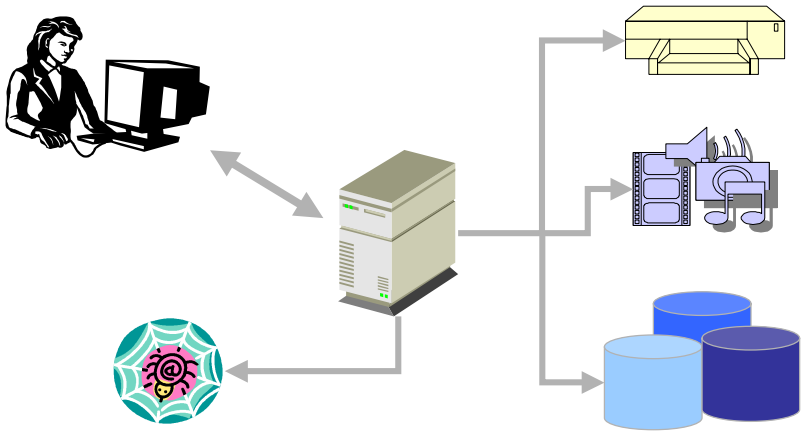
Sistemi Operativi



Perché esiste il Sistema Operativo ?

Un Computer è un sistema elettronico digitale complesso in grado di fornire diverse risorse:

- Un sistema di calcolo programmabile (il processore)
- Una memoria di lavoro (la RAM)
- Diversi sistemi di memorizzazione di massa (Dischi, Cd..)
- Protocolli di livello 1 e 2 sui sistemi di telecomunicazioni (schede di rete)
- ……

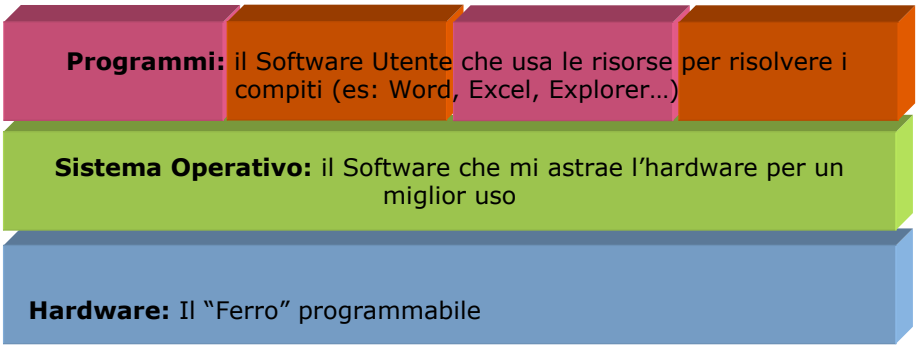
L'utente potrebbe usare direttamente tutte queste risorse, sono accessibili! Ha solo bisogno di conoscere ***qualche*** dettaglio:

- **dove inviare i comandi alla risorsa** (per esempio i comandi al processore vanno messi i determinate posizioni di memoria)
- **come scrivere i comandi** (il processore per esempio accetta sequenze di byte – il linguaggio macchina)
- **come fornire i dati per le elaborazioni** (il processore vuole che i dati vengano caricati in appositi registri)
- **dove e come ottenere i i risultati** (anche qui in appositi registri!)

Un bravo utente, in un ora di lavoro e con la documentazione corretta potrebbe quindi risolvere una o anche due moltiplicazioni, utilizzano forse un miliardesimo delle risorse disponibili nel sistema

Il Sistema Operativo astrazione del calcolatore

Per semplificare l'uso del sistema e migliorare quindi l'efficacia delle azioni dell'utente posso pensare di aggiungere, al di sopra del sistema hardware che è il sistema un Software che mi gestisca la complessità e mi fornisca degli strumenti più adatti



Organizzazione di un Sistema Operativo

Un Sistema Operativo è essenzialmente un **Gestore di Risorse**.

L'organizzazione di un sistema è quindi definita da come sono organizzate le diverse **risorse** componenti di un S.O. e le modalità di interconnessione tra di esse:

- **Sistemi monolitici**
- **Sistemi a livelli**
- **Macchine virtuali**
- **Modello cliente-servitore**

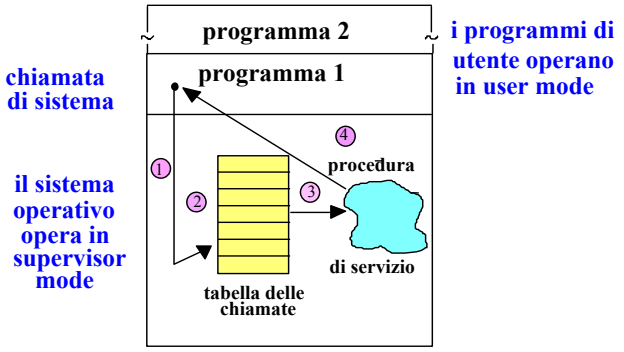
Per esempio una risorsa molto importante perché diventa uno strumento per l'uso di altre risorse è il **File System**

Come si possono utilizzare le risorse? Attraverso l'uso di **programmi;** l'esecuzione di un programma è detta **Processo**

Sistemi Monolitici

Il sistema operativo è scritto come un **insieme di procedur**e, tutte allo **stesso livello** di gerarchia.

