


# Le chiamate di sistema

Il SO quindi fornisce astrazioni e gestisce le risorse del computer, le **chiamate di sistema** sono l'interfaccia che il SO offre alle applicazioni per richiedere servizi.

 Tutti i SO hanno delle chiamate di sistema, queste possono differire per dettagli, ma essenzialmente fanno le stesse cose.

Il meccanismo di chiamate di sistema è strettamente legato al sistema operativo e all'hardware del computer, inoltre per eseguire questi meccanismi c'è bisogno di molta efficienza.

Per risolvere il problema di attaccamento all'hardware viene fornita una libreria di procedure per far sì che sia possibile fare chiamate di sistema da programmi C e spesso anche da altri linguaggi.

 UNIX libc si basa sulla libreria C POSIX.

Guardiamo adesso nel dettaglio i passaggi per effettuare una chiamata di sistema prendendo in dettaglio la chiamata di sistema `read`, innanzitutto scriviamone la sintassi:

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

`fd` = parametro che specifica il file

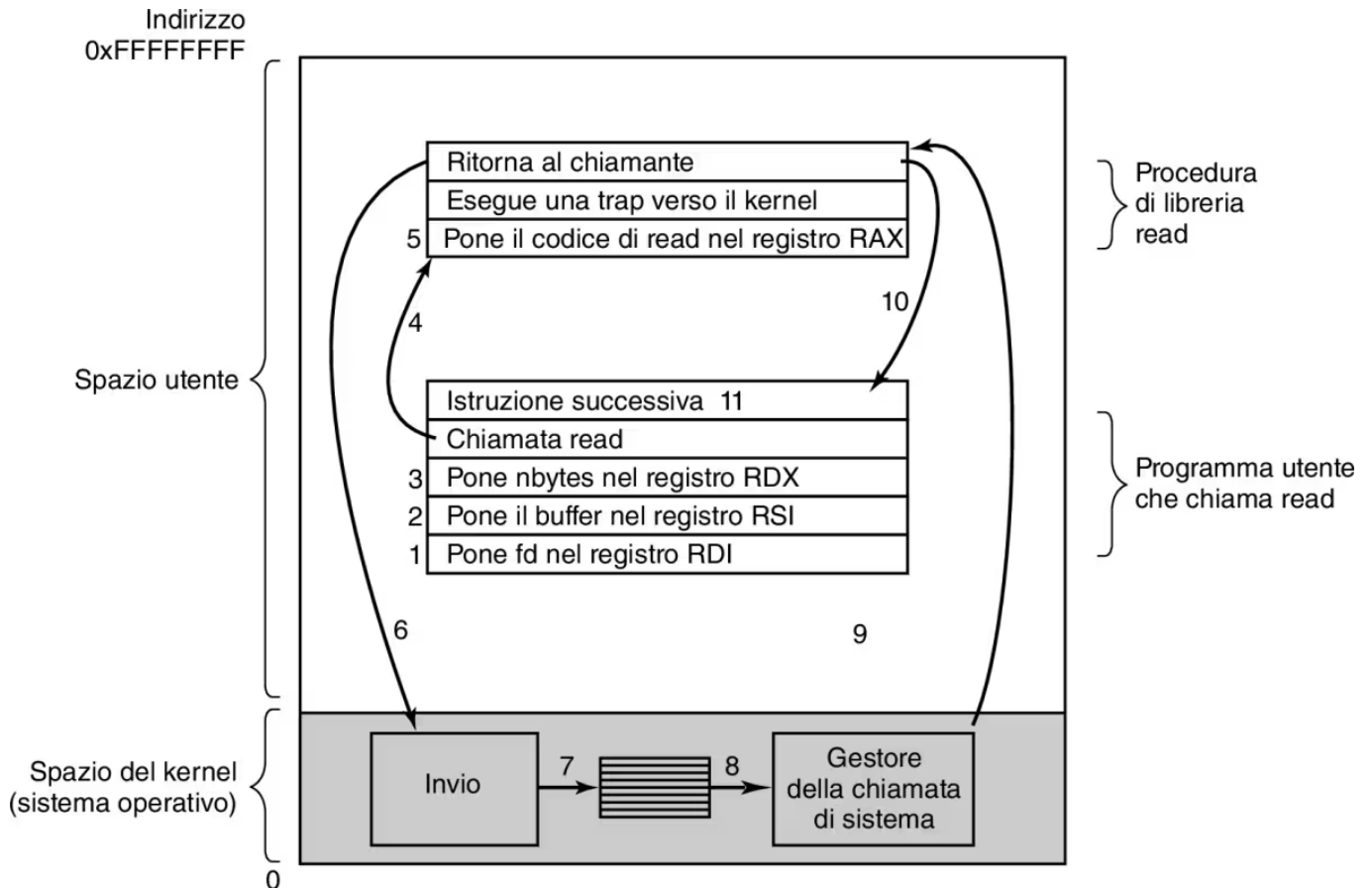
`buffer` = parametro che punta al buffer

