UNIVERSIDAD NEBRIJA

Grado en Ingeniería Informática

Sistemas Operativos

Práctica 5

Índice

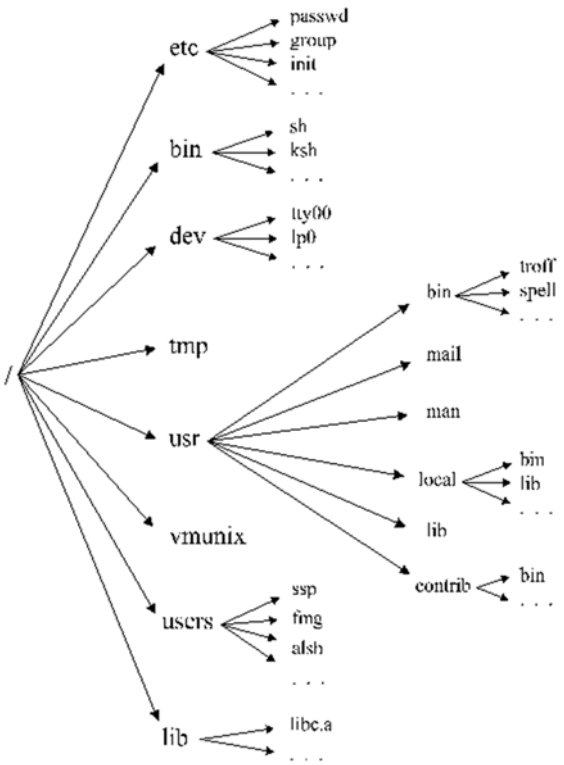
Práctica 5: Gestión de Ficheros y Directorios

Introducción

Un archivo se puede definir como un conjunto de datos con un nombre asociado. Los archivos suelen residir en dispositivos de almacenamiento secundario tales como discos duros. Algunos sistemas operativos imponen a los archivos una estructura determinada bien definida. Este no es el caso de UNIX, donde un archivo no es más que una secuencia de bytes (8bits).

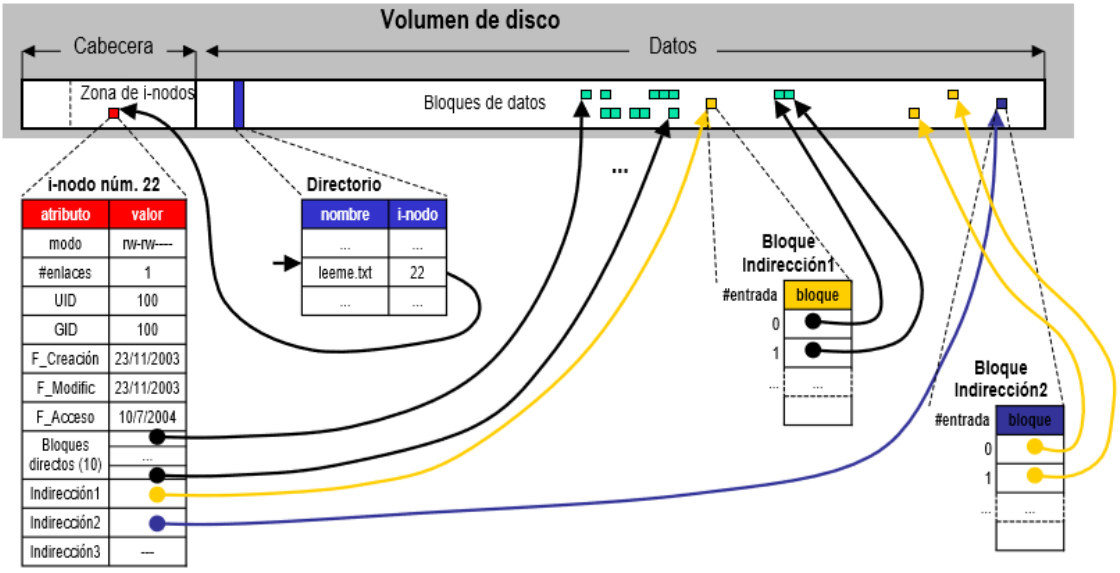
Un sistema de archivos se debe entender como aquella parte del sistema responsable de la administración de los datos en dispositivos de almacenamiento secundario. Por lo tanto, es el sistema de archivos el que debe proporcionar los medios necesarios para un acceso y almacenamiento seguro y privado de la información.

