1.6 Chiamate di Sistema

Abbiamo già visto che i sistemi operativi hanno due funzioni principali: **Fornire astrazioni ai programmi utente** e **gestire le risorse del computer**

Etrambe queste funzioni vengono eseguite automaticamente dal S.O senza intervento dell'Utente tramite le **Chiamate di Sistema (System Calls)**.

> ***DEF***: Da come indica il nome, le **chiamate di sistema** sono delle *chiamate effetuate dall'Utente/Programmatore/Processo* che *indicano il meccanismo usato da esso per richiedere un servizio/dato* (E.g: La cancellazione/lettura di un certo file) e vengono eseguite direttamente dal Kernel del S.O

Esse vengono implementate in programmi/compilatori scritti in linguaggio **Assembly (di solito nei compilatori) e nel C.**

Esempio di un codice in C che usa una chiamata di sistema UNIX

```
int fd = open("foo.txt", O_RDONLY | O_CREAT);
```

Il codice scritto sopra indica che viene creata una variabile di tipo `int` chiamata `fd` e gli viene assegnata la **syscall** (*abbreviazione di System Call*) `open()` che **apre un file** chiamato `foo.txt` **per leggerne il contenuto** senza scrivere al suo interno (notare bene il *`O_RDONLY` che definisce che il file deve solo letto*).

Se esso **non viene trovato, viene creato** (notare bene il *`O_CREAT` che definisce che deve essere creato se esso non esiste nella stessa directory dove viene eseguito il codice C*)

Se la chiamata di sistema **non può essere effettuata** (*sia per un parametro errato o errore su disco o altre cause*), **viene settato numero d'errore** che viene inserito in una variabile globale chiamata `errno`.

Questo segnalatore di errori però **da solo non fa nulla**, sarà poi compito del programmatore di *implentarlo* in una funzione di ***handling*** (`Exception Handler`).

Tipi di chiamate di Sistema

Le System Calls possono essere classificate in diverse categorie, al riguardo sono stati definiti in particolare i seguenti tipi di classificazione:

1. **Gestione dei file**:

   I programmi applicativi richiedono chiamate di sistema di questo tipo per **ottenere l'accesso alle tipiche operazioni sui file** come leggere, creare o modificare un file.

2. **Gestione e Comunicazione dei processi**:

   Tutti i processi di un S.O devono essere **controllati e devono comunicare tra di loro**, affinché possano essere interrotti in qualsiasi momento o essere pilotati da altri processi.

A tal fine le System Calls di questa categoria controllano ad esempio l'avvio o l'esecuzione oppure lo stop o l'interruzione dei processi e la creazione dei loro figli.

1.6.1.1 Chiamate di Sistema di gestione dei file

Funzionamento Tecnico durante una chiamata di sistema

```
Preparazione
```

Si prenda sempre come esempio il seguente codice in C:

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

Come preparazione, il programma di chiamata prima **mette i parametri** (*`fd, buffer, nbytes`*) **nello stack** dei registri della CPU.

Dopodichè, il compilatore C **mette i parametri nello stack in ordine inverso**.

Il secondo parametro **verrà passato come riferimento**, *cioè viene passato l'indirizzo di memoria del buffer, ma non il suo contenuto*.

```
Chiamata alla procedura
```

Viene **effettuata la chiamata alla procedura di libreria** (il `read`) e ne viene **messo il suo codice** (di solito scritto in Assembly) **in un registro**.

```
Chiamata TRAP
```

Viene eseguita un istruzione `TRAP` **che passa dalla usermode** (modalità utente, dove si ha più restrizioni per garantire più sicurezza) **alla kernel mode** (modalità kernel, dove vengono bypassate tutte le restrizioni) e effettua un **esecuzione a un indirizzo fisso** all'interno del kernel.

```
Esecuzione nel Kernel
```

Il codice del Kernel che parte in seguito all'istruzione `TRAP` **esanima il numero della chiamata di sistema** (come una sorta di ID) e poi **indirizza al corretto gestore** di chiamate di sistema tramite una tabella di puntatori ai gestori.

```
Esecuzione gestore delle chiamate
```

A questo punto **viene eseguito il gestore** della chiamate di sistema.

```
Ritorno in User Mode
```

Una volta che il gestore delle chiamate di sistema ha finito, **viene effettuata un altra TRAP, che effettua il ritorno in User Mode.**

```
Ritorno al chiamante
```

Dopo essere tornata in User Mode, la procedura **ritorna al programma chiamante** (nel nostro caso, il codice in C).

```
Conclusione e Pulizia
```

Per concludere il lavoro, il programma (sempre il nostro codice in C) **deve pulire lo stack** che ha precedentemente usato.

