Sistemas Operativos

Llamadas al Sistema

Facultad de Informática
Universidad Nacional de La Plata

S.O.

- ☑ Versión: Marzo 2025
- ☑ Palabras Claves: Sistemas Operativos, Harware, Kernel, llamadas al sistema

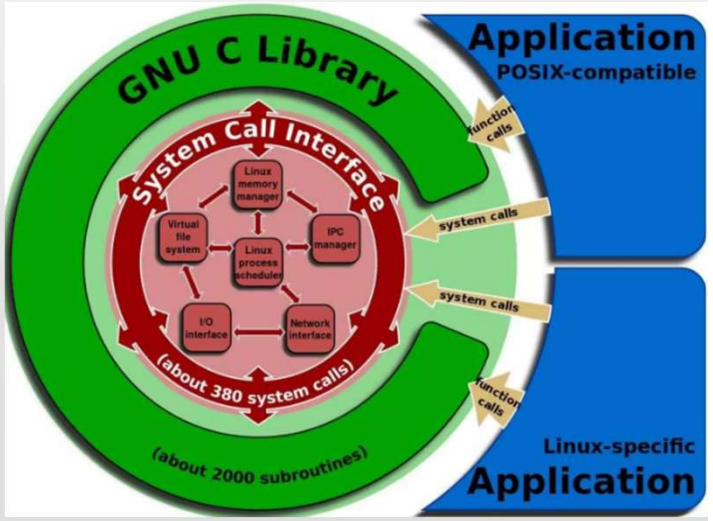
Los temas vistos en estas diapositivas han sido extraídos del libro de William Stallings (Sistemas Operativos: Aspectos internos y principios de diseño) y Conceptos de sistemas operativos (Silberschatz, Galvin, Gagne), https://linux-kernel-labs.github.io/

¿Qué es una llamada al sistema?

- Es el mecanismo utilizado por un proceso de usuario para solicitar un servicio al Sistema Operativo (SO)
- Recordemos:
  - Un proceso de usuario es una porción de código básico que ejecuta instrucciones de código propio en un entorno controlado y limitado de ejecución (sumas, restas, comprobaciones, etc.)
  - Está limitado a su propio espacio de direcciones y no posee acceso directo al hardware. Se ejecuta en Modo Usuario
- El responsable de garantizar que los procesos de usuario puedan ejecutarse controladamente y sin interferirse, es el SO. Se ejecuta en Modo Kernel

¿Qué es una llamada al sistema?

- ¿Cómo logra un proceso de usuario acceder a funciones o servicios que deben ser protegido por el Sistema Operativo?
- La respuesta es sencilla: **System Calls**
- Una llamada al sistema es un método que puede invocar un proceso para solicitar cierto servicio al SO
- El SO provee una interfaz de programación (API) que los procesos pueden utilizar en cualquier momento para solicitar recursos gestionados por él mismo.



¿Qué es una llamada al sistema?

- El SO actúa como un servidor que recibe continuamente requerimientos de sus clientes (Procesos) y determina cuál es el mejor mecanismo de satisfacerlos.
- Para poder llevar adelante la tarea, define un mecanismo y los parámetros que debe enviar el programador para solicitar los servicios. Del mismo modo define los valores de retorno
- En Unix, GNU/Linux la API que define estos mecanismos se llama **libc**