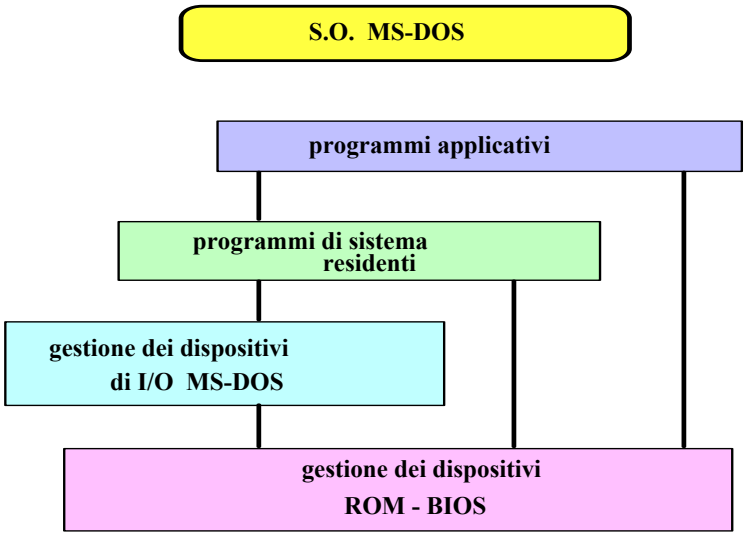
Le funzioni possono chiamarsi reciprocamente se sono invocabili attraverso **ben definite** interfacce tipicamente **interrupt software** o **trap**



Sistemi a Livelli

- Le funzioni del S.O. sono organizzate per **livelli gerarchici**
- Ogni livello definisce un **tipo** di servizio e le **modalità** per essere utilizzato dai **livelli superiori**
- **Macchine virtuali -** Ogni livello definisce una nuova macchina, aggiungendo nuove funzionalità alla precedente (non sono macchine estese, ma copie esatte del semplice hardware)
- Livelli come **aiuto** nel progetto per superare difficoltà pratiche nella realizzazione (gerarchia)

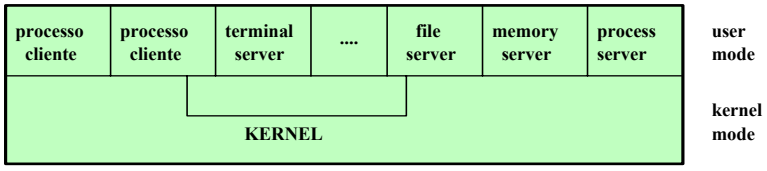
Esempio di Sistema a livelli: Ms-DOS



Modelli Cliente Servitore

Sistemi operativi **moderni:** la maggior parte delle funzioni del sistema operativo sono realizzate come **processi di utente**

- Per richiedere un servizio un **processo cliente** (utente) invia una richiesta ad un **processo servitore** (es.: *lettura di un file*)



- **Vantaggi:**
  - le funzioni sono divise in parti di dimensioni ridotte e ben specificate
  - protezione contro errori: le funzioni non hanno accesso al kernel, un guasto non blocca l'intera macchina (operano in *user mode*)
  - adatto in sistemi distribuiti

Classificazione dei Sistemi Operativi

I S.O. si classificano per soprattuto per due caratteristiche:

- **Organizzazione interna:**
  - Monoprogrammato
  - Multiprogrammato
- **Visibilità Utente:**
  - a divisione di tempo / sequenziale
  - interattivo / batch
  - special purpose / general purpose

I Sistemi Monoprogrammati

Si gestiscono i programmi in modo sequenziale nel tempo

Tutte le risorse hardware e software del sistema sono dedicate ad un solo programma per volta.

utilizzo della CPU = Tp/ Tt

Tp = tempo dedicato dalla CPU alla esecuzione del programma
Tt = tempo totale di permanenza del programma nel sistema
throughput = numero di programmi eseguiti per unità di tempo

Per avere alta efficienza, si vorrebbe avere un utilizzo vicino ad 1 e throughput elevato

→ **BASSA UTILIZZAZIONE DELLE RISORSE**

I Sistemi Multiprogrammati

Si **gestiscono** "contemporaneamente" più programmi indipendenti presenti nella memoria principale.

