# INTRODUCCIÓN A UNIX

Rodrigo García Carmona
Universidad San Pablo-CEU
Escuela Politécnica Superior

# OBJETIVOS

- Presentar **Unix** como un **ejemplo de sistema operativo** sobre el que poder aplicar los conceptos vistos en el tema anterior y sobre el que poder desarrollar los conocimientos a adquirir en los próximos temas.
- Dar a conocer las principales **abstracciones** sobre las que trabaja este sistema: **proceso** y **fichero**.
- Comentar las **interfaces** que se pueden encontrar en un sistema Unix.
- Presentar sus **llamadas al sistema** más importantes.
- Dar una descripción genérica de su **intérprete de órdenes**: la **shell** de Unix.

# CONTENIDO

- Conceptos básicos
  - Ficheros
  - Procesos
  - Protección
  - Señales
- Servicios
- El intérprete de órdenes

**Bibliografía**

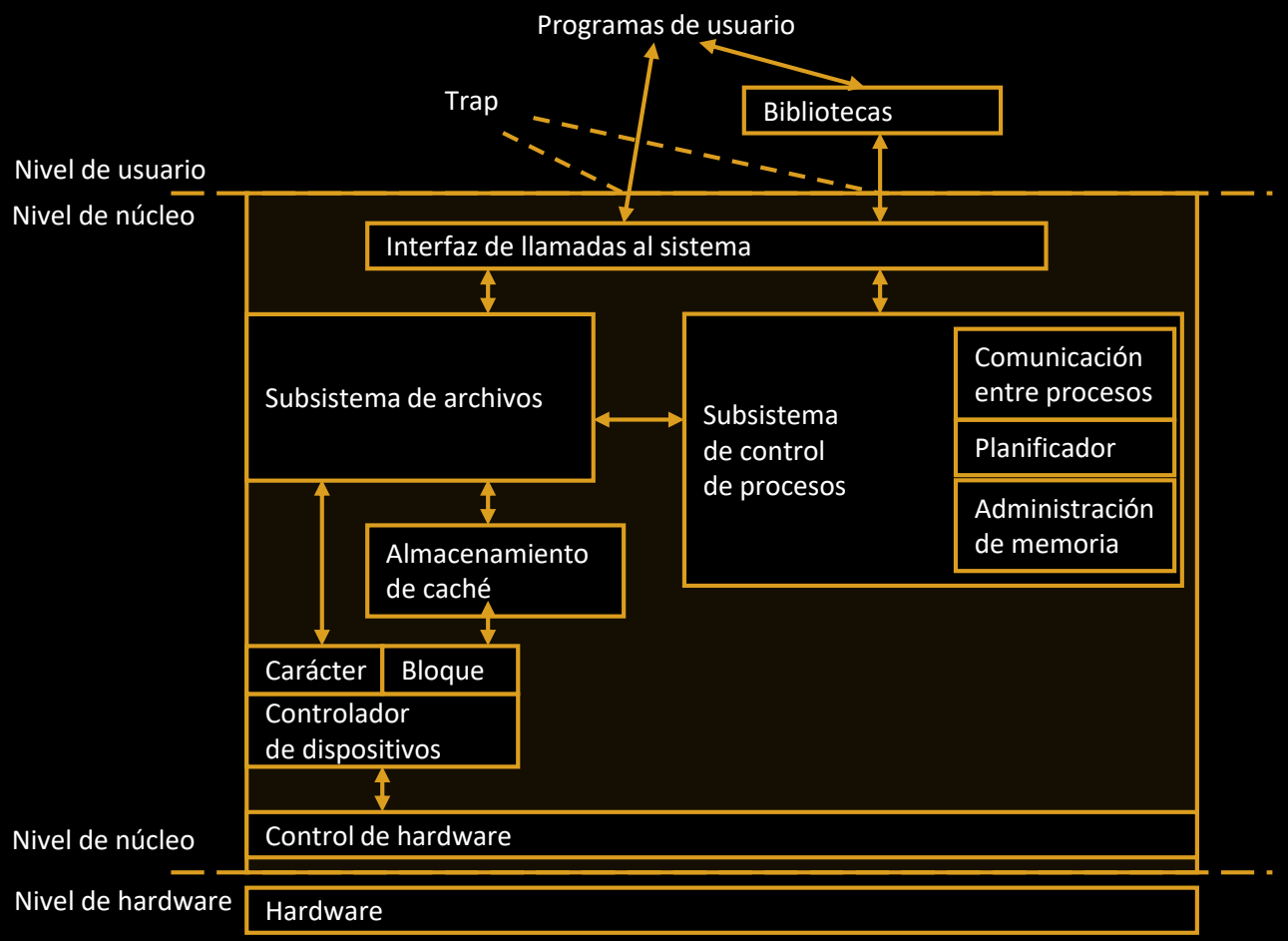
- W. Stallings: **Sistemas Operativos**.
  - Capítulo 2, 11 y 12.
- A.S. Tanenbaum: **Modern Operating Systems**.
  - Capítulos 10.

# CONCEPTOS BÁSICOS

# ¿QUÉ ES UNIX?

- Unix es un **sistema operativo multiusuario** y de **tiempo compartido** muy popular.
- Su ámbito de aplicación se extiende desde los **computadores personales** hasta los **grandes sistemas**.
- La primera versión fue escrita en ensamblador por una persona (Ken Thompson) en los laboratorios Bell a finales de los 60 como reacción a MULTICS. Versión monousuario para PDP-7. Posteriormente se adaptó a la familia de computadores PDP-11. El código fuente era público y alcanzó una gran difusión entre la comunidad científica.
- Dos líneas comerciales:
  - **System V** (desarrollada por AT&T)
  - **BSD** (Berkeley Software Distribution)
- Intentos de estandarización:
  - **POSIX:** Portable Operationg System Interface
  - **OSF:** Open Software Fundation
  - **UI:** Unix International

# NÚCLEO DE UNIX CLÁSICO



Ficheros

# CONCEPTOS BÁSICOS

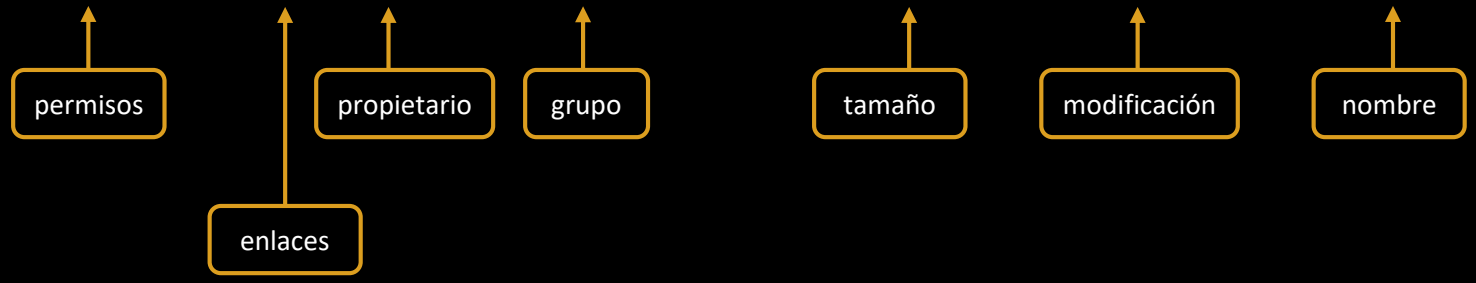
# FICHEROS

- Abstracción del espacio de almacenamiento secundario.
- **Tipos de fichero:**
  - **Normal (*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.
- **Atributos de ficheros:**

