

SISTEMAS OPERATIVOS: INTRODUCCIÓN Y CONCEPTOS BÁSICOS



Servicios del Sistema Operativo

ADVERTENCIA

- ❑ Las transparencias ayudan como simple gui3n de la clase pero no son los apuntes de la asignatura.
- ❑ El conocimiento exclusivo de este material no garantiza que el/la estudiante pueda alcanzar los objetivos de la asignatura.
- ❑ Se recomienda que el/la estudiante utilice todos los materiales bibliogr3ficos propuestos para complementar los conocimientos.

Objetivos

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

Contenidos

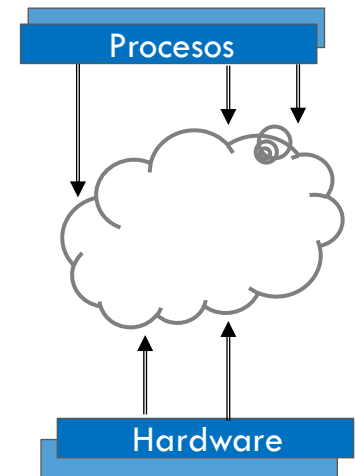
- Introducción a llamadas al sistema
- Mecanismo de llamada al sistema
- Llamadas para servicios de:
 - ▣ Gestión de procesos
 - ▣ Gestión de ficheros

Contenidos

- Introducción a llamadas al sistema
- Mecanismo de llamada al sistema
- Llamadas para servicios de:
 - ▣ Gestión de procesos
 - ▣ Gestión de ficheros

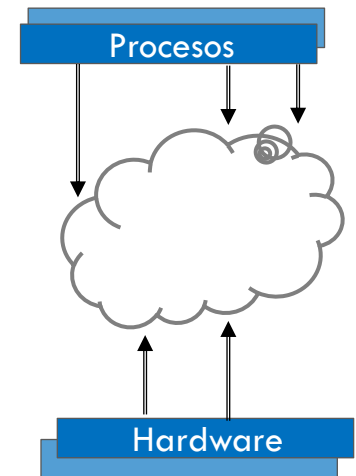
Ejecución del sistema operativo

- Durante el arranque.
- Una vez finalizado el arranque, se ejecuta en respuesta a eventos:
 - Llamada al sistema.
 - Excepción.
 - Interrupción hardware.
- En procesos de núcleo (firewall, etc.)



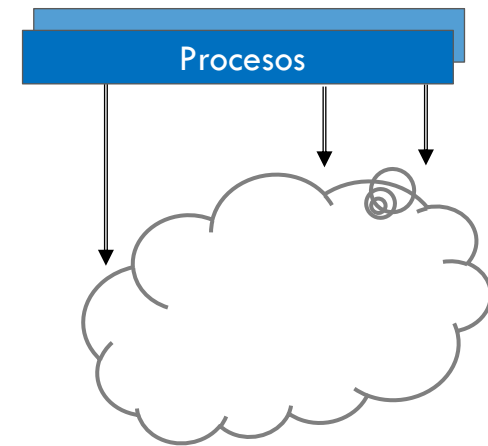
Eventos que activan el sistema operativo

- **Llamada al sistema.**
 - ▣ { Origen: “procesos”,
Función: “Petición de servicios” }
- **Excepción.**
 - ▣ { Origen: “procesos”,
Función: “Tratar situaciones de excepción” }
- **Interrupción hardware.**
 - ▣ { Origen: “hardware”,
Función: “Petición de atención del hw.” }

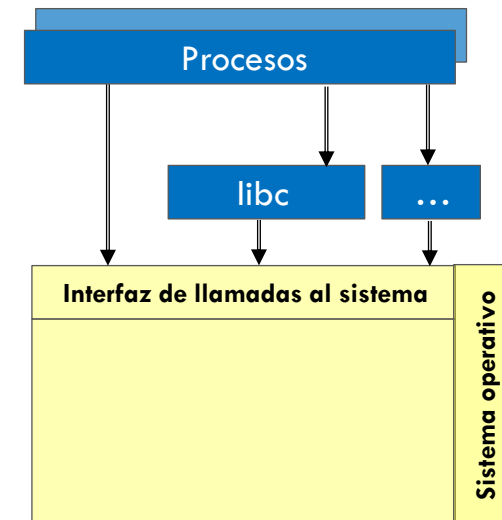


Servicios del sistema

- ❑ Gestión de procesos
- ❑ Gestión de memoria
- ❑ Gestión de ficheros
- ❑ Gestión de dispositivos
- ❑ Comunicación
- ❑ Mantenimiento



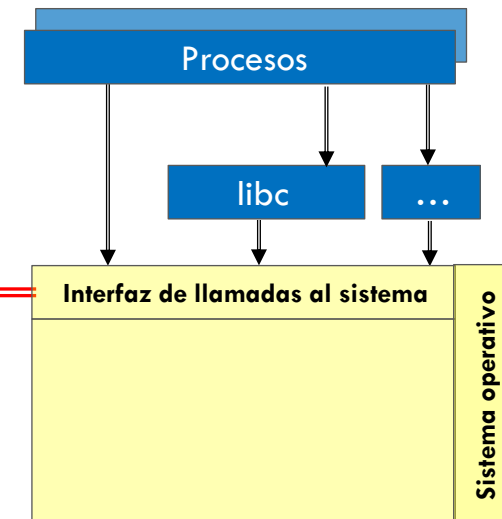
Llamadas al sistema vs librería sistema



Llamadas al sistema vs librería sistema



“servicios muy básicos de la casa”

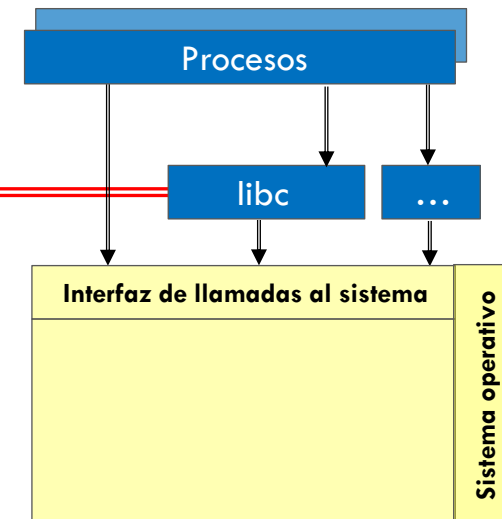


Llamadas al sistema vs librería sistema

11

<https://www.pexels.com/es-es/foto/tablas-de-cortar-cerca-del-horno-debajo-del-capo-2062426/>

Alejandro Calderón Mateos 



Llamadas al sistema vs librería sistema

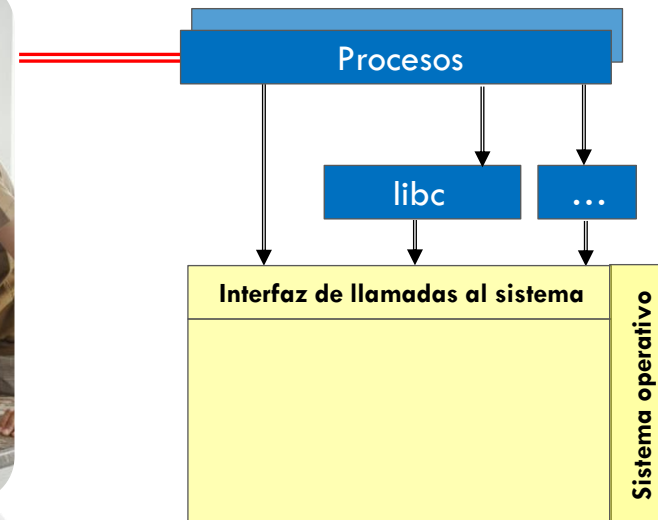
12

<https://www.pexels.com/es-es/foto/hombre-en-camisa-de-vestir-blanca-sentado-al-lado-de-una-mujer-en-vestido-naranja-426241>

Alejandro Calderón Mateos



“personas que utilizan los servicios”



Llamadas al sistema vs librería sistema memoria

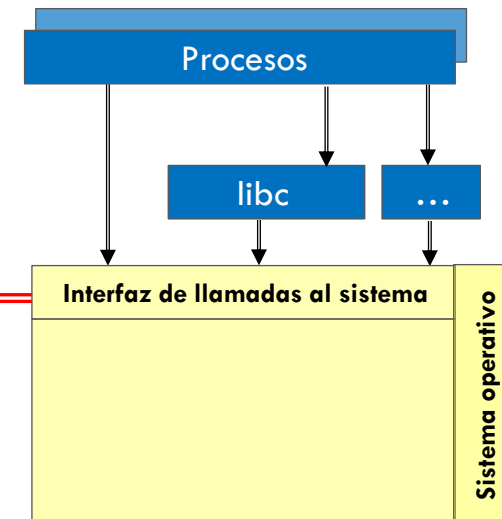
```
#include <unistd.h>
```

```
□ int      brk (void *);  
□ void     *sbrk (intptr_t);
```

```
□ int      close (int);  
□ off_t    lseek (int, off_t, int);  
□ ssize_t  read (int, void *, size_t);  
□ ssize_t  write (int, const void *, size_t);  
□ ...
```

```
#include <fcntl.h>
```

```
□ int      open (const char *path, int oflag, ... );  
□ int      creat (const char *path, mode_t mode);  
□ ...
```



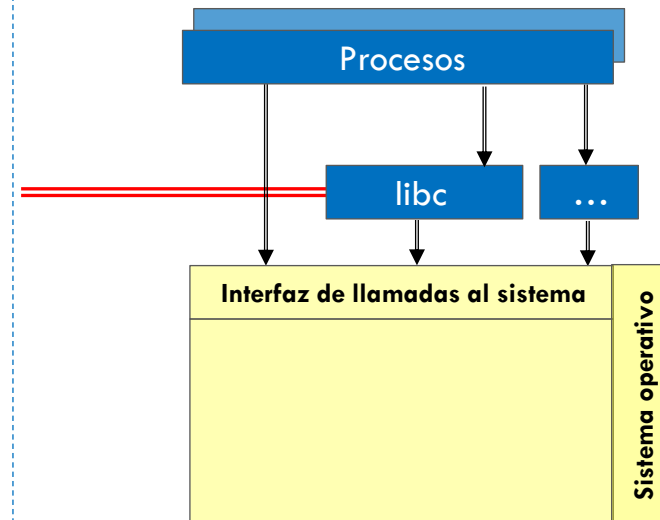
Llamadas al sistema vs librería sistema memoria

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

- ❑ void *malloc (unsigned long Size);
- ❑ void *realloc (void *Ptr, unsigned long NewSize);
- ❑ void *calloc (unsigned short NItems, unsigned short SizeOfItems);
- ❑ void free (void *Ptr);
- ❑ ...

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

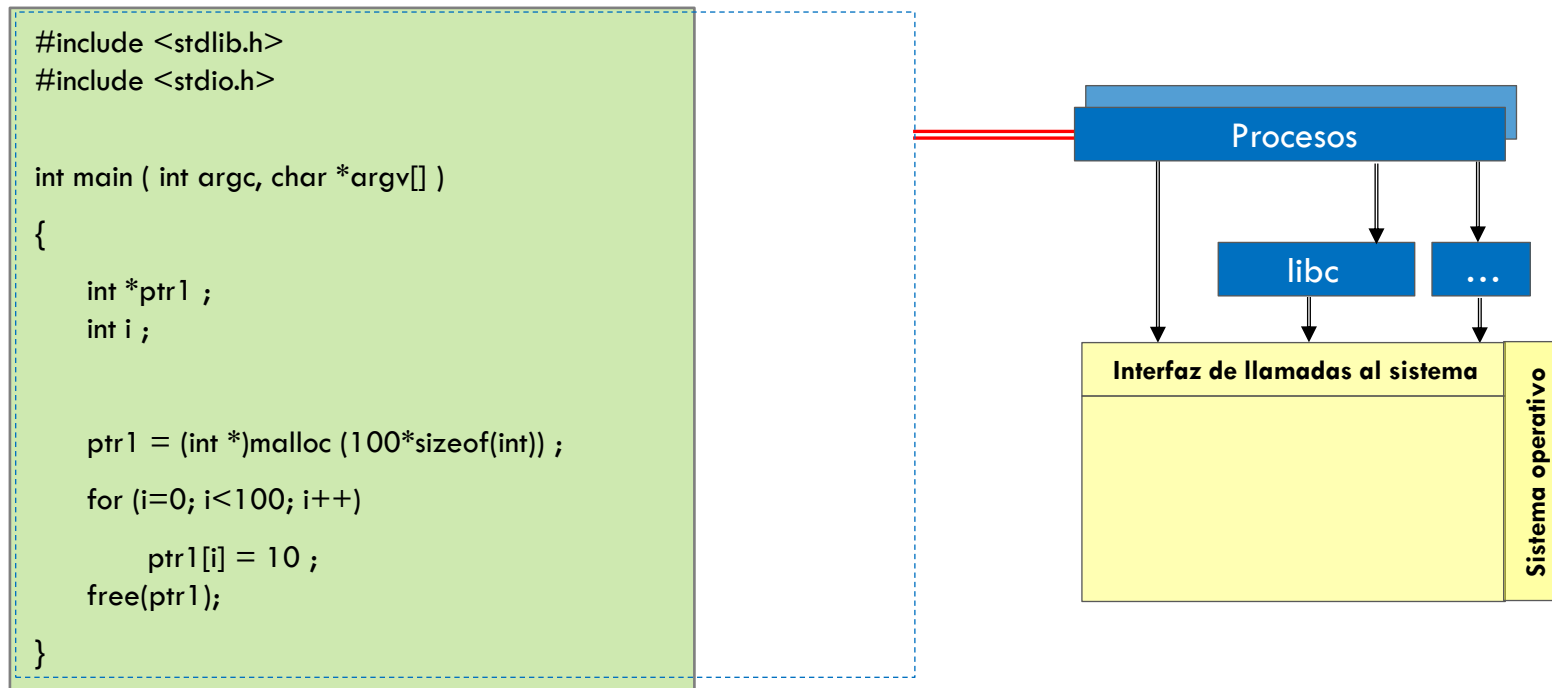
- ❑ FILE * fopen (const char *filename, const char *opentype);
- ❑ int fclose (FILE *stream);
- ❑ int feof(FILE *fichero);
- ❑ int fseek (FILE * stream, long int offset, int origin);
- ❑ size_t fread (void * ptr, size_t size, size_t count, FILE * f);
- ❑ int fscanf(FILE *f, const char *formato, argumento, ...);
- ❑ size_t fwrite(void *ptr, size_t size, size_t neltos, FILE *f);
- ❑ int fprintf(FILE *f, const char *fmt, arg1, ...);
- ❑ ...



