INTRODUZIONE

Informatica → dal francese “Informatique” = Information + Automatique

↓

Insieme di tecnologie di tipo elettronico utili per creare, immagazzinare e elaborare informazioni.

↓

Informare vuol dire dare forma a qualcosa, rendere consapevole qualcuno

La minima informazione è il bit (Binary Digit) cioè l'unità di informazione necessaria per la scelta effettuata tra due soli eventi ( 0 e 1, vero e falso); in modo da diminuire l’incertezza.

**Sistema informatico VS Sistema informativo** Sistema informativo→ Ha lo scopo di produrre “in tempo utile”, nelle “forme appropriate”, ed ai “giusti livelli” le sintesi necessarie per i processi decisionali.

Sistema informatico → sottoinsieme del sistema informativo dedicato alla gestione automatica di informazioni; quindi è l’insieme di parti con l’obiettivo di archiviare, elaborare e raccogliere informazioni in maniera elettronica.

**Computer** Il computer è un elaboratore elettronico digitale:

- Elaboratore: macchina in grado di immagazzinare ed elaborare dati in base ad una serie di istruzioni;
- Elettronico: utilizza componenti elettronici per elaborare le informazioni;
- Digitale: i dati sono memorizzati mediante cifre binarie;

Il calcolatore al suo interno è diviso in:

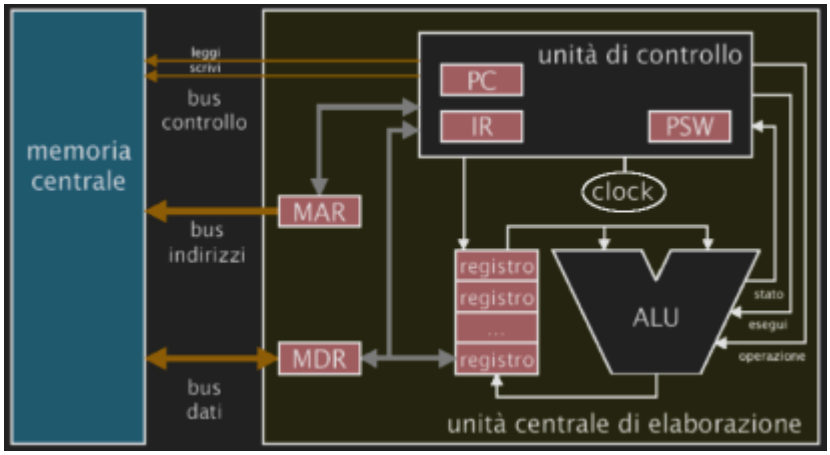
- Hardware: l’insieme di dispositivi (fisici) che compongono il calcolatore;
- Software: istruzioni e informazioni necessarie per risolvere i problemi;
- Firmware: è il confine fra le due parti precedenti, microprogrammi scritti dai costruttori o memorizzati su speciali memorie.

**Storia del calcolatore** Esistono diverse generazioni di calcolatori, ecco quali:

1. Gen 0 [calcolatori meccanici]: Il primo a costruire macchine calcolatrici automatiche fu lo scienziato francese Blaise Pascal (Pascalina), un dispositivo a ingranaggi azionati per mezzo di una manovella e in grado di effettuare addizioni e sottrazioni; aggiornata da Gottfried in modo da fare moltiplicazioni e divisioni.
2. Gen 1 [valvole termoioniche e relè]: Massachusetts Institute of Technology progettò il Differential Analyzer. Questa macchina era in grado di risolvere equazioni differenziali; il primo dei calcolatori elettrici e digitali.

Alan Mathison Turing riuscì a decrittografare i messaggi tedeschi usando un calcolatore elettromeccanico che poteva lavorare ad altissima velocità (Colossus).

Negli anni successivi alla guerra vennero sviluppati gli ENIAC (Electronic Numerical Integrator And Computer), tra i consulenti c'era Von Neumann che progettò l'EDVAC composto da registri dentro la CPU (composta a sua volta da Unità di controllo e aritmetico logica, ALU) e una memoria principale tutto collegato con bus.




Prima di ciò Von Neumann e altri progettarono IAS, denominata macchina di Von Neumann, una pietra miliare dell'informatica, che influenzò pesantemente lo sviluppo di tutti i calcolatori successivi.

In questa Gen per la priva volta si utilizzano gli 0 e 1 (accesso spento).

3. Gen 2 [i transistor]: il transistor fu inventato nel 1948 alla Bell Labs e fruttò ai suoi inventori il premio Nobel per la fisica.

Primo passo verso la miniaturizzazione che portò alla creazione di hardware meno ingombranti, più veloci e meno dispendiosi.

Anni '60 la DEC produce il PDP-8 con il BUS.

Inizia la miniaturizzazione.

4. Gen 3 [circuiti integrati]: inventati nel 1958 da Robert Noyce permisero di inserire dozzine di transistor su un singolo pezzo di silicio e quindi favorirono la costruzione di calcolatori più piccoli, più veloci e meno costosi.
5. Gen 4 [Very Large Scale Integration]: nel 1971 tre ingegneri della Intel crearono il primo microprocessore; comincia così l'era dei

PC.

Nel 1975 William Gates e Paul Allen, diedero vita a una piccolissima azienda che elaborava linguaggi per "computer": la Microsoft. Nello stesso periodo Steve

Jobs e Stephen Wozniak costruirono "Apple I".

6. Gen 5 [Quantistico & I.A.]: sviluppo dell'hardware attraverso i computer quantistici, basati non più sul concetto di bit ma di qbit e lo sviluppo del software attraverso l'implementazione delle intelligenze artificiali.

Nel prossimo futuro i calcolatori saranno sempre più veloci e capaci di autoprogrammarsi.

Esiste una "legge" che stima come i computer aumentino di potenza ogni 18 mesi, andando a raddoppiare il numero di transistor su un singolo chip; è la legge di Moore.

RAPPRESENTAZIONE DIGITALE DELL'INFORMAZIONE

L'informazione può essere rappresentata in modo:

- Digitale: Ogni dato viene codificato impiegando entità distinte individualmente e organizzate in modo opportuno;
- Analogica: Basata sull'impiego di dispositivi che realizzano una grandezza fisica che può variare in modo continuo.

**NUMERI A PRECISIONE FINITA** Rappresentati con un numero finito di cifre, le operazioni con questi numeri possono causare i seguenti errori se il loro risultato non appartiene all'insieme dei valori rappresentabili:

- Underflow: il risultato è minore del più piccolo valore rappresentabile;
- Overflow: il risultato è maggiore del più grande valore rappresentabile;
- Non appartenenza all'insieme: il risultato non appartiene all'insieme dei valori rappresentabili.

L'algebra dei numeri a precisione finita è diversa da quella convenzionale, la proprietà distributiva e la proprietà associativa non sono rispettate.

**NUMERI FLOATING-POINT** Per il trattamento di valori razionali o reali si adotta una notazione in cui la gamma dei valori esprimibili è indipendente dal numero di cifre significative. Questo sistema è detto floating-point. Esprimibile nel seguente modo:

$$n = f * 10^{e} \quad \rightarrow \textit{esponente}$$

$$\downarrow$$

mantissa

La precisione è determinata dalla mantissa f, mentre la gamma dei valori è determinato dall'esponente e.

Uno standard permette di valorizzare la mantissa (f) solo in un range fra 0.1 e 1 ($0.1 \leq f < 1$).

$3.14 = 31.4 * 10^{-1}$ sbagliato f > 1

$3.14 = 0.314 * 10^{1}$ corretto

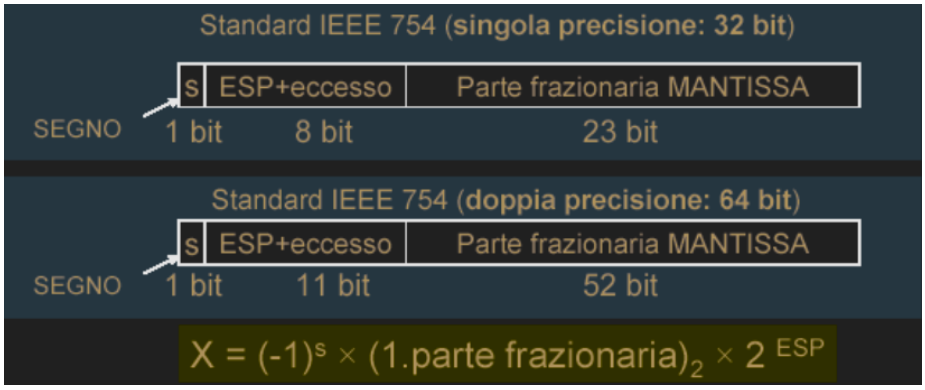
$0.1 \leq f < 1$

**IEEE 754** Con l'introduzione del seguente standard si sono uniti tutti i calcolatori su un unico formato per i floating-point binari. Ne esistono 3 formati:

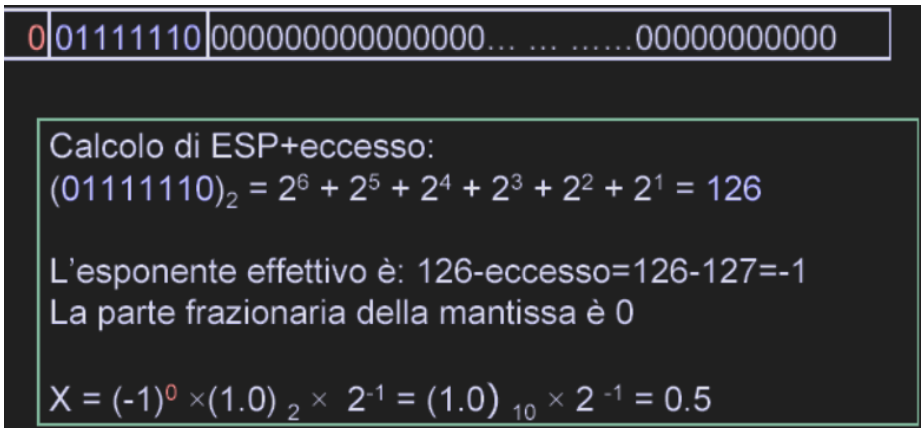
1. Singola precisione: Totale 32 bit [1 per il segno, 8 per gli esponenti, 23 mantissa];
2. Doppia precisione: Totale 64 bit [1 per il segno, 11 per gli esponenti, 52 mantissa];
3. Precisione estesa;

Per calcolare l'eccesso si usa la formula $2^{(n-1)} - 1$, con n = numero di bit.

Per calcolare in numero decimale avente il numero binario floating point si fa così:



**Es:**




**Normalizzazione** La mantissa binaria normalizzata deve presentare un 1 a sinistra della virgola binaria. L'esponente deve essere aggiustato di conseguenza.

Esempio di normalizzazione:

$4568.1875_{10}$

cambiamo la base

1000111011000.0011

normalizzo spostando la virgola di tot posizioni in modo da avere un 1. davanti

1.0001110110000011 * $2^{12}$ [12 sono le pos. spostate]

esprimo l'esponente in eccesso (127)

12 + 127 = 139 → 10001011 [il 127 è dato da $2^{n-1} - 1$ dove n = num. bit esponente]

rappresento il tutto in modo corretto

0 1 0 0 0 1 0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0

↓ ↓ ↓

segno esponente mantissa normalizzata

**SISTEMI DI NUMERAZIONE INFORMATICA** I sistemi di numerazione più utilizzati in informatica:

- Sistema decimale (base=10) 0 1 2 3 4 5 6 7 8 9 ;
- Sistema binario (base=2) 0 1: ogni cifra, detta bit (Binary digIT), può essere rappresentata direttamente tramite un livello elettrico di tensione ;
- Sistema ottale (base=8) 0 1 2 3 4 5 6 7;
- Sistema esadecimale (base=16) 0 1 2 3 4 5 6 7 8 9 A B C D E F: è utilizzato poiché è molto compatto e ben si presta alla traduzione in valori binari, poiché ogni cifra corrisponde esattamente a 4 cifre binarie.

Tutti questi sistemi sono a notazione posizionale, i sistemi di numerazione posizionale associano alle cifre un diverso valore in base alla posizione che occupano nella stringa che compone il numero.

**CONVERSIONI TRA BASI** Tabella utile per le conversioni:

| BINARIO | OTTALE/ESADECIMALE |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | A |
| 1011 | B |
| 1100 | C |
| 1101 | D |
| 1110 | E |
| 1111 | F |

**Binario → Decimale** Per convertire qualsiasi numero binario in uno decimale basta moltiplicare il numero per la base elevata alla posizione.