- Migliore **utilizzazione delle risorse** (riduzione dei tempi morti)

- **Maggiore complessità** del S.O.:
  - algoritmi per la gestione delle risorse (CPU, RAM, I/O)
  - protezione degli ambienti dei diversi programmi

- Il sistema è **sequenziale**, i processi che devono eseguire sono in coda ad attendere che il processo precedente termini oppure lasci libera la CPU

Sistemi a divisione di tempo

Astrazione di più **macchine virtuali;** non solo ho più programmi che eseguono contemporaneamente, ma ogni programma "vede" il sistema come se fosse suo perché una macchina virtuale gli concede questa astrazione. Il S.O. fa ***scheduling*** tra le diverse macchine virtuali concedendo le risorse fisiche per specifiche unità di tempo

- Ogni utente ha un proprio programma in memoria ed a ciascuno è dedicata una macchina virtuale

- Si abbreviano i tempi di attesa dei programmi corti

- Tempo impiegato dal S.O. per trasferire il controllo da un programma ad un altro (**overhead**)

I Sistemi Interattivi e Batch

SISTEMI BATCH

- I programmi sono inseriti **a lotti** nella memoria di massa e successivamente elaborati in multiprogrammazione
- **Obiettivo:** migliore utilizzazione possibile delle risorse (elevato ***troughput***)
- Scelta dell'insieme di programmi (***job mix***) in memoria principale che ottimizzano l'utilizzo delle risorse

SISTEMI INTERATTIVI

- Più utenti che utilizzano contemporaneamente il sistema di calcolo (a divisione di tempo)
- **Obiettivo:** migliorare i tempi di risposta per l'utente (in contrasto con l'ottimizzazione delle risorse)

- Scelta del processo che ha più urgenza di servizio

Sistemi General Purpose e Special Purpose

SISTEMI GENERAL-PURPOSE

- Centri di Calcolo di grandi dimensioni
- Utenze di tipo differenziato
- Modalità di uso batch ed interattivo

SISTEMI SPECIAL-PURPOSE

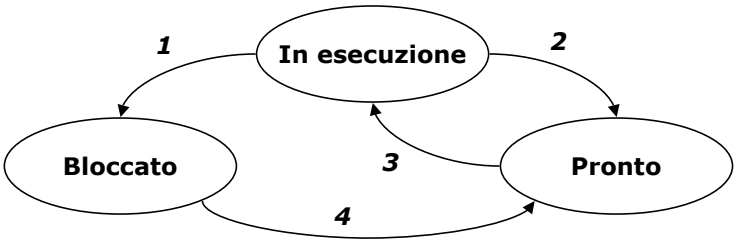
- Dedicati ad applicazioni specifiche, spesso con hardware speciale (dispositivi di I/O)
- Sistemi in tempo reale: il sistema di calcolo deve "rispondere" entro un limite di tempo (tempo di reazione)
  - tempo reale stretto: il tempo di risposta è molto stringente come vincolo e non può non essere rispettato (p.e. computer di volo in un aereo)
  - tempo reale debole: il tempo di risposta deve essere "ragionevole" (sistema di gestione di una catena di montaggio)

I processi

- Quello di **processo** è il concetto più centrale di ogni sistema operativo: l'**astrazione di un programma in esecuzione**.
- **Multiprogrammazione:** ogni processo dispone della propria CPU virtuale, ma nella realtà la CPU passa in continuazione da un processo all'altro.
- La differenza tra processo e programma è sottile:
  - il processo è un programma in esecuzione, inclusi i valori del program counter, dei registri e delle variabili;
  - il programma è l'algoritmo espresso in una notazione adeguata.
- **Gerarchia di processi:** un processo può originarne degli altri (es. FORK in UNIX) che continuano a girare in parallelo al padre. Si possono formare così alberi di processi interagenti.

Gli stati dei processi

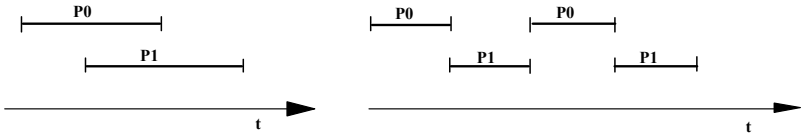
- L'interazione tra processi implica da parte della CPU la gestione degli **stati** di un processo: nella situazione più semplice **pronto, bloccato** ed **in esecuzione**.



- 1. Il processo si blocca in attesa di I\O
- 2. Lo schedulatore prende un altro processo
- 3. Lo schedulatore sceglie questo processo
- 4. Si rende disponibile il dato in ingresso

Le relazioni fra i processi

- **Processi Concorrenti:** due processi sono concorrenti se la loro esecuzione si sovrappone nel tempo; si parla di ***overlapping*** se ho più processori e di ***interleaving*** se ho un processore unico



  - **Processi indipendenti:** sono processi concorrenti la cui esecuzione non è influenzata dalla presenza degli altri processi

- **Processi interagenti:** i processi si influenzano tra loro in due modi:
  - ***Comunicano:*** si parla quindi di esecuzione coordinata
  - ***Competono:*** hanno bisogno delle stesse risorse e la loro esecuzione è influenzata dall'uso delle stesse

Comunicazione fra processi

Spesso i processi hanno necessità di scambiarsi dati, di comunicare fra loro, possibilmente in modo strutturato e senza utilizzare le interruzione quindi senza chiamare esplicitamente in causa il sistema operativo.