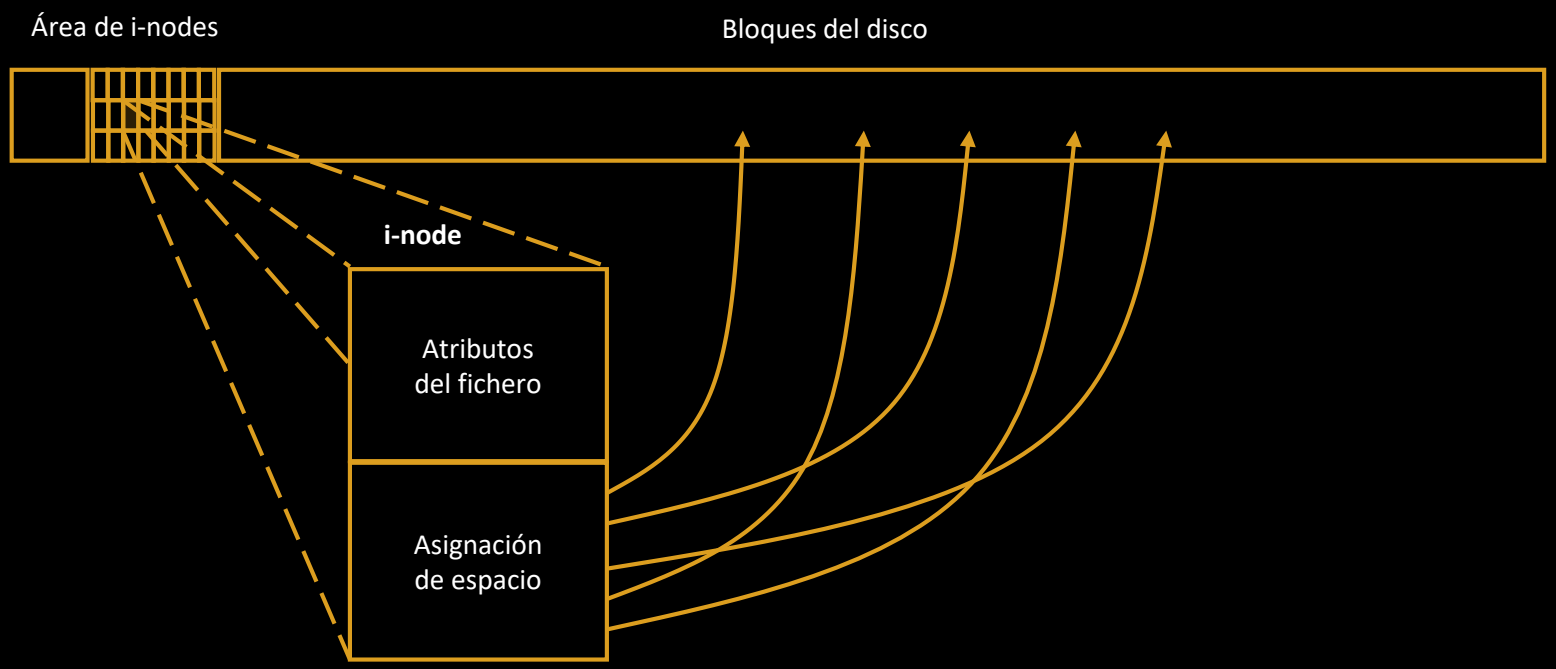
  Se mantienen en un nodo-i (*inode*), en un área reservada del disco.
  - Tipo de fichero.
  - Permisos de acceso (*permission bits*).
  - Propietario (*owner UID*).
  - Grupo propietario (*owner GID*).
  - Número de enlaces.
  - Instantes de creación, último acceso y última modificación.
  - Tamaño.

# VISUALIZACIÓN DE ATRIBUTOS DE FICHEROS

```
servidor:/home/ssoo> ls -la
total 5
drwxr-xr-x    4 root     root          128 2012-10-06 17:54 .
drwxr-xr-x   21 root     root          512 2012-10-06 17:51 ..
drwxr-xr-x    2 root     root          160 2012-10-02 17:12 material
-rw-r--r--    1 root     root         2374 2012-10-02 16:51 policy.txt
drwxr-xr-x    2 root     root           48 2012-10-02 17:02 practs
```

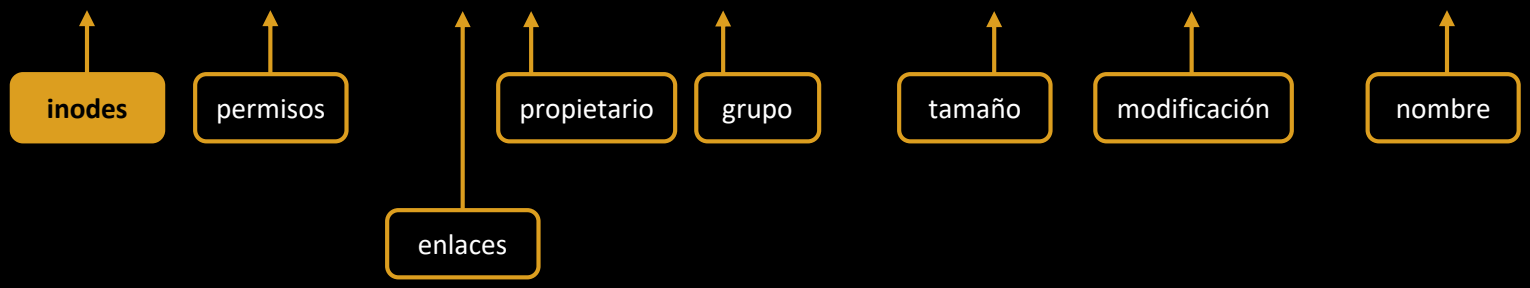


# IMPLEMENTACIÓN DE FICHEROS



# VISUALIZACIÓN DE INODES DE FICHEROS

```
servidor:/home/ssoo> ls -lai
total 5
 206244 drwxr-xr-x    4 root     root          128 2012-10-06 17:54 .
      2 drwxr-xr-x   21 root     root          512 2012-10-06 17:51 ..
 207398 drwxr-xr-x    2 root     root          160 2012-10-02 17:12 material
 204012 -rw-r--r--    1 root     root         2374 2012-10-02 16:51 policy.txt
 207586 drwxr-xr-x    2 root     root           48 2012-10-02 17:02 practs
```