Es: 1 0 0 1 1 0 0 1

7 6 5 4 3 2 1 0

$$1*2^7 + 1*2^4 + 1*2^3 + 1*2^0 = 128 + 16 + 8 + 1 = 153_{10}$$

Per convertire una parte frazionaria (Es: 0.101) usiamo lo stesso ragionamento ma con i numeri ad esponente negativo:

Es: 0.1 0 1

-1 -2 -3

$$1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 0{,}5 + 0 + 0{,}125 = 0{,}625_{10}$$

**Ottale → Decimale** Come prima prendiamo il numero e moltiplichiamo ogni cifra per la base elevata alla posizione:

Es: $32_8 \rightarrow 3*8^1 + 2*8^0 = 24 + 2 = 26_{10}$

**Esadecimale → Decimale** Come prima prendiamo il numero e moltiplichiamo ogni cifra per la base elevata alla posizione, in questo caso le cifre numeriche si fermano al 9, dal 10 al 15 le cifre diventano lettere (dalla A alla F):

Es: $1A_{16} \rightarrow 1 * 16^1 + 10 * 16^0 = 16 + 10 = 26_{10}$

**Binario → Ottale** Convertiamo le cifre binarie a gruppi di 3:

Es. 1 0 1 1 1 1 0 → ( 001) (011) (110)

↓ ↓ ↓

1 3 6 = $136_8$

**Ottale → Binario** Procedimento inverso rispetto al precedente:

Es: $136_8$ = ( 001) (011) (110) → 1 0 1 1 1 1 0

**Esadecimale → Binario** Convertiamo le cifre esadecimali a gruppi di 4:

Es: $7BA3_{16}$ = (0111) (1011) (1010) (0011)

↓ ↓ ↓ ↓

7 B A 3

**Decimale → Binario** Esistono due metodi per convertire da base 10 a base 2, ecco quali:

- Metodo 1 [Generale]:

Si procede sottraendo la più grande potenza di 2 abbastanza grande da contenere il numero da convertire, andando poi a mettere 1 nelle posizioni dell'esponente.

Per rendere più veloce il procedimento possiamo fare somme con i risultati delle potenze di 2, Es:

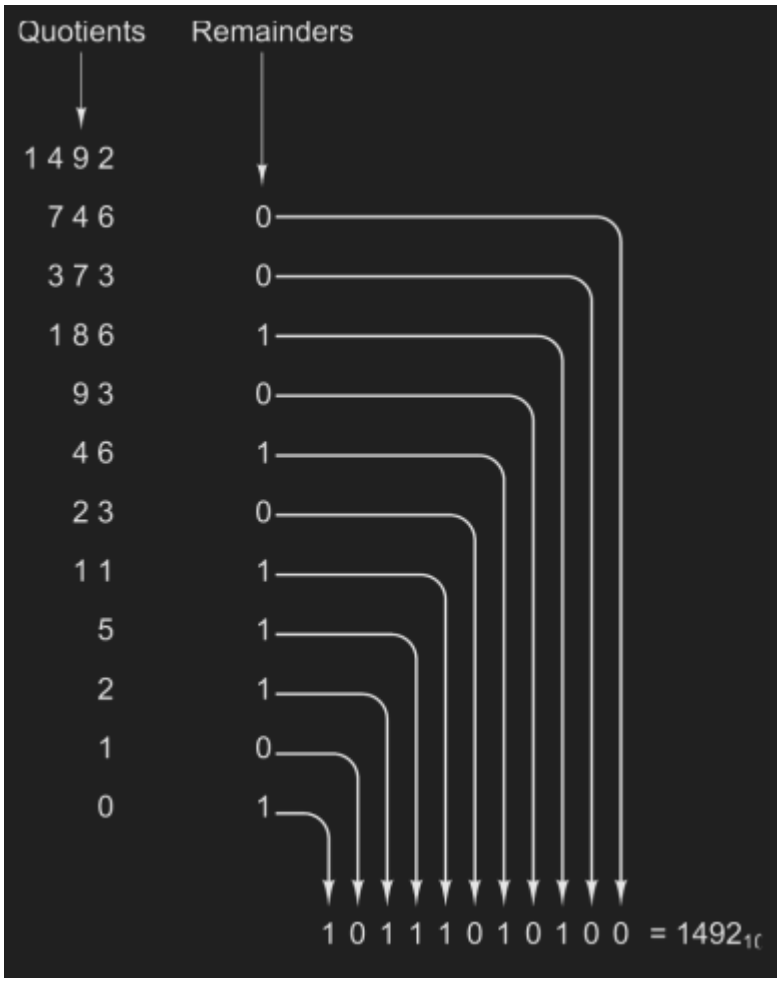
138 = 128 + 8 + 2

↓ ↓ ↓

$2^7$ $2^3$ $2^1$ = 10001010

- Metodo 2 [Divisione]:

In questo metodo andiamo a dividere per 2 il numero e contando se il resto è 0 o 1, alla fine delle divisioni prendiamo i resti in modo ricorsivo. Es:




- Conversione parte frazionaria:

Per convertire la parte a destra della virgola in un numero binario, dopo aver convertito normalmente la parte intera, dobbiamo moltiplicare la parte frazionaria per 2, se il risultato è > di 1 allora sottraiamo 1 per poi andare a moltiplicare di nuovo e ripetere i passaggi fino a quando non otteniamo come risultato 1 oppure un risultato già ottenuto. Esempio:

0,6

↓

0,6 * 2 = 1,2 (>1)

↓

1,2 -1 = 0,2

↓

0,2 * 2 = 0,4 (<1 niente sottrazione)

↓

0,4 * 2 = 0,8 (<1 niente sottrazione)

↓

0,8 * 2 = 1,6 (>1)

↓

1,6 - 1 = 0,6 (risultato già ottenuto, finiamo qui)

Ora per prendere il numero binario andiamo semplicemente a prendere i numeri prima della virgola nei risultati in maniera non ricorsiva !!!

Quindi il risultato sarà:

0,6 = 0.1001

(non prendo i risultati delle sottrazioni)

**ARITMETICA DEL CALCOLATORE** Diversa da quella comunemente utilizzata dalle persone; la precisione con cui i numeri possono essere espressi è finita.

Per rappresentare i numeri si utilizza il sistema binario.

La più importante unità di misura dell'informazione manipolata dal calcolatore è il BYTE, composto da 8 bit; sequenze di bit più lunghe di un byte sono denominate WORD.

**Somma** Il procedimento di somma binaria è equivalente a quello nel sistema decimale con eccezione del riporto che si genera quando entrambi gli addendi hanno valore 1. Es:

0 0 1 1 +

0 1 0 1 =

---

0 1 1 0 risultato

0 0 0 1 riporto

- La somma è uguale a 1 solo se la coppia di cifre è diversa.
- Il riporto è uguale a 1 solo se entrambe le cifre sono uno.
- In tutti gli altri casi produce 0.

L'ultimo riporto viene messo nel risultato

**Numeri binari negativi** Prima di vedere la sottrazione bisogna saper negare un numero, esistono 4 metodi:

1. Grandezza a segno: prevede l'utilizzo del bit più a sinistra per indicare il segno: 0 il +, 1 il -.

Es:

76 -> 01001100 -76 -> 11001100

2. Complemento a uno: andiamo a negare il numero, invertendo lo 0 con l'1 e viceversa.

Es:

-(76) -> -(01001100) -> 10110011

3. Complemento a due: prendiamo il numero negato e aggiungiamo 1.

Es:

-(76) -> -(01001100) -> 10110011 -> 10110100

4. Eccesso $2^{m-1}$ : Rappresenta i numeri come somma di se stessi con 2m-1 dove m è il numero di bit utilizzati per rappresentare il valore.

Es [con m=8 quindi il num. è memorizzato in un byte]:

$2^{m-1} = 2^7 = 128$ / -76 -> -76+128 = 52 = 00110100

**Sottrazione** Con i numeri binari si preferisce sommare un numero positivo ad uno negativo piuttosto che fare una sottrazione.

Si parla quindi di somma tra numeri binari in complemento a due, andando a utilizzare questo metodo:

- Il riporto generato dai bit più a sinistra viene ignorato.
- Se gli addendi sono di segno opposto non si può verificare un overflow.
- L'overflow si verifica se il riporto generato nel sommare i bit di segno è diverso dal riporto utilizzato per sommare i bit di segno.

Es:

Addendo 10 0 0 0 0 1 0 1 0

Addendo -3 1 1 1 1 1 1 0 1

---

Somma 7 0 0 0 0 0 1 1 1

Riporti 1 1 1 1 1 0 0 0

↓

riporto ignorato

**Prodotto** Il procedimento del prodotto viene riportato all'addizione in questo modo:

- Ogni cifra 1 del moltiplicatore fa scorrere a sinistra le cifre del moltiplicando, aggiungendo tanti 0 quanti ne richiede la sua posizione.
- I numeri così ottenuti vengono memorizzati e sommati tra di loro.

Es: 3 2 1 ← posizioni

70*14 = 01000110 * 00001110

↓

dovrò sommare il 70

per se stesso 3 volte (num di 1 = 3) e

ogni volta aggiungere tanti 0 in base alla posizione degli 1

1 0 0 0 1 1 0 0 +

1 0 0 0 1 1 0 0 0 =

---

1 1 0 1 0 0 1 0 0 +

1 0 0 0 1 1 0 0 0 0 =

---

$1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0 = 980_{10}$

CODIFICA DELL'INFORMAZIONE

**CODIFICA DEI CARATTERI** Ogni calcolatore dispone di un set di caratteri codificati sotto forma di numeri. La mappatura di essi è detta "tabella dei caratteri".

**ASCII** Utilizza 7 bit (l'8° usato per il controllo) con 128 combinazioni. I caratteri da 0 a 1F sono detti di controllo, non vengono stampati.

All'ASCII sono state aggiunte diverse estensioni per aggiungere caratteri non rappresentabili in precedenza, come ad esempio il Latin-1 con 8 bit con l'aggiunta delle lettere latine, accenti, ecc.

**UNICODE** Visto che l'ASCII non poteva rappresentare tutti i caratteri dei vari alfabeti si è scelto di creare un unico set per tutto il mondo, l'Unicode con 2 byte (16 bit) quindi da 0 65000 circa combinazioni.

**UTF-8** il Consorzio UNICODE ha sviluppato lo standard UTF-8 che prevede una rappresentazione dinamica dei caratteri da 1 a 4 byte per risolvere i problemi di compatibilità online fra mittente e destinatario.

**CODIFICA DELLE IMMAGINI** Le immagini si dividono in rester (scalari) e vettoriali (geometriche).

**Raster** In questa tipologia l'immagine è un insieme di parti distinte che possono essere codificate separatamente con sequenze di bit. Queste parti vanno discretizzate (scomposizione dell'immagine in un reticolo di punti) e quantizzate (codifica di ogni pixel con una sequenza di bit).

L'immagine è un insieme di pixel dove ognuno rappresenta un colore che insieme ad altri vanno a formare un'immagine.

La discretizzazione prevede:

- Definizione: DPI (dot per inches) cioè il numero di pixel;
- Risoluzione: dimensione griglia.

La Quantizzazione prevede la rappresentazione di ogni pixel con una sequenza di bit, tale rappresentazione è nota come codifica bitmap.

**Vettoriali** Si affidano a equazioni matematiche per disegnare le immagini, per questo è flessibile, le immagini possono essere ingrandite o rimpicciolite senza compromettere la qualità.

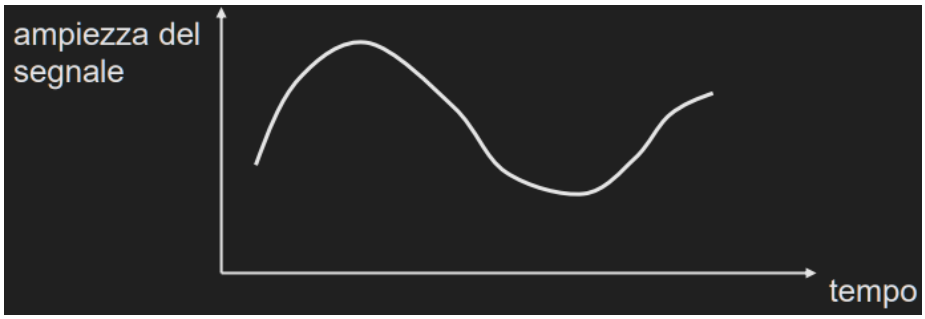
Si possono convertire in jpg e png ma non viceversa.

