SISTEMI OPERATIVI

Il software può essere diviso in due grandi classi:
- **I programmi di sistema:** che gestiscono le operazione del sistema di elaborazione.
- **I programmi applicativi:** che risolvono i problemi dei loro utilizzatori.

L'insieme dei programmi di sistema prende il nome di ***Sistemi Operativo.***
Lo scopo del sistema operativo è quello di rendere agevole l'interfaccia utente, di permettere di avere un accesso simultaneo alla stessa macchina da parte di più utenti, di eseguire più processi contemporanemante e di gestire le risorse del sistema di elaborazione.
In particoalre il Sitema Operativo gestisce:
- La memoria (**RAM**)
- La memoria di massa (**file system**)
- I processi

Struttura dei Sistemi Operativi

I sistemi operativi sono costituiti da un insieme di moduli, ciascuno determinato a svolgere una determinata funzione. I vari moduli interagiscono tra di loro secondo regole precise al fine di realizzare le funzionalità di base della macchina.

Processi

Processo = programma in esecuzione

Il **programma** è un'entità passiva (un insieme di byte contenete le istruzioni che devono essere eseguite).
Il **processo** è un'entità attiva :
- E' l'unità di lavoro/esecuzione all'interno del sistema. Ogni attività all'interno del SO è rappresntata da un processo.
- E' L'istanza di un programma in esecuzione.

Il gestore dei processi :

Il gestore dei processi non è nient'altro che il modulo che si occupa di **controllare, incronizzare, interrompere e riattivare** dei programmi in esecuzione cui viene assegnao un processore.

Il programa che si occupa della distribuzione del tempo di CPU tra i vari processi attivi è comunemente chiamato ***Scheduler,*** e si occupa anche di gestire la cooperazione tra le varie CPU in caso di elaboratori multi-processori. Esistono vari algoritmi di scheduling che tengono conto di varie esigenze e che possono essere più indicati in alcuni contesti piuttosto che altri.
La scelta dell'algortimo dipende da 5 criteri:
- ***Utilizzo del processo:*** devono essere ridotti al minimo i possibili tempi morti.
- ***Produttività :*** il numero di processi completati in un determinato periodo.
- ***Tempo di completamento :*** tempo che intercorre tra la sottimissione del processo ed il completamento della sua esecuzione.
- ***Tempo d'attesa :*** tempo in cui un processo pronto per l'esecuzione rimane in attesa della CPU
- ***Tempo di risposta :*** tempo che trascorre tra la sottomissione del processo e l'ottenimento della prima risposta.

Le politiche di Schedulazione sono raggruppabili in due categorie:
- ***Preemptive :*** la CPU in uso da parte di un processo può essere tolta e passata ad un altro in un qualsiasi momento.
- ***Non Preemptive :*** una volta che un processo ha ottenuto l'uso della CPU non può essere interrotto fino a che lui stesso non la rilascia.

I Sistemi Operativi che gestiscono l'esecuzione di un solo programma per volta sono catalogati con il nome di **mono-tasking.** Nei sistemi **mono-tasking** non è possibile interromper eun esecuzione di un programma per assegnare la CPU a un altro.

I Sistemi Operativi che permetto l'esecuzione di più programmi contemporaneamente sono definiti **multi-tasking.** Un programma può essere interrotto per iniziarne una altro. Ne sono un esempio Windows e Linux.
Un evoluzione dei sistemi multi-tasking sono i sistemi **time sharing** ogni programma in esecuzione viene eseguito ciclicamente per piccoli ***quanti di tempo.***
Se la velocità del processore è sufficentemente elevata si ha l'impressione di un'evoluzione contemporanea dei processi.