Questa attività naturalmente pone delle nuove problematiche.

- **Corse critiche:** quando due o più processi fanno riferimento ad una stessa risorsa si possono creare problemi di consistenza dell'uso di questa risorsa. Per evitare anomalie è necessario implementare la **mutua esclusione** nell'uso delle risorse con queste caratteristiche.
  - Es. la coda di uno spooler di stampa.

- **Sezioni critiche:** sono quelle parti di programma che accedono alla memoria condivisa, una loro opportuna gestione garantisce la corretta esecuzione delle operazioni.

Processi e programmazione concorrente

- **Mutua esclusione con attesa attiva:**
  - disabilitazione degli interrupt, previene il passaggio di controllo
  - lock, variabile di riferimento per il controllo della risorsa
  - alternanza stretta, algoritmo di Peterson, TSL (sol. hardware)
- **Sospensione e risveglio:**
  - SLEEP - WAKEUP, autonoma previa controllo
- **I semafori**
  - Variabili intere per indicare lo stato della risorsa
  - L'accesso ad un semaforo, in lettura o modifica, avviene in modo **atomico**, essenziale per evitare problemi di sincronizzazione e corse critiche (mutua esclusione).
- **I contatori di eventi**
  - Non impongono la mutua esclusione

Gestione concorrente ad alto livello

- **Monitor:**
  - Primitiva di sincronizzazzione ad alto livello, implementata in un package separato ed offerta come funzionalità.
  - Ad ogni istante un solo processo può essere attivo in un monitor
  - Il compito di implementare la mutua esclusione è delegato al compilatore.
  - Trasformando tutte le sezioni critiche in chiamate al monitor due processi non possono eseguire contemporaneamente le proprie sezioni critiche.
  - Il monitor implementa **variabili di tipo condizione** associando le operazioni di WAIT e SIGNAL, che segnalano rispettivamente l'impossibilità o la disponibilità a continuare.

Equivalenza tra primitive

- **Scambio di messaggi:**
  - Essenziale in ambiente distribuito, CPU multiple.
  - Protocolli e problematiche (correttezza, sicurezza).

- Le primitive presentate presentano equivalenza semantica

L'assegnazione del Processore: CPU scheduling

- **Scheduler:** quella parte del S.O.che decide a quale dei processi pronti presenti nel sistema assegnare il controllo della CPU

- I processi possono essere o in memoria principale o in memoria secondaria

- **Algoritmo di scheduling:** realizza un particolare criterio di scelta tra i processi pronti; ci sono diversi criteri di scelta:
  - **utilizzo della CPU**
  - **produttività:** throughput come uscite per unità di tempo
  - **tempo di "turnaround":** permanenza nel sistema
  - **tempo di risposta:** ottimizzare i tempi di risposta all'utente
  - **non privilegio (fairness):** garantire a tutti i processi l'uso equo delle risorse

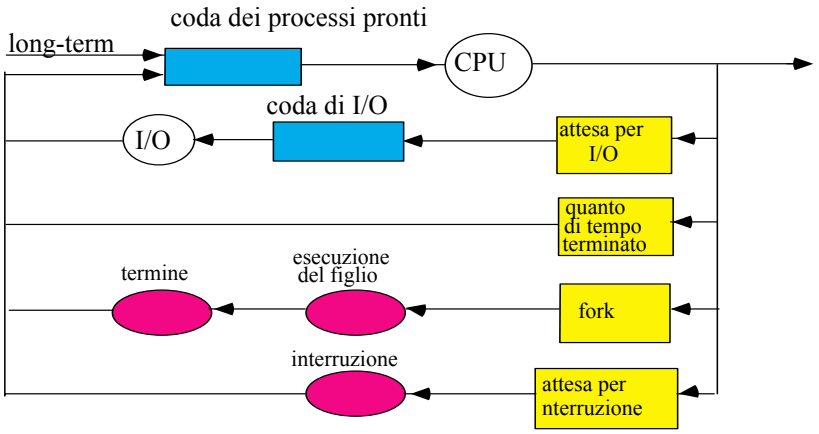
Gli Algoritmi di Scheduling

- **Preemptive:** un processo in esecuzione perde il controllo della CPU anche se logicamente può proseguire
- **Non Preemptive:** un processo in esecuzione prosegue fino al rilascio spontaneo della CPU:
  - terminazione
  - richiesta di I/O
  - sospensione per attesa di evento
- **Round Robin FCFS:** ogni volta che devo mettere in esecuzione un processo prendo il primo in lista
- **Round Robin con priorità:** privilegio i processi con una priorità più elevata
- **Round Robin SJF:** privilegio i processi piccolo che eseguono meno