**Compressione** Per risparmiare sulla dimensione si:

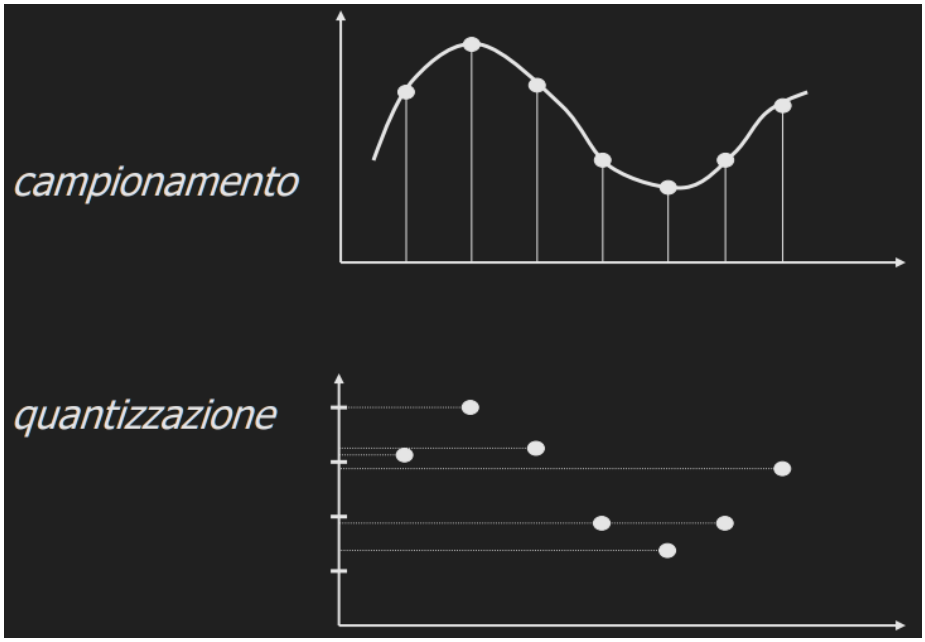
- usa una tavolozza (palette) per contenere i sottoinsiemi di colori rappresentabili;
- usano tecniche (TIFF) di compressione cercando di raggruppare le aree che hanno caratteristiche comuni;
- usano tecniche (GIF, JPG) di compressione che sfruttano la caratteristica dell'occhio umano di essere poco sensibile a lievi cambiamenti di colore in punti contigui, e quindi eliminano questi lievi cambiamenti appiattendo il colore dell'immagine.

**CODIFICA DEI SUONI** Per rappresentare le onde sonore che rappresentano il suono si usa la quantizzazione e il campionamento:

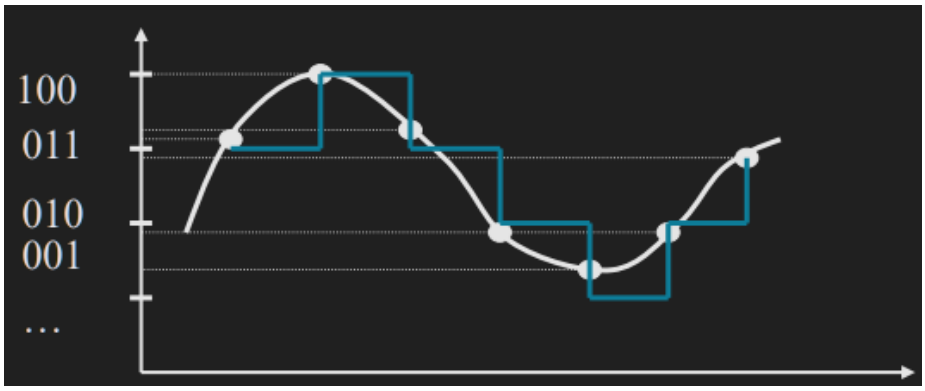
- campionamento: scelta di istanti in cui considerare il valore del segnale;
- quantizzazione: codifica dei campioni con un numero predefinito di bit.




Questo è il suono in maniera analogica.




Attraverso le tecniche precedentemente dette andiamo a individuare diversi punti nel dominio del tempo e dell'ampiezza.




Alla fine codifichiamo con una sequenza di bit il suono.

Per avere un segnale audio più simile all'originale dobbiamo avere intervalli di campionamento più piccoli (in modo da perdere meno segnale in mezzo a essi) e avere abbastanza bit nella fase di quantizzazione per descrivere il suono; ovviamente maggiore accuratezza significa maggior quantità di memoria occupata.

ALGEBRA BOOLEANA

È un sistema di logica matematica a due stati, dove le variabili possono assumere solo due stati: vero (1) o falso (0).

Si studia l'algebra booleana poiché le funzioni dell'algebra booleana sono isomorfe ai circuiti digitali: un circuito digitale può essere espresso tramite un'espressione booleana e viceversa.

Una funzione booleana ha una o più variabili in input e fornisce risultati che dipendono solo da queste variabili. Quindi una funzione booleana con n variabili in input avrà solo $2^n$ combinazioni possibili.

Qualsiasi funzione può essere espressa con l'ausilio di AND, OR e NOT detto ciò è possibile tradurre qualsiasi espressione in un circuito.

Qui i significati degli operatori:

| | | |
|---|---|---|
| AND | prodotto logico | =1 solo se tutti 1 |
| OR | somma logica | =1 se almeno un 1 |
| NOT | valore opposto | se A=1 allora $\bar{A}$=0 |

**Teoremi** Esistono proprietà per semplificare le espressioni booleane, ecco quali:

| Nome legge | Forma AND | Forma OR |
|---|---|---|
| Identità | 1A = A | 0 + A = A |

| | | |
|---|---|---|
| Annullamento | $0A = 0$ | $1 + A = 1$ |
| Impotenza | $AA = A$ | $A + A = A$ |
| Inverso | $A\bar{A} = 0$ | $A + \bar{A} = 1$ |
| Commutativa | $AB = BA$ | $A + B = B + A$ |
| Associativa | $(AB)C = A(BC)$ | $(A + B) + C = A + (B + C)$ |
| Distributiva | $A + BC = (A + B)(A + C)$ | $A(B + C) = AB + AC$ |
| Assorbimento | $A(A + B) = A$ | $A + AB = A$ |
| De Morgan | $\overline{AB} = \bar{A} + \bar{B}$ | $\overline{A + B} = \bar{A}\bar{B}$ |
| Assorbimento 2° | $A + \bar{A}B = A + B$ | \\ |

Per verificare le leggi e teoremi bisogna creare tabelle di verità, es di dimostrazione della 2° legge dell'assorbimento:

| A | B | $\bar{A}$ AND B | $\bar{A}$ AND B + A | A + B |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |

↑ ↑

Essendo queste due parti uguali il teorema è verificato

**Tabelle di verità** Con l'utilizzo delle tabelle di verità posso arrivare a una espressione andando a individuare quali combinazioni con output 1 rappresentano quel risultato.

Questo tipo di espressioni si chiamano in prima forma canonica, per arrivare a questa forma è necessario:

1. Identificare le righe in output (V) con risultato 1;
2. in queste righe scrivere la configurazione delle variabili che le definiscono (saranno in AND tra loro);
3. collegare tutte le configurazioni con l'OR.

Es:

| X | Y | Z | V |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Le righe evidenziate sono quelle con risultato (V) 1, quindi quelle che ci interessano. Ora andiamo a scrivere le loro configurazioni:

$$\overline{X}\,\overline{Y}Z$$

Riga 1 →

$$\overline{X}Y\overline{Z}$$

Riga 2 → $$\overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + XYZ$$

Risultato finale → $$X\overline{Y}\,\overline{Z}$$

Riga 4 →

$$XYZ$$

Riga 7 →

Due funzioni con ugual output si dicono equivalenti, andare a determinare la più semplice funzione booleana equivalente alla funzione data facilita l'interpretazione della funzione stessa e permette di semplificare anche i circuiti logici corrispondenti.

ARCHITETTURA DEI CALCOLATORI

Un calcolatore è un sistema composto da un numero elevato di componenti, il modello base è rappresentato dalla macchina di Von Neumann, composto da:

- CPU;
- Memoria;
- Periferiche;
- Bus.

**BUS** Gestito dalla CPU (master) e nessun'altra unità (slave) può accedervi autonomamente. Permette la comunicazione tra periferiche, CPU, cache, memoria, ecc.

I Bus possono essere:

- bus dati: costituiti da $2^n$ linee fisiche, passano i dati I/O della CPU;
- bus degli indirizzi: determina l'indirizzo di memoria dove si andrà a leggere o scrivere;
- bus di controllo: indica cosa fare nell'indirizzo di memoria ( andare a leggere o scrivere).

**SCHEDA MADRE** Contiene l'alloggiamento CPU, i bus, i connettori di memoria, i connettori I/O, ecc.

La scheda madre controlla un insieme di chip detti chipset.

Dentro la Motherboard sono presenti dei connettori PCI (Peripheral Component Interconnect) dove è possibile posizionare schede specifiche come la GPU o l'Ethernet card.

**CPU** Il cuore operativo del calcolatore, può essere chiamato anche processore o microprocessore, con il compito di eseguire i programmi presenti in RAM andando a leggere le loro istruzioni in sequenza.

È composta da:

- Unità di controllo: legge le istruzioni in RAM;
- ALU: esegue le operazioni (AND, OR, ...) necessarie per l'esecuzione;
- Registri: sono celle di memoria molto piccole dove si salvano temporaneamente i risultati delle operazioni, in oltre sono proprio i registri che determinano il tipo della CPU (32 o 64 bit).

Al momento non si va oltre i 64 bit per via dei problemi di retrocompatibilità che ci potrebbero essere con i programmi scritti in 128 bit.

I registri di dividono in:

- PC (Program Counter): Conserva l'indirizzo di memoria della prossima istruzione da seguire;
- IR (Instruction Register) contiene copia della codifica dell'istruzione da eseguire;
- MAR (Memory Address Register): contiene l'indirizzo di memoria dove bisogna leggere/scrivere.
- MDR (Memory Data Register): contiene l'informazione appena letta in memoria;
- PSW (Processor Status Word): indica in che stato è il programma.

La CPU lavora in modo ciclico andando a ripetere le operazioni qui sotto elencate fino alla fine dell'esecuzione:

- Caricamento (Fetch): acquisizione dalla memoria di un'istruzione del programma;

- Decodifica (Decode): identificazione del tipo di operazione da eseguire;
- Esecuzione (Execute): effettuazione delle operazioni corrispondenti all'istruzione.

Il ciclo di esecuzione con le seguenti operazioni è:

1. Segnale di avvio: il PC viene inizializzato con l'indirizzo della prima istruzione da eseguire [ fatto una volta sola ];
2. la CPU imposta MAR ← PC;
3. viene selezionata la cella da leggere, il contenuto viene inviato sul bus;
4. la CPU imposta IR ← MAR;
5. la CPU incrementa il PC in modo che punti alla prossima istruzione;
6. la CPU esamina il IR e determina le operazioni da svolgere;
7. tutte le unità interessate all'esecuzione vengono comandate;
8. al fine del ciclo la CPU torna al punto 2.

I colori dei numeri rappresentano le fasi di Fetch, Decode e Execute.

**Linguaggio macchina e assembly** Il linguaggio macchina è un linguaggio comprensibile direttamente dal processore della macchina.

L'assembler invece è la versione simbolica del linguaggio macchina in cui i nomi delle operazioni e degli operandi sono indicati con codici simbolici; è usato per parlare direttamente con la macchina.

Esempio programma assembler e traduzione in binario:

X = Y + 2 → LOAD Y, R1 carica Y in R (dove R è un indirizzo interno al processore [cache])

ADD 2, R1 aggiunge 2 al contenuto di R1

STORE R1, X scrive il contenuto di R1 in X (dove X, così come Y, è un indirizzo di memoria)

Ogni operazione viene codificata (che è sempre un Word o multipli) diversamente in base al formato stabilito dal costruttore dell'hardware, esempio:

| Codice operativo | Modo 1 | Op1 | Modo 2 | Op2 |
|---|---|---|---|---|

Dove:

1. Il codice operativo [ 4 bit ]: indica la codifica dell'operazione da fare (es: ADD 0001, LOAD 0110, ecc);
2. Il Modo 1/Modo 2 [ 2 bit ]: indica a cosa si riferisce Op1/Op2 (es: se è 00 allora Op1/Op2 è un numero di registro o 01 allora Op1/Op2 è un indirizzo, ecc);
3. I'Op1/Op2 [ 12 bit ]: è quel qualcosa indicato dal suo modo.

