SAPIENZA
UNIVERSITÀ DI ROMA

Corso di laurea in Ingegneria dell'Informazione
Indirizzo Informatica

Reti e sistemi operativi

Introduzione agli interrupt

Le interruzioni (interrupt)

- I sistemi operativi attuali si basano sugli interrupt → ogni intervento del S.O. **è guidato dal verificarsi di un interrupt**
- Quali sono le “cause” che scatenano un interrupt?
  - Un evento notificato da un dispositivo di **I/O** (es. dati in arrivo su un'interfaccia Ethernet) [← hardware]
  - Un'**eccezione** generata da un processo (es. accesso a memoria di un altro processo)[← software]
  - Un'esplicita richiesta di un processo attraverso una **syscall** (es. per accedere ad un file) [← software]
  - **Timer** interno [← hardware]

Interrupt per i dispositivi di I/O (1/5)

- I dispositivi di I/O (o periferiche) consentono di trasferire informazioni dalla memoria interna dell'elaboratore (memoria centrale, registri, ecc...) verso il mondo esterno (dispositivi di **output**) o viceversa (dispositivi di **input**).
- Ad ogni dispositivo è **univocamente associato uno o più indirizzi (stato, buffer interni, ecc)**: la CPU seleziona la periferica con cui vuole eseguire l'operazione di input o output inserendo tale indirizzo nel bus indirizzi.
- Nelle architetture **I/O memory mapped**, un sottoinsieme degli indirizzi di memoria è associato ai registri posti nei moduli di interfaccia dei dispositivi di I/O anziché a locazioni della memoria centrale.

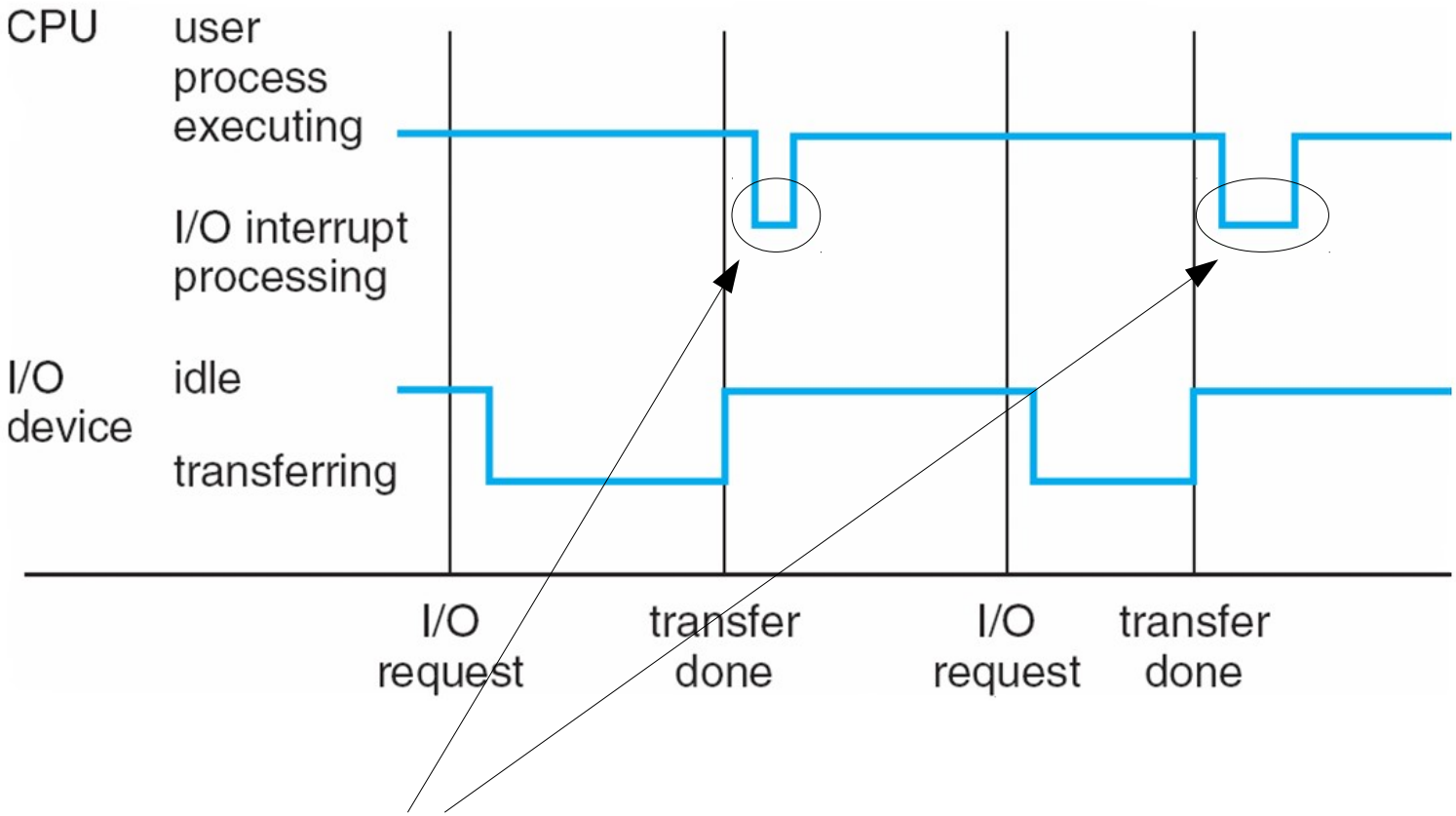
Interrupt per i dispositivi di I/O (2/5)

