

SISTEMAS OPERATIVOS: INTRODUCCIÓN Y CONCEPTOS BÁSICOS

Servicios del Sistema Operativo

ADVERTENCIA

2

- Este material es un simple guión de la clase: no son los apuntes de la asignatura.
- El conocimiento exclusivo de este material no garantiza que el alumno pueda alcanzar los objetivos de la asignatura.
- Se recomienda que el alumno utilice los materiales complementarios propuestos.

Objetivos

*Entender las llamadas
al sistema.*

3

- Comprender qué es un servicio del sistema operativo.
- Comprender los mecanismos que intervienen en una llamada al sistema.
- Conocer las características de la interfaz POSIX.
- Conocer los principales servicios ofrecidos por POSIX.

Ejecución del sistema operativo

4

- Una vez finalizado el arranque, el sistema operativo solamente se ejecuta en respuesta a interrupciones.

- El sistema operativo se activa cuando debe responder a:

- Una petición de servicio de un proceso.

→ $\frac{1}{p}d \sim$ daemon = demonio

- Una interrupción (periférico o reloj).

- Excepción hardware.

Están sobre el sistema operativo
no afectan a la máquina.

virus: rootkit, entra
en el kernel y tiene
acceso a todas las
instrucciones
y puede hacer lo
que quiera

Fases en la activación del Sistema Operativo

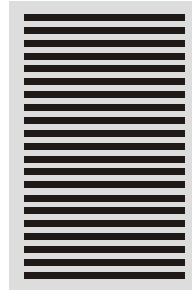
5

Avanza la ejecución ↓

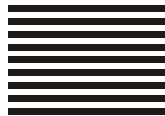
Proceso A



Sistema operativo



Proceso B



Lo primero que ejecuta es init y a raíz de esto van apareciendo más procesos que van saliendo de él. Como un árbol.

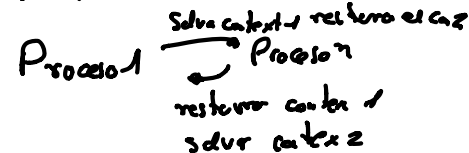


Se solicita el SO

Salva el estado del proceso A

Realiza la función pedida

Al cambiar de proceso debe encontrarse tal y como estaba antes del cambio. Eso es el contexto, almacena como estaba antes del cambio.



Se aprovecha los tiempos de espera para cambiar de proceso y ejecutar mientras tanto. Lo controla el planificador.
Concurrencia: ↗ Multiplexar

Todo proceso tiene un contexto

Activación de servicios

Lo primero es pasar de modo usuario a modo núcleo. En la llamada al SO puede hacer lo que quiera. Ese cambio de privilegio es gracias a las bibliotecas que tienen acceso a superusuario.

6

- Una invocación directa a una rutina del sistema operativo plantea problemas de seguridad.
 - ¿Cómo realizar cambios en el modo de ejecución de modo seguro?
- Utilizando una interrupción software (trap) se consigue la activación del sistema operativo de modo seguro.
 - Rutina de biblioteca:
 - Instrucciones de máquina que prepara la llamada al SO.
 - Instrucción de trap.
 - Instrucciones de proceso posterior de los resultados de la llamada al sistema operativo.

Servicios del sistema operativo:

Llamadas al sistema

7

- Interfaz entre aplicaciones y SO.
 - ▣ Generalmente disponibles como funciones en ensamblador.
 - ▣ Actualmente en otros lenguajes de alto nivel (C, C++, ...).
- Servicios típicos del sistema operativo
 - ▣ Gestión de procesos
 - ▣ Gestión de procesos ligeros *Si se crea, que se pueda consultar y matar*
 - ▣ Gestión de señales, temporizadores *Avisos y automatizaciones*
 - ▣ Gestión de memoria *← Ya lo veremos. →*
 - ▣ Gestión de ficheros y directorios
- Ejemplos de llamada
 - ▣ read: permite leer datos de un fichero
 - ▣ fork: permite crear un nuevo proceso

include <stdio.h>
4 header
declaración
de cabeceras.
biblioteca: Agrupación de funciones
completas.