Gestore della memoria

Il gestore della memoria è quel modulo del SO incaricato di assegnare la memoria ai vari task (per eseguire un task è necessario che il suo indirizzo sia caricato in memoria). La complessità del gestore della memoria dipende dl tipo di SO.
Nei programmi multi-tasking più prgorammi possono essere caricati contemporanramente in memoria. Molto psesso la memoria non è sufficente per contenere completamente tutto il codice dei vari task.
Si può **simulare** una memoria più grande tenendo nella memoria del sistema solo le parti di codice e dei dati che servono in quel momento, usando così il concetto di **memoria virtuale.**

I dati dei programmi non in esecuzione possono essere tolti dalla memoria centrale ed inseriti all'interno dell'**area di swap.**

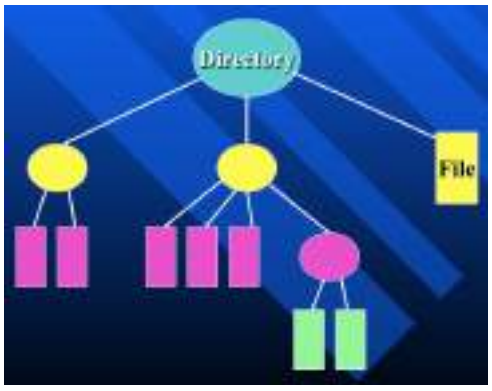
Gestore del File System

Il gestore del File System è quel modulo del sistema operativo incaricato di gestire le informazioni memorizzate sui dispositivi di memoria di massa. Il seguente gestore deve garantire la *correttezza* e la *coerenza* delle informazioni.

Nei sistemi multiutente, il gestore del File System deve mettere a disposizione dei meccanismi di protezione in modo tale da consentire agli utenti di proteggere i proprio dati dall'accesso da parte di altri utenti non autorizzati.
Le funzioni tipiche che devo svolgere sono:

- Fornire un meccanismo per **l'identificazion**e dei files
- Fornire opportuni metodi per **accedere** ai files
- Rendere **trasparente** la **struttura fisica** del supporto di **memorizzazione**
- Implementare **meccanismi di protezione** dei dati

Quasi tutti i Sistemi Operativi utilizzano un'organizzazione **gerarchica** dei File System.
L'elemento usato per raggruppare più file system è la **directory.**



Gestore dei dispositivi di I/O

Il gestore dei dispositivi di I/O è quel modulo del SO incaricato di assegnare i dispositivi ai task che ne fanno richiesta e di conrollare i dispositivi stessi. Da esso dipende la qualità e il tipo di periferiche riconosciute dl sistema.
Il controllo dei dispositivi di I/O avviene attraverso speciali programmi detti **Device Driver**.

Questi programmi implementano normalmente le seguenti funzioni:
- Rendono trasparenti le caratteristiche fisiche di ogni dispositivo.
- Gestiscono le comunicazioni dei segnali verso ogni dispositivo.
- Gestiscono i conflitti nel caso in cui due o più task vogliono accedere contemporaneamente allo stesso dispositivo.

Interfaccia Utente

Tutti i sistemi operativi implementano dei meccanismi per agevolare l'utlizzo del sistema da parte dell'utente. L'insieme di questi meccanismi di accesso al computer prende il nome di ***Interfaccia Utente.*** L'interfaccia utente si compone di due parti:
- **L'interfaccia testuale:** interprete dei comandi
- **L'interfaccia grafica:** l'output dei vari programmi viene visualizzato in maniera grafica all'interno di finestre e la presenza di disegni rende più intuitivo l'uso del calcolatore.

In passato si sviluppava sistemi operativi proprietari per le loro architetture.

La tendenza attuale è quella di Sistemi Operativi eseguibili su diverse piattaforme.

Linguaggio C

Tipo di opzione	Descrizione
\n	Ritorno a capo
\t	Tabulazione orizzontale
\b	Tabulazione verticale
\a	Torna indietro di uno spazio
\f	Salto pagina

Tipi di dichiarazione	Rappresentazione
Char	Carattere (es. 'a)
Int	Numero intero (es. 3)
Short	Numero intero corto
Long	Numero intero lungo
Float	Numero reale "corto" (es 14.4)
Double	Numero reale "lungo"

In C **non esiste il tipo boolean**: Si usa la convenzione che lo zero rappresenta il valore **falso** e l'uno il valore **vero** (tutti i valori diversi da zero rappresentano il vero)

printf ("<stringa>" , <elenco argomenti>) ; //permette di stampare in output
scanf ("<stringa>" , <elenco argomenti>) ; //permette di far inserire degli elementi dall'utente

scanf ("%d" , &n) ; // %d tipo d'argomento da inserire, &n assegna alla variabile n il valore inserito
printf ("Risultato: %d", n) ; // stampa il risultato inserito in n nel punto indicato