Quindi ora la traduzione del programma precedente in linguaggio macchina sarà:

Modi:

- 00 = registro;
- 01 = memoria;
- 10 = immediato (è un dato).

Cod. operativi:

- ADD = 0001;
- LOAD = 0110;
- STORE = 0111.

Traduzione:

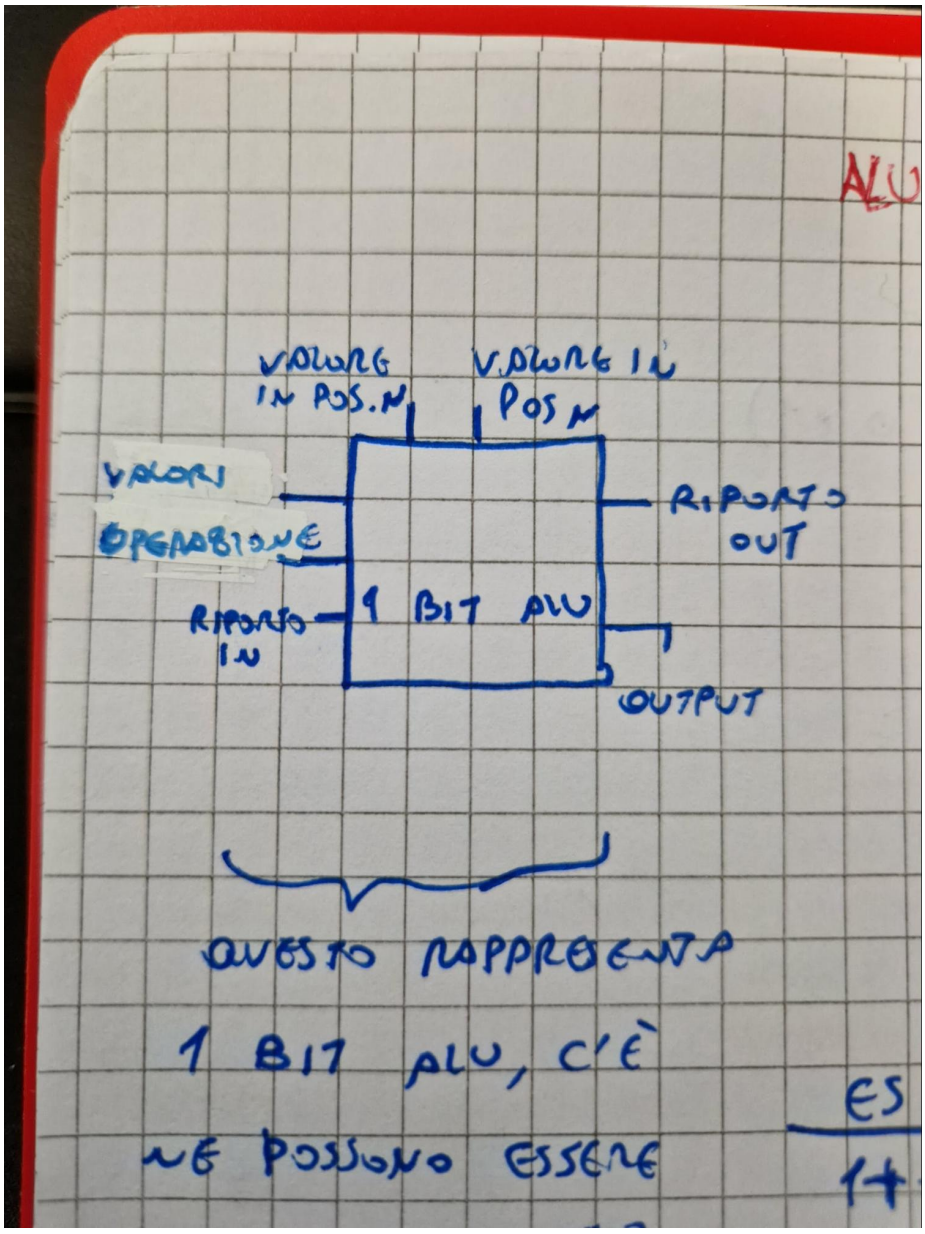
| | | | | |
|---|---|---|---|---|
| 0110 | 01 | 01101 | 00 | 00001 [*] |
| 0001 | 10 | 00010 | 00 | 00001 |
| 0111 | 00 | 00001 | 01 | 11100 |

Ogni riga della tabella equivale ad una riga istruzione del programma precedente.

[*] essendo questo R1 per tradurlo in binario prendiamo solo il numero (1 in questo caso).

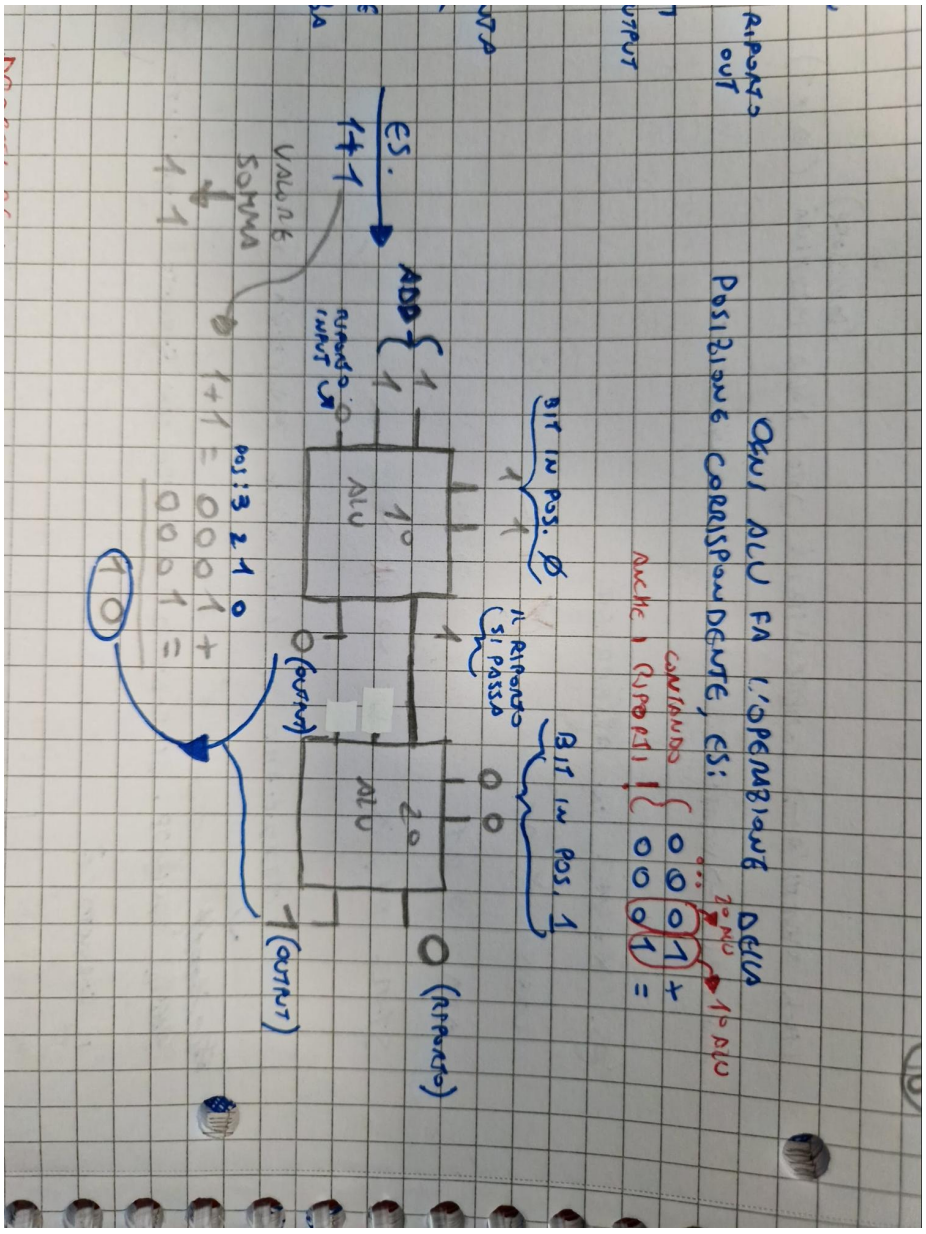
**ALU** Serve per effettuare operazioni di tipo aritmetico e logico booleano, un bit di

ALU è così rappresentato:




per rappresentare operazioni con più bit ne metto uno dietro l'altra.

Es:




Qui possiamo vedere una somma 01+01, ogni ALU fa l'operazione della posizione corrispondente, in questo caso la 1° ALU farà l'operazione fra i numeri della posizione 0 (1+1) invece la 2° quelli della posizione 1 (0+0).

**PROCESSORE**

**Clock e GHz** L'esecuzione di ogni processo viene misurata in clock, cioè la velocità del processore di eseguire operazioni.

L'esecuzione è sincrona, quindi prima di fare l'operazione successiva è necessario finire la precedente. Alla fine di ogni operazione c'è un tick, misurato in Hz (tick al secondo), che scandisce le varie istruzioni.

**CISC vs RISC** Nel mondo informatico esistono 2 correnti di pensiero sull'utilizzo della CPU:

- CISC: in questo caso si decide di dare al processore più operazioni possibili alla CPU anche a scapito di avere più cicli;
- RISC: in questo caso si decide che per ogni ciclo viene eseguita una e una sola istruzione, quindi ci saranno più istruzioni.

Nel RISC tutte le istruzioni e le istruzioni macchina vengono eseguite direttamente dall'hardware e viene utilizzato il pipelining e parallelismo.

**Pipelining** L'esecuzione di ogni istruzione viene suddivisa in più fasi (stadi) dove ognuna delle quali viene gestita da un hardware dedicato.

Per far in modo che questo accada bisogna garantire che le istruzioni non siano in conflitto (non devono dipendere l'una dall'altra).

**Parallelismo** Ne esistono di 2 tipi:

1. a livello di istruzione: più istruzioni vengono eseguite contemporaneamente all'interno della stessa CPU tramite tecniche di pipelining e processori superscalari;
2. a livello di processore: più CPU cooperano per la soluzione dello stesso problema, andando però ad abbandonare il modello Neumann.

**Prestazioni** Per calcolare il tempo di esecuzione abbiamo bisogno della seguente formula:

$$T_{esecuzione} = T_{clock} * (\sum_{i=1}^{n} N_i * CPI_I)$$

dove:

- Tclock : periodo di clock della macchina;
- $CPI_I$ : numero di clock per ogni istruzione di tipo i;
- $N_i$: numero di istruzioni di tipo i;

**MEMORIE** È la capacità di memorizzare le informazioni del calcolatore, deve essere veloce, capiente e persistente; per conciliare le diverse caratteristiche esistono due categorie:

- memoria centrale ( veloce ma non persistente, contiene i programmi in esecuzione);
- memoria di massa ( più lenta ma capiente e persistente).

**Tecnologie e struttura** Esistono diverse tecnologie usate ora e in passato per creare le memorie:

- elettroniche [ mem. centrale ]: veloci ma sempre alimentate, costo per bit elevato;
- magnetiche [ mem. massa ]: basso costo per bit ma non volatili, sono lente;
- ottiche: riscrivibili e utili alla grossa distribuzione.

La memoria centrale è strutturata in celle, successione ordinata di elementi binari raggruppati in unità minime, e ogni cella ha un proprio indirizzo ( si parte dallo 0).

Con K bit posso avere $2^K$ celle.

**Tipo di accesso** Esistono quattro modi per accedere ad una cella di memoria, ogni modo dipende dal tipo di memoria:

1. accesso sequenziale [ nastri ]: prima di poter arrivare alla cella interessata devo leggere le celle precedenti;
2. accesso casuale [ casuale ]: il tempo di accesso è indipendente dalla posizione e costante per tutte le celle;
3. accesso misto [ dischi ]: si accede in prossimità del dato e si procede in modo sequenziale;
4. accesso associativo [ cache ]: le celle si selezionano in base al tipo di dato contenuto, si usa per permettere alla CPU di avere subito i dati di un determinato processo.