- Si abbia ad esempio un dispositivo di input (es. una tastiera) e si voglia acquisire dei dati. Un'idea di base è quella di verificare ciclicamente lo stato della periferica per verificare se è disponibile un dato → **polling**.
- **Problema** → numero elevatissimo di cicli di attesa, la maggior parte dei quali inutili. Il processore potrebbe essere impiegato per eseguire istruzioni più utili.
- E' necessario disporre di un meccanismo che:
  - consenta al processore di passare ad eseguire altre attività, dopo aver avviato uno dei passi del protocollo di I/O;
  - preveda che **sia il dispositivo a farsi parte attiva per richiedere l'intervento del processore**, inviando ad esso un **apposito segnale (interrupt)** quando è pronto per il passo successivo del protocollo di I/O (richiesta di servizio)

Interrupt per i dispositivi di I/O (3/5)

- A fronte di questo segnale il processore potrà:
  - interrompere il programma in esecuzione,
  - passare ad eseguire una funzione di sistema operativo che implementa il protocollo di I/O di quello specifico device (questa sequenza di istruzioni si chiama **routine di servizio della interruzione** – **ISR**),
- Al termine dell'ISR, il sistema operativo pianificherà (schedule) l'esecuzione del programma interrotto

Interrupt per i dispositivi di I/O (4/5)



Esecuzione delle Interrupt Service Routines

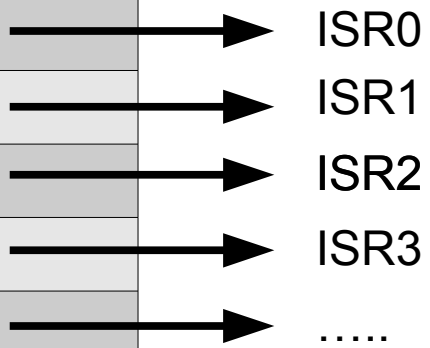
Interrupt per i dispositivi di I/O (5/5)

- Per gestire correttamente questo meccanismo di interruzione, è necessario che S.O. e CPU **in collaborazione** provvedano per:
  - Il **salvataggio del contesto (context switch)** in cui sta operando il processo che viene interrotto, per poterlo poi ripristinare al termine dell'interruzione (es. **contenuto registri, PC, ecc...**);
  - Il riconoscimento del dispositivo che ha provocato l'interruzione (**identificazione dell'interruzione**);
  - Di solito quando viene servito un interrupt la possibilità di generare altri interrupt viene disabilitata, con alcune eccezioni → L'ordinamento delle richieste di interruzione secondo una **gerarchia di priorità**.