Llamadas al sistema vs librería sistema memoria

15

Alejandro Calderón Mateos 



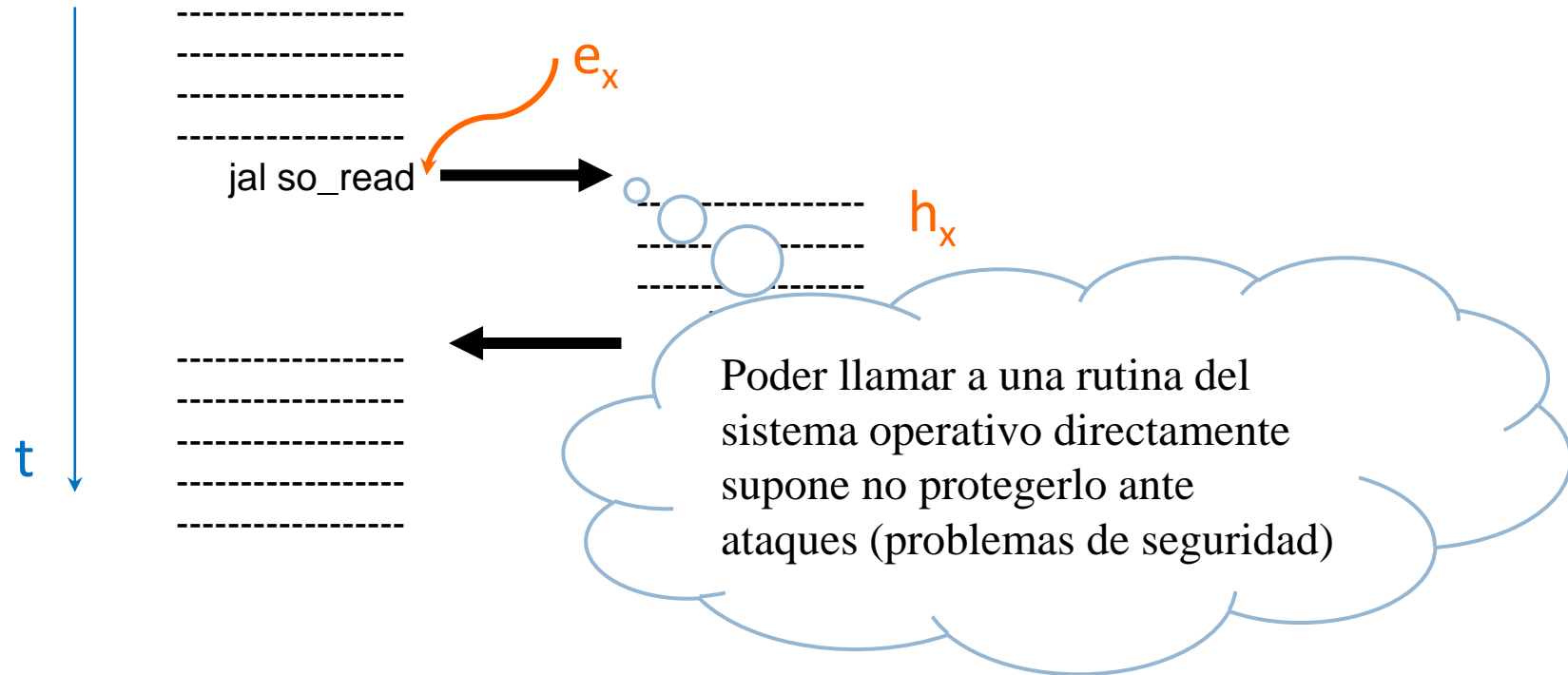
Contenidos

- Introducción a llamadas al sistema
- Mecanismo de llamada al sistema
- Llamadas para servicios de:
 - ▣ Gestión de procesos
 - ▣ Gestión de ficheros

Ejecución petición de servicio no es una llamada a una función...

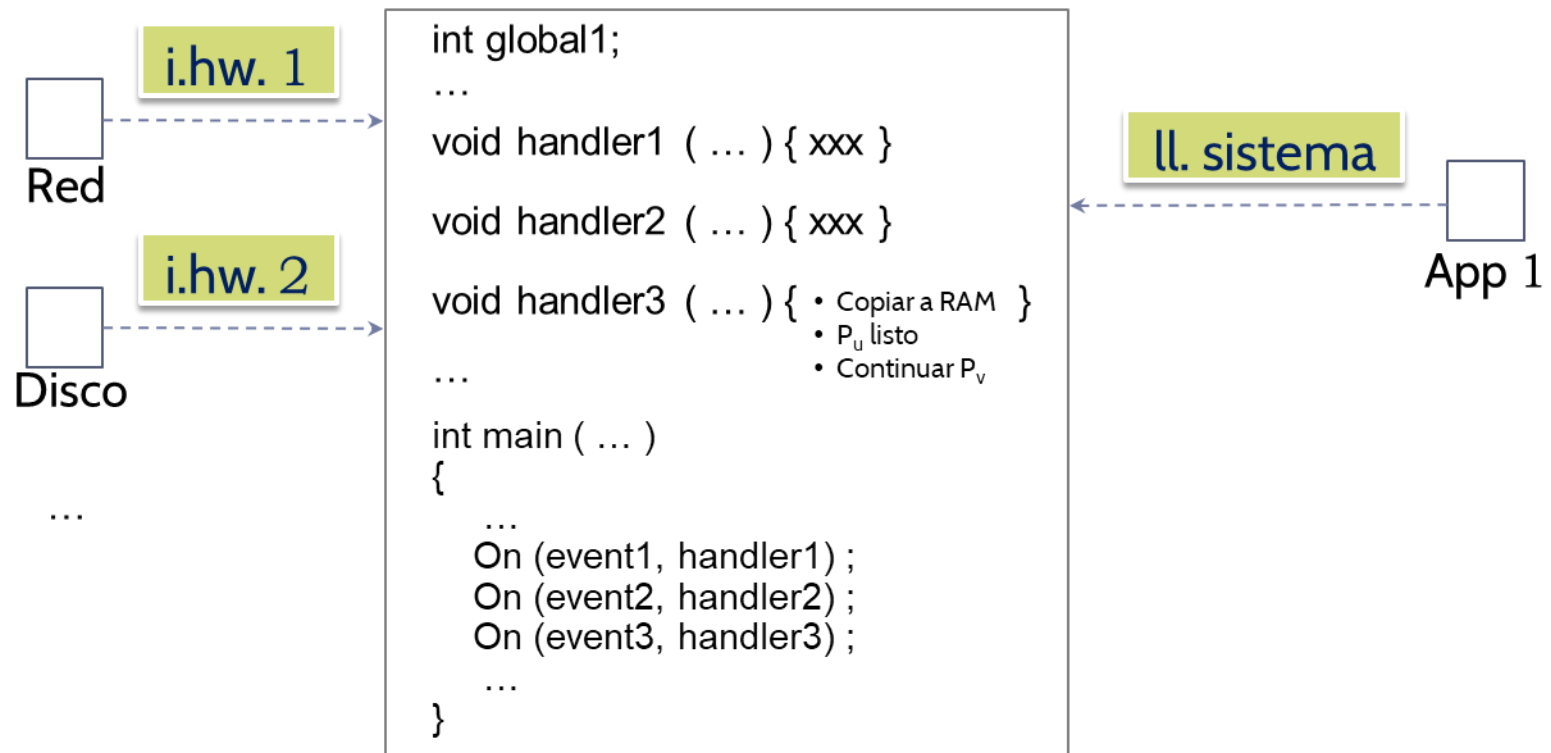
17

Alejandro Calderón Mateos 



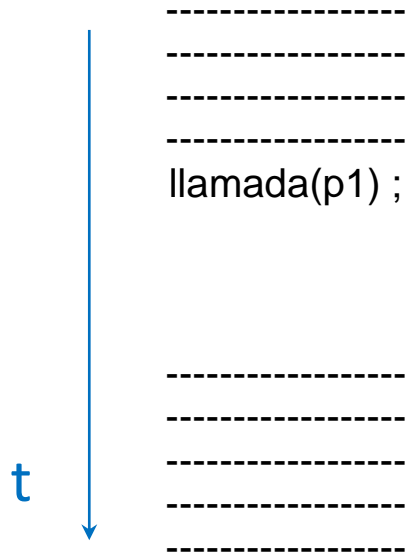
Ejecución tratando eventos

aspecto general



Ejecución petición de servicio

ejecución (general)

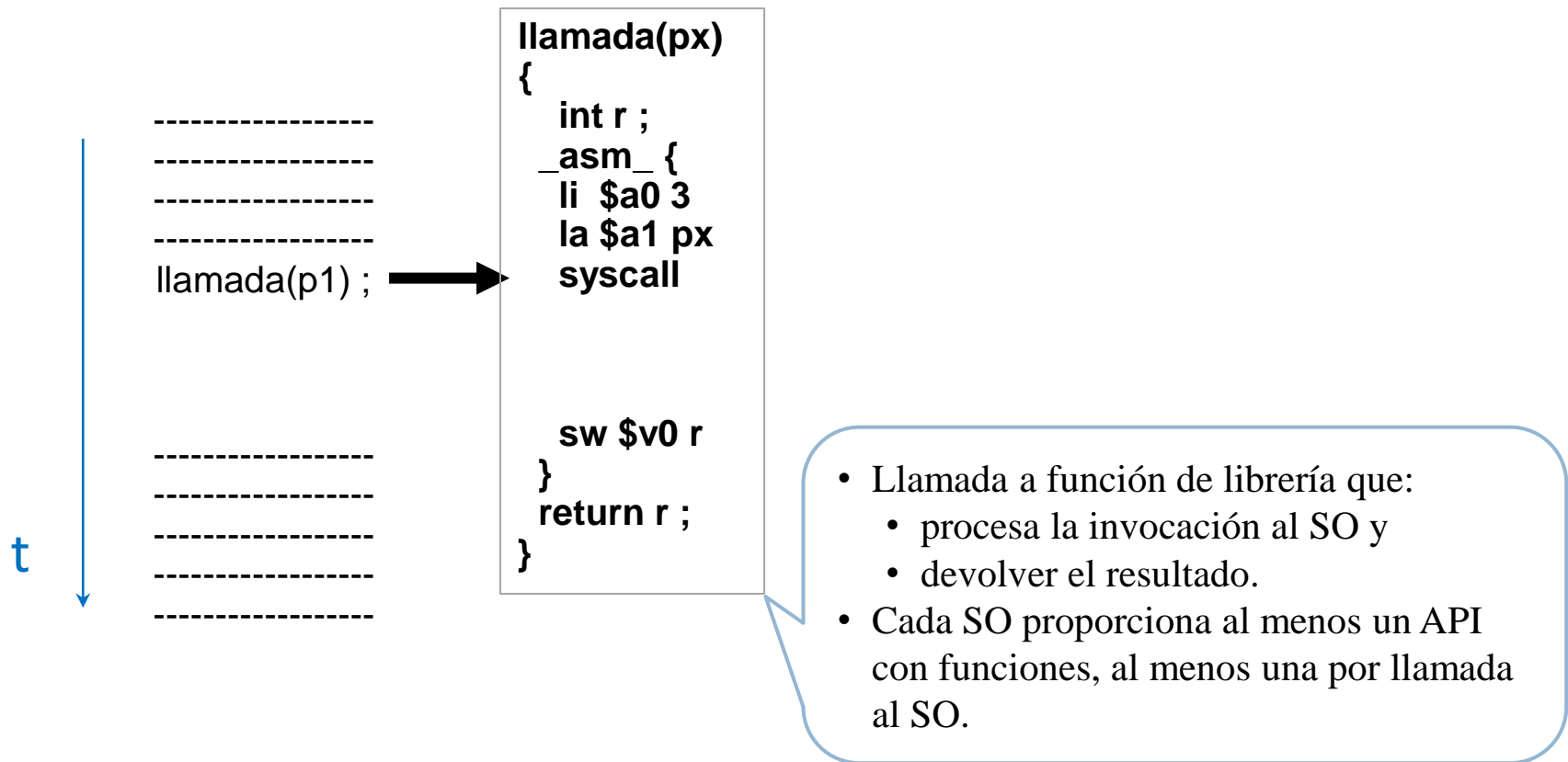


Ejecución petición de servicio

ejecución (general)

20

Alejandro Calderón Mateos 

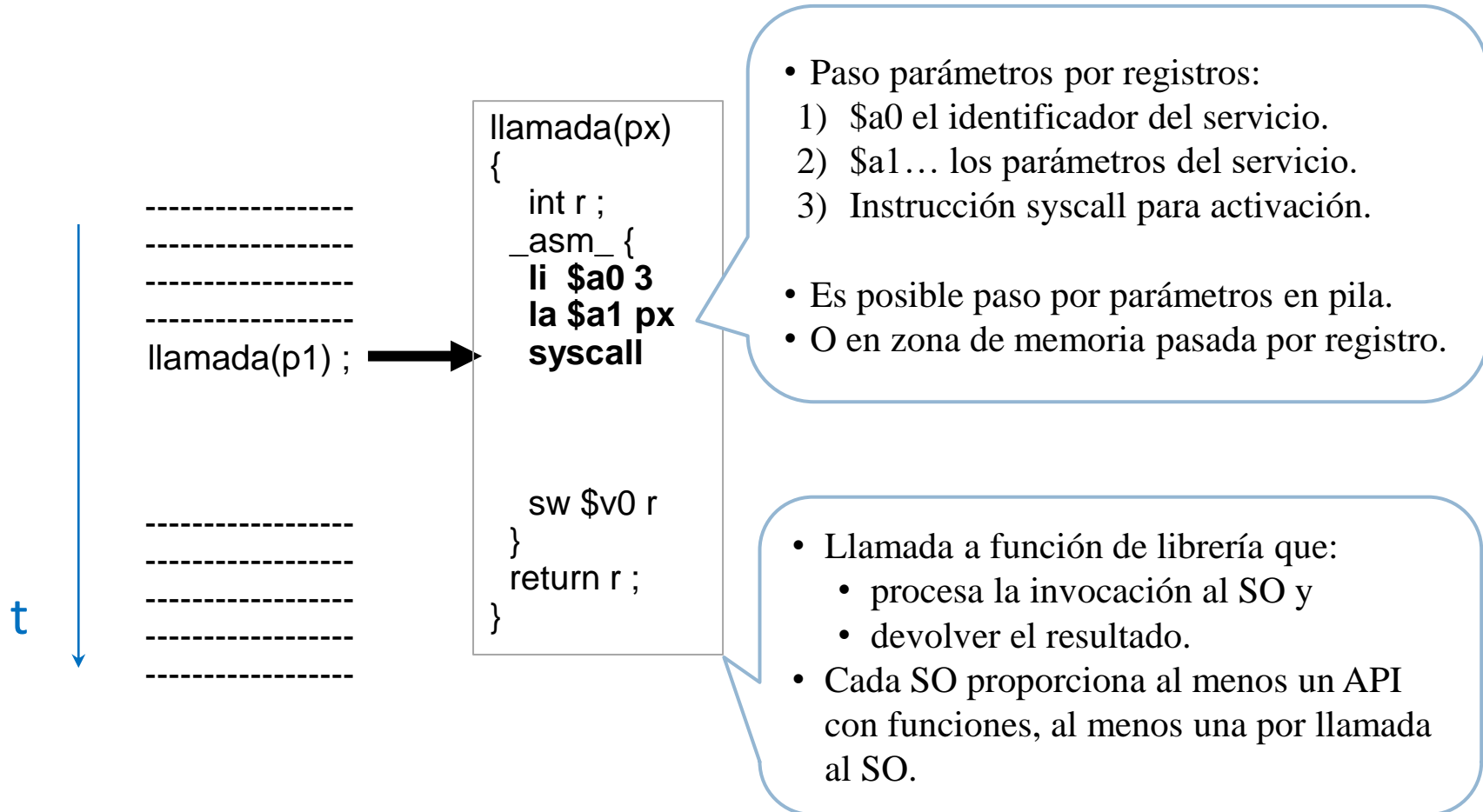


Ejecución petición de servicio

ejecución (general)