I Livelli di Scheduling

- **Long Term Scheduling:** determina quali programmi devono essere caricati dalla memoria di massa alla memoria principale:
  - controlla il grado di multiprogrammazione (numero di processi in memoria principale)
  - possibili criteri di scelta (I/O bound, CPU bound)
  - interviene ad intervalli di tempo lunghi con algoritmi più complessi
- **Short Term Scheduling:** seleziona tra i processi pronti in memoria principale quello cui assegnare la CPU:
  - Elevata frequenza di intervento
  - Efficienza piuttosto che precisione nella scelta del processo (basso overhead)
- **Dispatcher:** realizza le operazioni necessarie per assegnare il controllo della CPU al processo selezionato dal short-term scheduler:
  - Cambio di contesto
  - Ritorno allo stato utente
  - Mette in esecuzione il processo selezionato

Diagramma di flusso dello scheduler



Processi Pesanti e Processi Leggeri: Il Thread

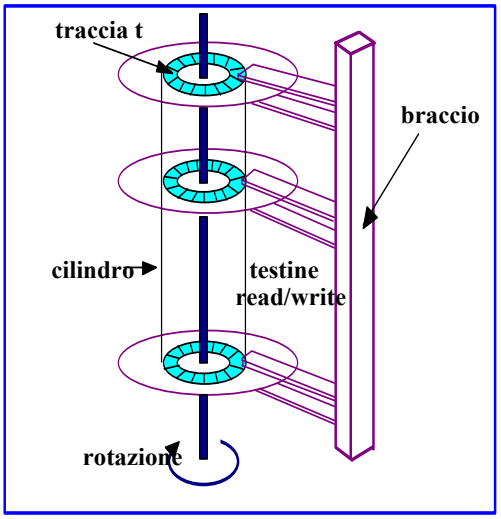
Ci sono due diversi modelli di Processo:

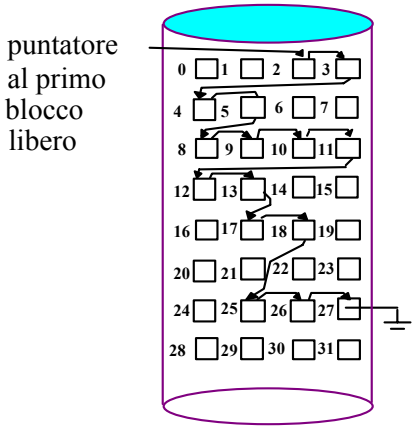
- **Processo Pesante:** ogni processo *vive* nella sua macchina virtuale, i processi sono separati tra loro e possono interagire solo attraverso system calls → **Le Aree di Codice, Heap e Stack sono separate ed autonome**
- **Processo Leggero (o Thread):** tutti i processi *vivono* nella stessa macchina virtuale, *vedono la stessa memoria (condividono lo stesso heap)* ma hanno separati flussi di esecuzione → **Le Aree di Codice e Heap sono condivise, le aree di Stack sono separate**

Il File System

- **Il File System** è la struttura che permette di gestire la memoria di massa attraverso il concetto di File
- Un **File** è un contenitore di dati memorizzato
- Una **Directory** è un file speciale che contiene altri file
- Le memorie di massa sono spesso realizzate da dischi (magnetici o ottici) il file system si occupa di nascondere la complessità della memorizzazione sul un disco dei dati

Come è fatto un disco





Il File System

- **Il S.O.** vede il disco come un vettore di blocchi (le cui coordinate di riferimento sono **cilindro,traccia,settore**)
- Un **File** è quindi una concatenazione di blocchi, una sequenza di terne di coordinate
- Per gestire lo spazio libero esiste una **Bit Map**: una sequenza di bit che rappresentano i blocchi pieni con un "1" ed i vuoti con "0"
  - 1111000110000001111 → è semplice capire quale spazio è libero
  - Non sempre esiste però uno spazio contiguo sufficiente
- Per **Frammentazione** si intende appunto la divisione di un file in diverse catene di blocchi non contigue. La frammentazione riduce drasticamente le prestazioni perché non posso leggere il file in un semplice giro della testina, ma devo seguire un percorso e ricostruire il file in memoria