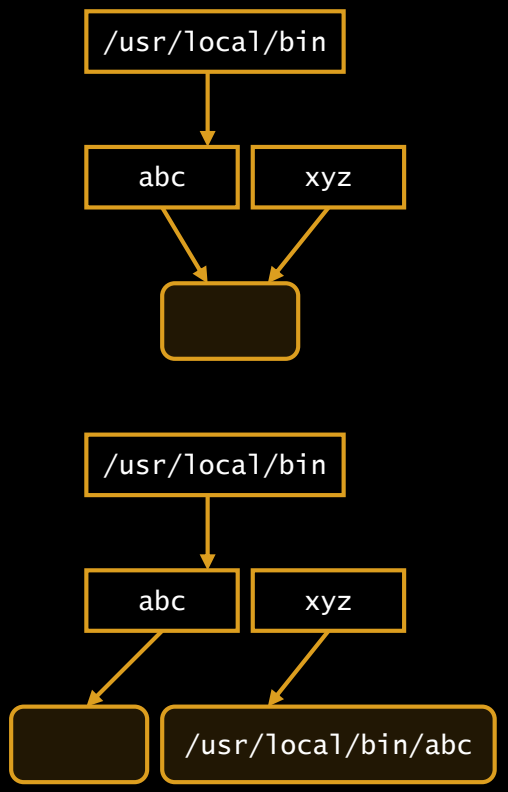


# DIRECTORIOS

- **Es un tipo de fichero que permite organizar los ficheros jerárquicamente:**
  - Mediante su vía o ruta de acceso (*path*).
  - Se consideran **rutas absolutas** si empiezan por /:
    - /home/rodrigo/profile
  - Si no comienzan por / se considera **relativas** al directorio de trabajo actual
    - ./profile asumiendo /home/rodrigo como directorio de trabajo
- **Existen siempre dos ficheros "especiales" en cada directorio:**
  - . (directorio actual)
  - .. (directorio superior o padre)
  - Asumiendo /home/rodrigo como el directorio actual, los siguientes *paths* son equivalentes:
    - ../traspas/SOS-1.ppt
    - ./../traspas/SOS-1.ppt
- Los **ficheros** pueden tener **más de un nombre**. Cada uno de esos nombres se denomina **enlace**.

# ENLACES FÍSICOS Y SIMBÓLICOS

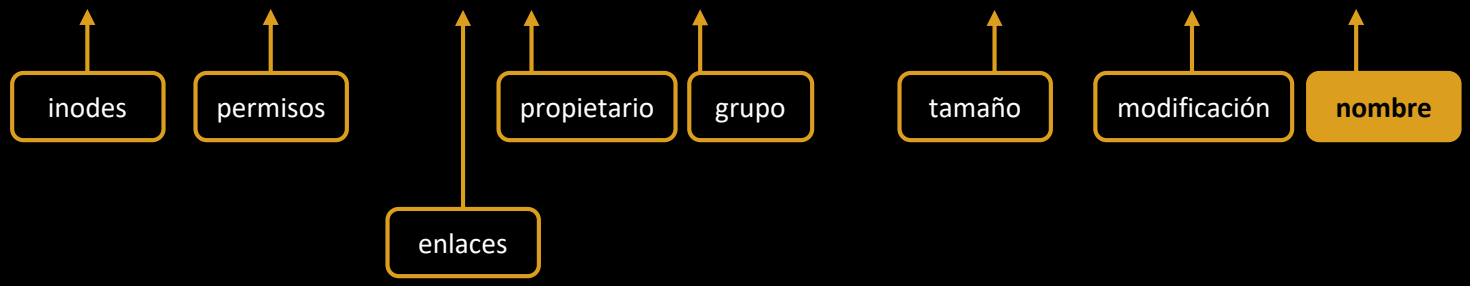
- **Enlaces físicos:**
  - En este caso todos los nombres hacen referencia a un mismo *inode*.
  - El fichero sólo se elimina del disco cuando se borran todos los enlaces (todas las entradas de directorio que lo referencian).
  - Sólo se permite (salvo al superusuario) enlazar ficheros regulares (no directorios).
- **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 crear enlaces circulares.
  - Permiten atravesar sistemas de ficheros que residen en dispositivos físicos distintos.



# VISUALIZACIÓN DE ENLACES FÍSICOS Y SIMBÓLICOS

```
servidor:/home/ssoo> ln policy.txt normas
servidor:/home/ssoo> ln -s policy.txt policysim

total 5
 231264 drwxr-xr-x    2 root     root          160 2012-10-02 16:57 material
  31544 -rw-r--r--    2 root     root         2374 2012-10-07 01:07 normas
  31544 -rw-r--r--    2 root     root         2374 2012-10-07 01:07 policy.txt
 317346 lrwxrwxrwx    1 root     root           10 2012-10-07 01:08 policysim -> policy.txt
 231368 drwxr-xr-x    2 root     root           48 2012-10-02 16:56 practicas
```



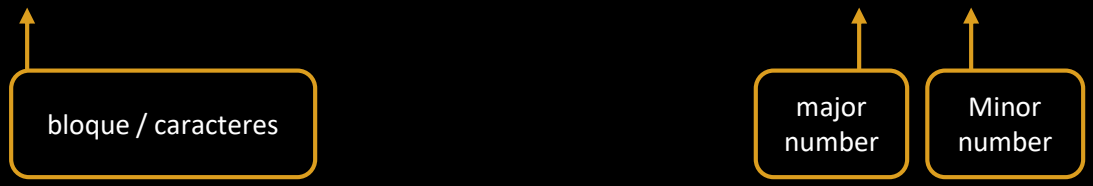
# ESTRUCTURA DE DIRECTORIOS TÍPICA DE UNIX

# FICHEROS ESPECIALES

- Se encuentran en el directorio **/dev**
- Son tratados exactamente igual que los ficheros regulares.
- Un dispositivo se considera una **secuencia de bytes**.
- **Se consideran dos tipos de ficheros especiales:**
  - **De bloque:** discos duros, unidades ópticas, discos RAM, particiones…
    - /dev/hda1, /dev/fd0
  - **De caracteres:** conexiones serie, interfaces de red, terminales, impresoras…
    - /dev/tty00, /dev/lp0
- **Los ficheros especiales de bloque 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:** identifica el *driver* asociado. Representa el tipo de dispositivo.
  - **Minor number:** identifica una unidad del tipo de dispositivo
  - **Ejemplo:** Disco duro 8,2; terminal 4,1.

