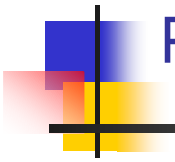
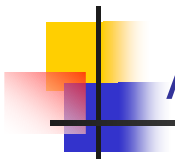


Ficheros y Directorios

Ampliación de Sistemas Operativos (prácticas)

E.U. Informática en Segovia

Universidad de Valladolid

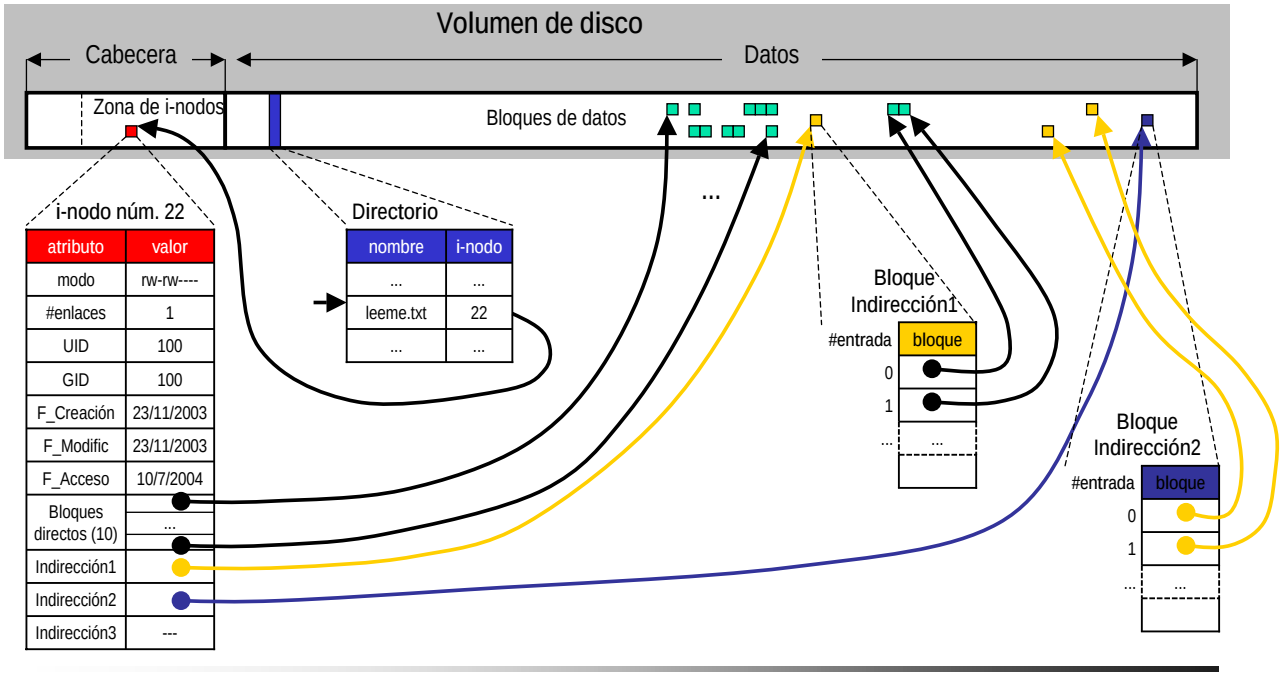


Archivos

- Archivos UNIX
  - Abstracción de datos que representa un espacio de almacenamiento, por lo general, en memoria secundaria y, por tanto, con carácter permanente
  - Tipos de archivos:
    - Regular: representan un fichero convencional (programa, texto, etc.)
    - Tuberías: ficheros sin nombre (soportados en memoria principal), de acceso secuencial y utilizados para comunicación entre procesos relacionados (padre-hijo)
    - Tuberías sin nombre: ficheros con nombre (soportados en memoria secundaria), de acceso secuencial y utilizados para comunicación entre procesos, no necesariamente, relacionados.
    - Directorio: Archivo especial que el sistema operativo utiliza para asociar nombres “lógicos” a los archivos
    - Especial: Archivo especial utilizado por el sistema operativo para representar a un dispositivo
  - Atributos de los archivos UNIX
    - Tipo de archivo
    - Propietario (*owner* UID) y grupo propietario (*owner* GID)
    - Permisos de acceso, números de enlaces
    - Instantes de creación, último acceso y última modificación
    - Tamaño