En UNIX los archivos están organizados en lo que se conoce como directorios. Un directorio no es más que un archivo especial, el cual contiene información que permite localizar en el dispositivo otros archivos. Los directorios pueden contener a su vez nuevos directorios, los cuales se denominan subdirectorios. A la estructura resultante de esta organización se la conoce con el nombre de estructura en árbol invertido. Un ejemplo típico de arbol de directorios UNIX lo tenemos representado en la figura:

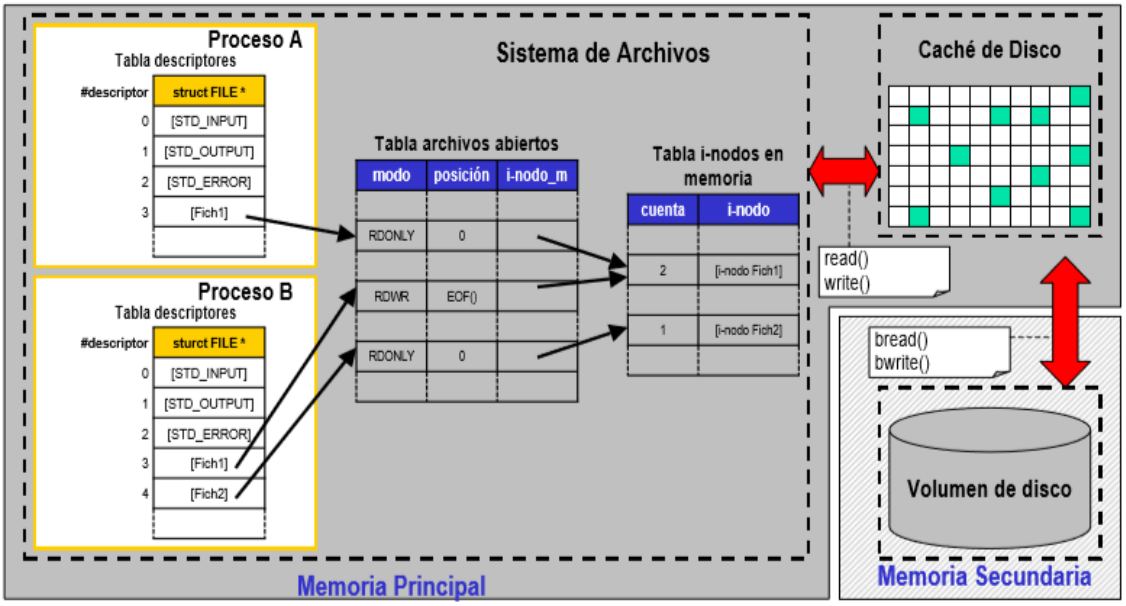


Ficheros en UNIX

- Representación ficheros UNIX **en disco**



- Representación ficheros UNIX **en memoria**



Operaciones sobre ficheros

La biblioteca estándar ofrece rutinas cómodas, construidas a partir de las llamadas al sistema. (La interfaz con estos servicios se encuentra en `<stdio.h>`).

Algunas de las operaciones que trabajan con ficheros son las siguientes:

- Crear y borrar un fichero
- Abrir y cerrar un fichero
- Leer en un fichero
- Escribir en un fichero
- Desplazarse por un fichero

Para trabajar con un fichero hay primero que abrirlo con una invocación a la función **`open`**. Ésta devuelve un **descriptor de fichero** (*file descriptor*), un número entero que servirá de identificador de fichero en futuras operaciones. Finalmente hay que cerrar el fichero, con la función **`close`**, para liberar los recursos que tengamos asignados.