Interrupt Vector

- L'interrupt vector contiene, per ogni tipo di interruzione, device, ecc... l'indirizzo dell'interrupt service routine corrispondente, definita dal sistema operativo

Interrupt ID	Puntatore a ISR
0 (es. reset)	ADR 0
1 (es. interfaccia seriale)	ADR 1
2 (es. TRAP)	ADR 2
3 (...)	ADR 3
……..	……..



(Modello semplificato)

Eccezioni

- Sono errori run-time generati da una sequenza di istruzioni “semanticamente” errata:
  - Divisione per zero
  - Accesso a memoria riservata
  - …
- L'hardware ha dei circuiti di rilevamento di tali errori, e solleva immediatamente un interrupt chiamato **TRAP**. Alcuni sistemi hanno più interrupt di tipo TRAP, dipendenti dall'eccezione sollevata, altri uno solo. Sarà compito del S.O. rilevare la causa precisa dell'interrupt.
- Ovviamente nell'interrupt vector ci sarà una (o più entry) corrispondenti a **TRAP**.

System call

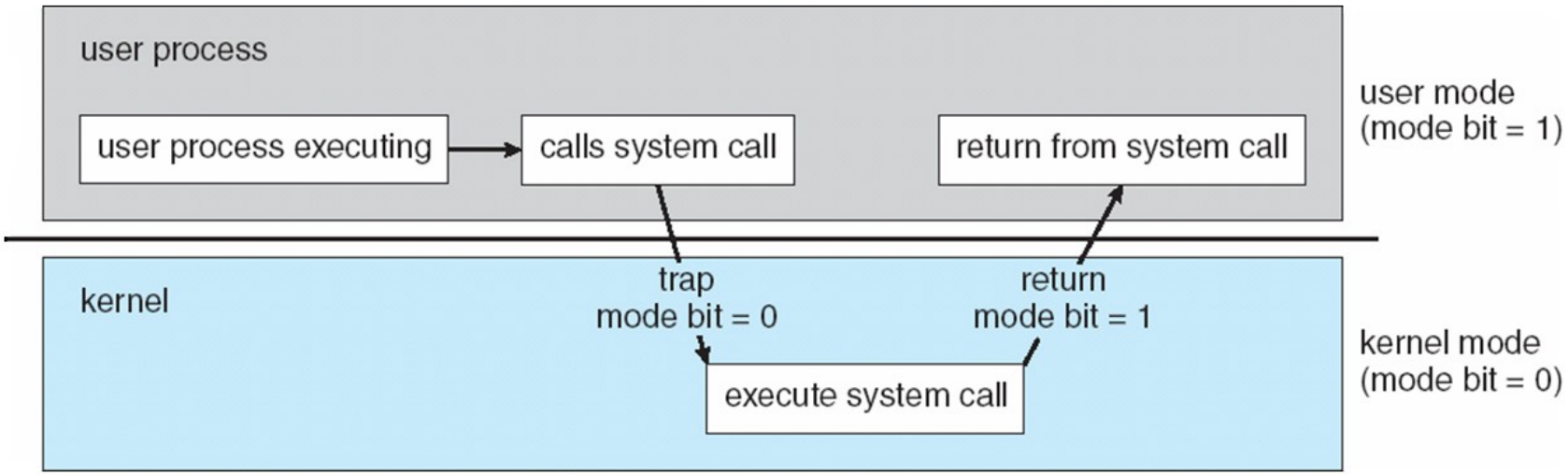
- Se un processo utente (ovvero, un programma in esecuzione che NON sia il S.O.) desidera effettuare un'operazione di I/O o una generica operazione che implica l'accesso diretto all'hardware, dovrà chiedere l'intervento del S.O. chiamando una delle funzioni da esso messe a disposizione → **syscall** (di solito scritte in C/C++ ma anche in assembly).
- La procedura è quella di sollevare esplicitamente un interrupt, ad esempio di tipo TRAP (come nel caso delle eccezioni) o di tipo specifico (es. **SWI**)
- Sarà compito del S.O. esaminare l'istruzione che ha generato l'interrupt ed agire di conseguenza

