Sistemas Operativos I

# Tema 2

# Introducción a UNIX

Equipo de Sistemas Operativos DISCA / DSIC

UPV

# Objetivos

1.- Presentar UNIX como un ejemplo de sistema operativo sobre el que poder aplicar los conceptos vistos en el tema anterior y sobre el que podrán aplicarse los conocimientos a adquirir en los temas siguientes.

2.- Dar a conocer las abstracciones principales sobre las que trabaja este sistema: proceso y fichero.

3.- Comentar las interfaces que se pueden encontrar en un sistema UNIX.

4.- Presentar sus llamadas más importantes.

5.- Dar una descripción genérica de su intérprete de mandatos.

# Introducción a Unix

## Contenido

1.- Conceptos básicos

- Ficheros
- Procesos
- Protección
- Señales

2.- Servicios

- Llamadas al sistema
- Programas del sistema

3.- El intérprete de órdenes

## Bibliografía


- *A. Silberschatz, P. Galvin.*

  Sistemas Operativos. 5ª ed. Capítulo 21.

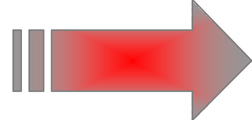
- *A. Tanenbaum*

  Modern Operating Systems.

- *Morgan.*

  Introducción al Unix sistema V

# Contenidos



## 1.- Conceptos básicos

- ✍Ficheros
- ✍Procesos
- ✍Protección
- ✍Señales

2.- Servicios

- ✍Llamadas al sistema
- ✍Programas del sistema

3.- El intérprete de órdenes

# 1.- Conceptos básicos

## ¿Qué es Unix?

- Unix es un sistema operativo multiusuario y de tiempo compartido muy popular.
  - Su ámbito se extiende desde los computadores personales hasta los grandes sistemas
  - La primera versión fue escrita por Ken Thompson en los laboratorios Bell a finales de los 60 como reacción a MULTICS. Versión monousuario, en lenguaje ensamblador para un PDP-7. Posteriormente se reescribió en C y se adaptó a la familia de computadores PDP-11. El código fuente era público y alcanzó una gran difusión entre la comunidad científica.

# 1.- Conceptos básicos

## ✍ Versiones:

✍ Dos líneas comerciales: System V (desarrollada por AT&T) y BSD (Berkeley Software Distribution).

✍ Intentos de estandarización:

- ✍ POSIX (Portable Operating System Interface)
- ✍ OSF (Open Software Foundation)
- ✍ UI (Unix International)

# 1.- Conceptos básicos

## ✍ Características:

- ✍ **Portabilidad:** al estar escrito en C con cuidadoso aislamiento de las rutinas dependientes del hardware.
- ✍ **Modular:** se concibe de manera que estimule la descomposición de una tarea grande en módulos más pequeños, con funciones bien definidas, fáciles de poner a punto e integrar.
- ✍ **Sistema multiusuario y multitarea:** proporciona protección de los datos privados sobre ficheros y del entorno de ejecución.
- ✍ **Sistema de ficheros jerárquico:** permite la unión de diversos sistemas de ficheros con el sistema principal, y una separación de directorios.

# 1.- Conceptos básicos

## ✍ *Características (ii):*

- ✍ **Independencia de dispositivo:** las entradas/salidas están integradas en el sistema de ficheros. Los ficheros y los dispositivos de E/S son tratados de una manera uniforme con el mismo conjunto aplicable de llamadas al sistema.

- ✍ **Interfaz con el usuario simple e interactiva:** el intérprete de órdenes (shell) consiste en un programa independiente que el usuario puede sustituir. La sintaxis de utilización es idéntica para todas las órdenes.

- ✍ **Proporciona un completo entorno de programación:** los filtros son utilidades simples que se concentran en realizar bien una sola función. Pueden combinarse de forma muy flexible utilizando los tubos y las redirecciones de E/S según las necesidades y preferencias de cada usuario.

# 1.- Conceptos básicos

✍ ***Características (iii):***

- ✍**Mantenimiento y evolución fáciles**: consecuencia directa de la modularidad. El sistema sigue evolucionando y se perfecciona y enriquece con nuevas funcionalidades.

- ✍**Carácter abierto**: permite ampliar fácilmente la funcionalidad con nuevos componentes sin tener que depender de un único fabricante.

# Contenidos

1.- Conceptos básicos

- **Ficheros**
- Procesos
- Protección
- Señales

2.- Servicios

- Llamadas al sistema
- Programas del sistema

3.- El intérprete de órdenes

# 1.1.- Ficheros

## ✍ Concepto

Abstracción del espacio de almacenamiento secundario

## ✍ Tipos de fichero

- ✍ Regular
  - ✍ Representa un fichero convencional de datos (programa, texto, ...)
- ✍ Directorio
  - ✍ Utilizados por Unix para asociar nombres a los ficheros
- ✍ Especial
  - ✍ Representa un dispositivo del sistema

# 1.1- Ficheros

## ✍ Atributos de ficheros

Se mantienen en un nodo-i, en un área reservada del disco.

- ✍ Tipo de fichero
- ✍ Propietario (owner UID)
- ✍ Grupo propietario (owner GID)
- ✍ Permisos de acceso (permission bits)
- ✍ Número de enlaces
- ✍ Instantes de creación, último acceso y última modificación
- ✍ Tamaño

# 1.1.- Ficheros

## Visualización de atributos de ficheros

```
$ ls -la
drwxr-xr-x   3 aespinos   gti         1024 Oct  7 10:02 .
drwx------   7 aespinos   gti         1024 Oct  7 10:02 ..
-rw-r--r--   1 aespinos   gti         1401 Oct  7 10:03 notas
drwxr-xr-x   2 aespinos   gti           24 Oct  7 10:02 tests
```

# 1.1.- Ficheros

## Implementación de ficheros

Bloques del disco

Área de nodos-i

Nodo-i

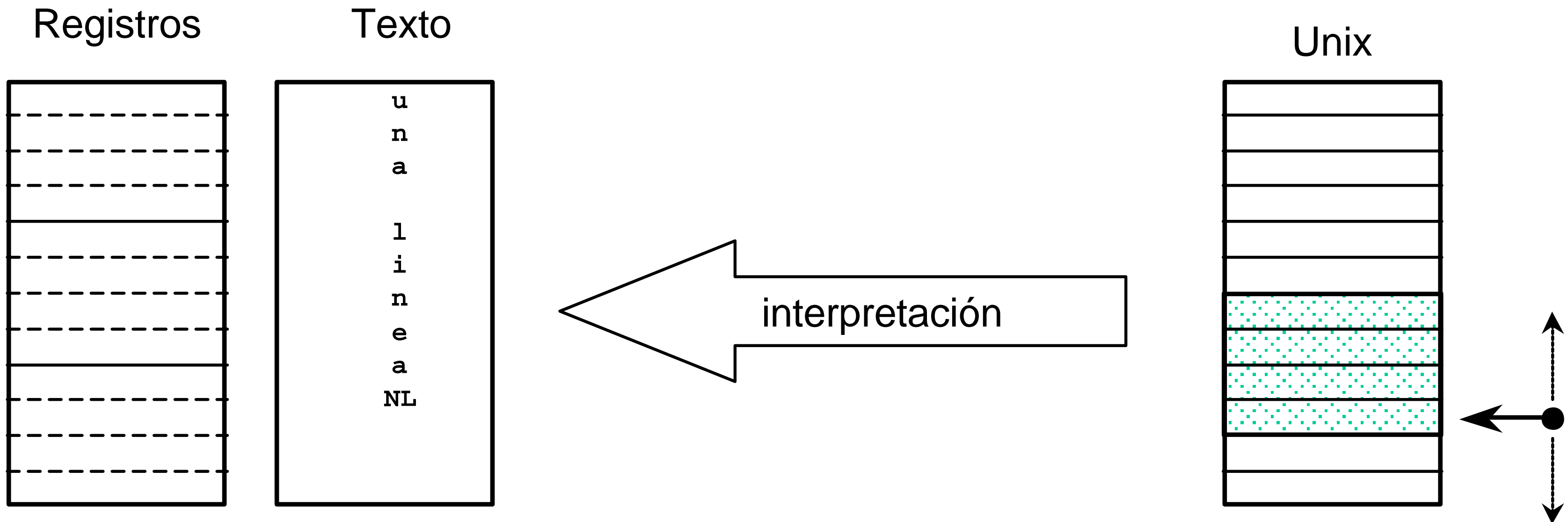
Atributos del Fichero

Asignación de espacio

# 1.1.1.- Ficheros regulares

- **Estructura:** vector lineal de bytes

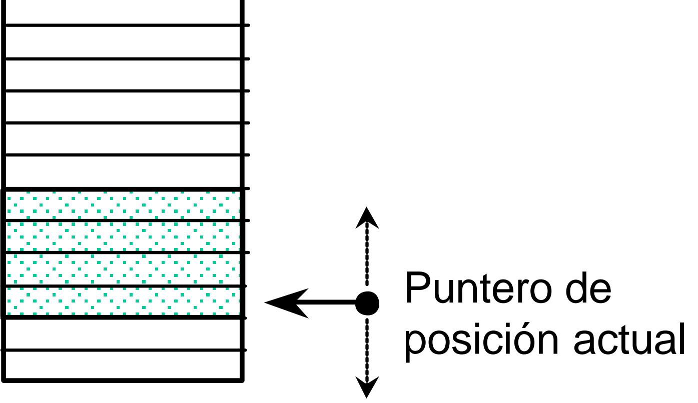
  - Las aplicaciones interpretan esta estructura formando otras estructuras más complejas



# 1.1.1.- Ficheros regulares

## Métodos de acceso

- **Secuencial**: Lecturas y escrituras de un número variable de bytes a partir del *puntero de posición actual*.
- **Directo**: Posibilidad de desplazar el *puntero de posición actual*




## Concurrencia

- Sin control de concurrencia. Las aplicaciones deben sincronizar su acceso a los ficheros

