

Sistemi Operativi

Introduzione ai sistemi operativi II semestre

Renzo Davoli
Alberto Montresor

Copyright © 2002-2022 Renzo Davoli, Alberto Montresor

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be found at:
<http://www.gnu.org/licenses/fdl.html#TOC1>

I semestre

- ◆ **Uso del S.O. UNIX: shell - command line interface** (*prova pratica + progetto*)
- ◆ **Linguaggio C** (*prova pratica + progetto*)
- ◆ **Programmazione UNIX: l'interfaccia delle System Call** (*prova pratica*)
- ◆ **Programmazione Concorrente** (*scritto "concorrenza"*)
- ◆ **Shell scripting** (*prova pratica*)
- ◆ **Python** (*prova pratica*)
- ◆ **Strumenti di sviluppo / gestione dei progetti (make, autotools, cmake, git..)** (*progetto*)
- ◆ **Progetto uMPS3 + phase 1** (*progetto*)

Cos'è un sistema operativo?

- ◆ **Definizione:**

- ◆ Un sistema operativo è livello di astrazione:
 - ◆ realizza il concetto di processo
 - ◆ Il “linguaggio” fornito dal S.O. è definito dalle system call
 - ◆ È implementato tramite un programma che *controlla l'esecuzione di programmi applicativi* e agisce come *interfaccia tra le applicazioni e l'hardware* del calcolatore

- ◆ **Obiettivi**

- ◆ Efficienza:
 - ◆ Un S.O. cerca di utilizzare in modo efficiente le risorse del calcolatore
- ◆ Semplicità:
 - ◆ Un sistema operativo dovrebbe semplificare l'utilizzazione dell'hardware di un calcolatore

S.O. come gestore di risorse

- ◆ **Alcune osservazioni:**

- ◆ Gestendo le risorse di un calcolatore, un S.O. controlla il funzionamento del calcolatore stesso...
- ◆ ... ma questo controllo è esercitato in modo "particolare"

- ◆ **Normalmente:**

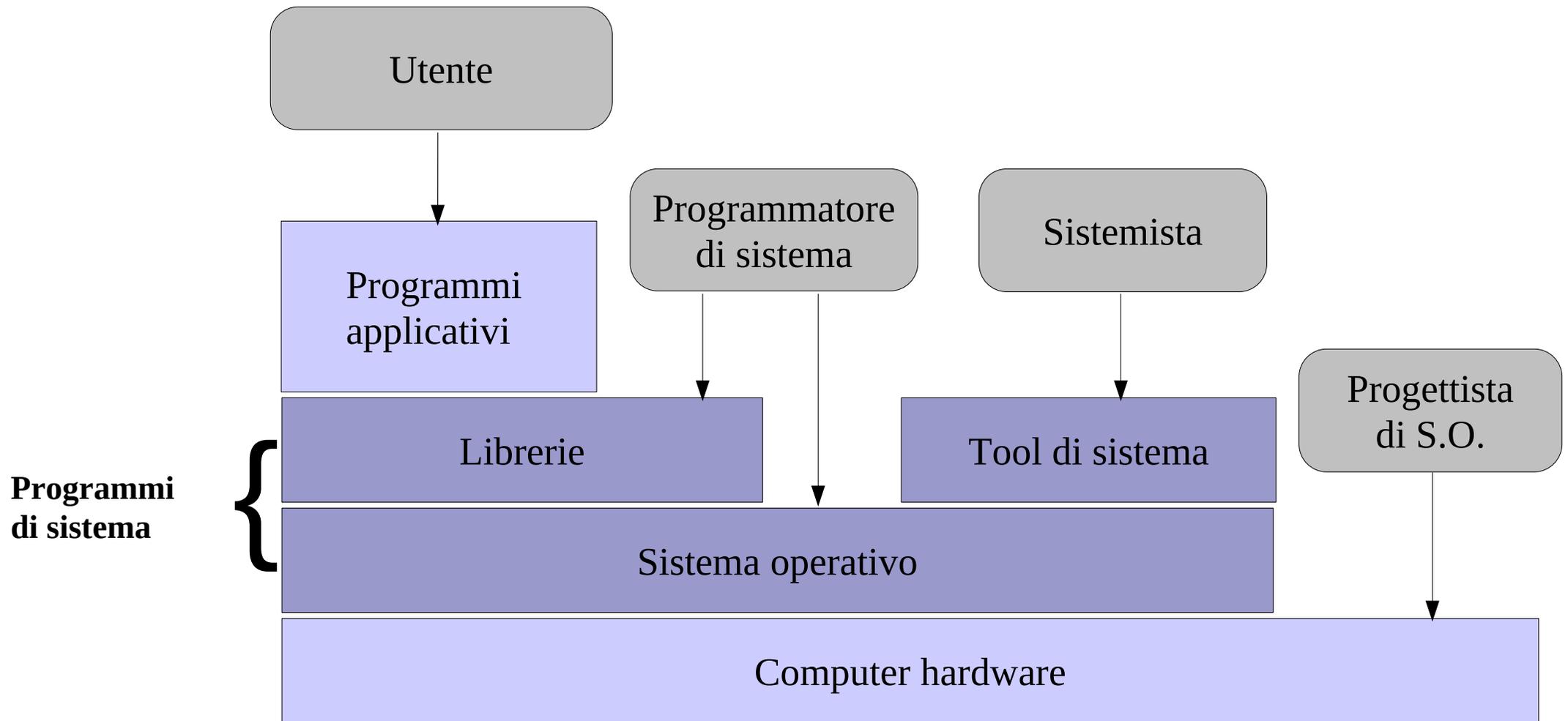
- ◆ Il meccanismo di controllo è esterno al sistema controllato
- ◆ Esempio: termostato e impianto di riscaldamento