Dual mode (1/2)

- **Problema**: di base il S.O. alle componenti hardware di “basso livello” attraverso una serie di funzioni assembly specifiche dell'architettura. Chi mi assicura che i processi utenti non usino esplicitamente tali funzioni, bypassando così il controllo del S.O. con possibili conseguenze catastrofiche?
- **Soluzione** → introduco (almeno) 2 modalità gestite in hardware (mode bit, nella CPU), esempio **kernel mode** e **user mode**. Alcuni operazioni (funzioni assembly) sono eseguibili solo in kernel mode.
- All'avvio il mode bit viene settato in modalità kernel mode e viene fatto partire il S.O. (il kernel, appunto!).

Dual mode (2/2)

- Ad ogni interrupt sollevato, la CPU setta il mode bit in modalità kernel mode e chiama l'ISR associata (che appunto fa parte del S.O.).
- Sarà compito del S.O. resettare il mode bit in user mode prima di riprendere l'esecuzione dei processi utente.



Timer

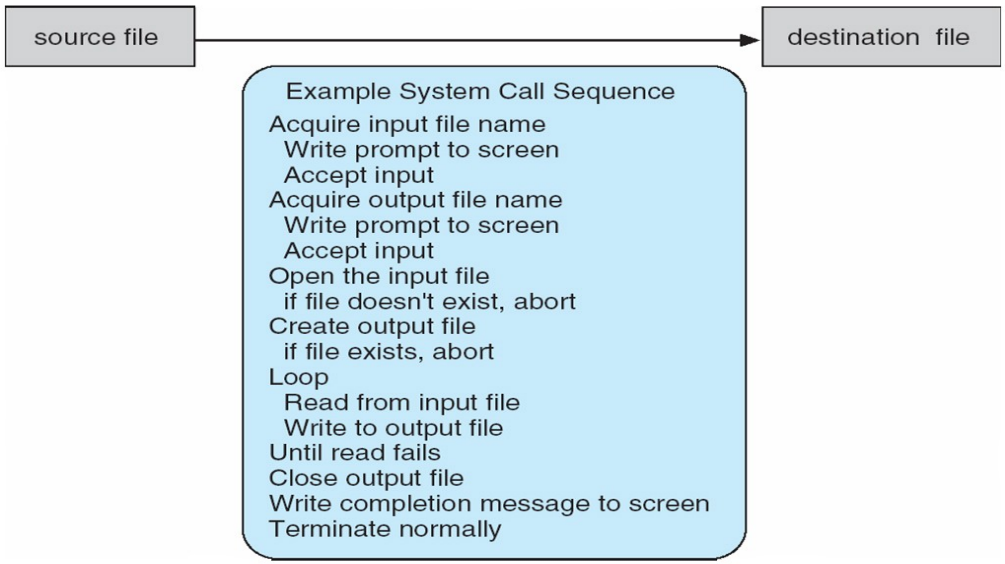
- **Problema**: se non arrivano interrupt da dispositivi di I/O ed un processo non chiama una syscall per un tempo indefinito il S.O. **non** interviene!
- **Soluzione**: ciclicamente vengono generate delle interruzioni da un **timer interno**, generalmente implementato attraverso un clock indipente ed un contatore incrementato ad ogni ciclo di clock.
- Il contatore viene settato dal sistema operativo

