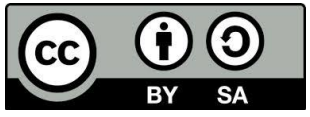
Llamadas al sistema

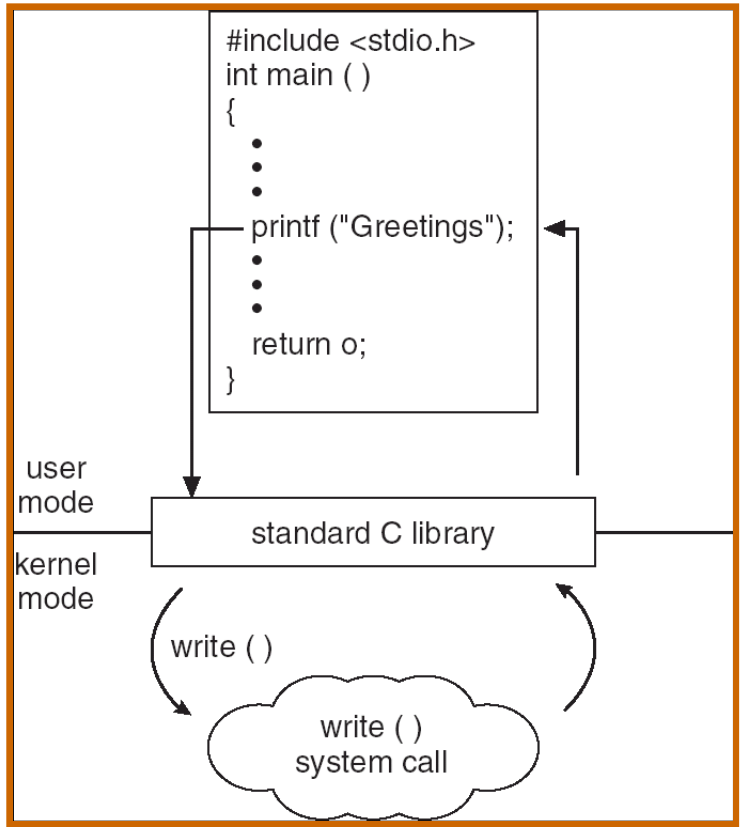
Arquitectura Interna de Linux - 2016



J.C. Sáez, M. Prieto
High Performance Computing Division ArTeCs Group
Universidad Complutense de Madrid

Llamadas al Sistema (I)

- Normalmente los procesos de usuario no hacen uso explícito de las llamadas al sistema
  - Las invocan vía una función de biblioteca (libc)



Llamadas al Sistema (II)

- El comando `strace` permite saber qué llamadas al sistema invoca un programa (Uso: `strace <comando_programa>`)

terminal

```
kernel@debian:~$ echo hola > a.txt
kernel@debian:~$ cat a.txt
hola
kernel@debian:p2$ strace cat a.txt
execve("/bin/cat", ["cat", "a.txt"], [/* 21 vars */]) = 0
brk(0)                                  = 0x21ce000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8b6d37e000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=108106, ...}) = 0
...
open("a.txt", O_RDONLY)                 = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=5, ...}) = 0
fadvise64(3, 0, 0, POSIX_FADV_SEQUENTIAL) = 0
mmap(NULL, 1056768, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8b6d25e000
read(3, "hola\n", 1048576)              = 5
write(1, "hola\n", 5hola
)                       = 5
read(3, "", 1048576)                    = 0
munmap(0x7f8b6d25e000, 1056768)         = 0
close(3)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
```

Llamadas al Sistema (III)

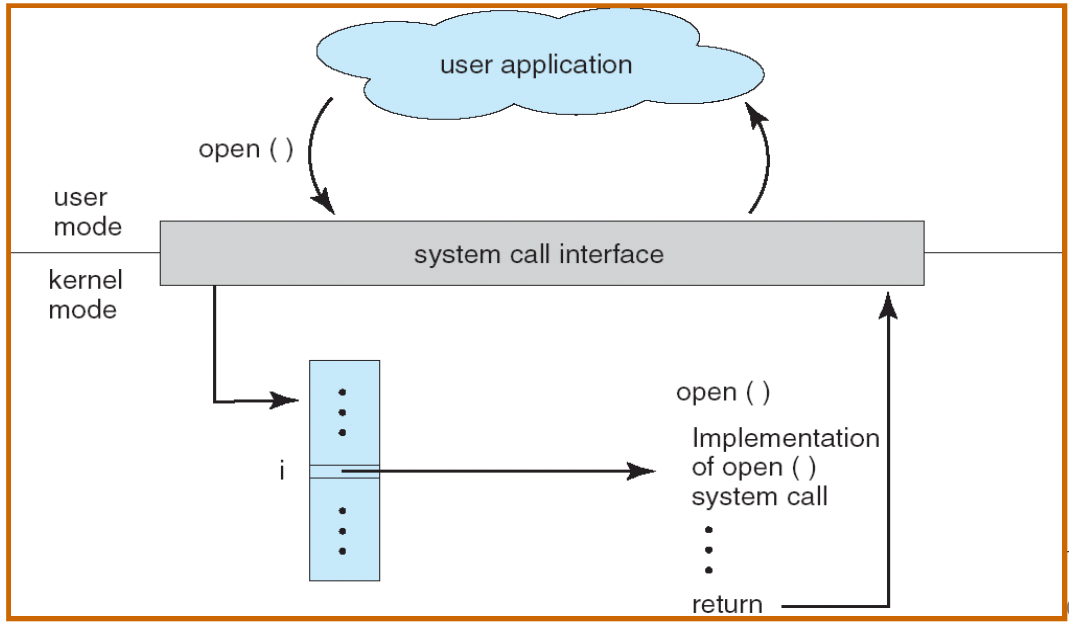
- Cuando un programa de usuario invoca una llamada al sistema se acaba invocando una función del código del SO
- En sistemas monolíticos como Linux esa función reside en espacio de kernel
  - Invocar la función $\rightarrow$ Forzar transición a modo kernel

