande aumenta el rendimiento al buscar menos bloques de datos e indirectos.*
- *Por si fuera poco en el caso anterior si el tiempo de búsqueda es proporcional a la distancia una agrupación grande aumenta la secuencialidad del fichero y mejora el rendimiento.*

Ventajas de una agrupación pequeña:

- *Disminuye la fragmentación interna (se pierde media agrupación por fichero).*

Como el sistema propuesto tiene ficheros contiguos ninguna de las ventajas de una agrupación grande se puede aplicar, luego interesa la agrupación más pequeña posible (1 bloque = 1 KB).

c) *Operación OPEN*
Entrada: RUTA (del fichero), permisos.
Salida: nº descriptor de fichero.
Algoritmo:

- *Obtener del superbloque el nº del inodo raíz (en INODO).*
- *Leer INODO.*
- *Mientras que INODO sea directorio y RUTA no vació*
  - *Quitar SiguienteDirectorio de RUTA*
  - *Buscar SiguienteDirectorio en datos de INODO.*
  - *Obtener nº de inodo de SiguienteDirectorio.*
  - *Leer INODO.*
- *Si INODO no existe.*
  - *Si primera vez -> crear fichero:*
    - *Crear inodo de fichero(los datos se copiaran después) (después ejecutara la rama de inodo es fichero)*
  - *Si sucesiva vez -> ERROR sistema de solo lectura.*
- *Si INODO es fichero*
  - *Comprobar permisos.*
  - *Guardar INODO en tabla de inodos del SO.*
  - *Crear descriptor en tabla de descriptores del proceso y enlazar descriptor con inodo.*
  - *Crear entrada en tabla intermedia (donde se guarda la posición relectura del fichero)*
  - *Devolver nº de descriptor*

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 4

Se dispone de un dispositivo de almacenamiento secundario de datos basado en chips de memoria con las siguientes características:

- No volátil.
- Dispositivo de acceso directo con bloques de 1 Kbyte.
- **Permite lectura y escritura**.
- **Tiene una alta tasa de bloques defectuosos que aumenta con el tiempo.**
- **Tiempo de búsqueda de bloques fijo e independiente de la posición del bloque.**
- Tiempo de transferencia de datos bajo.

Se pretende construir un sistema de fichero para el sistema operativo que utilice este dispositivo. Por tanto, se pide:

a) Construir el correspondiente esquema del volumen del sistema de ficheros para el dispositivo anterior (indicando sus componentes y los datos que los componen) para cada uno de los siguientes casos:

   1. Si no importa que se pierdan algunos ficheros (debido a los fallos), pero debe mantenerse el sistema de ficheros coherente el mayor tiempo posible.
   2. Si hay que evitar que se pierda cualquier dato o fichero durante el mayor tiempo posible.

   NOTA: En ambos casos hay que intentar optimizar el rendimiento del dispositivo y el espacio de almacenamiento desperdiciado.

b) Indique cual es, en su opinión, el tamaño de agrupación de bloques óptimo para los sistema de ficheros diseñados en el apartado a).

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIONES

a) Construir el esquema del volumen del sistema de ficheros para el dispositivo anterior (indicando sus componentes y los datos que los componen) que cumpla lo siguiente:

1 Si no importa que se pierdan algunos ficheros (debido a los fallos), pero debe mantenerse el sistema de ficheros coherente el mayor tiempo posible.

Una posibilidad es utilizar un esquema basado en grupos de bloques como el *Fast File System*.

Componentes necesarios del volumen:

- Sector de arranque: solo puede haber uno (la ROM solo reconoce uno). Solo es necesario si se quiere usar como disco de sistema.
- Grupos de bloques de tamaño fijo. Cada grupo contiene
  - Copia del superbloque: Contiene datos relativos al sistema de ficheros como número de inodos, inodo del directorio raíz, etc.
  - Inodos de los ficheros del grupo: contiene los atributos del fichero salvo el nombre. Incluye tamaño, fechas del fichero, etc. Incluye tabla de bloques del fichero (siguiendo el mismo esquema de UNIX). Para enlaces duros hace falta contador de enlaces Hace falta un atributo para indicar si es enlace simbólico, directorio o archivo.
  - Mapa de bits del grupo.
  - Bloques de datos del grupo.

