**TEMA 4**

**1. Describa las estructuras de datos que se crean en la memoria del sistema operativo para trabajar con archivos indicando para qué sirve cada una. Utiliza para ello como ejemplo el escenario en el que un programa abre el archivo ./dato.txt solo para lectura.**

Las estructuras son las siguientes:

● Bloque de control de archivos – una estructura donde hay un único elemento por archivo abierto en el sistema que contiene los metadatos del archivo (que se copian del disco) y otra información adicional necesaria para su gestión por parte del sistema operativo.

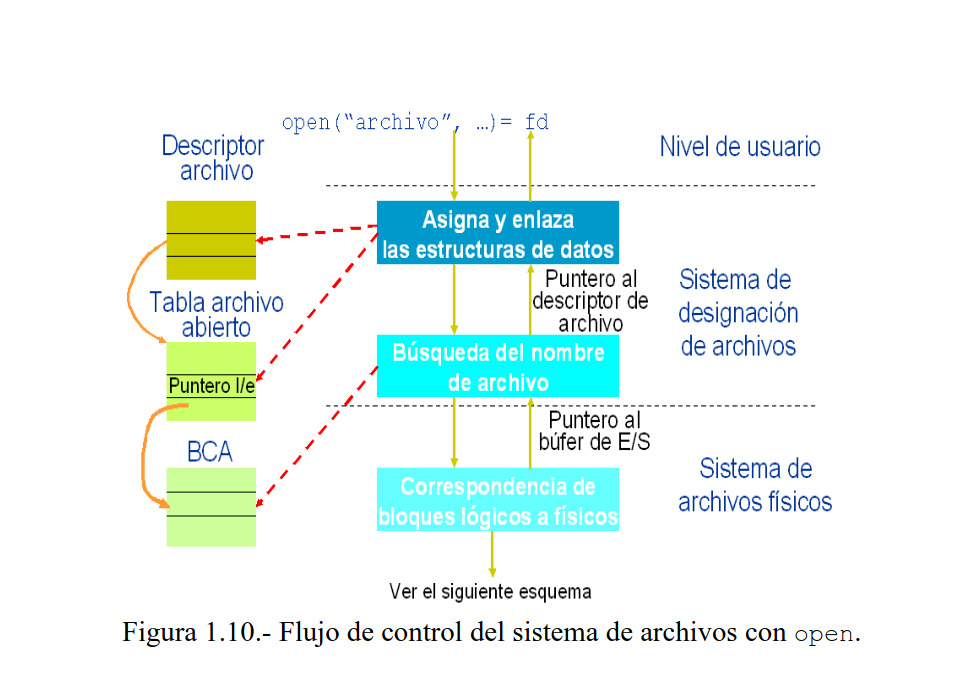
 ● Tabla de archivos abiertos – contiene un elemento por cada archivo abierto en el sistema. Cada elemento es una estructura de datos que suele contener la siguiente información: el puntero de lectura/escritura que marca la posición en el archivo donde se producirá la siguiente operación de lectura o escritura), información sobre el estado de apertura del archivo (si es de lectura, escritura, el tipo de archivo, si está bloqueado o no, etc.), y un puntero al bloque de control de archivos. Esta estructura es única para el sistema, e implementa el concepto de sesión de trabajo con un archivo.

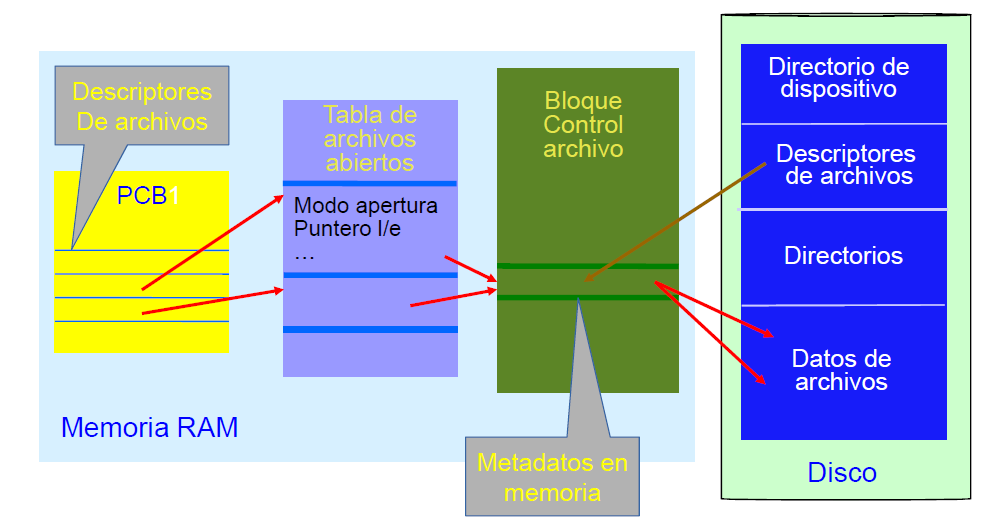
● Vector de descriptores de archivos – cada proceso posee como parte de su PCB (Bloque de Control del Proceso) una matriz de descriptores que los archivos que tiene abiertos. Para acceder a un archivo utiliza el índice devuelto por la llamada open(). Esta estructura, es una estructura por proceso.