21

Alejandro Calderón Mateos 

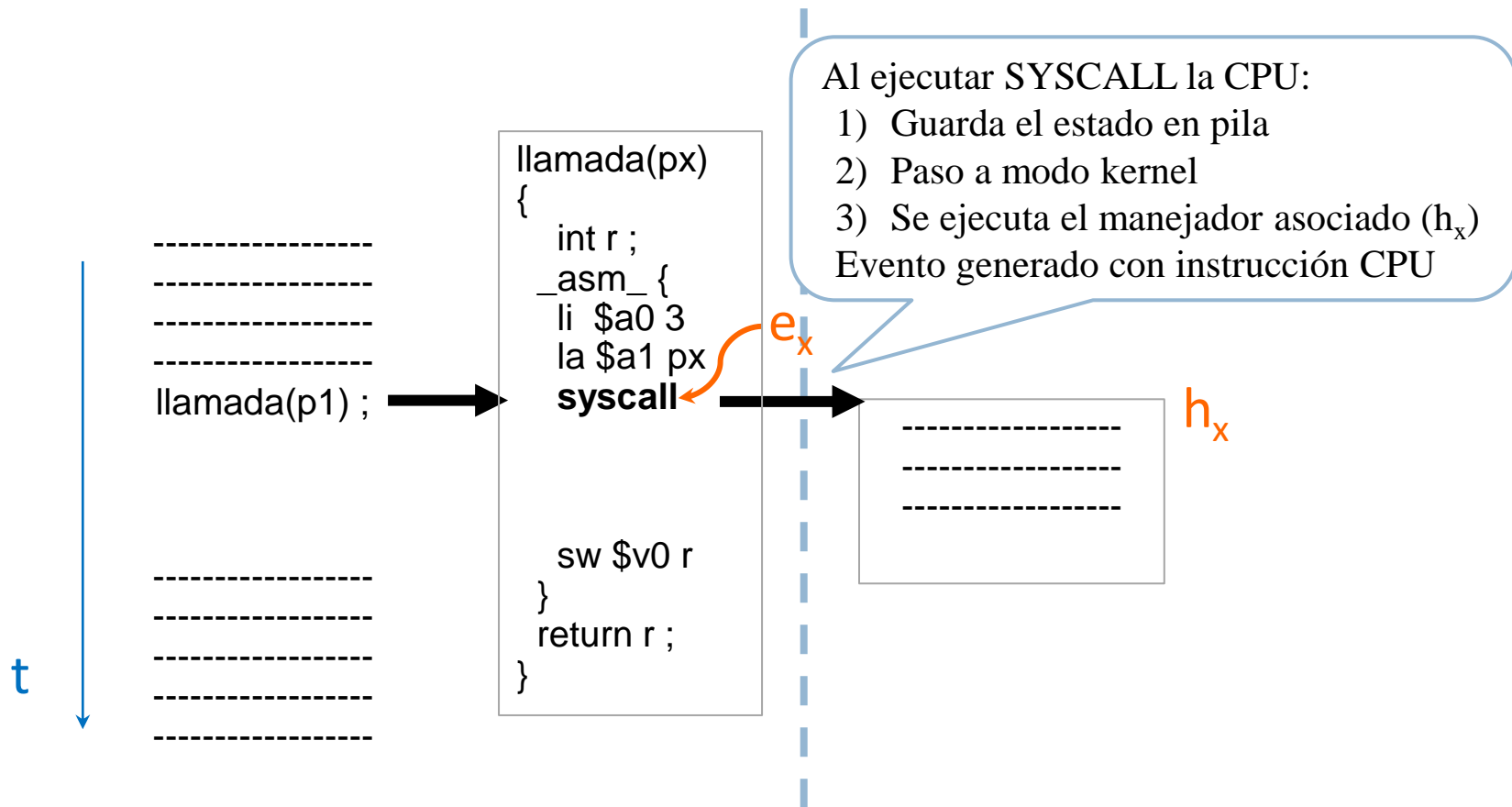


Ejecución petición de servicio

ejecución (general)

22

Alejandro Calderón Mateos

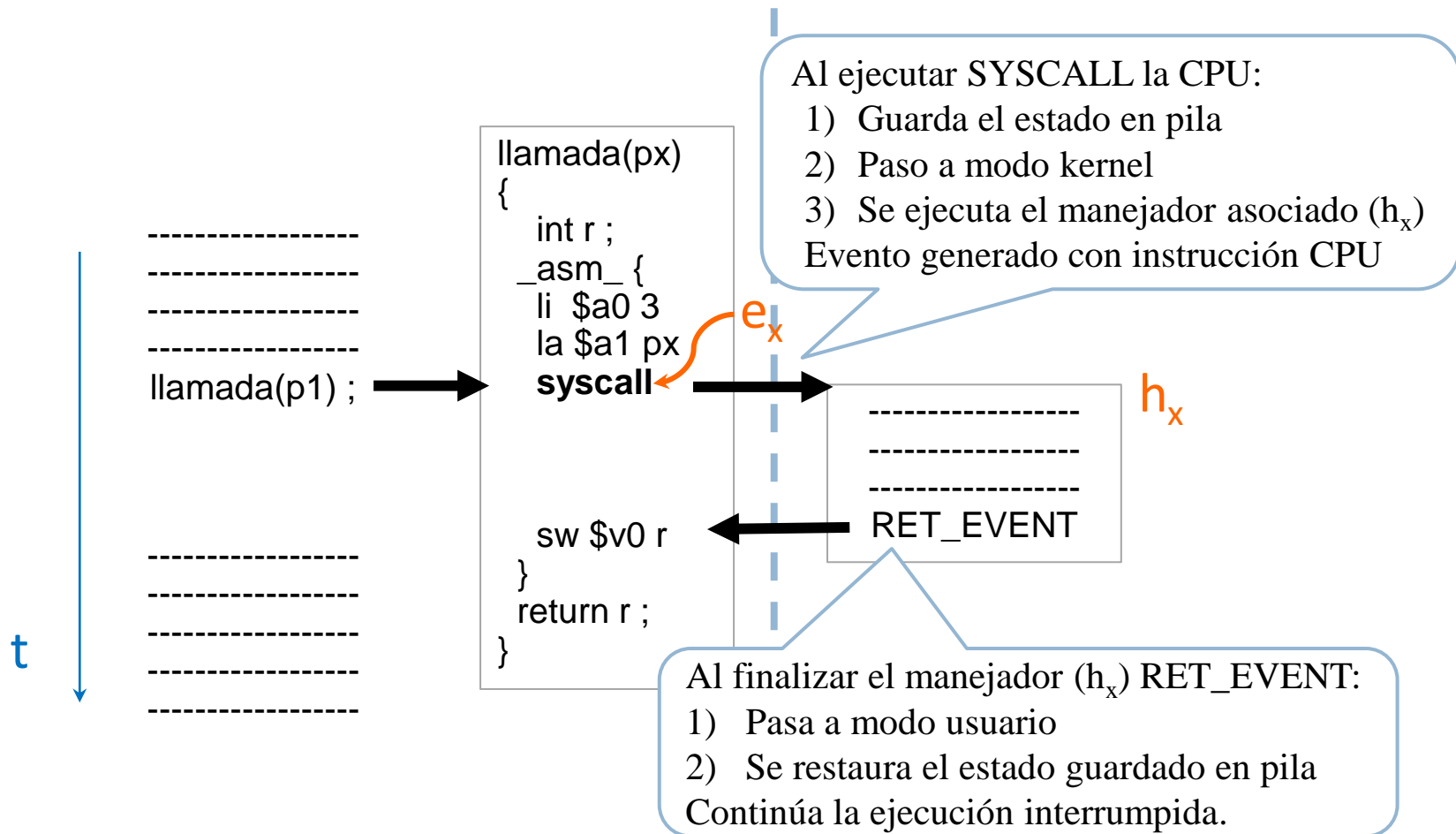


Ejecución petición de servicio

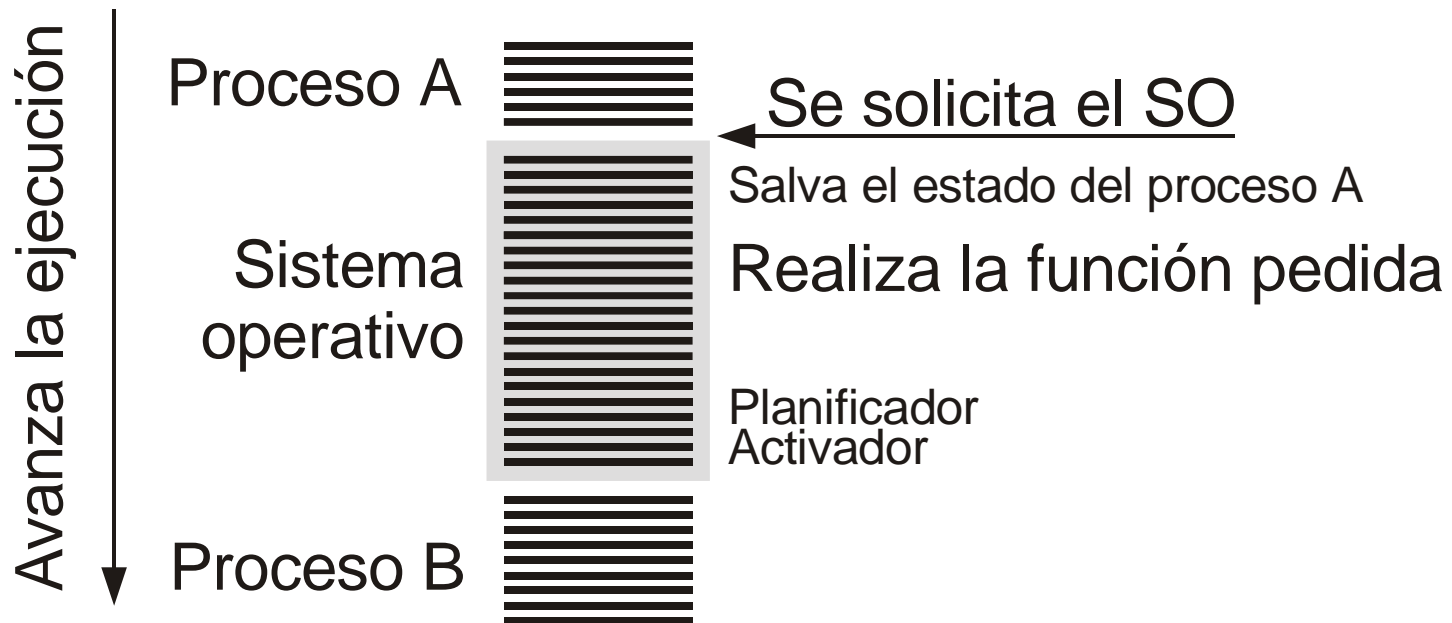
ejecución (general)

23

Alejandro Calderón Mateos



Fases en la activación del Sistema Operativo



Llamadas al sistema

tratamiento en Linux (1 / 7)

25

Alejandro Calderón Mateos 

/usr/src/linux/arch/x86/kernel/traps.c

```
void __init trap_init(void)
{
    ...
    set_intr_gate(X86_TRAP_DE, divide_error);
    set_intr_gate(X86_TRAP_NP, segment_not_present);
    set_intr_gate(X86_TRAP_GP, general_protection);
    set_intr_gate(X86_TRAP_SPURIOUS, spurious_interrupt_bug);
    set_intr_gate(X86_TRAP_MF, coprocessor_error);
    set_intr_gate(X86_TRAP_AC, alignment_check);

#ifdef CONFIG_IA32_EMULATION
    set_system_intr_gate(IA32_SYSCALL_VECTOR, ia32_syscall);
    set_bit(IA32_SYSCALL_VECTOR, used_vectors);
#endif

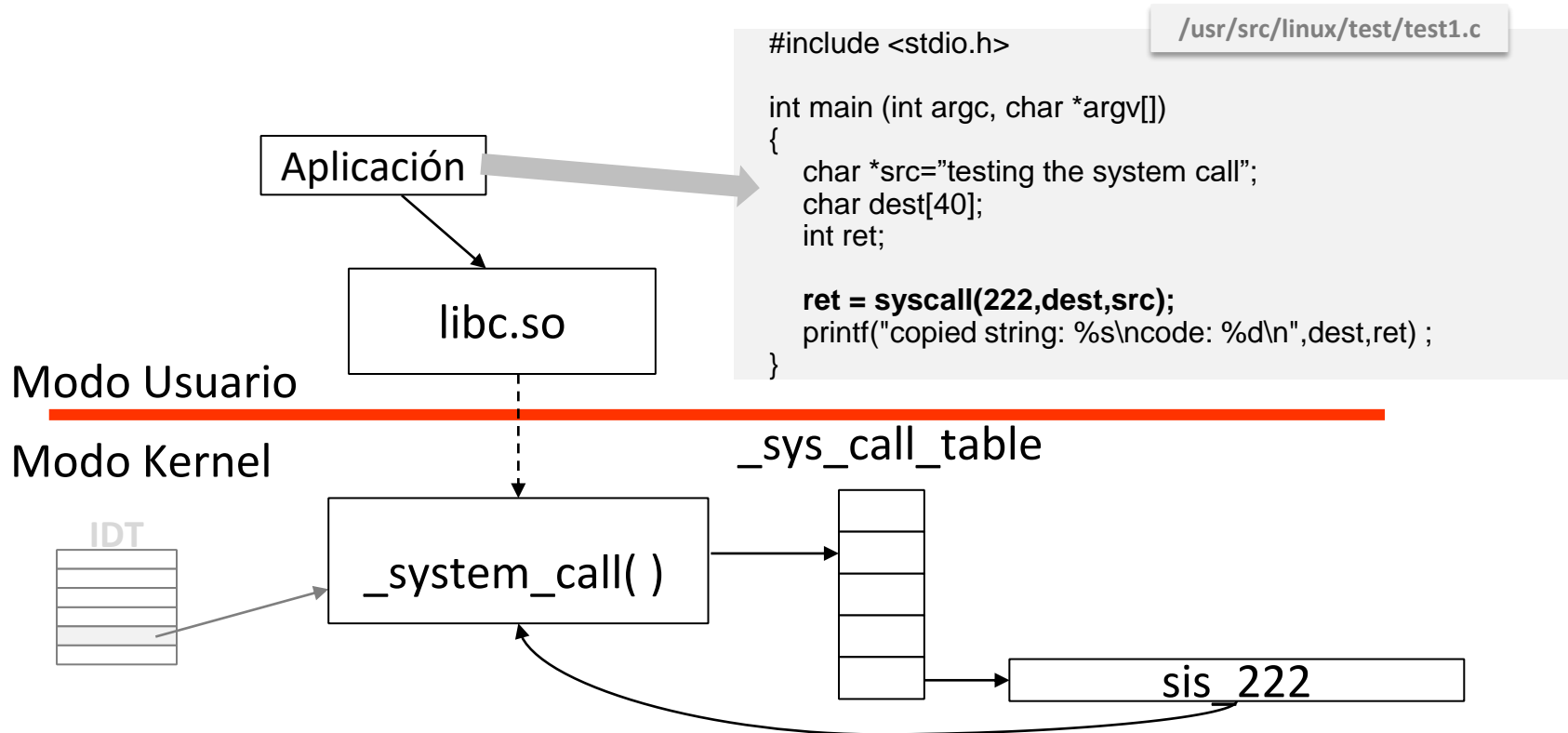
#ifdef CONFIG_X86_32
    set_system_trap_gate(SYSCALL_VECTOR, &system_call);
    set_bit(SYSCALL_VECTOR, used_vectors);
#endif
    ...
}
```

Llamadas al sistema

tratamiento en Linux (2/7)

26

Alejandro Calderón Mateos

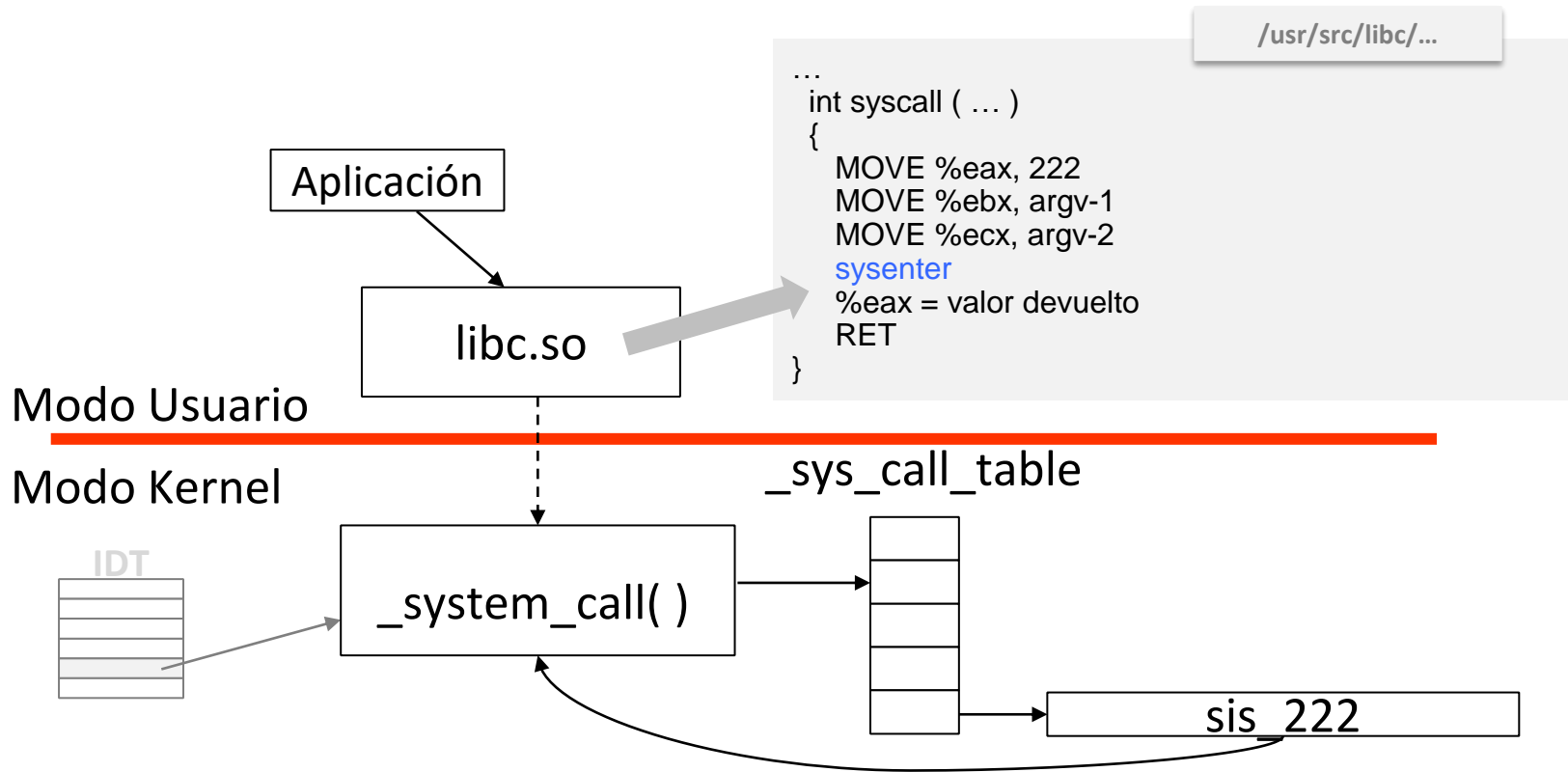


Llamadas al sistema tratamiento en Linux (3/7)

- Cada servicio del SO se corresponde con una función (API el cto. de todas).
- Dicha función encapsula invocación al servicio: parámetros, trap, retornar...

