

Universidad de Deusto

Llamadas al sistema Minix [Explicación]

Universidad de Deusto
Facultad de Ingeniería - ESIDE

Deustuko Unibertsitatea
Injeneritza Fakultatea - ESIDE

Universidad de Deusto

- **Introducción**
 - Uso de llamadas al sistema en Minix
 - de forma transparente al usuario
 - de forma directa desde programas en C (funciones)
 - durante la ejecución de una llamada se actualiza variable "errno"
 - después de la llamada, comprobar que no ha habido error
- **Tipos de llamadas**
 - Gestión de procesos
 - Señales
 - Gestión de ficheros
 - *Gestión de ficheros y directorios*
 - *Protección*
 - *Gestión del tiempo*

Sistemas Operativos

Universidad de Deusto

- Gestión de procesos (I)
 - Introducción
 - Un proceso consta de:
 - Código o instrucciones a ejecutar
 - Espacio de direcciones asociado (Memoria)
 - Información necesaria para ejecutarse (Entorno: PC, SP, ...)
 - PCB: representación de un proceso en el sistema
 - Cada proceso tiene un UID, el del usuario que lo lanza
 - Los procesos hijo tienen el mismo UID que su padre

Sistemas Operativos

Universidad de Deusto

- Gestión de procesos (II)
 - Creación de procesos ("fork")
 - Crea un proceso hijo que es la réplica EXACTA del que hizo la llamada, pero con diferente espacio de direcciones.
 - Única manera de crear procesos
 - Si todo va bien, devuelve:
 - al proceso hijo, un "0"
 - al proceso padre, el PID del hijo

```
int idhijo;  
if ( ( idhijo=fork() ) != 0 )  
    { /* Código del Padre */ }  
else  
    { /* Código del Hijo */ }
```

- No se puede crear un hijo que haga una cosa distinta al padre, pero sí se puede modificar la ejecución del proceso hijo.

Sistemas Operativos

Universidad de Deusto

- Gestión de procesos (III)
 - Cambiar ejecución de un proceso hijo (“execve”)
 - Se cambia el código que el proceso hijo tiene en memoria:
 - `execve(nomF, argv, envP)`
 - *nomF*: nombre del programa con el código a cargar en memoria
 - *argv*: puntero al vector de parámetros de ese programa
 - *envP*: puntero a un vector de pares “nombre=valor” que representan los valores de las variables de entorno

```
if ( ( idhijo=fork() ) != 0 )
    { /* Código del Padre */ }
else
    { /* Código del Hijo */
      execve( comando, parametros, NIL_PTR );
    }
```

Sistemas Operativos

Universidad de Deusto

- Gestión de procesos (IV)
 - Fin de un proceso (“exit”)
 - Invocada por un proceso hijo
 - Devuelve a su padre un valor entre 0 y 255 en una variable llamada estado (p. ej.)

```
int estado;

if ( ( idhijo=fork() ) != 0 )
    { /* Código del Padre */ }
else
    { /* Código del Hijo */
      exit( estado );
    }
```

- ¿Cómo espera el proceso padre a que acabe un proceso hijo?

Sistemas Operativos

Universidad de Deusto

- Gestión de procesos (V)
 - Espera a que finalice un proceso hijo ("wait")
 - Detecta el fin de un proceso cualquiera
 - Devuelve el PID del proceso que acaba de finalizar

```
while ( true ) {  
    leeComando( comando, parametros );  
    if ( ( idhijo=fork() ) != 0 )  
    {  
        while ( wait( &estado ) != idhijo ) /* nada */;  
    }  
    else  
    {  
        execve( comando, parametros, NIL_PTR );  
        exit( estado );  
    }  
}
```

Sistemas Operativos

Universidad de Deusto

- Gestión de procesos (VI)
 - Inciso:
 - Jerarquía de procesos: por creación y destrucción de procesos
 - Comunicación entre procesos: señales, mensajes y compartición de memoria (esta última no en Minix)
 - Aumentar memoria ocupada por un proceso ("brk")
 - Espacio de direcciones de un proceso

Segmento de Pila
Segmento de Datos
Segmento de Código

- Da error si se intenta aumentar el segmento de datos y éste se solapa con el segmento de pila (que casi siempre crece)
brk(dir. final del segmento de datos)
sbrk(nº bytes que queremos añadir)