* → moltiplicazione
\+ → somma
\- → differenza

Sintassi da utilizzare	Descrizione
%d	Dati di tipo int
%lf %l %f	Dati di tipo double Dati di tipo long Dati di tipo float
%c	Dati di tipo char
%s	Dati di tipo stringhe

/ → divisione senza resto
% → resto della divisione
++ → incremento
— → incremento

Operatori logici	Descrizione
&&	AND
||	OR
!	NOT

Operatori relazionali	Descrizione
x == y	Testa se il valore di x è uguale a y
x > y	Testa se x è maggiore di y
x >= y	Testa se x è maggiore uguale di y
x < y	Testa se x è minore di y
x <= y	Testa se x è minore uguale di y
x != y	Testa se x è diverso da y

Istruzione di controllo condizionali :

if (<espressione>) <istruzione1>
else <istruzione2>

Se <espressione> è vera allora si verifica <istruzione1> altrimenti si verifica <istruzione2>.

switch (<espressione>) {
case <costante1> : <istruzione1> [break]
case <costante2> : <istruzione2> [break]
...
default : <istruzione>
}

Se <espressione> vale <costanten> viene eseguita <istruzionen> altrimenti si verifica <istruzione>.

Istruzione di controllo iterative :

While (<epsressione>) {
<istruzione>
}

Fino a che <espressione> è vera allora si verifca istruzione.

for (<inizializzazione> , <condizione> , <incremento>) {
<istruzione>
}

Inizializzo una variabile eseguo <istruzione> e la incremento fino a quando non è verificata la condizione.

Puntatori

I puntatori permettono la gestione di strutture dinamiche. Un puntatore è una variabile che contiene l'indirizzo di memoria di un'altra variabile.

& l'operatore applicato ad una variaible restituisce il puntatore ad essa.
***** restituisce la variabile puntata.

Array

Un Array è una sequenza di elementi omogenei.

int <nomeArray>[<lunghezzaArray>] ;

Il primo elemento di ogni vettore è l'elemento **zero**

Gli Array occupano dello spazio in memoria. Il programattore specificandone la lunghezza permette al computer di stabilire una determinata quantità di memoria.

E' possibile definire un vettore multidimensionale nella seguente maniera che permette di determinare una matrice:

int <nomeArray>[<lunghezza1>] [<lunghezza2>] ;

Gestione della memoria:

Durante l'esecuzione di un progeamma C la memoria viene gestita in due diverse maniere:

- **Gestione statica :** viene allocata dal sistema operativo un'area di memoria fissa per tutta l'esecuzione del programma.
- **Gestione dinamica :** vengono allocate due aree di memoria che vengono usate quando necessario, e rese disponibli per successivi usi:
    1. *Lo Stack :* quando una funzione viene invocata vengono allocate automaticamente tutte le variabili locali e i parametri attuali sullo stack in un record di attivazione, successivamente quando l'esecuzione della funzione termina, il record di attivazione viene cancellato e lo stack viene riportato allo stato in cui era alla prima invocazione.
    2. *Lo Heap :* la gestione viene lasciata al programmatore mediante creazione e distruzione dinamica delle variabili.

L'allocazione dinamica della memoria viene mediante 4 diverse funzioni : **malloc** (alloca in maniera dinamica un numero preciso di byte) **, calloc** (alloca in maniera dinamica una precisa quantità di memoria per **n** oggetti) **, realloc** (rialloca uno spazio di memoria precedentemente allocato) **, free** (dealloca la memoria)**.**

La funzione free andrebbe SEMPRE usata alla fine di ogni programma su ognu variabile allocata dinamicamente.

La **gestione della memoria secondaria** consente la memorizzazione permanente dei dati e la loro condivisione tra programmi diversi. Ciò ha dei vantaggio quali:

- La mole di dati non consente la gestione della memoria principale.
- I dati hanno un'esistenza indipendente dalle esecuzioni dei programmi che li usano.

In memoria secondaria i dati sono salvati in file.

File :

Un file è un contenitore di informazioni permanente. La vita di un file è indipendente dalla vita del programma che me fa uso, quindi possiamo dire che un file continua a vivere anche al termine dell'esecuzione del programma.

A livello logico è una sequenza (insieme) di record.