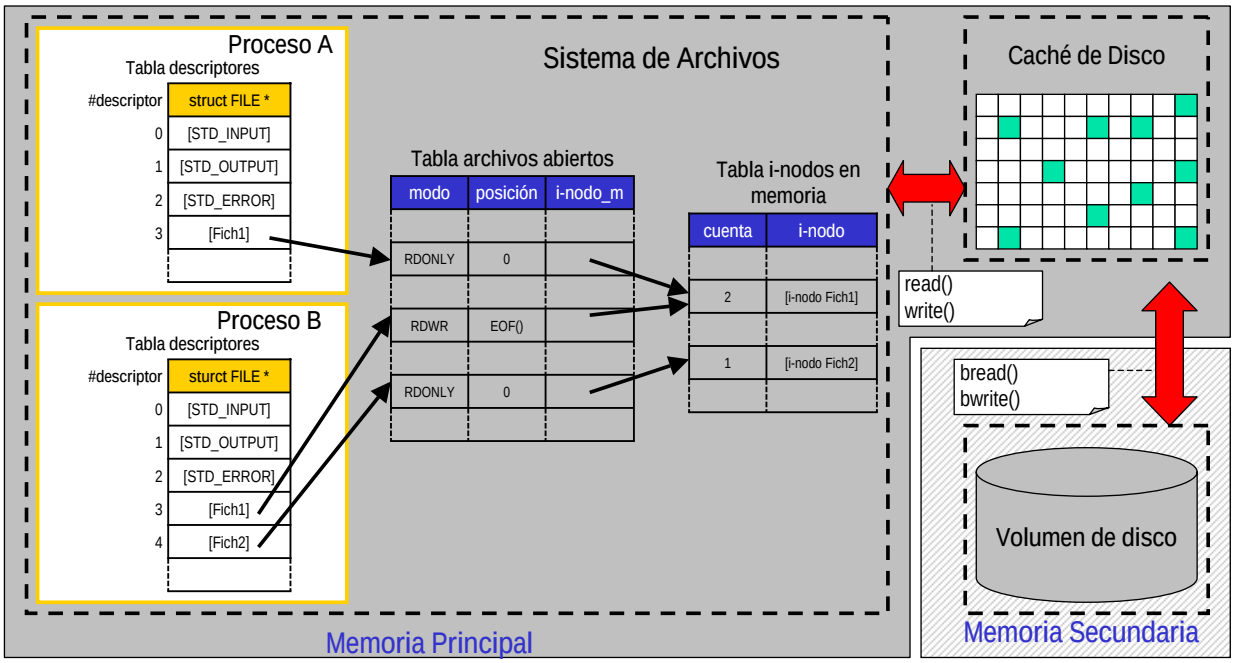
Ficheros

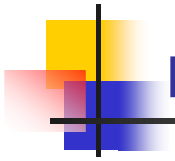
- Representación ficheros UNIX **en disco**



Ficheros

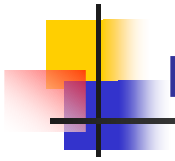
- Representación ficheros UNIX **en memoria**





Ficheros

- El sistema operativo lleva cuenta de los ficheros abiertos por un proceso mediante una tabla de descriptores de archivo, que es local
  - Todo proceso, tiene abiertos por defectos tres archivos estándar: de entrada (`stdin`), de salida (`stdout`) y de error (`stderr`), descriptores, 0, 1 y 2, respectivamente
  - Cuando un proceso abre un fichero ...
    - El proceso recibe un descriptor de fichero que es, simplemente, el índice de la primera entrada libre de la tabla local de descriptores. Este descriptor será necesario para invocar al resto de funciones de archivos
    - Además, se asigna una entrada de la tabla global de archivos abiertos y en ella se llevará cuenta de la posición actual dentro del archivo y del modo de apertura del archivo
    - Si el archivo no estaba previamente abierto, se carga en memoria su i-nodo correspondiente con lo que se dispone de la información necesaria para localizar el archivo en disco
    - Finalmente, el sistema de archivos no lee directamente bloques de disco, sino que lo hace el sistema caché de disco