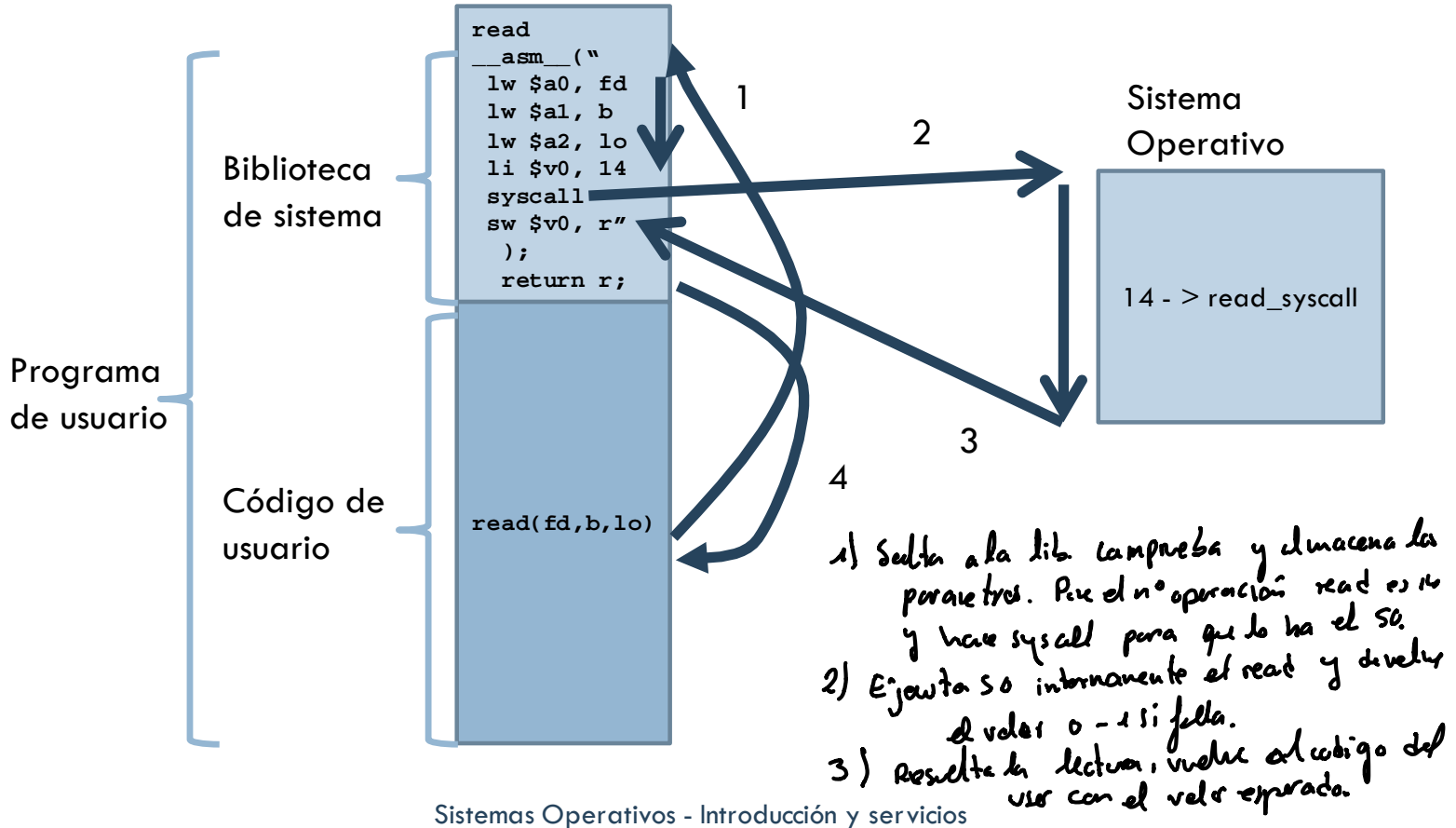
Invocación de la llamada

8

- Cada función de la interfaz de programación (API) se corresponde con algún servicio del sistema operativo.
 - ▣ La función es un envoltorio para el código que invoca el servicio del sistema operativo.
- Incluye la ejecución de una instrucción de trap que transfiere el control al sistema operativo mediante la generación de una interrupción.
- El sistema operativo trata la interrupción y devuelve el control al programa de usuario.

Invocación de la llamada

9



Selección de servicio

10

- Al existir una única instrucción de *trap* y múltiples servicios se hace necesario establecer algún mecanismo de paso de parámetros entre el proceso de usuario y el núcleo.
- Como mínimo siempre se debe pasar una especificación del servicio que se desea ejecutar.
 - ▣ Típicamente un identificador numérico.

Paso de parámetros

11

- Tres métodos genéricos para pasar parámetros a las llamadas al sistema:
 - En registros.
 - En una tabla de memoria, cuya dirección se pasa al SO en un registro.
 - Poner los parámetros en la pila del programa y dejar que el SO los extraiga.
- Cada SO proporciona sus propias llamadas al sistema:
 - Estándar POSIX en UNIX y LINUX.
 - Win32 en Windows NT.

Rutina de tratamiento

12

- La rutina de tratamiento debe:
 - ▣ Recuperar los parámetros enviados por el proceso de usuario.
 - ▣ Identificar el servicio que se desea ejecutar.
 - ▣ Determinar la dirección de la rutina de servicio adecuada (indexación en una tabla de rutinas de servicio).
 - ▣ Transferir el control a la rutina de servicio.