1.6.1.2 Chiamate di sistema della gestione dei processi

Tabella con tutti i processi più usati con descrizione e esempio in codice C

Tipo di Sys Call	**Funzione**	**In Linux**	**in Windows**	**Esempio con codice**
Controllo Processo	Crea un processo	fork()	CreateProcess()	process1 = fork()
Controllo Processo	Termina un processo	exit()	ExitProcess()	exit(statusCode)
Controllo Processo	Attende che un processo termini	waitpid()	WaitForSingleObject()	statProcess1 = waitpid(pid, &statloc, options)
Gestione File	Crea/Apre un file	open()	CreateFile()	file1 = open(file, O_CREAT, O_RDWR)

Tipo di Sys Call	Funzione	In Linux	in Windows	Esempio con codice
Gestione File	Legge un file	read()	ReadFile()	fileData = read(file, buffer, nbytes)
Gestione File	Scrive in un file	write()	WriteFile()	writeData = write(file, buffer, nbytes)
Gestione File	Chiude un file	close()	CloseHandle()	file1 = close(file)
Gestione File	Ottiene informazione sullo stato di un file	stat()	GetFileInformationByHandle()	fileStatus = stat(name, &buffer)
Gestione Directory	Crea una nuova Directory	mkdir()	CreateDirectory()	newDir = mkdir(name, mode)
Gestione Directory	Rimuove una Directory vuota	rmdir()	RemoveDirectory()	remDir = rmdir(name)
Gestione Filesystem	Crea un link simbolico che punta al primo parametro	link()	CreateSymbolicLinkA()	var = link(oldLink, newLink)
Gestione Filesystem	Rimuove un link simbolico	unlink()		var = unlink(newLink)
Gestione Filesystem	Monta un dispostivo	mount()	SetVolumeMountPointA()	var = mount(devPath, name, flag)
Gestione Filesystem	Smonta un dispostivo	unmount()	DeleteVolumeMountPointW()	var = unmount(devPath)
Gestione Directory	Cambia la directory corrente	chdir()	SetCurrentDirectory()	var = chdir(dirPath)
Gestione Directory	Cambia i bit di protezione di un file	chmod()	SetFileSecurity()	var = chmod(filePath, flags)
Gestione Processi	Manda un segnale di arresto al processo	kill()	killProcessByPID()	var = kill(pid, signal)
Gestione Processi	Sospende un processo	sleep()	Sleep()	var = sleep(pid)

Tipo di Sys Call	Funzione	In Linux	in Windows	Esempio con codice
Gestione Processi	Crea un pipe tra due file	pipe()	CreatePipe()	var = pipe(pid1, pid2)

Piccole Note riguardo alcune syscalls

Fork()

> Il processo figlio creato dal processo padre useguendo `fork()` avrà un **PID diverso da quello del padre**. Inoltre i **dati** del processo padre **verranno copiati nel figlio**, ma esso potrebbe prendere un via separata rispetto al padre e i cambiamenti dei valore presente nel figli **NON** verrano copiati nel processo padre.

Paremetri `O_RDONLY` , `O_WRONLY`, `O_RDWR` e `O_CREAT`

Questi parametri possono essere definiti durante le SysCalls riguardanti la Gestione dei File.

```
O_RDONLY
```

> Il parametro `O_RDONLY` indica che il file sarà **solo letto** (Read-Only)

```
O_WRONLY
```

> Il parametro `O_RWONLY` indica che il file sarà **solo scritto** e non letto (Write-Only)

```
O_RDWR
```

> Il parametro `O_RDRW` indica che il file verrà **sia letto che scritto** (Read-Write)

```
O_CREAT
```

> Il parametro `O_CREAT` indica che se il file non viene trovato sul disco, esso verrà **creato**