`nbytes` = numero di bytes da leggere

1. Il programma chiamante prepara i parametri da passare alla chiamata, solitamente memorizzandoli in un set di registri. (I primi tre passaggi sono inserire i valori nei parametri).
2. Il programma effettua la chiamata di procedura alla libreria.
3. La procedura di libreria inserisce, solitamente nel registro RAX, il numero della chiamata di sistema, questo numero identifica quale funzione del kernel deve essere eseguita.
4. Poi esegue un'istruzione **trap** per passare in modalità kernel e questa istruzione viene passata al Gestore della chiamata di sistema.
5. Viene eseguito il gestore delle chiamate di sistema e poi si ritorna alla procedura della libreria in posizione successiva rispetto all'istruzione `trap`.

6. L'istruzione successiva all'istruzione trap ha puntatore che fa ritornare al programma chiamante dal quale si può eseguire la prossima istruzione.

✎ La chiamata di sistema potrebbe bloccare il chiamante, ad esempio, se l'input desiderato non è accessibile. quindi viene bloccato il processo che verrà ripreso una volta che ci sono le condizioni disponibili.



## Alcune chiamate di sistema

# CHIAMATE DI SISTEMA PER LA GESTIONE DEI PROCESSI

Call	Description
<code>pid fork( )</code>	Creare un processo figlio identico al genitore
<code>pid waitpid( pid, &amp;statloc, options)</code>	Attendere che un processo figlio termini
<code>s = execve(name, argv, environp)</code>	Sostituire l'immagine centrale di un processo
<code>exit(status)</code>	Terminare l'esecuzione del processo e restituire lo stato

- Alcune delle principali chiamate di sistema POSIX.
- Il codice di ritorno `s` è -1 se si è verificato un errore. I codici di ritorno sono i seguenti: `pid` è l'id di un processo

# CHIAMATE DI SISTEMA PER LA GESTIONE DEI PROCESSI

Call	Description
<code>fd = open(file, how, ...)</code>	Aprire un file per la lettura, la scrittura o entrambe le operazioni.
<code>s = close(fd)</code>	Chiudere un file aperto
<code>n = read(fd, buffer, nbytes)</code>	Leggere dati da un file in un buffer
<code>n = write(fd, buffer, nbytes)</code>	Scrivere dati da un buffer in un file
<code>Position = lseek(fd, offset, whence)</code>	Spostare il puntatore del file
<code>s = stat(name, &amp;buf)</code>	Ottenere informazioni sullo stato di un file

- Il codice di ritorno `s` è -1 se si è verificato un errore. I codici di ritorno sono i seguenti: `fd` è un descrittore di file, `n` è un conteggio di byte, `position` è un offset all'interno del file.

# CHIAMATE DI SISTEMA PER LA GESTIONE DEL FILE SYSTEM

Call	Description
<code>s = mkdir(name, mode)</code>	Crea una nuova directory
<code>s = rmdir(name)</code>	Rimuove una directory vuota
<code>s = link(name1 , name2)</code>	Crea una nuova voce, nome2, che punta a nome1
<code>s = unlink(name)</code>	Rimuove una voce della directory
<code>s = mount(special, name, flag)</code>	Monta un file system
<code>s = umount(special)</code>	Smonta un file system

## ALTRE CHIAMATE DI SISTEMA

Call	Description
<code>s = chdir(dirname)</code>	Cambia la directory di lavoro
<code>s = chmod(name, mode)</code>	Modifica i bit di protezione di un file
<code>s = kill(pid, signal)</code>	Invia un segnale a un processo (NON UCCIDE)
<code>s = time(&amp;seconds)</code>	Ottiene il tempo trascorso dal 1° gennaio 1970

# API DI WINDOWS

UNIX	Win32	Descrizione
fork	CreateProcess	Crea un nuovo processo
waitpid	WaitForSingleObject	Può attendere l'uscita di un processo
execve	(none)	Createprocess = fork + execve
exit	ExitProcess	Termina l'esecuzione
open	createFile	Crea un file o apre un file esistente
close	CloseHandle	Chiude un file
read	ReadFile	Legge dati da un file
Write	WriteFile	Scrive dati in un file
lseek	SetFilePointer	Sposta il puntatore del file
stat	GetFileAttributesEx	Ottiene vari attributi del file
mkdir	CreateDirectory	Crea una nuova directory

- Le chiamate API Win32 che corrispondono grosso modo alle chiamate UNIX

## API DI WINDOWS (2)

- Le chiamate API Win32 che corrispondono grosso modo alle chiamate UNIX

UNIX	Win32	Description
Iseek	SetFilePointer	Move the tile pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdjr	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, so no umount
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	GetLocamme	Get the current time