ONE PIECE

3 - ENIES LOBBY

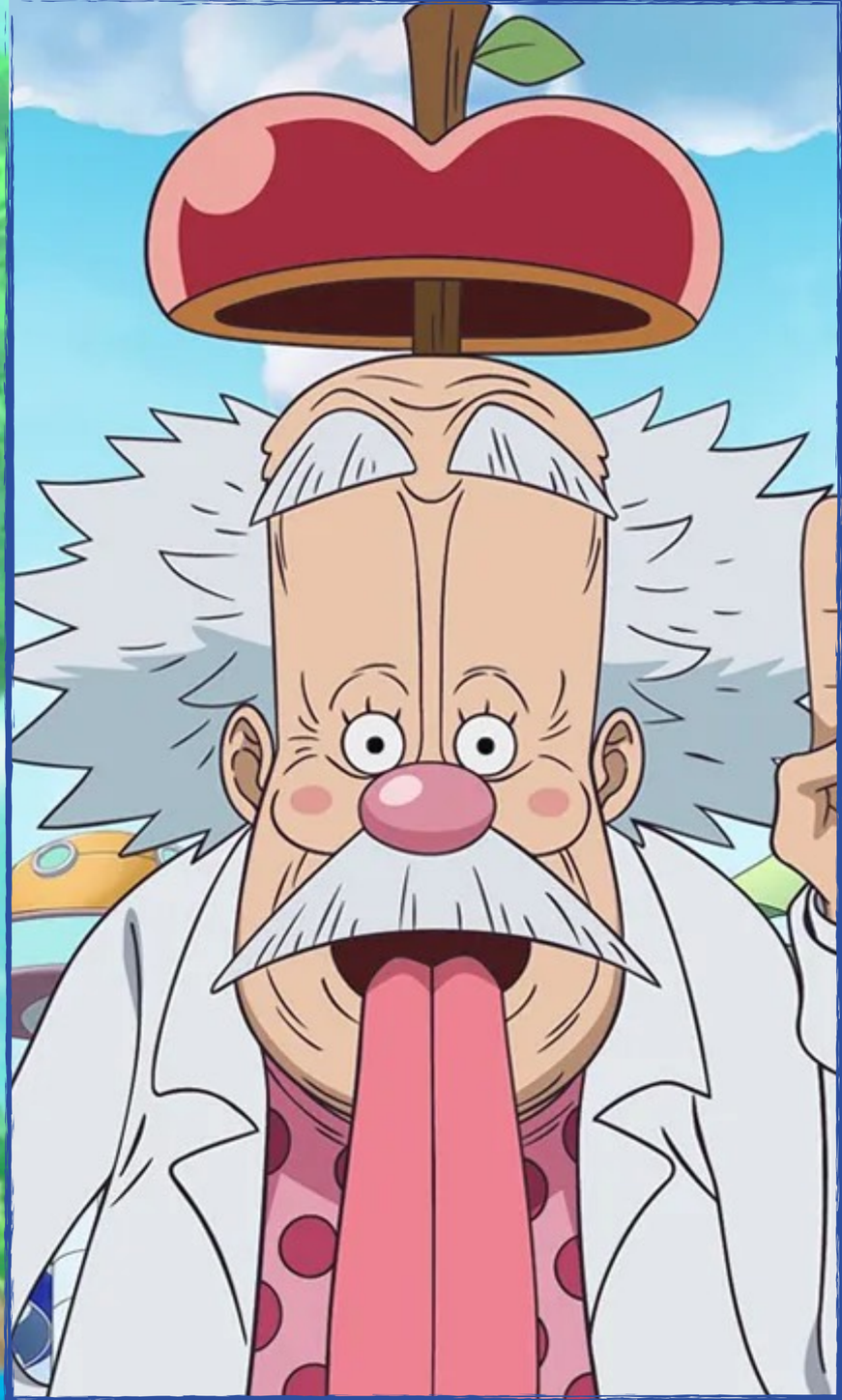
	Gen. - Feb. - Mar.
Enies Lobby	Introduzione
Nico Robin	Storia
Ohara	Architetture
Imperatori	Processi

OHARA

MEMORIA

"La complessità è il peggior nemico dell'affidabilità."

