

**SISTEMAS OPERATIVOS**  
**2º Curso – Dobles Grados Informática**  
Ejercicios del Tema 4

1. Describa las estructuras de datos que se crean en la memoria del sistema operativo para trabajar con archivos indicando para que sirve cada una. Utiliza para ello como ejemplo el escenario en el que un programa abre el archivo *./dato.txt* solo para lectura.
2. ¿Qué diferencias habría en el caso de que el kernel fuese de Linux?
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.
5. ¿Indicar cual es la estructura de un directorio en los sistemas de archivos: (a) V-FAT de Windows (b) ext2 de Linux?
6. Indicar como 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.
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.
  - b) Dibujar las estructuras de datos relativas al manejo del archivo citado y sus principales contenidos.
  - c) Indicar uno de los 2 posibles contenidos del archivo tras ejecutar una vez el citado programa.
  - d) Indicar que diferencias habría en las estructuras de datos, si en el citado programa eliminamos el indicador `CLONE_FILES` de la llamada `clone()`.

**Programa 1.-**

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <linux/unistd.h>
#include <sys/syscall.h>
#include <errno.h>
#include <sched.h>
#include <fcntl.h>

int fd;

int thread(void *p) {

    int n;
    char buferH[10];
    close(1);
    dup(fd);
    n=read(fd, buferH, 10);
    printf("Hebra leyó: %s", buferH);
}

main() {
    unsigned char stack[4096];
    int i, m;
    char buferM[15];
```

```

        setbuf(stdout, NULL);
        fd=open("archivo_texto", O_RDWR);
        i = clone(thread, (void **) stack+2048, CLONE_VM|CLONE_FILES|CLONE_FS|
                    CLONE_THREAD|CLONE_SIGHAND , NULL);
        if (i == -1)
            perror("clone");
        else
            printf("clone retorna: %d", i);
        m=read(fd, buferM, 15);
        printf("Main ha leido: %s", buferM);
    }
}

```

**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:

- Dibujar los descriptores de las hebras que crea el kernel de Linux al ejecutar el programa, y sus principales contenidos.
- 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.
- Indicar el contenido del archivo *archivo\_texto* tras ejecutar una vez el citado programa.
- Indicar que diferencias habría en las estructuras de datos, si en el citado programa hubiésemos puesto el indicador *CLONE\_FILES* de la llamada *clone()*.

#### Programa 1.-

```

#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sched.h>
#include <fcntl.h>

int fd;

int thread(void *p) {
    int fd2;
    symlink("archivo_texto", "nuevo_archivo");      (1)
    fd2=open("nuevo_archivo", O_RDWR);
    write(fd2, "BBBBB",5);
}

main() {
    unsigned char stack[4096];
    int i, m;
    char buferM[15];

    setbuf(stdout, NULL);
    fd=open("archivo_texto", O_RDWR);
    i= clone(thread, (void **) stack+2048, CLONE_VM|CLONE_FS|
                CLONE_SIGHAND , NULL);
    if (i == -1)
        perror("clone");
    else
        printf("clone retorna: %d\n", i);
    m=read(fd, buferM, 15);
    printf("Main ha leido: %s\n", buferM);
}

```

**9.** Sea el siguiente programa:

```

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

main() {
    char bufer1[] = "abcdef";
    char bufer2[] = "ghijkl";
    char bufer3[] = "";
}

```

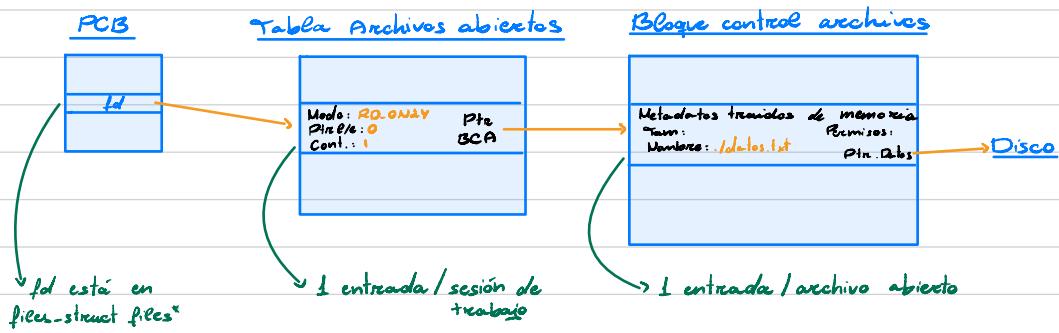
```
int fd1, fd2;

