SISTEMAS OPERATIVOS II

TEMA 2

Conceptos de Sistemas Operativos: El caso Unix

Área de Arquitectura y Tecnología de Computadores

Escuela Universitaria Politécnica de Teruel

http://????.????.es/SOII/

Conceptos de S. O.: El caso Unix

- **Bibliografía:**
  - Silberschatz & Galvin: Sistemas Operativos
  - Stallings: Sistemas Operativos
  - Robbins: Unix programación práctica
    - Capítulos 1, 2 y 3
  - Ayuda de Unix: “man”

Conceptos de S. O.: El caso Unix

- **Objetivos**:
  - Recordar y afianzar algunos conceptos clave de los S. O. relacionados con procesos, planificación de la CPU, ficheros y gestión de memoria

Conceptos de S. O.: El caso Unix

- Parte 1: **Procesos**
- Parte 2: **Planificación de la CPU**
- Parte 3: **Ficheros**
- Parte 4: **Gestión de memoria**

Conceptos de S. O.: El caso Unix

- Parte 3: **Ficheros**
  - El interfaz de dispositivos Unix
  - El i-nodo de Unix
  - El sistema de ficheros Unix
  - Tuberías: Pipes y Fifos
  - E/S sin bloqueo

Ficheros: Generalidades

- **Dispositivos: hardware y software**
  - **Def.-** Dispositivo o periférico: “el hardware”
    - pantalla, discos, cintas, teclado, impresora, ratones, interfaz de red, etc.
    - Los programas de usuario controlan las E/S de los periféricos mediante las llamada al sistema
  - **Def.-** Controlador de dispositivo: “el software: device drivers”
    - Ocultan los detalles del funcionamiento del dispositivo y lo protegen del uso no autorizado
    - Forman parte del S. O.

Ficheros: El interfaz de dispositivos Unix

- En Unix todos los dispositivos o periféricos están representados por ficheros “especiales”:
  - **Fichero regular:** fichero de datos ordinario en disco
    - _, d, p
  - **Fichero especial por bloques:** representa un dispositivo con características similares a un disco
    - b
  - **Fichero especial de caracteres:** representa un dispositivo con características similares a un teclado
    - c
  - Los ficheros especiales están localizados en el directorio **/dev**
  - Permite realizar E/S con las mismas llamadas al sistema

Ficheros: El interfaz de dispositivos Unix

- **Llamadas al sistema**
  - open, creat - open and possibly create a file or device

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int creat(const char *pathname, mode_t mode);
```

POSIX.1, Spec 1170

  - close - close a file descriptor

```
#include <unistd.h>
int close(int fd);
```

POSIX.1, Spec 1170

Ficheros: El interfaz de dispositivos Unix

- **Llamadas al sistema**
  - read - read from a file descriptor

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

POSIX.1, Spec 1170

  - write - write to a file descriptor

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

POSIX.1, Spec 1170

  - ioctl - control device

```
#include <sys/ioctl.h>
int ioctl(int d, int request, ...)
```

POSIX.1, Spec 1170

Ficheros: El sistema de ficheros Unix

- **Las particiones y el sistema de ficheros**
  - Cuando se da formato a un disco, se divide en regiones denominadas particiones
  - Cada partición puede tener un sistema de ficheros asociado
  - Un sistema de ficheros está formado por un árbol de directorios:
    - El nodo superior recibe el nombre de raíz

Ficheros: El sistema de ficheros Unix

- Algunos directorios especiales del sistema de ficheros:
  - **/dev** contiene las especificaciones de los dispositivos
  - **/etc** contienen información relacionada con la red, las cuentas de usuario e información específica de la máquina
  - **/home** directorio preestablecido para cada una de las cuentas de usuario
  - **/opt** directorio de instalación de aplicaciones estándar en System VR4
  - **/usr/include** directorio que contiene los ficheros include
  - **/var** ficheros del sistema que cambian y pueden hacerse muy grandes

Ficheros: El sistema de ficheros Unix

- **Directorios y rutas de acceso**
  - Los datos y los programas se organizan en ficheros para que se puedan guardar de manera permanente en disco
  - El S. O. mantiene en los directorios la relación entre el nombre del fichero y su ubicación física en disco
  - El directorio de trabajo asociado con el shell de inicio de sesión se especifica en la variable de entorno HOME
  - Los procesos tienen un directorio asociado, el directorio de trabajo activo, que se emplea para resolver la ruta de acceso
  - Un **.** indica el directorio activo
  - Dos **..** indican el directorio que se encuentra “encima” de él

Ficheros: El sistema de ficheros Unix

- **Directorios y rutas de acceso**
  - En Unix se buscan los ficheros ejecutables en todos los directorios que aparecen en la variable de entorno PATH
    - Cada uno de ellos se utiliza a la vez como directorio de trabajo
    - PATH=.:/usr/bin:/etc:/usr/local/bin:/usr/ccs/bin
  - **Ruta absoluta**
    - Especifica de modo único un fichero en el sistema de ficheros
    - Especifica todos los nodos del árbol de directorios que hay que recorrer desde la raíz hasta el fichero buscado
  - **Ruta relativa**

Ficheros: Rutas de acceso

- **Llamadas al sistema**
  - La función getcwd de la biblioteca de C devuelve la ruta de acceso del directorio de trabajo activo
  - getcwd, get_current_dir_name, getwd - Get current working directory