La estructura Bloque de Control de Archivos mantiene una copia de los metadatos de cada archivo abierto en el sistema para minimizar los accesos a disco en cada operación realizada sobre el archivo que afecta estos. Dado que ya tenemos una estructura por archivo ¿cual es el objetivo de tener una Tabla de Archivos Abiertos? Cómo indicamos antes, cada ítem de esta estructura implementa la sesión de trabajo sobre un archivo. De esta forma, si abrimos un mismo archivo a través de sendas llamadas open(), tendremos un único elemento en el Bloque de Control de Archivos, pero tantas entradas en la Tabla de Archivos Abiertos como llamadas de apertura realizadas.

La Figura 1.9(imagen siguiente a la nota) muestra la situación donde hemos abierto dos veces un mismo archivo, una para lectura y otra para escritura. Con esta estructura, además, podemos mantener diferentes punteros de lectura/escritura cada uno para una sesión de trabajo.

Como podemos observar en la Figura cada entrada de BCA mantiene un contador de referencias que indica cuantas referencias existen hacia el, es decir, cuantos punteros estan apuntando a esa entrada. Este contador sirve al sistema de archivos para saber no solo las referencias sino que cuando el contador alcanza el valor 0, sabe que la entrada se puede limpiar por que no esta siendo utilizada por ningún otro componente del sistema.