Il **metodo di accesso** a un file è la tecnina usata dal programma per accedere ai record del file:

- **fopen:** *apre* un file ovvero lo predispone ad una certa elaborazione. Se l'apertura del file non ha successo restituisce *NULL.*
    - **r [read] :** il file deve già essitere e viene apero in lettura.
    - **w [write] :** il file viene creato se non esiste o viene sovrascritto se già esiste

- **a [append] :** il viene viene creato se non esiste, viene aperto in scrittura con inserimenti solo in fondo al file.
- **+ [update] :** si usa in combinazione con una delle prime 3 modalità, permette l'aggiornamento del file.
- **b [binary] :**si usa in combinazione con una delle prime modalità per gestire il file in formato binario .

FILE * fopen (nomefile, modalità)

- **fclose:** *chiude* un file, ovvero rilascia chiudendo il relativo canale di comunicazione. Se la chiusrìura del file non ha successo restituisce *NULL.*

int* fclose (FILE * fp)

- **fputc:** scrive un singolo carattere in un file. Restituisce il carattere stesso se l'operazione è andata a buon fine, altimenti restituisce **EOF.**

int * fputc (int c, FILE * fp)

- **fputc:** scrive una stringa in un file senza aggiungere il carattere di fine stringa. Restituisce 0 se l'operazione è andata a buon fine, altimenti restituisce **EOF.**

int * fputs (char *s, FILE * fp)

- **fgetc:** legge un carattere di file specificato e restituisce il carattere letto, altrimenti **EOF.**

int * fgetc (FILE * fp)

- **getc:** legge un insieme di caratteri da file specificato e restituisce i caratteri letti, altrimenti *NULL.*

char * fgets (char *s, int n, FILE * fp)

- **fscanf:** legge caratteri da un file scendo il formato specificato e assegna i calori letti ai suoi successivi argomenti. Restituisce il numero di caratteri se l'operazione è andata a buon fine , altimenti restituisce **EOF.**

int * fscanf (FILE * fp, str_cont, elementi)

- **fprintf:** scrive caratteri in un file secondo il formato specificato. Restituisce il numero di caratteri scritti se l'operazione è andata buon fine altrimenti restituisce un valore negativo.

int * fprintf (FILE * fp, str_cont, elementi)

- **fread:** legge da un file un numero specificato di oggetti di una certa ampiezza e li memorizza in un vettore. Restituisce il numero di oggetti letti.

int fread (void *buf, int size, int count, FILE *fp)

- **fwrite:** scrive su un file un numero specificato di oggetti di una certa ampiezza prelevandoli da un vettore. Restituisce il numero di oggetti letti.

int fwrite(void *buf, int size, int count, FILE *fp)

- **fseek:** determina una posizione su un file. Le successive istuzioni I/O sul file operano a partire da questa posizione. Si specifica uno spiazzamento **s** e un origine **o.** La posizione è a **s** byte a partire da **o.** La funzione restituisce un valore nullo in caso di errore.

int fseek (FILE *fp, long s, int o)

- **ftell:** restituisce la posizione corrente su un file (0 se all'inizio), oppure, in caso di errore un numero negativo.

long ftell (FILE *fp)

- **fsetpos:** posiziona il file nella posizione memorizzata in una variabile di memoria. Restituisce un valore nullo in caso di errore.

int festpos (FILE *fp, fpos_t *crrent pos)

- **fgetpos:** memorizza in una variabile di memoria la posizione corrente su un file. Restituisce un valore nullo in caso di errore.

int fgetpos (FILE *fp, fpos_t *crrent pos)

Quando si rileva un errore in una operazione di accesso a file viene aggiornato l'indicatore di stato. Inoltre la variabile intera **errno** registra un numeri che da indicazione sul tipo di errore.

Linux

Con il termine **Kernel** indichiamo il cuore del sistema operativo e in particolare si occupa di **gestire :**
- le **comunicazioni** con l'hardware del sistema.
- i **file system** e la **memoria.**
- l'**accesso** alle risorse da parte dei processi.

Il **Major Number** rappresenta il valore più alto della revisione del kernel. Se il Major Number è pari allora viene considerato **stable,** se dispari viene considerato **development.**

Il **Revision Number** indica la revisione corrente. E' un numero progressivo che parte da 0. Tra due revisioni possono passare pochi giorni o varie settimane.