```
#include <unistd.h>
char *getcwd(char *buf, size_t size);
char *get_current_working_dir_name(void);
char *getwd(char *buf);
```

POSIX.1, Spec 1170

  - **Ejemplo**.-
    - getcwd (mycwd, PATH_MAX)

Ficheros: Directorios

- **Llamadas al sistema**
  - La función **opendir** proporciona un identificador de bloque de directorio para las demás funciones de directorio
  - opendir - open a directory

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
```

POSIX.1, Spec 1170

  - La llamada **closedir** es una llamada al sistema para finalizar el uso del directorio
  - closedir - close a directory

```
#include <sys/types.h>
#include <dirent.h>
int closedir(DIR *dir);
```

POSIX.1, Spec 1170

Ficheros: Directorios

- **Llamadas al sistema**
  - La llamada **readdir** devuelve un puntero a una estructura que contiene información sobre la siguiente entrada del directorio. Devuelve NULL cuando llega al final del directorio
  - readdir - read a directory (En linux es diferente)

```
#include <unistd.h>
#include <dirent.h>
struct dirent *readdir(DIR*dirp);
```

POSIX.1, Spec 1170

  - Nota.- The dirent structure is declared as follows:

```
struct dirent
{
    long d_ino;                 /* inode number */
    off_t d_off;                /* offset to this dirent */
    unsigned short d_reclen;    /* length of this d_name */
    char d_name [NAME_MAX+1];   /* file name (null-terminated) */
}
```

Ficheros: Directorios

- **Llamadas al sistema**
  - La llamada **rewinddir** es una llamada al sistema para situarse al comienzo del directorio
  - rewinddir - reset directory stream

```
#include <sys/types.h>
#include <dirent.h>
void rewinddir(DIR *dir);
```

POSIX.1, Spec 1170

  - La función **seekdir** posiciona la entrada en uso del directorio a la posición especificada por offset
  - seekdir - set the position of the next readdir() call in the directory stream

```
#include <dirent.h>
void seekdir(DIR *dir, off_t offset);
```

Spec 1170

Ficheros: Directorios

- **Llamadas al sistema**
  - La función **telldir** devuelve el desplazamiento dentro del directorio a la entrada en uso
  - telldir - return current location in directory stream

```
#include <dirent.h>
off_t telldir(DIR *dir);
```

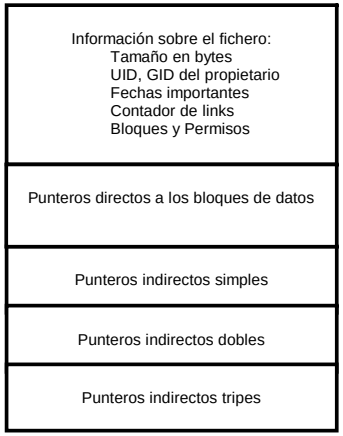
Spec 1170

Ficheros: El i-nodo de Unix

- **Representación de ficheros en Unix**
  - La descripción de un fichero en Unix se guarda en una estructura denominada i-nodo
  - El i-nodo contiene: el tamaño del fichero, su localización, el propietario, la fecha de creación, la fecha de última modificación, los permisos de acceso, etc.
  - Los directorios en Unix también se representan por i-nodos
  - El i-nodo no contiene el nombre del fichero