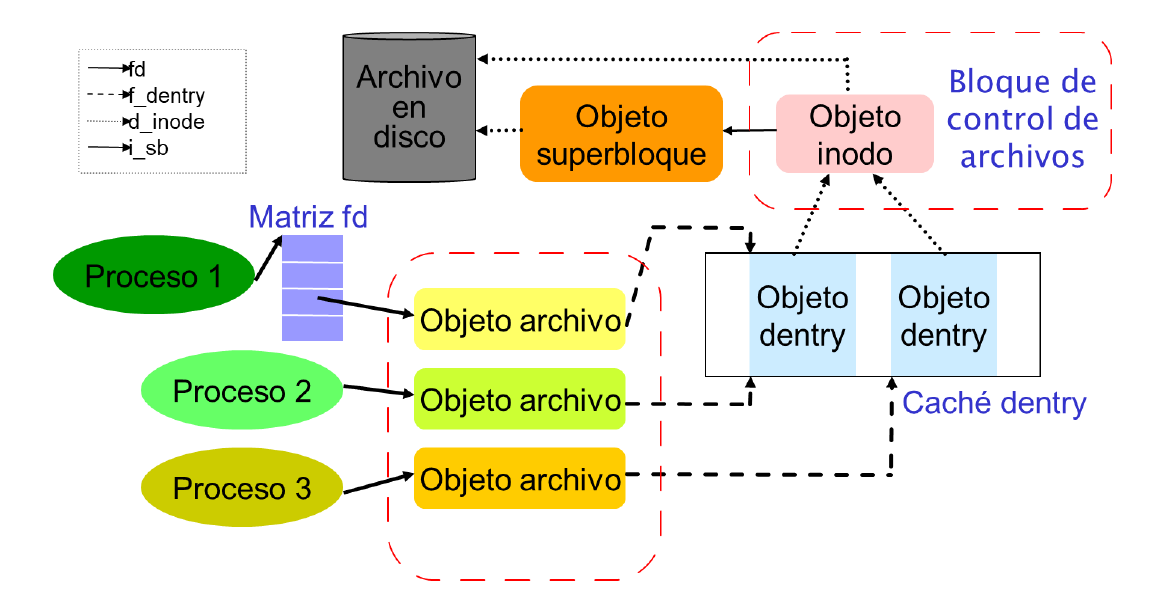
****

**2. ¿Qué diferencias habría en el caso de que el kernel fuese de Linux?**

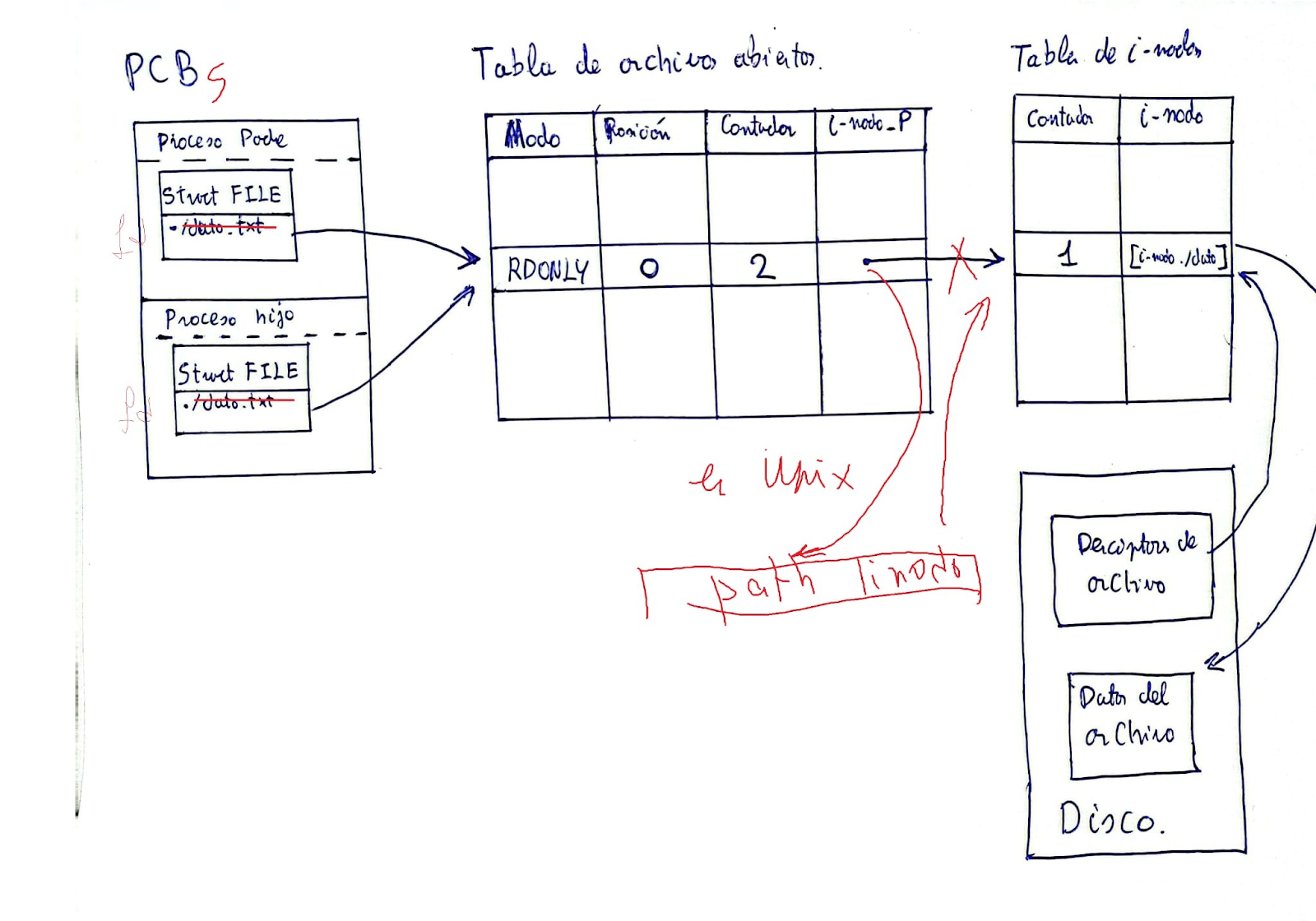
En el caso de Linux se utiliza un sistema de archivos virtual (VFS), la cual es la capa software entre el usuario y los diferentes tipos de SA. Ésta suministra al usuario/proceso una interfaz única para el acceso a archivos independientes del tipo de SA (soporta distintos tipos de sistemas de archivos).

VFS es un esquema orientado a objetos los cuales me van a permitir manipular los archivos. Se compone de:

* Objeto superbloque: Almacena información de un SA montado. (Se corresponde con el directorio de dispositivo). Contiene la lista de ficheros abiertos, puntero a la raíz…
* Objeto inodo: Almacena información de un SA específico (Bloque de control de archivos). Contiene los metadatos de un archivo en disco. A través del objeto inodo accedemos a la lista de inodos, la lista dentry…
* Objeto archivo: Representa un fichero abierto por un proceso. Se crea en respuesta a la llamada open y se destruye con close(). Los principales campos de esta estructura son un puntero de posicion del fichero (f\_pos), permisos (f\_mode), en este caso de solo lectura, contador de uso (f\_count), entre otros.
* Objeto dentry: Enlace entre una entrada de directorio y el archivo correspondiente.



**3. Si el proceso anterior realiza un fork() después de abrir el archivo, ¿cómo quedarían las estructuras de datos?**

****

**4. Explique el sistema de asignación de espacio en disco FAT y describe las ventajas e inconvenientes que presenta frente al método de asignación enlazado puro.**

FAT (File Allocation Table): Cada partición reserva un espacio para la FAT que contiene una entrada por bloque de disco y está indexada por el número de bloque. Además, una copia de la FAT en caché para reducir el tiempo de búsqueda.