Invocación de llamada

(Minor after defect)

13

```
int read(int fd, char * b, int lon) {  
    int r;  
    asm(  
        lw $a0, fd  
        lw $a1, b  
        lw $a2, lon  
        li $v0, 14  
        syscall  
        sw $v0, r  
    );  
    return r;  
}
```

do
barros

Metr
ensamblador
en C

READ_SYSCALL

TRAP

Interfaz del programador

14

- Esta interfaz ofrece la visión que como máquina extendida tiene el usuario del sistema operativo
- Cada sistema operativo puede ofrecer una o varias interfaces:
 - Linux: POSIX
 - Windows: Win32, POSIX

Estándar POSIX

↳ portable
↳ open source

no system
no interface
~ for x

15

- ❑ Interfaz estándar de sistemas operativos de IEEE.
- ❑ **Objetivo:** portabilidad de las aplicaciones entre diferentes plataformas y sistemas operativos.
- ❑ **NO** es una implementación. Sólo define una interfaz
- ❑ Diferentes estándares
 - ❑ 1003.1 Servicios básicos del SO
 - ❑ 1003.1a Extensiones a los servicios básicos
 - ❑ 1003.1b Extensiones de tiempo real
 - ❑ 1003.1c Extensiones de procesos ligeros
 - ❑ 1003.2 Shell y utilidades
 - ❑ 1003.2b Utilidades adicionales

UNIX03

16

- Single Unix Specification UNIX 03.
- Es una evolución que engloba a POSIX y otros estándares (X/Open XPG4, ISO C).
- Incluye no solamente la interfaz de programación, sino también otros aspectos:
 - Servicios ofrecidos.
 - Intérprete de mandatos.
 - Utilidades disponibles.

Características de POSIX

17

- Nombres de funciones cortos y en letras minúsculas:

- fork

no bifurcación binaria
↳ crea un proceso

Nombres representativos en inglés
solen 2 de cada fork kill

- read

terminar, no puede read

- close

open close
for(i=0; i < n; i++)
2¹⁰ ↳ Arbol profundidad 10.

- Las funciones normalmente devuelve 0 ^{éxito} en caso de éxito o -1 en caso de error.

- Variable errno. no error number / perror(errno) no dice de que es el error.

- Recursos gestionados por el sistema operativo se referencian mediante descriptores (números enteros)

Ejemplo: Ejecución de un mandato

Lo que hace el shell cada vez

18

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

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

```
int main(int argc, char** argv)
```

```
    pid_t pid;
```

```
    pid = fork();
```

```
    switch (pid) {
```

```
        case -1: /* error */
```

```
            exit(-1);
```

```
        case 0: /* proceso hijo */
```

```
            if (execvp(argv[1], &argv[1]) < 0) { perror("error"); }
```

```
            break;
```

```
        default:
```

```
            printf("Proceso padre");
```

```
            return 0;
```

```
    }
```

Shell

Relación jerárquica

Shell

~~Shell~~

prog cat fl

Comando
argumentos

argumento
argumentos

identificación
de proceso

El fork le
hace al hijo
el pid=0
y el padre tendrá
otro y pasará
a default

El padre tiene
pid != 0 y es
para el hijo

Le hace el padre
Duplica shell, pero el padre
e hijo funcionan a la vez.

del proceso (shell en caso)
al ejecutar
~ Si error, muere el hijo y que da el padre para seguir ejecutando.
y dice que error es.
~ Si error, devuelve negativo
Si el hijo ejecuta la operación y no falla continuará.

¡Ombie! Proceso - ...

SIGEXIT el hijo envía
una señal al padre.

Cuando que da un fork, le adopta
el init que siempre está y entonces se
puede morir

Servicio fork

19

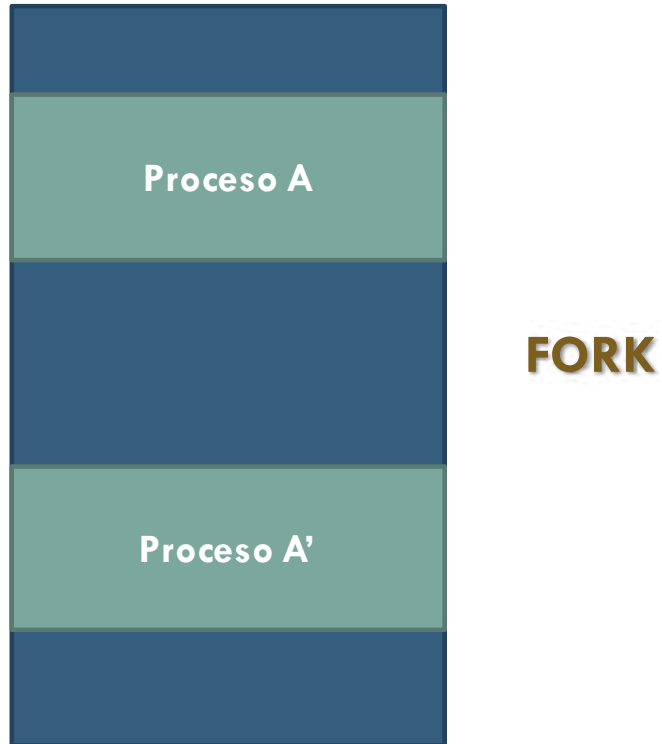
- `pid_t fork(void);`

- Duplica el proceso que invoca la llamada.
- El proceso padre y el proceso hijo siguen ejecutando el mismo programa.
- El proceso hijo hereda los ficheros abiertos del proceso padre.
 - Se copian los descriptores de archivos abiertos.
- Se desactivan las alarmas pendientes.