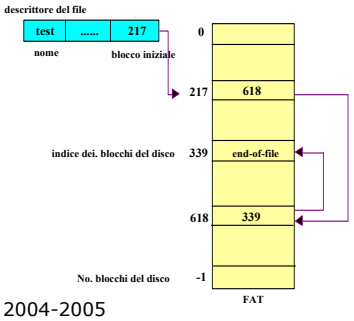
File System
Cache e Prestazioni

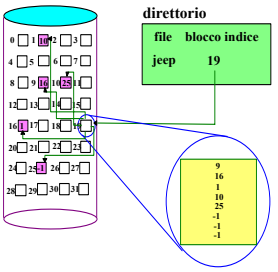
- I tempi di risposta di una CPU si misurano in ns, quelli di un disco in ms: le prestazioni dei dischi incidono di 6 ordini di grandezza di più sulle prestazioni complessive rispetto alla CPU
- Per migliorare le prestazioni devo avere le seguenti condizioni:
  - Cercare di avere sempre in memoria i dati che servono alla elaborazione (RAM lavora in µs)
  - Quando accedo al disco devo cercare di caricare più dati possibile cercando di "prevedere" un uso futuro
- I dischi vengono quindi rimappati dal File System in **Blocchi logici** (più grandi dei blocchi fisici) e la lettura e scrittura avviene solo per blocchi logici (meno letture/scritture e di dimensioni più grandi)
- Esiste un'area di memoria nota come **Cache** che mantiene alcuni blocchi logici in memoria per un accesso più veloce ai dati (se il dato è in cache non serve che io vada sul disco a leggerlo!)
  - Principio dell'uso multiplo del dato
  - Cosa succede se uso la cache con un dischetto?

File System
Struttura dei File

- Se un file è una catena di blocchi, come faccio a sapere quali blocchi lo compongono? Ci sono due soluzioni:

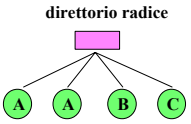
  –Esistono dei blocchi specifici (detti *nodi)* adibiti a contenere la lista dei blocchi appartenenti ad un file





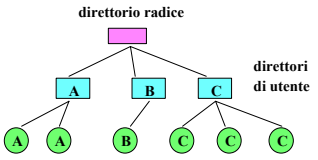
– Ogni blocco di un file in fondo contiene il riferimento al nodo successivo

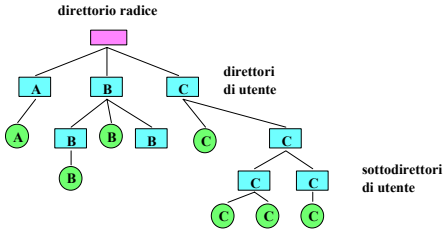
File System
Le Directories



–La struttura è sempre gerarchica: esiste in direttorio radice (root) che contiene una lista di direttori che a loro volta possono contenere altri direttori; ricordo che per il sistema **una directory è un file**

–Come per i processi ho pensato alle macchine virtuali così per il File System posso pensare a degli alberi disgiunti per gli utenti, gli utenti non possono però condividere nessun file

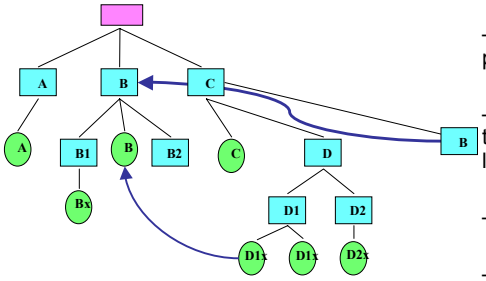




–In un file system di un sistema UNIX esiste il concetto di “home” di un utente come parte di un albero comunque unificato

File System
I link

- Un link è un riferimento ad un altro file: il file system traduce un link come l'accesso diretto al file in oggetto:
  - Posso avere diversi riferimenti allo stesso file in divere cartelle
  - Posso costruire strutture di directory a grafo anziché gerarchiche



–Si posso costruire soluzioni articolate che permettono di gesrie con flessibilità i dati

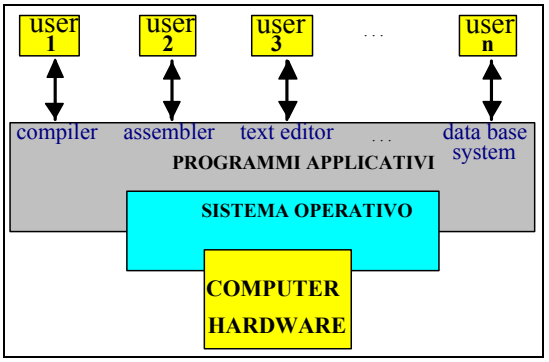
–Attenzione: la creazione di grafi può trasformare alcune procedure normali come la copia in operazioni cicliche infinite!

–In unix i link sono di 2 tipi fisici e simbolici

–In windows si devono usare programmi ad hoc (junction)

Il Sistema Operativo Unix

- **Unix** nasce nei laboratori Bells della At&T i primi anni 70
- Nasce come sistema operativo general purpouse per la gestione dei grossi sistemi mainframe.
- È un sistema **multi utente e multi tasking** basato sul principio della macchina virtuale



Unix
La Memoria Virtuale

- In Unix i processi sono **pesanti**, ogni processo quindi gira in una sua specifica macchina virtuale
- La macchina virtuale offre al processo l'astrazione di **memoria virtuale infinita:** ogni processo ha a disposizione tutti gli indirizzi (tipicamente da 0 a 4Gb nelle macchine a 32 bit e da 0 a 16Eb (16.000.000 Tb) nelle macchine a 64 bit)
- Il sistema si occupa automaticamente di gestire questa memoria virtuale:
  - Lo scheduler attiva le funzioni di mapping dalla memoria virtuale alla fisica
  - Lo SWAP gestisce il disco per supplire alle esigenze di memoria che superano la RAM disponibile
  - La memoria è divisa in pagine, queste pagine, solitamente della dimensione di 4 o 16k forniscono l'unità di allocazione da parte del sistema