Cada elemento de la tabla FAT representa un bloque, o agrupación de bloques (cluster), de disco. Un archivo viene representado por una lista enlazada de entradas de la FAT y el último elemento es un EOF.

Ventajas: mayor velocidad de acceso.

Inconveniente: manipulo la FAT en memoria y si se va la luz tengo problemas ya que se pierden las modificaciones que hayamos hecho en la FAT.

**5. ¿Indicar cual es la estructura de un directorio en los sistemas de archivos: (a) V-FAT de Windows (b) ext2 de Linux?**

1. **V-FAT de Windows**

El sistema de archivos FAT de forma resumida es una variante del método de asignación enlazada de ficheros. En este caso, creamos una lista enlazada a partir de una tabla que contiene una entrada para cada bloque de memoria donde se guarda el puntero al siguiente bloque.

Un directorio es un tipo especial de fichero, que en lugar de datos, contienen metadatos (punteros a otros ficheros).

Cada directorio tiene entradas de 32 bytes con la siguiente estructura:

* 8 bytes para el nombre de archivo
* 3 bytes para indicar el tipo de extensión (tipo de SA en el que reside).
* 1 byte para indicar cada tipo de atributo(cada uno un bit):
  + R = solo lectura
  + A = archivo
  + S = archivo de sistema
  + H = archivo oculto
  + D = directorio
  + V = etiqueta de volumen
* 10 bytes reservados para el tiempo y fecha de creación y la fecha del último acceso.
* 2 bytes para el tiempo de la última escritura.
* 2 bytes para la fecha de la última escritura.
* 2 ó 4 bytes para indicar el primer bloque (depende de si es un sistema FAT16 o FAT32 respectivamente).
* 4 bytes para el tamaño.

VFAT es una extensión del sistema de archivos FAT que se introduce en Windows 95 que permite entre otras funciones extender el nombre de los directorios a 255 caracteres, de forma que cuando se necesita más espacio para el nombre se utilizan entradas adicionales completando únicamente el nombre.

1. **ext2 de Linux**

En el SA ext2, los archivos se guardan en bloques de datos de igual longitud (se fija cuando se crea con mke2fs) de manera que el tamaño de cada fichero se redondea a un número entero de bloques (un archivo de 1025 bytes se guarda en dos bloques de 1024 bytes). Toda esta información se guarda en una estructura de datos llamada inodo que describe los bloques que ocupa un fichero y otros atributos (permisos, fecha de modificación, tipo de fichero…). Cada archivo en el sistema ext2 se describe con un único inodo que tiene un número que lo identifica.

Los directorios en ext2 son simplemente ficheros especiales que contienen punteros a los inodos de sus entradas de directorios.

Ext2 implementa los directorios como una clase especial de archivos cuyos bloques de datos almacenan nombres de archivos con su correspondiente número de inodo. Cada entrada tiene una serie de campos:

-> inode: Número de inodo

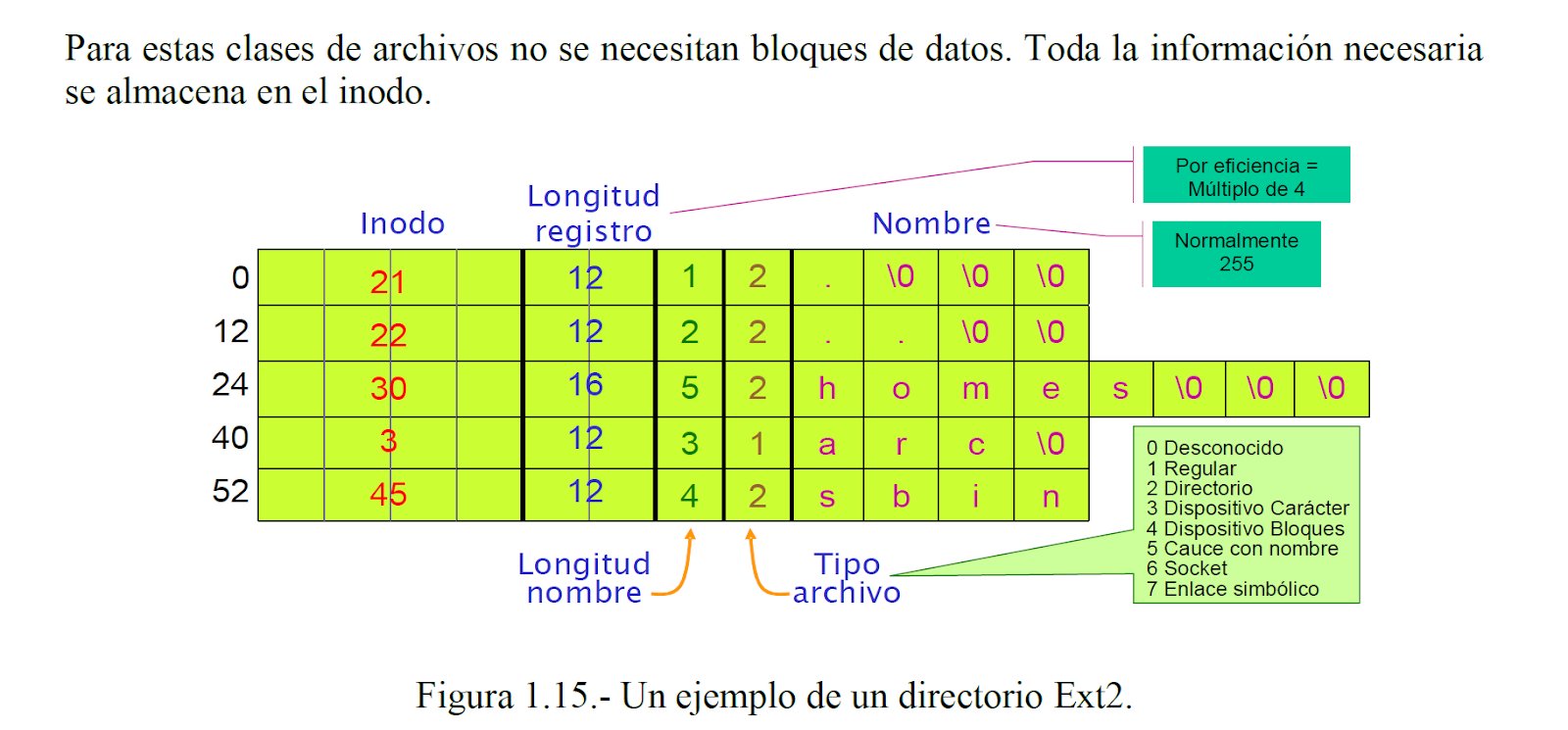
-> rec\_len: Longitud de la entrada del directorio(por razones de eficiencia, es      siempre múltiplo de 4, y, por tanto, se añade un nulo (\0), si es necesario, para rellenar el final del nombre del archivo). Puede interpretarse como un puntero a la siguiente entrada de directorio válida.

->name\_len: Longitud del nombre del archivo

->fyle\_type: Tipo de archivo

->name: Nombre del archivo

Si borramos una entrada de directorio, es suficiente poner el campo inode a 0 e incrementar adecuadamente el valor del campo rec\_len de la entrada anterior válida.



**6. Indicar cómo se implementa el concepto sesión de trabajo con un archivo de forma que el sistema permita a un proceso tener abiertas varias sesiones sobre un mismo archivo.**

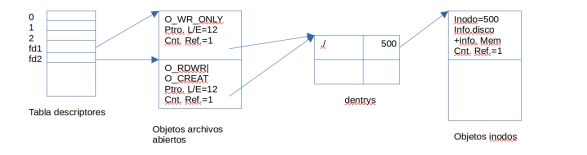
Mediante la estructura “archivo objeto” se describe la interacción entre un archivo abierto y un proceso, de manera que debe controlar el acceso de los diferentes procesos a tal archivo en la sesión de trabajo. Definimos sesión de trabajo sobre un archivo al conjunto de operaciones sobre un archivo que se realizan desde que abrimos un archivo hasta que lo cerramos. Para poder tener varias sesiones de trabajo, por ejemplo una para lectura y otra para escritura, se utilizan diferentes descriptores de archivos.

**7. Sea el programa que se muestra a continuación y un archivo de texto, denominado archivo\_texto, que contiene 30 caracteres “A”. Se pide que;**

**a) Dibujar los descriptores de las hebras que crea el kernel de Linux al ejecutar el programa, y sus principales contenidos.**

Los dos hilos comparten, entre otras EDs, la estructura files\_struct.

**b) Dibujar las estructuras de datos relativas al manejo del archivo citado y sus principales contenidos.**

****

Esta ED que he encontrado tiene dos archivos abiertos, sería lo mismo pero simplemente con un descriptor de archivo porque estamos sólo trabajando con uno. Habría que hacer una copia de fd1 en fd2 y que ambos apuntaran al único objeto de archivo abierto.