# 1.1.2.- Directorios

- **Concepto**
  - Es un tipo de fichero que permite organizar los ficheros jerárquicamente.
  - Establece la forma de nombrar los ficheros
- **Esquema de nombrado de ficheros**
  - Un fichero se identifica mediante su *vía o ruta de acceso*
  - Una vía de acceso es *absoluta* si comienza por /
    - `/home/so1/so101/.profile`
    - `/home/so2`

# 1.1.2.- Directorios

**/home/so1/so101/.profile**

# 1.1.2.- Directorios

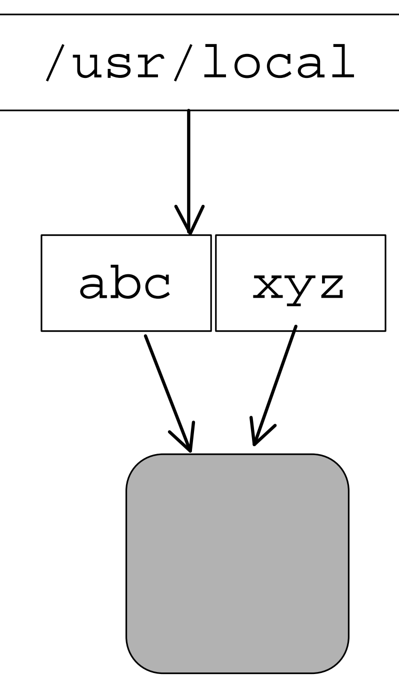
- Si no comienza por / se considera *relativa* al directorio de trabajo actual
  - **`so101/.profile`**, asumiendo **`/home/so1`** como el directorio actual
  - **`.profile`**, asumiendo **`/home/so1/so101`** como el directorio actual
- Las entradas . y .. pueden utilizarse para formar vías de acceso
  - Las siguientes vías de acceso son equivalentes, asumiendo **`/home/so1/so101`** como el directorio actual

    `../so102/ejemplo1`    `./../so102/ejemplo1`
- Los ficheros pueden tener más de un nombre.Cada uno de los nombres de un fichero se denomina un enlace. Existen dos tipos de enlace.

# 1.1.2.- Directorios

## Enlaces físicos

- En este caso todos los nombres hacen referencia a un mismo nodo-i.
- El fichero sólo se elimina del disco cuando se borran todos los enlaces (todas las entradas de directorio que lo referencian).
- Sólo se permite (salvo al administrador) enlazar ficheros regulares (no directorios).

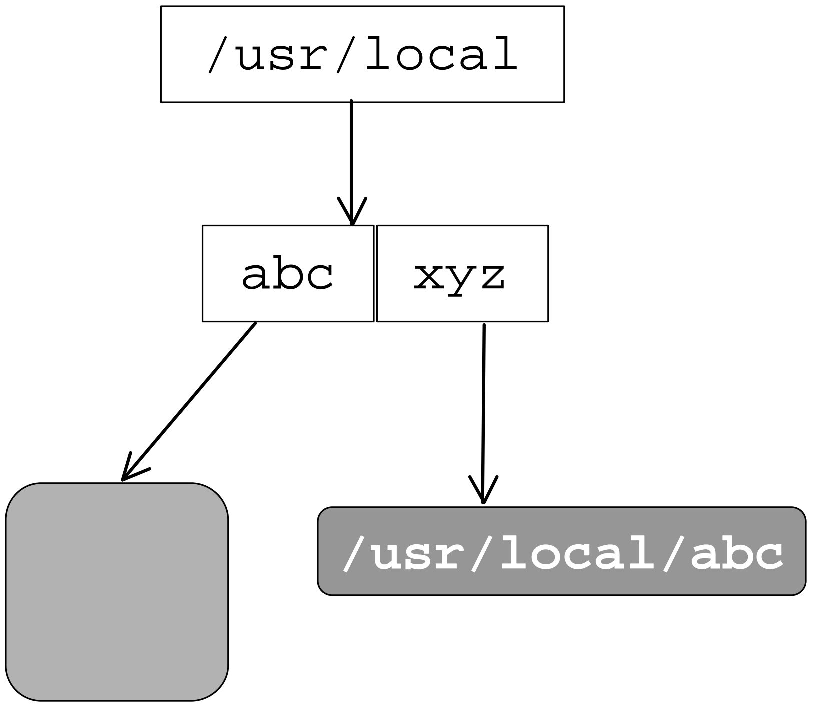


# 1.1.2.- Directorios

## ✍ Tipos de enlace (ii):

### Enlaces simbólicos

- El fichero se elimina cuando se borra el enlace físico. Si permanece el enlace simbólico provoca errores al tratar de accederlo.
- Se puede hacer con ficheros y directorios, existe la posibilidad de ciclos.
- Permiten atravesar sistemas de ficheros que residen en dispositivos físicos distintos.



# 1.1.2.- Directorios

## Visualización de enlaces físicos y simbólicos

- `$ ln notas actas` (físico)
- `$ ln -s notas notassim` (simbólico)

```
$ ls -li
210000 -rw-r--r--   2 agarcia  gti     19 Jun 19 11:33 actas
458390 drwxr-x---   6 agarcia  gti   1024 Nov 26  1997 ada
 57609 drwxr-x---   6 agarcia  gti   1024 Nov 14  1997 concu96
210000 -rw-r--r--   2 agarcia  gti     19 Jun 19 11:33 notas
210001 lrwxrwxrwx   1 agarcia  gti      5 Jun 19 11:52 notassim -> notas
210004 -rw-rw-rw-   1 agarcia  gti    776 Jun 19 12:06 p
657927 drwxr-x---   4 agarcia  gti   1024 Oct 24  1997 planif
 57608 drwxr-x---   2 agarcia  gti   1024 Nov 14  1997 practicas
```

nº nodo-i

# 1.1.2.- Directorios

## Estructura jerárquica. Punto de vista del usuario

/

dev | usr | home | etc | var

ttyp1 | lp0 | dsk010g

so1 | so2

so101 | so102 | so103

.profile | practica1 | problema2

ejemplo2 | ejemplo1 | .profile | seg

a1 | b2

```
$ln /home/so1/so101/problema2 /home/so1/so102/ejemplo2
```

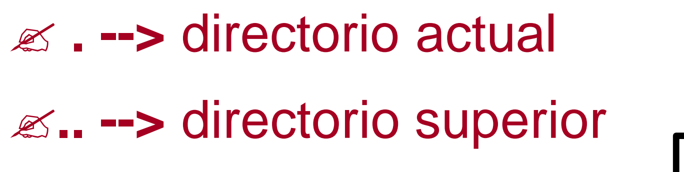
# 1.1.2.- Directorios

## Implementación de directorios

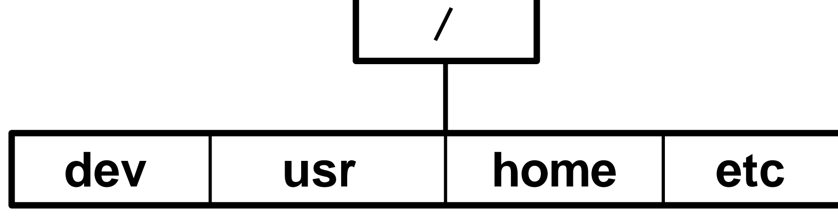
- La estructura básica es la del fichero regular
- Unix interpreta el fichero como si se encontrase organizado en registros
- A cada registro se le denomina una *entrada de directorio*
- Una entrada consta de dos campos
  - número de nodo-i
  - nombre del fichero

| | |
|---|---|
| 1 | . |
| 1 | .. |
| 4 | dev |
| 3 | usr |
| 9 | home |
| 456 | etc |

- Siempre existen al menos dos entradas:



  - **. -->** directorio actual
  - **.. -->** directorio superior



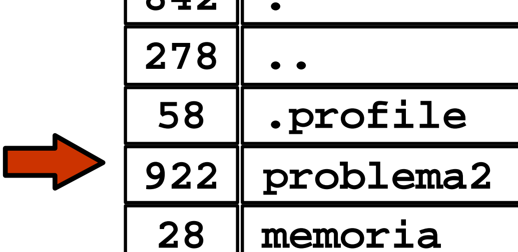
# 1.1.2.- Directorios

## Estructura jerárquica. Punto de vista de Unix

| 1 | . |
|---|---|
| 1 | .. |
| 2 | dev |
| 3 | usr |
| 4 | home |
| 5 | etc |
| 6 | var |

| 278 | . |
|---|---|
| 4 | .. |
| 842 | so101 |
| 583 | so102 |
| 345 | so103 |

| 2 | . |
|---|---|
| 1 | .. |
| 34 | ttyp1 |
| 108 | lp0 |
| 13 | dsk010g |



| 842 | . |
|---|---|
| 278 | .. |
| 58 | .profile |
| 922 | problema2 |
| 28 | memoria |