Ficheros: El i-nodo de Unix

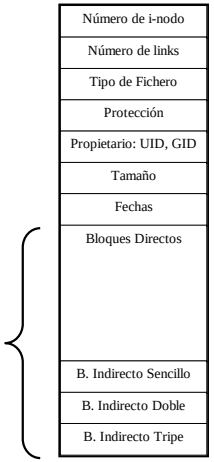
- **Estructura de un fichero en Unix:**



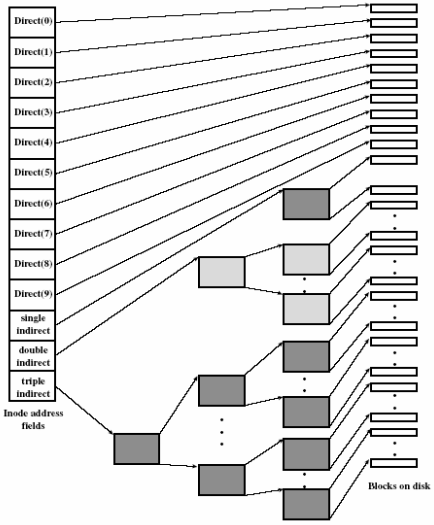
El sistema de ficheros Unix

- El i-nodo de Unix:

Esquema de direccionamiento de bloques



Unix: Esquema de direccionamiento de bloques



Ficheros: El i-nodo de Unix

- **Llamadas al sistema**
  - stat, fstat, lstat - get file status
    - #include <sys/stat.h>
    - #include <unistd.h>
    - int stat(const char *file_name, struct stat *buf);
    - int fstat(int filedes, struct stat *buf);
    - int lstat(const char *file_name, struct stat *buf);

El i-nodo de Unix: Llamadas al sistema

- **Estructura stat**

```
struct stat
{
    dev_t     st_dev;     /* device */
    ino_t     st_ino;     /* inode */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device type (if inode device) */
    off_t     st_size;    /* total size, in bytes */
    unsigned long st_blksize; /* blocksize for filesystem I/O */
    unsigned long st_blocks;  /* number of blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last change */
};
```

Ficheros: El i-nodo de Unix

- **Representación de directorios en Unix**
  - En Unix, un directorio es un fichero que contiene una serie de parejas formadas por el nombre de fichero y el i-nodo al que corresponde
  - Directorio: < nombre, i-nodo >

Ficheros: El i-nodo de Unix

- **Representación de directorios en Unix**
  - Ventajas de la representación de directorios en Unix:
    - El cambio de nombre de un fichero sólo requiere la modificación de la entrada del directorio
    - El fichero puede moverse de un directorio a otro moviendo la entrada del directorio
    - El fichero puede tener varios nombres o residir en varios directorios diferentes y sólo es necesario tener una copia física del fichero en disco
    - Las entradas del directorio son pequeñas, ya que la mayor parte de la información se guarda en el i-nodo
      - Notar que la longitud de las entradas del directorio son variables debido a que el nombre del fichero tiene longitud variable

Ficheros: El i-nodo de Unix

- **Enlaces o links**
  - **Def.-** Un enlace o link es una asociación entre el nombre de un fichero y un i-nodo
  - **Enlaces duros**: enlazan directamente los nombres de los ficheros con los i-nodos
  - **Enlaces suaves**: utilizan el fichero como un puntero a otro nombre de fichero: “enlace simbólico”
  - **Nota.-** Cada i-nodo contiene el número de links asociados, es decir, el número total de entradas de directorio que apuntan a dicho i-nodo

Ficheros: Enlaces o links

- **¿Cómo crear un enlace o link en Unix?**
  - Comando: **ln**
  - Llamada al sistema: **link**
  - link - make a new name for a file
    - #include <unistd.h>
    - int link(const char *oldpath, const char *newpath);

POSIX.1, Spec 1170

Ficheros: Enlaces o links

- **¿Cómo eliminar un enlace o link en Unix?**
  - Comando: **rm**
  - Llamada al sistema: **unlink**
  - unlink - delete a name and possibly the file it refers to
    - #include <unistd.h>
    - int unlink(const char *pathname);

      POSIX.1, Spec 1170

Ficheros: Enlaces o links

- **¿Cómo crear un enlace simbólico en Unix?**
  - Comando: **ln -s**
  - Llamada al sistema: **symlink**
  - symlink - make a symbolic link to a file
    - #include <unistd.h>
    - int symlink(const char *oldpath, const char *newpath);

      POSIX.1, Spec 1170

Ficheros: Programación en C

- **Funciones para manejar ficheros**
  - Funciones de primer nivel: “Llamadas al sistema”
    - Descriptores de fichero: “canales”
    - Fichero = Conjunto de bytes
  - Funciones de segundo nivel: “Funciones de librería”
    - Punteros a la estructura FILE
    - Fichero = Conjunto de registros
  - Los descriptores de fichero y los punteros a la estructura FILE son “entes” lógicos que nos permitir realizar operaciones de E/S independientes del disposivo utilizado

Ficheros: Programación en C

- **Funciones de primer nivel: “Llamadas al sistema”**
  - **Ejemplos:**
    - open
    - close
    - read
    - write
  - Descriptores de ficheros abiertos por defecto:

    #include <unistd.h>

    STDIN_FILENO = 0

    STDOUT_FILENO = 1

    STDERR_FILENO = 2

Ficheros: Programación en C

- **Funciones de segundo nivel: “Funciones de librería”**
  - **Ejemplos:**
    - fopen, fclose
    - fread, fwrite
    - gets, puts; scanf, printf; fscanf, fprintf
    - fflush
  - Punteros a la estructura file creados por defecto:

```
#include <stdio.h>
    stdin   ≈   0
    stdout  ≈   1
    stderr  ≈   2
```

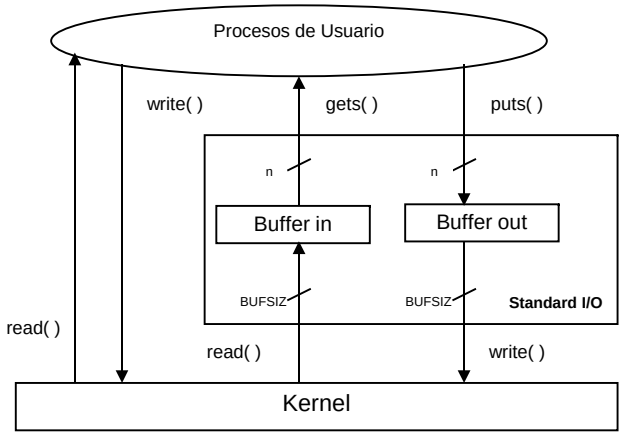

Ficheros: Programación en C

- **Punteros a la estructura FILE:** Funciones de E/S ANSI-C

```
typedef struct
{
  short leve;
  unsigned flags;
  char fd;                          /* descriptor de fichero */
  unsigned char hold;
  short bsize;
  unsigned char *buffer, *curp;     /* buffer de E/S */
  unsigned istemp;
  short token;
} FILE;
```


El sistema de ficheros Unix*

- Llamadas al sistema vs Funciones estándar de E/S



El sistema de ficheros Unix

- Llamadas al sistema vs Funciones estándar de E/S
  - Las funciones de librería agrupan las E/S, por lo tanto, se realizan menos llamadas al sistema
  - Llamada al sistema setbuf:
    - Se puede modificar el comportamiento de las funciones de librería:
      - Fully Buffered
      - Line Buffered
      - Unbuffered
    - Se puede modificar el tamaño de los buffers de E/S

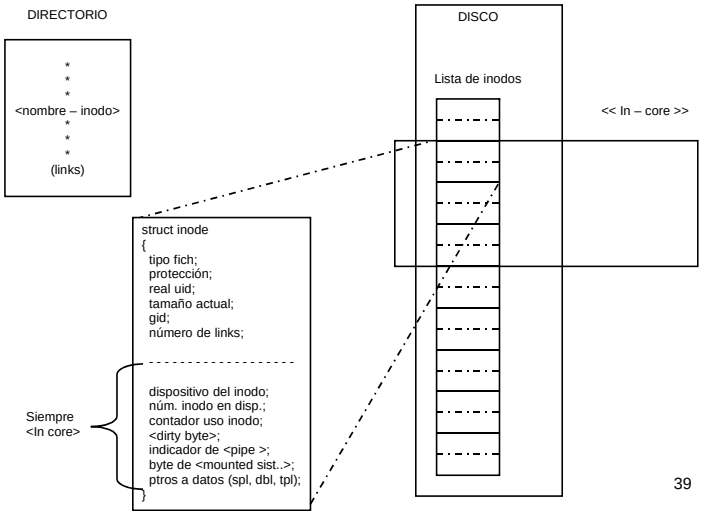
El sistema de ficheros Unix

- Estructura de un disco Unix
  - 1º Bloque de arranque
  - 2º Superbloque
    - Tamaño del sistema de ficheros
    - Número de bloques libres
    - Lista de los bloques libres
    - Número de i-nodos libres
    - Lista de i-nodos libres
    - Lista de bloques defectuosos
  - 3º Mapa de bits de i-nodos
  - 4º Mapa de bits de bloques
  - 5º i-nodos
  - 6º Bloques de datos