27

Alejandro Calderón Mateos

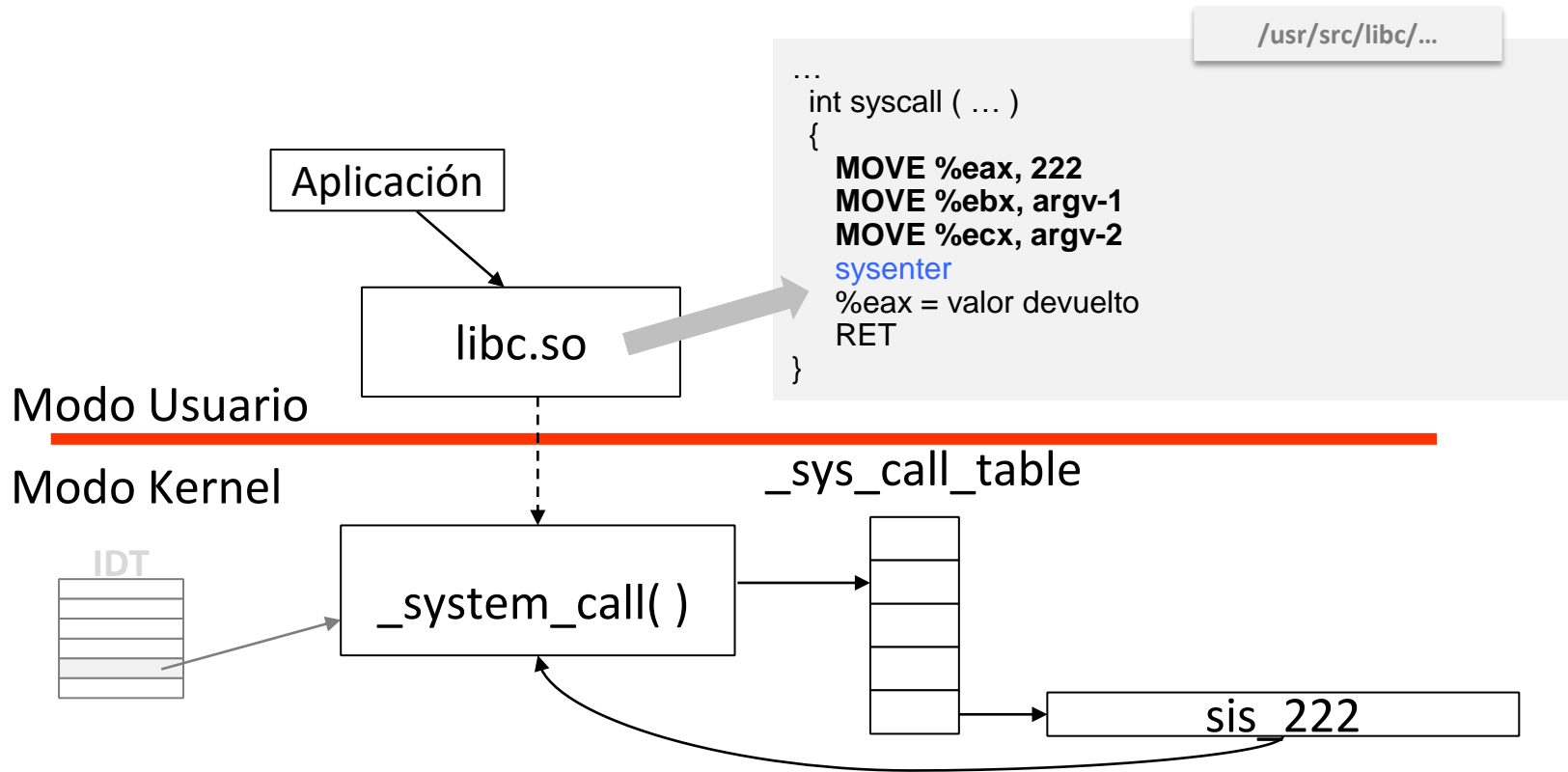


Llamadas al sistema tratamiento en Linux (3/7)

- Paso de parámetros por registro, pila o zona de memoria pasada por registro.
- Parámetro 1: identificador de servicio

28

Alejandro Calderón Mateos

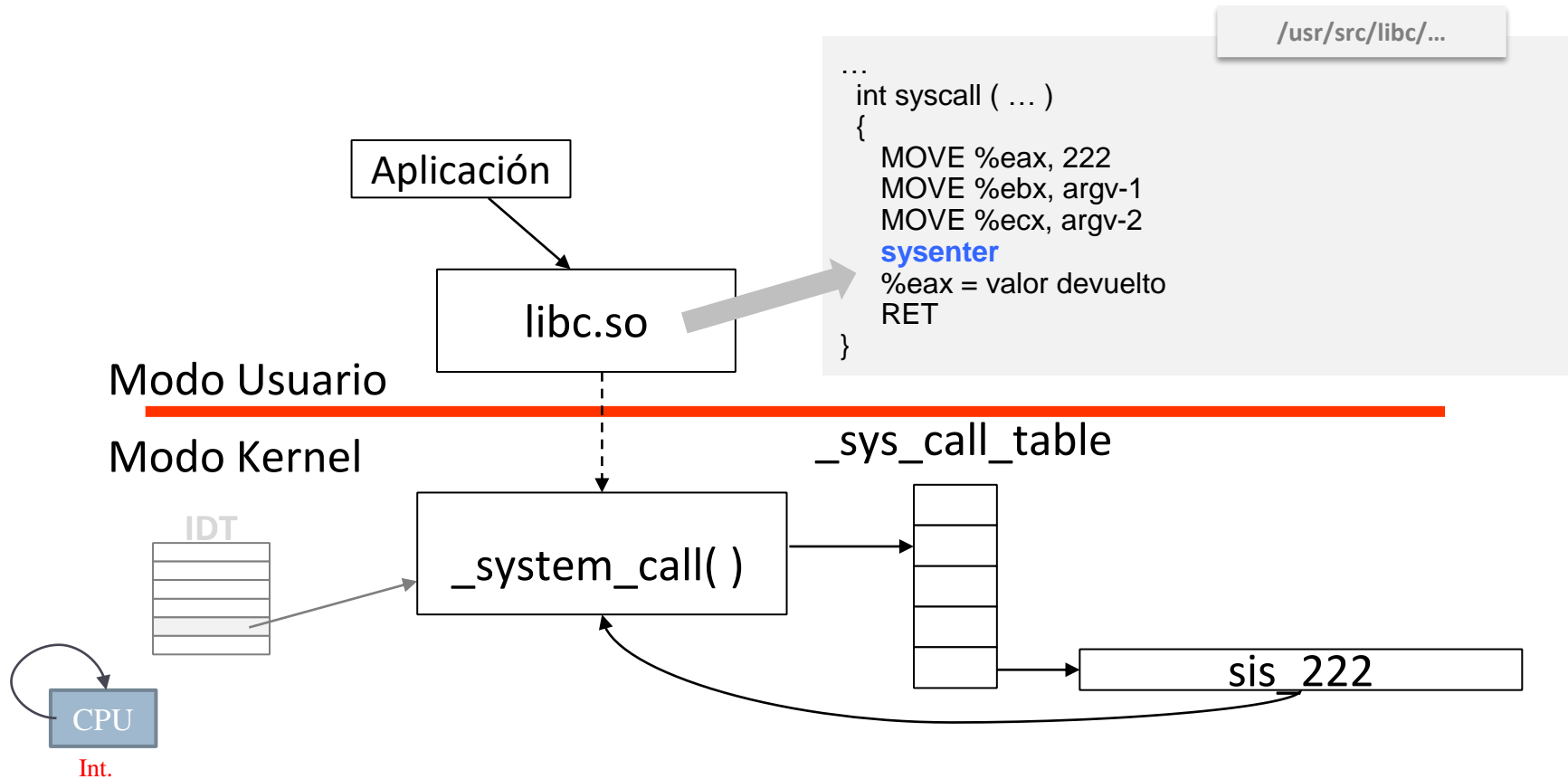


Llamadas al sistema tratamiento en Linux (3/7)

- El trap (sysenter en CPU x86) es una instrucción que genera un evento con tratamiento similar a interrupción hardware.

29

Alejandro Calderón Mateos

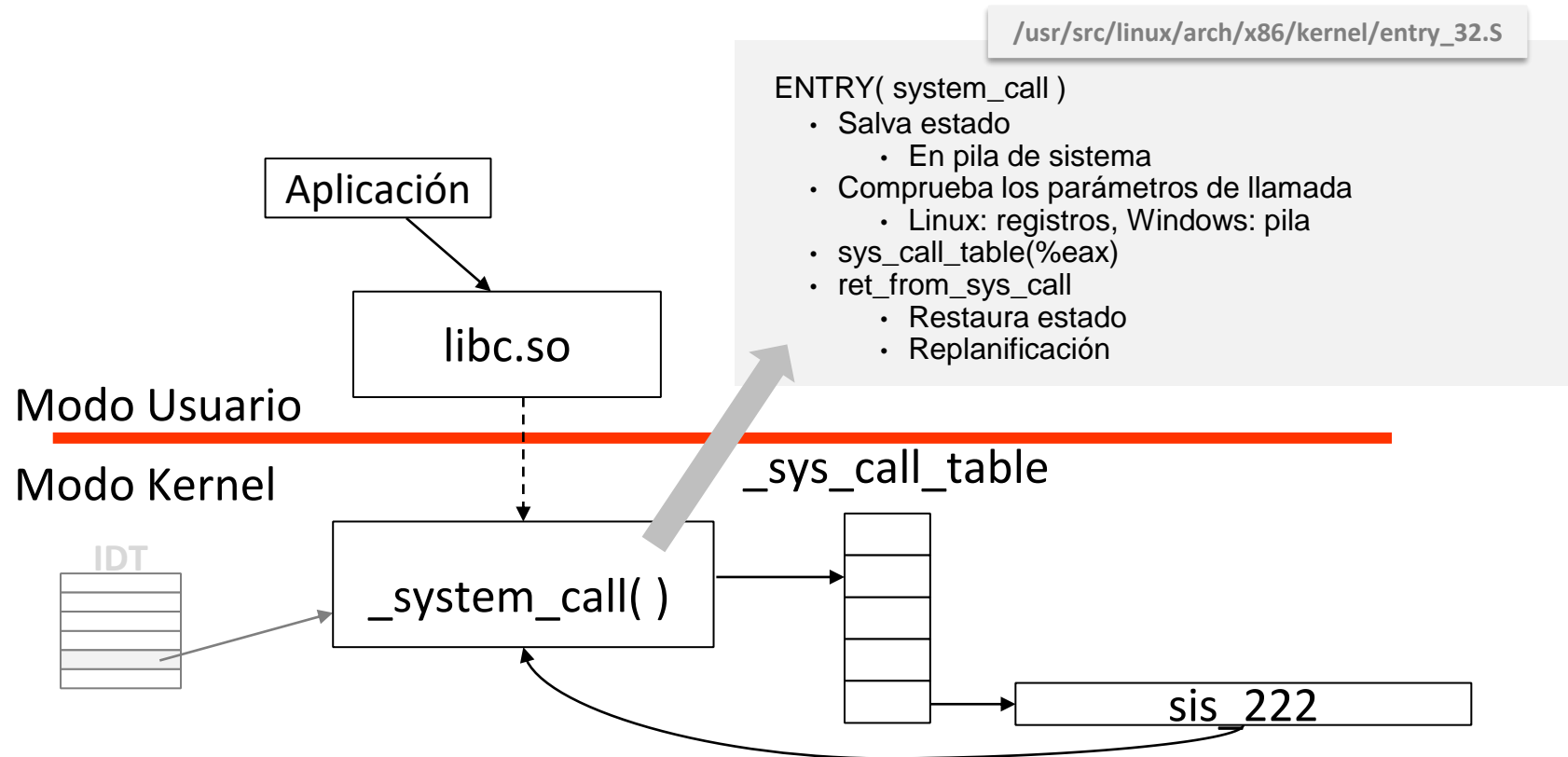


Llamadas al sistema tratamiento en Linux (4/7)

- Comprueba parámetros, determina función en SO a partir del identificador (indexar en `_sys_call_table`) e invoca.

30

Alejandro Calderón Mateos

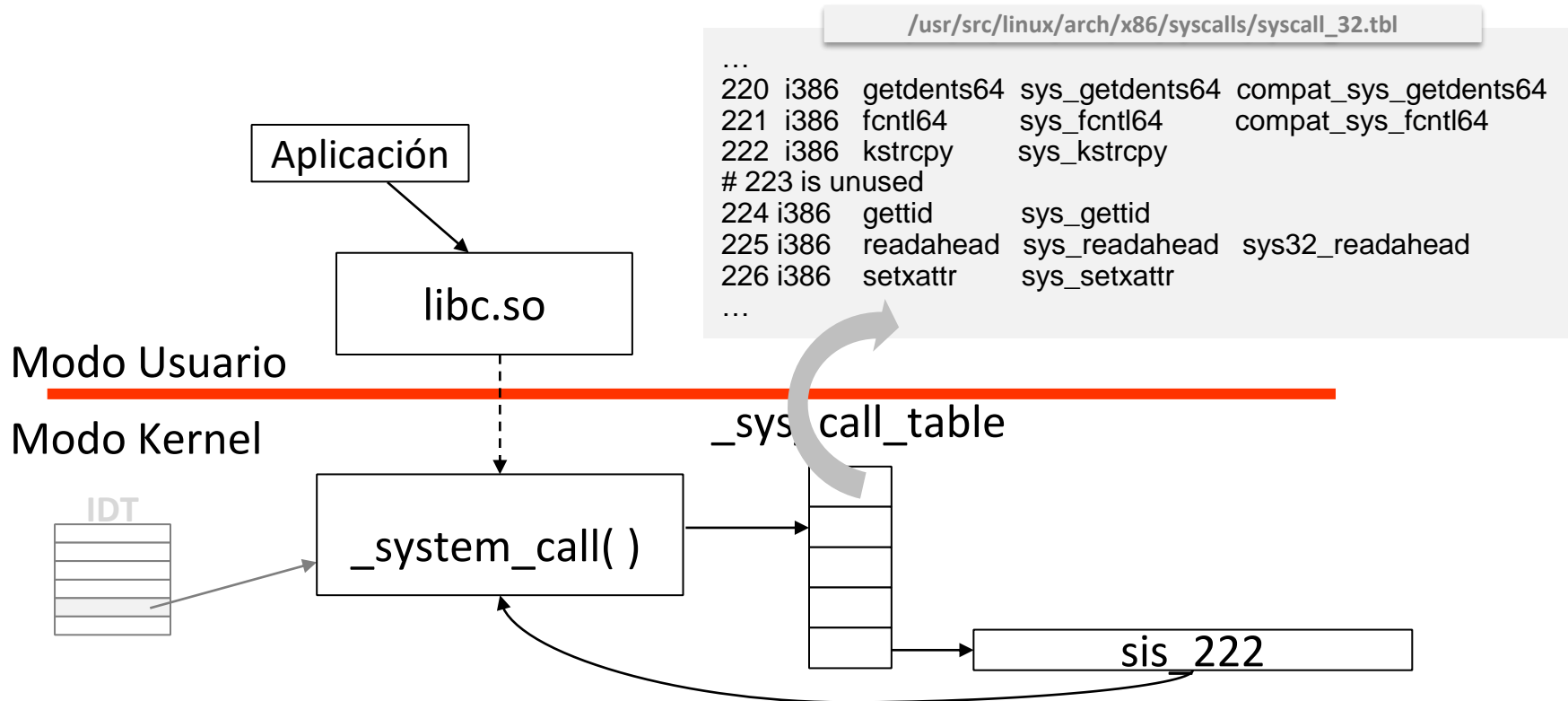


Llamadas al sistema

tratamiento en Linux (5/7)