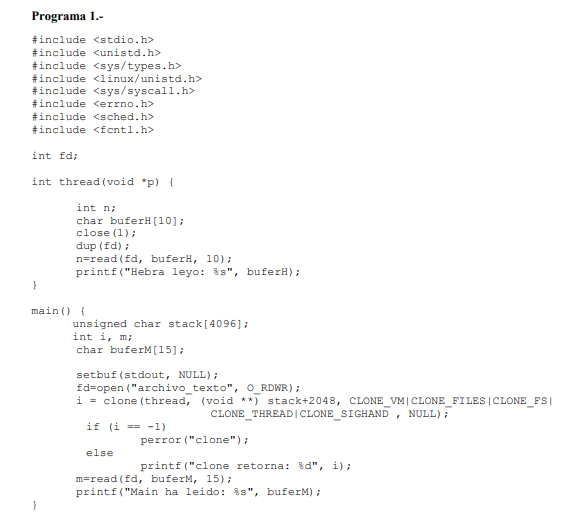
**c) Indicar uno de los 2 posibles contenidos del archivo tras ejecutar una vez el citado programa.**

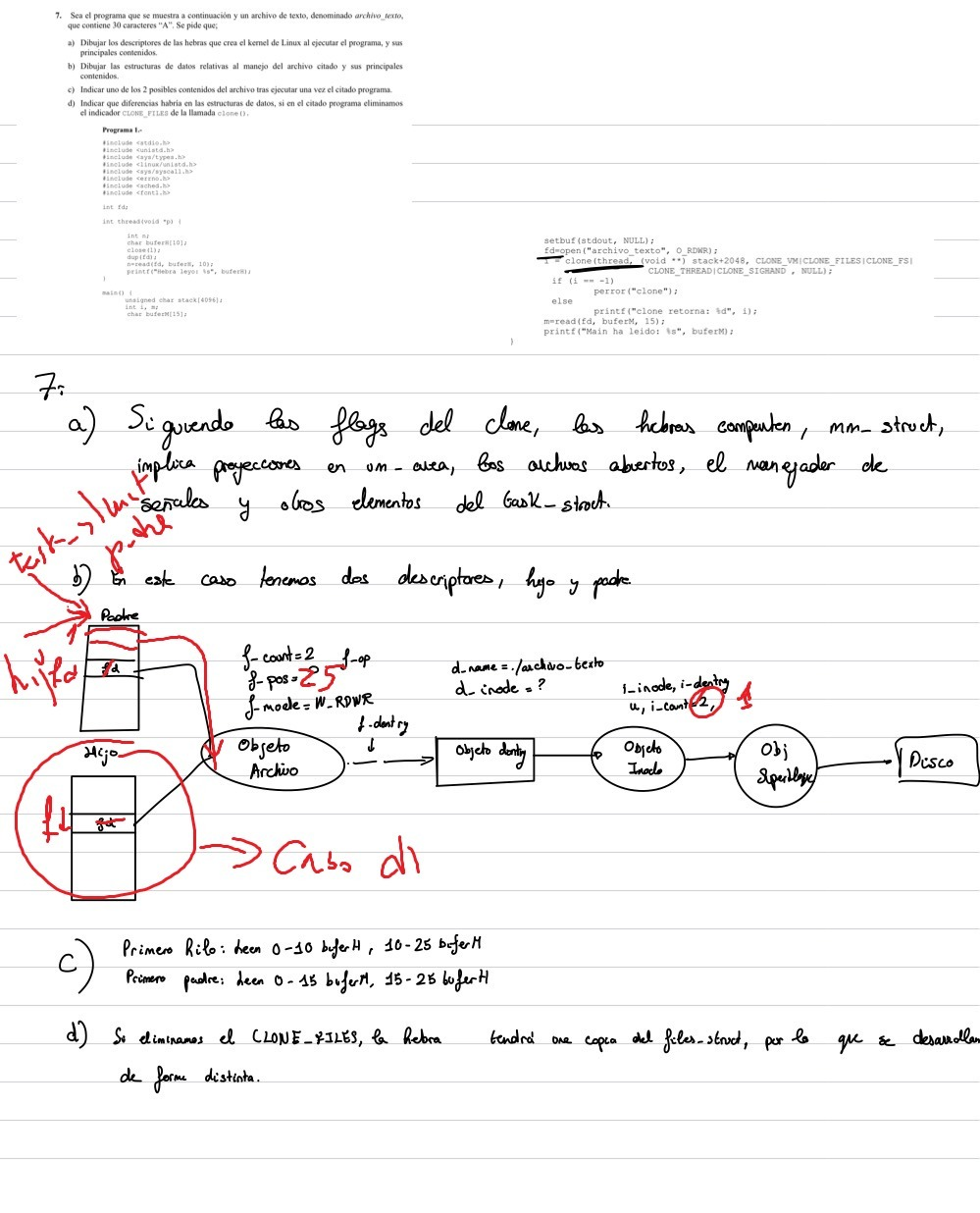
Se ejecuta primero read el hilo de main(): buferM=primeros 15 caracteres, buferH=caracteres del 15 al 25.