Transición modo usuario - modo kernel

- La libc ejecuta una instrucción máquina especial tipo TRAP
  - x86: `syscall`, `sysenter`, `int`; ARM: `swi`
- TRAP genera excepción software
  - El SO ejecuta una rutina de tratamiento de excepción
- **Problema:** Un único punto de entrada al kernel para todas las *syscalls*

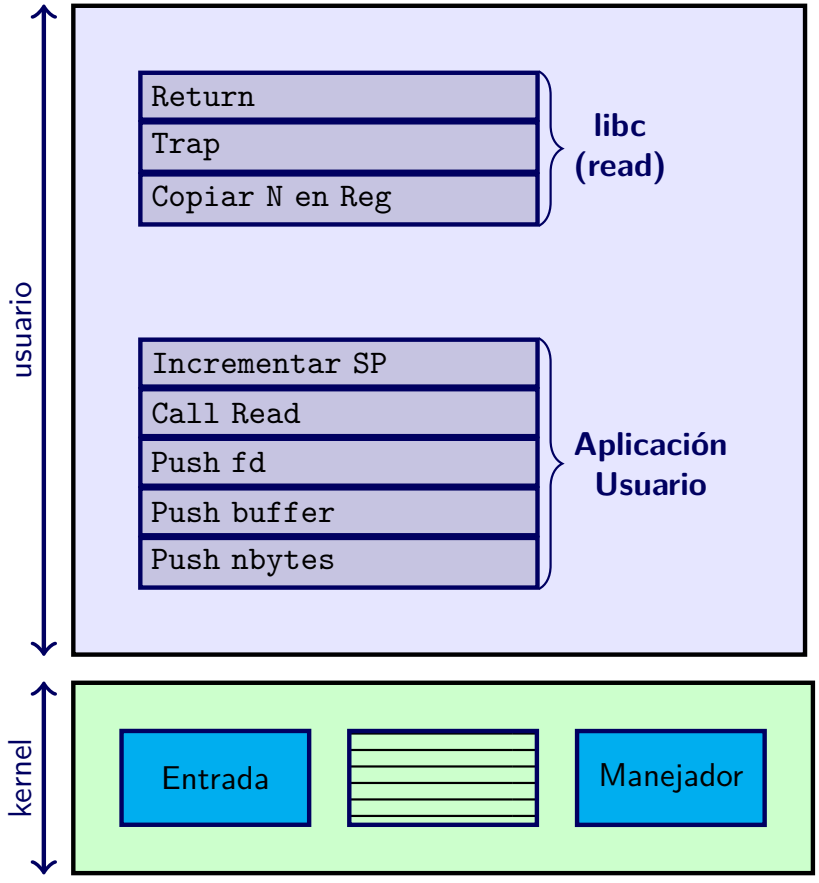
Llamadas al Sistema (IV)

- El mecanismo para implementar llamadas al sistema es dependiente de la arquitectura
- Cada llamada al sistema se identifica con un número que puede variar con la arquitectura



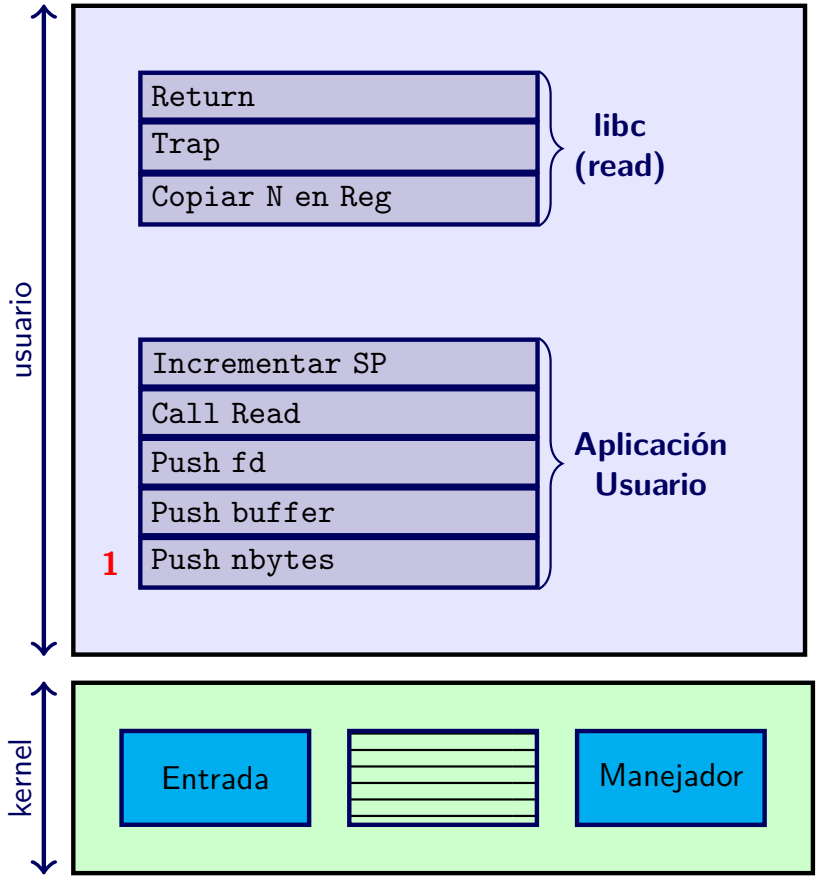
Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