31

Alejandro Calderón Mateos

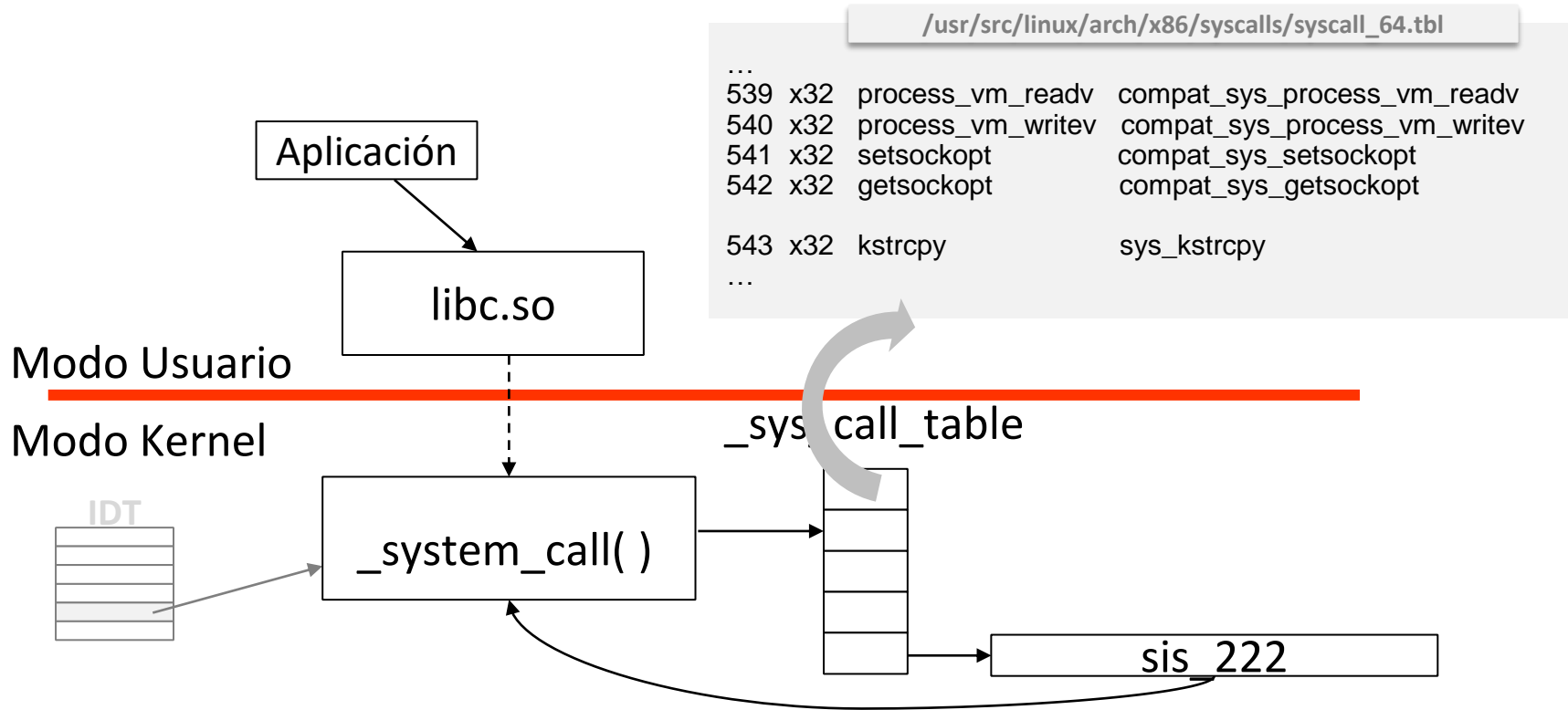


Llamadas al sistema

tratamiento en Linux (6/7)

32

Alejandro Calderón Mateos

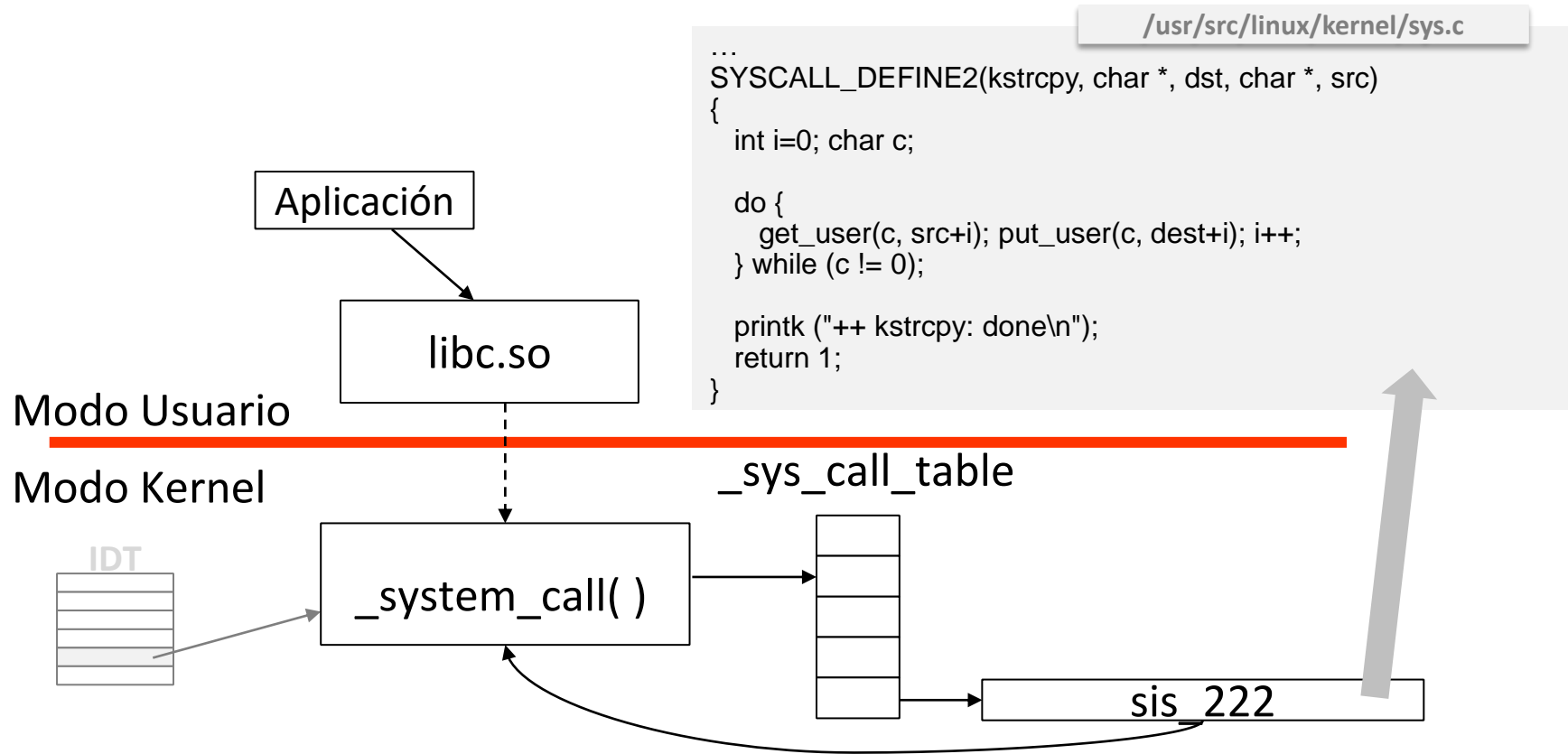


Llamadas al sistema

tratamiento en Linux (7/7)

33

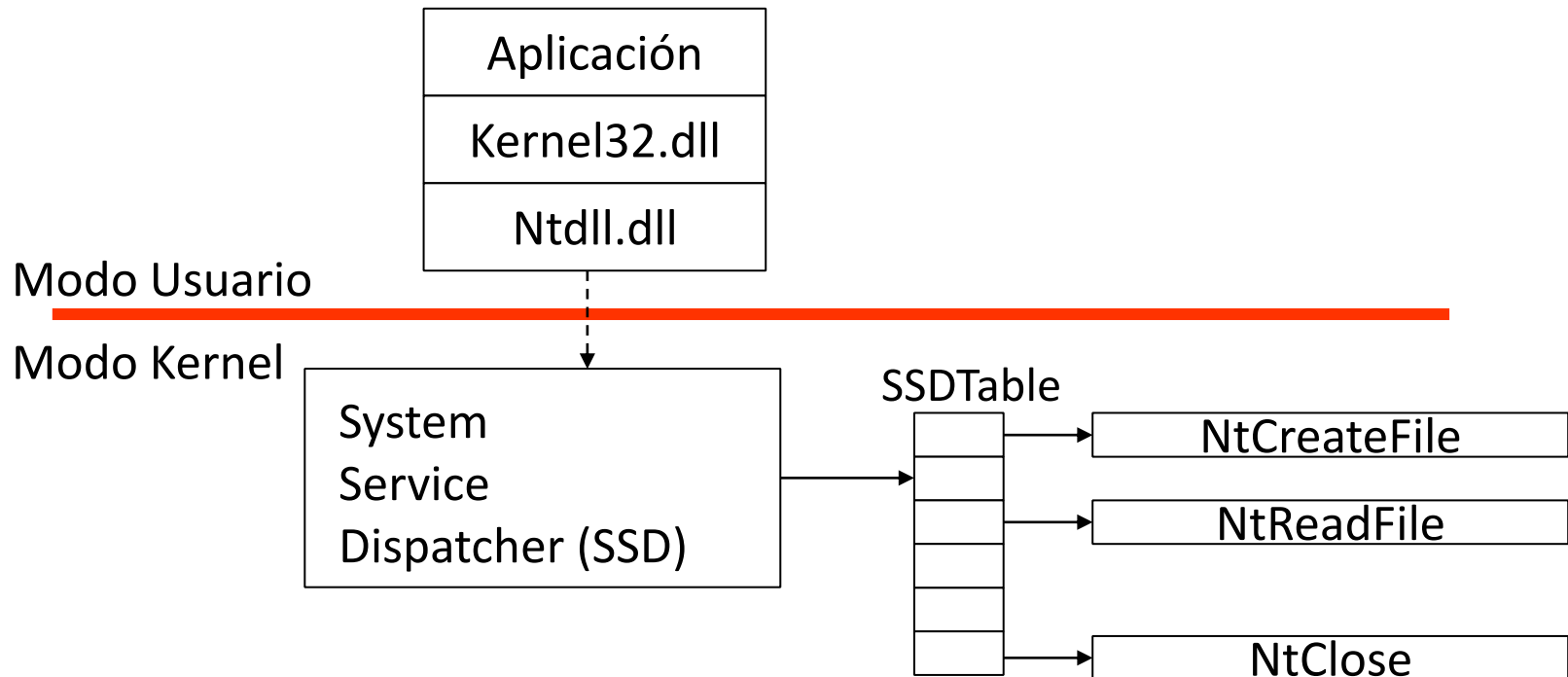
Alejandro Calderón Mateos



Llamadas al sistema tratamiento en Windows

34

Sistemas operativos: una visión aplicada



Interfaz del programador

- El conjunto de funciones que ofrecen los servicios del SO (encapsulando las llamadas) es la interfaz del programador.
 - ▣ Esta interfaz ofrece la visión que como máquina extendida tiene el usuario del sistema operativo
 - ▣ Mejor usar especificaciones de interfaces estándares.
- Cada sistema operativo puede ofrecer una o varias interfaces:
 - ▣ Linux: POSIX
 - ▣ Windows: Win32, POSIX

Estándar POSIX

- ❑ 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

Características de POSIX

- Nombres de funciones cortos y en letras minúsculas:
 - fork
 - read
 - close
- Las funciones normalmente devuelve 0 en caso de éxito o -1 en caso de error.
 - Variable errno.
- Recursos gestionados por el sistema operativo se referencian mediante descriptores (números enteros)

UNIX 03

- Single Unix Specification (SUS)
 - ▣ V1 (UNIX 95), V2 (UNIX 98), V3 (UNIX 03) y V4 (UNIX V7)
- 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.
- Ejemplo de UNIX 03: AIX, EulerOS, HP-UX, macOS

Contenidos

- Introducción a llamadas al sistema
- Mecanismo de llamada al sistema
- Llamadas para servicios de:
 - ▣ Gestión de procesos
 - ▣ Gestión de ficheros

Repaso

fork() + exec()

40

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```


Repaso

fork() + exec()

41

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso

fork() + exec()

42

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso

fork() + exec()

43


Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```



```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```



Repaso


fork() + exec()

44

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>


main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
         while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
         execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso


fork() + exec()

45

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>


main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {

        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {

        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso


fork() + exec()

46

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>


main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {

        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {

        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso


fork() + exec()

47

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
         while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* código del mandato ls */
#include <sys/types.h>
#include <stdio.h>

main() {

    /* código del ls */

    exit( 0 );
}
```

Repaso

wait() + exit()

48

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* código del mandato ls */
#include <sys/types.h>
#include <stdio.h>

main() {

    /* código del ls */

    exit( 0 );
}
```


Repaso

wait() + exit()

49

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

```
/* código del mandato ls */
#include <sys/types.h>
#include <stdio.h>

main() {

    /* código del ls */

    exit( 0 );
}
```

Repaso

wait() + exit()

50

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

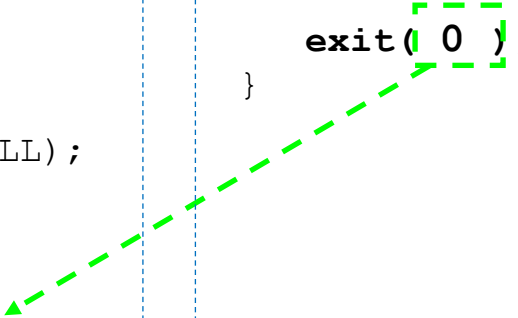
    exit(0);
}
```

```
/* código del mandato ls */
#include <sys/types.h>
#include <stdio.h>

main() {

    /* código del ls */

    exit( 0 );
}
```



Repaso

wait() + exit()

51

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execlp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso

wait() + exit()

52

Alejandro Calderón Mateos 

```
/* ejecutar el mandato ls -l */
#include <sys/types.h>
#include <stdio.h>

main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0)
    {
        execvp("ls", "ls", "-l", NULL);
        exit(-1);
    }
    else
    {
        while (pid != wait(&status));
    }

    exit(0);
}
```