(Edsger W. Dijkstra)



Architetture

Sistemi Operativi

Prof. Andrea Bianco

Architetture

Argomenti del capitolo

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

Dispositivi di I/O e DMA

Architetture

Architetture

Dispositivi di I/O e DMA

Dispositivi di I/O

- Controller o adapter: componente software del sistema operativo che si occupa della gestione dei dispositivi di I/O.
- Dispositivo: componente meccanica che viene gestita dal controller.

In generale, un controller può gestire più dispositivi contemporaneamente.

Le interfacce tra controller e dispositivo sono solitamente standard (SATA, SCSI, USB, FireWire) e si possono produrre dispositivi per una data interfaccia standard.

B U G

componente software del sistema operativo che si occupa della gestione dei dispositivi di I/O.

Correzione:
componente elettronica che comunica con la CPU e le altre unità via bus.

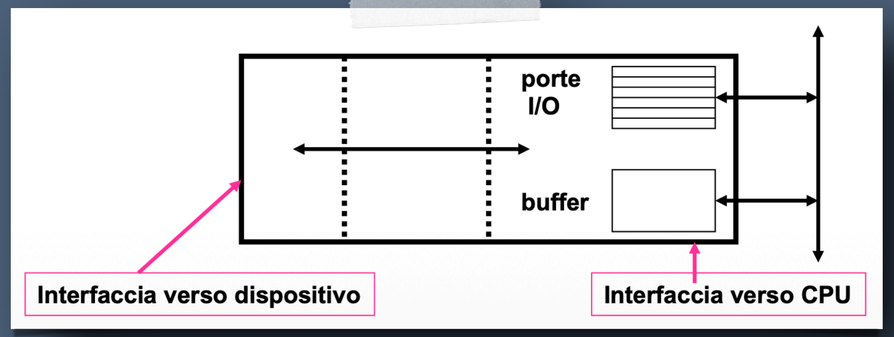
Architetture

Dispositivi di I/O e DMA

L'interfaccia tra CPU e controller viene usata dai programmi del SO che gestiscono i dispositivi (driver).

Questa interfaccia prevede:

- registri di controllo-porte I/O:
  - vengono usati dalla CPU per inviare comandi al controller
  - vengono usati dal controller per comunicare i risultati dei comandi e lo stato del dispositivo alla CPU
- buffer: serve per memorizzare dati durante le operazioni di I/O.



Architetture

Dispositivi di I/O e DMA

Due soluzioni per le comunicazioni tra la CPU e le porte di I/O:

Soluzione 1

Porte di I/O gestite con istruzioni macchina ad hoc.

- IN —> R, P: il valore della porta P viene caricato nel registro R
- OUT —> P, R: il valore del registro R viene caricato nella porta P

Le istruzioni IN e OUT possono essere eseguite anche in modalità user, purché il processo abbia i permessi corretti.

Alcune parti dei driver devono essere in assembly, perché nei linguaggi ad alto livello non si possono avere istruzioni corrispondenti.

BUG

Le istruzioni IN e OUT possono essere eseguite anche in modalità user, purché il processo abbia i permessi corretti.

Correzione:
Privilegiate (kernel), per impedire che in modalità user si riesca ad accedere ai dispositivi.

ANSIA

Ciclo funzionamento CPU.

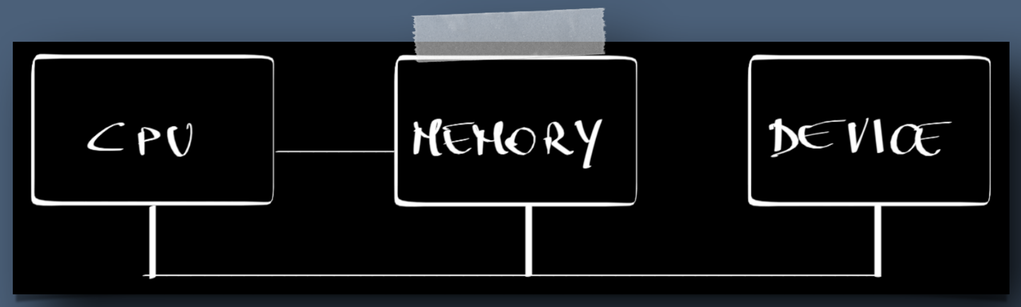
ANSIA

Ciclo funzionamento CPU.