Si può accedere alla memoria centrale mediante il registro MAR e si prelevano/scrivono dati tramite l'MDR in base al segnale di lettura/scrittura definito dal bus di controllo.

**RAM vs ROM vs Flash** Memoria RAM (Random Access Memory): ad accesso casuale realizzata mediante circuiti a semiconduttori, di tipo volatile;

Memoria ROM (Read-Only Memory): non volatile, scritta al momento della produzione e che non può essere cancellata. Contiene il BIOS;

Memorie FLASH: elettroniche non volatili ma riscrivibili.

**Tipi di RAM** I chip di memoria non vengono venduti singolarmente ma sono normalmente organizzati su schede stampate. I principali formati sono:

- SIMM (Single Inline Memory Module) : utilizza moduli DRAM quindi immagazzina ogni bit in un diverso condensatore (acceso 0 altrimenti 1), se il condensatore perde la carica, l'informazione è perduta;
- DIMM (Dual Inline Memory Module): utilizzano SDRAM e i connettori si trovano su entrambe i lati della scheda;
- SO-DIMM (Small Outline Dual In-line Memory Module): sono semplicemente più piccoli e costano di meno.

Attualmente sul mercato esistono due tipi distinti di RAM: quelle di tipo SDR (Single Data Rate) e quelle DDR (Double Data Rate). Quest'ultime (DDR) possono trasferire un byte sia sul fronte di salita che su quello di discesa del clock (lavorano il doppio rispetto alle SDR che fanno un'operazione a colpo di clock).

Le DDR sono arrivate alle Gen 5 e non sono retrocompatibili.

**Tipi di ROM** Le ROM esistenti sono:

- ROM (Read Only Memory): uniche veramente di sola lettura;
- PROM (Programmable Read Only Memory): programmabili una sola volta;
- EPROM (Erasable Programmable Read Only Memory): riprogrammate con apparecchiature speciali;
- EEPROM (Electrically Erasable Programmable Read Only Memory).

**Cache** Per ovviare al problema di aspettare che l'informazione arrivi dalla memoria alla CPU si è creata una memoria super veloce vicina al processore, denominata cache.

In questa memoria si caricano tutte quelle informazioni che la CPU avrà bisogno per un determinato processo andando a vedere qual è l'ultima cella utilizzata.

Al momento ne esistono 3 livelli:

1. L1 – Cache di livello 1, fa fisicamente parte del chip del microprocessore. Molto piccola e molto veloce;
2. L2 – Cache di livello 2, prima sulla scheda madre, adesso è nel core;
3. L3 – Cache di livello 3, è completamente condivisa tra i core dell'elaboratore.

**Memorie di massa** Nastri

Le prime memorie di massa create furono i nastri magnetici, permettono solo accesso sequenziale ai dati e i dati vengono scritti trasversalmente rispetto alla direzione.

Floppy Disk

Il Floppy disk è stato il primo supporto magnetico con accesso diretto ai dati. I floppy disk possono memorizzare fino a 1,44 MB, il funzionamento è simile a quello dell'hard disk.

Dischi magnetici (HDD)

Dispositivo elettromeccanico per la conservazione di informazioni sotto forma magnetica, su supporto rotante a forma di piatto su cui agiscono delle testine di lettura/scrittura.

La testina contenente un induttore ed è sospesa sopra la superficie e viene sostenuta da un cuscino d'aria.

È possibile partizionare il disco per dividere un unico HDD in più parti logiche.

Un hard disk è strutturato per registrare dati su cilindri, tracce (sequenza circolare di bit) e settori (composto da preambolo, dati e codice errore).

Per scrivere e leggere si usano due testine diverse, in scrittura quando la corrente negativa o positiva passa attraverso la testina, viene magnetizzata la superficie sotto la testina, invece in lettura quando una testina passa sopra un'area magnetizzata viene indotta una corrente positiva o negativa nella testina: questo permette di rileggere i bit memorizzati precedentemente.

La capacità di memorizzazione dipende principalmente dalla densità di registrazione, quindi quante informazioni possono essere immagazzinate in una determinata area.

Le prestazioni degli hard disk dipendono da:

- Seek time: è il tempo necessario per spostare le testine sul cilindro desiderato;
- Latency time: il tempo necessario affinché il settore interessato passi sotto la testina (definito dalla velocità di rotazione del disco).

Dischi a stato solido (SDD)

Non hanno veramente dei dischi fisici ma sono basati su componenti elettronici composti da chip NAND, cioè insieme di celle che includono un transistor MOSFET capace di memorizzare in una sola cella più combinazioni di 0 e 1 (e non solo true/false, 0/1) questo è possibile guardando la tensione elettrica data ad una cella, per questo sono molto veloci ma la loro capacità di tenere informazioni scende con il tempo.

Le memorie sono classificate da quanti valori di bit possono essere conservati, alcuni esempi sono:

- SLC (single level cell, un bit): un solo bit per cella (0/1);
- MLC (multi-level cell, due bit): 2 bit per cella (00/01/10/11);
- TLC (triple-level cell, tre bit): 3 bit per cella (000/001/010/011/100/...);
- QLC (quad-level cell, quattro bit) 4 bit per cella (0000/0001/0010/0011/0100/...).

Più si aumentano le combinazioni più il range di Volt necessari per rappresentare i valori si abbassa e quindi è più possibile, con il passare del tempo, che i valori diventino meno precisi.

Dischi ottici (CD)

Informazioni codificate per mezzo di fori (Pit) alternati con zone piane (Land) lungo un'unica spirale. Un passaggio PitLand o Land-Pit codifica un 1 invece l'assenza di variazioni codifica lo 0.

La lettura avviene a tramite un raggio laser che viene riflesso diversamente al passaggio su pit e land.

Lungo la spirale i dati sono memorizzati con la stessa densità, quindi il CD ruota con velocità angolare variabile per mantenere la medesima velocità lineare nelle diverse aree del CD.

I dischi possono suddividersi in:

- CD-ROM (Compact Disc-Read Only Memory): utilizzano la tecnologia dei CD per memorizzare dati informatici, ogni byte viene codificato da un simbolo di 14 bit, raggruppati poi in Frame(gruppo di 42 simboli) e a loro volta in Settori (gruppo di 98 frame);
- CD-R (Compact Disc-Recordable): sono registrabili dagli utenti senza l'utilizzo dello stampo, la riflettività di pit e land viene ottenuta "bruciando", tramite un raggio laser, uno strato di materiale colorato inserito tra il policarbonato e lo strato riflettente;
- CD-RW: dischi riscrivibili molto simili come concetto a quelli registrabili ma possiedono la possibilità di essere cancellati;
- DVD (Digital Video Disk o Digital Versatile Disk): Pit più piccoli, spirale più serrata e raggio laser rosso. Più capienti, veloci e con la possibilità di scrivere su due strati (tecnologia con doppio strato).

Altri standard DVD: DVD-R, DVD+R e DVD-RAM.

SISTEMI OPERATIVI

**DEFINIZIONI** Un sistema operativo è un insieme di programmi (software) che gestiscono l'hardware, fornendo una piattaforma ai programmi applicativi e agisce da intermediario tra l'utente e la struttura fisica del calcolatore.

Il S.O. si colloca fra l'hardware e il software o comunque ciò che vede e tocca l'utente, ha il compito di assegnare le risorse e decidere chi userà la CPU e per quanto tempo.

Un'altro compito del S.O. è quello di generalizzare le caratteristiche fisiche fino a renderle virtuali, andando anche a superare i limiti fisici di certe parti del calcolatore.

Un altro modo di definire il sistema operativo è quello di considerarlo come quello strato di software che fornisce a programmi e applicazioni una macchina

virtuale attraverso la quale accedere alle risorse hardware del computer. Infatti fornisce agli sviluppatori di software la possibilità di usare API semplici che nascondono le caratteristiche dell'hardware.

**STORIA** Dal primo S.O. ad adesso sono passati molti anni, ecco come vengono divisi:

1. 1° Gen [45-55]: in questo periodo venivano usate macchine a valvole gestite con codici scritti su schede perforate, ancora non esiste il concetto di Sistema Operativo ed ogni programma era auto-contenuto;
2. 2° Gen [55-65]: i transistor vengono implementati nei computer e vengono identificati due nuovi ruoli:

- Programmatore (con schede perforate e assembly);
- Operatore (che esegue il job o programma).

Al momento i sistemi operativi sono a lotti o batch, c'è l'assenza di interazione da parte dell'utente e i programmi con esigenze simili ( job) vengono raggruppati in lotti ed eseguiti in modo sequenziale (viene fatto un job alla volta);

3. 3° Gen [65-80]: nascono i circuiti integrati, la programmazione diventa ad alto livello, i S.O. diventano multitasking e con il time-sharing (l'esecuzione della CPU viene suddivisa in quanti di tempo assegnati a turno ai diversi programmi).

In questo periodo il MIT realizza il CTSS (Compatible Time Sharing System) introducendo la multiprogrammazione (più programmi in memoria si alternano nell'uso della CPU) e il time-sharing (ripartizione del tempo di CPU tra tutti i processi in memoria e attivi tramiti porzioni di tempo delle time slice).

Inoltre ecco alcune date importanti in questa generazione:

- 1969: nasce UNIX;
- 1980: sviluppato il sistema operativo QDOS;
- 1981: la Microsoft di Bill Gates, acquista i diritti del QDOS e lo rivende, con il nome di MS-DOS (IBM lancia il primo PC);
- 1984: rilasciato l'Apple Macintosh, con il sistema operativo a interfaccia grafica System 1.0;
- 1985: la Microsoft lancia la prima versione di Windows, il Windows 1.0;
- 1991: nasce Linux.

4. 4° Gen: L'attenzione si sposta sull'utente e sulle sue esigenze:

- Interfaccia grafica;
- Interattività;
- Collegamento in rete;

Nasce il Personal Computer.

**SERVIZI** I principali servizi che un S.O. offre sono:

- Interfaccia utente: che può essere a riga di comando, più veloce ma meno intuitiva, oppure GUI quindi con grafica.
- Esecuzione di programmi: il sistema deve poter caricare, eseguire e terminare senza problemi e crash un qualsiasi processo.

Il processo è formato da:

1. Algoritmo: sequenza di passi che consentono di risolvere un problema;
2. Programma: descrizione di un algoritmo tramite un linguaggio che ne rende possibile l'esecuzione da parte di un processore ;
3. Evento: esecuzione di una delle istruzioni da parte del processore;
4. Processo: sequenza di eventi prodotti da un processore nell'esecuzione di un programma.

Al giorno d'oggi tutti i S.O. operativi sono multitasking, cioè possono eseguire più programmi contemporaneamente, nei sistemi con un solo processore l'esecuzione contemporanea dei processi è virtuale.

Un processo può essere:

- Running: le sue istruzioni sono eseguite da un processore (solo un programma può esserlo);
- Ready: il processo è in attesa che le sue istruzioni vengano eseguite da un processore;
- Waiting: il processo è in attesa di un evento.

Oltre a questi stati un processo può essere new e terminated.

- Comunicazione tra i processi: il S.O. deve fornire supporto allo scambio di informazioni tra processi è lo può fare allocando spazi di memoria in cui i processi condividono i dati (Memoria condivisa) oppure via messaggi. Qualunque soluzione adottata supporta la sincronizzazione.
- Gestione memoria: il S.O. crea una macchina virtuale che consente di far riferimento a spazi di indirizzi virtuali, impedisce ad un processo di accedere a locazioni di memoria non autorizzate e ignora se il programma e i suoi dati siano fisicamente residenti in memoria centrale o su memoria di massa.

Questa macchina virtuale gestisce i file system cioè quel componente che permette al S.O. di:

- eseguire operazioni di lettura/scrittura da parte di programmi;
- creare, cancellare ed eseguire file;
- proteggere.

- Gestione delle periferiche: la macchina virtuale creata dal S.O. dispone delle proprie periferiche dediche ad ogni processo che mascherano le caratteristiche fisiche delle vere periferiche.

Il S.O. deve fornire strumenti adeguati alla gestione dell'I/O da parte dei programmi con protezione e condivisione, quest'ultimo attraverso lo Spooling (spostamento di dati in un buffer in attesa di essere smistati verso il dispositivo o l'applicativo che li deve elaborare).