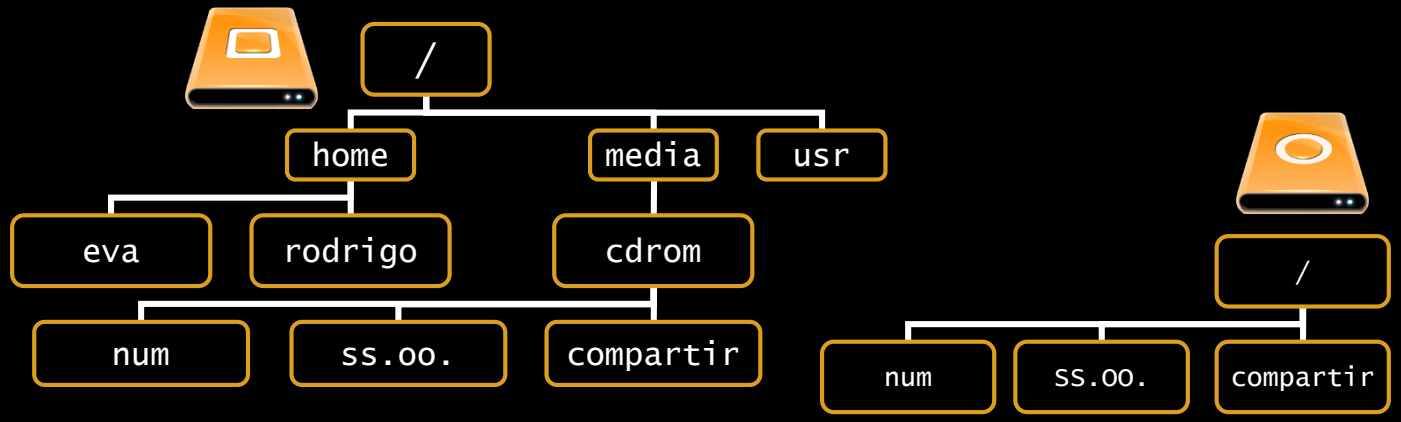
# VISUALIZACIÓN DE ATRIBUTOS DE FICHEROS ESPECIALES

```
servidor:/dev> ls -l
crw-rw-rw-    1 root     root       1,   3 2012-03-14 14:37 null
crw-------    1 root     root      10,   1 2012-03-14 14:37 psaux
crw-------    1 root     root       4,   1 2012-03-14 14:37 tty1
crw-rw-rw-    1 root     tty        4,  64 2012-03-14 14:37 ttyS0
crw-rw----    1 root     uucp       4,  65 2012-03-14 14:37 ttyS1
crw--w----    1 vcsa     tty        7,   1 2012-03-14 14:37 vcs1
crw--w----    1 vcsa     tty        7, 129 2012-03-14 14:37 vcsa1
crw-rw-rw-    1 root     root       1,   5 2012-03-14 14:37 zero
brw-rw----    1 root     disk       8,   1 2012-03-14 14:37 sda1
```



# 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.
- Permite utilizar dispositivos de almacenamiento **extraíbles o en red**.
- **Hace independiente el nombre del fichero del dispositivo físico.**
- El nombre absoluto del fichero es con la ruta de acceso al directorio de montaje.

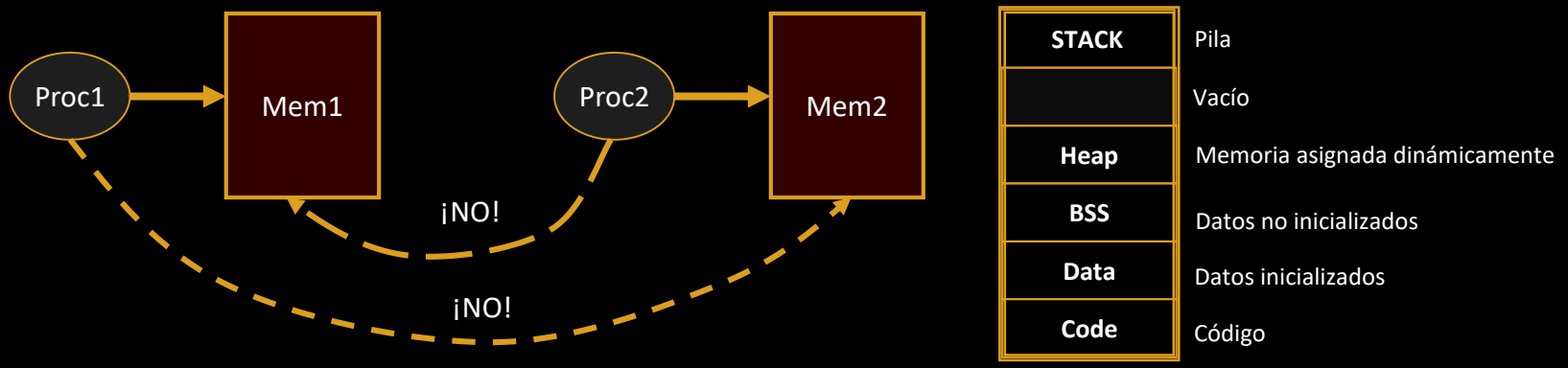


Procesos

# CONCEPTOS BÁSICOS

# PROCESOS

- Un **programa** es una **colección de instrucciones y datos** que se almacenan en un **fichero regular** en disco. En su inode 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.



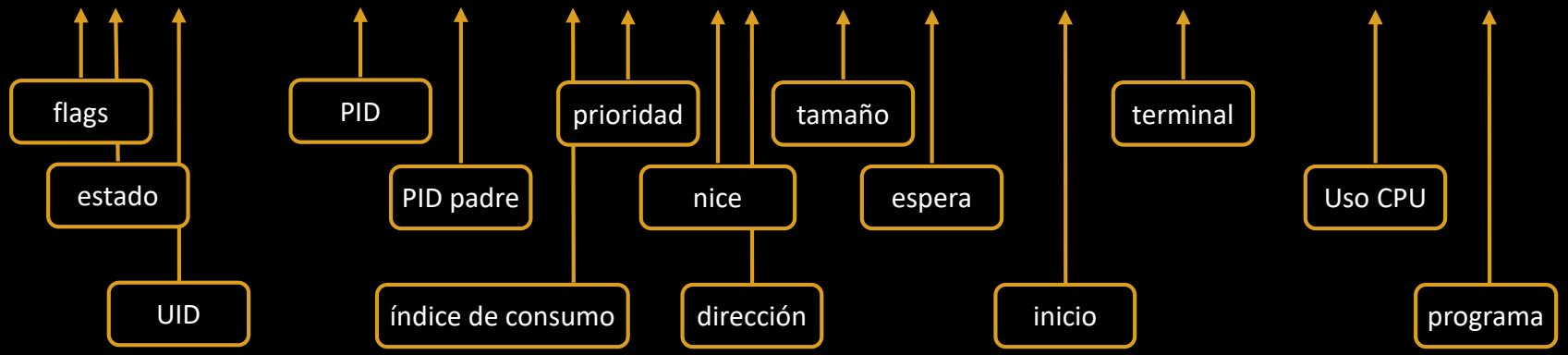
# ATRIBUTOS DE UN PROCESO

- **PID:** identificador del proceso.
- **PPID:** identificador del proceso padre.
- **rUID y efUID:** usuario propietario del proceso real y efectivo.
- **rGID y efGID:** grupo propietario del proceso real y efectivo.
- **Directorio actual:** de trabajo.
- **Máscara de creación de ficheros:** depende del *umask* del usuario.
- **Tratamiento de señales.**
- **Tiempos de consumo del procesador.**
- **Descriptores de ficheros abiertos.**
- **Imagen de memoria:** es un atributo privado de cada proceso.
  - Código del programa, área de datos, área de pila…
- **Contexto máquina:**
  - Contador del programa, puntero de pila, registros de uso general…
- **Atributos de planificación:**
  - Prioridad, tiempos de ejecución.