- ◆ **In un elaboratore:**

- ◆ Il S.O. è un programma, simile all'oggetto del controllo, ovvero le applicazioni controllate
- ◆ Il S.O. deve lasciare il controllo alle applicazioni e affidarsi al processore per riottenere il controllo

S.O. come macchina estesa

- **Visione "a strati" delle componenti hardware/software che compongono un elaboratore:**



S.O. come macchina estesa

- ◆ **In questa visione, un sistema operativo:**
 - ◆ nasconde ai programmatori i dettagli dell'hardware e fornisce ai programmatori una API conveniente e facile da usare
 - ◆ agisce come intermediario tra programmatore e hardware
- ◆ **Parole chiave:**
 - ◆ Indipendenza dall'hardware
 - ◆ Comodità d'uso
 - ◆ Programmabilità

S.O. come macchina estesa

- ◆ **Esempio: floppy disk drive**

- ◆ I floppy drive delle macchine Intel sono compatibili con il controllore NEC PD765
- ◆ 16 comandi
 - ◆ inizializzazione, avviamento motore, spostamento testina, lettura-scrittura, spegnimento motore
 - ◆ formato: vari parametri, impacchettati in 1-9 byte
 - ◆ esempio: comando read, 13 parametri
- ◆ al completamento, il driver restituirà 23 campi di stato e di errore racchiusi in 7 byte

S.O. come macchina estesa

- ◊ **Esempio senza S.O.**

```
li $t0, 0xDEFF12 # init
```

```
sw $t0, 0xB000.0040
```

```
li $t0, 0xFFDF # motor
```

```
sw $t0, 0xB000.0044
```

```
li $t0, 0xFFBB
```

```
sw $t0, 0xB000.0048
```

...

NB: Questo è esempio serve a dare un'idea,
la realtà è molto più complessa....

- ◊ **Esempio con S.O.**

```
fd = open("/etc/rpc");
```

```
read(fd, buffer, size);
```

S.O. come macchina estesa

- ♦ **Servizi estesi offerti da un S.O.**
- ♦ **(cono le classi di SysCall studiate nel I semestre):**
 - ♦ esecuzione di programmi
 - ♦ accesso semplificato/unificato ai dispositivi di I/O
 - ♦ accesso a file system
 - ♦ accesso a networking
 - ♦ accesso al sistema
 - ♦ rilevazione e risposta agli errori
 - ♦ accounting



Storia dei sistemi operativi

- ♦ **L'evoluzione dei sistemi operativi**
 - ♦ *è stata spinta* dal progresso tecnologico nel campo dell'hardware
 - ♦ *ha guidato* il progresso tecnologico nel campo dell'hardware
- ♦ **Esempio:**
 - ♦ Gestione degli interrupt
 - ♦ Protezione della memoria
 - ♦ Memoria virtuale
 - ♦

- ◆ **Perché analizzare la storia dei sistemi operativi?**

- ◆ Perché permette di capire l'origine di certe soluzioni presenti oggi nei moderni sistemi operativi
- ◆ Perché è l'approccio didattico migliore per capire come certe idee si sono sviluppate
- ◆ Perché alcune delle soluzioni più vecchie sono ancora utilizzate

- ◆ **Durante il corso:**

- ◆ illustreremo ogni argomento
 - ◆ partendo dalle prime soluzioni disponibili
 - ◆ costruendo sopra di esse soluzioni mano a mano più complesse
- ◆ non stupitevi quindi se alcune soluzioni vi sembreranno banali e ingenuie; sono soluzioni adottate 10,20,30,40 o 50 anni fa!

Storia dei Sistemi Operativi

- ♦ **Generazione 1: 1945 - 1955**
 - ♦ valvole e tavole di commutazione
- ♦ **Generazione 2: 1955 - 1965**
 - ♦ transistor e sistemi batch
- ♦ **Generazione 3: 1965 - 1980**
 - ♦ circuiti integrati, multiprogrammazione e time-sharing
- ♦ **Generazione 4: 1980 - oggi**
 - ♦ personal computer

Generazione 0

- ◆ **Babbage (1792-1871)**

- ◆ Cerca di costruire la macchina analitica (programmabile, meccanica)
- ◆ Non aveva sistema operativo
- ◆ La prima programmatrice della storia e' Lady Ada Lovelace (figlia del poeta Lord Byron)

Generazione 1 (1944-1955)

- ◆ **Come venivano costruiti?**

- ◆ macchine a valvole e tavole di commutazione

- ◆ **Come venivano usati?**

- ◆ solo calcoli numerici (calcolatori non elaboratori)
- ◆ un singolo gruppo di persone progettava, costruiva, programmava e manteneva il proprio computer

- ◆ **Come venivano programmati?**

- ◆ in linguaggio macchina
 - ◆ programmazione su tavole di commutazione
 - ◆ non esisteva il concetto di assembler!