Unix
La Filosofia del Sistema

- In UNIX ogni attività è delegata ad un processo di utente applicativo o di sistema che la esegue accedendo alle risorse virtuali del sistema
- La risorsa virtuale fondamentale è il **file system** che ci rappresenta la macchina virtuale completa
- Gli elementi che costiuiscono il sistema sono in realtà solo 3:
  - processore comandi (shell)
  - nucleo (primitive di sistema)
  - linguaggio di sistema (C)
- Tutte le risorse sono mappate attraverso il file system: ogni cosa quindi è visibile **attraverso la astrazione di file**

Unix
Il File System

- FILE COME STREAM DI BYTE: NON previste organizzazioni logiche o accessi a record
- FILE SYSTEM gerarchico
  - ALBERO di sottodirettori come SISTEMA di NOMI come puntatori ai file corrispondenti raggruppati logicamente
  - OMOGENEITÀ dispositivi e file: **TUTTO è file**
- FILE è l'astrazione unificante del sistema operativo
  - file ordinari
  - file direttori
  - accesso ad altri file
  - file speciali (dispositivi fisici) contenuti nel direttorio /dev

Unix
Organizzazione del File System

- NOMI di file per arrivare alle informazioni
  - ASSOLUTI: dalla radice */nome2/nome6/file*
  - RELATIVI: dal direttorio corrente (o nello stesso) *nome6/file* *file*
- Direttorio corrente identificato da **.**
- Padre del direttorio corrente identificato da **..**
- Ogni utente ha un direttorio a default, il direttorio in cui l'utente lavora all'ingresso nel sistema
- I nomi (e la sintassi in generale) sono case-sensitive ("A" è diverso da "a")
- Si usano abbreviazioni nei nomi dei file (*wild card*)
  - * per una qualunque stringa
  - ? per un qualunque carattere
    - file* ==> file, file1, file2, file3, filetemp
    - file? ==> file1, file2, file3
    - *ile ==> file, cile

Unix
La Gestione degli Utenti

- Ogni utente è identificato da un ***username o login name***
- Gli utenti possono accedere contemporaneamente al sistema richiedendo l'accesso attraverso la procedura di *login*: il sistema verifica l'identità dell'utente attraverso una password
- All'interno del sistema ogni utente ha il permesso di "fare" determinate cose: significa cioè che ***ha il diritto*** di accedere a determinate risorse
- Siccome in unix ***tutto è file*** la gestione dei diritti, e quindi della protezione e sicurezza del sistema è affidata al File System
- Gli utenti sono raggruppati in gruppi: i gruppi possono condividere diversi diritti perché fanno ad esempio attività affini e quindi hanno bisogno dell'accesso alle stesse risorse

Unix
I diritti sui File

- Ogni File ha un proprietario (utente)
- Il proprietario di un file ha la possibilità di regolare i diritti di accesso a questo file da parte degli altri utenti
- Ci sono 3 tipi di accessi:
  - Da parte del proprietario stesso **(user)**
  - Da parte di un utente dello stesso gruppo del proprietario **(group)**
  - Da parte di tutti gli altri utenti **(other)**
- Per ogni tipo di utilizzatore, i modi di accesso al file sono:
  - lettura **(r),** scrittura **(w)** ed esecuzione **(x)**
- Ogni file è marcato dal proprietario
  - user-ID del proprietario;
  - group-ID del gruppo;
  - un insieme di 12 bit di protezione

12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	1	1	1	0	0	1	0	0
SUID	SGID	sticky	R	W	X	R	W	X	R	W	X
			User			Group			Others		

Unix
Esempio di direttorio

```
bash-2.05$ ls -al
total 116
drwxrwxr-x   6 fabiot   wwwlia      1024 Oct 16 09:43 .
drwxrwxr-x  42 wwwlia   wwwlia      1536 Sep 29 18:31 ..
-rw--w----   1 fabiot   wwwlia       153 Jun 19  2002 .Xauthority
-rw--w----   1 fabiot   wwwlia     10051 Oct 14 16:36 .bash_history
drwx-w----   2 fabiot   wwwlia       512 Jul 24 18:54 .ssh
-rw-rw-r--   1 fabiot   wwwlia      4583 Jul  9 16:38 DateAppelli.htm
-rw-rw-r--   1 fabiot   wwwlia      4519 Sep  3 20:00 MatDid.htm
-rw-rw-r--   1 fabiot   wwwlia      8667 May 14 13:05 biblio.htm
drwxrwxr-x   3 fabiot   wwwlia      1024 May  5 13:02 fb03
-rw-rw-r--   1 fabiot   wwwlia      3581 Sep  3 20:04 index.html
-rw-rw-r--   1 fabiot   wwwlia      8464 Jul  9 16:39 risultatiA270603.htm
-rw-rw-r--   1 fabiot   wwwlia      7986 Jul  9 16:39 risultatiB270603.htm
drwxrwxr-x   3 fabiot   wwwlia      1024 Feb 18  2003 sist02
drwxrwxr-x   2 fabiot   wwwlia       512 May 28 12:50 tech03
bash-2.05$
```