Repaso

`wait()` + `exit()`

Servicio fork

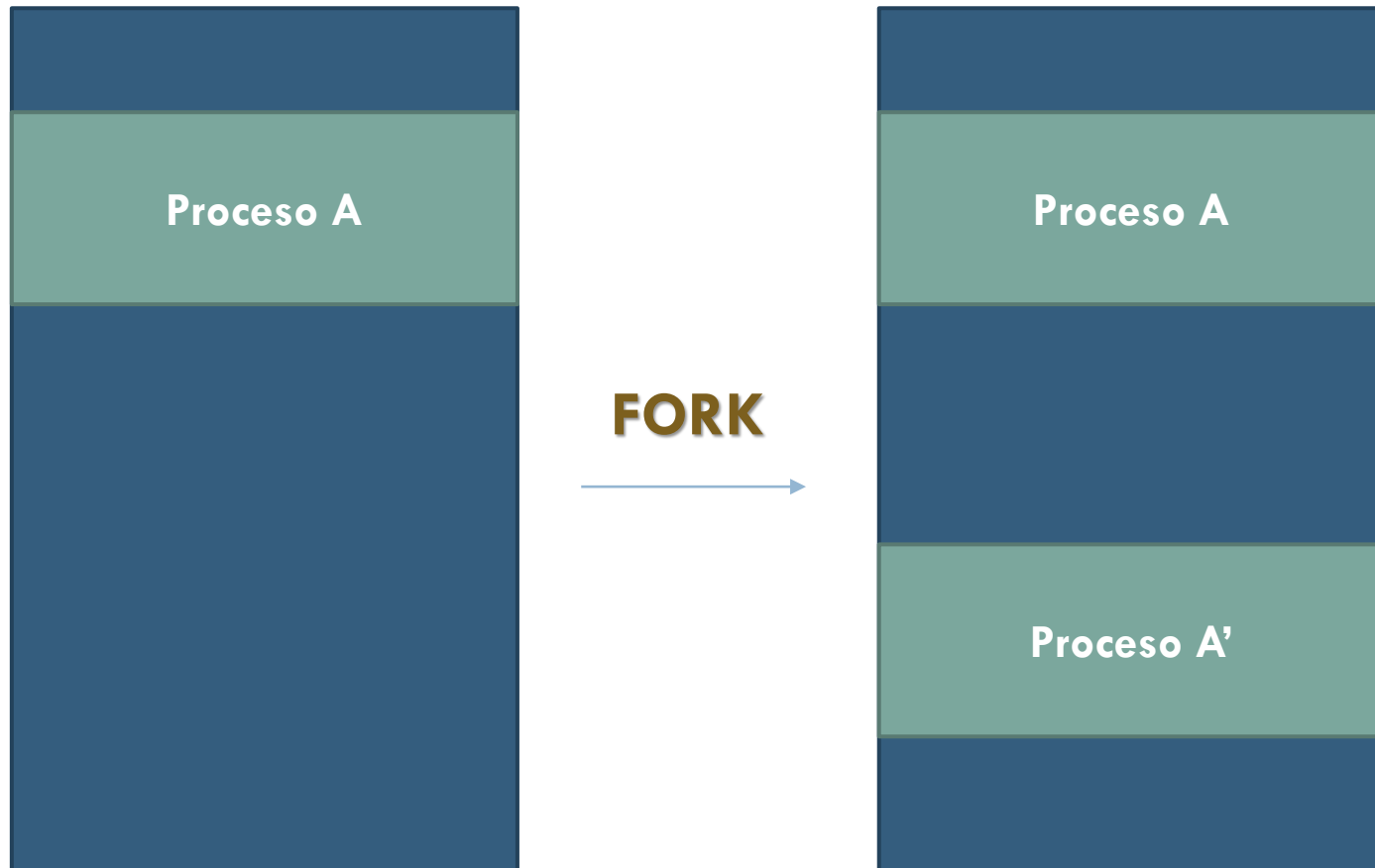
```
pid_t fork(void) ;
```

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

Servicio fork

55

Sistemas operativos: una visión aplicada



Servicio exec

Servicio único pero múltiples funciones de biblioteca.

```
int exec1(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[])
```

□ Parámetros:

- path: Ruta al archivo ejecutable.
- file: Busca el archivo ejecutable en todos los directorios especificados por PATH.

□ Devuelve:

- Devuelve -1 en caso de error, **en caso contrario no retorna.**

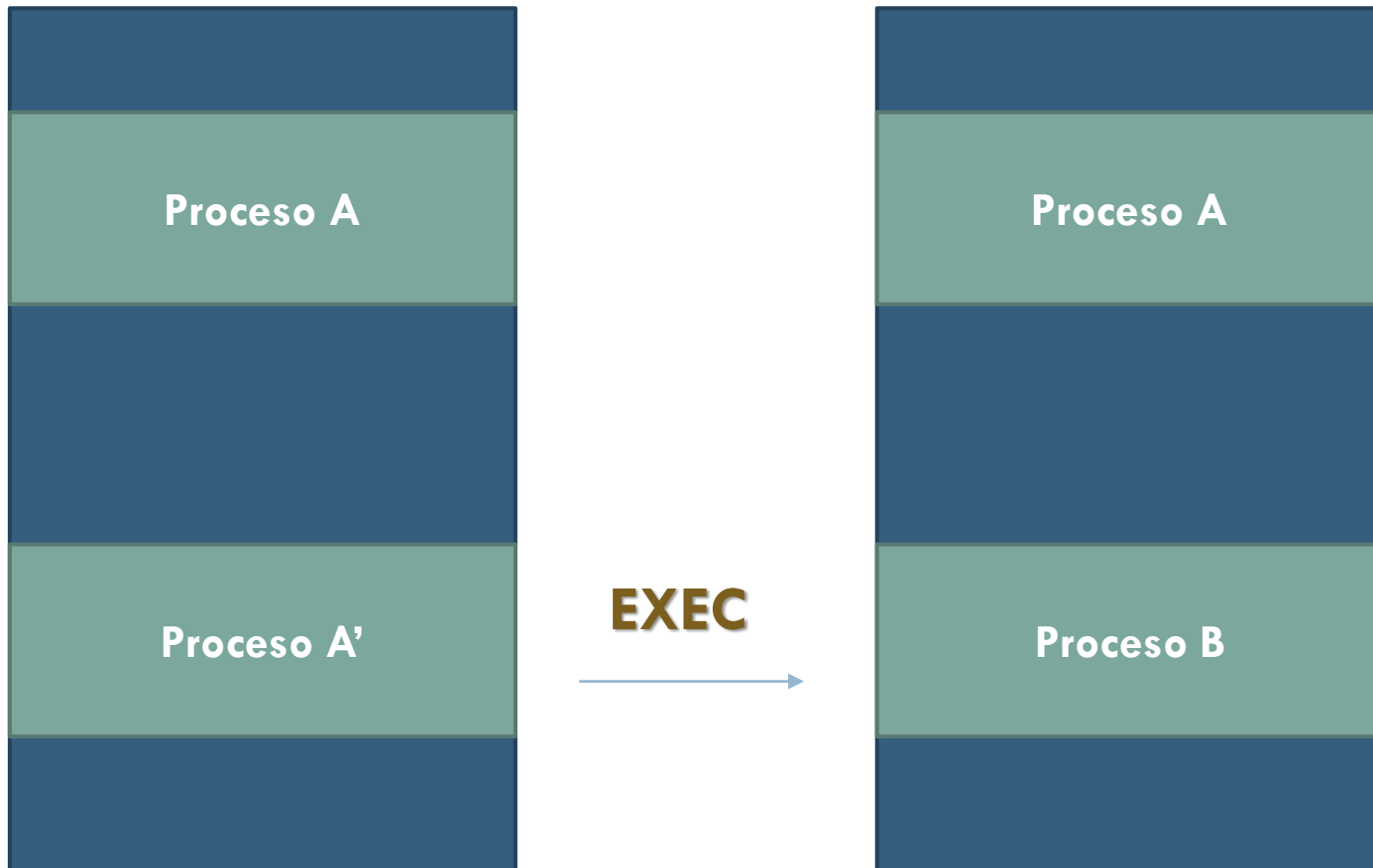
□ Descripción:

- Cambia la imagen del proceso actual.
- 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 exec

57

Sistemas operativos: una visión aplicada



Servicio exit

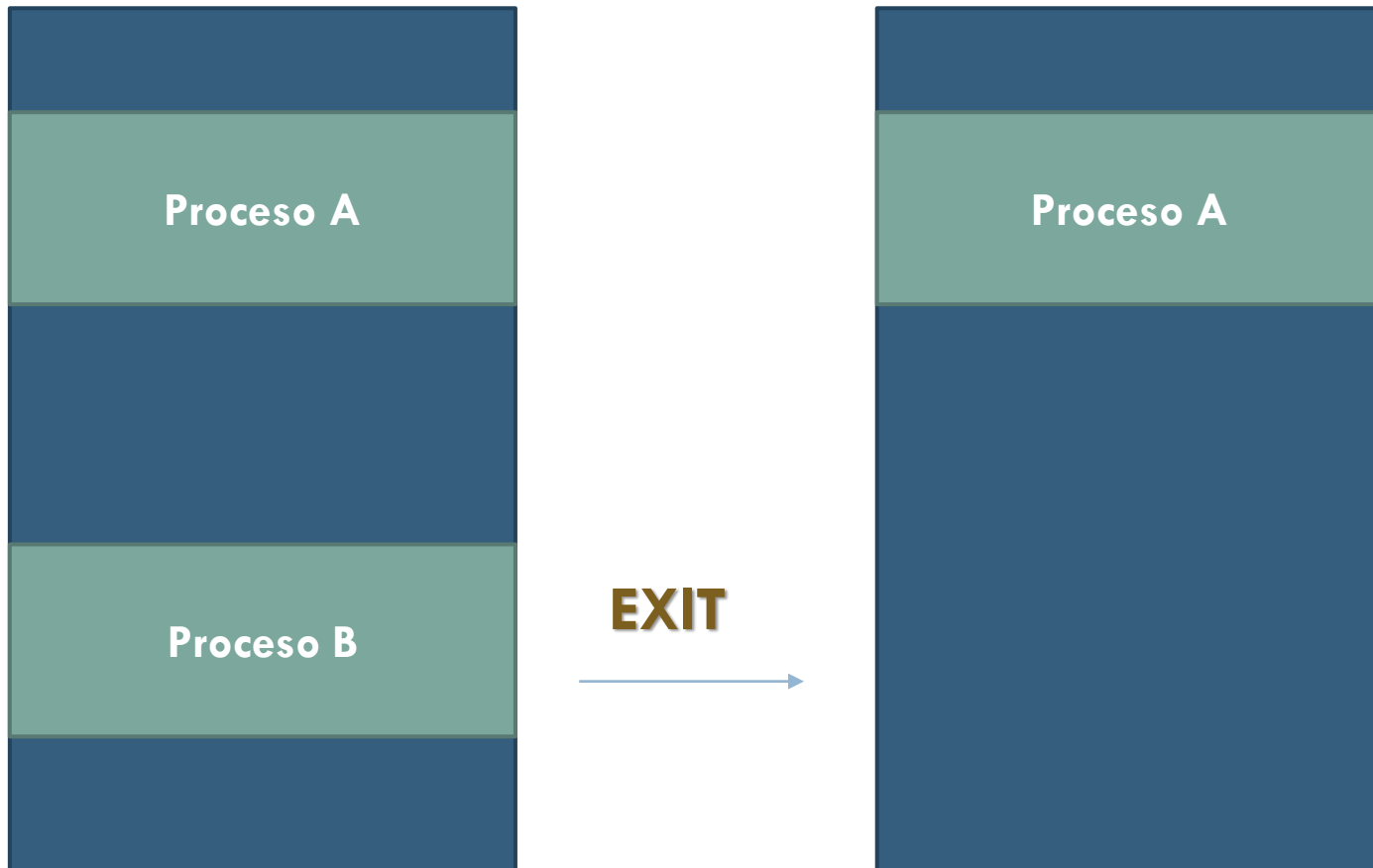
```
void exit(status) ;
```

- **Parámetros:**
 - ▣ status: valor que el padre recupera en la llamada wait()
- **Devuelve:**
- **Descripción:**
 - ▣ Finaliza la ejecución del proceso.
 - ▣ Se cierran todos los descriptores de ficheros abiertos.
 - ▣ Se liberan todos los recursos del proceso.
 - ▣ Se libera el BCP del proceso.

Servicio exit

59

Sistemas operativos: una visión aplicada



Contenidos

- Introducción a llamadas al sistema
- Mecanismo de llamada al sistema
- Llamadas para servicios de:
 - Gestión de procesos
 - Gestión de ficheros

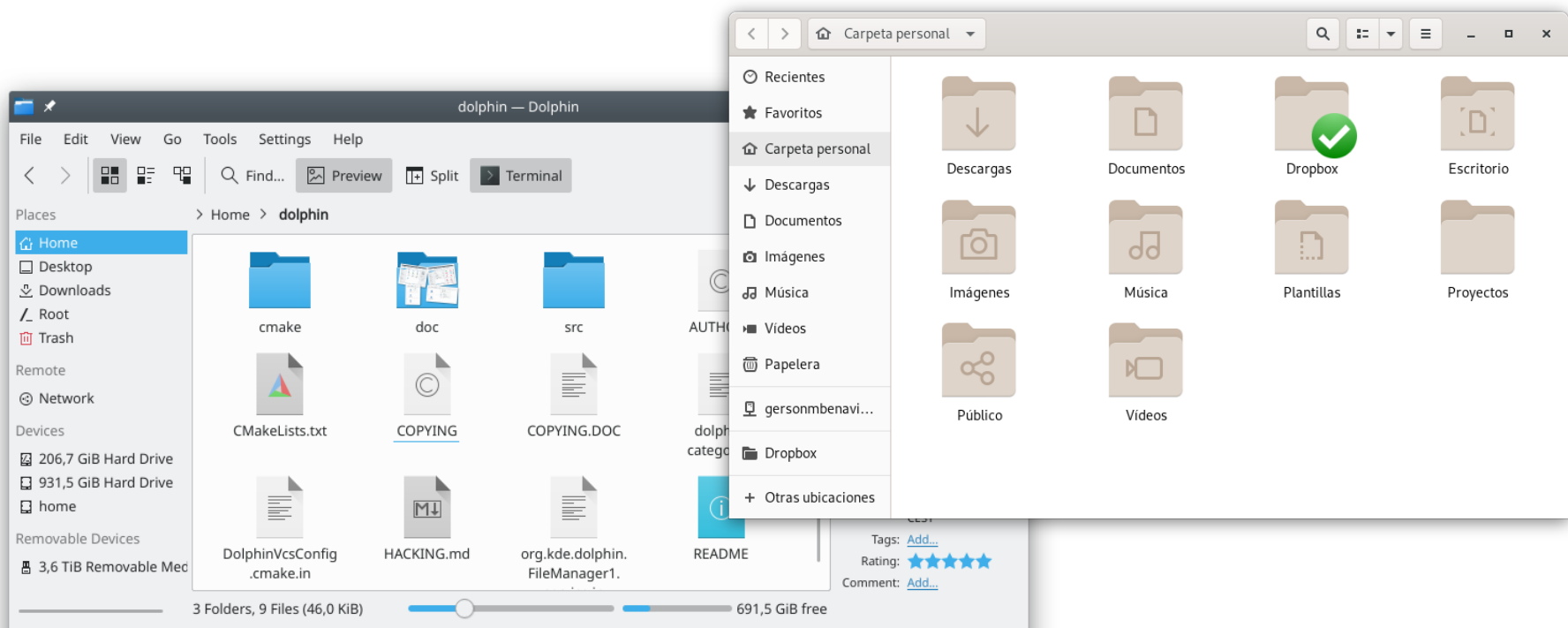
Directorio (carpetas)

61

https://es.wikipedia.org/wiki/GNOME_Archivos#/media/Archivo:GNOME_Archivos_3_36_3.png
[https://en.wikipedia.org/wiki/Dolphin_\(file_manager\)#/media/File:Dolphin-default-view.png](https://en.wikipedia.org/wiki/Dolphin_(file_manager)#/media/File:Dolphin-default-view.png)

Alejandro Calderón Mateos 

- Estructura de datos que permite agrupar un conjunto de ficheros según el criterio del usuario.



Directorio (carpetas)

62

https://es.wikipedia.org/wiki/GNOME_Archivos#/media/Archivo:GNOME_Archivos_3_36_3.png
[https://en.wikipedia.org/wiki/Dolphin_\(file_manager\)#/media/File:Dolphin-default-view.png](https://en.wikipedia.org/wiki/Dolphin_(file_manager)#/media/File:Dolphin-default-view.png)

Alejandro Calderón Mateos 

□ Nombres jerárquicos para la identificación.

□ Tipo de nombrado de directorio:

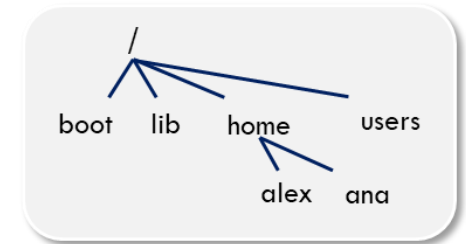
- **Nombre absoluto:** especificación del nombre respecto al directorio raíz (/ en LINUX, \ en Windows)

- **Nombre relativo:** especificación del nombre respecto a un directorio distinto del raíz.