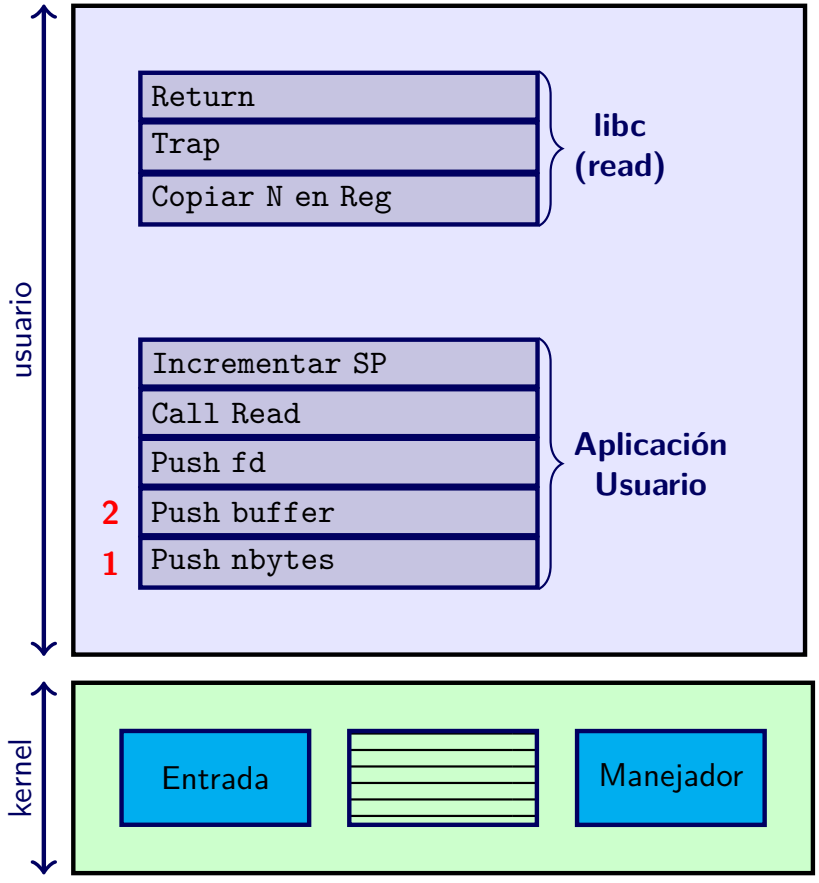
Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



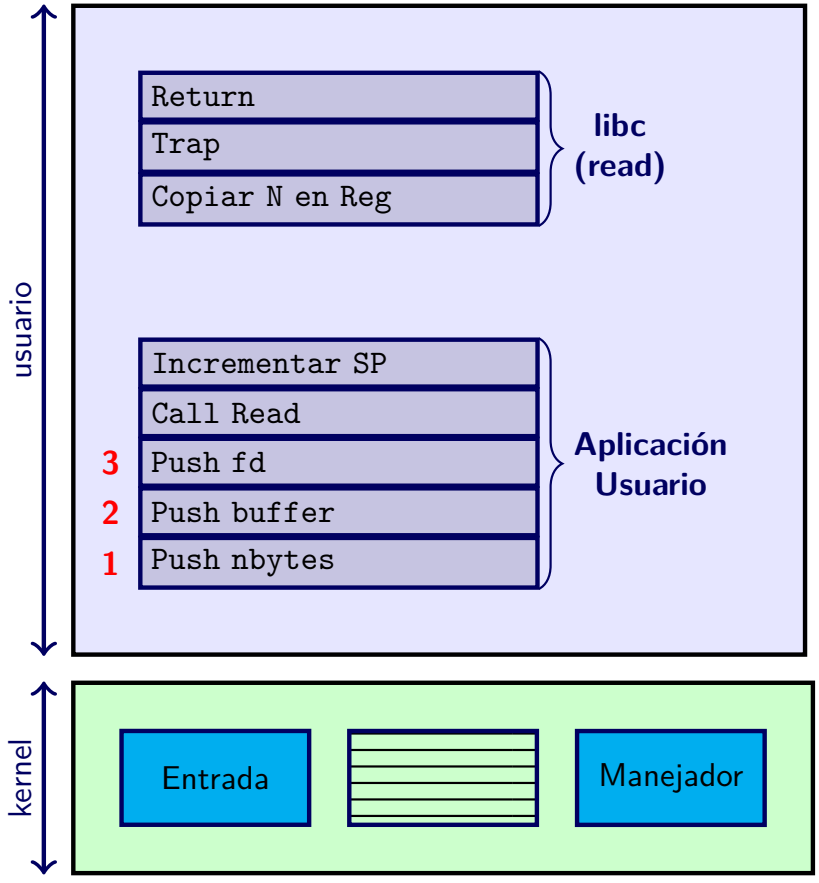
Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