Ficheros: llamadas al sistema

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

Ficheros: open

- **open**: Apertura/creación de ficheros

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

int open (const char *ruta, int flags)
int open (const char *ruta, int flags, mode_t modo)
```

- Descripción
  - Asocia un descriptor de fichero con un fichero para su posterior uso en otras llamadas al sistema (`read`, `write`, etc.)
  - El descriptor se corresponde con el índice de la primera entrada libre en la tabla de descriptores. Los descriptores de ficheros abiertos son atributos heredables al hacer `fork`
  - Flags
    - `O_RDONLY`, `O_WRONLY`, `O_RDWR`
    - `O_CREAT`, `O_EXCL`, `O_TRUNC`, `O_APPEND`
  - Modo
    - Directamente en octal. Ejemplos: 0755, 04755
    - Utilizando macros a nivel de bit predefinidas:
      - `S_IRUSR (0400), S_IWUSR (0200), S_IXUSR (0100), S_IRWXU (0700), S_IRGRP (0040), S_IWGRP (0020), S_IXGRP (0010), S_IRWXG (0070), S_IROTH (0004), S_IWOTH (0002), S_IXOTH (0001), S_IRWXO (0007), S_ISUID (04000), S_ISGID (02000)`

Ficheros: read/write

- **`read/write`**: lectura/escritura de ficheros

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t nb)
ssize_t write(ubt fd, const void *buf, size_t nb)
```

- Descripción
  - `read`: Solicita leer `nb` bytes del fichero cuyo descriptor es `fd` y que éstos sean almacenados en el buffer `buf`. El tamaño de `buf` debe ser mayor igual que `nb`
  - `write`: Solicita la escritura de `nb` bytes, tomados del buffer `buf`, en el fichero cuyo descriptor es `fd`
  - Por defecto, `read` y `write` son bloqueantes, pero pueden ser interrumpidas por una señal
  - `read` puede leer menos bytes de los solicitados, si se llega al final del fichero
- Valor de retorno
  - El número de bytes escritos o leídos efectivamente
  - -1: error o interrupción por señal (`errno = EINTR`, en este último caso)
  - 0: intento de leer después de final de fichero
- Errores
  - `fd` no es un descriptor válido
  - `buf` no es una dirección válida
  - Disco lleno o de solo lectura, en caso de escritura

Ficheros: lseek y close

- **lseek**: posicionamiento en un fichero

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

off_t lseek (int fd, off_t offset, int base)
```

- Descripción
  - Desplaza el puntero de acceso en un fichero `offset` bytes respecto a la base especificada por:
    - SEEK_SER: respecto al principio del fichero
    - SEEK_CUR: respecto a la posición actual
    - SEEK_END: respecto al final del fichero
- Valor de retorno
  - Nueva posición del puntero de acceso (desde el inicio del fichero)
  - -1: en caso de error (`fd` no se corresponde con un archivo regular)
- **close**: cierre de un fichero

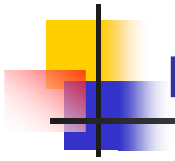
```
#include <unistd.h>

int close (int fd)
```

- Descripción
  - Cierra el descriptor de fichero `fd`
- Valor de retorno
  - 0: en caso de éxito
  - -1: en caso de error (`fd` no es un descriptor válido)

Ficheros: ejemplo

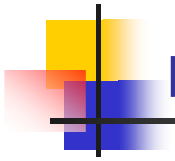
```
#include <sys/stat.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

#define TAM 1024
#define NEW (O_WRONLY|O_CREAT|O_EXCL)
#define PERM (S_IRUSR|S_IWUSR)

int main (int argc, char *argv[]) {
  int fd_orig, fd_dest;
  int leidos;
  char buffer[TAM];

  /* Comprobación argumentos */
  if (argc != 3) {
    fprintf(stderr, "Uso: %s fich_orig
    fich_dest\n", argv[0]);
    exit(1);
  }
```

```
  /* Apertura ficheros */
  if ((fd_orig= open(argv[1], O_RDONLY)) == -1) {
    fprintf(stderr, "Fallo al abrir %s\n", argv[1]);
    exit(1);
  }
  if ((fd_dest= open(argv[2], NEW, PERM)) == -1) {
    fprintf(stderr, "Fallo al crear %s\n", argv[2]);
    exit(1);
  }
  /* Copia del archivo */
  while ( (leidos=read(fd_orig, buffer, TAM)) > 0)
    if (write(fd_dest, buffer, leidos) != leidos) {
      fprintf(stderr, "Error al copiar\n");
      exit(1);
    }
  if (leidos == -1) {
    fprintf(stderr, "Error al leer\n");
    exit(1);
  }