Ejemplo de System Call: read

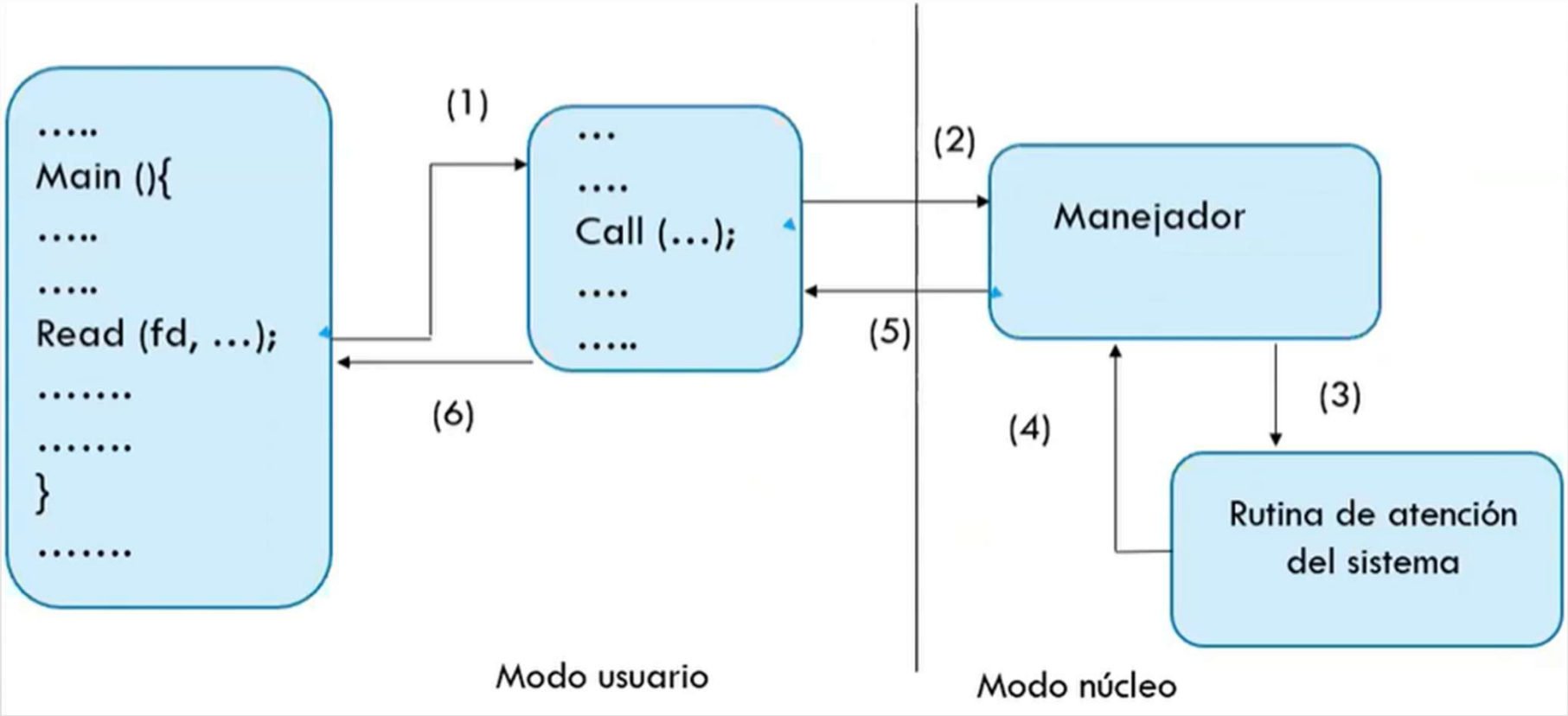
- Analicemos como funciona el mecanismo con un ejemplo:
- Cuando un proceso necesita leer un archivo, requiere acceder al hardware. Este acceso solo puede realizarlo el SO, con lo cual expone un servicio que permite que sea invocado por el proceso:

```
count=read(file, buffer, nbytes);
```

  - file: archivo que se quiere leer
  - buffer: área de memoria en la que será leído
  - nbytes: cantidad de bytes a leer

Ejemplo de System Call: read

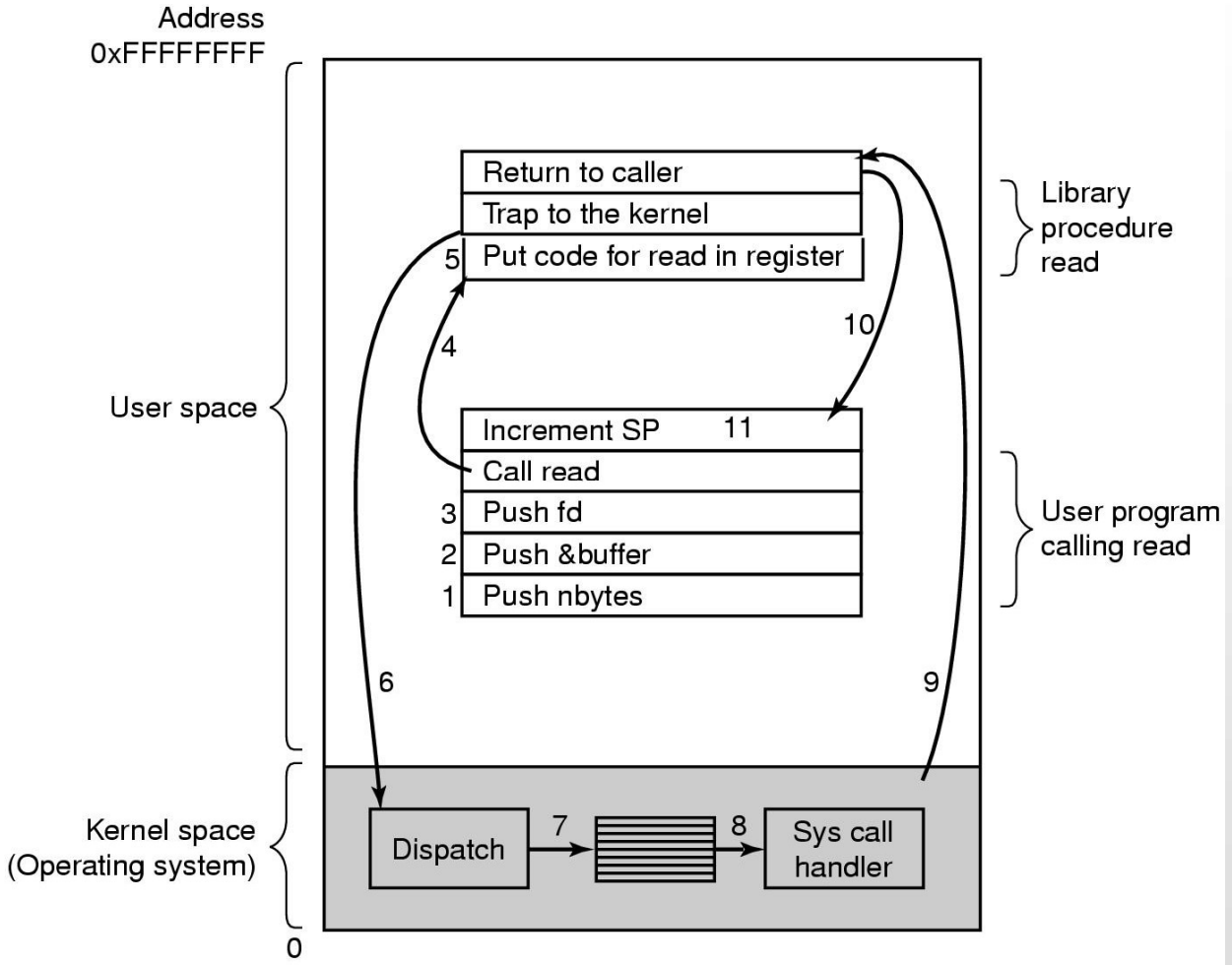
```
count=read(file, buffer, nbytes);
```