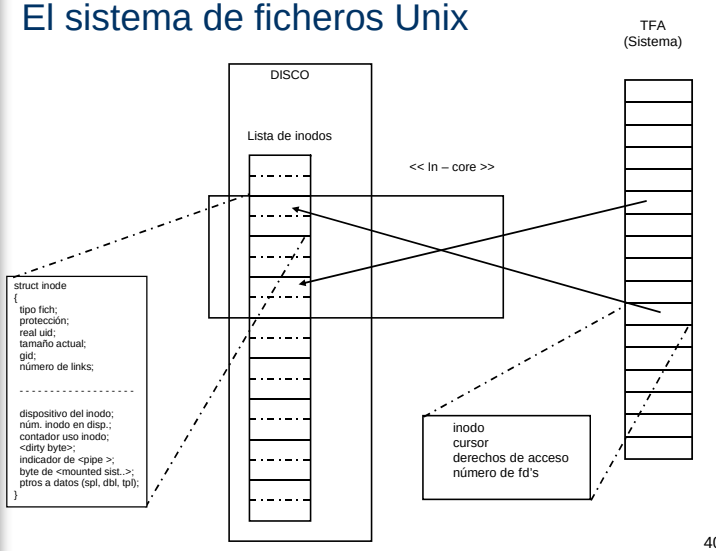
El sistema de ficheros Unix

- Principales estructuras de control
  - Superbloque
  - I-nodo en disco (ino.h)
  - I-nodo en memoria (inode.h)
    - Información del i-nodo en disco
    - Número de aperturas
    - Campos especiales para tuberías (pipes)
  - TDF: Tabla de Descriptores de Fichero
  - TFA: Tabla de Ficheros Abiertos
  - Tabla de i-nodos en memoria

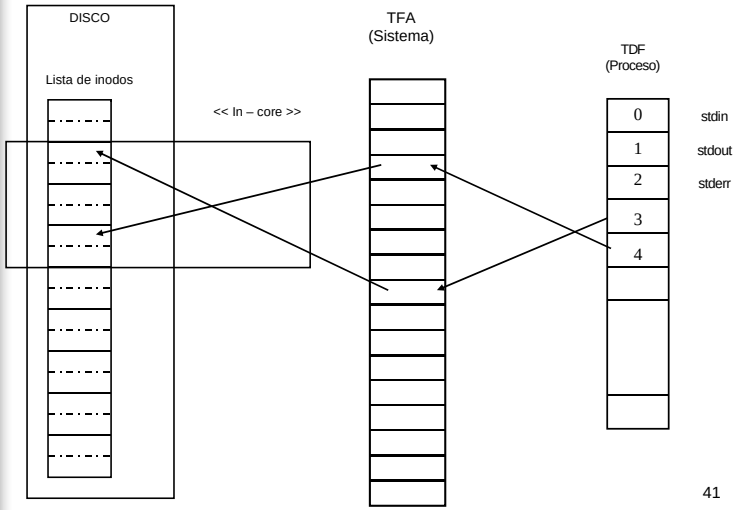
El sistema de ficheros Unix



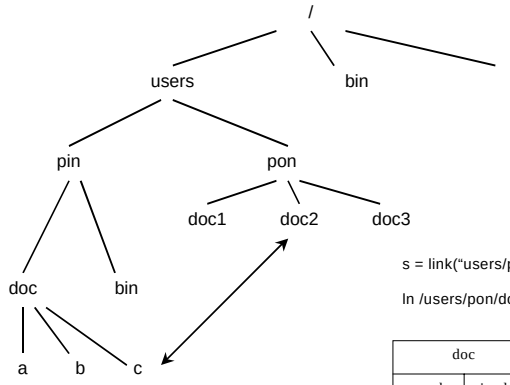
El sistema de ficheros Unix



El sistema de ficheros Unix



El sistema de ficheros Unix



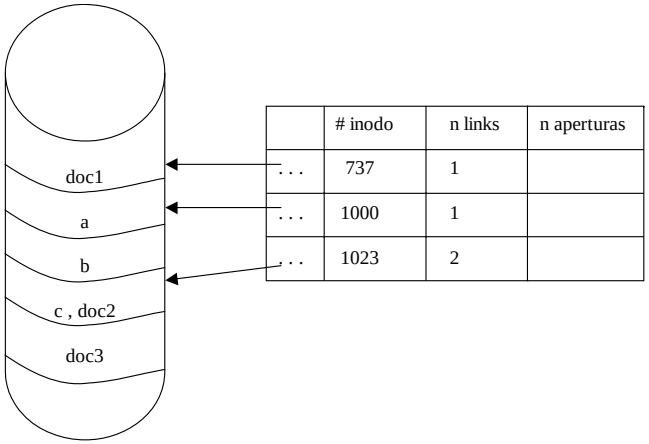
s = link("users/pon/doc2", "doc/c");