- ◆ **Nessun sistema operativo!**

Generazione 1 (1944-1955)

◆ **Principali problemi**

- ◆ grossi problemi di affidabilità (guasti frequenti)
- ◆ rigidità nell'assegnazione dei ruoli;
 - ◆ non esiste il concetto di programmatore come entità separata dal costruttore di computer e dall'utente
- ◆ utilizzazione lenta e complessa; l'operatore doveva:
 - ◆ caricare il programma da eseguire
 - ◆ inserire i dati di input
 - ◆ eseguire il programma
 - ◆ attendere il risultato
 - ◆ ricominciare dal punto 1.
- ◆ tutto ciò a causa dell'assenza del sistema operativo

Generazione 1 (1944-1955)

- ♦ **Fraasi celebri**

(da una lezione di P. Ciancarini)

- ♦ Nel futuro i computer arriveranno a pesare non più di una tonnellata e mezzo (Popular Mechanics, 1949)
- ♦ Penso che ci sia mercato nel mondo per non più di cinque computer (Thomas Watson, presidente di IBM, 1943)
- ♦ Ho girato avanti e indietro questa nazione (USA) e ho parlato con la gente. Vi assicuro che questa moda dell'elaborazione automatica non vedrà l'anno prossimo (Editor di libri scientifici di Prentice Hall, 1947)

Generazione 2 (1955-1965)

- ◆ **Come venivano costruiti?**
 - ◆ introduzione dei transistor
 - ◆ costruzione di macchine più affidabili ed economiche
- ◆ **Come venivano usati?**
 - ◆ le macchine iniziano ad essere utilizzate per compiti diversi
 - ◆ si crea un mercato, grazie alle ridotte dimensioni e al prezzo più abbordabile
 - ◆ avviene una separazione tra costruttori, operatori e programmatori
- ◆ **Come venivano programmati?**
 - ◆ linguaggi ad "alto livello": Assembly, Fortran
 - ◆ tramite schede perforate
- ◆ **Sistemi operativi batch**

Generazione 2 (1955-1965)

- ♦ **Definizione: job**
 - ♦ Un programma o un'insieme di programmi la cui esecuzione veniva richiesta da uno degli utilizzatori del computer
- ♦ **Ciclo di esecuzione di un job:**
 - ♦ **il programmatore**
 - ♦ scrive (su carta) un programma in un linguaggio ad alto livello
 - ♦ perfora una serie di schede con il programma e il suo input
 - ♦ consegna le schede ad un operatore
 - ♦ **l'operatore**
 - ♦ inserisce schede di controllo scritte in JCL
 - ♦ inserisce le schede del programma
 - ♦ attende il risultato e lo consegna al programmatore
- ♦ **Nota: operatore != programmatore == utente**

Generazione 2 (1955-1965)