| 4 | . |
|---|---|
| 1 | .. |
| 278 | so1 |
| 987 | so2 |

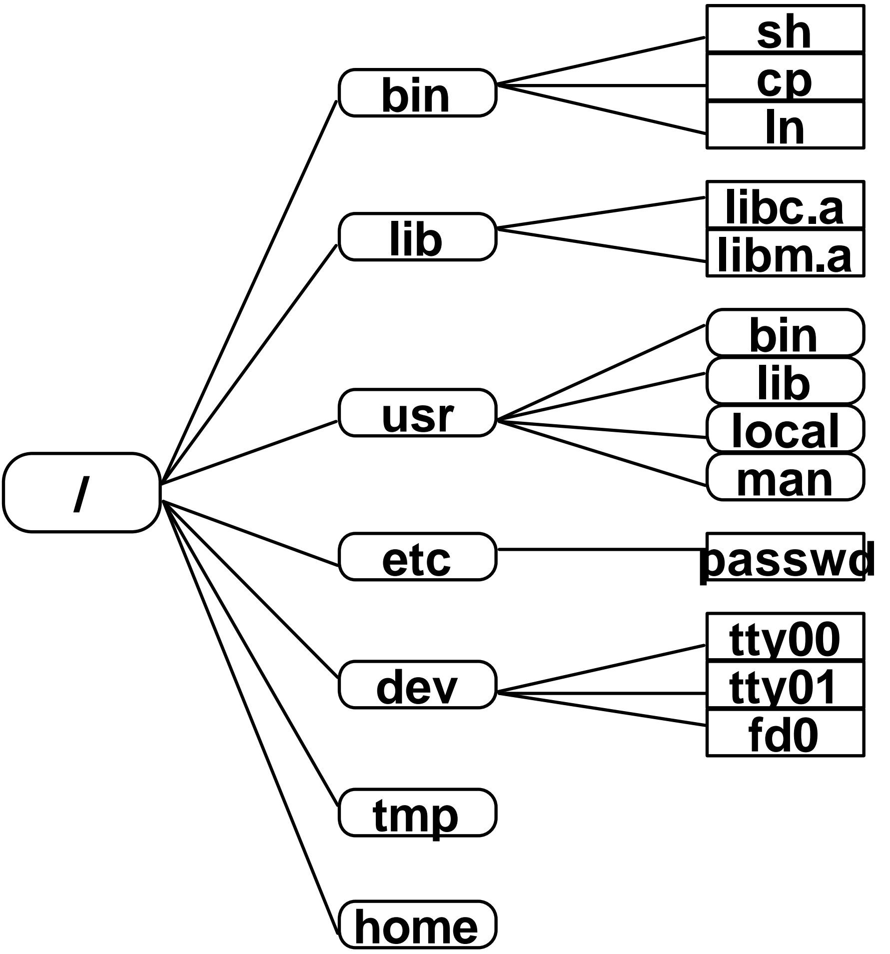
| 987 | . |
|---|---|
| 4 | .. |

| 583 | . |
|---|---|
| 278 | .. |
| 39 | .profile |
| 408 | ejemplo1 |
| 922 | ejemplo2 |
| 93 | seg |

**Enlace físico**

# 1.1.2.- Directorios

## Estructura de directorios típica en Unix




ejecutables básicos

bibliotecas de lenguajes

mandatos de usuario
otras bibliotecas
software local
manuales
ficheros de administración

ficheros especiales
uno por cada dispositivo de E/S

ficheros temporales

directorios de usuarios

# 1.1.3.- Ficheros especiales

## Representan los dispositivos de entrada-salida

- Se encuentran en el directorio `/dev`
- La interfaz para los dispositivos es idéntica a la de los ficheros normales
- Un dispositivo se considera una secuencia de bytes
- Se distinguen dos tipos de ficheros especiales
  - bloques (discos duros, flexibles, CD-ROM, discos RAM, particiones)

    `/dev/fd0, /dev/hda`
  - caracteres (líneas serie, interfaces LAN, terminales, cintas, impresoras)

    `/dev/tty00, /dev/lp0`

# 1.1.3.- Ficheros especiales

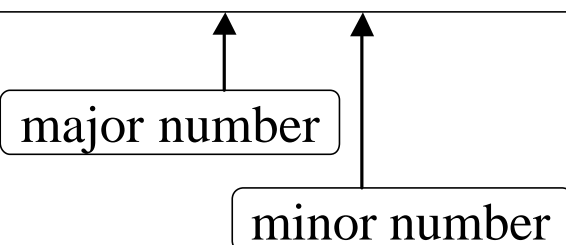
## Representan los dispositivos de entrada-salida

- Los ficheros especiales de bloques son direccionables. Se puede leer o escribir a partir de una cierta posición. Los ficheros especiales de caracteres sólo admiten el acceso secuencial
- Se caracterizan por dos atributos especiales
  - *major number :* identificativo del manejador de dispositivo asociado. Representa el tipo del dispositivo
  - *minor number* : identifica una unidad del tipo de dispositivo. (Disco duro D, terminal 34)

# 1.1.3.- Ficheros especiales

## ✍ Visualización de atributos de ficheros especiales

```
$ ls -l /dev
brw-r-----   1 root     disk        3,   1 May 28  1997 hda1
brw-r-----   1 root     disk        3,   2 May 28  1997 hda2
brw-r-----   1 root     disk        3,   3 May 28  1997 hda3
brw-r-----   1 root     floppy      2,   0 May 28  1997 fd0
brw-r-----   1 root     floppy      2,   1 May 28  1997 fd1
crw-------   1 root     root        4,   0 Jun  9 07:09 tty0
crw-------   1 root     root        4,   1 Jun 18 15:01 tty1
crw-------   1 root     root        4,   2 Jun 10 11:36 tty2
crw-r-----   1 root     lp          6,   0 May 28  1997 lp0
crw-r-----   1 root     lp          6,   1 May 28  1997 lp1
brw-r-----   1 root     disk        1,   0 May 28  1997 ram0
brw-r-----   1 root     disk        1,   1 May 28  1997 ram1
crw-rw-rw-   1 root     sys         1,   3 May 28  1997 null
```

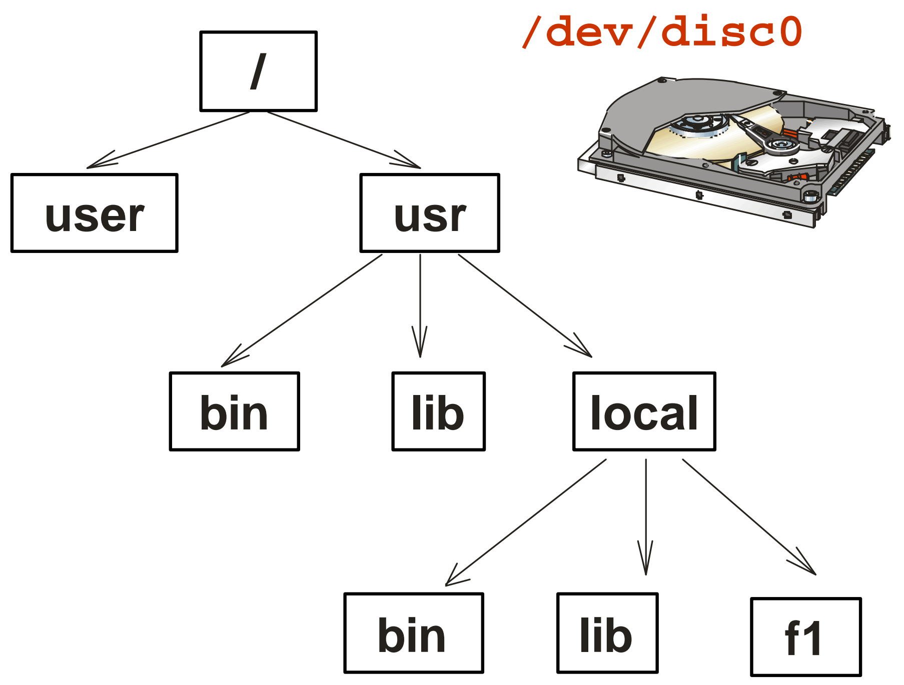


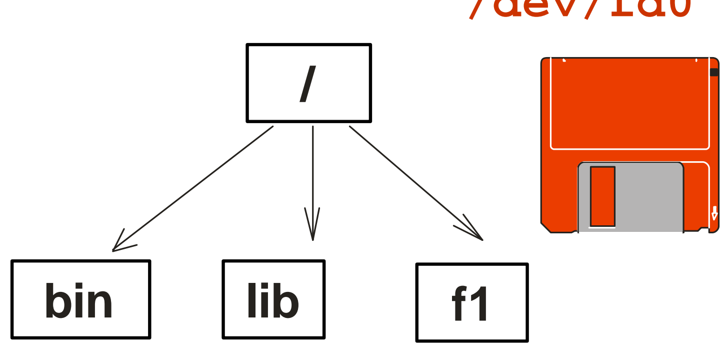
# 1.1.4.- Montaje de directorios

- **Permite añadir un sistema de ficheros a un directorio de otro sistema de ficheros**
  - Integra en un único árbol todo el sistema de directorios
  - Hace independiente el nombre del fichero del dispositivo físico
  - Permite utilizar dispositivos de almacenamiento extraíbles.

# 1.1.4.- Montaje de directorios

Ejemplo: **`mount /dev/fd0 /usr/local`**




el nombre absoluto del fichero f1 es: **`/usr/local/f1`**

# Contenidos

1.- Conceptos básicos

- Ficheros

- **Procesos**
- Protección
- Señales

2.- Servicios

- Llamadas al sistema
- Programas del sistema

3.- El intérprete de órdenes

# 1.2.- Procesos

## ✍ Concepto

- ✍ Un programa es una colección de instrucciones y datos que se almacenan en un fichero regular en disco. En su nodo-i el fichero se marca como ejecutable.
- ✍ Los usuarios normalmente crearán ficheros ejecutables utilizando editores, compiladores y montadores. Para poder ejecutar un programa el sistema operativo crea un proceso.
- ✍ Cada proceso tiene asociado un espacio de direcciones. El espacio de direcciones contiene el código, los datos del programa y su pila.

