ONE PIECE

3 - ENIES LOBBY

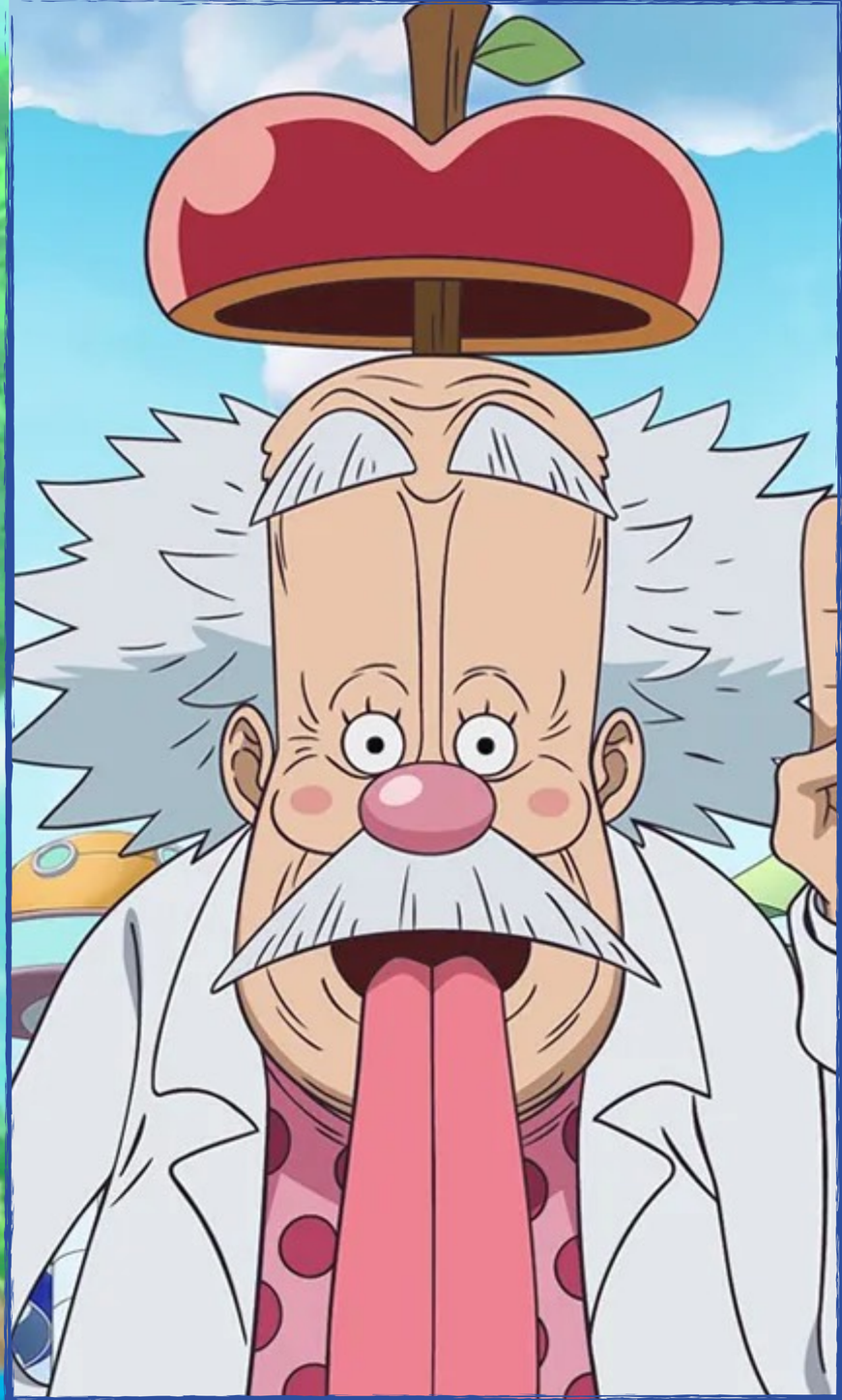
	Gen. - Feb. - Mar.
Enies Lobby	Introduzione
Nico Robin	Storia
Ohara	Architetture
Imperatori	Processi

OHARA

MEMORIA

"Il tempo speso ad aspettare è il peggior spreco di risorse."