ln /users/pon/doc2 doc/c

doc	
nombre	inodo
a	1000
b	1015
c	1023

pon	
nombre	inodo
doc1	737
doc2	1023
doc3	1090

El sistema de ficheros Unix

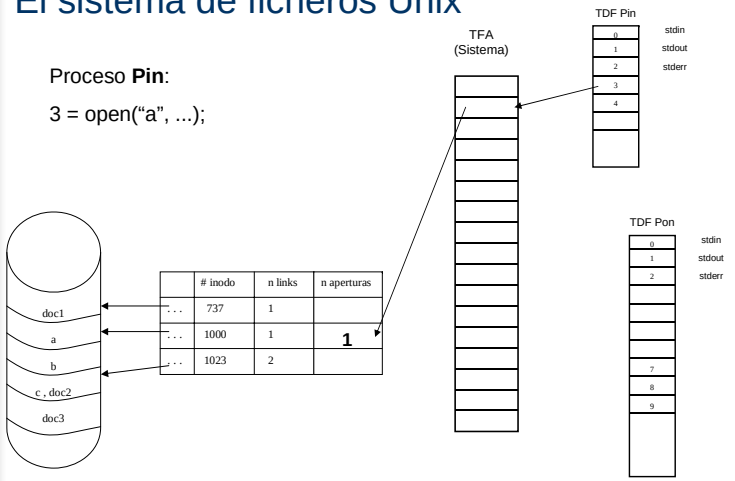


	# inodo	n links	n aperturas
. . .	737	1	
. . .	1000	1	
. . .	1023	2	

El sistema de ficheros Unix

Proceso **Pin**:

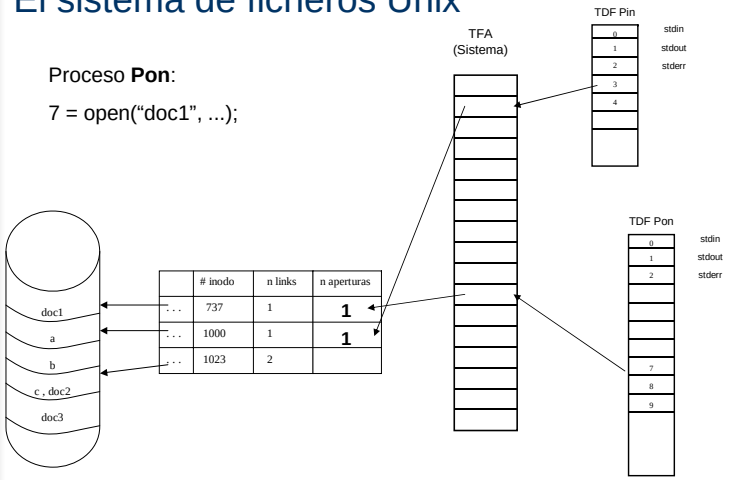
3 = open("a", ...);



El sistema de ficheros Unix

Proceso **Pon**:

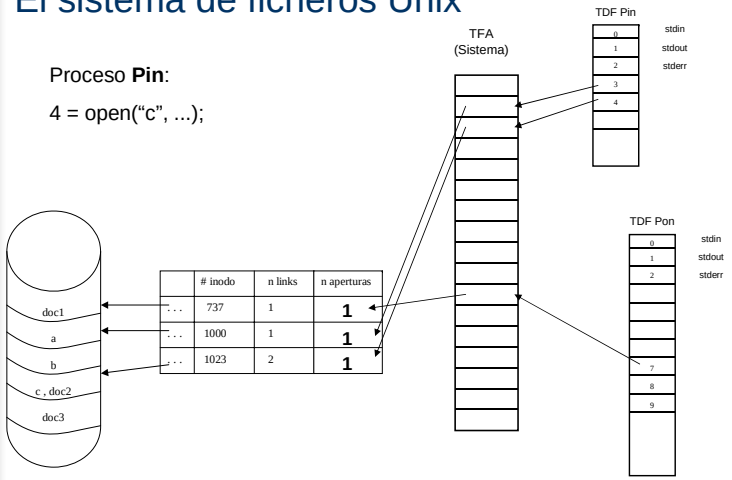
7 = open(“doc1”, ...);



El sistema de ficheros Unix

Proceso **Pin**:

4 = open(“c”, ...);