Invocazione delle syscall

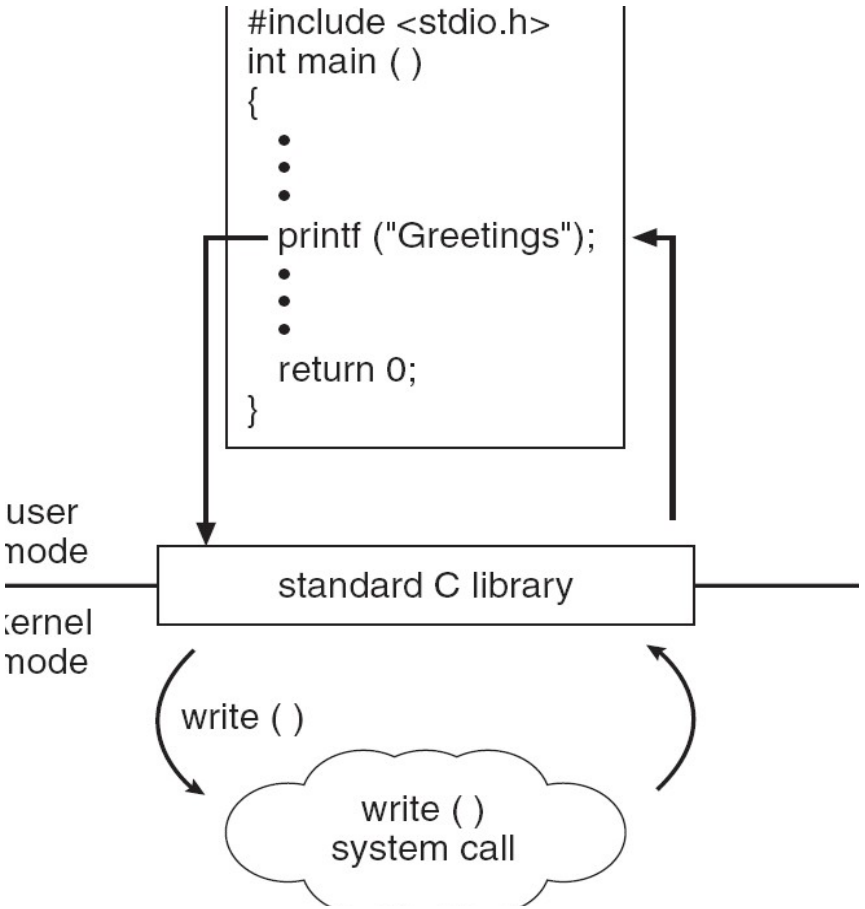
- **Syscall**: Generalmente operazioni semplici e di (relativo) basso livello → solitamente vengono implicitamente chiamate attraverso funzioni di librerie di livello più alto
  - Win32 API for Windows
  - **POSIX API**
  - **librerie standard del C**.
- Motivazioni per usare librerie di alto livello:
  - Portabilità
  - Semplicità

Chiamata implicita di syscall

- **Un esempio**: Leggo un file di testo e lo scrivo su di un altro: tale operazione potrebbe essere portata a termina usando solamente funzioni di libreria standard C quali **fopen()**, **fclose()**, **fread()**, **fwrite()** (o simili).
- In realtà vengono chiamate moltissime syscall → ovvero il **S.O. viene di fatto chiamato moltissime volte**! → **Overhead non trascurabile**! In realtà vengono utilizzate delle efficienti politiche (es. buffering) per evitare ciò.