Risposta:
Fetch - Decode - Execute

Architetture

Dispositivi di I/O e DMA



Soluzione 2

<u>Memory Mapped I/O</u>

Ad ogni porta di I/O è assegnato un indirizzo di memoria e non servono istruzioni ad hoc.

Questi indirizzi non sono visibili dai programmi, che devono pertanto invocare il SO per accedere ai dispositivi.

I driver possono essere scritti in C.

È più complicata la gestione della cache, in quanto il contenuto di queste locazioni è modificabile dal dispositivo e non solo dalla CPU.

Tutti i moduli di memoria ed i dispositivi di I/O devono esaminare tutti i riferimenti di memoria. Il tutto si complica se usiamo un bus dedicato tra CPU e Memoria (vedi immagine).

Architetture

Dispositivi di I/O e DMA

Tre possibili soluzioni per eseguire I/O per un programma P.

Si vogliono trasferire n byte verso il (buffer del controller del) dispositivo da un buffer di memoria b.

Soluzione 1

<u>Programmed I/O</u>

Codice eseguito dal SO su richiesta di P:

```
for(i = 0; i < n, i++)
{
        while(device_status_reg != READY){ } /* busy waiting */
        buffer = b[i]
}
```

buffer: buffer I/O per scambio dati.

device_status_reg: porta I/O per testare lo stato del device.

Nel Programmed I/O la CPU può svolgere altro lavoro mentre il dispositivo trasferisce i dati.

BUG

Nel Programmed I/O la CPU può svolgere altro lavoro mentre il dispositivo trasferisce i dati.

Correzione:
La CPU rimane inutilizzata mentre aspetta che il dispositivo abbia finito di trattare il singolo byte (busy waiting). Non si sfruttano gli interrupt.

Architetture

Dispositivi di I/O e DMA

Soluzione 2

Interrupt Driven I/O

Codice eseguito dal SO su richiesta di P:

```
while(device_status_reg != READY){/* busy waiting, solo all'inizio*/ }
buffer = b[0];
c = 1;
i = 1;
scheduler( ); /*P lascia la CPU, la riotterrà ad operazione conclusa*/
```

Codice eseguito dall'interrupt handler:

```
if (c == n)
{
        unblock_user( ); /* il programma P può ripartire */
}
else
{
        buffer = b[i]; /*chiedo al device il prossimo transfer*/
        /*e attendo il prossimo interrupt*/
        i = i +1;
        c = c + 1;
}

return_from_interrupt( )
```

Non abbiamo più busy waiting: si evita che la CPU sia impegnata solo per aspettare che il device tratti il singolo byte.

Abbiamo però n interruzioni, una per ogni byte, che causano ovviamente overhead.