Existen al menos tres descriptores ya establecidos en la ejecución de un programa (ya los han abierto por nosotros).

- El descriptor 0 es la entrada estándar (normalmente el teclado)
- El descriptor 1 es la salida estándar (normalmente la pantalla)
- El descriptor 2 el fichero estándar de visualización de errores (también la pantalla, normalmente).

Los ficheros en UNIX permiten tanto el acceso directo como el secuencial. Cada fichero abierto dispone de un puntero que se mueve con cada lectura o escritura. Hay una función especial llamada **`lseek`** para posicionar ese puntero donde se quiera dentro del fichero.

Las llamadas al sistema **`creat`** y **`open`** admiten un parámetro entero en el que se especifican los permisos con los que se crea un archivo.

Los permisos se forman como un número de 9 bits, en el que cada bit representa un permiso, tal y como se muestra en el cuadro (es el mismo orden con el que aparecen cuando hacemos un **`ls`**).

Se toman los nueve permisos como tres números consecutivos de 3 bits cada uno. Un bit vale 1 si su permiso correspondiente está activado y 0 en caso contrario.

Por ejemplo, los permisos **`rw-r--r-x`** son el número octal 0645.

```
/* Crea un fichero con permisos RW-R--R-- */

int fd = creat ( "mi_fichero", 0644);
```

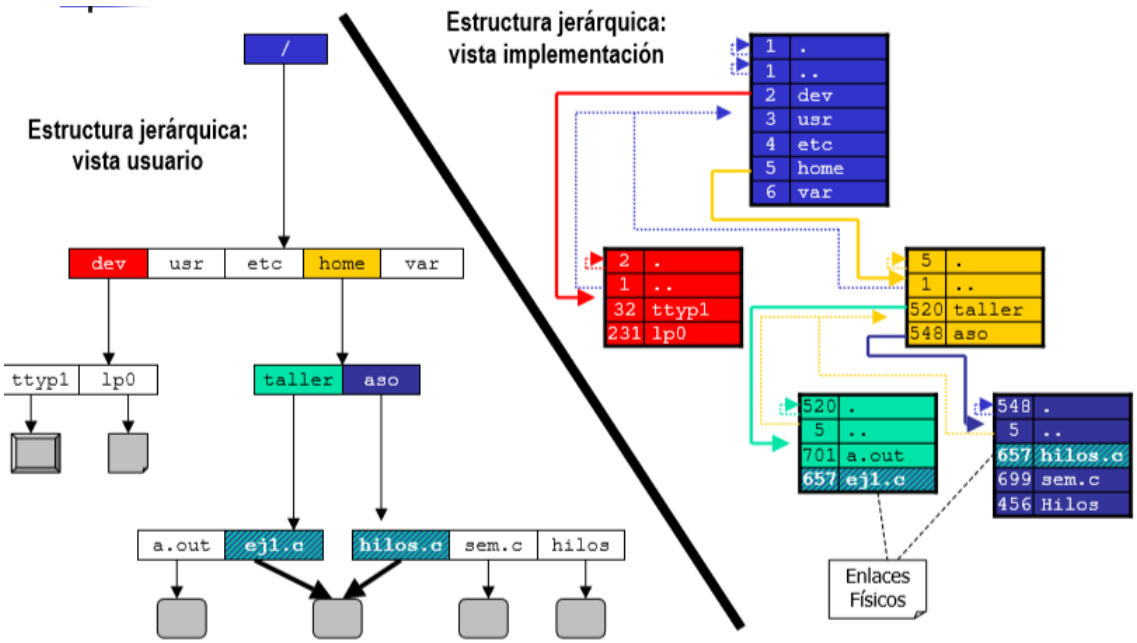
Ficheros	
open	Apertura/creación de ficheros
read	Lectura de ficheros
write	Escritura de ficheros
close	Cierre de un fichero
lseek	Posicionamiento en un fichero
stat	Obtener información del i-nodo de un fichero