(Andrew S. Tanenbaum)



Architetture

Sistemi Operativi

Prof. Andrea Bianco

Architetture

Argomenti del capitolo

- CPU e Interrupt
- Dispositivi di I/O e DMA
- System call

System Call

Architetture

Architetture

System Call

Program Interrupt

Oltre agli hardware interrupt, altri due tipi di program interrupt:

- Eccezioni
- Software Interrupt

Architetture

System Call

Eccezioni

Dette anche interrupt interni:

- Aritmetic fault: eccezioni aritmetiche, divisione per zero, overflow
- Page fault: eccezioni di indirizzamento, (capitolo Memoria)
- Violation fault: violazione delle protezioni di memoria (capitolo Memoria)

Le eccezioni sono eventi asincroni, indipendenti dall'istruzione in esecuzione.

Sono eventi causati da situazioni anomale interne al programma, che devono essere trattati dal SO. E' possibile che il SO imponga la terminazione immediata del programma.

BUG

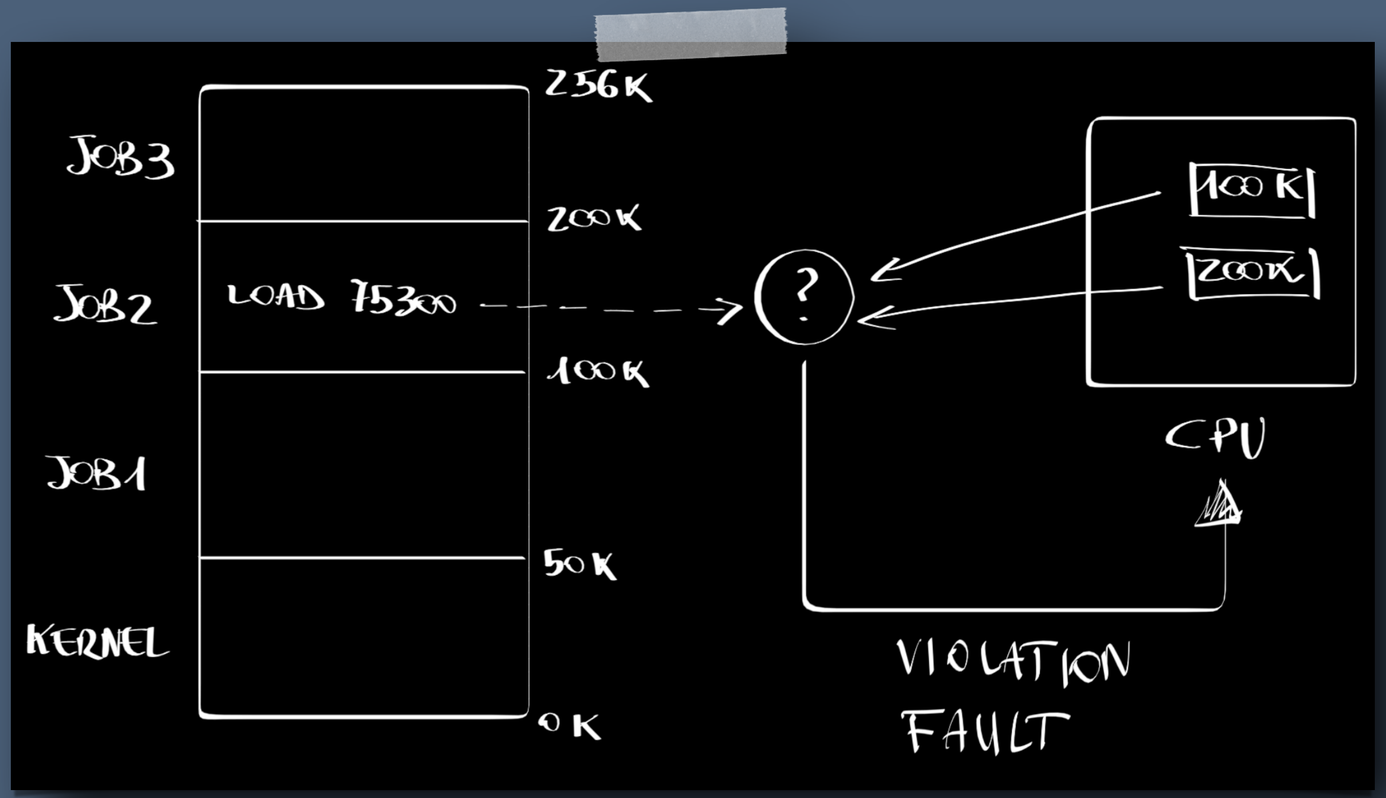
Le eccezioni sono eventi asincroni, indipendenti dall'istruzione in esecuzione.