Ejemplo de System Call: read

- Las llamadas al sistema se llevan a cabo en una serie de pasos

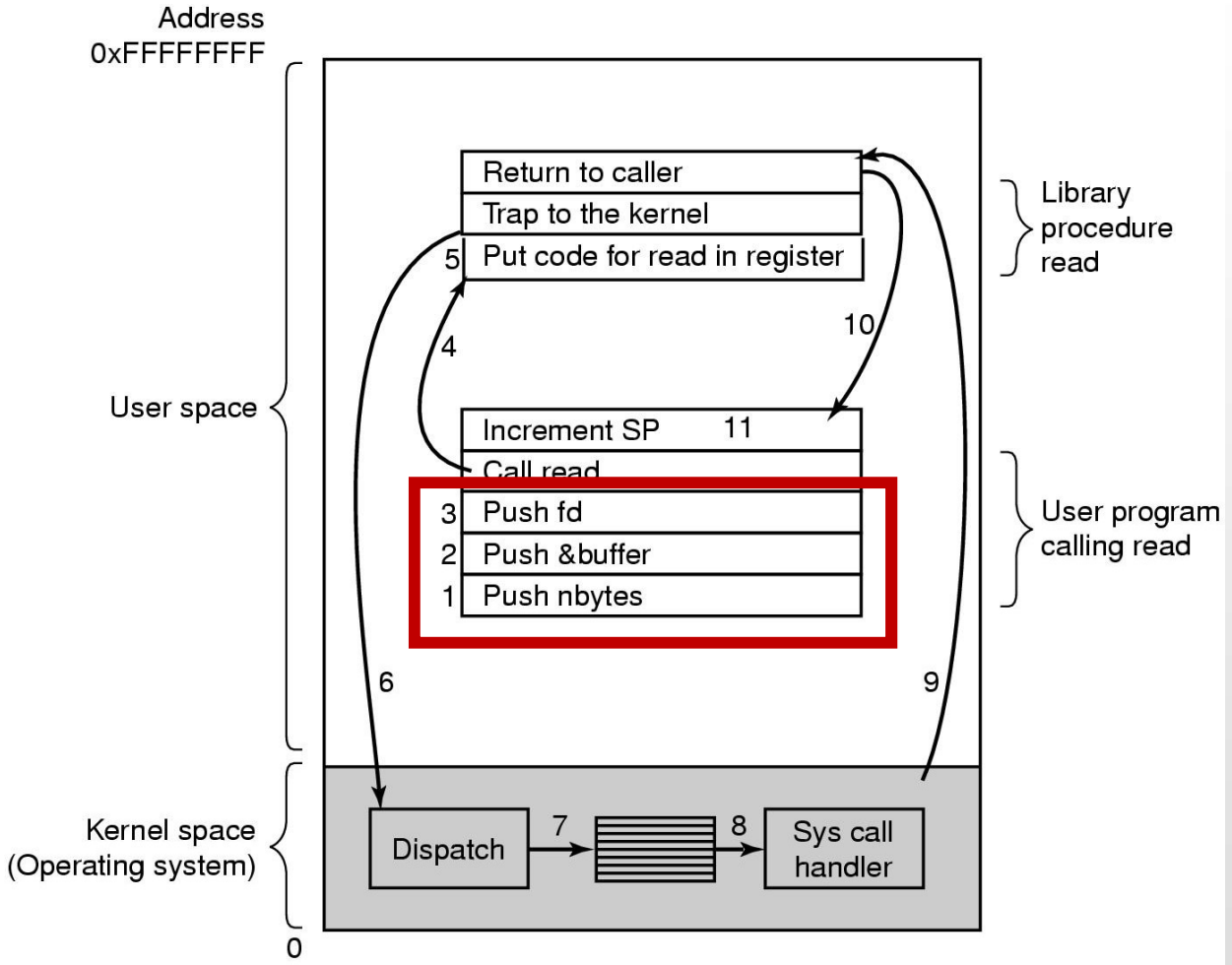
```
count=read(file, buffer, nbytes);
```