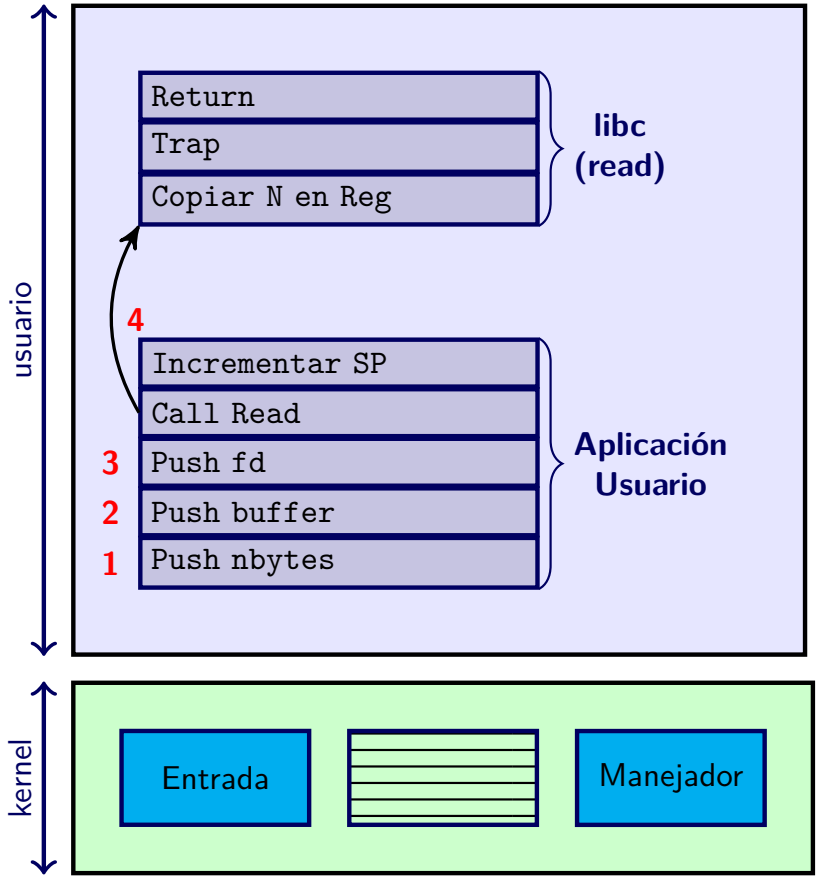
Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



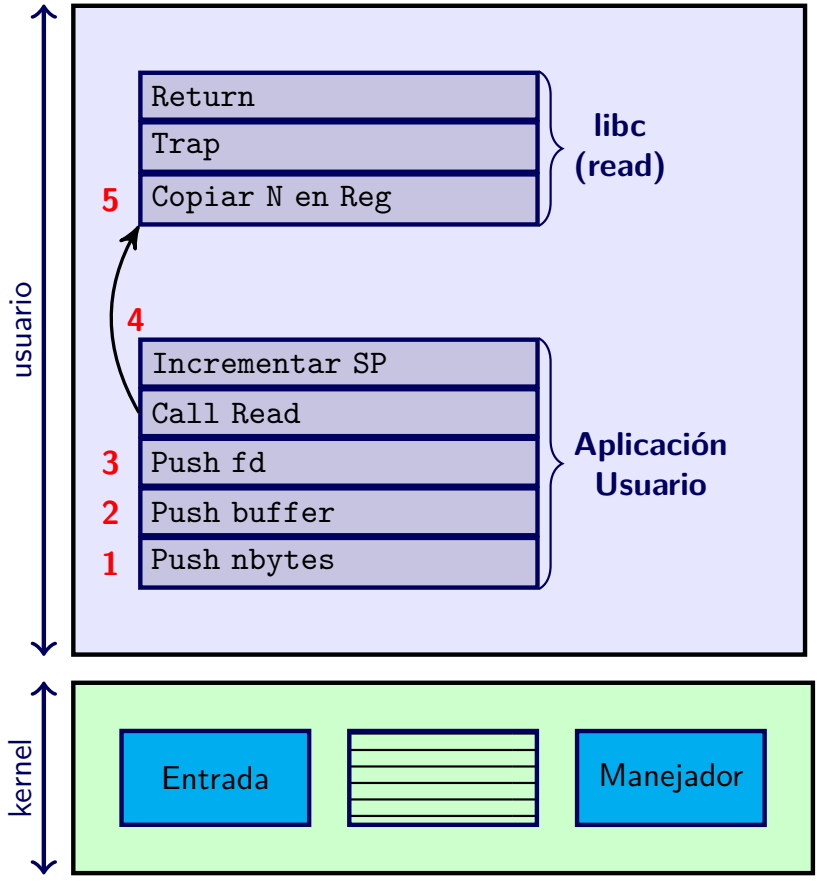
Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