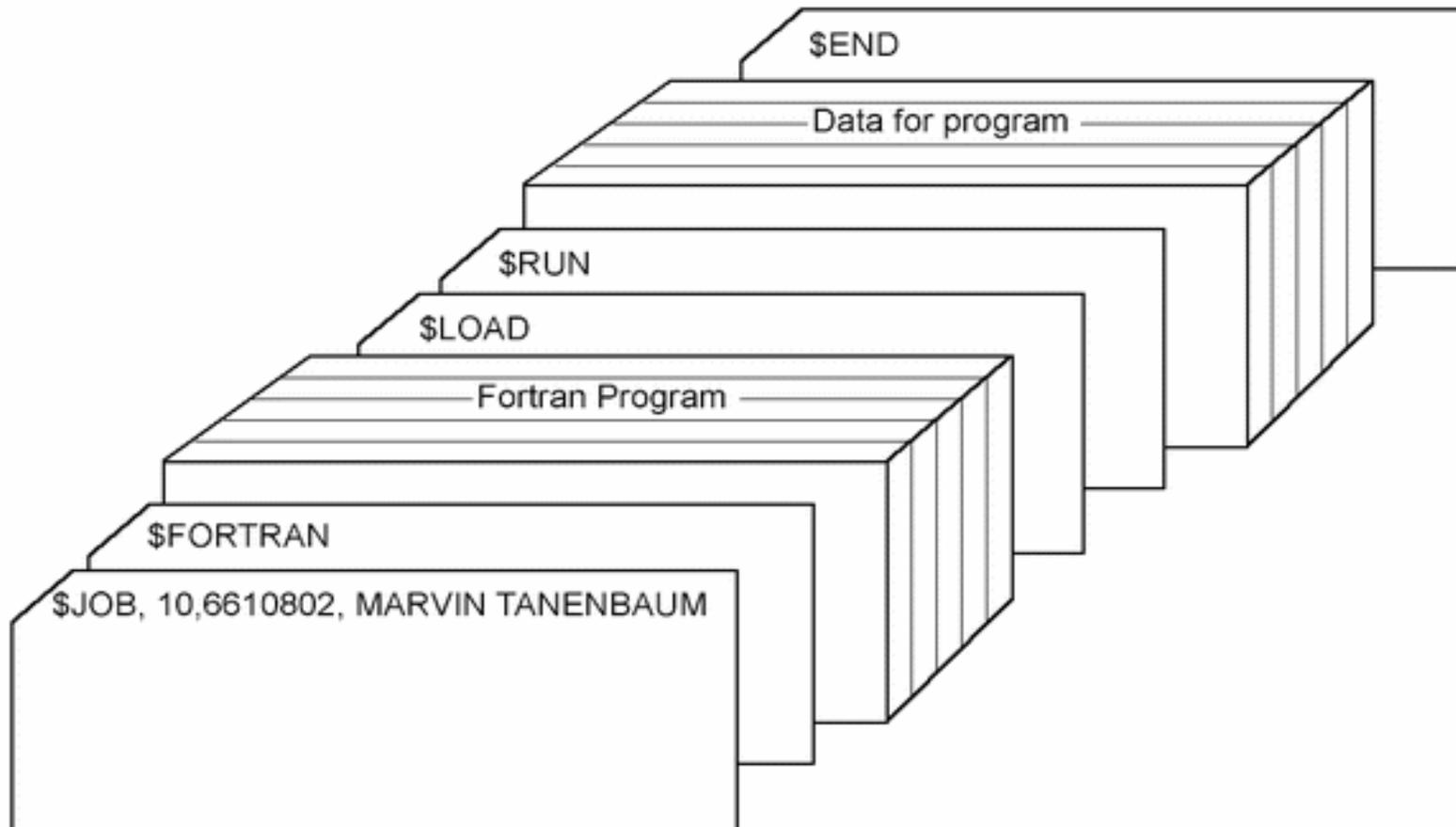
- ◆ **Sistema operativo**

- ◆ *primi rudimentali esempi di sistema operativo*, detti anche monitor residenti:
 - ◆ controllo iniziale nel monitor
 - ◆ il controllo viene ceduto al job corrente
 - ◆ una volta terminato il job, il controllo ritorna al monitor
- ◆ il monitor residente è in grado di eseguire una sequenza di job, trasferendo il controllo dall'uno all'altro
- ◆ Esegue un solo job alla volta

- ◆ **Detti anche sistemi batch ("infornata")**

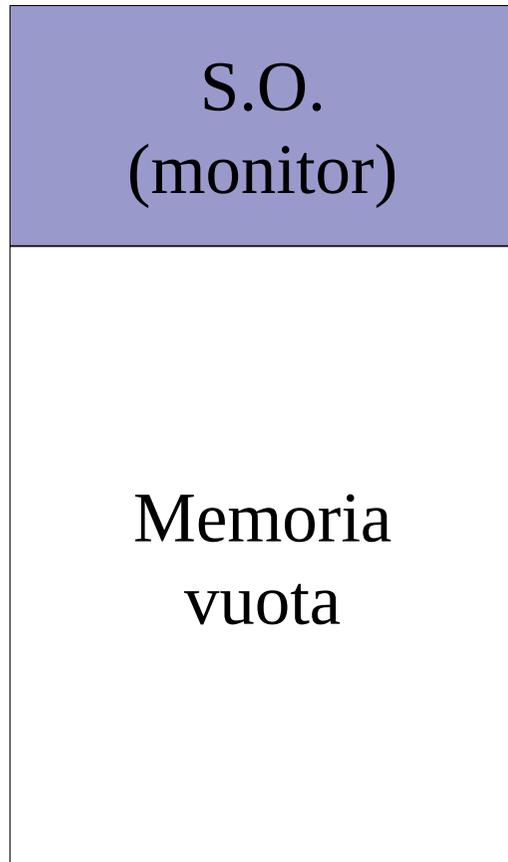
Generazione 2 (1955-1965)

- ♦ **Job Control Language (JCL-FMS)**

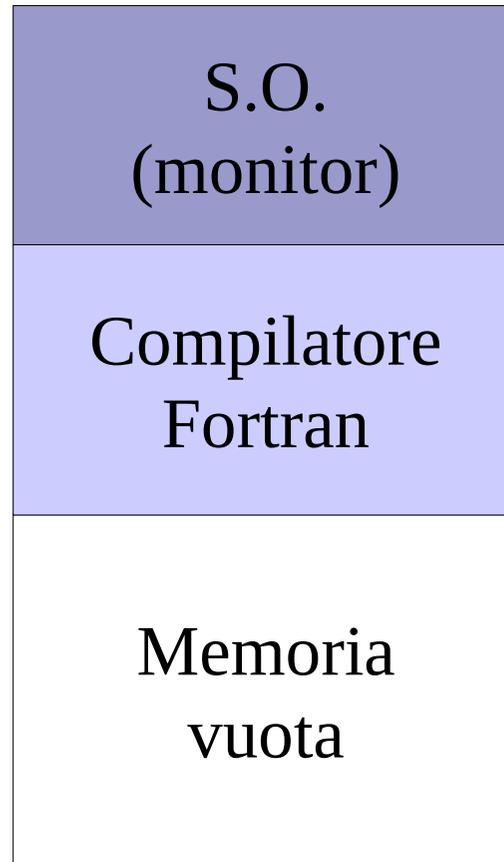


Generazione 2 (1955-1965)

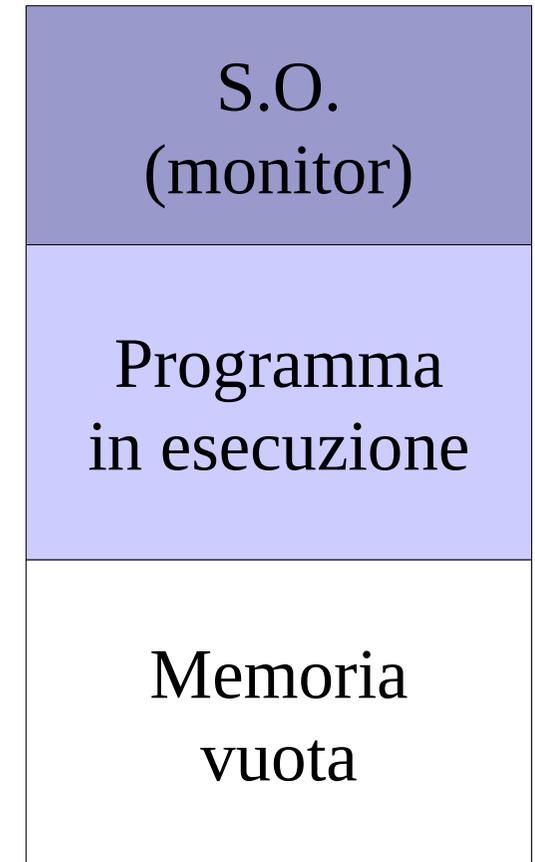
- Memory layout per un S.O. batch (versione semplificata)