Correzione:
Contrariamente agli hardware interrupt, non sono eventi asincroni rispetto all'esecuzione del programma.

Architetture

System Call

Esempio di violation fault con uso di registri LBR/UBR.



Architetture

System Call

Software interrupt

Sono causati da un'istruzione apposita, solitamente chiamata Software Interrupt Instruction, o, semplicemente, **trap**.

Sono usati dai programmi per chiedere esplicitamente l'intervento del SO.

Ogni system call utilizza una diversa istruzione TRAP.

La TRAP ha un parametro che specifica il tipo di intervento richiesto. Il parametro può essere passato in 2 modi:

- come operando della TRAP (se la TRAP prevede operandi)
- sullo stack

BUG

Ogni system call utilizza una diversa istruzione TRAP.

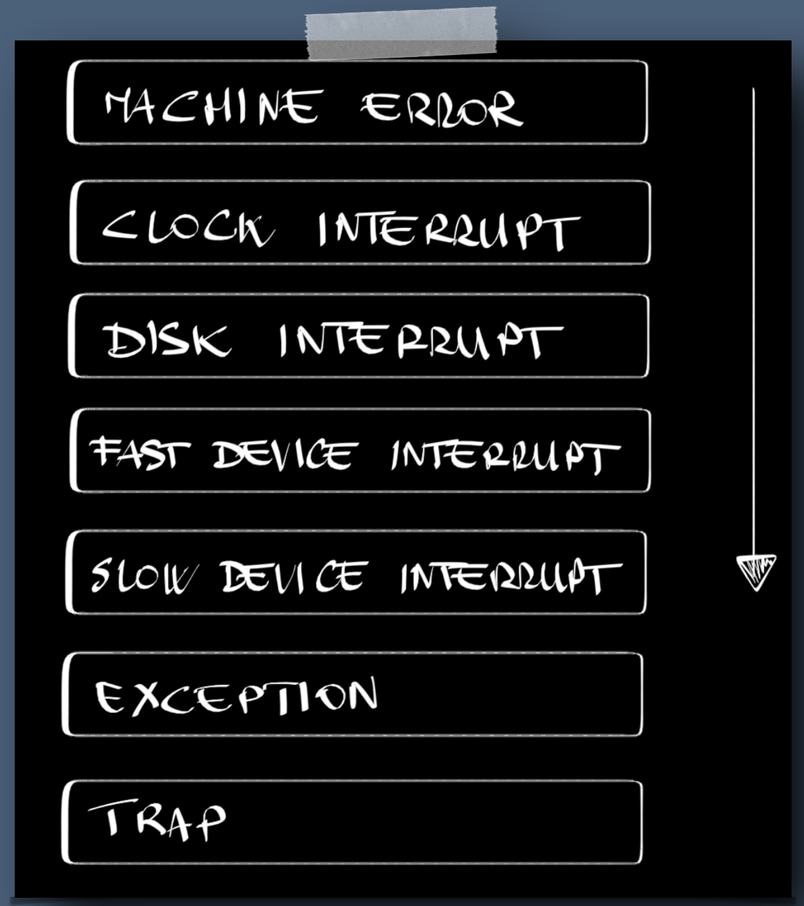
Correzione:
Il linguaggio macchina prevede un'unica istruzione per chiedere tutti i possibili interventi del SO, l'istruzione TRAP.

Architetture

System Call

I software interrupt vengono trattati come gli hardware interrupt.

La gerarchia di priorità tipica è aggiornata come segue:



ANSIA

Cosa succede quando si ritorna da un interrupt handler?

ANSIA

Cosa succede quando si ritorna da un interrupt handler?

Risposta:
Lo scheduler decide se ridare il controllo al programma interrotto o ad un altro.

Architetture

System Call

(*) TRAP 10 richiesta lettura valore del clock.

Il programma che esegue la TRAP si attende il risultato in uno dei registri generali, tipicamente il registro numero 0.