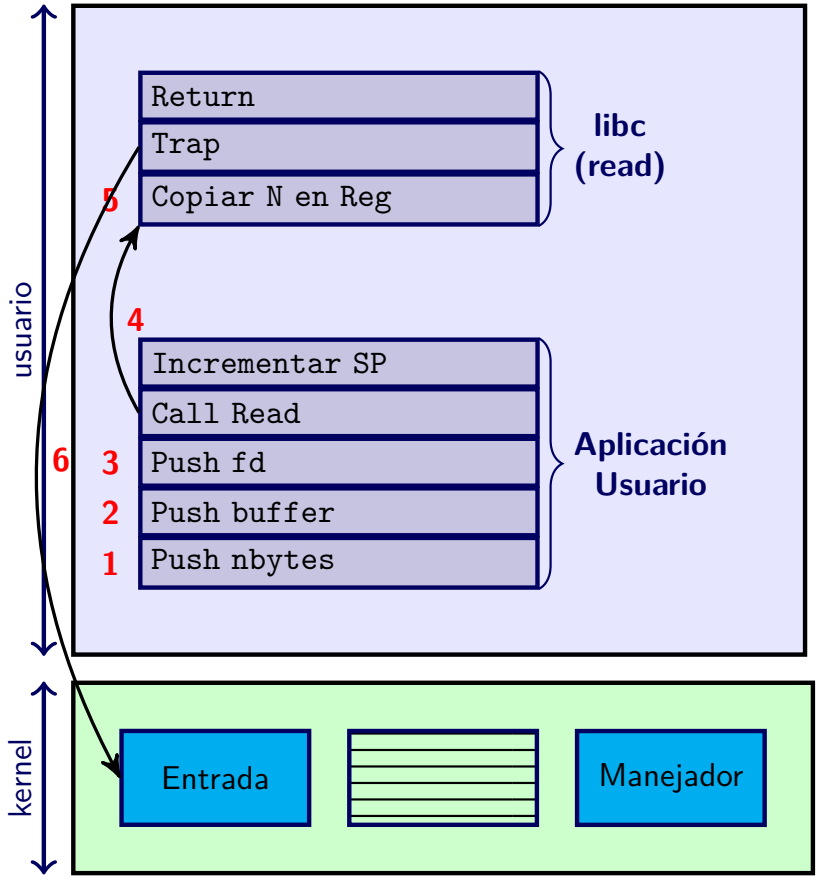
Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



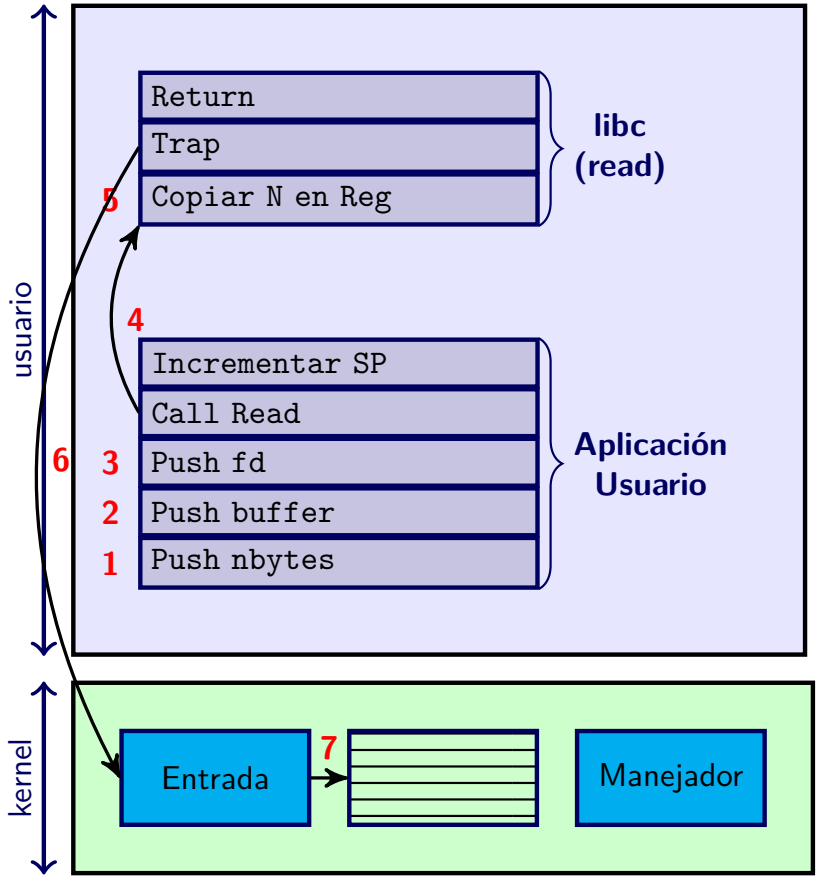
Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