LINUX	***MACINTOSH***	***WINDOWS***
Gli utenti Linux hanno il benificio di avere bassi requisiti di risorse. Il seguente sistema operativo può essere installato su qualsiasi computer, fatta eccezione per i computer Macintosh che hanno al loro interno Mac OS X	I computer Macintosh hanno istallato al loro inteerno Mac OS X. Sono computer molto più costosi di un semplice computer, può addirittura arrivare a costare il doppio di un Windows Pc	Il Sistema Operativo di Windows è un sistema versatile che può essere installato su qualsiasi PC.

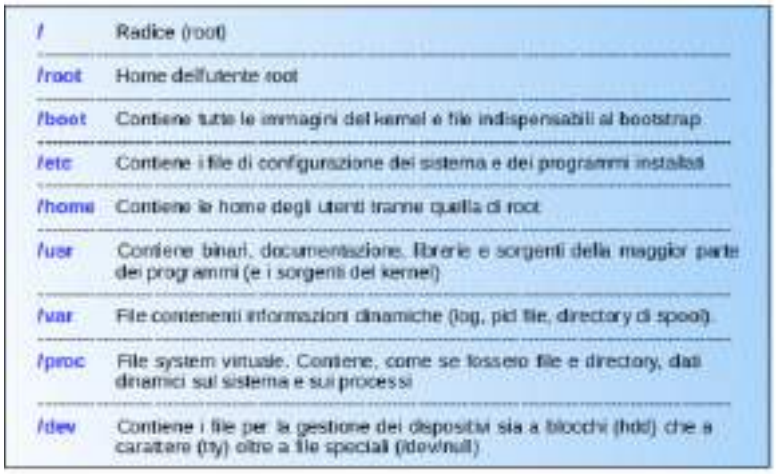
In ambienti Unix esistono differenze fra i vari utenti definite dai **permessi, dall'accesso ai file e dai comandi** che un utente può lanciare.
- I semplici utenti possono **scrivere, leggere e modificare file** solo all'interno del loro ambiente, e possono lanciare semplici comandi che **non** influenzano sulla configurazione del sistema.
- L'utente **root** è l'unico che può accedere completamente alle risorse del sistema, in fase di installazione una macchina Linux consiglia di scegliere una password di rott anche piuttosto complicata e di creare immediatamente un utente normale con il quale operare per tutte le attività **non amministrative.**

Per **file system** si intende l'astrazione (metodi o protocolli) con cui si organizza i file su un supporto fisico ad accesso casuale. Le informazioni inerenti un oggetto di un file system sono contenute in un **inoide** che viene identificato da un numero progressivo e descrive le caratteristiche di un determinato oggetto. Un sistema Linuz è dotato di una **directory principale** chiamata **root** ed indica con / sotto la quale si trovano tutte le altre directory o tutti gli altri file system eventualmente montati sul sistema.
Il principo è radicalmente diverso da quello presente nel mondo Windows, dove ogni device o risorsa ha un suo nome o lettera identicativa al cui interno si trovano le directory del relativo file system.

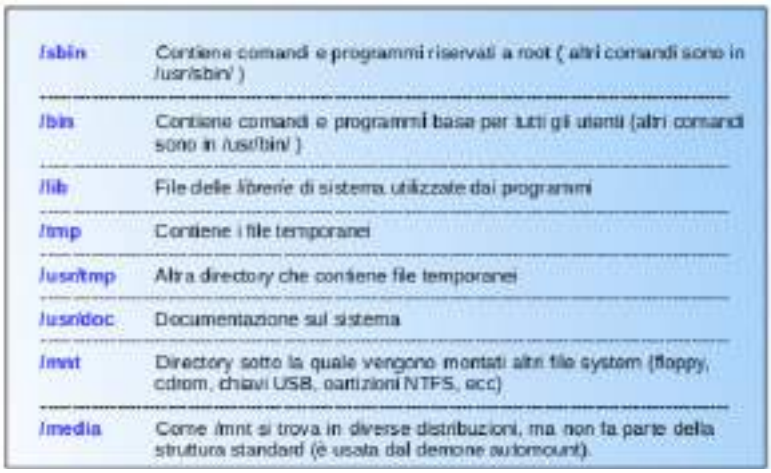
Il Terminale è un interfaccia testuale del sistema operativo che permette di eseguire diversi comandi inseriti da tastiera. Lo ***shell*** è una parte del sistema operativo che permette agli utenti di interagire con il sistema operativo, impartendo comandi e richiedendo l'avvi odi altri prgorammi . Esistono varie Shell disponibili, quella più comunemente usata prende il nome di ***bash.***