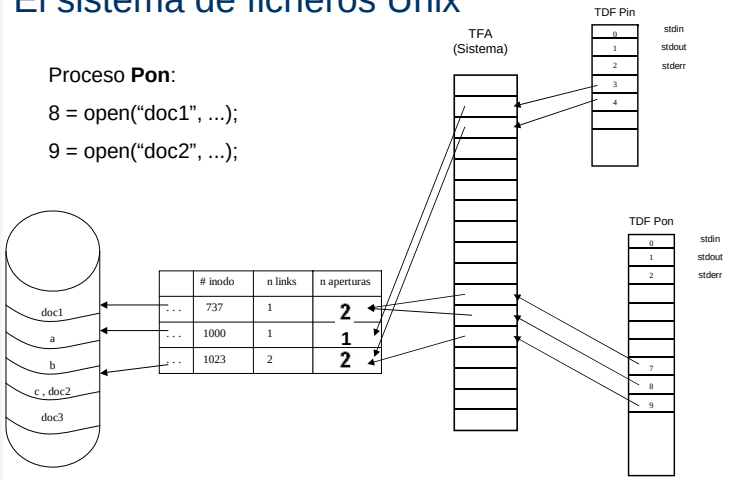


El sistema de ficheros Unix

Proceso **Pon**:

8 = open(“doc1”, ...);

9 = open(“doc2”, ...);



Ficheros: Programación en C

- **¿Qué pasa cuando un programa emplea buffers de E/S y termina anómalamente?**
  - “La finalización de un función de librería de de E/S (fread, fwrite) no garantiza que los datos ya estén en disco o en memoria”
  - El S. O. antes de dar la orden de E/S copia los datos en una “cache buffer” de sistema
  - Posible solución:
    - deshabilitar el empleo del buffer con la función de C: **setbuf( )**
  - setbuf, setvbuf - assign buffering to a stream

```
#include <stdio.h>
void setbuf(FILE *stream, char *buf);
int setvbuf(FILE *stream, char *buf, int type, size_t size);
```

Ficheros: Redireccionamientos y Filtros

- **Redireccionamientos**
  - Se puede modificar tanto la entrada como la salida estándar modificando la tabla de descriptores de fichero
  - Ejemplos:
    - ">" muchos intérpretes entienden este símbolo como un rediccionamiento de la salida estándar
    - "<" muchos intérpretes entienden este símbolo como un redireccionamiento de la entrada estándar
  - Los proceso hijos heredan los redireccionamientos ya que heredan la tabla de descriptores de ficheros del padre.

Ficheros: Redireccionamientos y Filtros

- **Filtros**
  - **Def.-** Un filtro es un programa que lee de la entrada estándar, realiza una transformación de la lectura y escribe el resultado en la salida estándar
  - Ejemplos:
    - cat, sort, head, tail, move, grep, awk

Ficheros: Pipes y Fifos

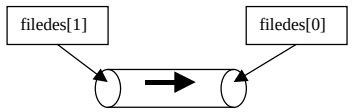
- **Pipes**
  - Las pipes, o "unnamed pipes" sulen representar una conexión entre la salida estándar y la entrada estándar a través de un buffer de comunicación
  - No tienen nombre
  - Son unidireccionales: un proceso lee y otro escribe
  - No están representadas en el Sistema de Ficheros mediante un nombre
  - Sólo se puede acceder a ella a través de los descriptores de fichero asociados
  - Puede ser utilizada únicamente por procesos que tengan parentesco

Ficheros: Pipes

- **Llamadas al sistema**
  - La llamada **pipe( )** crea un buffer de comunicación
  - pipe - create pipe

```
#include <unistd.h>
int pipe(int filedes[2]);
```

POSIX.1, Spec 1170



Ficheros: Pipes y Fifos

- **Fifos**
  - Las Fifos, o “named pipes” suelen representar un buffer de comunicación compartido.
  - Están representadas por ficheros especiales y persisten aun después de que todos los procesos las hayan cerrado
  - Cualquier proceso que tenga los permisos apropiados puede tener acceso a una fifo
  - No es necesario que los procesos que comunican estén relacionados, sólo deben compartir el sistema de ficheros
  - Creación de Fifos:
    - Comandos **mknod**, **mkfifo**
    - Función de C **mkfifo( )**
    - Llamada al sistema **mknod( )**

Ficheros: Fifos

- **Creación de fifos**
  - Llamada al sistema **mknod**
  - mknod - make a directory, or a special or ordinary file

```
#include <sys/stat.h>
int mknod(const char *path, mode_t mode, dev_t dev);
```

S_IFIFO fifo special

POSIX.1, Spec 1170

  - Función estándar de la librería de C **mkfifo**
  - mkfifo - create a new FIFO special file (a named pipe)

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo ( const char *path, mode_t mode );
```

ANSI C

Ficheros: Pipes y Fifos

- **Llamadas al sistema**
  - dup2 cierra el descriptor de fichero asociado al segundo parámetro y le asigna el descriptor de fichero del primer parámetro
  - dup, dup2 - duplicate a file descriptor