- Devuelve:
 - -1 el caso de error.
 - En el proceso padre: el identificador del proceso hijo.
 - En el proceso hijo: 0

Servicio fork

20



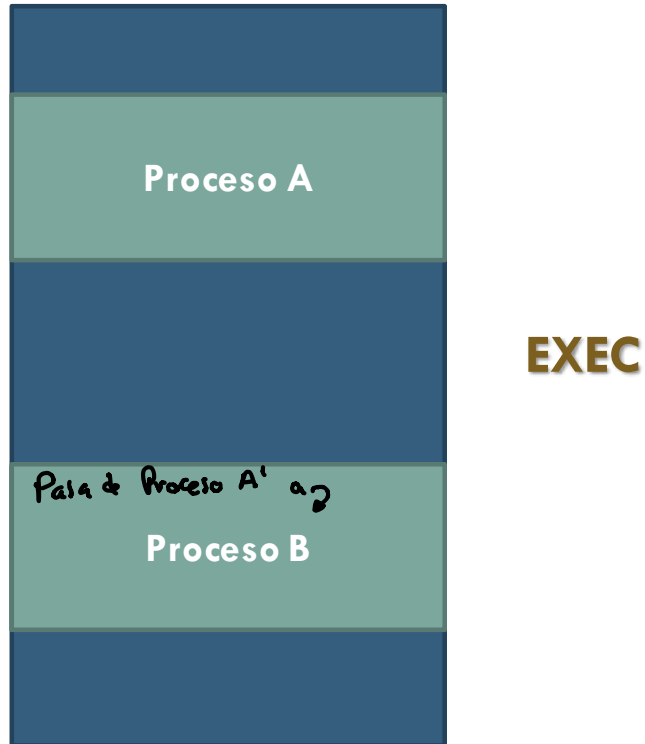
Servicio exec

21

- Servicio único pero múltiples funciones de biblioteca. *Siempre el primer argumento es el propio programa.*
 - *re argumentos en forma de lista*
`int execl(const char *path, const char *arg, ...);`
`int execv(const char* path, char* const argv[]);`
`int execve(const char* path, char* const argv[], char* const envp[]);`
`int execvp(const char *file, char *const argv[])`
- Cambia la imagen del proceso actual.
 - `path`: Ruta al archivo ejecutable.
 - `file`: Busca el archivo ejecutable en todos los directorios especificados por PATH.
- Descripción:
 - Devuelve -1 en caso de error, en caso contrario no retorna.
 - El mismo proceso ejecuta otro programa.
 - Los ficheros abiertos permanecen abiertos.
 - Las señales con la acción por defecto seguirán por defecto, las señales con manejador tomarán la acción por defecto.

Servicio fork

22



Servicio exit

23

- Finaliza la ejecución del proceso.

Escritor al final del programa

- `void exit(status);`

error negativos

correcto
falla

- Se cierran todos los descriptores de ficheros abiertos.
- Se liberan todos los recursos del proceso.
- Se libera el BCP del proceso.


Ejemplo: Ejecución de un mandato

24

```
#include <sys/types.h>
#include <stdio.h>

int main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)    { /* proceso hijo */
        execlp("ls","ls","-l",NULL);
        exit(-1);
    }
    else            /* proceso padre */
        printf("Fin del padre\n");
    return 0; /* Invoca a exit(0) */
}
```



Sistemas Operativos - Introducción y servicios

Operaciones genéricas sobre ficheros

Un fichero es una manera más persistente de almacenar información, para reutilizarla o ...

- **crear:** Crea un fichero con un nombre y unos atributos.
- **borrar:** Borra un fichero a partir de su nombre.
- **abrir:** Abre un fichero a partir de su nombre para permitir operaciones de acceso.
- **cerrar:** Cierra un fichero abierto.
- **leer:** Lee datos de un fichero abierto a un almacén en memoria.
- **escribir:** Escribe datos a un fichero abierto desde un almacén en memoria.
- **posicionar:** *salto a una posición del fichero.* Mueve el apuntador usado para acceder al fichero, afectando a operaciones posteriores. *nombre*
- **control:** Permite manipular los atributos de un fichero. *Consultar atributos de un fichero*