# VISUALIZACIÓN DE ATRIBUTOS DE UN PROCESO

```
servidor:/> ps -lf

F S UID          PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 S root        4384  4381  0  75   0 -  1219 wait4  09:08 pts/1    00:00:00 -bash
0 T root        4438  4384  0  80   0 -   427 signal 09:12 pts/1    00:00:00 top
0 R root        4497  4384  0  81   0 -   872 -      09:17 pts/1    00:00:00 ps -l
```



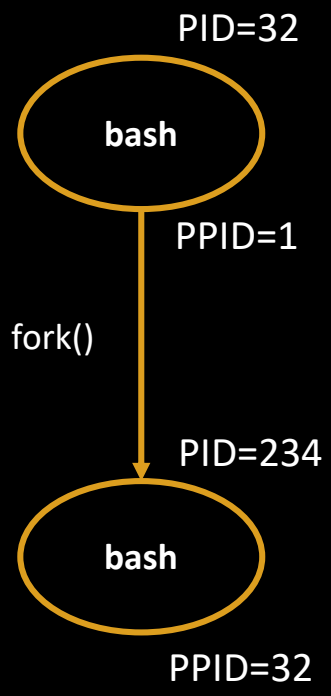
# PROCESO DE EJECUCIÓN EN UNIX

Compuesto de dos pasos:

- **Creación de proceso.**
  - Por copia.
  - Instrucción *fork()*.
- **Ejecución de programa.**
  - Sobre el proceso copiado.
  - Instrucción *exec()*.

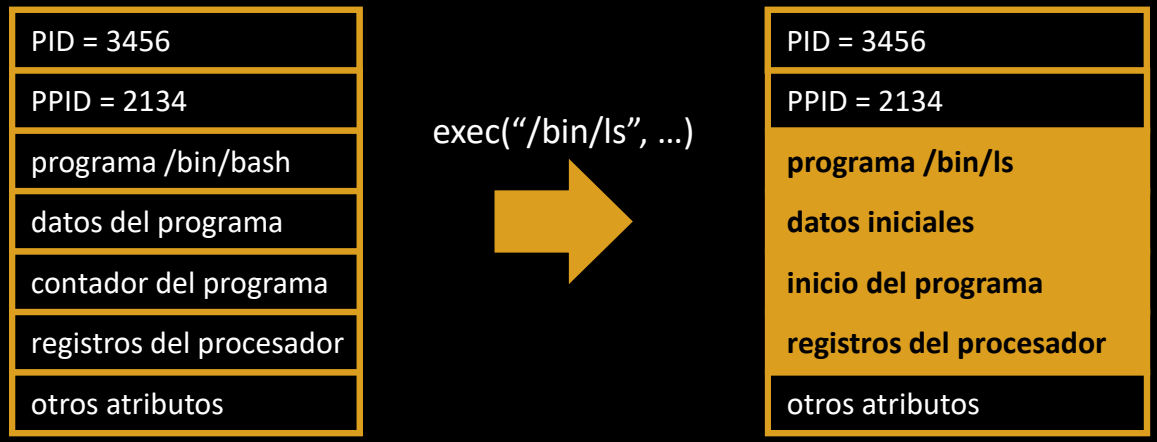
# CREACIÓN DE PROCESOS

- Unix utiliza un mecanismo de **creación por copia**.
- El proceso hijo es una **copia exacta** de su proceso padre.
- **El proceso hijo hereda la mayoría de los atributos del proceso padre:**
  - Imagen de memoria.
  - UID, GID.
  - Directorio actual.
  - Descriptores de ficheros abiertos.
- Unix asigna un **identificador (PID)** a cada proceso en el momento de creación del mismo.
- Todo proceso conoce el **identificador de su proceso padre (PPID)**.
- La ejecución del hijo es **concurrente** e **independiente**.



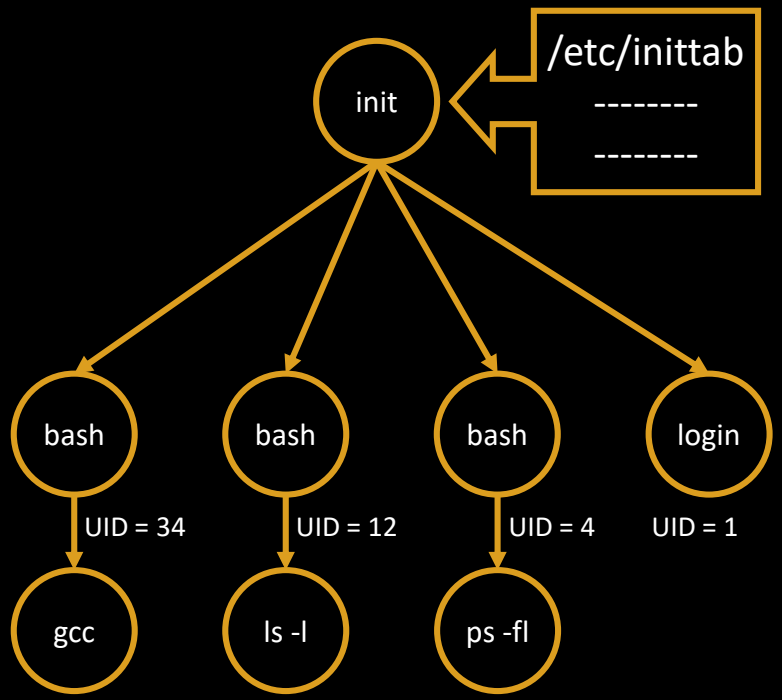
# EJECUCIÓN DE PROGRAMAS

- 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 empieza a ejecutarse desde el inicio.
- Se **preserva** el resto de su **entorno**.