```
#include <unistd.h>
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

POSIX.1, Spec 1170

Ficheros: Lectura y escritura

- **read y write en ficheros**
  - Unix proporciona acceso secuencial a los ficheros a través de las llamadas al sistema **read** y **write**
  - read - read from a file descriptor

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

POSIX.1, Spec 1170

  - Valores devueltos:
    - número de bytes leidos
    - 0 si es fin de fichero
    - -1 en caso de error
      - errno = EINTR read ha sido interrumpida por una señal que se va a capturar

Ficheros: Lectura y escritura

- **read y write en ficheros**
  - Unix proporciona acceso secuencial a los ficheros a través de las llamadas al sistema **read** y **write**
  - write - write to a file descriptor

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

POSIX.1, Spec 1170

  - Valores devueltos:
    - número de bytes escritos
    - -1 en caso de error
      - errno = EINTR write ha sido interrumpida por una señal que se va a capturar

Ficheros: Lectura y escritura

- **Read y write en pipes**
  - **read en pipes**:
    - Si la tubería está vacía ⇒ bloqueo
    - Si la tubería contiene datos ⇒ lee los datos
    - Fin de fichero en pipe: La tubería está vacía y no hay procesos con descriptores de escritura abiertos para esa tubería
  - **write en pipes**:
    - Si la tubería está llena ⇒ bloqueo
    - Si la tubería no está llena ⇒ escribe los datos
    - Si el descriptor de lectura está cerrado:
      - El kernel envía la seña SIGPIPE al proceso que intenta escribir.
      - Nota: No se permite la escritura en una tubería sin descriptor de lectura

Ficheros: E/S sin bloqueo

- Cuando un proceso intenta leer en un buffer de comunicaciones, se bloquea hasta que la entrada esta disponible
- Problema:
  - ¿Que ocurre, si un proceso debe vigilar más de un buffer de comunicaciones de entrada?
  - El proceso se bloqueará en cuanto encuentre un buffer vacío
- Solución:
  - Utilizar E/S sin bloqueo

Ficheros: E/S sin bloqueo

- Idea:
  - Un proceso puede ejecutar un read( ) y si no hay datos de entrada disponibles, regresar de inmediato
    - Utilizar la bandera **O_NDELAY** y **O_NONBLOCK** asociada al descriptor de fichero
  - El proceso intentará continuamente leer de los descriptores de entrada, uno detrás de otro:
    - Método denominado de escrutinio o polling
- Inconveniente:
  - La CPU se utiliza de una manera ineficiente: **“Espera activa”**

Ficheros: E/S sin bloqueo

- La llamada al sistema **fcntl** modifica la bandera y el estado asociadas a un descriptor de fichero
  - fcntl - manipulate file descriptor

```
#include <unistd.h>
#include <fcntl.h>
int fcntl(int fd, int cmd);
int fcntl(int fd, int cmd, long arg);
int fcntl(int fd, int cmd, struct flock * lock);
```

POSIX.1, Spec 1170

Ficheros: E/S sin bloqueo

- **Llamada al sistema select**
  - Permite vigilar, sin bloqueo, varios descriptores de fichero para lectura, escritura y existencia de una condición de excepción
  - select, FD_CLR, FD_ISSET, FD_SET, FD_ZERO – synchronous I/O multiplexing

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
    struct timeval *timeout);
```

  - Valores devueltos:
    - Número de descriptores listos
    - –1 en caso de error
      - errno = EINTR — select ha sido interrumpida por una señal que se va a capturar

Ficheros: E/S sin bloqueo

- **Llamada al sistema select**
  - Utiliza campos de bits para representar los descriptores de ficheros
  - Macros para poder manejar los campos de bits asociados a los descriptores de ficheros que representan:
  - select, FD_CLR, FD_ISSET, FD_SET, FD_ZERO – synchronous I/O multiplexing #include <sys/time.h>

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
FD_CLR(int fd, fd_set *fdset);
FD_ISSET(int fd, fd_set *fdset);
FD_SET(int fd, fd_set *fdset);
FD_ZERO(fd_set *fdset);
```

Ficheros: E/S sin bloqueo

- **Llamada al sistema select**
  - Utilización de los macros antes de ejecutar select:
    - FD_SET añade un bit en fdset que estará asociado al descriptor de fichero que se pasa como primer argumento
    - FD_CLR elimina el bit en fdset que está asociado al descriptor de fichero que se pasa como primer argumento
    - FD_ZERO borra todos los bits en fdset
  - Utilización de los macros después de ejecutar select:
    - FD_ISSET permite determinar si un descriptor de fichero ha sido actualizado en la máscara de bits fdset
  - **Ejercicio**: Probar el funcionamiento de select