Mapa del volumen

Sector de Arranque	Grupo de bloques 1	Grupo de bloques 2	Grupo de bloques 3	...

Esquema de un grupo de bloques

Superbloque	Mapa de bits	Inodos	Datos	...

Grupo: …… NIA: …… Nombre y apellidos: ……..

Las ventajas son:

- Repartir la información crítica por el volumen para evitar fallos de una zona concreta.
- Tener copias de seguridad de la información imprescindible (superbloque).

2 Si no debe perderse ningún fichero.
La mejor solución es dividir el dispositivo en dos volúmenes iguales y hacer una copia de todo.

Mapa del volumen

Sector de Arranque	Super bloque	Mapa de bits	Inodos	Datos	Sector de Arranque	Super bloque	Mapa de bits	Inodos	Datos

Hay dos formas de gestionar esta solución:

i. Modificando el sistema de fichero para que cada vez que modifique una estructura en el F.S. lo haga por duplicada.
ii. Modificando el driver del dispositivo para que cada bloque que modifique en un volumen también lo modifique en el otro. (Esta solución es conocida como RAID 1 y es mejor que la anterior).

3 El tiempo de acceso no depende de la posición luego tener bloques pequeños no tiene coste par el rendimiento del sistema. Así se optimiza mejor el espacio de almacenamiento (menos fragmentación)

Luego la mejor solución es el bloque más pequeño posible. Es decir, el bloque del sistema de ficheros debe ser igual al bloque del dispositivo (1 KB)

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 5

Se ha decidido la compra un disco duro de 100 MiB para tareas de alto rendimiento.

El administrador de sistemas de la empresa ha decidido dar una serie de parámetros en la configuración del sistema de ficheros tipo UNIX (tamaño del volumen, tamaño de bloque y número de i-nodos) con el objetivo de maximizar el rendimiento del sistema en lo que se refiere a número de accesos a disco.

Las características finales que se aplicaran a la hora de realizar la creación del sistema de ficheros mediante el comando mkfs son:

- Tamaño de bloque: 1024 bytes.
- Tamaño de la dirección de los bloques: 4 bytes.
- Numero de i-nodos : 50
- Un bloque de carga (Boot) que ocupa 2 bloques.
- Un Superbloque que ocupa 8 Kbytes y se usa un Mapa de Bits para indicar que bloques están usados y cuales libres.
- Campos de un i-nodo:
    - Atributos del fichero:
        - Id. del Propietario y del grupo.
        - Permisos de lectura para el dueño, grupo y resto del mundo.
        - Permisos de escritura para el dueño, grupo y resto del mundo.
        - Permisos de ejecución para el dueño, grupo y resto del mundo.
    - Contador de enlaces (1 byte).
    - 4 punteros indirectos.
    - 4 punteros indirectos simples.
    - 4 punteros indirectos dobles.

Responder a las siguientes preguntas:

a) ¿Qué tamaño máximo podrá tener un fichero almacenado en este disco? Razona la respuesta.

b) Si el objetivo es minimizar el acceso a bloques de disco ¿Qué parámetro crees que debería incrementar/decrementar el administrador para conseguir dicho propósito?

c) Suponiendo que el sistema de ficheros sea capaz de utilizar enlaces duros y simbólicos.
Calcular:

- Número máximo de enlaces duros (además del nombre inicial) que se pueden realizar de un fichero en este disco.
- Número máximo de enlaces simbólicos que se pueden realizar de un fichero.