 - Ejemplo: (estando en /users/) alex/correo.txt

 - Relativos al directorio de trabajo o actual:

basado en el directorio en el que se encuentre el usuario (directorio de trabajo)



□ Directorios especiales:

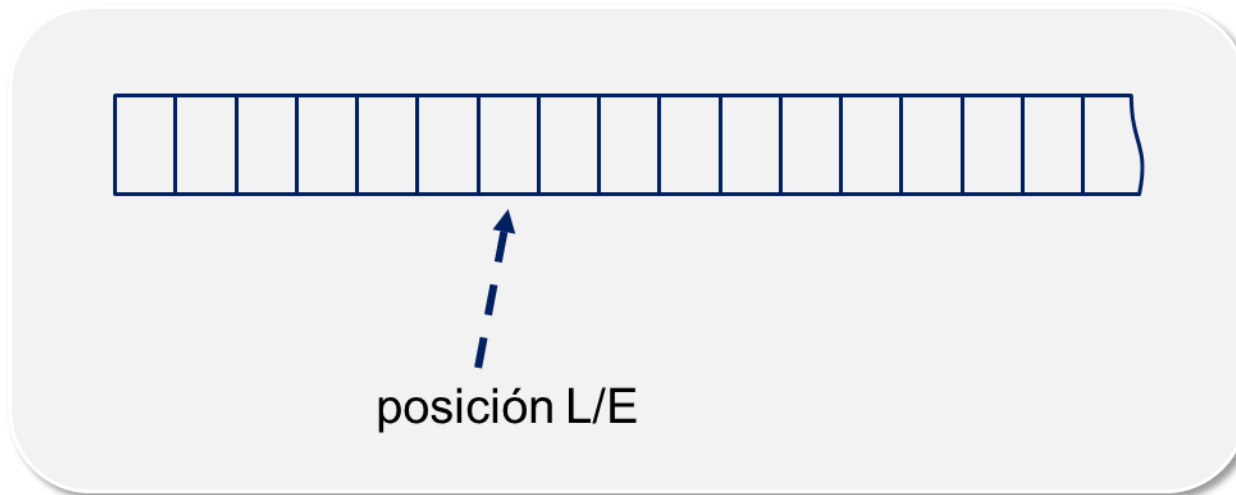
- Directorio actual o directorio de trabajo: `.` (Ej.: `cp /alex/correo.txt .`)

- Directorio padre: `..` (Ej.: `ls ..`)

- Directorio base del usuario+a: **\$HOME** (Ej.: `ls -las $HOME`)

Fichero o archivo

- Conjunto de información relacionada que ha sido definida por su creador.
- Habitualmente el contenido es representado por una secuencia o tira de bytes (visión lógica):



Atributos típicos de un fichero

- **Nombre:** identificador para los usuarios del fichero.
- **Tipo:** tipo de archivo (para los sistemas que lo necesiten)
 - Ej.: extensión (.exe, .pdf, etc.)
 - **Tipos de fichero: normales, directorios, especiales.**
- **Localización:** identificador que ayuda a la localización de los bloques del dispositivo que pertenecen al archivo.
- **Tamaño:** tamaño actual del fichero.
- **Protección:** control de qué usuario puede leer, escribir, etc.
- **Día y hora:** instante de tiempo de último acceso, de creación, etc. que permite la monitorización del uso del archivo.
- **Identificación de usuario:** identificador del creador, dueño del archivo, etc.

Nombre de ficheros y directorios

- 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.**
- Las entradas `.` y `..` pueden utilizarse para formar rutas de acceso
 - ▣ `../include/stdio.h`

Operaciones genéricas sobre ficheros

- **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:** Mueve el apuntador usado para acceder al fichero, afectando a operaciones posteriores.
- **control:** Permite manipular los atributos de un fichero.

Abstracción de fichero

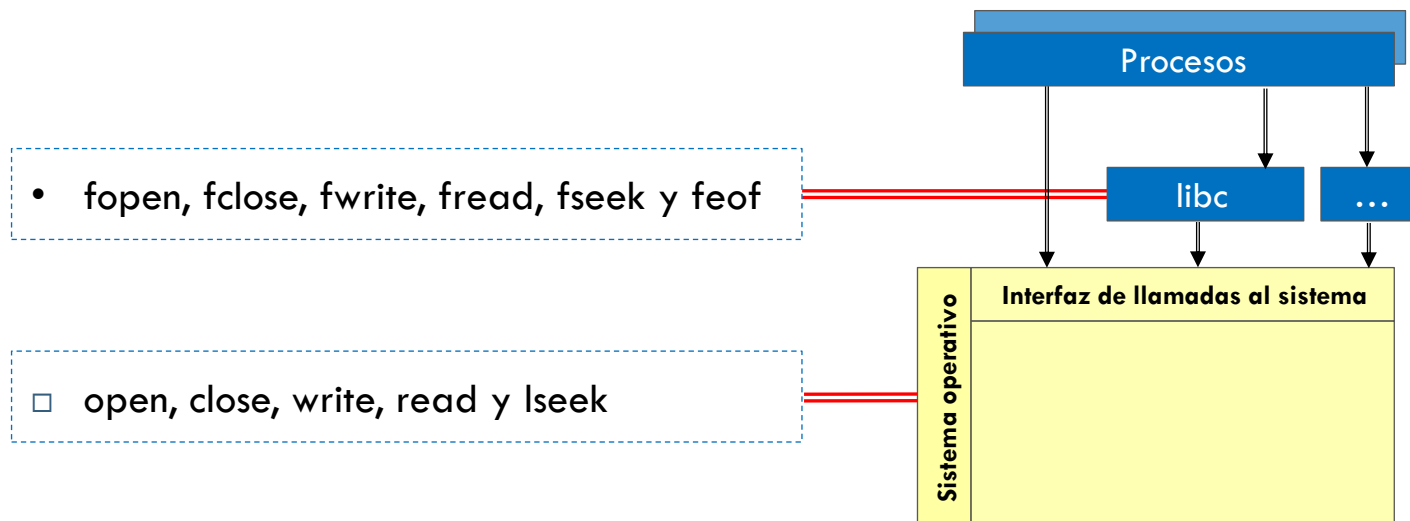
- Se mantiene un **puntero** asociado a cada fichero abierto.
 - ▣ Indica la posición a partir de la cual se realizará la siguiente operación.
- La mayoría de operaciones usan **descriptores de ficheros**:
 - ▣ Identifica una sesión de trabajo con un fichero:
 - Un número entero entre 0 y 64K.
 - Se 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

Llamadas al sistema vs librería sistema

sistema de ficheros

68

Alejandro Calderón Mateos 



Llamadas al sistema vs librería sistema

escribir en fichero

69

Alejandro Calderón Mateos 

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

int main ( int argc, char *argv[] )
{
    int fd1 ;
    char str1[10] ;
    int nb ;

    fd1 = open ("/tmp/txt1",
                O_CREAT|O_RDWR, S_IRWXU);
    if (-1 == fd1) {
        perror("open:");
        exit(-1);
    }

    strcpy(str1,"hola");
    nb = write (fd1,str1,strlen(str1));
    printf("bytes escritos = %d\n",nb);

    close (fd1);
    return (0) ;
}
```

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

int main ( int argc, char *argv[] )
{
    FILE *fd1 ;
    char str1[10] ;
    int nb ;

    fd1 = fopen ("/tmp/txt2","w+");
    if (NULL == fd1) {
        printf("fopen: error\n");
        exit(-1) ;
    }

    strcpy(str1,"mundo");
    nb = fwrite (str1,strlen(str1),1,fd1);
    printf("items escritos = %d\n",nb);

    fclose (fd1) ;
    return (0) ;
}
```

Llamadas al sistema vs librería sistema

leer de fichero

70

Alejandro Calderón Mateos 

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

int main ( int argc, char *argv[] )
{
    int fd1 ;
    char str1[10] ;
    int nb, i ;

    fd1 = open ("/tmp/txt1", O_RDONLY);
    if (-1 == fd1) {
        perror("open:");
        exit(-1);
    }

    i=0;
    do {
        nb = read (fd1, &(str1[i]), 1);
        i++;
    } while (nb != 0) ;
    str1[i] = '\0';
    printf("%s\n", str1);

    close (fd1);
    return (0);
}
```

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

int main ( int argc, char *argv[] )
{
    FILE *fd1 ;
    char str1[10] ;
    int nb, i ;

    fd1 = fopen ("/tmp/txt2", "r");
    if (NULL == fd1) {
        printf("fopen: error\n");
        exit(-1) ;
    }

    i=0;
    do {
        nb = fread (&(str1[i]), 1, 1, fd1) ;
        i++ ;
    } while (nb != 0) ; /* feof() */
    str1[i] = '\0' ;
    printf("%s\n", str1);

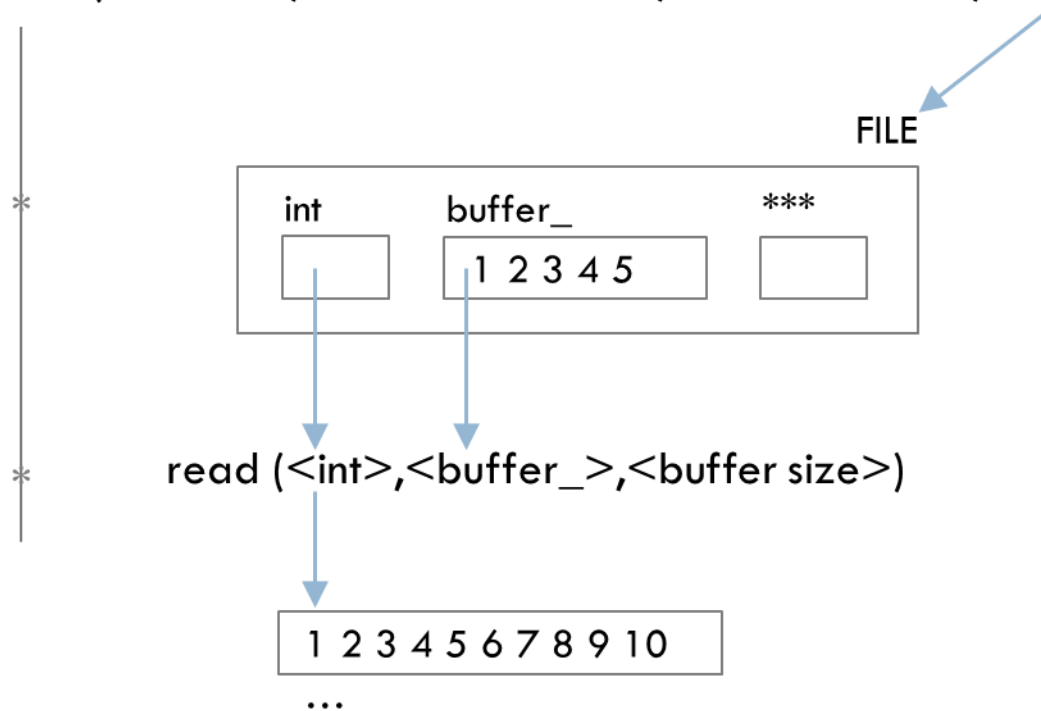
    fclose (fd1);
    return (0);
}
```

Funcionalidad extendida

71

Alejandro Calderón Mateos 

`fread (<buffer>, <size of one elto>, <num. of eltos>, <FILE *>)`



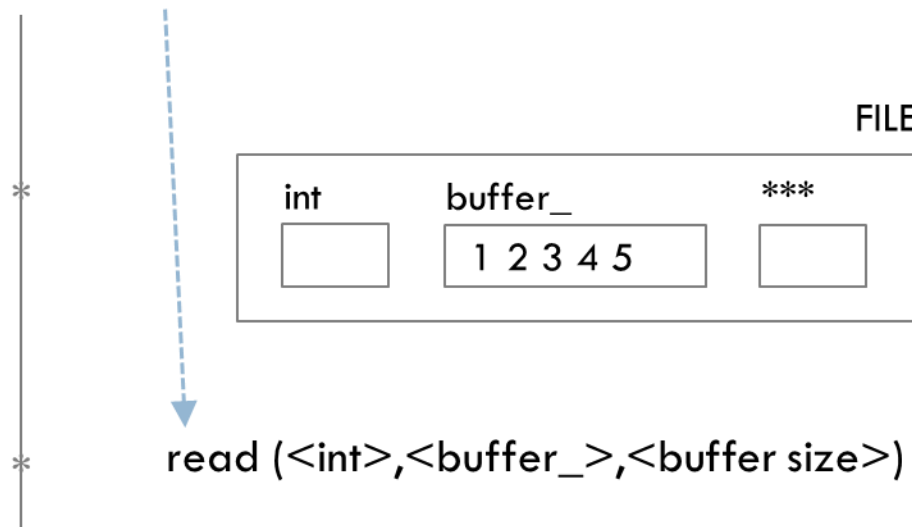
Un puntero a FILE contiene el descriptor de fichero y un buffer intermedio (principalmente)...

Funcionalidad extendida

72

Alejandro Calderón Mateos 

`fread (<buffer>, <size of one elto>, <num. of eltos>, <FILE *>)`



1 2 3 4 5 6 7 8 9 10
...

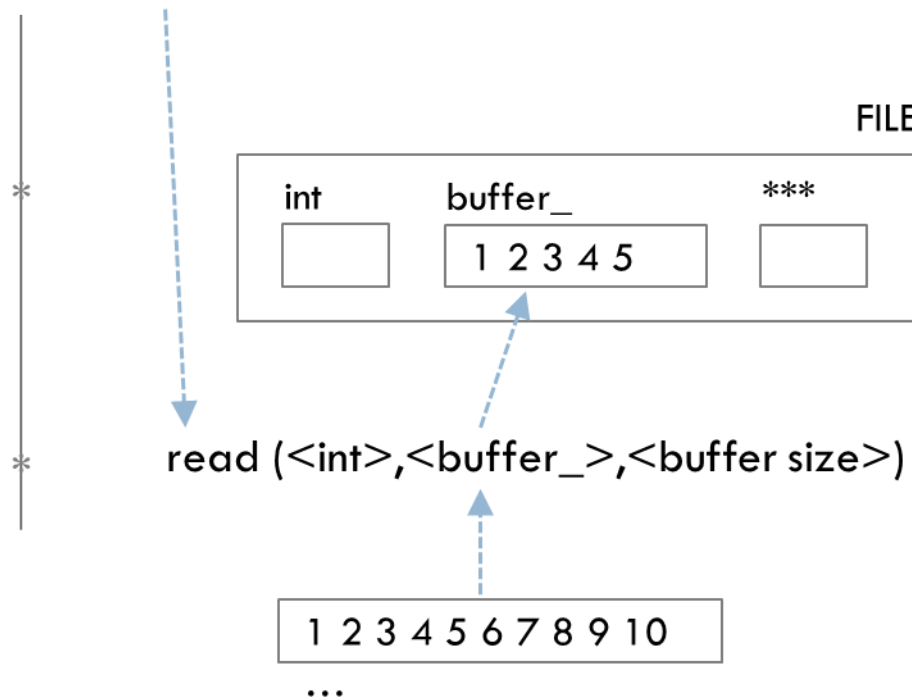
... de manera que cuando se pide la primera lectura, se realiza una lectura sobre el buffer (cuyo tamaño es mayor que el elemento pedido)...

Funcionalidad extendida

73

Alejandro Calderón Mateos 

`fread (<buffer>, <size of one elto>, <num. of eltos>, <FILE *>)`



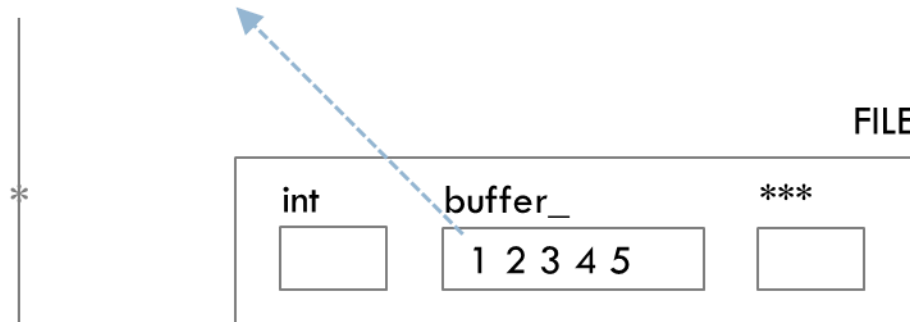
... los datos se cargan en el buffer y se copian la porción pedida al proceso que hace el fread...