Ejemplo creación y escritura de un fichero:

```
#include <string.h>     /* Función strlen() */
#include <fcntl.h>      /* Modos de apertura y función open()*/
#include <stdlib.h>     /* Funciones write() y close() */

main ( int argc, char* argv[] )
{
   /* Cadena que se va a escribir */
   const char* cadena = "Hola, mundo";

   /* Creación y apertura del fichero */
   int fichero = open ("mi_fichero", O_CREAT|O_WRONLY,0644);

   /* Comprobación de errores */
   if (fichero==-1)
   {
        perror("Error al abrir fichero:");
        exit(1);
   }

   /* Escritura de la cadena */
   write(fichero, cadena, strlen(cadena));
   close(fichero);
   return 0;

}
```

Directorios en UNIX:

En la siguiente figura se muestra la vista de usuario frente a la vista de implementación del sistema de directorios en UNIX:



Directorios	
mkdir	Crear un directorio
rmdir	Eliminación de un directorio vacío
opendir	Abrir un directorio
readdir	Retornar la siguiente entrada de un directorio
closedir	Cerrar un directorio
Link	Crear una nueva entrada de directorio para otra existente
Unlink	Eliminar una entrada de directorio

mkdir/rmdir: Crear/eliminar directorios

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

int mkdir (char *ruta, mode_t modo)
int rmdir (char *ruta)
```

opendir, readdir, closedir: Manejo de directorios

```
#include <dirent.h>
#include <systypes.h>

DIR *opendir (char *ruta)
struct dirent *readdir (DIR *dirptr)
int closedir (DIR *dirptr)
```

Descripción

- **opendir**: abre el directorio especificado y retorna un puntero a un directorio
- **readdir**: retorna un puntero a un `struct dirent` que contiene los detalles de la siguiente entrada de directorio. El nombre del archivo es accesible mediante `direntp->dname`. Llamando repetidas veces a esta función se recorrerá secuencialmente el directorio. Devuelve `NULL` al llegar al final

```
struct dirent {
  long           d_ino;       /*número de i-nodo */
  off_t          d_off;       /* desplazamiento hasta la siguiente entrada */
  unsigned short d_reclen;    /* Longitud de este registro */
  unsigned char  d_type;      /* Tipo de fichero */
  char           d_name[256]; /* Nombre de fichero */
```

- **closedir**: cierra un directorio

Ejemplo: Programa que lista un directorio

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <errno.h>

int main (int argc, char *argv[]) {
  DIR *dir_ptr;
  struct dirent *ent_dir_ptr;
  /* Comprobación argumentos */
  if (argc != 2) {
    fprintf(stderr, "Uso: %s directorio\n", argv[0]);
    exit(1);
  }
  /* Apertura del directorio */
  if ( (dir_ptr = opendir(argv[1])) == NULL ) {
    fprintf(stderr, "Error al abrir el directorio %s (%s)\n", argv[1], strerror(errno) );
    exit(1);
  }
  /* Lectura del directorio */
  while ( (ent_dir_ptr = readdir(dir_ptr)) != NULL )
    printf("%s\n", ent_dir_ptr -> d_name);
  /* Cierre del directorio */
  closedir(dir_ptr);
  exit(0);
}
```

Enlaces físicos

link, unlink: Crea, borrar entradas de directorio

```
#include <unistd.h>

int link (char *ruta1, char *ruta2)
int unlink (char *ruta)
```