Solución

El tamaño máximo de un fichero depende de las referencias del inodo y del Nº de direcciones de bloque que caben en un bloque de datos (referencia indirecta) que es igual al tamaño del bloque (1024 bytes) entre el tamaño de la dirección de bloque (4 bytes), así:

REFERENCIAS	Nº BLOQUES
4 bloques directos	4
4 referencia indirecta simple:	4*1024 / 4 = 1024
4 referencia indirecta doble:	4*(1024 /4)*(1024 / 4) = 262144

Por tanto el Nº de bloques es: 4 + 1.024 + 262.144 = 263.172 bloques;

Así, el tamaño máximo (en bytes) de un fichero en este sistema de ficheros es:

263172 * 1024 = 269488128 bytes =>263.172 KB (aproximadamente 263 MiB)

Dado que el disco es de 100 MB, un fichero de tamaño máximo no cabría; por lo tanto el tamaño de un fichero viene limitado por la capacidad del disco. Para este caso, el tamaño máximo del disco deberá de calcularse de la siguiente forma:

Tamaño máximo de un fichero = (Tamaño del disco / Tamaño del bloque) – Numero de bloques ocupados por el resto de estructuras.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Las estructuras típicas que nos define el enunciado son:

Boot	SuperBloque	Mapas de Bits	Nodos-I	Datos y Directorios

Por lo tanto:

Tamaño Boot = 2 bloques

Tamaño Superbloque = 8 bloques

Tamaño Mapa de Bits = $(100 * 2^{20}) / 2^{10}$ = 102.400 bits => 13 bloques (12.5 por exceso)

Tamaño Nodos-i = 50 bloques

Tamaño máximo del fichero = $(100* 2^{20} / 2^{10})$ - (2+8+13+50) = 102327 bloques

Tamaño máximo en bytes = 102.240 * 1 Kbytes= 102240 KBytes (~ 99,84 MiB)

b) En general, el parámetro que se usa para conseguir menor número de accesos a disco es el tamaño de bloque. La transferencia de un fichero requiere buscar cada bloque que lo forma en disco, esperar el tiempo de latencia y hacer la transferencia de datos.

La contrapartida al tamaño grande de bloque es la cantidad de fragmentación interna es elevada y eso provoca que el porcentaje de uso de disco decrezca.

c) El número de enlace duros solo depende del tamaño del contador de enlaces. que indica el número de enlaces duros realizados, el contador de enlaces de un inodo de este sistema de ficheros dispone de 1 byte con lo cual puede haber desde 0 hasta 255 enlaces duros. Dado que el nombre inicial también es un enlace duro, el Nº de posibles enlaces duros, además del nombre inicial, será de 254 enlaces.

El número de enlaces simbólicos solo depende del número de ficheros que se puedan crear. Dado que el disco solo dispone de 50 inodos, el número máximo de ficheros será de 50. Como mínimo en el disco existen dos inodos ocupados (el del directorio raíz y el del fichero "original"). Por lo tanto el número máximo de enlaces simbólicos a un fichero puede ser 48.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 6

Se tiene un sistema de ficheros tipo Unix con la siguiente información:

- Tabla de I-nodos:

Nº Inodo	1	2	3		
Tipo	Directorio	Directorio	Fichero		
Contador Enlaces Fis.	3	2	1		
Direcc. Bloque Datos	11	12	13		
…….					

- Bloques de datos:

Nº Bloque	11	12	13		
Contenido	**. 1**	**. 2**	Datos del fichero		
	.. 1	**.. 1**			
	d 2	**f1 3**			

Se pide:
Modificar las tablas anteriores para reflejar el estado después de las 2 siguientes operaciones:

ln f1 /d/f2 Enlace físico de /d/f2 al fichero f1

ln –s f1 /d/f3 Enlace simbólico de /d/f3 al fichero f1

Grupo: …… NIA: …… Nombre y apellidos: ……..

Solución

Tabla de I-nodos:

Nº Inodo	1	2	3	**4**	
Tipo	Directorio	Directorio	Fichero	**Enlace Simbólico**	
Contador Enlaces Fis.	3	2	2	**1**	
Direcc. Bloque Datos	11	12	13	**14**	
…….					