1. Stato iniziale



2. Dopo il caricamento del compilatore fortran



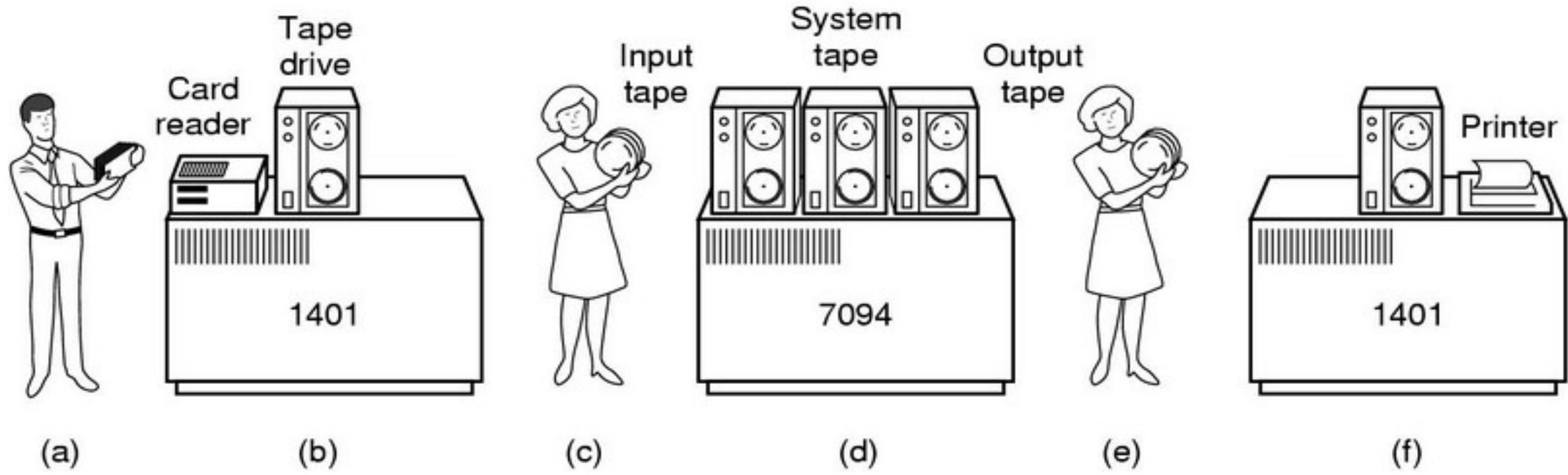
3. Dopo la compilazione

Generazione 2 (1955-1965)

- ◆ **Principali problemi**

- ◆ Molte risorse restavano inutilizzate:
 - ◆ durante le operazioni di lettura schede / stampa, durante il caricamento di un nuovo job, il processore restava inutilizzato
 - ◆ parte della memoria restava inutilizzata
- ◆ Primo miglioramento (ma non una soluzione)
 - ◆ caricamento di numerosi job su nastro (off-line)
 - ◆ elaborazione (output su nastro)
 - ◆ stampa del nastro di output (off-line)

Generazione 2 (1955-1965)



Generazione 3 (1965-1980)

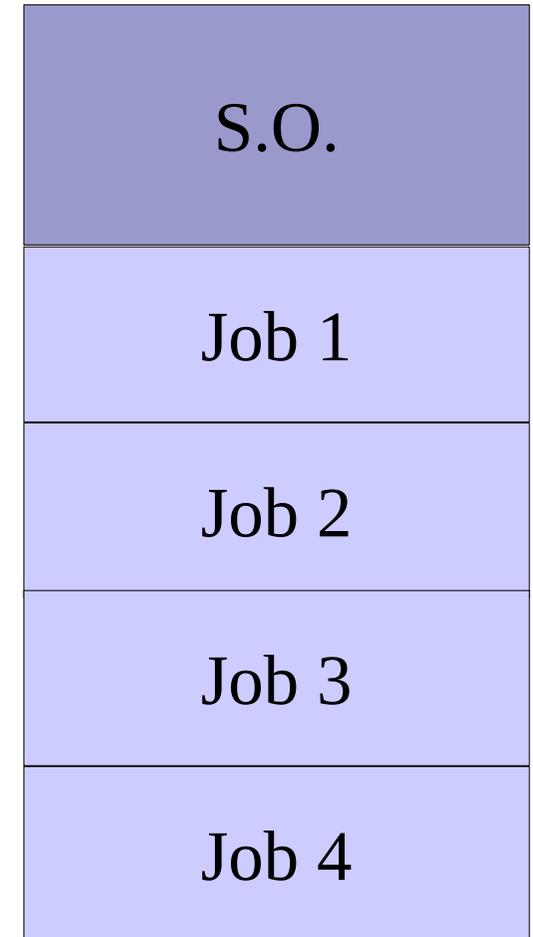
- ♦ **Come venivano costruiti?**
 - ♦ circuiti integrati
- ♦ **Come venivano usati?**
 - ♦ man mano sparisce la figura dell'operatore come "interfaccia" degli utenti verso la macchine
 - ♦ utente == operatore
- ♦ **Come venivano programmati?**
 - ♦ linguaggi ad "alto livello": C, shell scripting
 - ♦ editor testuali, editor grafici, compilatori
 - ♦ accesso al sistema da terminali
- ♦ **Quale sistemi operativi venivano usati?**
 - ♦ non più batch ma interattivi
 - ♦ multi-programmazione
 - ♦ time sharing