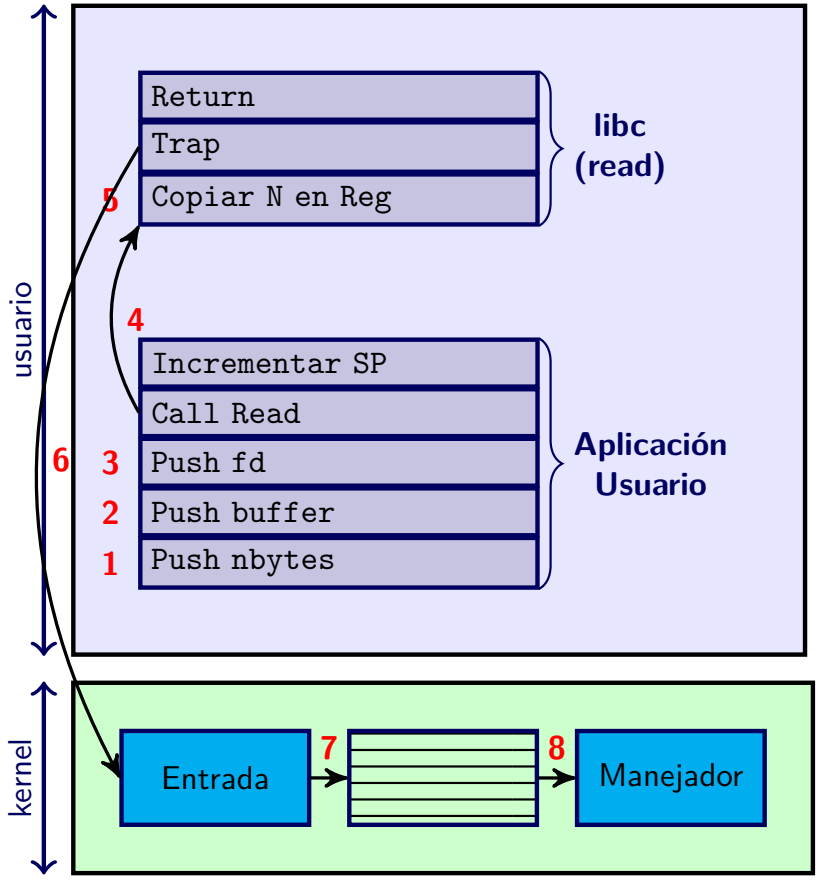
Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



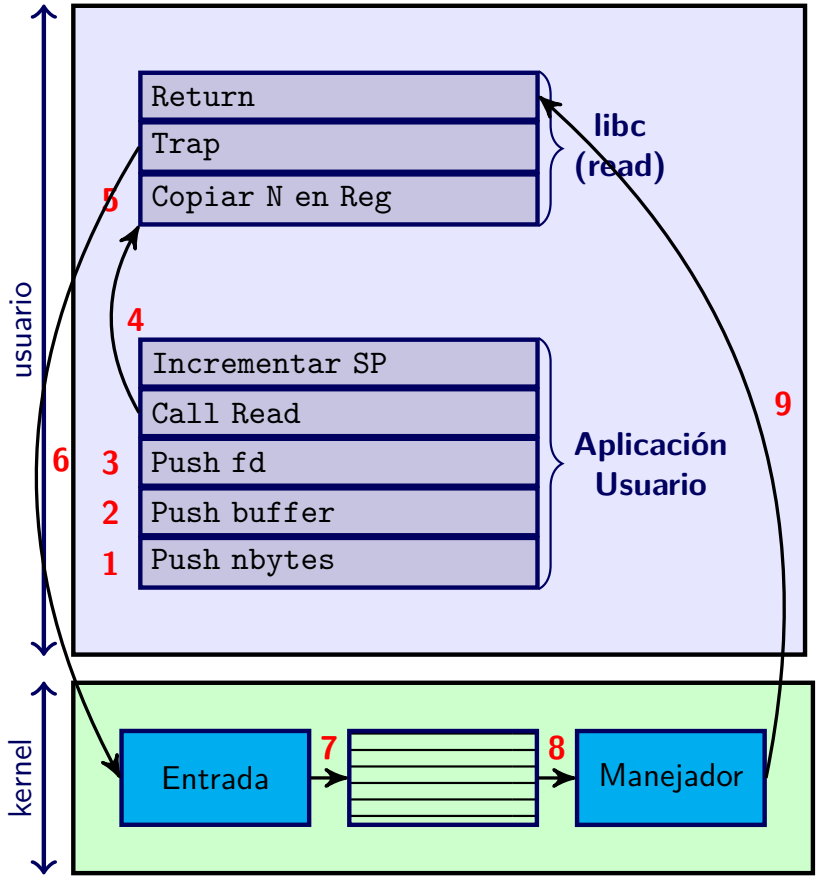
Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