fd1=open("archivo", O_RDWR|O_CREAT);
write(fd1, bufer1, 6);
fd2=open("archivo", O_WRONLY);
lseek(fd2, 6, SEEK_CUR);
write(fd2, bufer2, 6);
read(fd1, bufer3, 6);
printf("Leo de archivo: %s\n", bufer3);
close(fd1);
close(fd2);
}
```

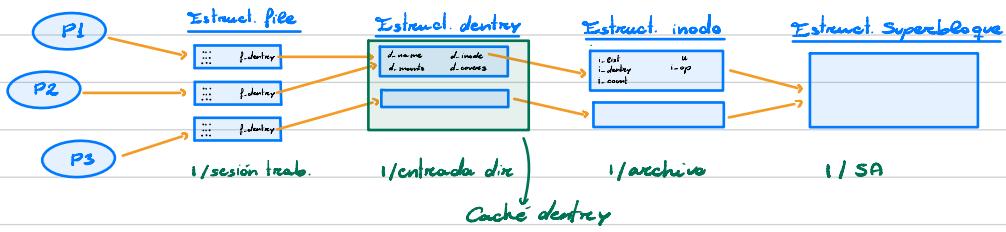
- a) Dibujar las estructuras de datos en memoria que utiliza el subsistema de archivos para manipular *archivo* y qué información relevante contienen las mismas.
- b) ¿Qué mostrará en pantalla la instrucción `printf` al imprimir `bufer3`?
10. En sistemas Unix, creamos una partición de disco para espacio 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 realizado sobre un archivo.

## Ejercicios Tema 4

- 1.- Para la gestión del archivo, se so en cada PCB mantiene un puntero al file descriptor guardado en una Tabla de Archivos Abiertos. Este guardará todos los datos relevantes para la sesión de trabajo sobre el archivo del proceso y un puntero a los metadatos del archivo, tales como nombre, tamaño, etc. y un puntero a los datos del archivo en disco



- 2.- En el caso de Linux, se utiliza el VFS, con las estructuras de este.



Los procesos guardan un puntero a un archivo **file** por cada archivo abierto en su **file\_struct files\***. Este tendrá campos como **f\_pos** (**ptr. E/e**), **f\_op** (**ptr. a operaciones**), **f\_count** (**Contador de usos**) y **f\_dentrey**, que apuntará a la estructura **dentrey** correspondiente al archivo, la cual vinculará el nombre de este con su **struct. inodo** correspondiente.

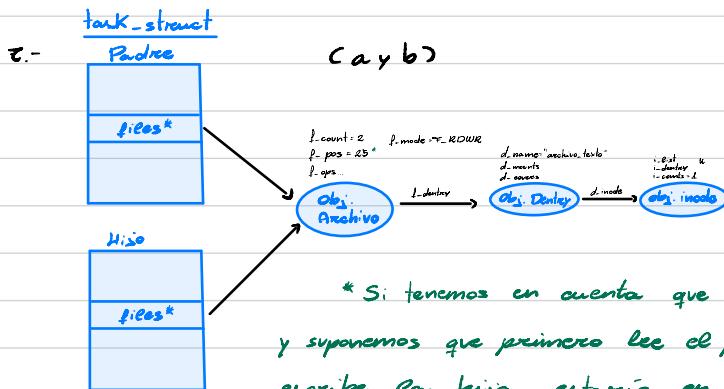
Este, a su vez, guardará todos los metadatos del archivo así como información de enlace como un ptre a lista de objetos inodo, o lista de objetos dentro o contador de uso. Guardará también un campo referente a su SA, el cual estará representando por un objeto superbloque

- 3.- Si se realiza después de abrir el archivo, el PCB del hijo heredará una copia de los file descrip. del padre, teniendo estas sesiones de trabajo diferentes pero sobre un mismo archivo
- 4.- En sistema FAT, utilizado sobre todo en pen-drives y discos extraíbles, guarda uno o varios bloques por partición para tablas FAT, indexadas por número de bloque y guardando en cada entrada el siguiente bloque o si es el último del archivo.  
Con respecto al sistema de enlazado puro, el FAT favorece más el acceso aleatorio, aunque con el inconveniente de "perder" el espacio de las tablas.
- 5.- En ambos son representadas como listas con 1 entrada por archivo. En cuanto al formato de estas:
  - a) <nombre><extensión><atributos>><tiempo y fecha de ... creación y ult.modif><Últ.acceso><1º bloque><tamaño>