Se ejecuta primero el read del hilo creado: buferH=10 primeros caracteres, y buferM=15 siguientes.

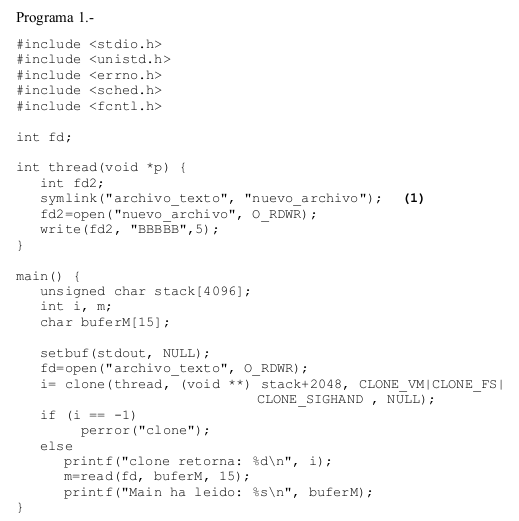
**d) Indicar qué diferencias habría en las estructuras de datos, si en el citado programa eliminamos el indicador CLONE\_FILES de la llamada clone().**

Cada proceso tiene su propia ED files\_struct. En el momento del clonado, se copia en el hilo la del main(), pero luego evoluciona de forma separada. En el hilo, se hace el dup() y el puntero de lectura/escritura es diferente.

****



**8. Sea el programa que se muestra a continuación y un archivo de texto, denominado archivo\_texto, que contiene 30 caracteres “A”. Se pide que:**