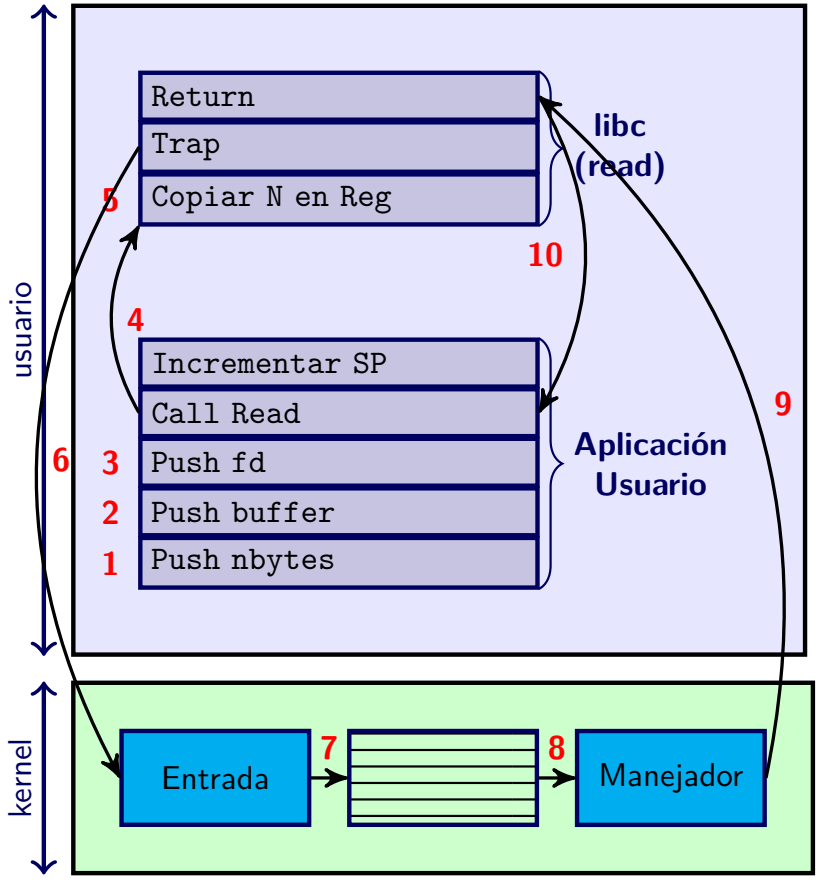
Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



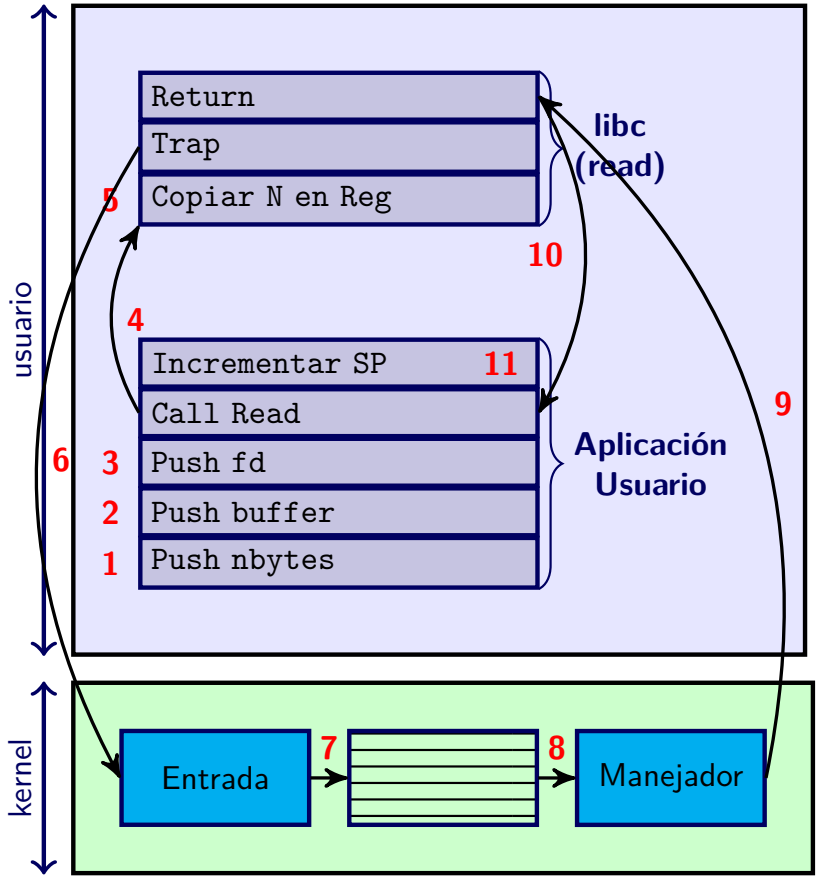
Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



Llamadas al Sistema (V)

```
count = read (fd, buffer, nbytes)
```



Llamadas al Sistema en Linux

- Dos partes:
  1. Transición Usuario-kernel-Usuario – dependiente de la arquitectura
     - Gestión paso parámetros
     - Transición modo
     - Redirección a la función correspondiente
  2. Handler function - Implementación de la llamada (independiente de la arquitectura)

Modo usuario

Invocación vía `syscall()`

```
#include <sys/syscall.h>
#define __NR_gettid 186

long gettid()
{
  return (long) syscall(__NR_gettid);
}
```

Modo kernel

Implementación de la llamada al sistema

`kernel/sys.c`

```
SYSCALL_DEFINE0(gettid)
{
    return task_tgid_vnr(current);
}
```

Llamadas al Sistema en Linux

- Dos partes:
  1. Transición Usuario-kernel-Usuario – dependiente de la arquitectura
     - Gestión paso parámetros
     - Transición modo
     - Redirección a la función correspondiente
  2. Handler function - Implementación de la llamada (independiente de la arquitectura)

Modo usuario

Invocación vía `syscall()`

```
#include <sys/syscall.h>
#define __NR_gettid 186

long gettid()
{
  return (long) syscall(__NR_gettid);
}
```