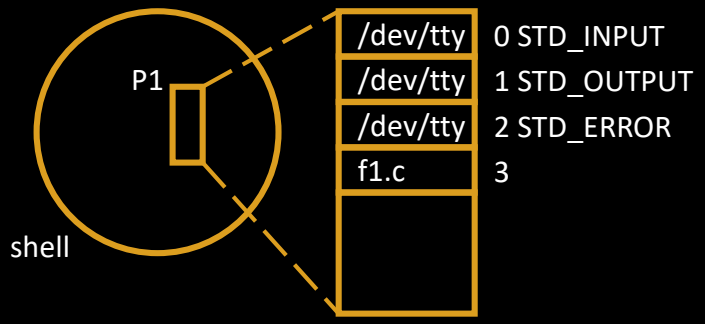
# JERARQUÍA Y HERENCIA

- Se establece una jerarquía similar a la del sistema de ficheros.
- **init** es el **padre** de todos los procesos que se ejecutan en un sistema Unix. Es creado en el arranque del sistema.
- Cada **usuario es propietario de una rama del árbol** cuya raíz es el intérprete de órdenes.
- Dicha rama se ejecuta con el **entorno de usuario** gracias al mecanismo de herencia.



# DESCRIPTORES DE FICHEROS

- Todo proceso tiene una serie de descriptores de ficheros. Son los ficheros que ese proceso está utilizando.
- Todos los ficheros que tiene abierto un proceso se **registran en una tabla indexada** por los descriptores de fichero.
- Unix **define tres descriptores con un significado especial**, que representan la entrada/salida del proceso.
- Suelen estar asociados con el terminal:
  - **0 STD_INPUT:** entrada normal de datos.
  - **1 STD_OUTPUT:** salida normal de datos.
  - **2 STD_ERROR:** salida normal de errores.
- Pueden ser cambiados (redirigidos) a otros ficheros distintos.

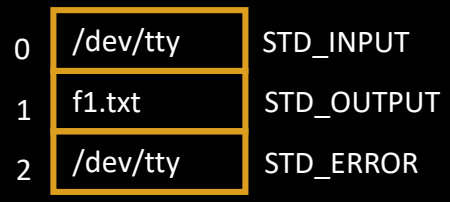



```
fd = open("f1.c", O_RDWR);
…
read(fd, &buffer, nbytes);
…
write(fd, &buffer, nbytes);
…
close(fd);
```

# REDIRECCIONES

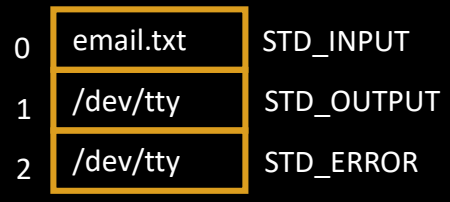
- **Redirección de la salida estándar:**

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



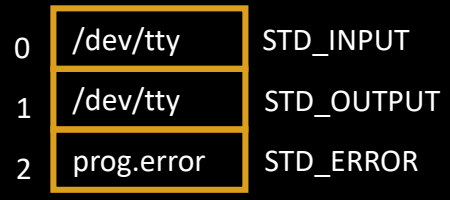

- **Redirección de la entrada estándar:**

```
$ mail rodrigo < email.txt
```




- **Redirección de la salida de error estándar:**

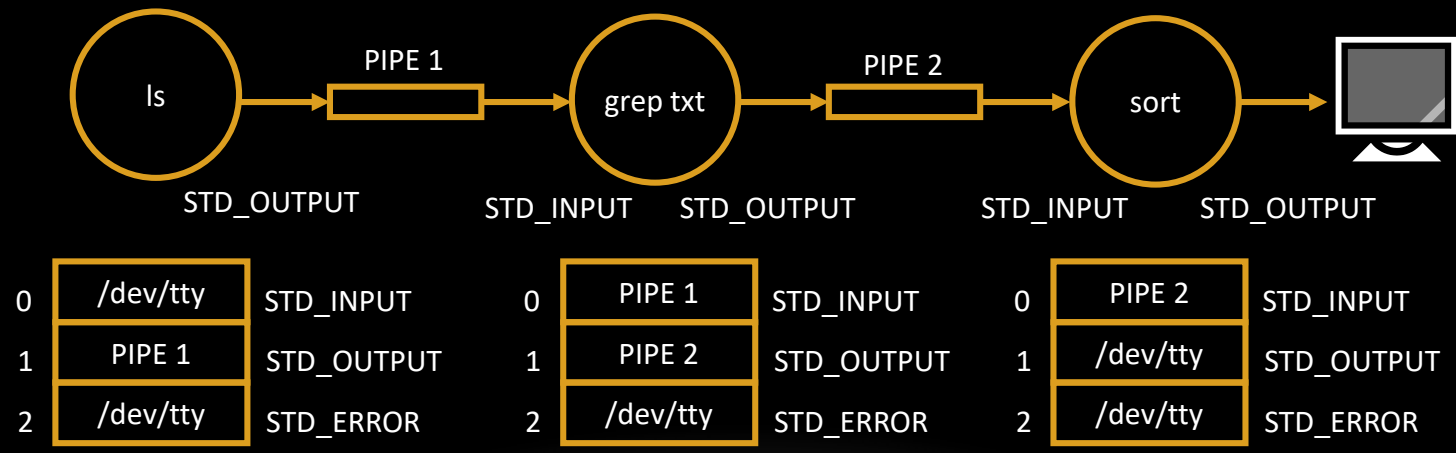
```
$ gcc prog.c 2> prog.error
```



# COMUNICACIÓN Y SINCRONIZACIÓN ENTRE PROCESOS

- La comunicación entre procesos en Unix se realiza mediante **tuberías (*pipes*)**.
- Comunican la salida de un proceso con la entrada de otro.
- 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 *pipes* puede compartirse gracias al mecanismo de herencia.

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



Protección

# CONCEPTOS BÁSICOS

# PROTECCIÓN

- Mecanismo para **controlar los accesos** que los procesos realizan a los recursos del sistema y de los usuarios (ficheros, memoria, dispositivos de E/S).
- **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 de permiso |

# BITS DE PERMISO E IDS

- **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
  - El **fichero** recibe los atributos del **proceso** que lo crea

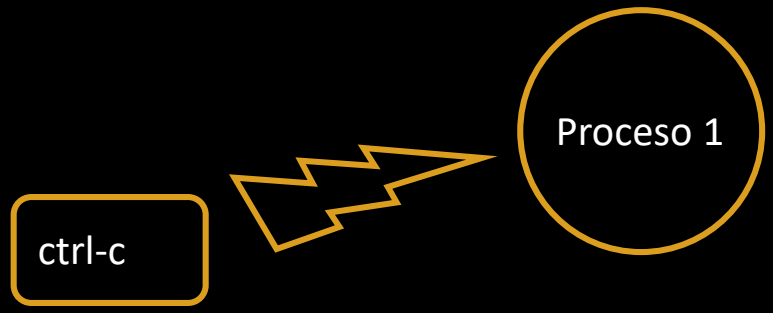
    ownerUID := efUID  ownerGID := efGID
- **Bits de permiso:**
  - 9 bits de permiso en grupos de tres: (propietario, grupo, otros).
  - Formatos: rwxr-xr-x, 0755
  - **Interpretación:**
    - **Ficheros regulares:** lectura, escritura, ejecución.
    - **Directorios:** listado, crear/eliminar enlaces, "búsqueda".
    - **Especiales:** lectura, escritura, --------
- **Reglas de protección:**
  - Si efUID = 0, se concede permiso (superusuario), si no…
  - 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 tercer grupo de bits.