# 1.2.- Procesos

## ✍ Atributos del proceso

- ✍Identificador del proceso. PID
- ✍Identificador del proceso padre. PPID
- ✍Usuario propietario del proceso. rUID y efUID (real y efectivo)
- ✍Grupo propietario del proceso. rGID y efGID (real y efectivo)
- ✍Directorio actual
- ✍Máscara de creación de ficheros
- ✍Tratamiento de señales

# 1.2.- Procesos

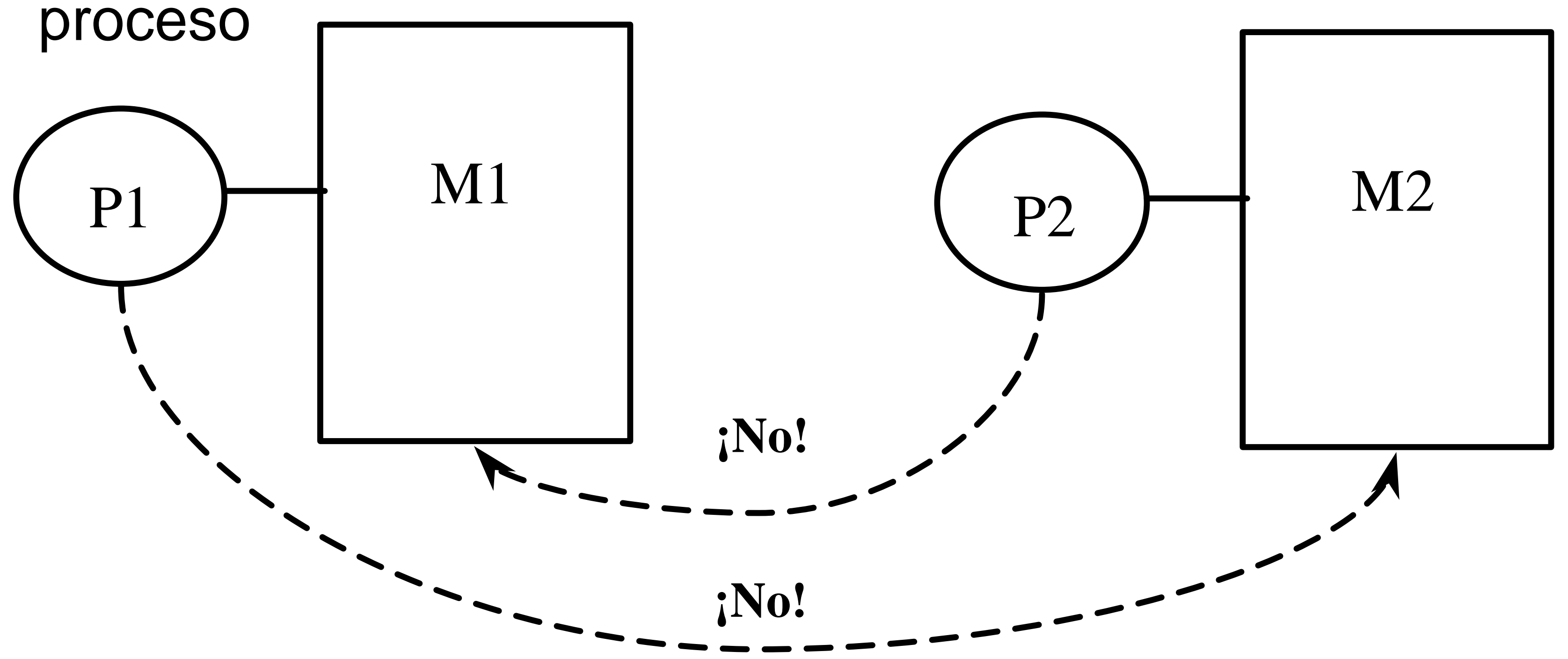
## Atributos del proceso (cont.)

- Tiempos de consumo de procesador
- Descriptores de ficheros abiertos
- Imagen de memoria:
  - código del programa, área de datos, área de pila
- Contexto máquina:
  - contador de programa, puntero de pila, registros de uso general, ...
- Atributos de planificación:
  - prioridad, tiempos de ejecución

# 1.2.- Procesos

## ✍ Imagen de memoria

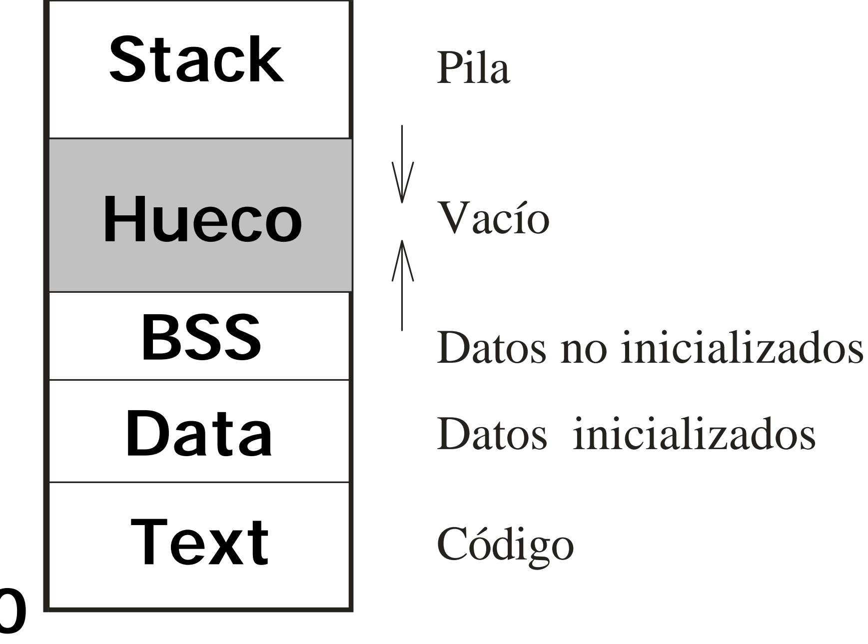
- ✍ La imagen de memoria es un atributo privado de cada proceso




**Leer, escribir, ejecutar**

## ✍ Imagen de memoria

✍ Imagen de memoria segmentada



# 1.2.- Procesos

## ✍ Visualización de atributos

```
$ ps -fl

F S      UID   PID  PPID  C PRI NI    ADDR   SZ    WCHAN    STIME TTY      TIME COMD
1 S aespino 17682 17681  0 158 20   a2a540   55   5a3440 10:14:55 ttyp3    0:00 -ksh
1 S    root 17681   171  0 154 20   9ff800   13   29cb30 10:14:55 ttyp3    0:00 rlogind
1 R aespino 17708 17706  2 178 20   9ffa00   20          10:17:36 ttyp3    0:00 ps -fl
1 S aespino 17706 17698  0 158 20   a2a6c0  132   5a4640 10:17:14 ttyp3    0:00 bash

flags       pid                 nice                      inicio
  estado          pid padre          direccion                     terminal
    UID                 indice consumo      tamaño                          uso CPU
                          prioridad              espera                         programa
```

# 1.2.- Procesos

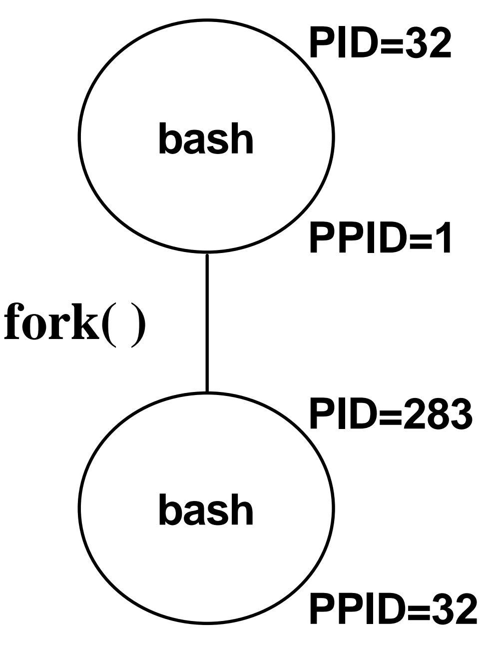
## Creación de Procesos

- Unix utiliza un mecanismo de creación por copia
- El proceso hijo es una réplica exacta de su proceso padre
- El proceso hijo hereda la mayoría de atributos del proceso padre:
  - imagen de memoria
  - UID, GID
  - directorio actual
  - descriptores de ficheros abiertos

# 1.2.- Procesos

## Creación de Procesos

- Unix asigna un identificador a cada proceso denominado PID en el momento de la creación del mismo
- Todo proceso conoce el identificador de su proceso padre, PPID
- La ejecución del hijo es concurrente e independiente



# 1.2.- Procesos

## ✍ Ejecución de programas

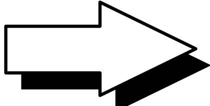
- ✍ Unix utiliza un mecanismo de sustitución para la ejecución de programas
- ✍ Las instrucciones y los datos del proceso se sustituyen por las instrucciones y los datos de un nuevo programa ejecutable
- ✍ El programa comienza a ejecutarse desde el inicio
- ✍ Se preserva el resto de su entorno

# 1.2.- Procesos

## ✍ Ejecución de programas

| PID=4323 |
|---|
| PPID=4319 |
| programa /bin/ksh |
| datos del programa |
| contador del programa |
| registros del procesador |
| otros atributos |

**exec("/bin/ls", ...)**

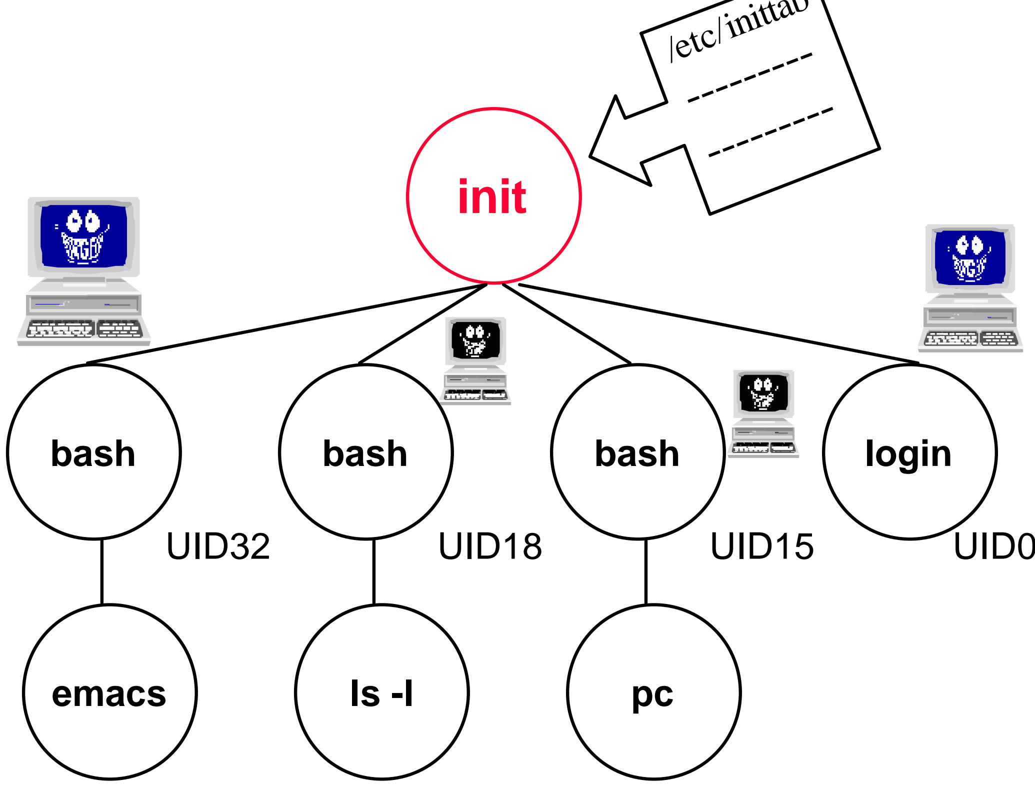


| PID=4323 |
|---|
| PPID=4319 |
| **programa /bin/ls** |
| **datos iniciales** |
| **inicio del programa** |
| **registros del procesador** |
| otros atributos |