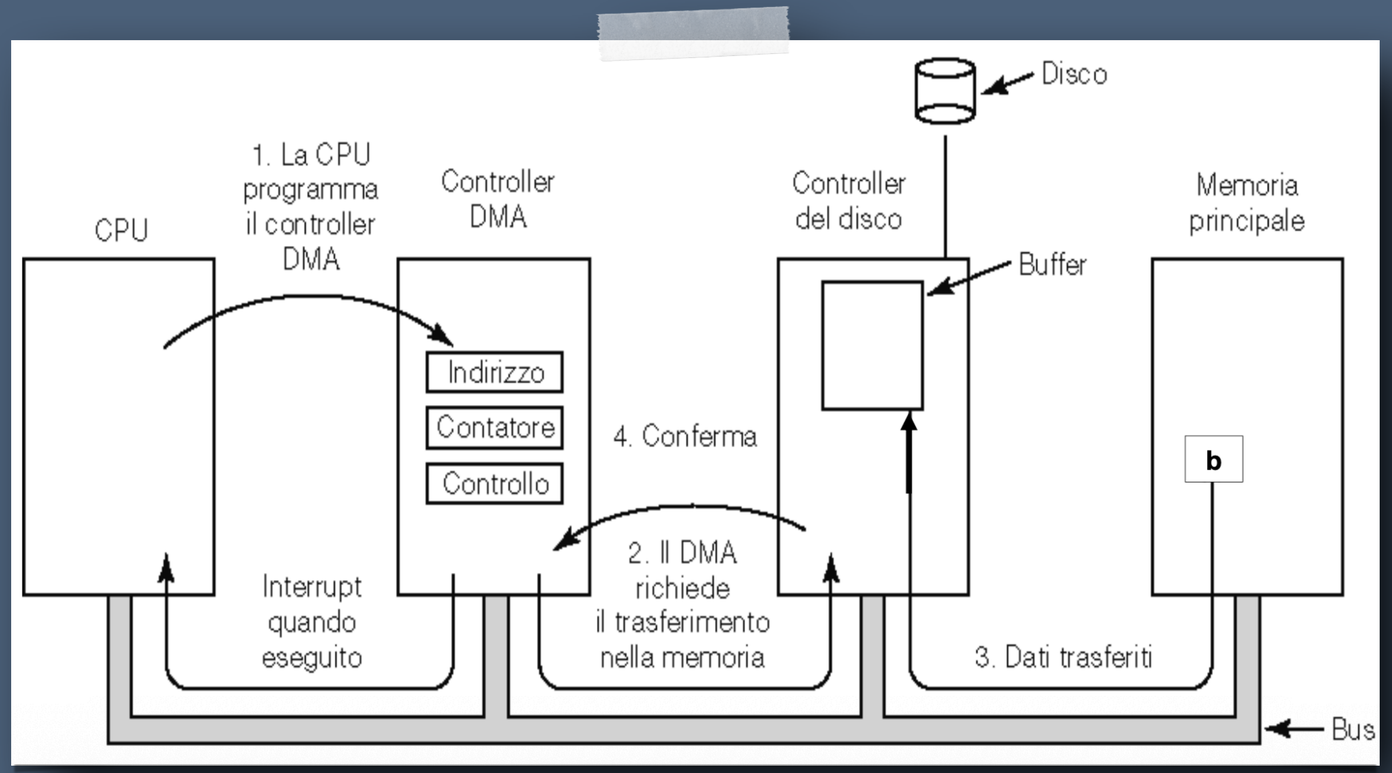
Architetture

Dispositivi di I/O e DMA

Soluzione 3

I/O con DMA

L' architettura prevede un controller Direct Memory Access (DMA) capace di accedere direttamente alla memoria e di lavorare in parallelo con la CPU.



Architetture

Dispositivi di I/O e DMA

Soluzione 3

Codice eseguito dal SO su richiesta di P:

```
set_up_DMA_controller( ); /* imposto i registri del DMA controller*/
scheduler( ); /*P lascio la CPU, la riotterrò ad operazione conclusa*/
```

Codice eseguito dall'interrupt handler che gestisce l'interrupt inviato dal DMA per comunicare la conclusione dell'operazione:

```
unblock_user( ); /* il programma P può ripartire */
return_from_interrupt( )
```

Il ciclo con il trasferimento degli n byte viene effettuato dal controller DMA, che, al termine del ciclo, invia un interrupt alla CPU. Il tutto mentre la CPU fa altro lavoro.