Bloques de datos:

Nº Bloque	11	12	13	**14**	
Contenido	**. 1**	**. 2**	Datos del fichero	**f1**	
	.. 1	**.. 1**			
	d 2	**f1 3**			
		f2 3			
		f3 4			

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 7

Sea el sistema de ficheros tipo Unix cuya información se representa de la siguiente forma:

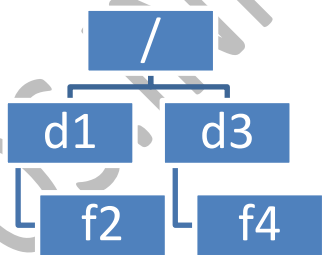
- Tabla de I-nodos:

Nº Inodo	0	1	2		
Tipo					
Contador enlaces					
Nº bloque datos					
…….					

- Bloques de datos:

Nº Bloque	0	1	2		
Contenido					

Suponiendo que en el superbloque se indica que el directorio raíz corresponde al inodo 0 y que en el sistema de ficheros sólo se tiene almacenada la siguiente estructura de ficheros y directorios (donde dX corresponde a un directorio y fX a un fichero)



Se pide:

a. Rellenar la tabla de i-nodos y la de bloques de datos para reflejar esta estructura. Se ha de rellenar todos los campos indicados de las tablas, siendo como orden de rellenado el número que acompaña al directorio/fichero (dX/fX).
b. Modificar las tablas anteriores para reflejar la ejecución de "cp /d1/f2 /d1/f5" siendo f2 un fichero cuyo contenido ocupa un solo bloque.
c. Enumere brevemente los accesos a disco realizados para la operación del apartado b en caso de estar la caché vacía y usar política *write-through* (escritura inmediata).

Grupo: …… NIA: …… Nombre y apellidos: ……..

Solución

a.

- Tabla de I-nodos:

Nº Inodo	0	1	2	3	4	
Tipo	**Dir.**	**Dir.**	**Fich.**	**Dir.**	**Fich.**	
Contador enlaces	**4**	**2**	**1**	**2**	**1**	
Nº bloque datos	**0**	**1**	**2**	**3**	**4**	

- Bloques de datos:

Nº Bloque	0	1	2	3	4	
Contenido	**. 0** **.. 0** **d1 1** **d3 3**	**. 1** **.. 0** **f2 2**	**Datos fich 2**	**. 3** **.. 0** **f4 4**	**Datos fich 4**	

b.

- Tabla de I-nodos:

Nº Inodo	0	1	2	3	4	**5**
Tipo	Dir.	Dir.	Fich.	Dir.	Fich.	**Fich.**
Contador enlaces	4	2	1	2	1	**1**
Nº bloque datos	0	1	2	3	4	**5**

- Bloques de datos:

Nº Bloque	0	1	2	3	4	5
Contenido	. 0 .. 0 d1 1 d3 3 **f5 5**	. 1 .. 0 f2 2	Datos fich 2	. 3 .. 0 f4 4	Datos fich 4	**Datos fich 5 (= datos fich 2)**

c. accesos(cp /d1/f2 /d1/f5) = 10
inodo / + datos / + inodo d1 + datos d1 + inodo f2 + datos f2 + inodo f5 + datos f5 + datos /d1 + inodo /d1

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 8

Se dispone de un sistema de ficheros de las siguientes características:

- I-nodos con la siguiente información:
  - Tipo: Fichero, directorio o enlace simbólico
  - Contador de enlaces.
  - Nº de bloque de datos directo.
  - Nº de bloque indirecto simple.
- Cuenta con 5 i-nodos disponibles
- Cuenta con 6 bloques de datos disponibles
- Tiene un mapa de i-nodos y un mapa de bloques libres.
- El tamaño de bloque es de 8KB

a. Rellene las siguientes tablas para el caso de un sistema de ficheros vacío que se haya añadido lo siguiente:
   - Un fichero en el directorio raíz llamado "f1" de 10KB de tamaño.