# 1.2.- Procesos

## Jerarquía y herencia

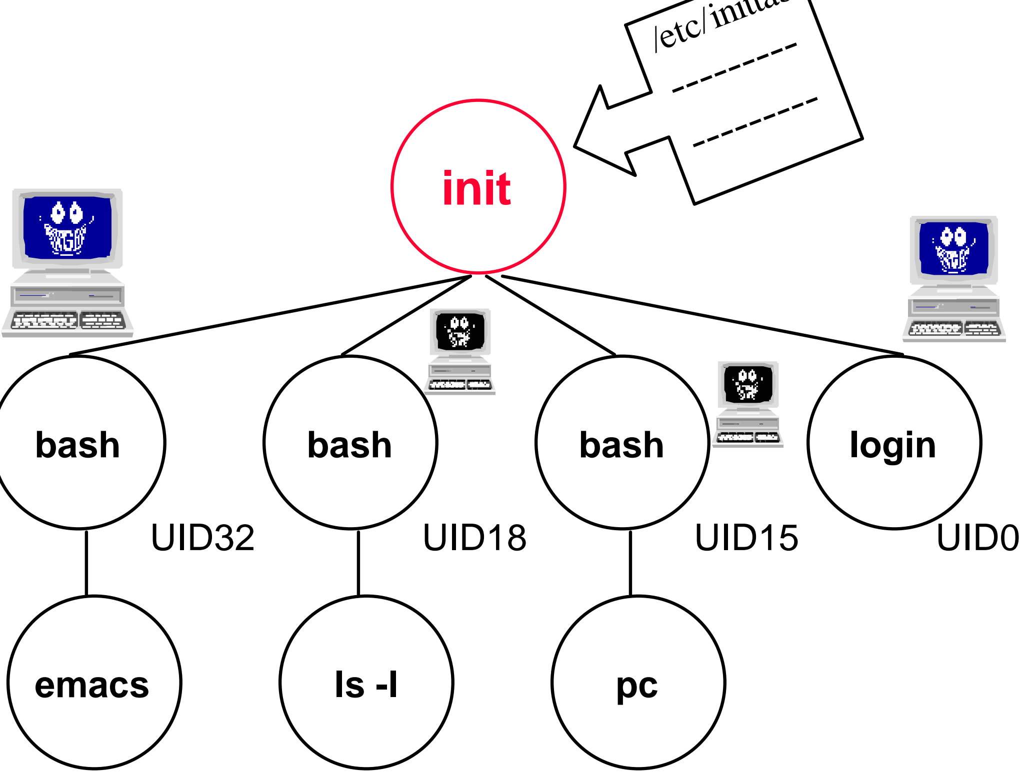
- Se establece una jerarquía similar a la del sistema de ficheros
- **Init** es el patriarca de todos los procesos que se ejecutan en un sistema Unix. Es creado en el arranque del sistema



# 1.2.- Procesos

## Jerarquía y herencia

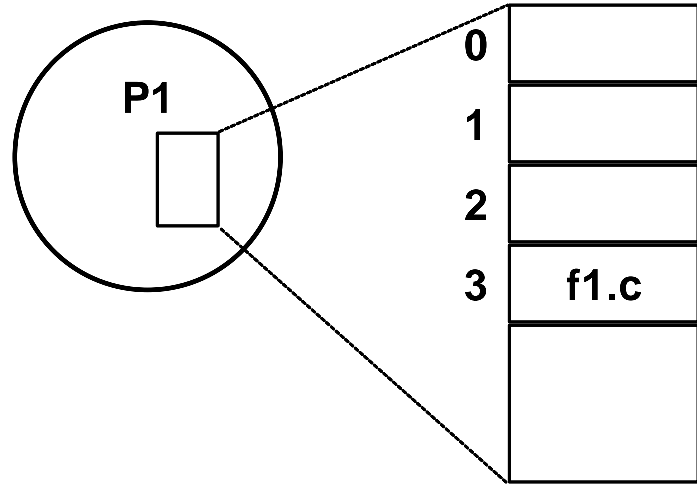
- Cada usuario es propietario de una rama del árbol cuya raíz es el intérprete de mandatos
- Dicha rama se ejecuta con el entorno del usuario gracias al mecanismo de herencia



# 1.2.- Procesos

## Descriptores de fichero

- Representan la asociación del proceso con la entrada-salida que utiliza
- Todos los ficheros que un proceso tiene abiertos se registran en una tabla indexada por los descriptores de fichero

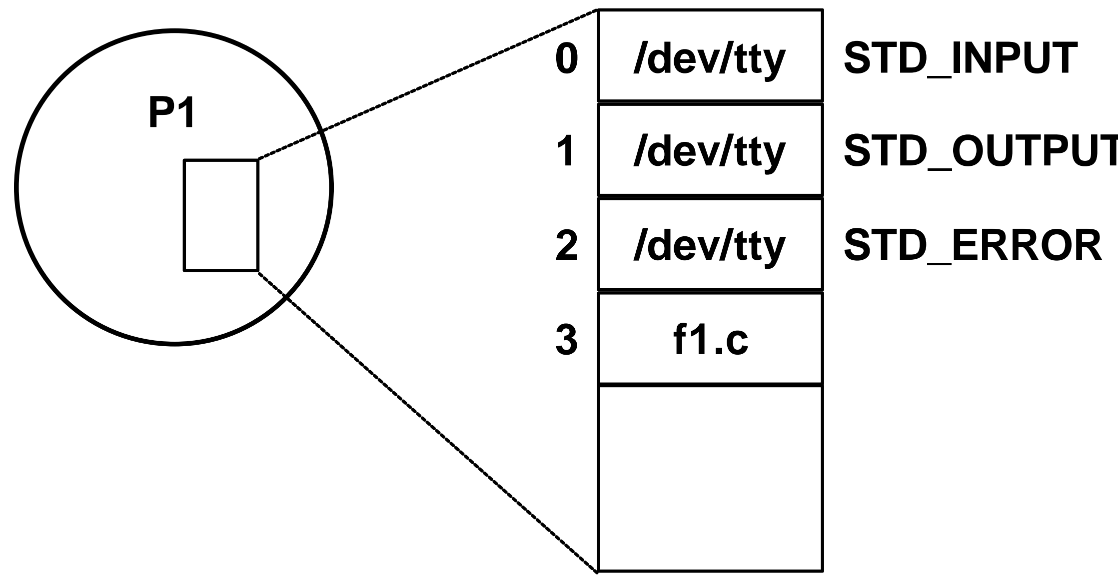



```
fd = open("f1.c", O_RDWR);
...
read(fd, &buffer, nbytes);
...
write(fd, &buffer, nbytes);
...
close(fd);
```

# 1.2.- Procesos

## ✍ Descriptores de fichero

- ✍ Unix define tres descriptores con un significado especial, los cuales suelen estar asociados con el terminal (gracias al mecanismo de herencia siendo el shell el padre)
  - ✍ 0 STD_INPUT: entrada normal de datos
  - ✍ 1 STD_OUTPUT: salida normal de datos
  - ✍ 2 STD_ERROR: salida normal de errores



# 1.2.- Procesos

- Redirección de la salida estándar

```
$ cat a1 a2 a3 > f1.txt
```

| | | |
|---|---|---|
| 0 | /dev/tty | STD_INPUT |
| 1 | f1.txt | STD_OUTPUT |
| 2 | /dev/tty | STD_ERROR |

- Redirección de la entrada estándar

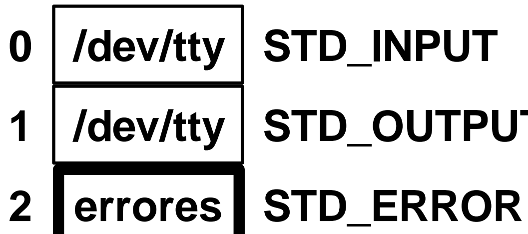
```
$ mailx so102 < mensaje
```

| | | |
|---|---|---|
| 0 | mensaje | STD_INPUT |
| 1 | /dev/tty | STD_OUTPUT |
| 2 | /dev/tty | STD_ERROR |

# 1.2.- Procesos

✍ **Redirección de la salida estándar de error**

```
$ cc programa1.c 2> errores
```