Ejemplo de System Call: read

- El primer paso para poder invocar a read (implementado en la librería de usuario), es meter en la pila los parámetros necesarios

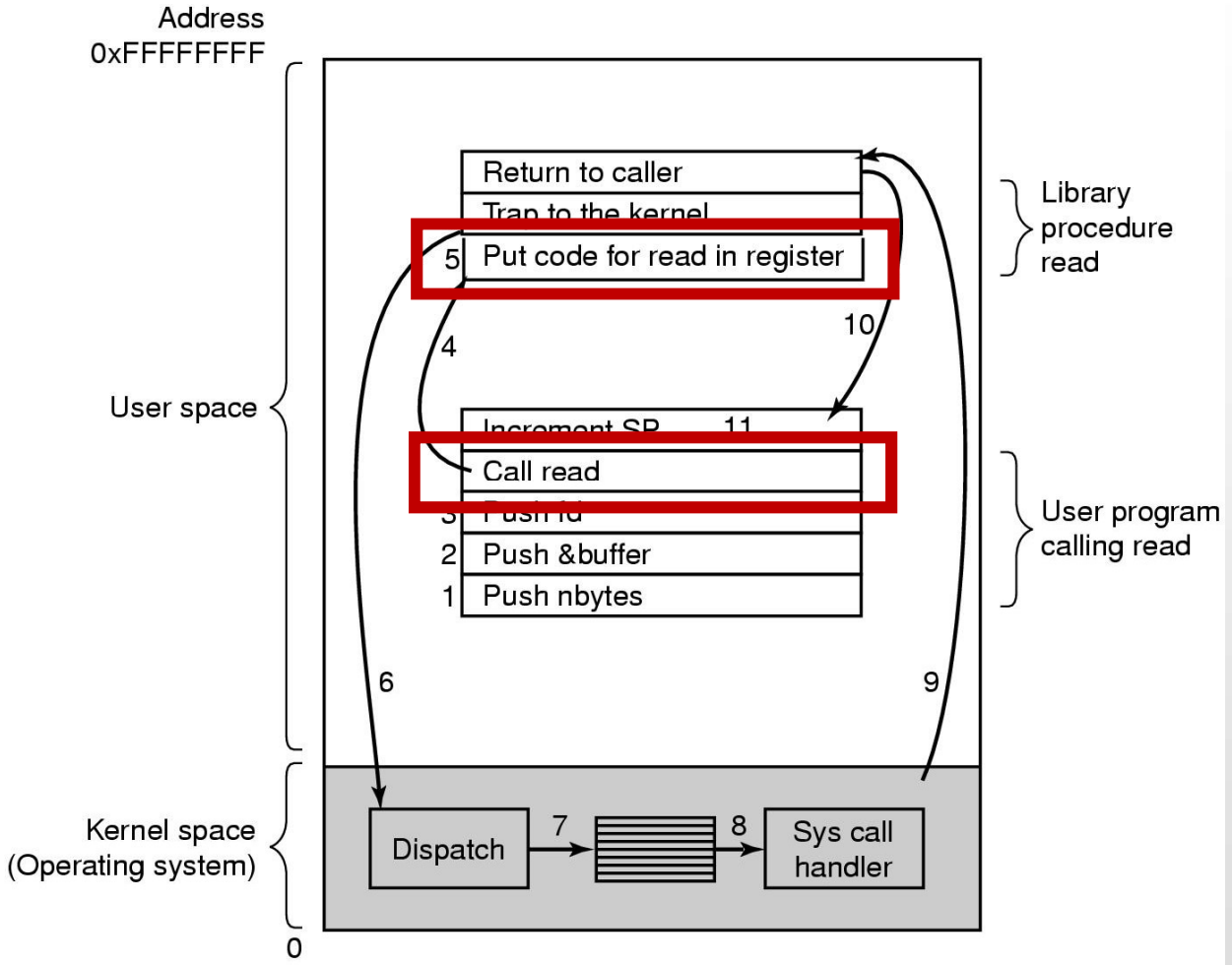
```
count=read(file, buffer, nbytes);
```



Ejemplo de System Call: read

- Se invoca a la función read implementada en la librería y se comienza su ejecución
- Read será una función sencilla que básicamente es la que indicará el número de System Call que se debe ejecutar y permitirá realizar la llamada al sistema correspondiente