Funcionalidad extendida

74

Alejandro Calderón Mateos 

`fread (<buffer>, <size of one elto>, <num. of eltos>, <FILE *>)`



`read (<int>, <buffer_>, <buffer size>)`

1 2 3 4 5 6 7 8 9 10

...

...y la siguiente vez que se hace una lectura, si está en el buffer (memoria) se copia directamente de él. De esta forma se reduce las llamadas al sistema, lo que acelera la ejecución.

Fichero o archivo: interfaz C99

75

Alejandro Calderón Mateos 

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>

#define BSIZE 1024

int main ( int argc, char *argv[] )
{
    FILE *fd1 ; int i; double tiempo ;
    char buffer1[BSIZE] ;
    struct timeval ti, tf;

    gettimeofday(&ti, NULL);
    fd1 = fopen ("/tmp/txt2", "w+");
    if (NULL == fd1) {
        printf("fopen: error\n");
        exit(-1) ;
    }
    setbuffer(fd1,buffer1,BSIZE) ;
    for (i=0; i<8*1024; i++)
        fprintf(fd1,"%d",i);
    fclose (fd1) ;

    gettimeofday(&tf, NULL);
    tiempo= (tf.tv_sec - ti.tv_sec)*1000 +
            (tf.tv_usec - ti.tv_usec)/1000.0;
    printf("%g milisegundos\n", tiempo);
    return (0) ;
}
```

❑ Compilar (gcc -o b b.c)
y ejecutar con

❑ BSIZE=1024

❑ BSIZE=0

❑ Resultados:

❑ BSIZE=1024

■ T=0.902 milisegundos

❑ BSIZE=0

■ T=14.866 milisegundos

x 15

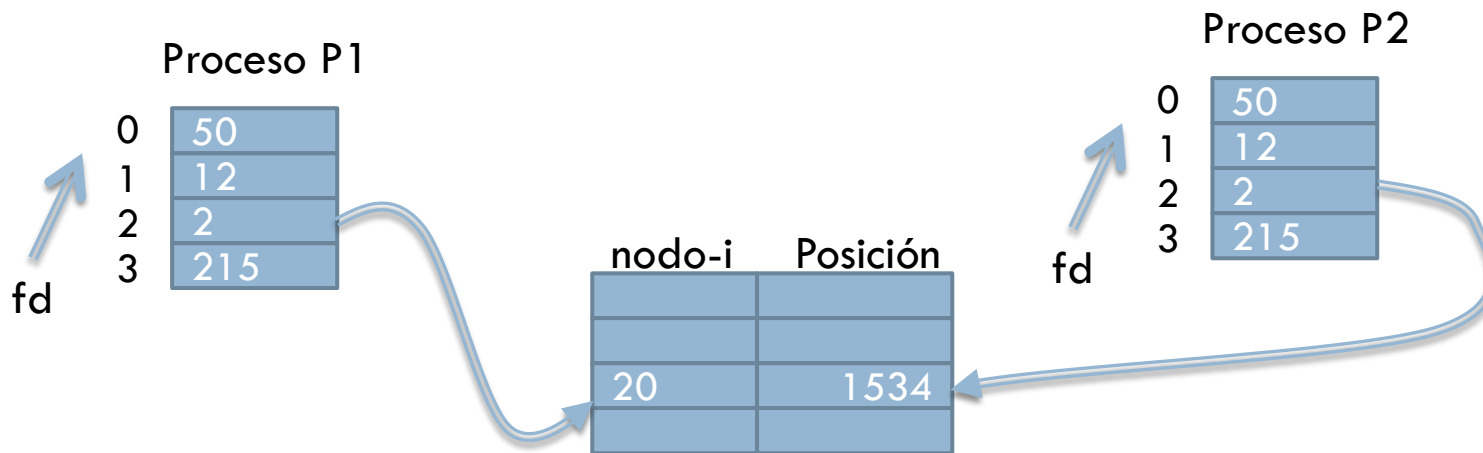
Interacción entre procesos y ficheros

76

Sistemas operativos: una visión aplicada



- 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 rwxr-xr-x

CREAT – Creación de fichero

Servicio	<pre>#include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> int creat(char *name, mode_t mode);</pre>
Argumentos	name Nombre de fichero mode Bits de permiso para el fichero
Devuelve	Devuelve un descriptor de fichero ó -1 si error.
Descripción	Crea un fichero nuevo si no existe o trunca si existe.

CREAT – Creación de fichero

□ Ejemplos:

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

□ Descripción:

- El fichero se abre para escritura.
- Si no existe crea un fichero vacío.
 - $UID_{\text{dueño}} = UID_{\text{efectivo}}$
 - $GID_{\text{dueño}} = GID_{\text{efectivo}}$
- Si existe lo trunca sin cambiar los bits de permiso.

UNLINK – Borrado de fichero

Servicio	<pre>#include <unistd.h> int unlink(const char* path);</pre>
Argumentos	<code>path</code> 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

Servicio	<pre>#include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> int open(char *name, int flag, ...);</pre>
Argumentos	<ul style="list-style-type: none">▣ name nombre del fichero (puntero al primer caracter).▣ flags opciones de apertura:<ul style="list-style-type: none">■ O_RDONLY Sólo lectura■ O_WRONLY Sólo escritura■ O_RDWR Lectura y escritura■ O_APPEND Posicionar el puntero de acceso al final del fichero abierto■ O_CREAT Si existe no tiene efecto. Si no existe lo crea■ O_TRUNC Trunca si se abre para escritura
Devuelve	Un descriptor de fichero ó -1 si hay error.
Descripción	Apertura de fichero (o creación con O_CREAT).

OPEN – Apertura de fichero

□ Ejemplos:

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

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

CLOSE – Cierre de fichero

Servicio	<pre>#include <unistd.h> int close(int fd);</pre>
Argumentos	fd descriptor de fichero.
Devuelve	Devuelve 0 ó -1 si error.
Descripción	El proceso pierde la asociación a un fichero.

READ – Lectura de fichero

Servicio	<pre>#include <sys/types.h> ssize_t read(int fd, void *buf, size_t n_bytes);</pre>
Argumentos	<ul style="list-style-type: none">❑ <code>fd</code> descriptor de fichero❑ <code>buf</code> zona donde almacenar los datos❑ <code>n_bytes</code> número de bytes a leer
Devuelve	Número de bytes realmente leídos ó -1 si error.
Descripción	<ul style="list-style-type: none">❑ Transfiere <code>n_bytes</code>. 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

Servicio	<pre>#include <sys/types.h> ssize_t write(int fd, void *buf, size_t n_bytes);</pre>
Argumentos	<ul style="list-style-type: none">❑ <code>fd</code> descriptor de fichero❑ <code>buf</code> zona de datos a escribir❑ <code>n_bytes</code> número de bytes a escribir
Devuelve	Número de bytes realmente escritos ó -1 si error.
Descripción	<ul style="list-style-type: none">❑ Transfiere <code>n_bytes</code>. 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

Servicio	<pre>#include <sys/types.h> #include <unistd.h> off_t lseek(int fd, off_t offset, int whence);</pre>
Argumentos	<ul style="list-style-type: none">❑ fd Descriptor de fichero❑ offset desplazamiento❑ whence base del desplazamiento
Devuelve	La nueva posición del puntero ó -1 si error.
Descripción	<ul style="list-style-type: none">❑ Coloca el puntero de acceso asociado a fd❑ La nueva posición se calcula:<ul style="list-style-type: none">■ SEEK_SET posición = offset■ SEEK_CUR posición = posición actual + offset■ SEEK_END posición = tamaño del fichero + offset

FNCTL – Modificación de atributos

Servicio	<pre>#include <sys/types.h> int fnctl(int fildes, int cmd /* arg*/ ...);</pre>
Argumentos	<ul style="list-style-type: none">▣ <code>fildes</code> descriptor de ficheros▣ <code>cmd</code> 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

Servicio	<pre>#include <unistd.h> int dup(int fd);</pre>
Argumentos	<ul style="list-style-type: none">▣ <code>fd</code> descriptor de fichero
Devuelve	Un descriptor de fichero que comparte todas las propiedades del <code>fd</code> ó -1 si error.
Descripción	<ul style="list-style-type: none">▣ Crea un nuevo descriptor de fichero que tiene en común con el anterior:<ul style="list-style-type: none">■ 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

Servicio	<pre>#include <unistd.h> int ftruncate(int fd, off_t length);</pre>
Argumentos	<ul style="list-style-type: none">▣ 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

Servicio	<pre>#include <sys/types.h> #include <sys/stat.h> int stat(char *name, struct stat *buf); int fstat(int fd, struct stat *buf);</pre>
Argumentos	<ul style="list-style-type: none">▣ 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	Devuelve 0 ó -1 si error.
Descripción	Obtiene información sobre un fichero y la almacena en una estructura de tipo struct stat

STAT – Información sobre un fichero

```
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 modificacion */  
    time_t    st_ctime;   /* último modificacion de datos */  
};
```

STAT – Información sobre un fichero

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

<code>S_ISDIR(s.st_mode)</code>	Cierto si directorio
<code>S_ISCHR(s.st_mode)</code>	Cierto si especial de caracteres
<code>S_ISBLK(s.st_mode)</code>	Cierto si especial de bloques
<code>S_ISREG(s.st_mode)</code>	Cierto si fichero normal
<code>S_ISFIFO(s.st_mode)</code>	Cierto si pipe o FIFO

UTIME – Alteración de atributos de fecha

Servicio	<pre>#include <sys/stat.h> #include <utime.h> int utime(char *name, struct utimbuf *times);</pre>
Argumentos	<ul style="list-style-type: none">▣ name nombre del fichero▣ times estructura con las fechas de último acceso y modificación.<ul style="list-style-type: none">■ 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 (1/3)

```
#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);
    }

    /* 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);
}
```

```
#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;
```

Ejemplo: Copia de un fichero en otro (2/3)

```
#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);
    }

    /* 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);
}
```

```
/* 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 (3/3)

```
#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);
    }

    /* 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);
}
```

```
/* 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)

```
void main(void) {
    pid_t pid;
    int status, fd;

    close(1) ;

    fd = open("fichero", O_WRONLY|O_CREAT|O_TRUNC, 0644);
    if (fd < 0)    {
        perror("open");
        exit(-1);
    }
    pid = fork();
    // ...
}
```


Servicios POSIX para directorios

□ **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

- `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

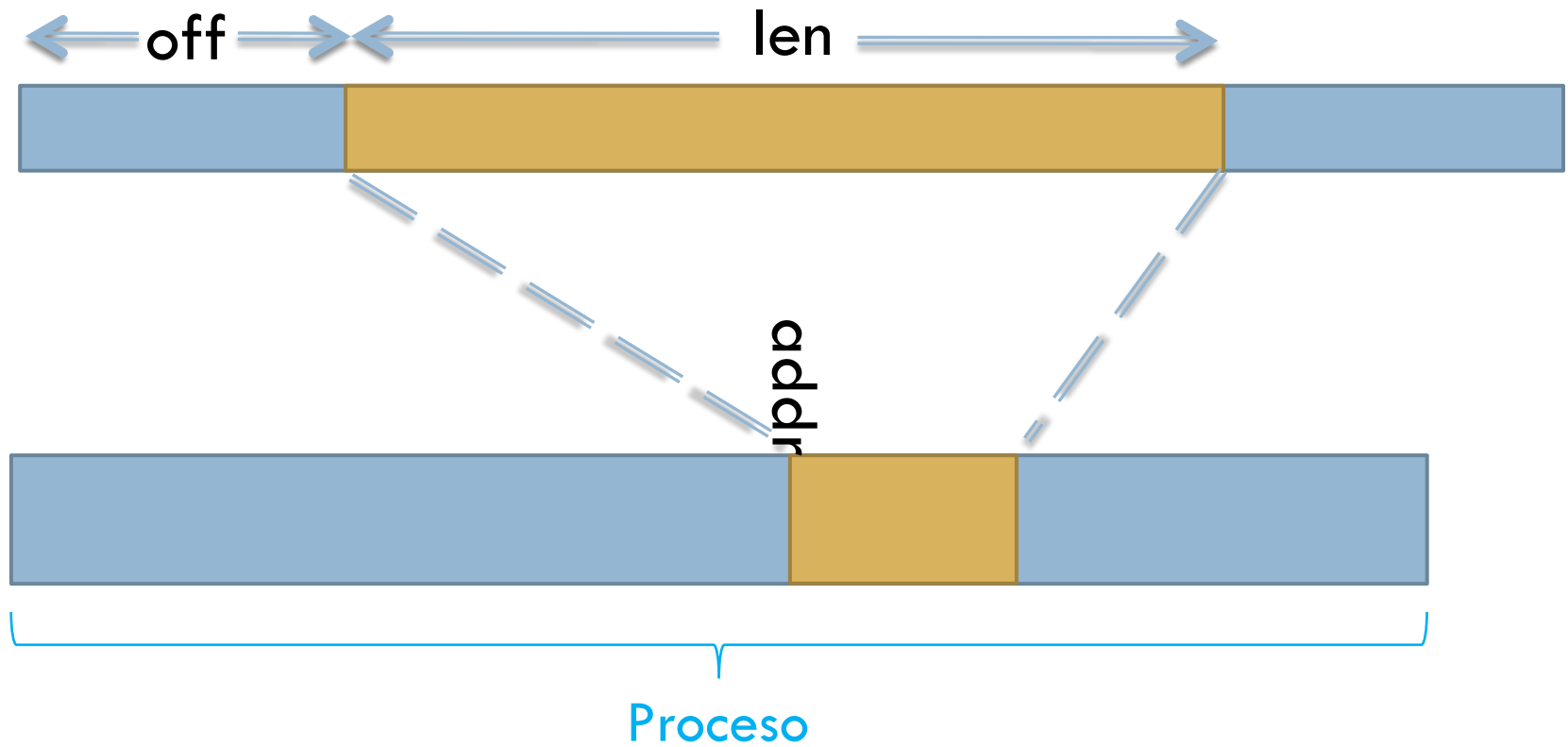
Contenidos

- Introducción a llamadas al sistema
- Mecanismo de llamada al sistema
- Llamadas para servicios de:
 - ▣ Gestión de procesos
 - ▣ Gestión de ficheros
 - Proyección de fichero

Proyección POSIX

100

Sistemas operativos: una visión aplicada



Proyección POSIX: mmap

Servicio	<pre>void *mmap(void *addr, size_t len, int prot, int flags, int fildes, off_t off);</pre>
Argumentos	<ul style="list-style-type: none">▣ <code>addr</code> dirección donde proyectar. Si <code>NULL</code> el SO elige una.▣ <code>len</code> especifica el número de bytes a proyectar.▣ <code>prot</code> el tipo de acceso (lectura, escritura o ejecución).▣ <code>flags</code> especifica información sobre el manejo de los datos proyectados (compartidos, privado, etc.).▣ <code>fildes</code> representa el descriptor de fichero del fichero o descriptor del objeto de memoria a proyectar.▣ <code>off</code> desplazamiento dentro del fichero a partir del cual se realiza la proyección.
Devuelve	Devuelve la dirección de memoria donde se ha proyectado el fichero.
Descripción	Establece una proyección entre el espacio de direcciones de un proceso y un descriptor de fichero u objeto de memoria compartida.

Proyección POSIX: mmap

- `int prot`: 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.
- `int flags`: 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: munmap

Servicio	<code>void munmap(void *addr, size_t len);</code>
Argumentos	<ul style="list-style-type: none">▣ <code>addr</code> dirección donde está proyectado.▣ <code>len</code> especifica el número de bytes proyectados.
Devuelve	Nada.
Descripción	Desproyecta parte del espacio de direcciones de un proceso comenzando en la dirección <code>addr</code> .

Ejemplo: copia de un fichero (1 / 2)

```
#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;
}
```

```
#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);
```


Ejemplo: copia de un fichero (2/2)

```
#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;
}
```

```
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;
}
```

Ejemplo: contar el número de blancos en un fichero (1 / 2)

```
#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;
}
```

```
#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;
```

Ejemplo: contar el número de blancos en un fichero (2/2)

```
#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;
}
```

```
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;
}
```

SISTEMAS OPERATIVOS: INTRODUCCIÓN Y CONCEPTOS BÁSICOS



Servicios del Sistema Operativo