ESERCIZIO

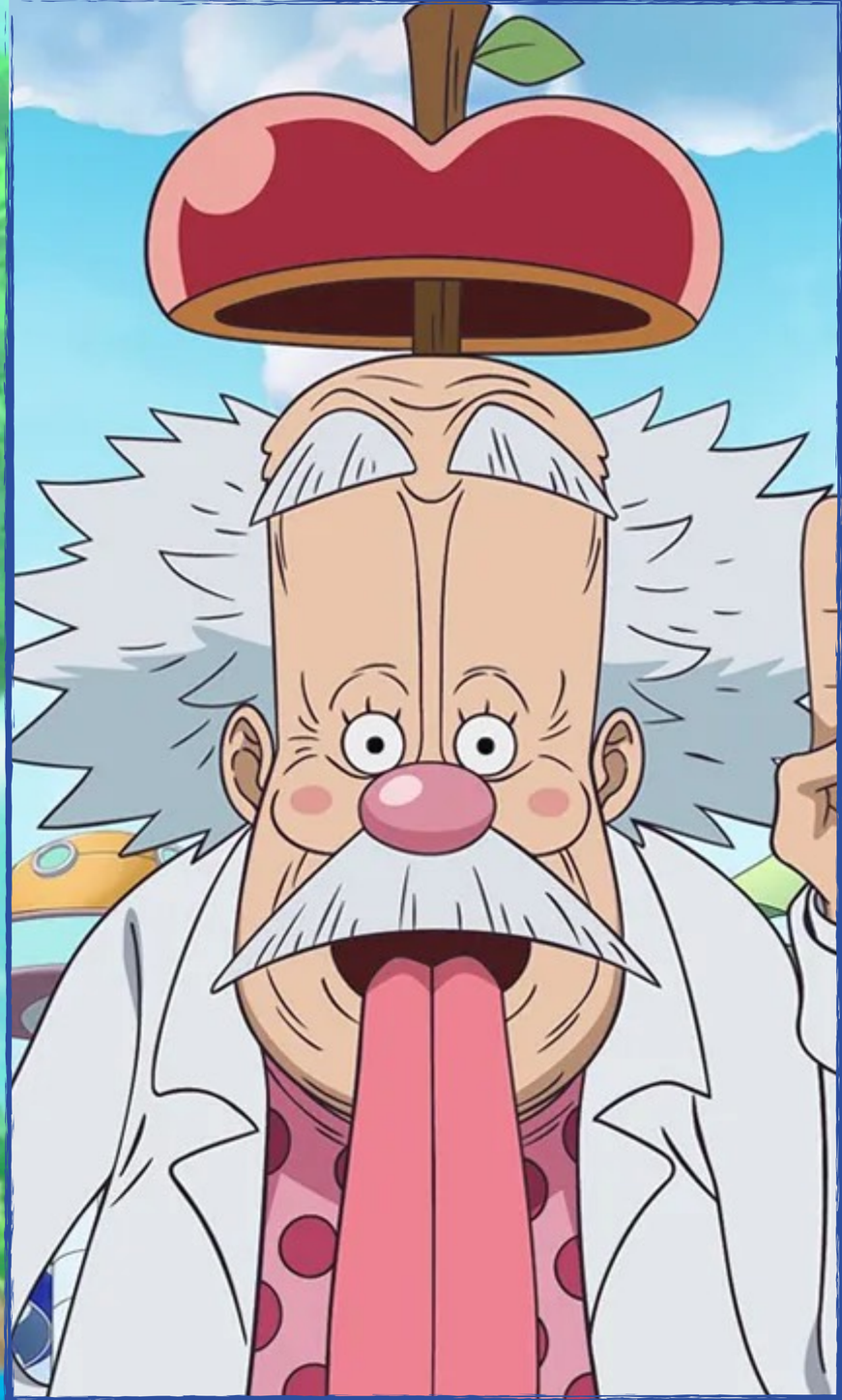
Confronta Programmed I/O, Interrupt Driven I/O e DMA compilando una tabella con:

- uso della CPU
- numero di interrupt
- overhead
- scenario in cui ciascuna soluzione è più conveniente

MEMORIA

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

(Andrew S. Tanenbaum)



COMPITI

Fai un esempio concreto (disco, rete, USB):

- quale tecnica di I/O useresti
- perché le altre due sarebbero meno adatte