Questo valore deve essere nel registro 0 dopo che il programma ha ripreso il controllo, quindi l'interrupt handler va nella zona di memoria dove il valore di tale registro è stato salvato e lo cambia, memorizzando il valore del clock.

(Capitolo Processi, dove i registri generali vengono salvati)

Architetture

System Call

System call

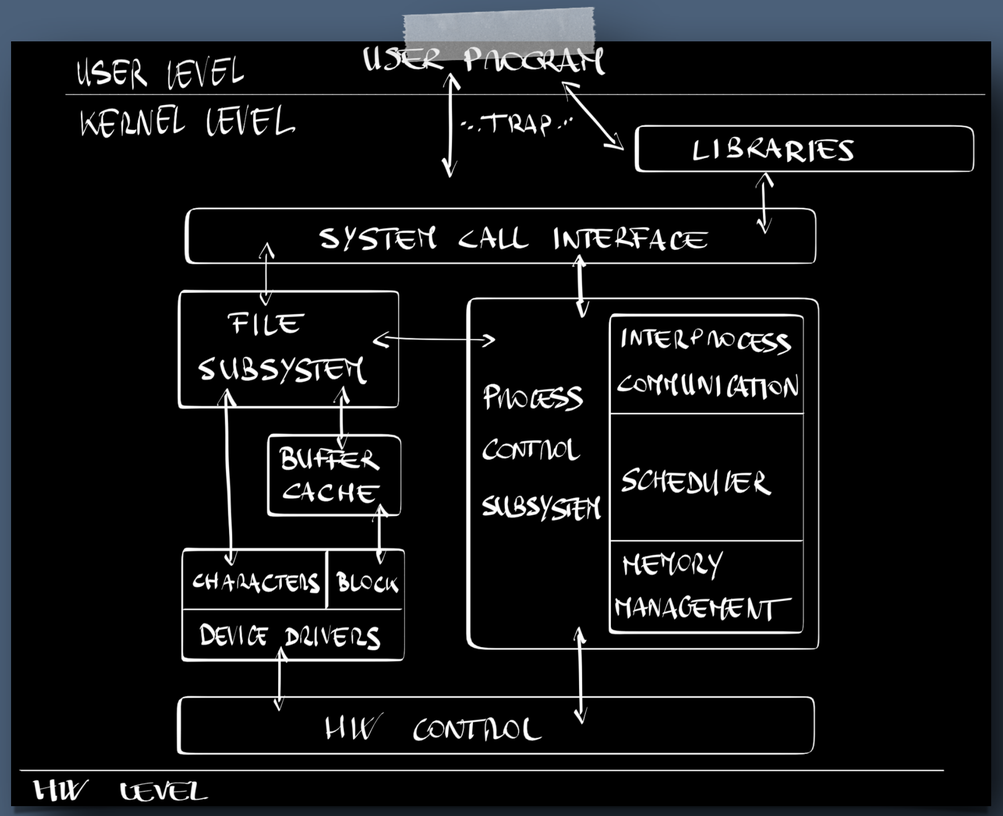
È una richiesta al SO effettuata da un programma.

Le system call sono lo strumento messo a disposizione dei programmi per avvalersi dei servizi offerti dal SO.

Vengono realizzate con la TRAP. Abbiamo una system call per ogni parametro valido per la TRAP. Se i bit dell'IC sono n, abbiamo <= 2^n system call. Tipicamente, n >= 8.

Il programma che invoca la system call ottiene un risultato, solitamente memorizzato in un registro.

Le system call sono l'interfaccia tra i programmi utente ed il SO.



Architetture

System Call

Il numero delle System call a disposizione dipende dal SO.

UNIX/Linux —> 100 system call circa.

Alcune per interagire con il file subsystem, che gestisce file e dispositivi.

Esempi:

- create, per creare un file
- open, per creare un accesso ad un file/device
- read, per leggere n byte da un file/device (n parametro)
- write, per scrivere n byte in un file/device (n parametro)
- close, per invalidare un accesso ad un file/device

Altre per interagire con il Process Control Subsystem (capitolo Processi).

Architetture

System Call

Invocazione di una system call

1. Assember: istruzione TRAP

Le system call hanno parametri che devono essere passati opportunamente.

Non è facile perché bisogna sapere dove l' interrupt handler della TRAP va a cercarli.