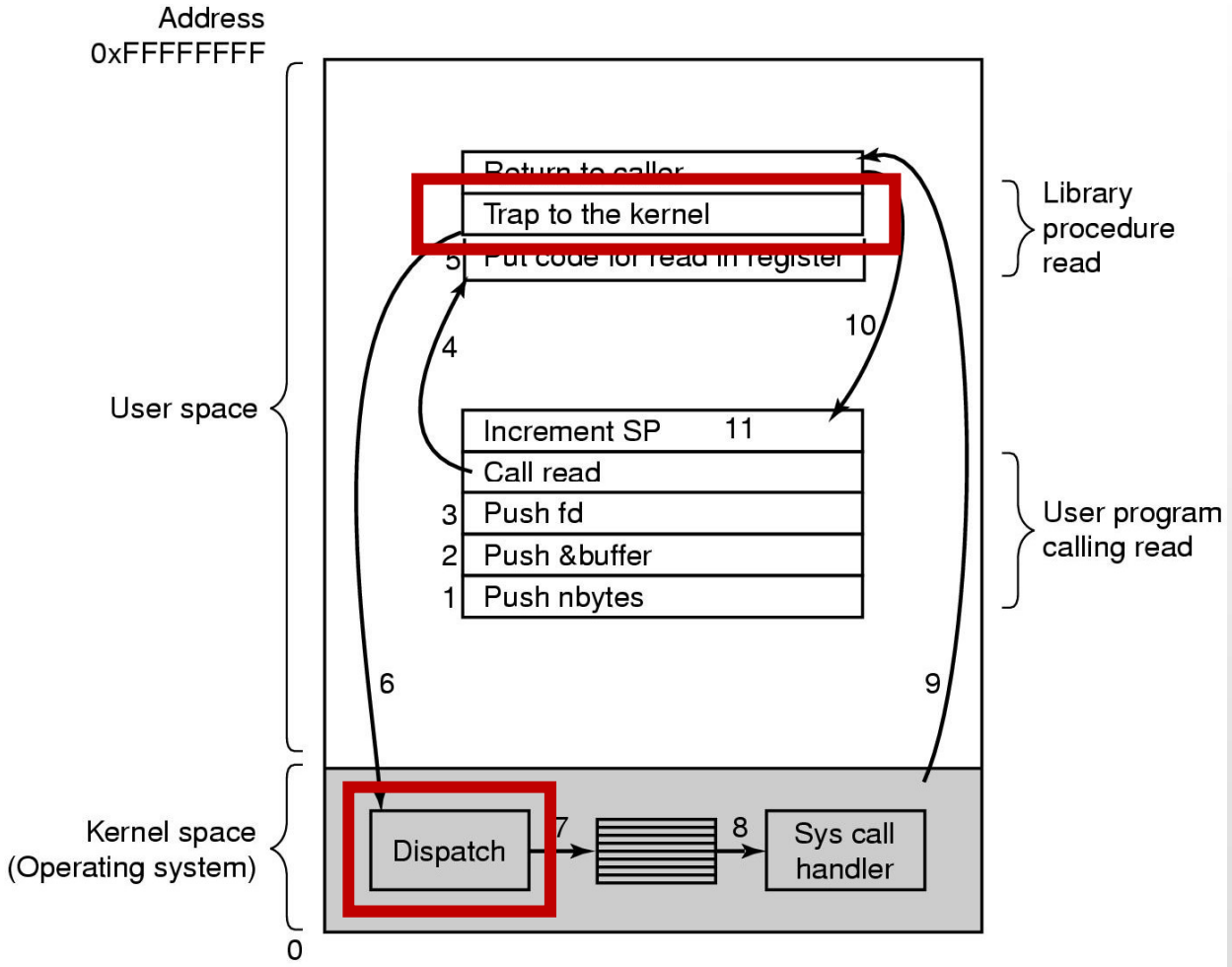
```
count=read(file, buffer, nbytes);
```



Ejemplo de System Call: read

- La función read será la encargada de ejecutar el TRAP (Interrupción por Software) para cambiar de modo Usuario a modo Kernel y pasar el control al SO
- Para todas las System Calls se utiliza la misma interrupción
- La forma de identificar la Syscall invocada es a través del valor del registro