****

**a) Dibujar los descriptores de las hebras que crea el kernel de Linux al ejecutar el programa, y sus**

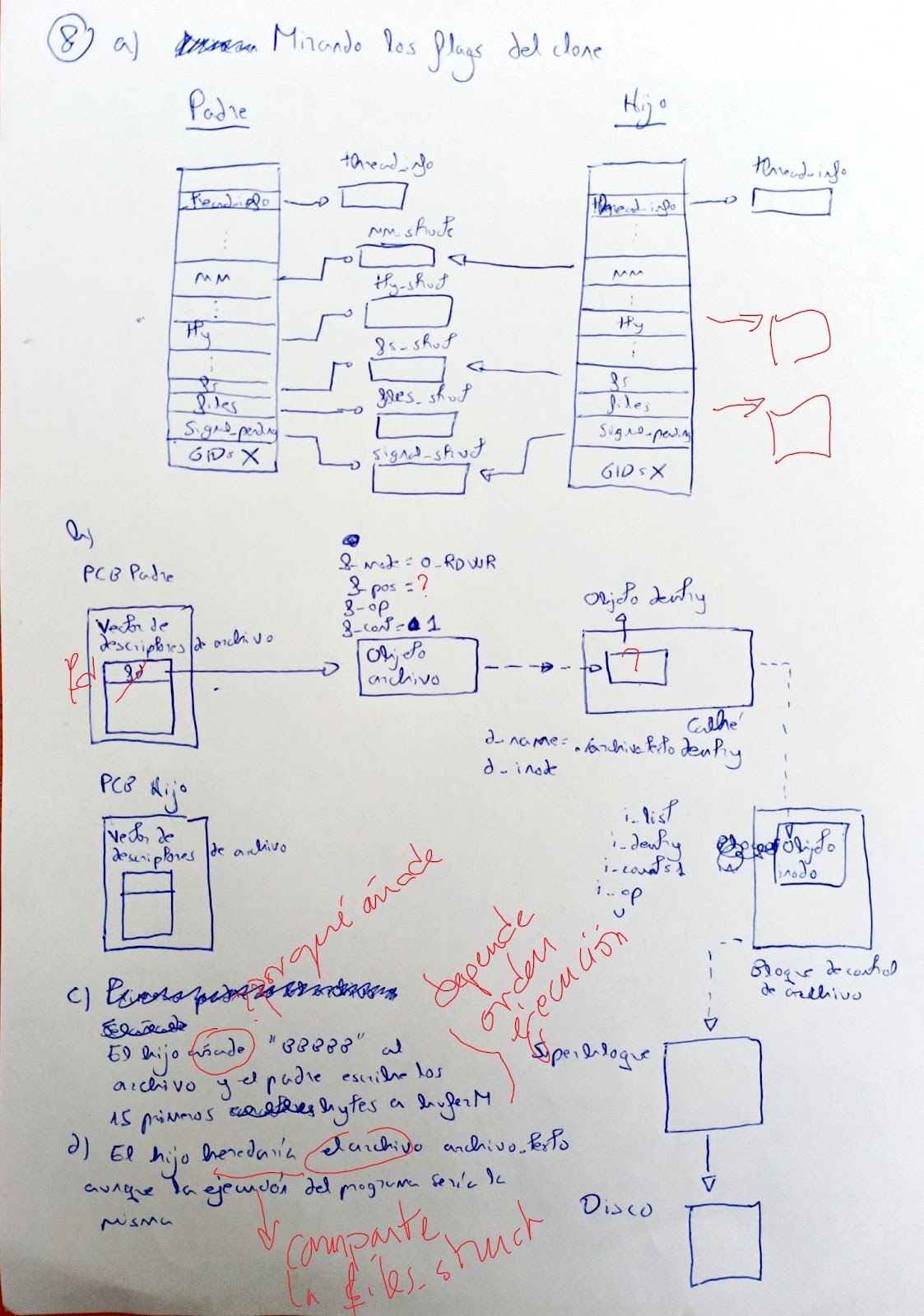
**principales contenidos.**

**b) Dibujar las estructuras de datos relativas al manejo de archivos y sus principales contenidos cuando se alcanza el punto marcado con (1) en el programa.**

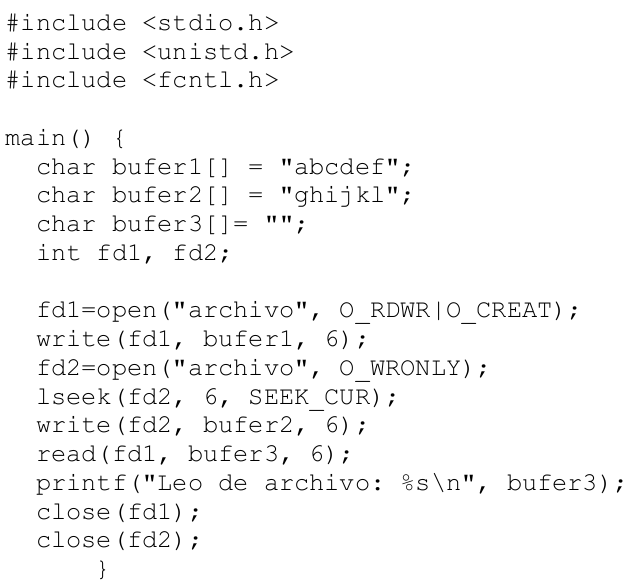
**c) Indicar el contenido del archivo archivo\_texto tras ejecutar una vez el citado programa.**

**d) Indicar qué diferencias habría en las estructuras de datos, si en el citado programa hubiésemos puesto**

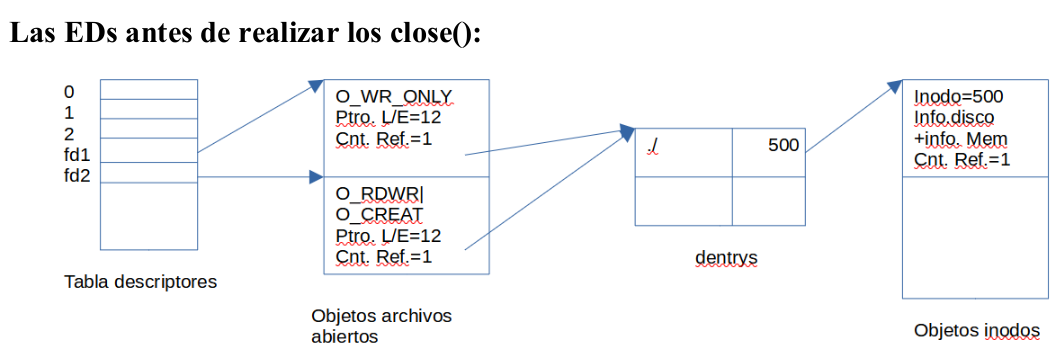
**el indicador CLONE\_FILES de la llamada clone().**

****

**9.Sea el siguiente programa:**

****

**a) Dibujar las estructuras de datos en memoria que utiliza el subsistema de archivos para manipular archivo y qué información relevante contienen las mismas.**



**b) ¿Qué mostrará en pantalla la instrucción printf al imprimir bufer3?**

“ghijkl”. El puntero de fd2 se mueve hasta la posición 12 y modifica el archivo, pero el puntero de fd1 sigue en posición 6, que es donde se había dejado al escribir previamente.

**10. En sistemas Unix, creamos una partición de disco para espacio de intercambio (swapping). Indicar que método de asignación de espacio es el más adecuado para esta partición. Comparar este esquema con el de Windows donde el intercambio se realiza sobre un archivo.**

- Linux: la asignación contigua que es la que tiene mejor tiempo de acceso. El problema es que debemos sobredimensionar la partición de swap por la fragmentación, ya que no se puede modificar dinámicamente.

 - Windows: se hace sobre un archivo. Peor tiempo de acceso (asignación dada por el sistema de archivos) pero puede crecer dinámicamente.