# 1.2.- Procesos

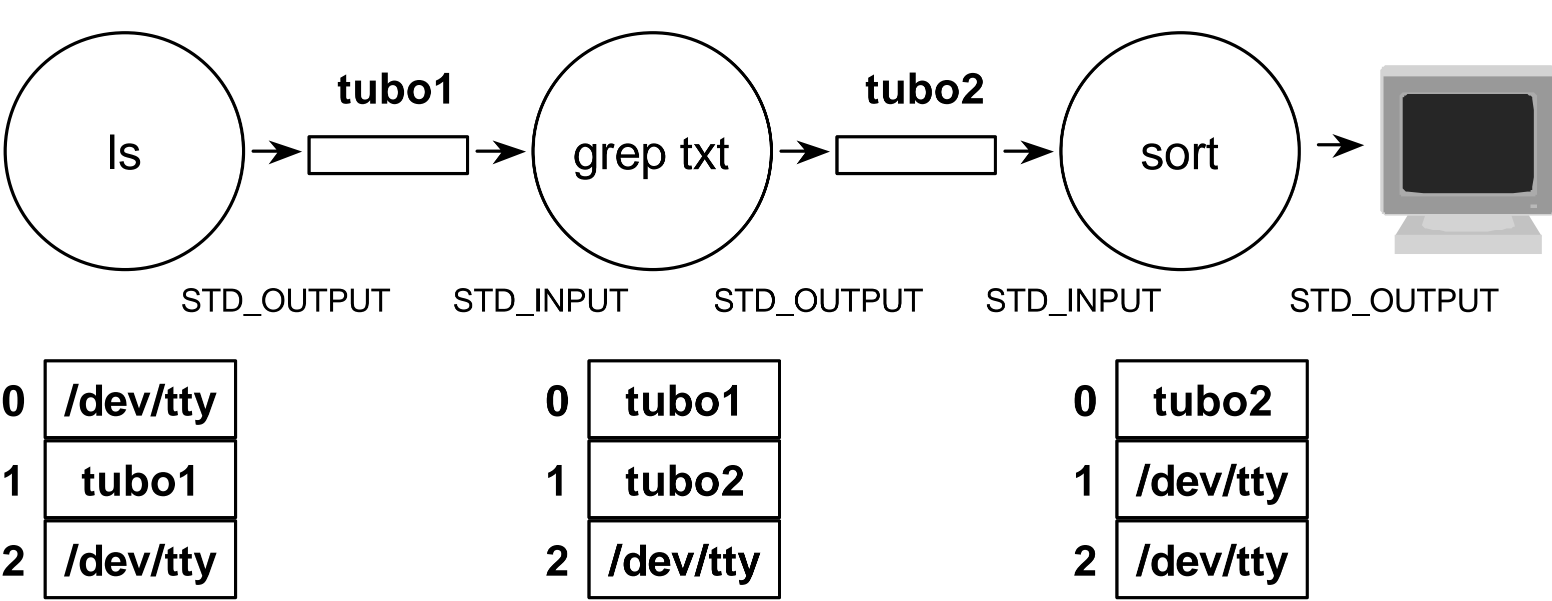
## Comunicación y sincronización entre procesos

- La comunicación entre procesos en Unix se realiza mediante ***tubos***
- Son buffers de capacidad limitada donde cada elemento es un byte
- Una operación de lectura detiene al proceso si el buffer está vacío
- Una operación de escritura detiene al proceso si el buffer está lleno
- Los tubos pueden compartirse gracias al mecanismo de herencia

# 1.2.- Procesos

## Ejemplo de uso

```
$ ls | grep txt | sort
```




| | ls |
|---|---|
| 0 | /dev/tty |
| 1 | tubo1 |
| 2 | /dev/tty |

| | grep txt |
|---|---|
| 0 | tubo1 |
| 1 | tubo2 |
| 2 | /dev/tty |

| | sort |
|---|---|
| 0 | tubo2 |
| 1 | /dev/tty |
| 2 | /dev/tty |

# Contenidos

1.- Conceptos básicos

- Ficheros
- Procesos
- **Protección**
- Señales

2.- Servicios

- Llamadas al sistema
- Programas del sistema

3.- El intérprete de órdenes

# 1.3.- Protección

## ✍ Concepto

- ✍ Mecanismo para controlar los accesos que los procesos realizan a los recursos del sistema y de los usuarios (ficheros, memoria, dispositivos de entrada-salida)

## ✍ Protección en Unix

- ✍ La protección está basada en contrastar los atributos del proceso con los atributos del fichero y determinar si la operación puede efectuarse

| Atributos del proceso |
| --- |
| **efUID**<br>**efGID** |

| Atributos del fichero |
| --- |
| **owner UID**<br>**owner GID**<br>**bits permiso** |

# 1.3.- Protección

## ✍ Asignación de atributos

- ✍ El proceso recibe los atributos gracias al mecanismo de herencia y a la información recogida en la tabla de usuarios (/etc/passwd)
  - ✍ nombre:contraseña_cifrada:**UID**:**GID**:descripción:directorio HOME:shell inicial
- ✍ El fichero recibe los atributos del proceso que lo crea
  - ✍ ownerUID := efUID ownerGID := efGID

# 1.3.- Protección

## Bits de permiso

- 9 bits de permiso en grupos de tres: (propietario, grupo, otros)
- formatos: rwxr-xr-x, 0755
- Interpretación
  - Ficheros regulares: lectura, escritura, ejecución
  - Directorios: listar contenidos, crear o eliminar enlaces, "búsqueda".
  - Especiales: lectura, escritura, --------

# 1.3.- Protección

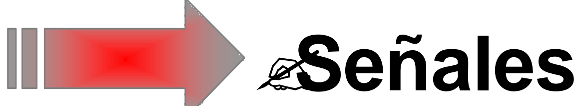
## Reglas de protección

- si efUID = 0, se concede permiso (superusuario)
- si efUID = ownerUID, se utiliza el primer grupo de bits; si no
- si efGID = ownerGID, se utiliza el segundo grupo de bits; si no
- se utiliza el último grupo de bits

# Contenidos

1.- Conceptos básicos

- Ficheros
- Procesos
- Protección



- **Señales**

2.- Servicios

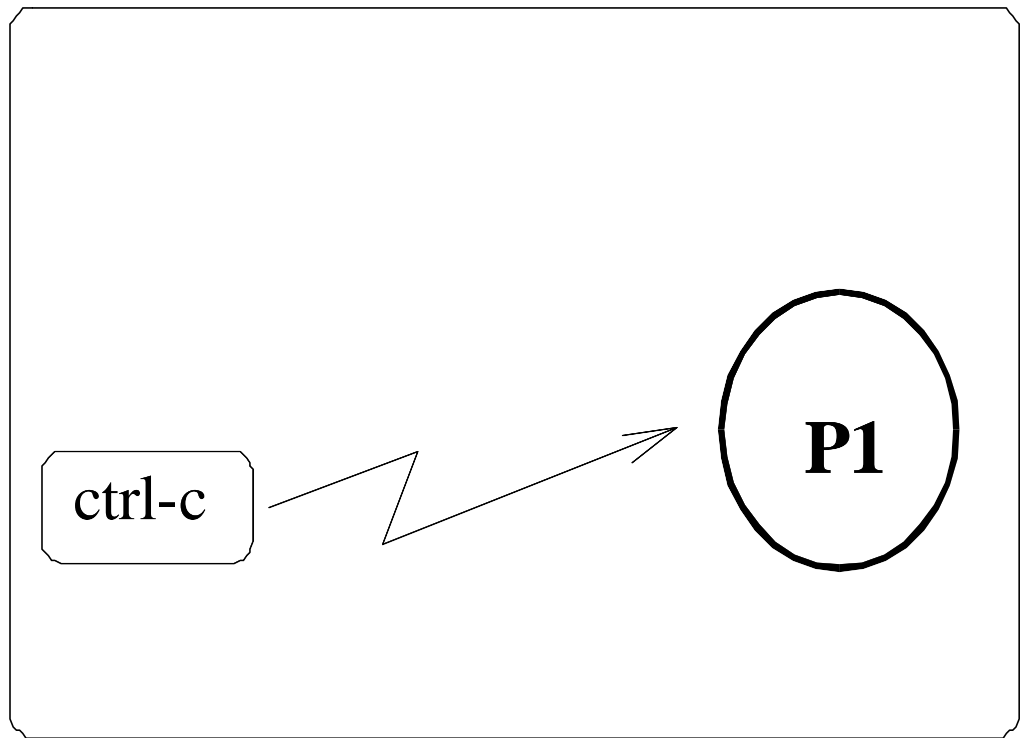
- Llamadas al sistema
- Programas del sistema

3.- El intérprete de órdenes

# 1.4.- Señales

## ✍ Concepto:

- ✍ Evento que interrumpe la ejecución normal de un proceso



# 1.4.- Señales

- **Una señal puede ser originada desde:**
  - El sistema operativo
  - El propio proceso
  - Otro proceso
  - El usuario