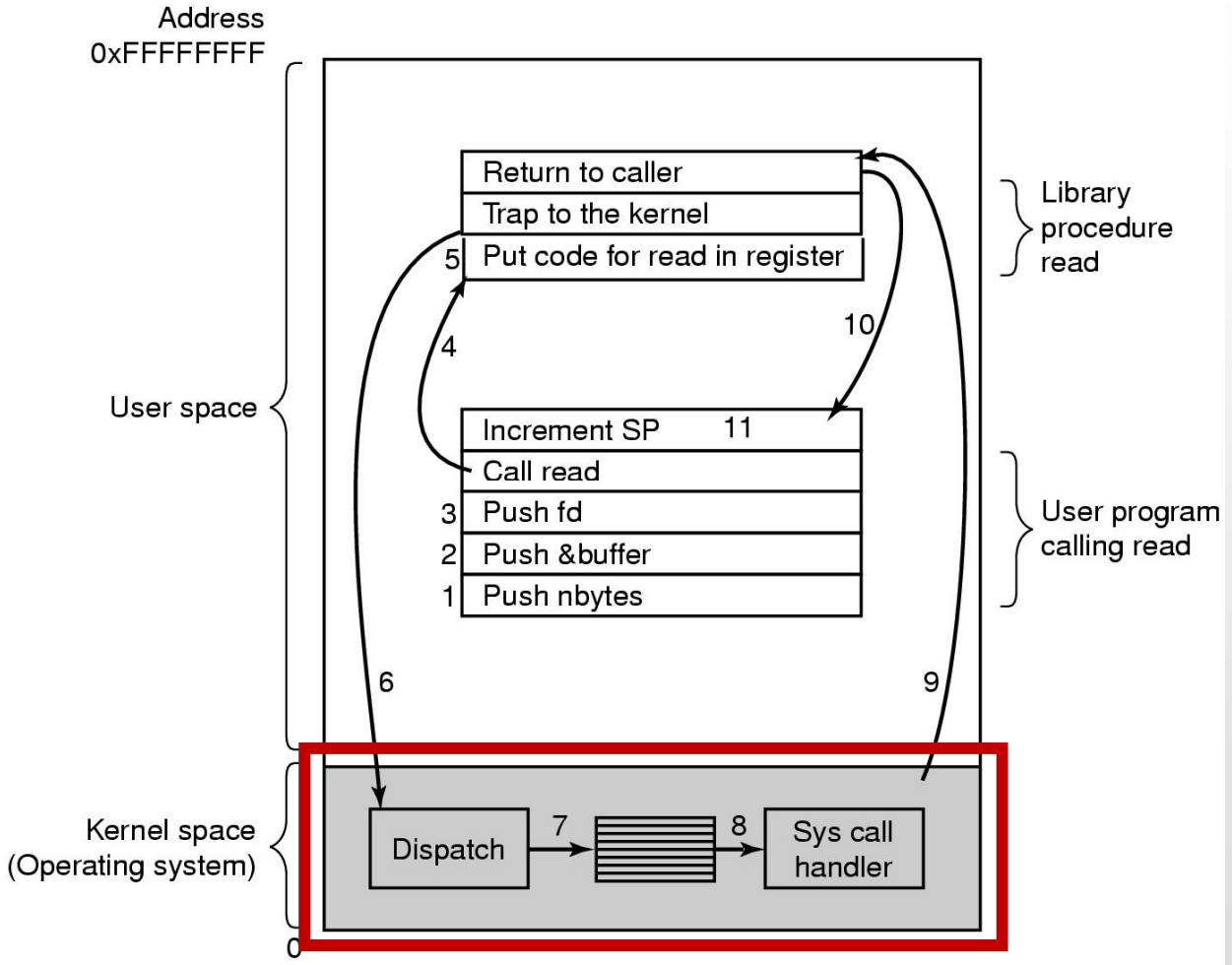
```
count=read(file, buffer, nbytes);
```



Ejemplo de System Call: read

- Una vez que el SO tiene el control, verificará cuál es la llamada al sistema que debe atender y ejecutará el código correspondiente
- En este punto se accede al dispositivo de almacenamiento para obtener el archivo solicitado y leerlo en memoria