## Señales

# CONCEPTOS BÁSICOS

# SEÑALES

- Evento que interrumpe la ejecución normal de un proceso.
- **Origen de la señal:**
  - El sistema operativo.
  - El propio proceso.
  - Otro proceso.
  - El usuario.



# SEÑALES EN UNIX

| Valor | Nombre | Descripción |
|---|---|---|
| 17 | SIGCHLD | Muerte de un hijo. |
| 02 | SIGINT | Interrumpir. Ctrl-C del teclado. |
| 11 | SIGSEGV | Violación de segmento; un proceso intenta acceder a una posición fuera de su espacio de direcciones virtual. |
| 04 | SIGILL | Instrucción ilegal. |
| 10 | SIGUSR1 | Señal 1 definida por el usuario. |
| 12 | SIGUSR2 | Señal 2 definida por el usuario. |
| 19 | SIGSTOP | Suspensión del proceso. |
| 15 | SIGTERM | Terminación del proceso. |
| 09 | SIGKILL | Matar. Terminación del proceso. No enmascarable ni manejable. |

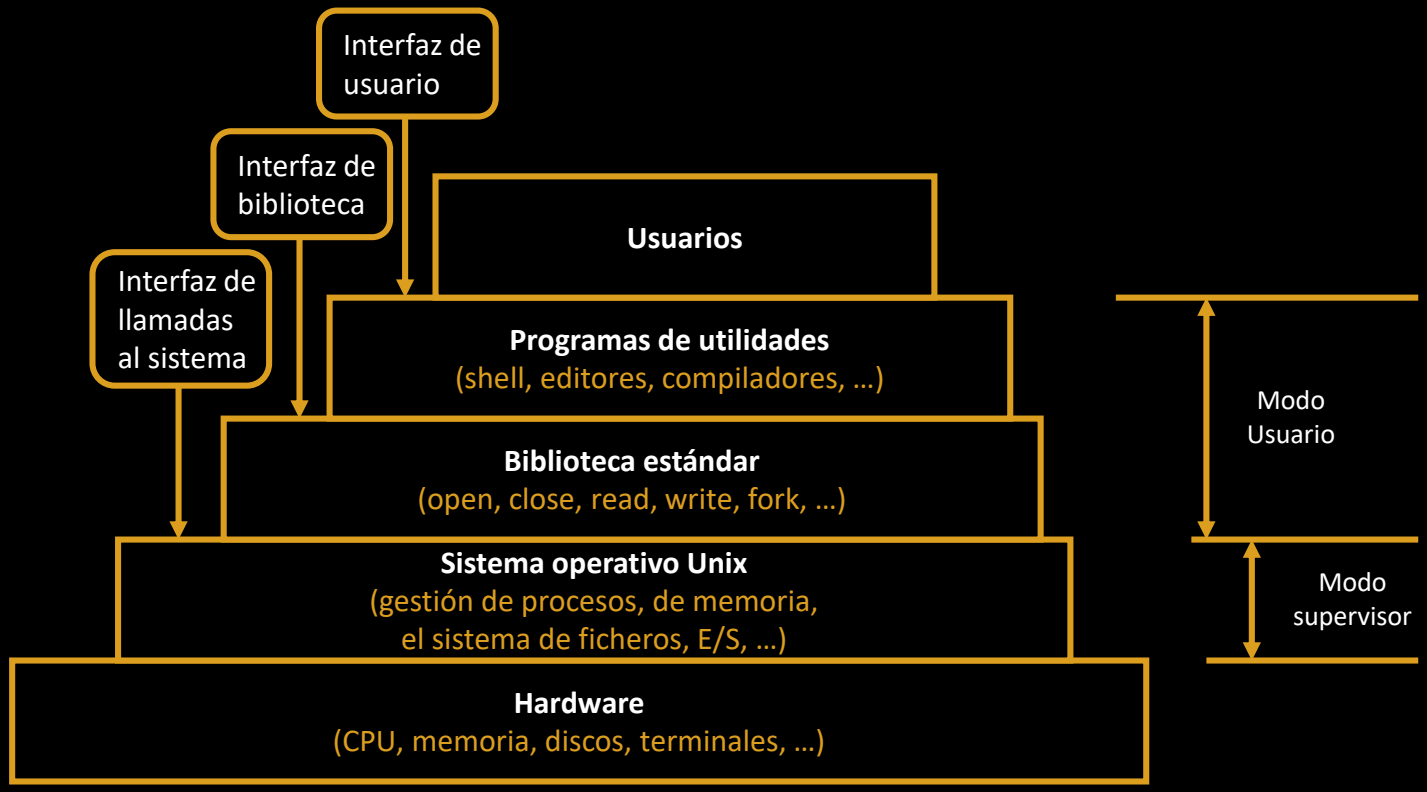
# TRATAMIENTO EN POSIX

- **POSIX permite que un proceso:**
  - Asuma el tratamiento **por defecto**.
  - **Ignore** la señal.
  - **Capture** la señal: ejecución del manejador asociado
  - **Enmascare** la señal:
    - Si ya se está tratando una señal, se evita tratar otras de menor prioridad.
- Dos señales han de ser **obedecidas siempre**:
  - SIGKILL.
  - SIGSTOP.