Parametro system call ≠ Parametro istruzione (individuazione TRAP)

Architetture

System Call

Invocazione di una system call

2. C: libreria di funzione

Le funzioni di libreria sono in assembler e sono invocabili dal programma in C.

Le funzioni di libreria invocano la TRAP.

In questo caso i parametri della system call possono essere passati alle funzioni di libreria come normali parametri di funzione.

Il programmatore può ragionare come se la system call fosse una normale chiamata di funzione.

Architetture

System Call

Esempio di programma in C per UNIX/Linux:

```
main( )
{
        int fd; /* fd è un intero */
        char buf[20], buff[30]; /* buf e buff sono array di caratteri */
        fd = open("fileDiProva",O_RDONLY); /* funzione di libreria */
        read(fd,buf,20); /* funzione di libreria */ /* leggo 20 byte */
        read(fd,buff,30); /* funzione di libreria */ /* leggo altri 30 byte */
}
```

<u>Open</u>, UNIX system call con due parametri:

- nome del file da aprire
- tipo di apertura

La open restituisce un *file descriptor*, cioè un accesso al file, che è un intero.

La funzione di libreria open ha lo stesso nome della system call, è invocabile dal programma in C specificando i parametri "fileDiProva" e O_RDONLY, cioè apertura solo in lettura, nell' esempio.

La funzione open invoca la system call open, cioè invoca la TRAP con il parametro che corrisponde alla open.

Architetture

System Call

```
main( )
{
        int fd; /* fd è un intero */
        char buf[20], buff[30]; /* buf e buff sono array di caratteri */
        fd = open("fileDiProva",O_RDONLY); /* funzione di libreria */
        read(fd,buf,20); /* funzione di libreria */ /* leggo 20 byte */
        read(fd,buff,30); /* funzione di libreria */ /* leggo altri 30 byte */
}
```

Read: UNIX system call con 3 parametri:

- nome del file
- buffer di memoria su cui scrivere
- numero di byte da leggere

La funzione di libreria read ha lo stesso nome della system call, è invocabile dal programma in C specificando i parametri "fileDiProva", buf e 20 nella prima chiamata, "fileDiProva", buff e 30 nella seconda.

La funzione read invoca la system call read, cioè invoca la TRAP con il parametro che corrisponde alla read.

Architetture

System Call

#include <fcntl.h>

Serve per importare le definizioni del file header fcntl.h, dove sono definite alcune costanti, quali la costante O_RDONLY usata nella open.

Il programma visto usa le funzioni di libreria open e read, che rispettano lo standard POSIX: il programma può girare su qualsiasi sistema UNIX/Linux che rispetti lo standard, perché certamente il sistema ha queste funzioni di libreria.

Il programma non gira su Windows o altri sistemi che non supportano le funzioni di libreria open e read.

I programmi in C sono portabili, però devo usare le funzioni della libreria quali fopen, fread, ecc...

Queste funzioni hanno un numero e tipo di parametri diversi rispetto ad open e read. Nel caso UNIX/Linux, la fopen chiama la open, la fread chiama la read.

Architetture

System Call

Mentre in UNIX/Linux di norma abbiamo una funzione di libreria per ogni system call, in Windows non è così.

APIWin32: definisce alcune migliaia di procedure che i programmi dovrebbero usare per invocare i servizi del SO (ruolo delle funzioni di libreria).

Le system call sono però molte meno, alcune procedure non invocano le system call ma vengono eseguite in modo user.

Esempi:

- CreateFile, per creare o aprire file,
- ReadFile, per leggere da un file,
- WriteFile, per scrivere su un file,
- CloseHandle, per chiudere un file.