Generazione 3 - Multiprogrammazione

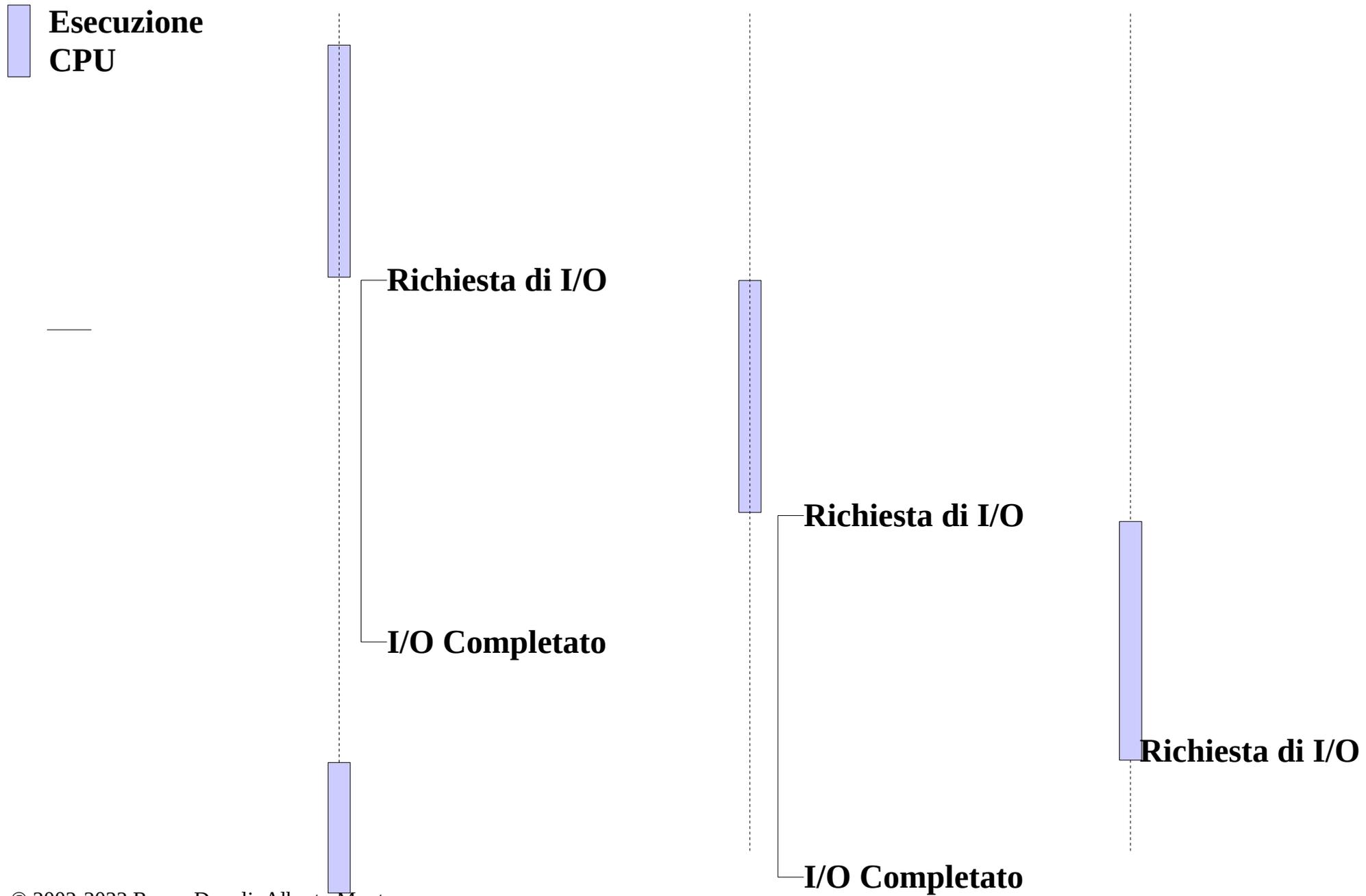
- ♦ **Definizione:** *multiprogrammazione*
 - ♦ utilizzare il processore durante i periodi di I/O di un job per eseguire altri job
- ♦ **Vantaggi**
 - ♦ il processore non viene lasciato inattivo (*idle*) durante operazioni di I/O molto lunghe
 - ♦ la memoria viene utilizzata al meglio, caricando il maggior numero di job possibili

Generazione 3 - Multiprogrammazione

- ♦ **Caratteristiche tecniche:**
 - ♦ Più job contemporaneamente in memoria
 - ♦ Una componente del S.O. detto scheduler si preoccupa di alternarli nell'uso della CPU
 - ♦ quando un job richiede un'operazione di I/O, la CPU viene assegnata ad un altro job.