**API e System call** Le chiamate a sistema (system call), costituiscono l'interfaccia tra il processo utente e il sistema operativo. Il programmatore usa le API che specificano le funzioni a disposizione dei programmatori, nonché parametri da usare e valori restituiti.

Per funzionare quando il processo chiama il S.O. si sospende in attesa che questo termini le attività richieste (return from system call), se l'attività richiede tempo, il processo può entrare in stato di wait ed essere riattivato solo all'arrivo di un opportuno interrupt.

**CLASSIFICAZIONE S.O.** I sistemi operativi vengono classificati come:

- Real time: tutti quei sistemi di guida autonoma, biomedicali o militari;
- Singolo utente – singolo programma: come MS-DOS;
- Singolo utente – multiprogramma: come Windows o Mac OS;
- Multiutente: come Windows Server, Unix o Linux;
- Multiprocessore: come Windows NT, Solaris, Digital UNIX, OS/2 e Linux.

**STRUTTURA S.O.** Alcuni S.O. meno recenti non hanno una struttura ben definita e sono realizzati con obiettivi legati più che altro alle dimensioni e all'efficienza.

Al contrario i S.O. più recenti sono divisi in moduli con funzioni ben specifiche.

**Sistemi stratificati** I sistemi operativi possono essere suddivisi in parti di codice più piccoli e più gestibili, per far ciò si ricorre al metodo stratificato che prevede di suddividere il SO in un certo numero di livelli o strato, dove ogni strato offre servizi a quello superiore e usa quelli dello strato inferiore.

Questo tipo di progettazione permette la definizione più marcata degli stati e dei servizi ed un'efficienza complessiva del sistema; lo strato più interno è il kernel.

**Kernel** Il kernel è una parte del S.O. che fornisce alle applicazioni l'accesso sicuro e condiviso all'hardware ma non contiene applicazioni o processi ( es: interfaccia utente)

**Kernel monolitico** In questo modello c'è una completa astrazione dell'hardware sottostante, una stretta integrazione interna dei componenti rende un buon kernel monolitico estremamente efficiente; in questo caso però non è possibile aggiungere un nuovo dispositivo hardware senza aggiungere il relativo modulo al kernel, operazione che richiede la ricompilazione del kernel.

**PROCESSI** Il processo è un programma in esecuzione, è un'entità attiva e può generare altri processi.

Esso è l'unità di lavoro del sistema in esecuzione, e il sistema stesso è un insieme di processi.

Nel processo si distinguono diverse parti:

- Algoritmo: sequenza di passi che consentono di risolvere un problema;
- Programma: descrizione di un algoritmo tramite un linguaggio che ne rende possibile l'esecuzione da parte di un processore;
- Processo: sequenza di eventi prodotti da un processore nell'esecuzione di una istanza di programma.

**Time sharing** Per permettere l'esecuzione di più processi allo stesso tempo in computer con un singolo processore si utilizza la tecnica del time sharing che divide l'utilizzo della CPU fra tutti i processi dando a ciascuno un margine di tempo detto time slice.

Questo periodo di tempo è velocissimo per non far percepire rallentamenti all'utente, ogni programma ha più time slice e aspetta il suo turno per utilizzare la CPU.

**Rappresentazione in memoria** Lo spazio di memoria di un processo può essere diviso in sezioni:

- Codice: contiene il codice del programma (sola lettura);
- Dati: in questa sezione vengono immagazzinate le variabili statiche (la sua dimensione può essere modificata).
- Stack: usato per allocare dinamicamente le variabili locali usate nelle funzioni, per passare parametri alle funzioni e per restituire valori dalle stesse;
- Heap: memoria allocata dinamicamente durante l'esecuzione del programma.

Il S.O. rappresenta ogni processo con un PCB (Process Control Block) una sorta di "fotografia" del processo nel momento in cui smette di usare la CPU nel suo time slice; il PCB è necessario per permettere al processo di riprendere esattamente dal momento del suo blocco. Il PCB salva:

- Stato del processo;
- Istruzione successiva;
- Registri della CPU;
- Informazioni dello scheduling;
- ...

Il sistema operativo ha il compito di evitare i deadlock cioè quando un processo si blocca e a sua volta blocca altri in maniera irreversibile, ciò accade spesso quando più processi cercano di utilizzare le stesse risorse andando in conflitto.

**Stati del processo** Durante tutto il suo ciclo di vita il processo può avere 5 stati:

1. New: il processo è appena stato creato;
2. Ready: il processo adesso aspetta il suo turno per essere assegnato ad un processore;
3. Running: il processo sta utilizzando un processore;
4. Waiting: il processo è in attesa di qualcosa, ciò può accadere se esso aspetta un segnale input dall'utente;
5. Terminated: il processo ha finito il suo ciclo di vita.

**Tipi di processi** I processi si dividono in due categorie:

- I/O-Bound: si differenziano per la poca esecuzione di calcoli, infatti in questi processi stanno per la maggior parte del tempo nello stato di waiting, eseguendo istruzioni I/O;
- CPU-Bound: spendono la maggior parte del tempo in computazione a fronte di pochissime operazioni di I/O, passano spesso dallo stato di ready a quello di running.

**Context Switch** Quando la CPU cambia il processo in esecuzione, il sistema deve salvare lo stato del vecchio processo, determinare quale sia il nuovo processo da eseguire, caricare lo stato del nuovo processo e avviarlo. Queste attività prendono il nome di Context Switching, il tempo per farli deve essere il più piccolo possibile.

Processi e sottoprocessi

Tra le possibili cause di context switch può esserci la creazione di un processo, il processo che crea altri processi è detto padre quelli creati sono detti figli.

In questo caso si crea una struttura ad albero dove la radice è il processo antenato di quella dopo.

Resource sharing

Il processo padre può:

- condivide con il figlio le proprie risorse (e viceversa);
- condividere un sottoinsieme con il figlio;
- non condividere risorse con il figlio.

Esecuzione

Esistono due possibili strade quando si parla di esecuzione fra processi e sotto-processi:

1. padre e figlio eseguono concorrentemente;
2. Il padre attende che il figlio termini.

Spazio degli indirizzi

Quando bisogna salvare in memoria i processi e sottoprocessi il S.O. può:

- salvare il figlio come duplicato del padre;
- usare il figlio per caricare un programma.

**Terminazione processo** Con l'istruzione exit il processo chiede al S.O. di terminare, in questo modo tutte le risorse del processo vengono deallocate.

Nel caso di processi padre e processi figli è il padre a decidere quando terminarli (abord), ma per farlo deve conoscerne l'identità.

Il padre decide di terminare il figlio se:

- il figlio ha ecceduto nell'uso delle risorse;
- il compito assegnato al figlio non è più richiesto;
- il padre termina e quindi il figlio non ha più ragione di esistere (terminazione a cascata).

**Processi cooperanti** I processi concorrenti possono essere:

- Indipendenti: la loro esecuzione non può influire né essere influenzata da altri processi.
- Cooperanti: l'esecuzione di un processo influisce su quella degli altri.

Con la cooperazione si permette l'information sharing (utile per consentire all'utente la condivisione delle risorse), la speed-up nella computazione, la modularità (scomporre le funzionalità di un sistema in più parti) e infine la convenienza di offrire più funzionalità in modo contemporaneo.

Per cooperare si utilizza la modalità dei sistemi a memoria condivisa, in questo modo i processi condividono una parte di memoria, solitamente il S.O. impedisce accessi alla memoria di altri processi a meno che questo non avvenga entro precise modalità predefinite dal S.O. stesso.

**Producer & Consumer** Un paradigma per la programmazione concorrente è il problema del produttore/consumatore, dove nascono due scenari:

1. unbounded-buffer: dove c'è il buffer senza limiti;
2. bounded-buffer: il buffer ha una dimensione limitata, per questo il consumatore deve attendere se il vettore è vuoto o il produttore deve attendere se il vettore è pieno.

Per questo il buffer deve essere condiviso dal produttore e dal consumatore, è fornito da S.O. o essere uno spazio di memoria condiviso.

**Condivisione tramite messaggi** Esiste uno strumento che consente la comunicazione fra processi e la loro sincronizzazione senza memoria, è chiamato Message Passing, consiste nell'invio di messaggi (send) e la ricezione (receive) tra processi.

Per farlo i due processi devono instaurare una comunicazione fra loro che può avvenire in modo fisico, tramite bus, o in via logica.

La comunicazione può essere:

- Sincrona o asincrona: dove il processo si può bloccare o no durante la comunicazione;
- Diretta o indiretta: la comunicazione tra processi avviene processo a processo o tramite mailbox;
- Buffering;

**THREAD** Fino ad ora abbiamo considerato il processo come un programma in esecuzione in un unico flusso di controllo, ma al giorno d'oggi i S.O. consentono di eseguire processi attraverso più flussi di controllo, che sono solitamente chiamati thread.

I thread sono l'unità base dell'uso della CPU

Nello stesso processo possono nascere più thread, per questo si parla di multithreading, un processo tradizionale è detto processo pesante perché ha un'unico thread.

**Benefici** Con i thread e il multithreading abbiamo una responsiveness (tempo di risposta) più alta.

Grazie alla definizione di thread, nello stesso task è possibile il resource sharing , in modo da permettere all'applicazione di avere più thread che operano su uno stesso spazio di indirizzi; anche per questo è più facile creare e distruggere thread.

**SCHEDULING** Lo scheduler della CPU è il meccanismo sul quale sono basati i sistemi che supportano la multiprogrammazione andando a massimizzare il tempo d'uso della CPU.

In questi sistemi le prestazioni sono influenzate dal tipo di attività dei processi, in un processo di tipo I/O bound i cicli di CPU burst (momento dove il processo lavora in CPU) sono molti ma molto brevi perché interrotti da I/O; invece in un processo di tipo CPU bound i cicli di CPU burst sono meno ma più duraturi.

I cicli CPU burst sono solitamente molto brevi

per evitare rallentamenti.

**Ready queue** Lo scheduler a breve termine (altro nome della multiprogrammazione) lavora nella ready queue,una struttura dati dello scheduler dove sono parcheggiati tutti i processi nello stato di ready.

Quando un nuovo processo entra nella ready queue si possono adottare due differenti politiche:

- Preemptive scheduling: il processo che arriva ha una priorità, se è maggiore rispetto a quella del processo in esecuzione lo spodesta.
- Non preemptive: la CPU rimane al processo in esecuzione fino alla fine.

**Dispatcher** È quella parte del S.O. che passa effettivamente il controllo della CPU al processo selezionato dallo scheduler; deve eseguire lo switching (ripristino del contesto del processo) e il riavvio dell'esecuzione.

**Algoritmi di scheduling** Il confronto tra algoritmi avviene in base:

1. Utilizzo di CPU;
2. Throughput (numero di processi completati nell'unità di tempo);
3. Tempo di turnaround (tempo complessivamente necessario per eseguire il processo);
4. Tempo di attesa(trascorso dal processo nella coda dei ready);
5. Tempo di risposta.

FCFS/FIFO

Il primo processo ad arrivare sarà il primo a finire, quando il processo entra in coda il suo PCB viene messo in fondo.

Non è molto efficace perché un solo processo molto lungo può bloccare molti processi che avrebbero eseguito velocemente (effetto convoglio).

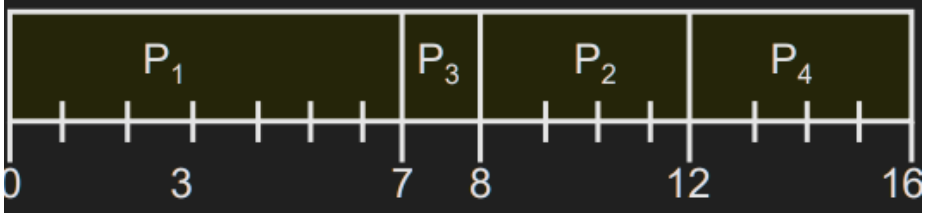
SJF

L'algoritmo Shortest Job First nasce con l'idea di privilegiare i processi più corti. In caso il CPU burst sia uguale (durano ugual tempo) si utilizza la tecnica FIFO.