Sistemas Operativos

Universidad de Deusto

- Señales (I)
 - Interrupciones SW que se usan para:
 - que un proceso pueda interrumpir a otros
 - indicar a otros procesos que se ha detectado un error por HW (lo detecta el núcleo y avisa con señales al resto de capas del SO)
 - Recepción de señales:
 - Cuando un proceso recibe una señal, muere
 - Para evitarlo, el proceso usa la llamada "signal":

signal(TIPO_SEÑAL, TRATAMIENTO)

- solo vale para una vez
- captura la señal recibida, evita la muerte del proceso
- salva el Entorno Volátil (EV) del proceso
- ejecuta el tratamiento que se le pasa como parámetro
- recupera el EV del proceso, para dejarlo como estaba

Sistemas Operativos

Universidad de Deusto

- Señales (II)
 - Parámetros de la "signal"
 - TIPO_SEÑAL: nº de señal
 - TRATAMIENTO: puntero a función que trata la señal

Para estos parámetros se pueden usar ctes. definidas en "signal.h"

- Tipo: SIGINT, SIGQUIT, SIGPIPE, SIGALRM, SIGTERM, SIGKILL
- Tratamiento: SIG_IGN (ignorar la señal)

- Si el proceso vuelve a recibir la señal, se aplica el tratamiento por defecto -> muere -> ¿Solución?

```
.....
signal( SIGTERM, tratamiento );
.....
/* llega SIGTERM y se ejecuta el tratamiento */
.....
/* llega SIGTERM y muere el proceso */
```

```
void tratamiento() {
  printf( "Ha llegado SIGTERM" );
  signal( SIGTERM, tratamiento );
}
```

Sistemas Operativos

Universidad de Deusto

- Señales (III)
 - Envío de señales:
 - llamada "kill" (solo para procesos con igual UID)

kill(PID_DESTINO, TIPO_SEÑAL)

- Se hace desde un proceso
- Si proceso con PID_DESTINO no ha hecho signal, morirá
- llamada "alarm"

alarm(SEGUNDOS)

- un proceso pide que le envíen una señal de alarma (SIGALRM)
- la recibe al cabo de cierto tiempo (especificado en la función)
- "alarm" devuelve el tiempo que falta para que se dé la señal

Sistemas Operativos

Universidad de Deusto

- Señales (IV)
 - llamada "alarm"
 - hacer "signal" (capturar la señal) antes de hacer "alarm"

```
.....
alarm( 1 );
signal( SIGALRM, tratamiento );
.....
/* Si llega SIGALARM antes de que se ejecute
la instrucción signal, el proceso muere. MAL */
```

```
.....
signal( SIGALRM, tratamiento );
alarm( 1 );
.....
/* Se ejecuta "alarm" y el proceso no se para.
Sigue ejecutándose hasta que llegue la señal.*/
```

- cada proceso solo puede tener una señal de alarma pendiente

```
.....
signal( SIGALRM, tratamiento );
alarm( 5 );
alarm( 3 );          /* MAL */
.....
```

- alarm(0) -> anula todas las señales de alarma pendientes

Sistemas Operativos

Universidad de Deusto

- Señales (V)
 - llamada "pause"
 - bloquea a un proceso hasta que reciba una señal

```
while ( true ) {  
    signal( SIGALRM, SIG_IGN );  
    alarm( n );  
    pause();  
    /* acción a realizar */  
}  
/* Ejecuta acción cada n segundos */
```

– A tener en cuenta:

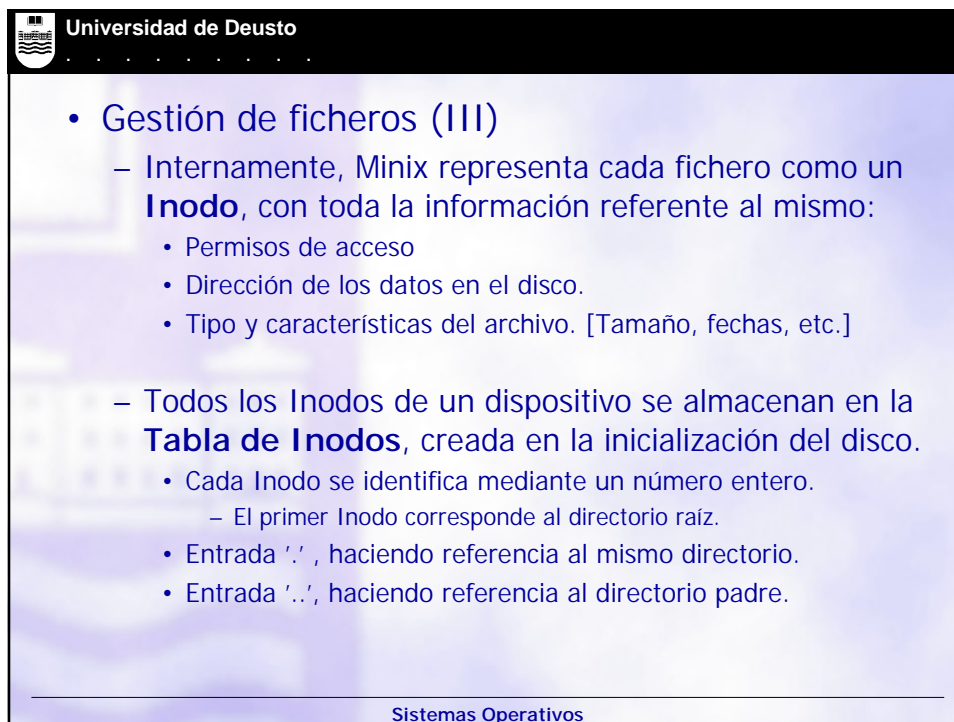
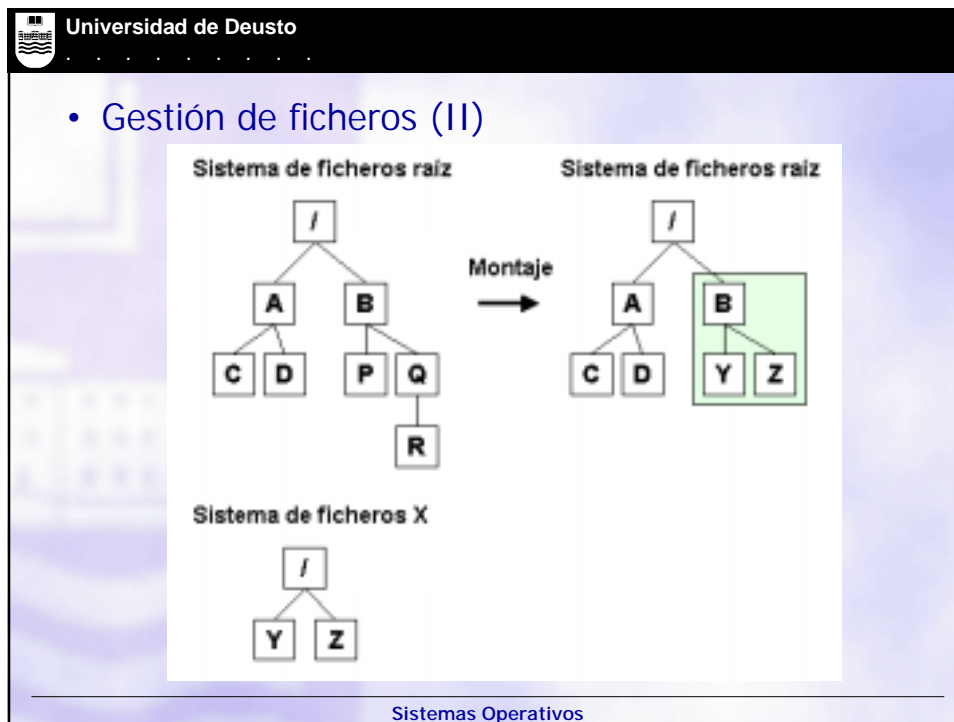
```
signal( SIGTERM, tratamiento );  
if ( ( idhijo=fork() ) != 0 )           // El hijo también tiene las "signal" del  
{ /* Código del Padre */           } // padre, pero solo aquellas que están  
else                                   // antes de hacer la "fork"  
{ /* Código del Hijo */           }
```

Sistemas Operativos

Universidad de Deusto

- Gestión de ficheros (I)
 - **Sistema de ficheros:**
 - Estructura jerárquica de ficheros y directorios.
 - **Objetivo:**
 - Ocultar detalles del dispositivo de almacenamiento.
 - En el sistema Minix:
 - Un **único sistema de ficheros**, comienza en directorio raíz /
 - Reside en el dispositivo principal. [Partición del disco duro]
 - Pueden existir otros SSFF, **pero** deben ser montados en el SF raíz para su utilización.