Generazione 3 - Multiprogrammazione



Generazione 3 - Multiprogrammazione

- ♦ **S.O. multiprogrammati: quali caratteristiche?**
 - ♦ routine di I/O devono essere fornite dal S.O.
 - ♦ gestione della memoria
 - ♦ il sistema deve allocare la memoria per i job multipli presenti contemporaneamente
 - ♦ CPU scheduling
 - ♦ il sistema deve scegliere tra i diversi job pronti ad eseguire
 - ♦ allocazione delle risorse di I/O
 - ♦ Il sistema operativo deve essere in grado di allocare le risorse di I/O fra diversi processi

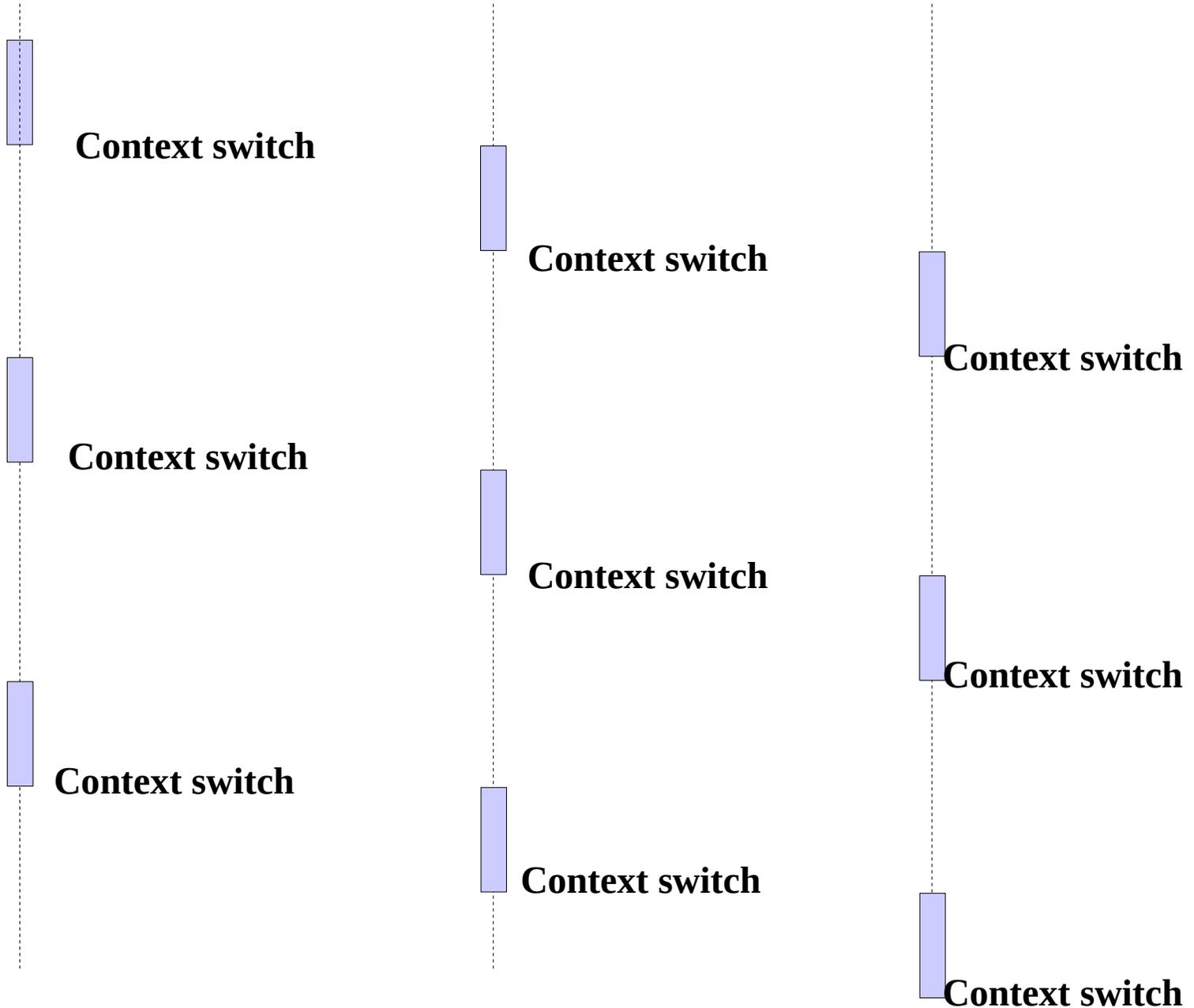
Generazione 3 - Time-sharing

- ◆ **Definizione - Time sharing**

- ◆ E' l'estensione logica della multiprogrammazione
- ◆ L'esecuzione della CPU viene suddivisa in un certo numero di quanti temporali
- ◆ Allo scadere di un quanto, il job corrente viene interrotto e l'esecuzione passa ad un altro job
 - ◆ anche in assenza di richieste di I/O
- ◆ Necessità di un dispositivo hardware (interval timer)
- ◆ I passaggi (*context switch*) avvengono così frequentemente che più utenti possono interagire con i programmi in esecuzione

Generazione 3 - Time-sharing

 **Esecuzione CPU**



Generazione 3 - Time-sharing

- ◆ **S.O. time-sharing: quali caratteristiche?**
 - ◆ *Gestione della memoria*
 - ◆ Il numero di programmi eseguiti dagli utenti può essere molto grande; si rende necessario la gestione della *memoria virtuale*
 - ◆ *CPU Scheduling*
 - ◆ Lo scheduling deve essere di tipo *preemptive* o *time-sliced*, ovvero sospendere periodicamente l'esecuzione di un programma a favore di un altro
 - ◆ *Meccanismi di protezione*
 - ◆ La presenza di più utenti rende necessari meccanismi di protezione (e.g. protezione nel file system, della memoria, etc.)