Servicios POSIX para ficheros

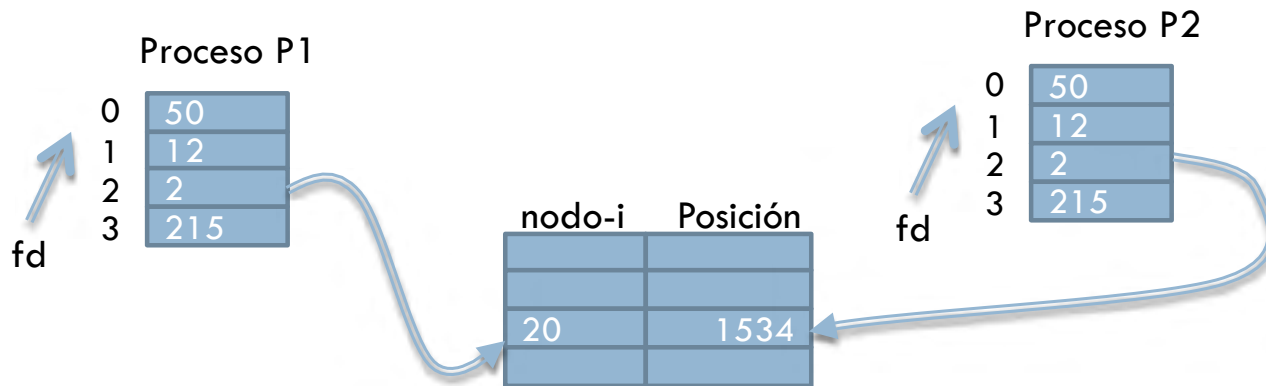
26

- **Visión lógica:**
 - Un fichero
- Se mantiene un puntero asociado a cada fichero abierto.
 - El puntero indica la posición a partir de la cual se realizará la siguiente operación.
- La mayor parte de las operaciones trabajan con descriptores de ficheros:
 - Un número entero entre 0 y 64K.
 - Se obtiene al abrir el fichero (open).
 - El resto de operaciones identifican el fichero por su descriptor.
- **Descriptores predefinidos:**
 - 0: entrada estándar
 - 1: salida estándar
 - 2: salida de error

Servicios POSIX para ficheros

27

- Cada proceso tiene asociada una tabla de ficheros abiertos.
- Cuando se duplica un proceso (fork):
 - ▣ Se duplica la tabla de archivos abiertos.
 - ▣ Se comparte la tabla intermedia de nodos-i y posiciones.



- **Protección:**
 - ▣ dueño grupo mundo
 - ▣ rwx rwx rwx
- **Ejemplos:** 755 indica `rw-r--r--`

Ficheros, directorios y servicios en POSIX

28

- Tipos de fichero:
 - ▣ Normales .
 - ▣ Directorios.
 - ▣ Especiales.
- Nombres de fichero y directorio:
 - ▣ Nombre completo (empieza por /)
 - `/usr/include/stdio.h`
 - ▣ Nombre relativo al directorio actual (no empieza por /)
 - `stdio.h` asumiendo que `/usr/include` es el directorio actual.
 - ▣ La entradas `.` y `..` pueden utilizarse para formar rutas de acceso
 - `../include/stdio.h`

CREAT – Creación de fichero

29

□ Servicio:

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

□ Argumentos:

- name Nombre de fichero
- mode Bits de permiso para el fichero

□ Devuelve:

- Devuelve un descriptor de fichero ó -1 si error.

CREAT – creación de fichero

30

□ Descripción:

- El fichero se abre para escritura.
- Si no existe crea un fichero vacío.
 - `UID_dueño = UID_efectivo`
 - `GID_dueño = GID_efectivo`
- Si existe lo trunca sin cambiar los bits de permiso.

□ Ejemplos:

```
fd = creat("datos.txt", 0751);  
fd = open("datos.txt",  
          O_WRONLY | O_CREAT | O_TRUNC, 0751);
```

UNLINK – Borrado de fichero

31

□ Servicio:

```
#include <unistd.h>  
int unlink(const char* path);
```

□ Argumentos:

- path nombre del fichero

□ Devuelve:

- Devuelve 0 ó -1 si error.

□ Descripción:

- Decrementa el contador de enlaces del fichero. Si el contador es 0, borra el fichero y libera sus recursos.

OPEN – Apertura de fichero

32

□ Servicio:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(char *name, int flag, ...);
```

□ Argumentos:

- name puntero al nombre del fichero
- flags opciones de apertura:
 - O_RDONLY Sólo lectura
 - O_WRONLY Sólo escritura
 - O_RDWR Lectura y escritura
 - O_APPEND El puntero de acceso se desplaza al final del fichero abierto
 - O_CREAT Si no existe no tiene efecto. Si no existe lo crea
 - O_TRUNC Trunca si se abre para escritura

Open – Apertura de fichero

33

□ Devuelve:

- Un descriptor de fichero ó -1 si hay error.

□ Ejemplos:

```
fd = open( "/home/juan/datos.txt" );
```

```
fd = open( "/home/juan/datos.txt",  
          O_WRONLY | O_CREAT | O_TRUNC, 0750 );
```

CLOSE – Cierre de fichero

34

- Servicio:

```
int close(int fd);
```

- Argumentos:

- fd descriptor de fichero

- Devuelve:

- Cero ó -1 si error.

- Descripción:

- El proceso pierde la asociación a un fichero.

READ – Lectura de fichero

35

□ Servicio:

```
#include <sys/types.h>
ssize_t read(int fd, void *buf, size_t n_bytes);
```

□ Argumentos:

- fd descriptor de fichero
- buf zona donde almacenar los datos
- n_bytes número de bytes a leer

□ Devuelve:

- Número de bytes realmente leídos ó -1 si error

□ Descripción:

- Transfiere n_bytes. Puede leer menos datos de los solicitados si se rebasa el fin de fichero o se interrumpe por una señal.
- Después de la lectura se incrementa el puntero del fichero con el número de bytes realmente transferidos.

WRITE – Escritura de fichero

36

□ Servicio:

```
#include <sys/types.h>
ssize_t write(int fd, void *buf, size_t n_bytes);
```

□ Argumentos:

- `fd` descriptor de fichero
- `buf` zona de datos a escribir
- `n_bytes` número de bytes a escribir

□ Devuelve:

- Número de bytes realmente escritos ó -1 si error

□ Descripción:

- Transfiere `n_bytes`. Puede escribir menos datos de los solicitados si se rebasa el tamaño máximo de un fichero o se interrumpe por una señal.
- Después de la escritura se incrementa el puntero del fichero con el número de bytes realmente transferidos.
- Si se rebasa el fin de fichero el fichero aumenta de tamaño.

LSEEK – Movimiento del puntero de posición

37

□ Servicio:

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

□ Argumentos:

- fd Descriptor de fichero
- offset desplazamiento
- whence base del desplazamiento

□ Devuelve:

- La nueva posición del puntero ó -1 si error.

□ Descripción:

- Coloca el puntero de acceso asociado a fd
- La nueva posición se calcula:
 - SEEK_SET posición = offset
 - SEEK_CUR posición = posición actual + offset
 - SEEK_END posición = tamaño del fichero + offset

FNCTL – Modificación de atributos

38

□ Servicio:

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

```
int fnctl(int fildes, int cmd /* arg*/ ...);
```

□ Argumentos:

- fildes descriptor de ficheros

- cmd mandato para modificar atributos, puede haber varios.

□ Devuelve:

- 0 para éxito ó -1 si error

□ Descripción:

- Modifica los atributos de un fichero abierto.

DUP – Duplicación de descriptor de fichero

39

□ Servicio:

```
int dup(int fd);
```

□ Argumentos:

- `fd` descriptor de fichero

□ Devuelve:

- Un descriptor de fichero que comparte todas las propiedades del `fd` ó -1 si error.

□ Descripción:

- Crea un nuevo descriptor de fichero que tiene en común con el anterior:
 - Accede al mismo fichero
 - Comparte el mismo puntero de posición
 - El modo de acceso es idéntico.
- El nuevo descriptor tendrá el menor valor numérico posible.

FTRUNCATE – Asignación e espacio a un fichero

40

□ Servicio:

```
#include <unistd.h>
int ftruncate(int fd, off_t length);
```

□ Argumentos:

- fd descriptor de fichero
- length nuevo tamaño del fichero

□ Devuelve:

- Devuelve 0 ó -1 si error.

□ Descripción:

- El nuevo tamaño del fichero es length. Si length es 0 se trunca el fichero.

STAT – Información sobre un fichero

41

□ Servicio:

```
#include <sys/types.h>
#include <sys/stat.h>
int stat(char *name, struct stat *buf);
int fstat(int fd, struct stat *buf);
```

□ Argumentos:

- name nombre del fichero
- fd descriptor de fichero
- buf puntero a un objeto de tipo struct stat donde se almacenará la información del fichero.

□ Devuelve:

- Cero ó -1 si error

STAT – Información sobre un fichero

42

□ Descripción:

- ▣ Obtiene información sobre un fichero y la almacena en una estructura de tipo `struct stat`:

```
struct stat {  
    mode_t  st_mode; /* modo del fichero */  
    ino_t   st_ino;  /* número del fichero */  
    dev_t   st_dev;  /* dispositivo */  
    nlink_t st_nlink; /* número de enlaces */  
    uid_t   st_uid;  /* UID del propietario */  
    gid_t   st_gid;  /* GID del propietario */  
    off_t   st_size; /* número de bytes */  
    time_t  st_atime; /* último acceso */  
    time_t  st_mtime; /* última modificación */  
    time_t  st_ctime; /* última modificación de datos */  
};
```

STAT – Información sobre un fichero

43

- Comprobación del tipo de fichero aplicado a `st_mode`:

`S_ISDIR(s.st_mode)` Cierto si directorio

`S_ISCHR(s.st_mode)` Cierto si especial de caracteres

`S_ISBLK(s.st_mode)` Cierto si especial de bloques

`S_ISREG(s.st_mode)` Cierto si fichero normal

`S_ISFIFO(s.st_mode)` Cierto si pipe o FIFO

UTIME – Alteración de atributos de fecha

44

□ Servicio:

```
#include <sys/stat.h>
#include <utime.h>

int utime(char *name, struct utimbuf *times);
```

□ Argumentos:

- name nombre del fichero
- times estructura con las fechas de último acceso y modificación.
 - time_t actime fecha de acceso
 - time_t mctime fecha de modificación

□ Devuelve:

- Devuelve 0 ó -1 si error

□ Descripción:

- Cambia las fechas de último acceso y última modificación según los valores de la estructura struct utimbuf

Ejemplo: Copia de un fichero en otro

45

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

#define BUFSIZE 512

main(int argc, char **argv) {
    int fd_ent, fd_sal;
    char buffer[BUFSIZE];
    int n_read;

    /* abre el fichero de entrada */
    fd_ent = open(argv[1],
                  O_RDONLY);
    if (fd_ent < 0) {
        perror("open");
        exit(-1);
    }

    /* crea el fichero de salida */
    fd_sal = creat(argv[2], 0644);
    if (fd_sal < 0) {
        close(fd_ent);
        perror("open");
        exit(-1);
    }
}
```

Ejemplo: Copia de un fichero en otro

46

```
/* bucle de lectura del fichero de entrada */
while ((n_read = read(fd_ent, buffer, BUFSIZE)) > 0) {
    /* escribir el buffer al fichero de salida */
    if (write(fd_sal, buffer, n_read) < n_read) {
        perror("write2");
        close(fd_ent); close(fd_sal);
        exit(-1);
    }
}

if (n_read < 0) {
    perror("read");
    close(fd_ent); close(fd_sal);
    exit(-1);
}
close(fd_ent); close(fd_sal);
exit(0);
}
```

Ejemplo: Redirección (ls > fichero)

47

```
void main(void) {  
    pid_t pid;  
    int status;  
    int fd;  
  
    fd = open("fichero", O_WRONLY|O_CREAT|O_TRUNC, 0644);  
    if (fd < 0)    {  
        perror("open");  
        exit(-1);  
    }  
    pid = fork();
```

Servicios POSIX para directorios

48

□ **Visión lógica:**

- Un directorio es un fichero con registros tipo estructura DIR
- Por tanto se pueden operar como un fichero, pero **!NO SE PUEDEN ESCRIBIR DESDE PROGRAMA, SOLO LEER!**

□ **Estructura DIR:**

- `d_ino;` // *Nodo_i*
- `d_off;` // *Posición en el fichero del elemento del directorio*
- `d_reclen;` // *Tamaño del directorio*
- `d_type;` // *Tipo del elemento*
- `d_name[0];` // *Nombre del fichero **de longitud variable***

- ¡Ojo! Al ser el nombre de longitud variable no se pueden manipular como registros de longitud fija
- Solución: llamadas al sistema para manejar directorios

Servicios POSIX para directorios

49

- `DIR *opendir(const char *dirname);`
 - ▣ Abre el directorio y devuelve un puntero al principio de tipo DIR
- `int readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result);`
 - ▣ Lee la siguiente entrada de directorio y la devuelve en una struct dirent
- `long int telldir(DIR *dirp);`
 - ▣ Indica la posición actual del puntero dentro del archivo del directorio
- `void seekdir(DIR *dirp, long int loc);`
 - ▣ Avanza desde la posición actual hasta la indicada en “loc”. Nunca saltos atras.
- `void rewinddir(DIR *dirp);`
 - ▣ Resetea el puntero del archivo y lo pone otra vez al principio
- `int closedir(DIR *dirp);`
 - ▣ Cierra el archivo del directorio

¿Objetivo cumplido?

50

- ❑ Comprender qué es un servicio del sistema operativo.
- ❑ Comprender los mecanismos que intervienen en una llamada al sistema.
- ❑ Conocer las características de la interfaz POSIX.
- ❑ Conocer los principales servicios ofrecidos por POSIX.

SISTEMAS OPERATIVOS: INTRODUCCIÓN Y CONCEPTOS BÁSICOS

Material complementario

Proyección en POSIX

52

- `void *mmap(void *direc, size_t lon, int prot, int flags, int fd, off_t desp);`
- Establece proyección entre espacio de direcciones de un proceso y un archivo.
 - ▣ Devuelve la dirección de memoria donde se ha proyectado el archivo.
 - ▣ `direc`: dirección donde proyectar. Si NULL SO elige una.
 - ▣ `lon`: especifica el número de bytes a proyectar
 - ▣ `prot`: Protección para la zona (se pueden combinar con |).
 - ▣ `flags`: Propiedades de la región.
 - ▣ `fd`: Descriptor del fichero que se desea proyectar en memoria.
 - ▣ `desp`: Desplazamiento inicial sobre el archivo.