  /* Cierre de archivos y finalización */
  close(fd_orig);
  close(fd_dest);
  exit(0);
}
```



Directorios

- Concepto
  - Es un tipo de fichero que permite organizar los ficheros jerárquicamente
  - Establece la forma de asociar a los ficheros un nombre lógico o nombre de fichero
- Implementación de directorios
  - La estructura básica es la de un fichero regular
  - Linux/Unix interpreta el fichero como si se encontrase organizado en registros
    - A cada registro se le denomina **entrada de directorio**
    - Cada entrada de directorio consta de dos campos:
      - Número de i-nodo
      - Nombre del fichero
    - En todo directorio, siempre existen al menos dos entradas:
      - . directorio actual
      - .. directorio padre



Directorios

- Esquema de nombres de ficheros
  - Un fichero se identifica mediante su vía o **ruta de acceso**, que puede ser absoluta (comienza por /, por ejemplo, /home/taller/ejemplo.txt) o relativa al directorio actual (por ejemplo, suponiendo que el directorio actual es /home, se puede escribir taller/ejemplo.txt)
  - Los ficheros pueden tener más de un nombre, cada uno de ellos, denominado **enlace**
  - Existen dos tipos de enlace
    - **Enlaces físicos**: son entradas de directorio que referencian al mismo i-nodo
      - El fichero sólo se elimina del disco cuando se borran todos los enlaces físicos (todas las entradas de directorio que lo referencian)
      - Sólo se permite (salvo el administrador) enlazar ficheros regulares (no directorios)
    - **Enlaces simbólicos**: entrada de directorio A que referencia otra entrada de directorio B, a través de un fichero de tipo diferenciado “enlace simbólico”
      - El fichero se elimina cuando se borra el enlace físico. Si permanece el enlace simbólico provoca errores al tratar de accederlo (concepto similar a los accesos directos de los sistemas Windows)
      - Se puede enlazar simbólicamente con ficheros y directorios
      - Permiten atravesar sistemas de ficheros que residen en dispositivos físicos diferentes

Directorios

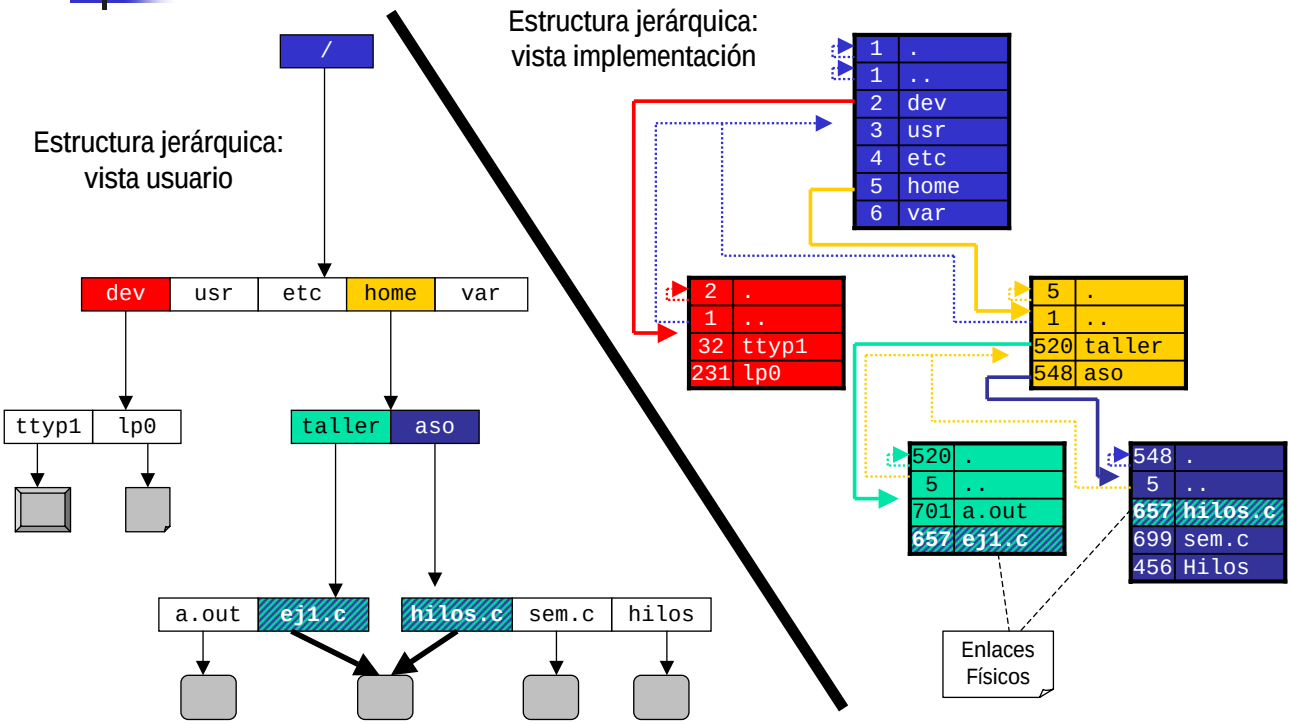
- Tipos de enlaces: ejemplo

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]$
```