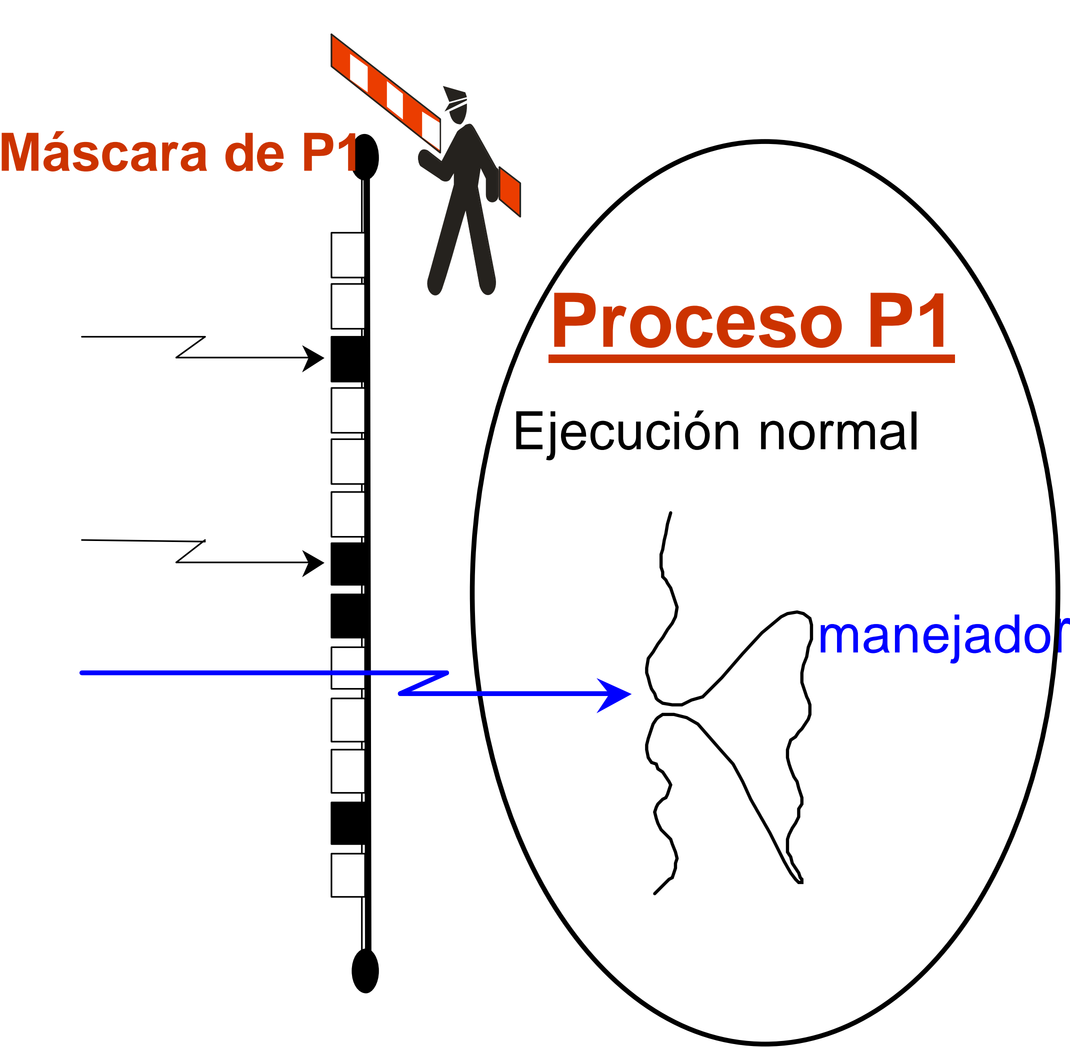
# 1.4.- Señales

## ✍ Las señales están predefinidas por POSIX:

| Número | Nombre | Origen |
|---|---|---|
| 1 | SIGHUP | Conexión de comunicación rota |
| 2 | SIGINT | <ctrl> C del teclado |
| ... | | |
| 8 | SIGFPE | Excepción de coma flotante |
| 9 | SIGKILL | No enmascarable ni manejable |
| 10 | SIGBUS | Error de bus. Violación de memoria |
| 14 | SIGALRM | Generada por el reloj del sistema |

# 1.4.- Señales

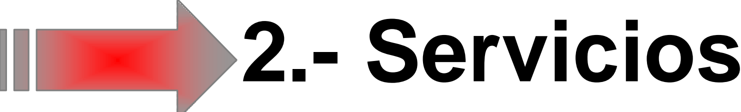
- **POSIX permite que un proceso**
  - asuma el tratamiento por defecto: terminación del proceso
  - ignore la señal
  - capture la señal: ejecución del manejador asociado
  - enmascare la señal



# Contenidos

1.- Conceptos básicos

- ✍Ficheros
- ✍Procesos
- ✍Protección
- ✍Señales



**2.- Servicios**

- ✍Llamadas al sistema
- ✍Programas del sistema

3.- El intérprete de órdenes

# 2.- Servicios del sistema

## Interfaces en Unix

Interfaz de usuario

Interfaz de biblioteca

Interfaz de llamadas al sistema

Usuarios

Programas de utilidades
(shell, editores, compiladores, ...)

Biblioteca estándar
(open, close, read, write, fork, ...)

Sistema Operativo Unix
(gestión de procesos, de memoria,
el sistema de ficheros, E/S, ...)

Hardware
(CPU, memoria, discos, terminales, ...)

Modo usuario

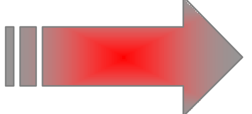
Modo supervisor

# Contenidos

1.- Conceptos básicos

- Ficheros
- Procesos
- Protección
- Señales

2.- Servicios



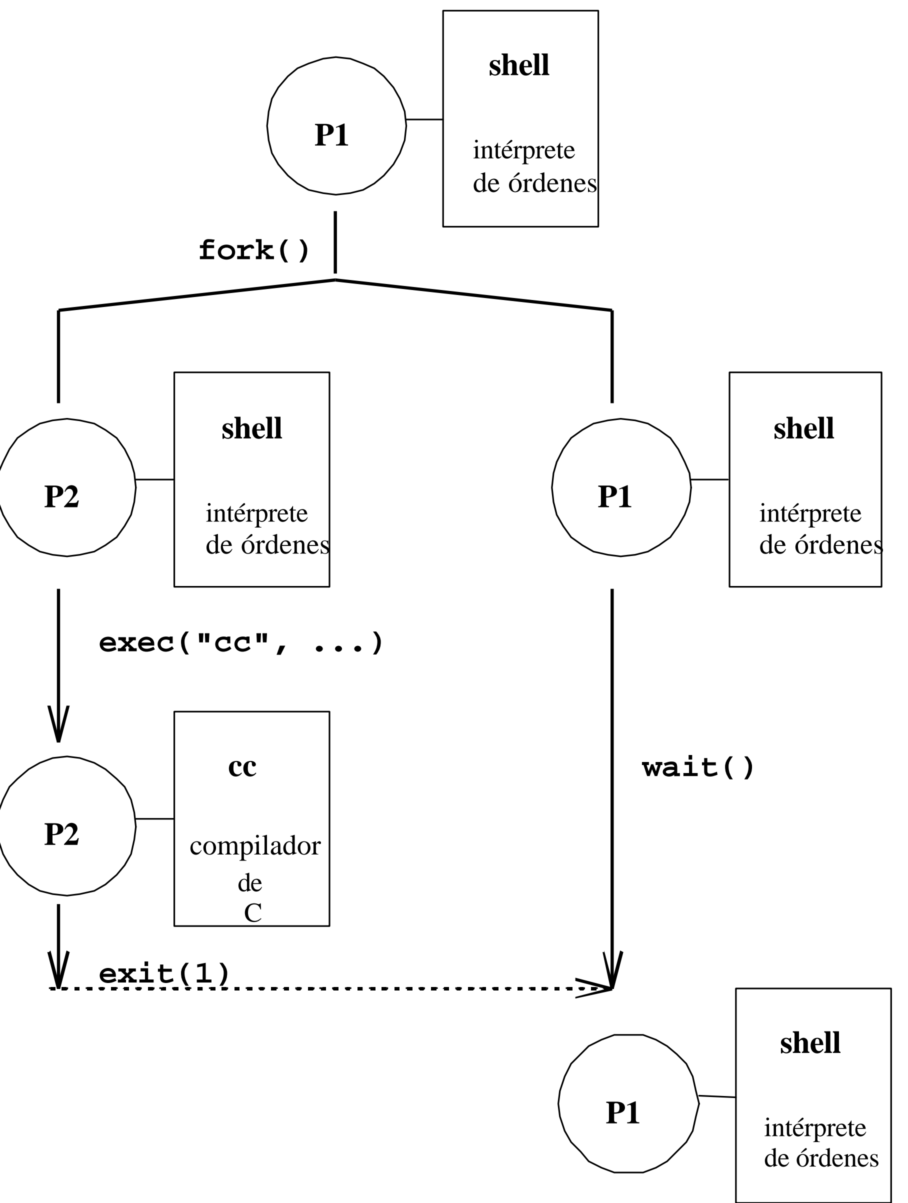
- **Llamadas al sistema**
- Programas del sistema

3.- El intérprete de órdenes

# 2.1.- Llamadas al sistema de Unix

| Gestión de procesos | Descripción |
|---|---|
| pid = fork ( ) | Crear un proceso hijo |
| s = waitpid (pid, status, opts) | Esperar finalización de un hijo |
| s = execve(name, argv, envp) | Cambiar imagen de memoria |
| exit(status) | Invocar finalización y devolver estado |
| | |
| **Gestión de señales** | **Descripción** |
| s = sigaction (sig, act, oact) | Especificar una acción para una señal |
| s = kill (pid, sig) | Enviar una señal a un proceso |
| residual = alarm (seconds) | Planificar una señal SIGALRM al cabo de un tiempo |
| s=pause () | Suspender el que invoca hasta que llegue una señal |
| | |
| **Gestión de memoria** | **Descripción** |
| size = brk (addr) | Modificar tamaño seg. de datos (no POSIX) |

# 2.1.- Ejemplos




## El shell de Unix: estructura simple

```
while(TRUE) {
 imprimir_prompt();
 leer_orden(orden, param);

 pid=fork();                      /* crear hijo */

 if (pid !=  0) {                 /* código del padre */
  waitpid(-1, &status, 0);        /* esperar hijo */

 }else {                          /* código del hijo */
  exec(orden, params, 0);         /* cambiar imagen */
                                  /* de memoria */
  error("No se puede ejec. la orden");
  exit(1);
 }
}
```

# 2.1.- Ejemplos

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

/* Código de fo1.c, ver una traza en las dos hojas siguientes. */
int main ( ) {
    pid_t pid;
    pid = fork ( );
    switch (pid) {
    case -1:
        printf ("No he podido crear el proceso hijo\n");
        break;
    case 0:
        printf ("Soy el hijo, mi PID es %d y mi PPID es %d\n", getpid( ), getppid( ));
        sleep (20);  /* Suspende el proceso durante 20 segundos. */
        break;
    default:
        printf ("Soy el padre, mi PID es %d y el PID de mi hijo es %d\n", getpid( ), pid);
        sleep (30);  /* Suspende el proceso durante 30 segundos.  Acaba antes el hijo. */
     }
    printf ("Final de ejecución de %d \n", getpid( ));
    exit (0);
}
```

# 2.1.- Ejemplos