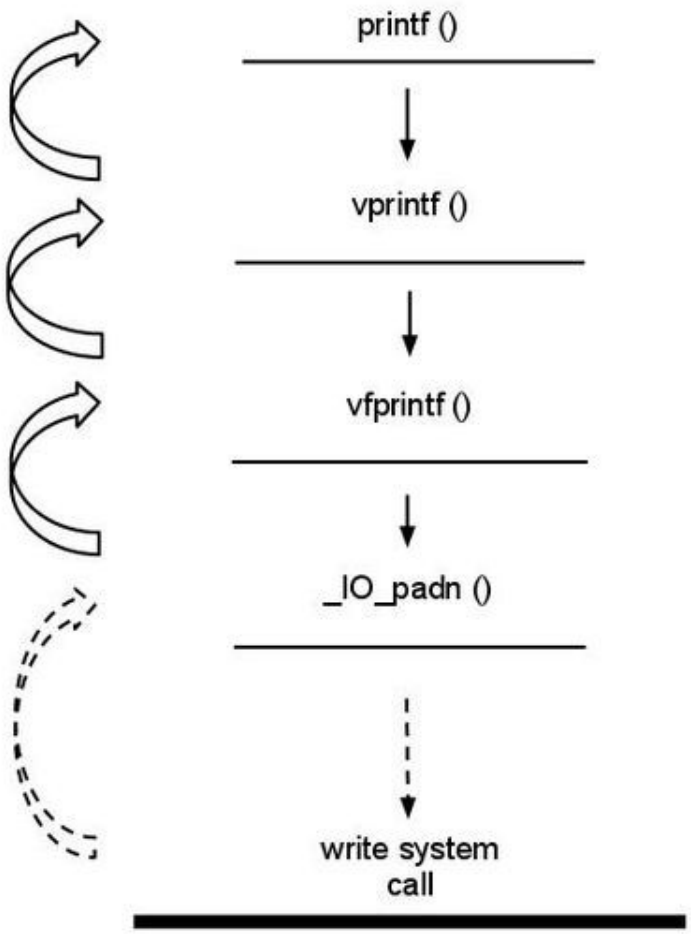


Un esempio: printf()



(Modello semplificato)