Questo algoritmo è effetto da starvation, quindi è possibile che i processi molto lunghi non avvengano mai; esistono due modalità:

- Non preemptive: quando un processo ottiene la CPU non viene interrotto;

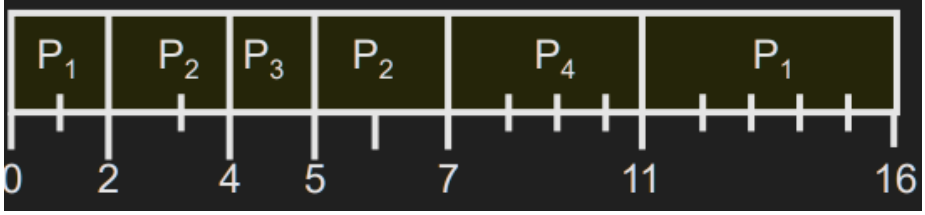
| Processo | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0.0 | 7 |
| $P_2$ | 2.0 | 4 |
| $P_3$ | 4.0 | 1 |
| $P_4$ | 5.0 | 4 |




esempio

- Preemptive: quando arriva un nuovo processo con CPU burst minore del CPU burst rimanente per il processo correntemente in esecuzione, il nuovo processo prende la CPU.

| Processo | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0.0 | 7 |
| $P_2$ | 2.0 | 4 |
| $P_3$ | 4.0 | 1 |
| $P_4$ | 5.0 | 4 |




esempio

Priority scheduling

Si usa un valore numerico per indicare quale processo fare prima, più il valore è alto più velocemente deve accedere alla CPU; più il tempo di attesa in coda aumenta per un processo più il valore della sua priorità aumenta (aging).

Round Robin

Unico algoritmo visto fino ad ora adatto ai S.O. time sharing, è una variante del FCFS.

Utilizza uno scheduler chiamato Round Robin dove la coda è gestita in maniera circolare, dove ogni volta che un processo viene messo in esecuzione, lo scheduler imposta un timer che ne limita il tempo di esecuzione.

Code multiple

Questo algoritmo gestisce i processi di classe diverse; divisi in foreground, quelli interattivi, e background (quelli batch).

In questo algoritmo la ready queue viene partizionata in più code separate e ciascun processo è associato in modo statico ad una coda.

La particolarità è l'utilizzo di scheduler diversi per cose diverse, ad esempio RR sui processi foreground e FCFS per i processi in background.

Per sapere da che coda prelevo il prossimo processo da eseguire si utilizzano scheduling tra le code che possono essere:

- il non eseguire nessun processo della coda n se la coda n-1 non è vuota;
- il definire dei quanti di tempo tra le code.

Code multiple con feedback

Nella code multiple normale ogni processo rimane sempre nella stessa coda, si può invece fare in modo che i processi mutino dinamicamente posizione, ad esempio in funzione del CPU burst che hanno.

Andando a creare un struttura dove la coda di livello più alto è quella che contiene i processi con CPU burst inferiore.

Un processo viene inserito inizialmente nella coda più veloce per poi retrocedere alla coda di livello inferiore quando non termina nel time slice di quella coda.

**CONCORRENZA** I processi possono essere eseguiti in modo:

- Concorrente: competono per l'accesso ad una risorsa;
- Parallelo: sono eseguiti su diverse unità funzionali.

Qualunque sia l'esecuzione si generano problematiche sull'integrità dei dati condivisi, con il race condition si indica l'accesso a dati senza sincronizzazione e controlli; per ovviare al problema si utilizzano parti di codice dette atomiche.

**Critical section** È una parte di codice, costruita attorno alle variabili condivise, che avranno un unico utilizzatore andando a scongiurare l'inconsistenza.

Una Sezione Critica (CS) è una porzione di codice in cui i dati condivisi da processi cooperanti possono essere manipolati

La CS è preceduta da una sezione di ingresso e una di uscita.

Ogni thread ha una critical section in cui accedere per modificare variabili condivise, l'esecuzione di queste parti deve garantire il MUTEX (mutua esclusione, è un meccanismo di blocco).

Il Thread non deve rimanere bloccato per sempre (Liveness), per questo tramite il Bounded waiting, al thread può essere bloccato l'accesso solo un numero limitato di volte.

**Test and Set** In questa modalità si utilizzano variabili do lock, allocate all'interno della memoria condivisa che funziona come lucchetto; se per esempio questa cella ha valore 0 allora è possibile accedere alla variabile altrimenti no.

Il valore viene settato dal processo che utilizza la risorsa nella CS.

Un problema di questa modalità è l'interruzione forzata tramite interrupt dei processi che aspettano la variabile in lock; per ovviare è stato creato il Test and Set Lock (TSL) una variabile che effettua i controlli.

**Semafori** Per superare l'impossibilità della generalizzazione delle regioni critiche (difficoltà nel risolvere problemi complessi) sono nati i semafori.

Un semaforo "S" è una variabile intera cui si può accedere, escludendo l'inizializzazione, solo tramite due operazioni atomiche predefinite: wait() e signal().

Al contrario del MUTEX, dove un processo deve detenere il mutex prima di poter acquisire la risorsa, un semaforo è un meccanismo di segnalazione dove i processi possono indicare se stanno acquisendo o rilasciando la risorsa.

Quindi ricapitolando:

- I mutex hanno lo scopo di proteggere una risorsa condivisa, in modo che più processi non possano accedervi contemporaneamente;
- un semaforo è un meccanismo di segnalazione tra processi e serve a condividere una risorsa che può essere usata contemporaneamente da un numero limitato di processi

**Problemi concorrenza** Quando più processi concorrono per una risorsa e si applicano algoritmi per gestirli si può incappare in alcuni errori:

- Busy waiting: si verifica quando un thread attende il verificarsi di una condizione (come la modifica di un semaforo) e lo fa verificando contin-

uamente se la condizione sia diventata vera. Questo consuma tempo in CPU;

- Starvation: avviene se un thread in attesa di una certa risorsa, non riesce mai ad accedervi perché arrivano continue richieste da thread con privilegi maggiori;
- Deadlock: due o più thread aspettano indefinitamente un evento che può essere causato solo da uno dei thread bloccati;
- Producer & Consumer: due processi condividono un buffer di dimensione limitata, il "produttore" riempie un buffer, un item alla volta, finchè non è pieno e il "consumatore" svuota il buffer, un item alla volta finchè non è vuoto;
- Readers & Writers: thread scrittori e lettori devono accedere in modo concorrente ad un file, dove più letture possono avvenire contemporaneamente ma le scritture sono esclusive;
- 5 filosofi: in questo problema si usa un esempio dove ci sono 5 (o n, dove n>1) filosofi, che rappresentano i processi, a tavola per mangiare e pensare. Se un filosofo vuole mangiare deve usare due bastoncini ma tra un filosofo e l'altro c'e' un bastoncino solo, ogni filosofo può prendere solo una bacchetta alla volta e non può sottrarre bacchette dalle mani degli altri filosofi.

In questo esempio è impossibile che tutti i filosofi mangino allo stesso tempo altrimenti si ruberebbero le risorse fra loro, per questo in maniera alternata i filosofi devono mangiare e pensare.

**Monitor** I monitor sono costrutti di livello più alto rispetto ai semafori, ogni monitor è una struttura dati (o un oggetto) che permette la mutua esclusione e la sincronizzazione su strutture dati condivise da più thread ed un solo thread alla volta può essere attivo all'interno di un monitor.

Il monitor:

- garantisce la mutua esclusione (un solo thread alla volta può eseguire il codice contenuto nel monitor);
- la possibilità di usare metodi pubblici utilizzabili da processi;
- un insieme di variabili di condizione su cui possono essere invocate le primitive wait e signal.

L'ipotesi fondamentale per la realizzazione di semafori e monitor è l'esistenza di memoria condivisa a cui i diversi thread possono fare riferimento; sfortunatamente nei S.O. moderni non esiste shared memory.

Per ovviare al problema si utilizzano tecniche di message passing.

**GESTIONE DELLA MEMORIA** Per funzionare c'è bisogno di due tipi di indirizzi che sono fra loro uniti tramite un legame, questi indirizzi sono:

- Indirizzo fisico: è l'indirizzo nello spazio di memoria centrale che individua in modo univoco una parola in esso contenuta;
- Indirizzo logico: è astratto ed esiste solo all'interno dello spazio di indirizzamento del processo, e ne rappresenta lo spiazzamento rispetto alla prima parola. Questo indirizzo viene utilizzato come riferimento per accedere alla posizione di memoria fisica.

La corrispondenza fra i due indirizzi può avvenire durante la compilazione, durante il caricamento o in esecuzione.

**Rilocazione** È quando un programma viene allocato in memoria, quindi in questo momento si decide l'indirizzo fisico che avrà.

La rilocazione avviene in maniera:

- Statica: nel momento di esecuzione il S.O., tramite il LOADER, decide lo spazio di memoria dove andrà allocato il processo.
- Dinamica: il programma viene caricato in memoria ogni esecuzione, andando a cambiare indirizzo.

**MMU** Detto anche MEMORY-MANAGEMENT UNIT è un dispositivo con il compito di eseguire la corrispondenza tra indirizzi in fase di esecuzione. Ciò è possibile grazie al registro di rilocazione, dove la somma fra questo valore e la parte indirizzo della istruzione (indirizzo logico) crea l'indirizzo fisico.

**Politiche di allocazione** Insieme di regole per gestire la memoria, le politiche vengono valutate tramite:

1. Memoria sprecata;
2. Sovraccarico temporale: complessità computazionale delle operazioni di allocazione/deallocazione;
3. Sovraccarico nelle operazioni di accesso alla memoria

Le allocazioni possono essere:

- Contigue: i programmi salvati vengono messi uno dopo l'altro in base all'avvio, ne esistono 4 tipologie:

1. Monitor Monoprocesso: modo più semplice, la memoria viene divisa in due dove in una area viene allocata la parte residente del S.O. e nell'altra vengono allocati i processi transienti.
2. Statica: suddividiamo la memoria in modo che in ogni partizione (di dimensione fissa) si possa caricare un programma e quindi fare eseguire

un processo. Si usa il TDP, una tabella di descrizione delle partizioni, per monitorare lo stato delle partizioni.

3. Dinamica: La suddivisione della memoria avviene dinamicamente in base alle esigenze dei processi, quindi non c'è il problema di non avere partizioni troppo piccole per alcuni processi; anche qui necessito del TDP in unione con una lista delle aree libere in memoria. Con la partizione dinamica abbiamo il problema della frammentazione esterna infatti si possono creare delle aree libere tra le partizioni che non possono essere utilizzate.
4. Segmentazione: creata per superare la frammentazione esterna, andando a ridurre le dimensioni delle zone allocate andando a dividere un programma in segmenti che verrà allocato in modo contiguo.

- NON contigue: si basa sulla suddivisione del processo in pagine logiche con corrispondenti pagine fisiche. Sostanzialmente viene allocata solo la parte di programma utilizzato.

Questa tecnica è detta paging o paginazione, colui che si occupa di caricare (swap-in) o deallocare (swap-out) è detto pager, se al caricamento di una pagina il componente non la trova in memoria o c'è un errore nella gestione degli indirizzi viene generato un page fault.

Quando avviene l'errore, dovuto alla mancanza della pagina, si genera un segnale di trap che informa il S.O, a questo punto il S.O. cerca nel disco la pagina mancante e la carica nella memoria fisica; al caricamento si modifica una tabella che ogni processo ha che indica quali pagine contiene.

È cruciale scegliere bene il meccanismo di sostituzione delle pagine in modo da tenere basso il numero dei page fault.