Modo kernel

Implementación de la llamada al sistema

`kernel/sys.c`

```
SYSCALL_DEFINE0(gettid)
{
    return task_tgid_vnr(current);
}
```

Esta macro define una función

```
long sys_gettid(void);
```

Llamadas al Sistema en Linux (II)

Paso de Parámetros x86

- A través de registros
  - NR SYSCALL `eax`
  - Parámetros: `ebx,ecx,edx,esi` y `edi` (6 ó + parámetros: puntero al espacio de usuario)
    - `copy_from_user, copy_to_user`

Conmutación modo

- Varios métodos
  - Tradicional: `int 0x80` (interrupción software 128) – call gate –
  - Pentium II y Superiores: `sysenter` y `sysexit`
- Se llaman indirectamente para conservar compatibilidad via "puntero a función" `call 0xFFFFE000`

Llamadas al Sistema (III)

Valores de Retorno (long)

- x86: registro `eax`
  - 0 o Positivo: éxito
  - Negativo (-1 .. -511): error
- Definición de errores (positivo)
  - `<asm-generic/errno-base.h>` (errores clásicos)
  - `<asm-generic/errno.h>`
- La función `syscall()` devuelve -1 en caso de error
  - El código de error queda almacenado en la variable `errno` (usar `perror()`)

Llamadas al Sistema en Linux (IV)

Tabla de llamadas al sistema

- `sys_call_table`
  - Se ubica en el segmento de datos del kernel
- Descripción de alto nivel para la tabla en x86
  - 64 bits: `arch/x86/syscalls/syscall_64.tbl`
    - Entrada para `gettid()`

      ```
      186 common  gettid  sys_gettid
      ```
  - 32 bits: `arch/x86/syscalls/syscall_32.tbl`
    - Entrada para `gettid()`

      ```
      224 i386    gettid  sys_gettid
      ```

Invocación de llamada al sistema

```c
#include <linux/errno.h>
#include <sys/syscall.h>
#include <linux/unistd.h>
#include <stdio.h>
#define __NR_gettid 186

long mygettid(void) {
    return (long) syscall(__NR_gettid);
}

int main (void) {
        printf("El código de retorno de la llamada gettid es %ld\n",
             mygettid());
        return 0;
}
```

Añadir nueva llamada al sistema (I)

Dar de alta la llamada en la *syscall table*

- Escoger el siguiente número libre (dependiente de arquitectura)
- En x86_64
  - Añadir entrada al final de `arch/x86/syscalls/syscall_64.tbl`

```
...
315     common      sched_getattr      sys_sched_getattr
316     common      lin_hello          sys_lin_hello
```

- En IA-32
  - Añadir entrada al final de `arch/x86/syscalls/syscall_32.tbl`

```
...
352     i386    sched_getattr      sys_sched_getattr
353     i386    lin_hello          sys_lin_hello
```

Añadir nueva llamada al sistema (II)

Implementar la llamada

- Implementación

```
#include <linux/syscalls.h> /* For SYSCALL_DEFINEi() */
#include <linux/kernel.h>

SYSCALL_DEFINE0(lin_hello)
{
    printk(KERN_DEBUG "Hello world\n");
    return 0;
}
```

- ¿Dónde incluir la implementación?

Añadir nueva llamada al sistema (II)

Implementar la llamada

- Implementación

```
#include <linux/syscalls.h> /* For SYSCALL_DEFINEi() */
#include <linux/kernel.h>

SYSCALL_DEFINE0(lin_hello)
{
    printk(KERN_DEBUG "Hello world\n");
    return 0;
}
```

- ¿Dónde incluir la implementación? → 2 opciones
  1. Crear fichero .c nuevo con la implementación (ej: `mysyscall.c`)
     - Necesario modificar `Makefile` del directorio
  2. Incluir la implementación en fichero .c existente
- Por último, compilar, instalar el kernel y reiniciar

Modificación de Makefile

Modificación en `kernel/Makefile` (comp. `kernel/mysyscall.c`)

```
#
# Makefile for the linux kernel.
#

obj-y    = fork.o exec_domain.o panic.o \
       cpu.o exit.o itimer.o time.o softirq.o resource.o \
       sysctl.o sysctl_binary.o capability.o ptrace.o timer.o user.o \
       signal.o sys.o kmod.o workqueue.o pid.o task_work.o \
       extable.o params.o posix-timers.o \
       kthread.o sys_ni.o posix-cpu-timers.o \
       hrtimer.o nsproxy.o \
       notifier.o ksysfs.o cred.o reboot.o \
       async.o range.o groups.o smpboot.o mysyscall.o
…
```

Invocar la nueva llamada

```
#include <linux/errno.h>
#include <sys/syscall.h>
#include <linux/unistd.h>
#include <stdio.h>
#ifdef __i386__
#define __NR_HELLO    353
#else
#define __NR_HELLO    316
#endif

long lin_hello(void) {
    return (long) syscall(__NR_HELLO);
}

int main() {
      return lin_hello();
}
```

Licencia

Arquitectura Interna de Linux - Llamadas al sistema
Versión 0.3

Este documento (o uno muy similar) está disponible en
`https://cv4.ucm.es/moodle/course/view.php?id=70009`