# SERVICIOS

# INTERFACES DE UN SISTEMA UNIX

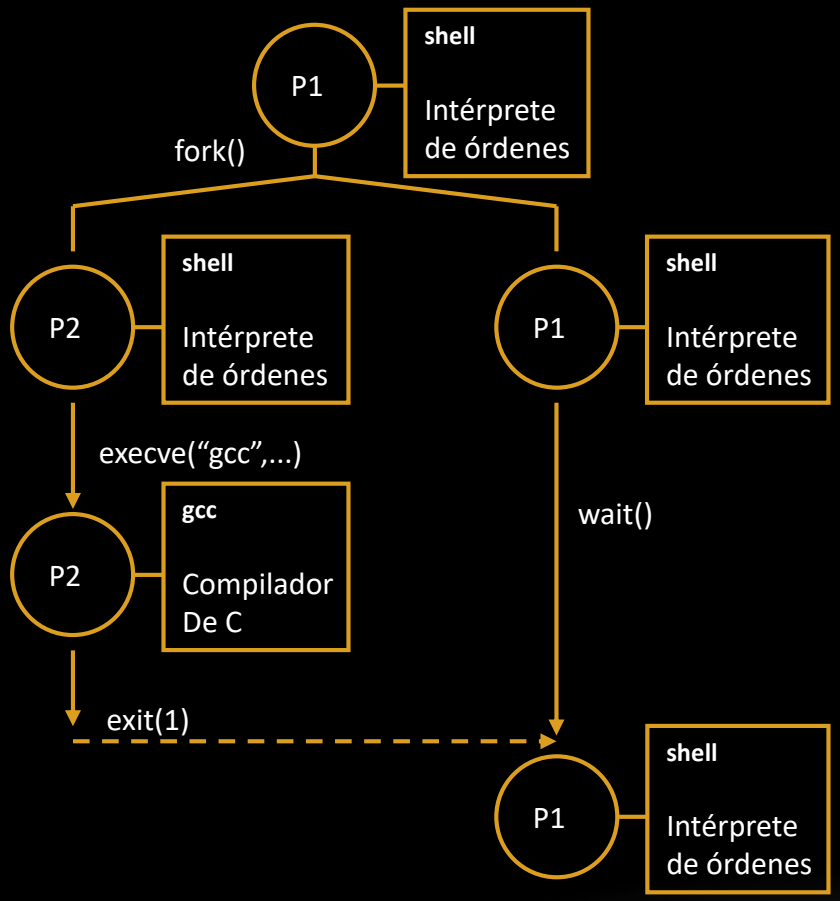
- Niveles de un sistema informático



# LLAMADAS A SISTEMA UNIX (I)

| Gestión de procesos | Descripción |
|---|---|
| pid = fork() | Crear un proceso hijo |
| s = waitpid(pid, status, opts) | Esperar finalización de un hijo |
| s = execv(name, argv) | Cambiar imagen de memoria |
| s = execvp(name, argv) | Cambiar imagen de memoria. Busca “name” en los directorios del PATH |
| S = execl(name, arg0, arg1, …, argn, NULL) | Cambiar imagen de memoria. Argumentos listados uno a uno. Terminar con un null pointer |
| 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 la señal |

# EJEMPLO DE USO DE *FORK* Y *EXECVE*




**El shell de Unix: estructura simple**

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

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

  if (pid != 0) {   /* código del padre*/

    /* esperar al hijo*/
    waitpid(-1, &status, 0);

  } else          /* código del hijo*/

    /* cambiar imagen de memoria*/
    execv(orden, params);

    exit(1);
  }
}
```

# PROGRAMA DE EJEMPLO: *FORKPROG*

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

/* Código de forkprog.c */
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 el PID de mi padre (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);
}
```

# EJECUCIÓN DE *FORKPROG*

```
servidor:/home/ssoo/progs> ./forkprog &
Soy el hijo, mi PID es 20874 y el PID de mi padre (PPID) es 20873
Soy el padre, mi PID es 20873 y el PID de mi hijo es 20874

servidor:/home/ssoo/progs> ps l
F   UID   PID  PPID PRI  NI   VSZ  RSS WCHAN  STAT TTY        TIME COMMAND
0     0 20842 20839  15   0  4888 1696 wait4  S    pts/0      0:00 -bash
0     0 20873 20842  15   0  1336  312 schedu S    pts/0      0:00 ./forkprog
0     0 20874 20873  15   0  1336  312 schedu S    pts/0      0:00 ./forkprog
0     0 20875 20842  21   0  3488 1484 -      R    pts/0      0:00 ps l

Final de ejecución de 20874

servidor:/home/ssoo/progs> ps l
F   UID   PID  PPID PRI  NI   VSZ  RSS WCHAN  STAT TTY        TIME COMMAND
0     0 20842 20839  16   0  4888 1696 wait4  S    pts/0      0:00 -bash
0     0 20873 20842  15   0  1336  312 schedu S    pts/0      0:00 ./forkprog
0     0 20874 20873  15   0     0    0 exit   Z    pts/0      0:00 [forkprog] <defunct>
0     0 20876 20842  21   0  3488 1484 -      R    pts/0      0:00 ps l

Final de ejecución de 20873
```

# PROGRAMA DE EJEMPLO: *EXECPROG*

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

/* Código de execprog.c */
int main(int argc, char *argv[])
{
    int i;
    printf("\nEjecutando el programa (%s). Sus argumentos son:\n",argv[0]);
    for (i = 0; i < argc; i++)
        printf("\targv[%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);
}
```

# EJECUCIÓN DE *EXECPROG*

```
servidor:/home/ssoo/progs> ./execprog ls -l &

Ejecutando el programa (./execprog). Sus argumentos son:
        argv[0]: ./execprog
        argv[1]: ls
        argv[2]: -l

total 4
-rwxr-xr-x    1 root     root         11592 Oct 21 09:07 execprog
-rw-r--r--    1 root     root           551 Oct 21 09:07 execprog.c
-rwxr-xr-x    1 root     root         11907 Oct 21 08:49 forkprog
-rw-r--r--    1 root     root           677 Oct 21 08:49 forkprog.c
```

# OTRAS LLAMADAS A SISTEMA UNIX

| Gestión de ficheros | Descripción |
|---|---|
| fd = creat(name, mode) | Crear un fichero |
| fd = open(name, mode) | Abrir un fichero para lectura y/o escritura |
| s = close(fd) | Cerrar un fichero abierto |
| n = read(fd, buffer, nbytes) | Leer de un descriptor de fichero sobre un buffer |
| n = write(fd, buffer, nbytes) | Escribir de un buffer sobre un un descriptor de fichero |
| pos = lseek(fd, offset, nbytes) | Posicionar el puntero |
| s = stat(name, buffer) | Obtener atributos de un fichero de su inode |
| 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 |

# PROGRAMAS DE SISTEMA

- Utilidades del sistema operativo que se ejecutan como **procesos de usuario** y proporcionan un entorno más cómodo.
- Son **programas** escritos en un lenguaje de programación (como C) que realizan **llamadas al sistema**.
- **Clasificación:**
  - **Tratamiento de ficheros y directorios:** mkdir, cp, mv, ls…
  - **Filtros:** sort, grep, head, tail, …
  - **Editores, compiladores, ensambladores, editores de enlace…**
  - **Sistema de ventanas:** X11
  - **Comunicaciones:** mail, ftp, rlogin, ssh, …
  - **Intérprete de órdenes:** sh, ksh, bash, …

# EL INTÉRPRETE DE ÓRDENES

# EL INTÉRPRETE DE ÓRDENES

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

    $ prog –optn

    $ ls –al

    $ ps –eaf

    $ cat f1 f2
  - **Variables / órdenes internas:**

    $ PATH=$PATH:$HOME/bin

    $ unset PATH

    $ export PATH

    $ echo $PATH

# EJEMPLOS DE USO DE LA *SHELL*

- **Redirecciones**

  $ cat f1 > f2

  $ echo hola > f2

  $ mail rgc@ceu.es < eniac.txt

- **Pipes**

  $ cat f1 f2 |more

  $ cat f1 f2 |grep hola | wc > fichero

- **Ejecución en background**

  $ gcc –Wall –c f1.c f2.c f3.c &

  $ (inmediato)

- **Control de flujo/parámetros**

```
$ for i in sso*
> do
> cp $i /home/a12345
> 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 copiado
fi
```

# NÚCLEO DE UNIX CLÁSICO