Mapas de i-nodos y bloques:

I-nodos:

Nº I-nodo	0	1	2	3
Tipo				
Contador enlaces				
Nº bloque datos enlace directo				
Nº bloque datos enlace indirecto simple				

Bloques de datos:

Nº Bloque	0	1	2	3	4	5
Contenido						

b. ¿Cuál sería el tamaño máximo de un fichero en este sistema de ficheros definido? Razone la respuesta.

c. Cómo cree que afectaría cambiar el tamaño de bloque al tamaño máximo de un fichero en dicho sistema de ficheros. Suponga que cuenta con 48KB para los bloques de datos. Razone la respuesta.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Solución

a.

Mapas de i-nodos y bloques:

1	1	0	0	0	1	1	1	1	0	0

I-nodos:

Nº I-nodo	0	1	2	3
Tipo	Directorio	Fichero		
Contador enlaces	2	1		
Nº bloque datos enlace directo	0	1		
Nº bloque datos enlace indirecto simple	--	2		

Bloques de datos:

Nº Bloque	0	1	2	3	4	5
Contenido	. 0 .. 0 f1 1	8KB primeros de datos de f1	3 (enlaces a bloques de datos)	2KB restantes de datos de f1		

b.

El tamaño máximo de un fichero en dicho sistema de ficheros vendría limitado por el número de bloques de datos que se tienen. En este caso se cuenta **únicamente con 6 bloques de datos**.

Visto en el ejemplo anterior que se deberá disponer de **1 bloque de datos para las entradas del directorio inicial** y **otro bloque de datos para el enlace indirecto simple** que se especifica en el enunciado se contará únicamente con 4 bloques de datos que pudieran contener datos de un fichero.

Al tener **4 bloques de datos disponibles** como máximo y ser cada uno **de 8KB** se llega a la conclusión que el tamaño máximo de un fichero en este sistema de ficheros será de **32KB**

c.

Si se **redujera el tamaño de bloque** en dicho sistema de ficheros se podrían tener **más bloques en los datos**, por ejemplo, si el tamaño del bloque fuera de 2KB se contarían con 24 bloques, ya que hemos visto que 2 bloques los utilizaríamos para el directorio raíz y para el enlace indirecto simple, se podrían tener ficheros mayores, en este caso de 44KB**.** Por lo que se **aumentaría el tamaño máximo de un fichero**.

Al contrario pasaría si **aumentáramos el tamaño de bloque**, por ejemplo, en el caso de bloques de 16KB sólo se contaría con 3 bloques que únicamente permitirían tener ficheros de 16KB máximo. Por lo que se **reduciría el tamaño máximo de un fichero**.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 9

Disponemos de una maquina monoprocesador y queremos implementar un sistema de ficheros para un sistema operativo UNIX con un *kernel* monolítico no expulsivo y con caché de bloques similar al que se ha ido presentando en la asignatura.

El sistema de ficheros a implementar tiene la siguiente estructura en disco:

1 bloque	**2** bloques	**n** bloques
Superbloque	Bloques con inodos	Bloques de datos
0	1	3 N

Donde hay que considerar que:

- El tamaño de bloque es de 4096 bytes (4 KiB).
- Hay 100 bloques de datos en total en el sistema de ficheros.
- La máquina es de 32 bits, los enteros son de 32 bits por tanto.
- Solo hay ficheros (que pertenecen a un directorio raíz común para todos)
- Hay como máximo 32 ficheros.
- Cada fichero tendrá **diez** bloques de datos asociados.
- Cada i-nodo ocupa 256 bytes y contiene, entre otras cosas, el nombre del fichero que será como máximo de 32 caracteres.
- El superbloque ocupará un bloque de disco y será el bloque inicial (bloque 0).
- El superbloque contendrá la gestión de espacio libre/ocupado.
  Tiene un vector de 32 caracteres donde el carácter '0' indica que el i-nodo asociado está libre y el carácter '1' indica que el i-nodo asociado está ocupado.
  De igual forma tiene un vector de 100 caracteres donde '0' indica que el bloque asociado a esa posición del vector está libre y '1' que está ocupado.
- Cada fichero tiene su puntero de lectura y escritura (no compartido), y no se podrá desmontar el sistema de ficheros si hay alguno abierto.