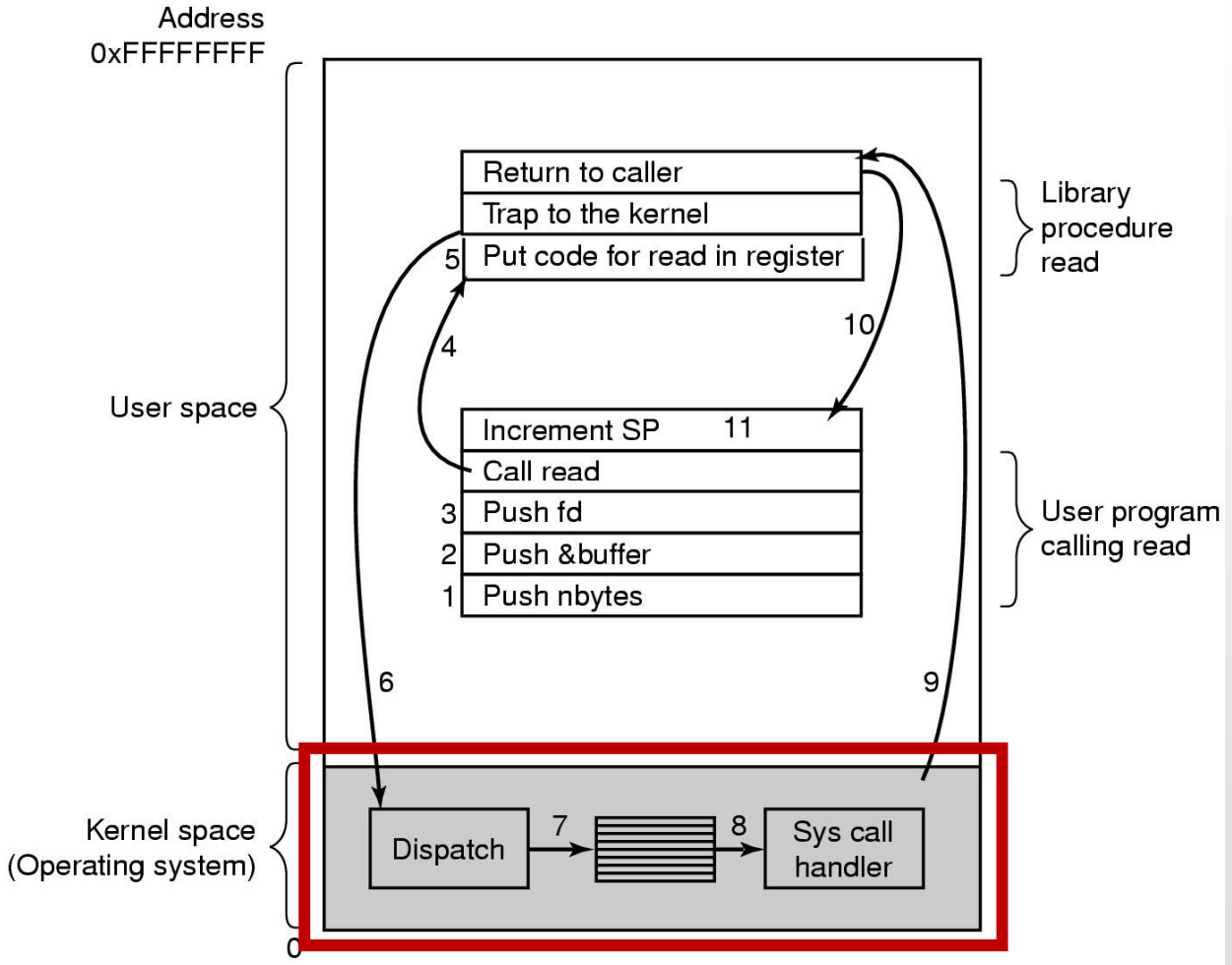
```
count=read(file, buffer, nbytes);
```



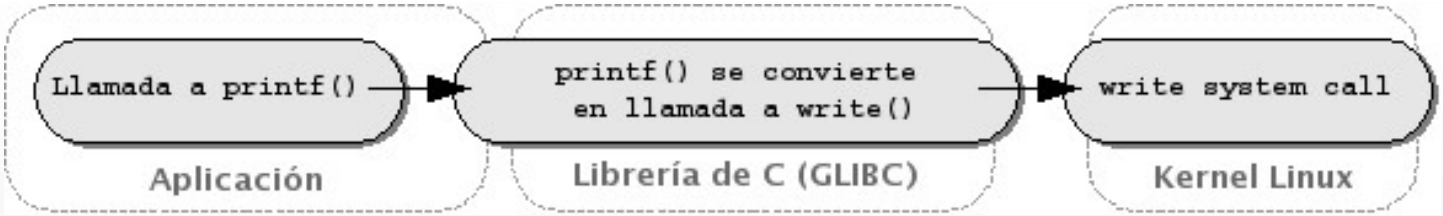
Ejemplo de System Call: read

- Una vez que el SO tiene el control, verificará cuál es la llamada al sistema que debe atender y ejecutará el código correspondiente
- En este punto se accede al dispositivo de almacenamiento para obtener el archivo solicitado y leerlo en memoria

```
count=read(file, buffer, nbytes);
```



Ejemplo de Sustem Call: write



- El código de la librería realiza las siguientes acciones
- Para activar la system call se indica:
  - El número de syscall que se quiere ejecutar
  - Los parámetros de esa syscall
- Luego se emite un aviso al SO (Trap) para pasar a modo Kernel y gestionar la system call
- Se evalúa la system call deseada y se ejecuta

Ejemplo de Sustem Call: write - registros

- Registros x86 en 32 bits
  - EAX lleva el numero de syscall que se desea ejecutar
  - EBX lleva el primer parámetro
  - ECX lleva el segundo parámetro
  - EDX …
  - ESI
  - EDI
- Instrucción que inicia la system call: int 80h (32 bits, syscall en 64 bits)

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to
       file referred to by the
```

```
# 32-bit system call numbers
#
# The format is:
# <number> <abi> <name> <entr
#
# The abi is always "i386" fo
#
0       i386    restart_sysca
1       i386    exit
2       i386    fork
3       i386    read
4       i386    write
5       i386    open
6       i386    close
```

```
SYNOPSIS
       #include <stdlib.h>

       void exit(int status);

DESCRIPTION
       The  exit() function causes normal process termination and the value of
       status & 0377 is returned to the parent (see wait(2)).
```