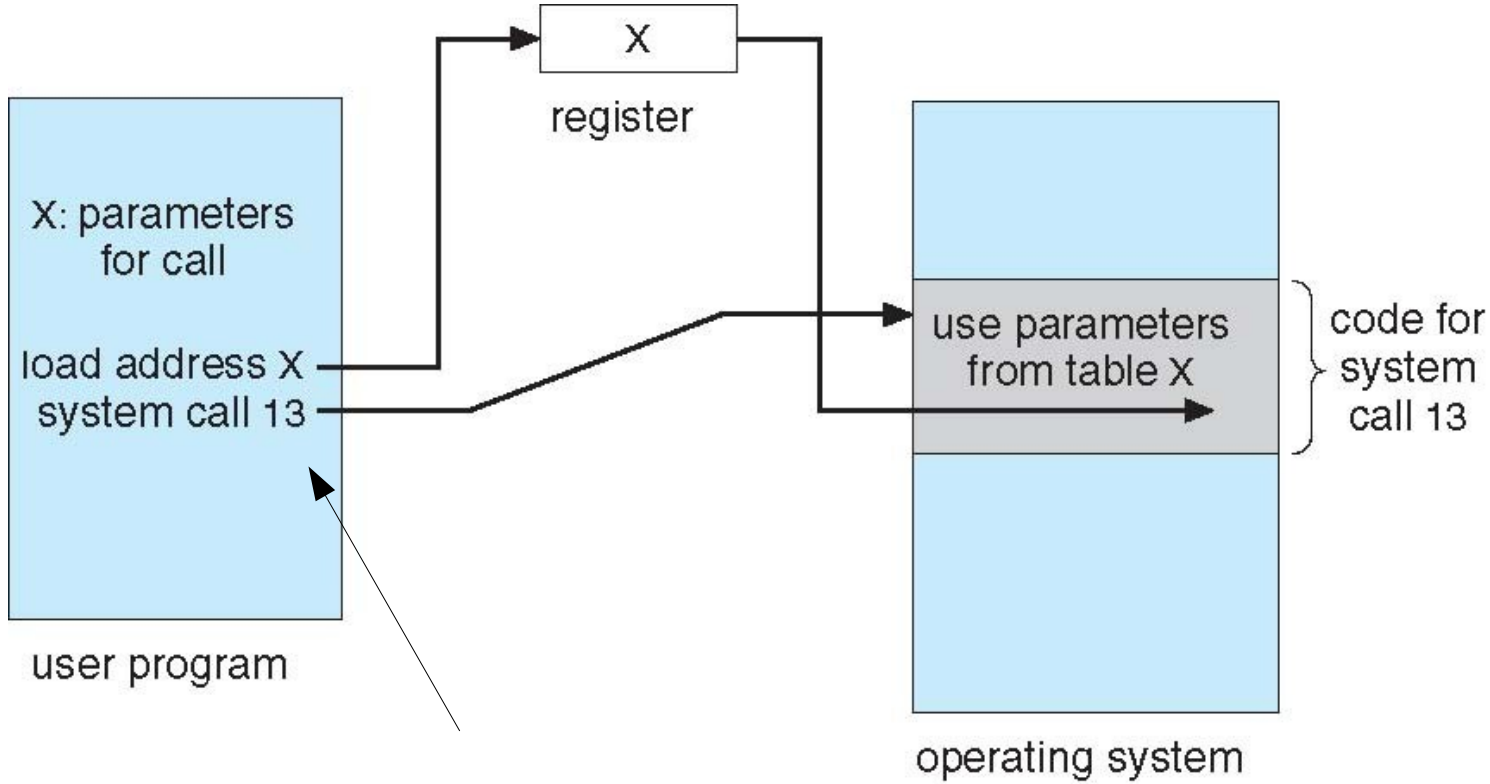
printf() in un S.O. reale



Syscall: passaggio di parametri (1/2)

- Attraverso un set di registri → limita la dimensione dei parametri
- Attraverso un puntatore ad blocco di dati (es. un array o una struttura in C) il cui valore (**indirizzo**) è passato attraverso un registro
- Attraverso lo stack:
  - processo chiamante → push() dei parametri
  - S.O. → pop() dei parametri

Syscall: passaggio di parametri (2/2)



Di solito le syscall sono identificate da un numero, che corrisponde ad una entry di una tabella ove è memorizzato l'indirizzo della porzione di codice corrispondente

Tipi di syscall (1/2)

- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes

Tipi di syscall (2/2)

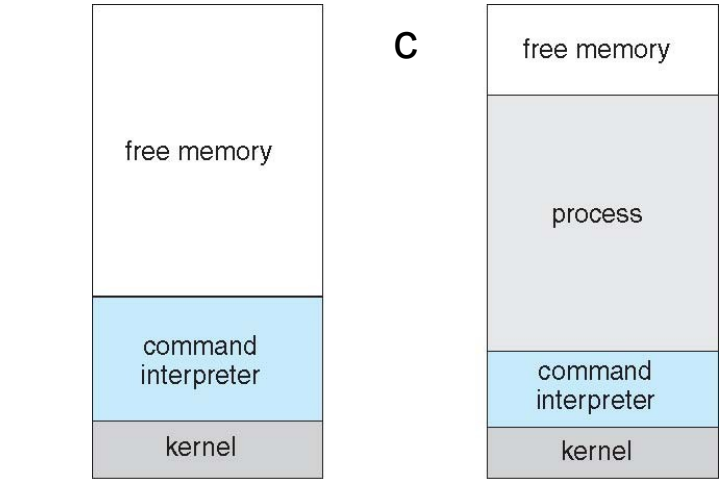
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information