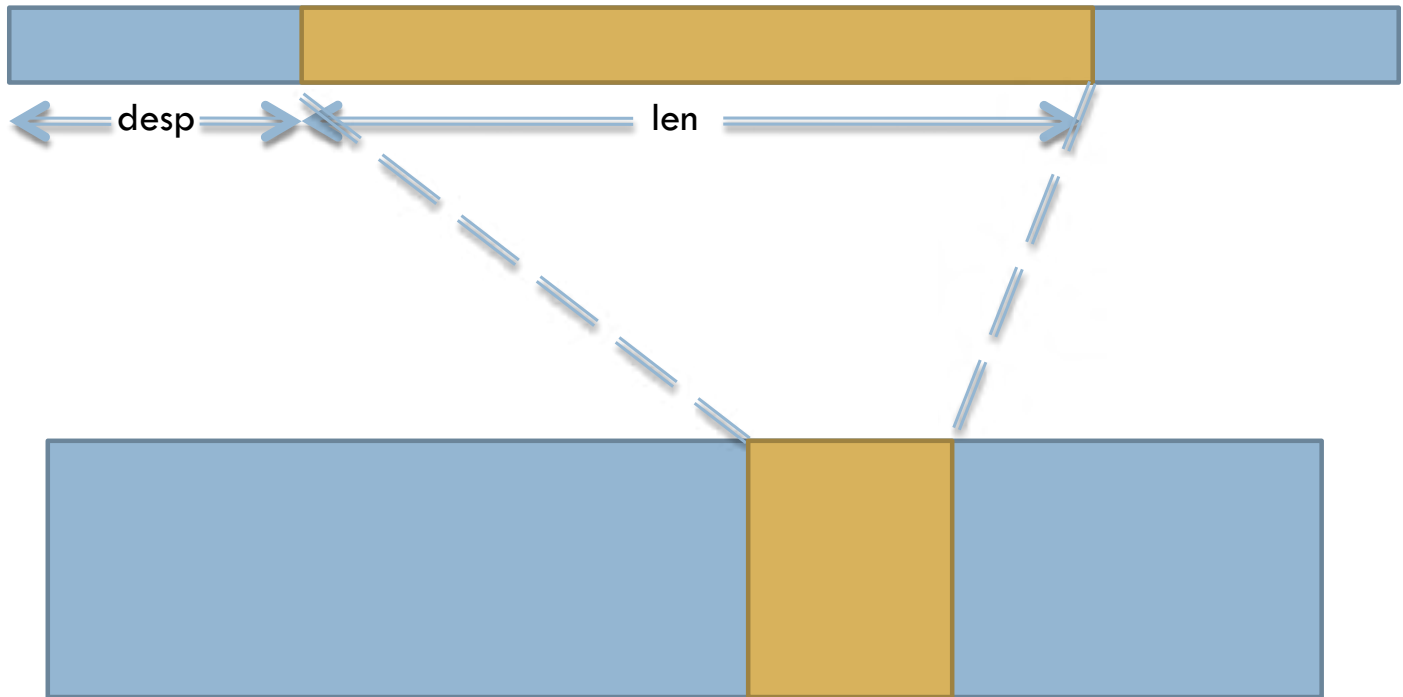
Proyección POSIX: mmap

53

- Tipos de protección:
 - ▣ PROT_READ: Se puede leer.
 - ▣ PROT_WRITE: Se puede escribir.
 - ▣ PROT_EXEC: Se puede ejecutar.
 - ▣ PROT_NONE: No se puede acceder a los datos.
- Propiedades de una región de memoria:
 - ▣ MAP_SHARED: La región es compartida. Las modificaciones afectan al fichero. Los procesos hijos comparten la región.
 - ▣ MAP_PRIVATE: La región es privada. El fichero no se modifica. Los procesos hijos obtienen duplicados no compartidos.
 - ▣ MAP_FIXED: El fichero debe proyectarse en la dirección especificada por la llamada.

Proyección POSIX

54



Proceso

Desproyección en POSIX

55

- `void munmap(void *direc, size_t lon);`
 - Desproyecta parte del espacio de direcciones de un proceso desde la dirección `direc` hasta `direc+lon`.

Ejemplo: Contar el número de blancos en un fichero

56

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

int main() {
    int fd;
    struct stat dstat;
    int i, n;
    char c,
    char * vec;

    fd = open("datos.txt",O_RDONLY);
    fstat(fd, &dstat);

    vec = mmap(NULL, dstat.st_size,
               PROT_READ, MAP_SHARED, fd, 0);
    close(fd);
    c =vec;
    for (i=0;i<dstat.st_size;i++) {
        if (*c==' ') {
            n++;
        }
        c++;
    }
    munmap(vec, dstat.st_size);
    printf("n=%d,\n",n);
    return 0;
}
```


Ejemplo: Copia de un fichero

57

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

int main() {
    int i, fd1, fd2;
    struct stat dstat;
    char * vec1, *vec2, *p, *q;

    fd1 = open("f1", O_RDONLY);
    fd2 = open("f2",
        O_CREAT|O_TRUNC|O_RDWR, 0640);
    fstat(fd1, &dstat);
    ftruncate(fd2, dstat.st_size);

    vec1 = mmap(0, bstat.st_size,
        PROT_READ, MAP_SHARED, fd1, 0);
    vec2 = mmap(0, bstat.st_size,
        PROT_READ, MAP_SHARED, fd2, 0);

    close(fd1); close(fd2);

    p = vec1; q = vec2;
    for (i = 0; i < dstat.st_size; i++) {
        *q++ = *p++;
    }

    munmap(fd1, bstat.st_size);
    munmap(fd2, bstat.st_size);

    return 0;
}
```