Unix
I bit di controllo esecuzione

Nella lista dei bit di proprietà di un file ci sono 3 bit speciali:

- SUID bit set-user-id bit (bit numero 12)
  - Se a 1 in un file di programma eseguibile, l'utente in esecuzione (il processo) è **assimilato al creatore di quel file**, solo per la durata della esecuzione
    - Questo permette al programma di accedere a risorse di esclusiva proprietà del creatore del programma
    - Senza suid, un programma potrebbe causare errori, durante l'esecuzione per operazioni di lettura/scrittura su altri file di cui l'utente non ha i diritti relativi
    - Per esempio: **passwd**, è il programma che cambia la password di un utente, per fare questa operazione modifica il file **/etc/passwd** di proprietà esclusiva del superutente (o utente root), che non può essere modificato direttamente. Solo il suid lo rende possibile
- SGID bit set-group-id bit (bit numero 11)
  - come SUID bit, agisce però sui diritti di gruppo
- sticky bit (bit numero 10)
  - il sistema tende a mantenere in memoria principale la immagine del programma, anche se non è in esecuzione
  - Questa indicazione è trascurata nelle ultime versioni dal SO

Unix
La Shell: il processore dei comandi

La Shell è un programma di sistema che ha il compito di accettare i comandi dell'utente da terminale o file comandi (script) operando opportune trasformazioni:

- La shell mette in esecuzione i comandi uno dopo l'altro:

```
loop forever
    <accetta comando da console>
    <esegui comando>
end loop;
```

- Un comando, o un programma invocati in una shell ottengono dalla shell stessa un ambiente di esecuzione o environment
  - la shell gestisce lo stato applicativo della macchina virtuale del sistema operativo
- Le shell sono *"rientranti", ovvero presenti in memoria contemporaneamente e in esecuzione contemporaneamente facendo riferimento allo stesso codice, e possono essere chiamate ricorsivamente*
- *Ogni shell gestisce un processo separato → quasi tutti i programmi eseguono in una shell separata*

Unix
La Shell: Esecuzione di un Comando

- lo shell attivo mette in esecuzione un secondo shell:
  - Il secondo shell
    - esegue le sostituzioni (wild cards)
    - ricerca il comando
    - esegue il comando
- Lo shell padre attende il completamento dell'esecuzione del sottoshell (comportamento sincrono)

AMBIENTE DI SHELL:

- insieme di variabili di shell
  - la variabile PATH indica i direttori in cui ricercare ogni comando da eseguire
  - la variabile HOME indica il direttorio di accesso iniziale
  - Altre veriabili utente (es: CLASSPATH)

Unix
La Shell: I comandi per il File System

Sintassi:

- comando [-opzioni] [argomenti] <CR>

Un comando o termina con un ritorno, oppure è separato con ; da altri comandi nella stessa linea

DIRETTORI

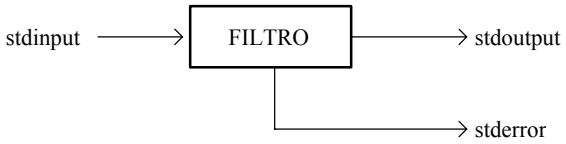
- mkdir <nomedir>
- rmdir <nomedir>
- cd <nomedir>
- Pwd
- ls [<nomedir>/<nomefile>]

TRATTAMENTO FILE

- ln <oldfile> <newfile>
- cp <filesorg> <filedest>
- mv <vecchionomefile> <nuovonomefile>
- rm <nomefile>
- cat <nomefile>
- file <nomefile>
- pr <file1> <file2> <filen>

Unix
La Shell: Ridirezione dell' I/O

- Tutti i Comandi Unix sono **Filtri:**
  - Eseguono un elaborazione partendo da un input e generano un output
    - Standard input → console
    - Standard output → console
    - Standard error → console



- ridirezione dell'input <comando> < <fileinput>
- ridirezione output <comando> > <fileoutput>
  <comando> >> <fileoutput>
- Pipeline <comando> | <comando>

Unix
La Shell come linguaggio

- La shell di UNIX definisce un vero e proprio **linguaggio di programmazione**, le cui caratteristiche principali sono:
  - Uso di **variabili**
  - **Procedure** con **passaggio dei parametri** di ingresso
  - Istruzioni di **controllo di flusso**
  - Gestione della **concorrenza**
  - Gestione di eventi asincroni, cioè i **segnali**