Sistemas Operativos



Universidad de Deusto

- Gestión de ficheros (IV)
 - Inodo de Minix

Tipo de fichero
Permisos de acceso
Número de enlaces
Propietario
Grupo del propietario
Tamaño en bytes
Fechas de creación, último acceso y última modificación
Punteros directos a bloques de datos
Puntero indirecto simple
Puntero indirecto doble
Puntero indirecto triple

Sistemas Operativos

Universidad de Deusto

- Gestión de ficheros (V)
 - **Protección:** Permisos de acceso

tipo de archivo	propietario	grupo	otros
- d l b c	r w x	r w x	r w x

- En el momento de abrir el fichero (lectura/escritura) se comprueban estos permisos y se permite el acceso o no.

- **Procesos:**
 - Cada proceso tiene asociado un directorio de trabajo.
 - Cada proceso tiene su conjunto de descriptores de fichero.
 - Ficheros especiales que se abren en el arranque del sistema:
 - Entrada estándar (teclado): df = 0
 - Salida estándar (pantalla): df = 1
 - Salida de error (pantalla): df = 2

Sistemas Operativos

Universidad de Deusto


- Gestión de ficheros (VI)
 - Creación de un fichero:
`df = creat(nombre, permisos)`
 - Si ya existe el fichero, lo sobrescribe.
 - Además de crearlo, lo deja abierto para su escritura. [df]
 - Apertura de un fichero:
`df = open(nombre, modo)`
 - modo: [0:lectura, 1:escritura, 2:lectura/escritura]
 - Antes de utilizar un fichero es necesario abrirlo.
 - Un fichero puede ser utilizado por varios procesos, pero sólo se abre una vez (aunque haya varias *open*). Simplemente se asigna un *df* para cada proceso que abrió el fichero.

Sistemas Operativos

Universidad de Deusto


- Gestión de ficheros (VII)
 - Lectura/Escritura de un fichero:
`nBytes = read(df, &buff, nBytes)`
`nBytes = write(df, &buff, nBytes)`
 - Ambas devuelven el nº de bytes transferidos:
 - Si se devuelve un nº negativo, **error**.
 - Si devuelve 0, no se ha leído ni escrito nada.
 - `lseek(df, desplazamiento, origen)`
 - **origen**: inicio, posición actual o final de fichero.
 - Permite al programador posicionarse en un punto del fichero.

Sistemas Operativos

 Universidad de Deusto


- Gestión de ficheros (VIII)
 - Cierre de un fichero:
`close(df)`
 - Para romper la asociación del fichero con el descriptor
 - Creación de inodos:
`df = mknod(nombre, permisos, dirección)`
 - Sólo puede ser invocada por el super-usuario.
 - Lectura de información de inodos:
`stat(nombre, &buffer)`
`fstat(df, &buffer)`
 - Consultan la información contenida en el inodo del fichero especificado por *nombre* o por *df*.
 - El campo *buffer* está definido en el fichero "*stat.h*"

Sistemas Operativos

 Universidad de Deusto


- Gestión de ficheros (IX)
 - Duplicación de descriptores de fichero:
`df2 = dup(df1)`
 - Es posible asignar más de un descriptor de fichero para el mismo fichero.
 - `dup()` devuelve el descriptor de fichero (un entero) más bajo que no esté siendo utilizado. Esta circunstancia permite, por ejemplo, redireccionar la entrada y salida estándar, utilizando las instrucciones adecuadas, ya que:
 - `printf()` escribe en el fichero con `df=1`.
 - `scanf()` lee del fichero con `df=0`.

Sistemas Operativos

 Universidad de Deusto

- Gestión de ficheros (X)
 - **Comunicación de procesos mediante pipes:**
 - La creación de un pipe implica la creación de un fichero temporal, que será utilizado por dos procesos para comunicarse.
 - pipe(a)
 - Donde 'a' es un array de enteros de dos posiciones en el que se devolverán un descriptor de lectura (a[0]) y otro de escritura (a[1]).
 - Cada proceso tiene sus propios descriptores de fichero.
 - Un pipe puede utilizarse en los dos sentidos, aunque es habitual usarlo sólo en uno.
 - Es necesario cerrar siempre los ficheros después de usarlos.

Sistemas Operativos

 Universidad de Deusto

- Gestión de ficheros (XI)
 - **Llamada ioctl()**
ioctl(df, operación, argumentos)
 - Esta llamada al sistema se utiliza para trabajar con dispositivos especiales de caracteres (terminales).
 - En función del tipo de dispositivo, podrán realizarse unas operaciones u otras, y con unos argumentos determinados.
 - En el fichero "sgtty.h" se definen todas las operaciones que se pueden hacer sobre los terminales.

Sistemas Operativos