```
bash-2.01$ gcc -o fo1 fo1.c
bash-2.01$ fo1&
[2] 27888
bash-2.01$ Soy el hijo, mi PID es 27889 y mi PPID es 27888
Soy el padre, mi PID es 27888 y el PID de mi hijo es 27889

bash-2.01$ ps l
 FLAGS   UID   PID  PPID PRI  NI   SIZE   RSS WCHAN        STA TTY TIME COMMAND
   100  2000 27816 27815  15   0   1852  1196 build_mmap_ S   p5  0:00 bash
     0  2000 27818 27816   0   0   5080  3520 msgget       S   p5  0:01 emacs fo1.c
     0  2000 27888 27816   4   0    736   288 send_sig     S   p5  0:00 fo1
     0  2000 27890 27816  17   0   1044   544              R   p5  0:00 ps l
    40  2000 27889 27888   4   0    736   284 send_sig     S   p5  0:00 fo1
bash-2.01$ ps l
Final de ejecución de 27889

 FLAGS   UID   PID  PPID PRI  NI   SIZE   RSS WCHAN        STA TTY TIME COMMAND
   100  2000 27816 27815  15   0   1852  1196 build_mmap_ S   p5  0:00 bash
     0  2000 27818 27816   0   0   5080  3520 msgget       S   p5  0:01 emacs fo1.c
     0  2000 27888 27816   2   0    736   288 send_sig     S   p5  0:00 fo1
     0  2000 27891 27816  17   0   1044   544              R   p5  0:00 ps l
    44  2000 27889 27888   2   0      0     0 do_munmap    Z   p5  0:00 (fo1 <zombie>)
bash-2.01$ Final de ejecución de 27888
```

# 2.1.- Ejemplos

```
bash-2.01$ ps l
 FLAGS   UID   PID  PPID PRI  NI   SIZE   RSS WCHAN       STA TTY TIME COMMAND
   100  2000 27816 27815  15   0   1852  1196 build_mmap_ S   p5  0:00 bash
     0  2000 27818 27816   0   0   5080  3520 msgget      S   p5  0:01 emacs fo1.c
     0  2000 27892 27816  17   0   1044   544             R   p5  0:00 ps l
```

# 2.1.- Ejemplos

```
#include <stdio.h>
/* Este es el código fuente del programa exe1.c, ver una traza en las dos hojas
   siguientes. */
int main (int argc, char *argv[ ])
{
    int i;
    printf ("\n Ejecutando el programa (%s). Sus argumentos son: \n",argv[0]);
    for ( i = 0; i < argc; i ++ )
        printf ("    argv[%d] : %s \n", i, argv[i]);

    sleep(10);  /* Para el proceso durante 10 segundos. */
    /* Sustituye el programa en ejecución por el especificado en los argumentos
       recibidos. */
    if (execvp (argv[1],&argv[1]) < 0) {
        printf ("Error en la invocación \n");
        exit (1);
        };

    exit (0);
}
```

# 2.1.- Ejemplos

```
bash-2.01$ exe1 ls -l&
[1] 27915
Ejecutando el programa (exe1). Sus argumentos son:
    argv[0] : exe1
    argv[1] : ls
    argv[2] : -l
bash-2.01$ ps l
 FLAGS    UID   PID  PPID PRI  NI   SIZE   RSS WCHAN        STA TTY TIME COMMAND
   100   2000 27906 27905  15   0   1848  1196 build_mmap_ S   p6  0:00 bash
     0   2000 27915 27906   7   0    736   288 send_sig    S   p6  0:00 exe1 ls -l
     0   2000 27916 27906  17   0   1044   544             R   p6  0:00 ps l
bash-2.01$ total 32
-rw-r-----   1 so2      so2            597 Dec 14 10:49 espe1.c
-rw-r-----   1 so2      so2            576 Dec 14 10:49 espe2.c
-rwxr-xr-x   1 so2      so2           4554 Dec 14 11:29 exe1
-rw-r-----   1 so2      so2            427 Dec 14 11:29 exec.c
-rwxr-xr-x   1 so2      so2           4770 Dec 14 11:12 fo1
-rw-r-----   1 so2      so2            576 Dec 14 10:59 fo1.c
-rw-r-----   1 so2      so2            963 Dec 14 10:49 fo2.c
-rw-r-----   1 so2      so2           2977 Dec 14 10:49 inc-dec-01.c
-rw-r-----   1 so2      so2           3236 Dec 14 10:49 inc-dec-02.c
-rwxrwxr-x   1 so2      so2           4563 Dec 14 10:50 nuevoprog
-rw-r-----   1 so2      so2            463 Dec 14 10:49 prog1.c
-rw-r-----   1 so2      so2            302 Dec 14 10:49 prog2.c
bash-2.01$
```

# 2.1.- Ejemplos

```
bash-2.01$ exe1 ps l &
[1] 27917
bash-2.01$
Ejecutando el programa (exe1). Sus argumentos son:
    argv[0] : exe1
    argv[1] : ps
    argv[2] : l

bash-2.01$ ps l
 FLAGS   UID   PID  PPID PRI  NI   SIZE   RSS WCHAN       STA TTY TIME COMMAND
   100  2000 27906 27905  15   0   1848  1196 build_mmap_ S   p6  0:00 bash
     0  2000 27917 27906   4   0    736   288 send_sig    S   p6  0:00 exe1 ps l
     0  2000 27918 27906  17   0   1044   544             R   p6  0:00 ps l
bash-2.01$
bash-2.01$
 FLAGS   UID   PID  PPID PRI  NI   SIZE   RSS WCHAN       STA TTY TIME COMMAND
   100  2000 27906 27905  10   0   1848  1196 lance32_sta S   p6  0:00 bash
100000  2000 27917 27906   4   0   1040   540             R   p6  0:00 ps l

[1]+  Done                        exe1 ps l
bash-2.01$
```

# 2.1.- Llamadas al sistema de Unix

| Gestión de ficheros/directorios | Descripción |
|---|---|
| fd = creat(name, mode) | Crear un fichero |
| fd = open(file, how) | Abrir un fichero para lectura y/o escritura |
| s = close(fd) | Cerrar un fichero abierto |
| n = read (fd, buffer, nbytes) | Leer de un desc. de fichero sobre un buffer |
| n = write (fd, buffer nbytes) | Escribir de un buffer sobre un desc. de fichero |
| pos = lseek (fd, offset, whence) | Posicionar el puntero de posición |
| s = stat (name,buf), | Obtener atributos de un fichero de su nodo-i |
| s = mkdir(name, mode) | Crear un directorio |
| s = rmdir(name) | Borrar un directorio vacío |
| s = link (name1, name2) | Crear una entrada de directorio para un fichero existente |
| s = unlink(name) | Borrar una entrada de directorio |
| s = chdir(dirname) | Cambiar el directorio de trabajo |
| s = chmod(name, mode) | Cambiar bits de protección de un fichero |

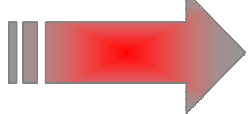
# Contenidos

1.- Conceptos básicos

- Ficheros
- Procesos
- Protección
- Señales

2.- Servicios

- Llamadas al sistema



- **Programas del sistema**

3.- El intérprete de órdenes

# 2.2.- Programas del sistema

## Concepto

- Utilidades del sistema operativo que se ejecutan como procesos de usuario y proporcionan un entorno más cómodo.

## Clasificación

- Tratamiento de ficheros y directorios: mkdir, cp, mv, ls ……
- Filtros: grep, sort, head, tail ….
- Editores, compiladores, ensambladores, editores de enlace …
- Sistemas de ventanas: X11
- Comunicaciones: mail, ftp, rlogin ….
- Intérpretes de órdenes: sh, ksh, bash

# 2.2.- Ejemplo

## ✍ Programa micp.c

```
#define BUFSIZE 1024

void copy(char *from, char *to)
{
  int fromfd, tofd, nread;
  char buf[BUFSIZE];

  if ((fromfd = open(from, O_RDONLY)) == -1) perror(from);

  if ((tofd = creat(to,0666)) == -1) perror(to);

  while ((nread = read(fromfd, buf, sizeof (buf))) > 0)
      if (write(tofd, buf, nread) != nread) perror("write");

  if (nread == -1) perror("read");

  if (close(fromfd) == -1 || close(tofd) == -1)perror("close");
}
```

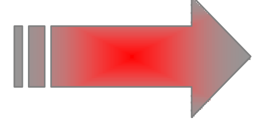
# Contenidos

1.- Conceptos básicos

- ✍Ficheros
- ✍Procesos
- ✍Protección
- ✍Señales

2.- Servicios

- ✍Llamadas al sistema
- ✍Programas del sistema



**3.- El intérprete de órdenes**

# 3.- El intérprete de órdenes

## ✍ Concepto

- ✍ Es la interfaz primaria entre el usuario y el sistema operativo.
- ✍ Es un programa que lee una orden (o un fichero de texto con múltiples órdenes) introducida por un usuario, la analiza y la ejecuta.
- ✍ En UNIX, el intérprete de órdenes es un programa que se ejecuta como un proceso de usuario.
- ✍ Ejemplos: sh, bash, ksh, csh, ...

# 3.- El intérprete de órdenes

## El Shell de Unix: órdenes

- Variables / órdenes internas

```
$ PATH=/usr/bin:/bin
$ PATH=$PATH:$HOME/bin
$ unset PATH
$ export PATH
$ echo $PATH
```

- Órdenes externas

```
$ prog -optn
$ ls -al
$ ps -eaf
$ cat f1 f2
$ grep hola f1
```

# 3.- El intérprete de órdenes

## ✍ El Shell de Unix: órdenes

✍ Redirecciones

```
$ cat f1 > f2
$ echo hola > f2
$ mailx peter@cs.uu.edu < fichero
```

✍ Tubos

```
$ cat f1 f2 | more
$ cat f1 f2 | grep hola | wc > fitxer
```

✍ Ejecución en *background*

```
$ cc -c f1.c f2.c f3.c &
$ (inmediato)
```

# 3.- El Shell de Unix: órdenes

✍Control del flujo/ parámetros

```
$ for i in dso*
> do
> mv $i ./alumnes
> echo moviendo $i
> done
```

✍Shell scripts: ficheros que contienen órdenes de Shell

```
if cp $1 $2
  then
    echo copiado $1 a $2
  else
    echo No se ha podido hacer la copia
fi
```