ESERCIZIO

Disegna e spiega il flusso completo di una system call:

- istruzione TRAP
- salvataggio dei registri
- passaggio user → kernel
- ruolo dell'interrupt handler
- ritorno al programma utente

ESERCIZIO

Disegna e spiega il flusso completo di una system call:

- istruzione TRAP
- salvataggio dei registri
- passaggio user → kernel
- ruolo dell'interrupt handler
- ritorno al programma utente

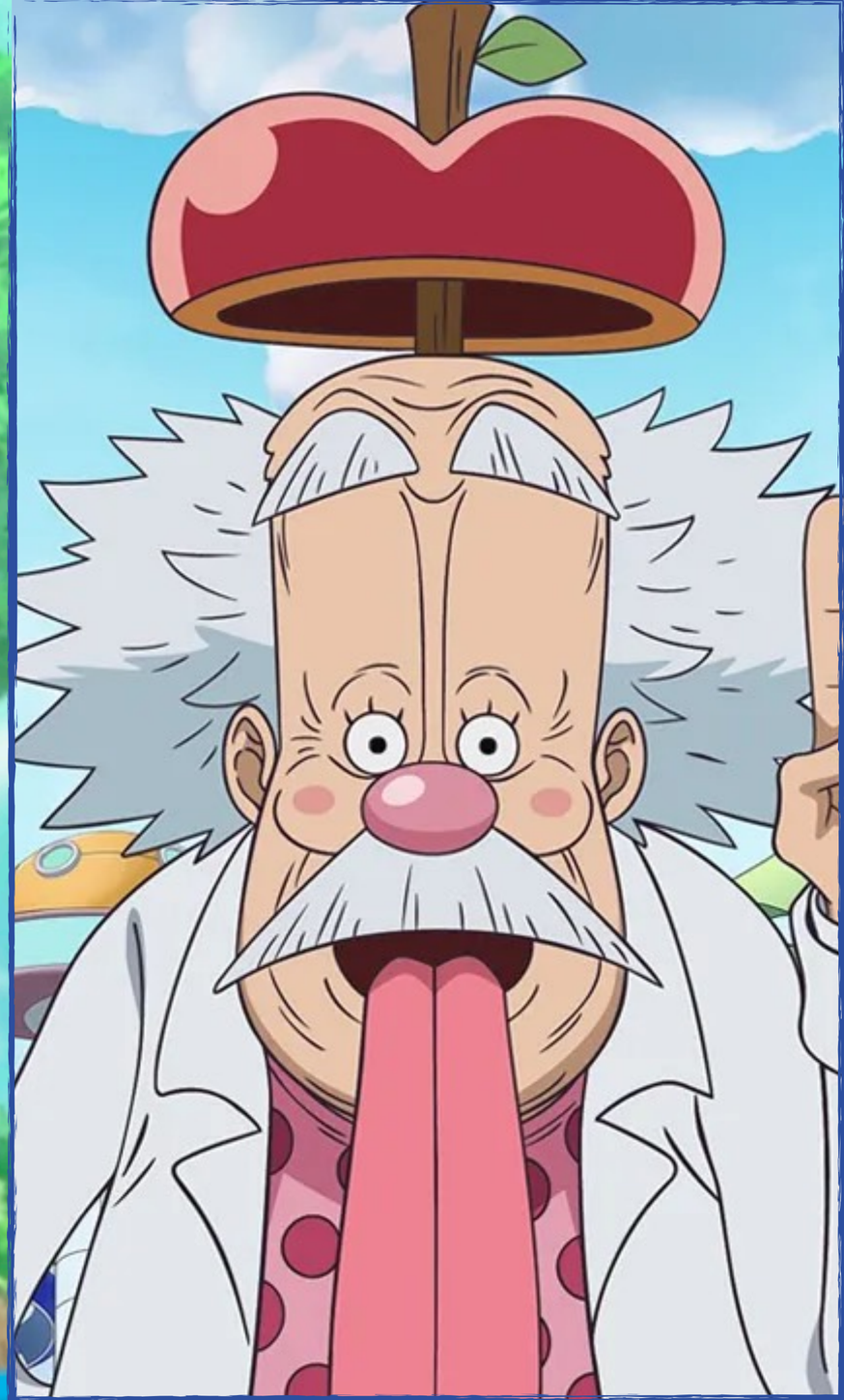
Risposta:

- passaggio user → kernel
- istruzione TRAP
- ruolo dell'interrupt handler
- salvataggio dei registri
- ritorno al programma utente

MEMORIA

"Ogni problema dell'informatica può essere risolto con un ulteriore livello di indirezione."

(David J. Wheeler)



COMPITI

Spiega in 10–15 righe:
Perché le system call sono considerate un compromesso tra sicurezza e flessibilità.

Rispondi: perché Windows espone migliaia di API ma ha molte meno system call reali rispetto alle funzioni disponibili ai programmatori?