Se pide:

a) Complete el diseño de las estructuras de disco dadas indicando los campos que tendrá el superbloque y un i-nodo de este sistema de ficheros
b) Diseñar las estructuras en memoria que permitan trabajar con el sistema de ficheros anteriormente descrito.
c) Indicar en pseudocódigo las funciones de tratamiento de bloques: alloc, free y bmap, e i-nodos: ialloc, ifree y namei.
d) Diseñar la función de interfaz de sistema **u**mount.

Grupo: …… NIA: …… Nombre y apellidos: ……..

NOTAS:

- alloc devuelve el identificador del primer bloque libre e
- ialloc el identificador del primer i-nodo libre,
- a free se le pasa el identificador de un bloque y este quedará libre,
- a ifree se le pasa el identificador de un i-nodo y este quedará libre,
- bmap tiene dos parámetros: identificador de inodo y desplazamiento dentro del fichero y devuelve el identificador de bloque asociado, y por último
- namei se le pasa solo el nombre de fichero y devuelve el i-nodo asociado.

Solución

a) Completar el diseño de estructuras en disco:

Superbloque

- **Int Número mágico**
- **Int Tamaño de bloque, inicializado a 4096**
- **Int Número de bloques, inicializado a 100**
- **Int Tamaño del inodo, inicializado a 256**
- **Int Número de inodos, inicializado a 32**
- **Char inodos_estado[32], inicialmente todos a '0'**
- **Char bloques_estado[100], inicialmente todos a '0'**

Inodo

- **Char nombre[32]**
- **Int bloques[10]**
- **Int tamaño**

b) Diseño de estructuras en memoria:

Superbloque sb
Inodo inodos[32]
Sesion sesiones[32]

Donde Sesión es una estructura con, al menos, los siguientes campos:

- **Int Posición**
- **Int Abierto**

c) alloc, free, bmap, ialloc, ifree y namei:

```
int alloc ( void )
{
   for (int=0; i<sb.numero_de_bloques; i++)   // para cada bloque…
   {
       if (sb.bloques_estado[i] == '0') {          // si está libre
          sb.bloques_estado[i] = '1';              // se ocupa
          return i;                                // se devuelvo el identificador
       }
   }
   return -1;                                      // si no hay libre, devolver -1
}

void free ( int block_id )
{
   sb.bloques_estado[block_id] = '0';          // devolver el bloque
}

int bmap ( int inodo_id, int offset )
{
    // offset | 4096 -> ordinal del bloque donde está ese offset
    return inodos[offset / sb.tamanyo_bloque].bloques;
}

int ialloc ( void )
{
   for (int=0; i< sb.numero_de_inodos; i++)       // para cada inodo…
   {
       if (sb.inodos_estado[i] == 0) {             // si inodo libre
          sb.inodos_estado[i] = 1;                 // ocupar inodo
          memset(&(inodos[i]), 0, sizeof(inodo));  // poner valores por defecto
          return i;                                // devolver identificador
       }
   }
   return -1;                                      // si no hay inodo libre, devolver -1
}
```

```
void ifree ( int inodo_id )
{
   sb.inodos_estado[inodo_id] = 0;              // liberar i-nodo
}

int namei ( char *fname )
{
   for (int=0; i< sb.numero_de_inodos; i++)      // buscar inodo con nombre <fname>
   {
       if (! strcmp(inodos[i].nombre, fname))
            return i;                             // devolver identificador
   }
   return -1;                                     // si no está el inodo, devolver -1
}
```

d) umount:

```
int umount ( void )
{
   // asegurarse de que todos los ficheros están cerrados
   for (int=0; i< sb.numero_de_inodos; i++)
  {
       if (sesiones[i].abierto == 1)
          return -1;
   }

   // escribir bloque 0 de sbloques[0] a disco
   bwrite(DISK, 0, &(sb) );

   // escribir los i-nodos a disco
   bwrite(DISK, 1, &(inodos[0]) );
   bwrite(DISK, 2, &(inodos[16]) );

   return 1;
}
```