```
_start:

    ; sys_write(stdout, message, length)

    mov eax, 4          ; sys_write syscall
    mov ebx, 1          ; stdout
    mov ecx, message    ; message address
    mov edx, 14         ; message string length
    int 80h

    ; sys_exit(return_code)

    mov eax, 1          ; sys_exit syscall
    mov ebx, 0          ; return 0 (success)
    int 80h

section .data
    message: db 'Hello, world!',0x0A    ; message and newline
```

System Calls - Categorias

- Categorías de system calls:
  - Control de Procesos
  - Manejo de archivos
  - Manejo de dispositivos
  - Mantenimiento de información del sistema
  - Comunicaciones

System calls - Categorias (cont.)

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

System calls - Categorias (cont.)

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Systems Calls - Ejemplos

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

System Calls – GNU/Linux

- ☑ En GNU/Linux, las System Calls son identificadas por un número
  - ☑ Pueden tener como máximo 6 (seis) parámetros
  - ☑ Para x86 en 32 bits está definida en arch/x86/entry/syscalls/syscall_32.tbl
  - ☑ Para x86 en 32 bits está definida en arch/x86/entry/syscalls/syscall_64.tbl
- ☑ La primer tarea que realiza el dispatcher cuando se produce una interrupción es verificar el número en la tabla correspondiente y ejecutar las funciones asociadas

System Calls – GNU/Linux

- La primer tarea que realiza el dispatcher cuando se produce una interrupción es verificar el número en la tabla correspondiente y ejecutar las funciones asociadas

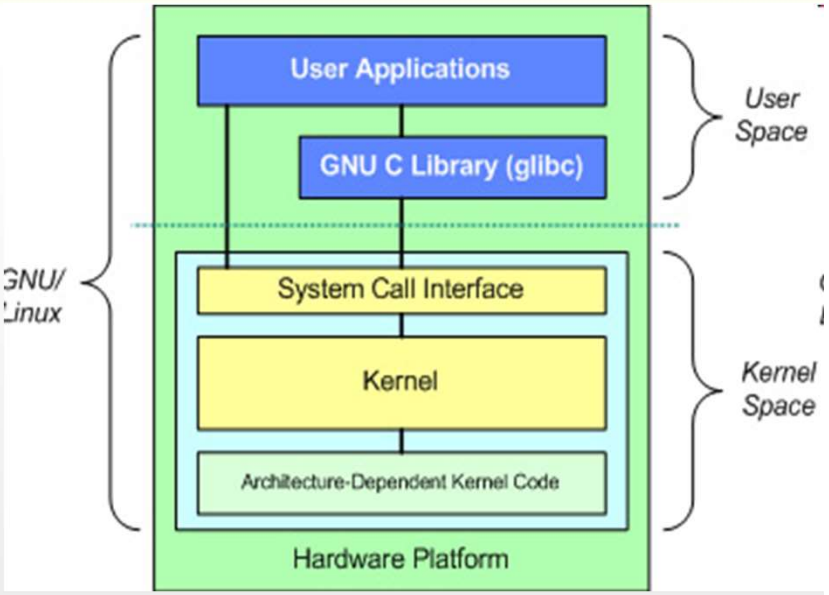
```
# The __x64_sys_*() stubs are created on-the-fly for sys_*() system calls
#
# The abi is "common", "64" or "x32" for this file.
#
0       common  read                    sys_read
1       common  write                   sys_write
2       common  open                    sys_open
3       common  close                   sys_close
4       common  stat                    sys_newstat
5       common  fstat                   sys_newfstat
6       common  lstat                   sys_newlstat
7       common  poll                    sys_poll
8       common  lseek                   sys_lseek
9       common  mmap                    sys_mmap
```

System Calls – GNU/Linux

- Los parámetros de la System Call deben manejarse con cuidado, dado que se configuran en el espacio de usuario:
  - No se puede asumir que sean correctos
  - En el caso de pasarse punteros, no pueden apuntar al espacio del Kernel por cuestiones de seguridad
    - De no verificarse, en un read por ejemplo el buffer podría tener una dirección del Kernel y sobrescribir datos sensibles
  - Los punteros deben ser siempre válidos
    - De no verificarse podría producir Kernel Panic.
  - El Kernel deberá tener acceso al espacio de usuario con APIs especiales que garanticen que se accede al espacio de direcciones de quien invocó la System Call (`get_user()`, `put_user()`, `copy_from_user()`, `copy_to_user()`)

System Calls – GNU/Linux

- En UNIX la API principal para la invocación de la System Call es libc:
  - Es la API principal del SO
  - Provee las librerías estándar de C
  - Es una interfaz entre aplicaciones de usuario y las System Calls (System Call Wrappers).



- La funcionalidad de la libc y las System Calls están definidas por el estándar POSIX
  - Busca proveer una interfaz común para lograr portabilidad
  - Busca que la interacción sea con la API y no con el Kernel