Directorios



Directorios: llamadas al sistema

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

Directorios: mkdir/rmdir

- **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)
```

- Descripción
  - **mkdir**: crea el directorio `ruta`
  - Los permisos del nuevo directorio, indicados por el parámetro `modo`, estarán afectados también por la `umask`, tendrá como propietario el UID efectivo del proceso y contendrá, por defecto, las entradas `.` y `..`
  - **rmdir**: elimina el directorio indicado por `ruta`, siempre que esté vacío
- Valor de retorno
  - 0 si no hay error
  - -1 en caso de error

Directorios: opendir, readdir, closedir

- **`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
- Valor de retorno
  - 0 si no hay error
  - -1 en caso de error

Directorios: opendir, readdir, closedir (ejemplo)

```
#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);
}
```

Directorios: link, unlink

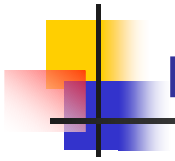
- **`link`**, **`unlink`**: Crea, borrar entradas de directorio

```
#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
- Valores de retorno
  - 0 si funciona con éxito
  - -1 en caso de error

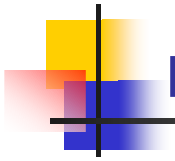
Protección

- Protección: mecanismo para controlar los accesos que los procesos realizan a los recursos del sistema y del resto de usuarios (ficheros, memoria, dispositivos de E/S)
- Protección Unix/Linux
  - Está basada en comparar los atributos del proceso con los atributos del recurso, para determinar si la operación puede efectuarse
  - Atributos de protección de un proceso
    - Identificador de usuario
      - UID real (`rUID`): identificador del usuario que ha creado el proceso
      - UID efectivo (`eUID`): identificador del usuario para el que se ejecuta el proceso
    - Identificador de grupo
      - GID real (`rGID`): identificador de grupo
      - GID efectivo (`eGID`): identificador de grupo efectivo
  - Atributos de protección de un fichero
    - Usuario y grupo propietario: `ownerUID` y `ownerGID`
    - Bits permiso: 9 bits de permiso en grupos de tres: propietario (`u`), grupo (`g`) y otros (`o`) (`rwxr-xr-x` ó `0755`)
      - Para ficheros: lectura (`r`), escritura (`w`) y ejecución (`x`)
      - Para directorios: listar contenido(`r`), crear/eliminar enlaces (`w`), “atravesar” (`x`)
      - Para ficheros especiales: lectura (`r`), escritura (`w`)
    - Bits `SETUID`, `SETGID`



Protección

- Asignación de atributos
  - El fichero (recurso) recibe los atributos del proceso que lo crea, es decir, `ownerUID` ← `UID` y `ownerGID` ← `GID`
  - Un proceso puede cambiar de `UID` o `GID` cuando haga `exec()` sobre un fichero con los bits `SETUID` o `SETGID` activados
    - Si el ejecutable tiene el bit `SETUID` activo, el `eUID` del proceso pasa a ser igual al `ownerUID` del fichero ejecutado
    - Si el ejecutable tiene el bit `SETGID` activo, el `eGID` del proceso pasa a ser igual al `ownerGID` del fichero ejecutado
- Reglas de protección
  - Cuando un proceso intenta realizar un acceso sobre un archivo, se hacen las siguientes comprobaciones
    - Si el proceso tiene `eUID=0` (usuario `root`), no se realiza ninguna comprobación y se admite el acceso
    - Si el proceso tiene `eUID` igual al ownerUID del archivo, se comprueban los permisos de la parte de propietario (`u`)
    - Si no, si el proceso tiene `eGID` igual al del grupo del propietario del archivo, se comprueban los permisos de la parte del grupo (`g`)
    - Si no, se comprueba los permisos para el resto de usuarios (`o`)



Protección

- **`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
- Valor de retorno
  - 0 si funciona con éxito
  - -1 en caso de error