Esempi di syscall in Windows e UNIX

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Esempio 1: MS-DOS

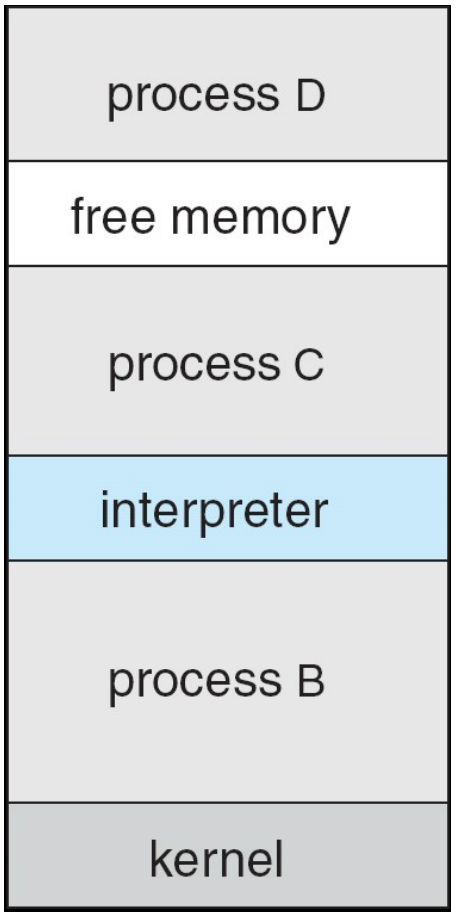
- Monotask → interprete caricato all'avvio, il S.O. lancia i programma, eventualmente sacrificando memoria occupata dall'interprete
- Al termina “ripristina” da disco l'interprete, notificando il risultato dell'esecuzione.



At system startup Running a program

Esempio 2: FreeBSD

- Multitasking UNIX → usa una shell (terminale)
  - I comandi sono **programmi**
- Syscall **fork()** per creare un **nuovo** processo
- Syscall **exec()** per **caricare** il programma in memoria (ovvero dare al nuovo processo un codice) e **lanciarlo**.
- La shell, a scelta dell'utente, può:
  - Attendere la fine del nuovo processo (in ***background***, comando **bg**)
  - Proseguire in parallelo (in ***foreground***, comando **fg**)



Riferimento sui sistemi UNIX

- Google → ”Open Group Base Specifications”

INDEX

Search..

[Alphabetic | Topic | Word Search | more...]

Select a Volume:
[Base Definitions | Shell & Utilities | System Interfaces | Rationale]

[Frontmatter]

Firefox Search Plugin

Base Definitions

1. Introduction
2. Conformance
3. Definitions
4. General Concepts
5. File Format Notation
6. Character Set
7. Locale
8. Environment Variables
9. Regular Expressions
10. Directory Structure and Devices
11. General Terminal Interface
12. Utility Conventions
13. Headers

IEEE

The Open Group Base Specifications Issue 6
IEEE Std 1003.1, 2004 Edition
Copyright © 2001-2004 The IEEE and The Open Group

THE Open GROUP

GoTo: Non-Frames Index | Register

This standard has been jointly developed by the IEEE and The Open Group. It is both an IEEE Standard and an Open Group Technical Standard.

Abstract: The 2004 edition incorporates Technical Corrigendum Number 1 and Technical Corrigendum 2 addressing problems discovered since the approval of the 2001 edition. These are mainly due to resolving integration issues raised by the merger of the Base documents.

This standard defines a standard operating system interface and environment, including a command interpreter (or "shell"), and common utility programs to support applications portability at the source code level. This standard is the single common revision to IEEE Std 1003.1-1996, IEEE Std 1003.2-1992, and the Base Specifications of The Open Group Single UNIX Specification, Version 2. This standard is intended to be used by both applications developers and system implementors. It comprises four major components (each in an associated volume):

1. General terms, concepts, and interfaces common to all volumes of this standard , including utility conventions and C-language header definitions, are included in the Base Definitions volume (XBD).
2. Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume (XSH).
3. Definitions for a standard source code-level interface to command interpretation services (a "shell" and common utility programs for application programs are included in the Shell and Utilities volume (XCU).
4. Extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of this standard and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume (XRAT).

Frontmatter (*Informative*)

[Preface | Participants | Referenced Documents | Trademarks | Typographical Conventions]

Tables of Contents by volume: [XBD | XSH | XCU | XRAT]

Links: [Alphabetic Index | Topical Index | About the HTML version | Downloads | Report a defect]

UNIX ® is a registered Trademark of The Open Group.
POSIX ® is a registered Trademark of The IEEE.
Copyright © 2001-2004 The IEEE and The Open Group, All Rights Reserved