  - b) <inodo><long\_registro><long.nombre><tipo><nombre>

6.- La implementación de esto se realiza a través de la Tabla de Archivos Abiertos, en la cual quedan un entrada por sesión de trabajo, pudiéndole referir varias al mismo archivo. En cada entrada se guardan los campos relevantes para esa sesión de trabajo

- Ptr. c/e
- Contador de uso
- Ptr. a operaciones
- Ptr. a BCA, donde se encuentran los metadatos.

10.- El más adecuado y utilizado comúnmente por los sistemas Unix es el contigo, simplificando la carga o lectura de la partición. Aunque este mecanismo es más complejo que el de archivo de Windows, tiene la ventaja de que no causa fragmentación de disco.



\* Si tenemos en cuenta que printf escribe en el archivo y suponemos que primero lee el padre y después lee y escribe en la hija, estaría en la posición 42. (EOF)

Además, los task\_struct por los flags de clone(), compartirán esp. de direcciones, descriptores de archivo, struct\_fs, GID y manejador de señales

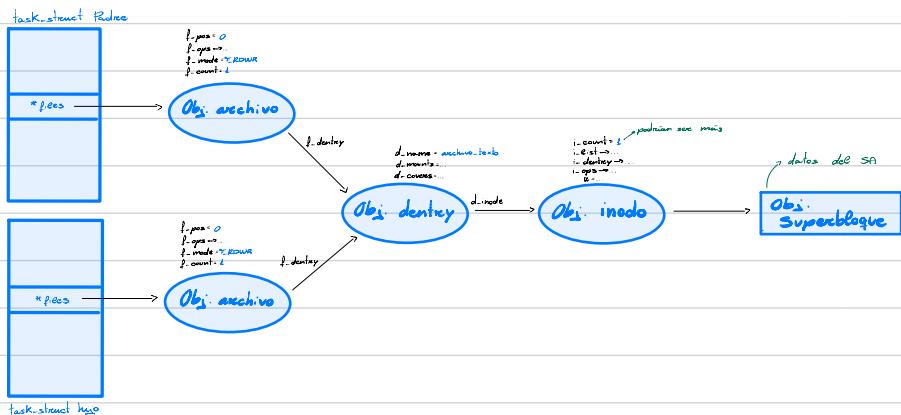
c) Suponiendo que primero lee el padre y más tarde se ejecuta el hijo, el contenido será

$\text{AA...A}^{\times 25} \text{Hebra leyo: AA...A}^{\times 10}$

d) Con el cloneC(), el hijo heredaría una copia del fd del padre en el momento en el que se ejecuta este, de forma que tendrían sesiones de trabajo diferentes.

8.- a) Como a cloneC() se le pasan las flags CLONE\_VM, CLONE\_FS, CLONE\_SIGHAND, los descriptores de los hebras padre e hijo compartirán los campos espacio virtual, datos de sistemas de archivos y manejadores de señales. Sin embargo, como no está presente CLONE\_FILES, la hebra hijo heredará una copia de la Tabla de Archivos Abiertos del padre.

b)

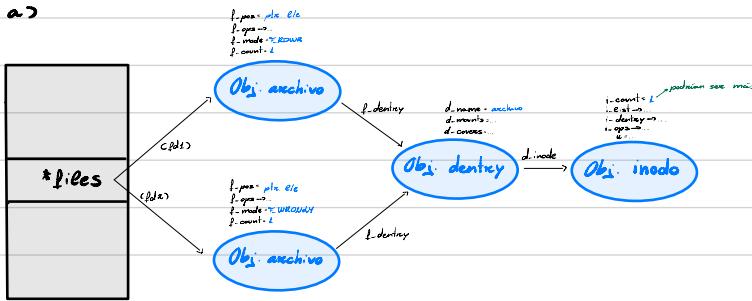


c) El padre no modifica el archivo ya que solo lee e imprime por pantalla. El hijo sin embargo, al escribir en un archivo enlazado con archivo texto, sí modificará este. El contenido del archivo será:

~~BBBBBAA...A~~  
x25

d) Si incluimos esta flag, ambas hebras compartirían un mismo objeto archivo y, por lo tanto, la sesión de trabajo, afectándose el uno al otro. En cuanto a los campos de estas, se modificará únicamente file → f\_count = 2 en el objeto archivo.

9.- a)



b) En principio, a la hora de realizar el read() que guarda en el bufer3, saltaría un segmentation fault ya que es un array de tamaño l, al ser igual a "\0" en un principio. Si dividimos esto, el búfer 3 quedará "ghijkl" ya que se escribe primero búfer1 con una sesión de trabajo, con esa otra se escribe búfer2 a continuación para más tarde leer desde la primera.