Descripción

- A las entradas de directorio se las denomina enlaces (*links*)
- Cuando se crea un fichero, se genera automáticamente una entrada de directorio
- **link**: crea una nueva entrada de directorio ruta2 que apunta al mismo fichero que una entrada de directorio ruta1, ya existente. Ambas entradas deben estar en el mismo sistema de ficheros
- **unlink**: elimina una entrada de directorio y decrementa el número de enlaces al fichero, en su correspondiente i-nodo
  - Si el número de enlaces pasa a ser cero, entonces se elimina el fichero y su i-nodo
  - Si un proceso lo tiene abierto cuando se invoca **unlink**, se borra la entrada y se difiere la eliminación del fichero al momento en que se cierre

Enlaces Hard o Soft links:

Enlace físico: `ln <fich> <enlace_fis>`
- Se refiere al mismo i-nodo que el fichero con el que se enlaza, en este ejemplo 30998

Enlace lógico: `ln -s <fich> <enlace_sim>`
- Se refiere a un i-nodo diferente (30996) que es un archivo diferente, pero también de un tipo especial (tipo l)

```
[fdiaz@localhost fdiaz]$ ln ej1.c enlacefisico
[fdiaz@localhost fdiaz]$
[fdiaz@localhost fdiaz]$ ln -s ej1.c enlacesimbolico
[fdiaz@localhost fdiaz]$
[fdiaz@localhost fdiaz]$ ls -li
total 88
123906 drwxrwxr-x    3 fdiaz    fdiaz        4096 oct 25 11:00 Desktop/
 61954 drwxr-xr-x    2 fdiaz    fdiaz        4096 oct 25 11:00 Documents/
 30994 -rwxrwxr-x    1 fdiaz    fdiaz       11407 oct 26 10:51 ej1*
 30998 -rw-rw-r--    2 fdiaz    fdiaz          63 oct 26 10:50 ej1.c
 30998 -rw-rw-r--    2 fdiaz    fdiaz          63 oct 26 10:50 enlacefisico
 30996 lrwxrwxrwx    1 fdiaz    fdiaz           5 nov 25 05:56 enlacesimbolico -> ej1.c
 30999 -rwxrwxr-x    1 fdiaz    fdiaz       12294 oct 26 11:36 hilos1*
 31000 -rw-rw-r--    1 fdiaz    fdiaz         432 oct 26 11:36 hilos1.c
 31002 -rwxrwxr-x    1 fdiaz    fdiaz       12180 oct 26 12:18 hilos2*
 31004 -rw-rw-r--    1 fdiaz    fdiaz         641 oct 26 12:18 hilos2.c
 31003 -rwxrwxr-x    1 fdiaz    fdiaz       12467 oct 26 12:52 hilos3*
 31007 -rw-rw-r--    1 fdiaz    fdiaz         769 oct 26 12:52 hilos3.c
 61953 drwx------    2 fdiaz    fdiaz        4096 oct 26 12:52 tmp/
[fdiaz@localhost fdiaz]$
```

chmod/chown: modificación de los atributos de protección de ficheros

```
#include <sys/types.h>
#include <sys/stat.h>

int chmod (char *ruta, mode_t modo)
int chown (char *ruta, uid_t propietario, gid_t grupo)
```

Descripción

- **chmod**: modifica los bits de permiso (**rwx**, **suid**, **sgid**, ...) con los bits dados por **modo**, para el fichero especificado por **ruta**
- **chown**: asigna nuevo propietario y grupo a un fichero
- Existen variantes **fchmod** y **fchown** que trabajan con descriptores de fichero en lugar de con nombres de ficheros

Entregables:

1. Crear un programa que lea diez caracteres, a partir de la posición 100, de un fichero ya existente.

2. Crear un programa que copie un archivo en otro.

3. Escribir un programa que calcule la suma de los bytes ocupados por todos los ficheros y directorios que estén contenidos a partir de un directorio dado como parámetro. ¿Qué ocurre cuando hay dos enlaces hard que hacen referencia al mismo fichero? Haz que en estos casos el espacio ocupado se considere sólo una vez.