**Algoritmi di caricamento** I diversi approcci alla scelta della pagina da scaricare sono:

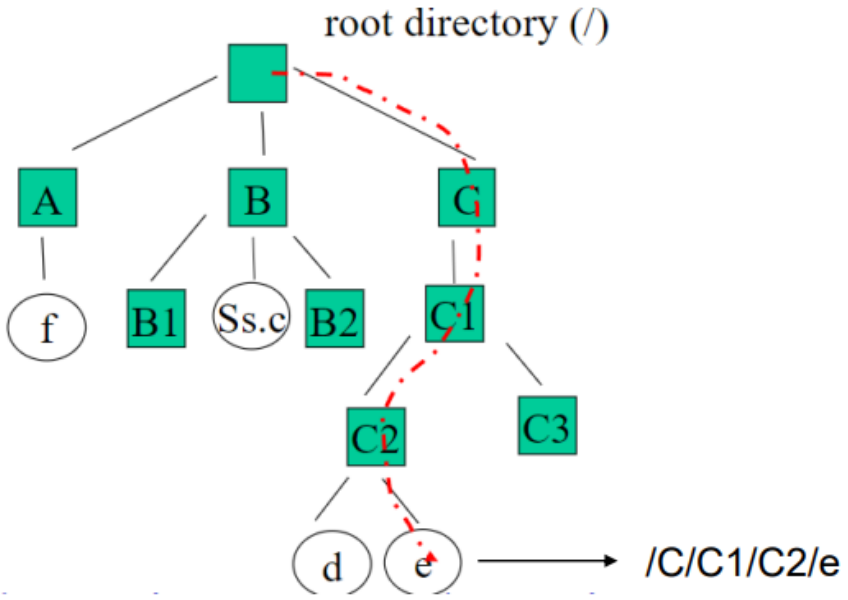
1. FIFO: l'algoritmo associa ad ogni pagina il tempo in cui è stata caricata in memoria e, se c'è necessità di scaricare una pagina, sceglie la pagina più vecchia;
2. Least Recently Used (LRU): l'algoritmo associa a ciascuna pagina l'istante in cui è stata usata per l'ultima volta e in caso di page fault con swap scarica la pagina che non viene acceduta da più tempo;
3. Least Frequently Used (LFU): viene usato un contatore che mantiene il numero degli accessi e viene scaricata la pagina che ha avuto in numero minore di hit;
4. Most Frequently Used (MFU): viene usato un contatore che mantiene il numero degli accessi e si scarica la pagina che ha avuto in numero maggiore di hit;

**GESTIONE DEI FILE SYSTEM** Il file system è la parte del SO che si occupa di mantenere i dati/programmi in modo persistente tramite file e directory.

**Accesso ai file** In questo tipo di accesso i byte possono essere letti in qualsiasi ordine, nei moderni sistemi operativi tutti i file sono automaticamente ad accesso diretto.

I file system esistenti sono tutti di tipo gerarchico è utilizzano i path name per indicare la posizione univoca di ogni file.

Si utilizza un grafo ad albero per determinare le posizioni, con la radice principale detta "root directory"; ogni file è univocamente determinato dal cammino che lo collega alla radice.




**Implementazione FS** Per creare un nostro FS dobbiamo risolvere 3 quesiti:

1. Implementazione file: normalmente i dati sono memorizzati in unità di ampiezza fissa e si devono memorizzare gli attributi e la posizione dei singoli blocchi;
2. Implementazione directory: le cartelle sono file con uno specifico formato;
3. Gestione disco: per farlo dobbiamo tenere traccia della root directory e anche in quali spazi del disco ci sono file e dove no.

Implementazione file

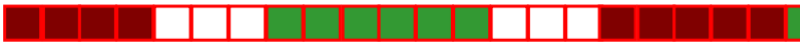
Per allocare i file si può utilizzare la tecnica dell'allocazione contigua che consiste nel memorizzare ogni file o in un gruppo di blocchi contigui detti run.

Questo porta enormi svantaggi come la possibilità di sprecare spazio se l'ultimo blocco non è del tutto pieno e nel disco saranno presenti buchi non grandi abbastanza per tenere altri file nelle parti di file cancellate.

Esempio:

File allocati in maniera contigua 

Caso di file più piccolo rispetto ai blocchi

Disco dopo la cancellazione di alcune parti 

Un'altra tecnica è quella delle liste concatenate, dove ogni blocco contiene l'indirizzo di quello successivo, in questo modo non c'è frammentazione esterna e il file può essere salvato su più parti di memoria portando però ad un'accesso alle informazioni molto lento.

La tecnica principale usata da Unix è la Index-node o i-node, in questo modo solo gli i-node dei file in uso devono risiedere in RAM e lo spazio è proporzionale al numero massimo di file aperti e non dipende dall'ampiezza del disco.

Infatti ogni i-node ha associato un numero univoco all'interno del dispositivo e ogni file presente è identificato come un collegamento fisico all'i-node tramite il suo numero. Quando un programma cerca di accedere ad un file tramite un nome (es. documento.txt), il sistema operativo cerca l'i-node corrispondente e recupera tutte le informazioni sopra descritte per operare correttamente con il file.

Implementazione Directory

Devono permettere di recuperare tutte le informazioni relative ai file contenuti il punto fondamentale è associare il nome del file ad attributi e dati (indirizzo/i dei blocchi).

La soluzione più semplice è far contenere alla directory una tabella con un elemento per ogni file, dove attributi e indirizzi dei blocchi del file X sono memorizzati direttamente nell'elemento della tabella relativo ad X (usata da FAT).

Gestione disco

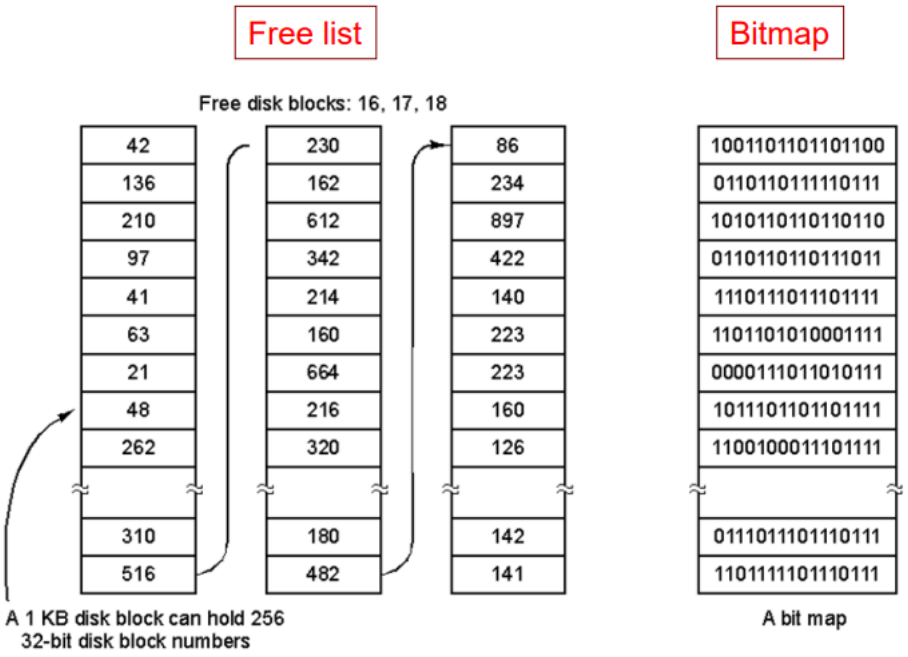
Tutti i FS dividono i file in blocchi NON CONTIGUI di ampiezza fissata ed eseguono letture e scritture su blocchi o multipli.

Per decidere l'ampiezza dei blocchi dobbiamo chiederci che caratteristica vogliamo per il nostro FS, con blocchi piccoli usiamo meglio lo spazio disco (diminuiscono la frammentazione interna) con blocchi grandi velocizziamo gli accessi.

Per tenere traccia dei blocchi liberi su disco abbiamo 2 alternative:

1. Free list: lista concatenata di blocchi pieni di indirizzi di blocchi liberi;
2. Bitmap: una mappa di bit con un bit per ogni blocco, mantiene la contiguità dei blocchi.

Esempio:




Gestione errori e affidabilità

Se una traccia ha un settore difettoso il FS in automatico sostituisce il settore difettoso con uno di riserva, andando a slittare tutto per evitare il problema.

Il FS esegue backup periodici in caso di crash e problemi accidentali in modo da non perdere dati.

Con Stable Storage si indica una tecnica che permette di mettersi al riparo da errori che si verificano durante una scrittura, infatti nello stable storage, ogni volta che eseguiamo una write si garantisce che il valore scritto è corretto oppure è uguale a quello vecchio.

**GESTIONE DISPOSITIVI I/O** I dispositivi I/O sono molti ed eterogenei con obiettivi differenti (memorizzazione, trasmissione, interazione, ecc) per questo

è necessario uniformare gli standard per alleggerire la loro gestione da parte del S.O.

Il kernel ha un sottosistema apposito per la gestione I/O con driver che offrono un'interfaccia uniforme per l'accesso ai dispositivi.

Alcuni meccanismi hardware sono comuni fra tutti i dispositivi, e sono:

- Porta: punto di connessione fisica (via cavo o etere) tra il sistema di calcolo e il device di I/O;
- Bus: insieme di connessioni al sistema di calcolo utilizzate da più dispositivi contemporaneamente;
- Controller: processore dedicato con microcodice che svolge attività specializzate sul device di I/O.

**Registri e controller** Tipicamente una porta di I/O opera sulla base di 4 registri:

1. Status: bit che indicano lo stato della porta ;
2. Control: viene usato per attivare un comando o per cambiare il modo operativo del dispositivo;
3. Data-in: byte disponibili in lettura;
4. Data-out: byte disponibili in scrittura.

Per accedere ai registri si usa il controller e l'accesso può avvenire:

- Mappando i dispositivi in memoria: i registri sono visti nello spazio di indirizzamento della memoria, ciò permette di scrivere driver in linguaggio ad alto livello andando a sacrificare la protezione;
- Mappando i dispositivi sull' I/O: si accede ai registri tramite istruzioni specifiche evitando problemi di gestione cache.

Comunicazione controller

Per comunicare con il controller possiamo usare diversi modi:

- Polling: la CPU controlla periodicamente lo stato del controller per verificare se una operazione è completata o no.

Si basa sul paradigma handshaking con l'utilizzo di 2 bit per coordinare la relazione produttore/consumatore (il 1° detto bit busy, il 2° command ready); chi controlla il bit busy è in polling.

Questo procedimento diventa inefficiente perchè, sebbene le interrogazioni siano frequenti, si trova raramente un dispositivo libero;

- Interrupt: il controller segnala alla CPU il completamento dell'operazione richiesta, infatti è presente un hardware apposito, la linea di richiesta dell'interrupt, controllato direttamente dalla CPU.

Quando si rileva un segnale la CPU blocca quello che sta facendo ed esegue una routine apposita detta interrupt handler, il compito della routine è quello di determinare perché la CPU si è bloccata e portare a termine l'operazione I/O;

- DMA (Direct Memory Access): la segnalazione alla CPU viene mandata per blocchi e non per byte, grazie ad un accesso diretto del controller DMA sulla memoria centrale. Inoltre la CPU interviene direttamente nella gestione degli interrupt e questa attività è molto dispersiva, per questo può essere utile un controller dedicato che trasferisca i dati direttamente dall'I/O alla memoria centrale senza contattare la CPU.

Il nuovo controller DMA trasferisce su 2 livelli:

1. CPU-DMA: la CPU chiede al controller DMA di avviare il trasferimento e il controller DMA segnala il completamento (tramite interrupt);
2. DMA-Device Controller-Memoria: il controller DMA agisce trasferendo i dati dalla memoria centrale al controller del device e viceversa.

**Interfacce per applicazioni [Driver]** Il S.O. deve offrire alle applicazioni un'interfaccia astratta per far utilizzare il sistema I/O alle applicazioni, per farlo si necessita dei driver.

I driver costituiscono dunque lo strato del S.O. più vicino all'hardware dell'I/O. È un software, spesso scritto in assembly, che permette ad un sistema operativo di pilotare un dispositivo hardware.

In questo modo il S.O. agisce in modo indipendente dall'hardware e questo va a vantaggio di chi implementa il S.O. e di chi produce hardware.

**Sottosistema I/O - Servizi** I principali servizi offerti dal sottosistema di I/O del kernel sono:

- Scheduling: cioè gestire l'ordine con cui le richieste di I/O sono gestite, il problema è particolarmente rilevante nella gestione degli accessi al disco;
- Buffering: si utilizza un buffer, cioè una regione di memoria che contiene dati mentre questi sono trasferiti da un dispositivo, per permettere la bufferizzazione. La tecnica è essenziale perché i dispositivi potrebbero avere velocità diverse oppure avere diverse dimensioni dei blocchi trasferiti. Un'altra importante caratteristica è la semantica di copia, cioè garantire che la versione dei dati scritta su un dispositivo sia la stessa versione presente in memoria;
- Caching. si utilizza una cache per migliorare le prestazioni dei dispositivi di I/O, al contrario del buffer la cache non può contenere dati di cui non esiste altra copia;

- Spooling: si utilizza lo spool, cioè un buffer contenente output per dispositivi che non possono accettare flussi di dati da più processi contemporaneamente, come ad esempio la stampante.

Lo spooler è l'unico programma che ha accesso diretto alla periferica, e gestisce il buffer offrendo agli utenti alcune operazioni aggiuntive come la visualizzazione della coda di attesa dei processi o la cancellazione/sospensione di uno di essi.