Generazione 3 - Storia

- ♦ **Compatible Time-Sharing System (CTSS) (1962)**
 - ♦ introdusse il concetto di multiprogrammazione
 - ♦ introdusse il concetto di time-sharing
 - ♦ Codice sorgente rilasciato nel settembre 2004:
<http://www.piercefuller.com/library/ctss.html?id=ctss>
- ♦ **Multics (1965)**
 - ♦ introduzione del concetto di processo
- ♦ **Unix (1970)**
 - ♦ derivato da CTSS e da Multics
 - ♦ sviluppato inizialmente ai Bell-labs su un PDP-7

Unix – Un po' di storia

- ♦ **La storia di UNIX – in breve**

- ♦ portato dal PDP-7 al PDP-11
(1^a volta che un S.O viene utilizzato in due architetture diverse)
- ♦ riscritto in linguaggio C per renderlo portabile
(anche questa una 1^a volta, visto che i S.O. venivano scritti in assembly)
- ♦ Inizialmente, veniva usato solo all'interno di Bell Labs
- ♦ Nel 1974, viene pubblicato un articolo
 - ♦ licenze proprietarie
 - ♦ licenze “libere” alle università
- ♦ Due varianti
 - ♦ Bell Labs
 - ♦ BSD (Berkeley Software Distribution, ora OpenBSD e FreeBSD)

Generazione 4 (1980-) - Personal computer

- ◆ **Ancora una frase celebre**

- ◆ Non c'è ragione per cui qualcuno possa volere un computer a casa sua
(Ken Olson, fondatore di Digital, 1977)

- ◆ **Punti chiave**

- ◆ I personal computer sono dedicati a singoli utenti
- ◆ L'obiettivo primario diventa la facilità d'uso
- ◆ Essendo dedicati a singoli utenti, i sistemi operativi per PC sono in generale più semplici
- ◆ Interfacce utente (ma NON fanno parte del S.O.)
- ◆ Grave errore: sottovalutare la sicurezza

Sistemi paralleli

- ◆ **Definizione - *Sistema parallelo***
 - ◆ un singolo elaboratore che possiede più unità di elaborazione
- ◆ **Tassonomia basata sulla struttura**
 - ◆ ***SIMD - Single Instruction, Multiple Data***
 - ◆ Le CPU eseguono all'unisono lo stesso programma su dati diversi
 - ◆ ***MIMD - Multiple Instruction, Multiple Data***
 - ◆ Le CPU eseguono programmi differenti su dati differenti
- ◆ **Tassonomia basata sulla dimensione**
 - ◆ A seconda del numero (e della potenza) dei processori si suddividono in
 - ◆ ***sistemi a basso parallelismo***
pochi processori in genere molto potenti
 - ◆ ***sistemi massicciamente paralleli***
gran numero di processori, che possono avere anche potenza non elevata

Sistemi paralleli

- ♦ sistemi *tightly* coupled
 - ♦ Bus / memoria condivisa
 - ♦ Pochi processori / basso livello di parallelismo. Bus: collo di bottiglia
- ♦ sistemi *loosely* coupled
 - ♦ Processori con memoria privata interconnessi da canali
 - ♦ Tanti processori / alto livello di parallelismo.
- ♦ Vantaggi dei sistemi paralleli
 - ♦ incremento delle prestazioni

Sistemi paralleli

- ◆ **Symmetric multiprocessing (SMP)**

- ◆ Ogni processore esegue una copia identica del sistema operativo
- ◆ Processi diversi possono essere eseguiti contemporaneamente
- ◆ Molti sistemi operativi moderni supportano SMP

- ◆ **Asymmetric multiprocessing**

- ◆ Ogni processore è assegnato ad un compito specifico; un processore master gestisce l'allocazione del lavoro ai processori slave
- ◆ Più comune in sistemi estremamente grandi

Sistemi distribuiti

- ♦ **Definizione - *Sistema distribuito***
 - ♦ Sono sistemi composti da più elaboratori indipendenti (con proprie risorse e proprio sistema operativo), collegati da una rete, appaiono come se fossero un unico sistema
- ♦ **Vantaggi dei sistemi distribuiti**
 - ♦ Condivisione di risorse
 - ♦ Suddivisione di carico, incremento delle prestazioni
 - ♦ Affidabilità
 - ♦ (latenza maggiore rispetto ai sistemi paralleli)

Sistemi distribuiti

- ◆ **Sistemi operativi**

- ◆ forniscono condivisione di file
- ◆ forniscono la possibilità di comunicare
- ◆ ogni computer opera indipendentemente dagli altri

- ◆ **Sistemi operativi distribuiti**

- ◆ minore autonomia tra i computer
- ◆ dà l'impressione che un singolo sistema operativo stia controllando tutti gli elaboratori che fanno parte del sistema distribuito

Sistemi real-time

- ◆ **Definizione: *sistemi real-time***
 - ◆ Sono i sistemi per i quali la correttezza del risultato non dipende solamente dal suo valore ma anche dall'istante nel quale il risultato viene prodotto

Sistemi real-time

- ♦ **I sistemi real-time si dividono in:**
 - ♦ *hard real-time*:
 - ♦ se il mancato rispetto dei vincoli temporali può avere effetti catastrofici
 - ♦ e.g. controllo assetto velivoli, controllo centrali nucleari, apparecchiature per terapia intensiva
 - ♦ *soft real-time*:
 - ♦ se si hanno solamente disagi o disservizi
 - ♦ e.g. programmi interattivi
- ♦ **N.B.**
 - ♦ real-time non significa necessariamente esecuzione veloce