Uno script di ***shell bash*** è un file di testo che : contiene comandi shell, inizia con la stringa

#!/*bin*/*bash (* i primi due caratteri indicano che il file è uno script o meglio un programma interpretato, la stringa successiva è il pathname dell'interprete per il programma)**,** e ha permesso di esecuzione.



/	Radice (root)
/root	Home dell'utente root
/boot	Contiene tutte le immagini del kernel e file indispensabili al bootstrap
/etc	Contiene i file di configurazione del sistema e dei programmi installati
/home	Contiene le home degli utenti tranne quella di root
/usr	Contiene binari, documentazione, librerie e sorgenti della maggior parte dei programmi (e i sorgenti del kernel)
/var	File contenenti informazioni dinamiche (log, pid file, directory di spool)
/proc	File system virtuale. Contiene, come se fossero file e directory, dati dinamici sul sistema e sui processi
/dev	Contiene i file per la gestione dei dispositivi sia a blocchi (hdd) che a carattere (tty) oltre a file speciali (/dev/null)



/sbin	Contiene comandi e programmi riservati a root (altri comandi sono in /usr/sbin/)
/bin	Contiene comandi e programmi base per tutti gli utenti (altri comandi sono in /usr/bin/)
/lib	File delle *librerie* di sistema utilizzate dai programmi
/tmp	Contiene i file temporanei
/usr/tmp	Altra directory che contiene file temporanei
/usr/doc	Documentazione sul sistema
/mnt	Directory sotto la quale vengono montati altri file system (floppy, cdrom, chiavi USB, partizioni NTFS, ecc)
/media	Come /mnt si trova in diverse distribuzioni, ma non fa parte della struttura standard (è usata dal demone automount).

Il sistema operativo Linux permette tre diverse relazioni con i file in base ai permessi che si possiedono:

- ***r :*** permette la semplice lettura
- ***w :*** permette di scrivere / modificare quel determinato file
- ***x :*** permette di eseguire il file

In relazione ai file possiamo definire tre diversie composizioni di persone che possono lavorare sui file. Si può essere: **proprietario** (una singola persona che possiede il file), **gruppo** (ogni persona appartenente al gruppo ha la proprietà del file) , **others** (tutte le altre persone che non possiedo direttamente il file perchè non sono ne proprietari ne fanno parte di un gruppo) .

Root User è un supervisore che ha qualsiasi permesso su qualsiasi documento. Coloro che non sono i possessori del root account vengono denominati ***Normal User*** e hanno permessi limitati. Ma in un qualsiasi momento un Normal User può *salire* di livello prendendo la nomina di Root User con ilseguente comando :

su < parameters > username

Nella seguente maniera un Normal User diventa un Root User, ma può tornare ad essere un normal user con il comando ***exit.***

Non è necesario che un utente diventi root per assumere determianti privilegi. Basta usare il comando **sudo ,** con il seguente comando senza diventare root si possiedo gli stessi privilegi.

COMANDI TEMRINALE LINUX

Una Lista di comandi uò essere eseguita su un'unica linea di comando semplicemente usando dei caratteri speciali quali:

- **comando1 ; comando2 :** indipendentemente da come terminano i due comandi vengono eseguuiti contemporanemente
- **comando1 || comando2 :** il comando 2 entra in esecuzione solo se comando1 è termianto con exit status diverso da 0.
- **comando1 & comando2 :** il comando1 viene eseguito in background, il comando2 in foreground.

- **comando1 && comando2 :** il comando1 viene eseguito sempre, mentre il comando2 solo se il primo è terminato con successo.

Il comando **time** avvia un programma che quando termina visualizza sullo standard error il tempo impiegato per eseguirlo diviso in tre valori:
- **real :** il tempo di esecuzione reale.
- **user :** il tempo di CPU utente (il tempo impiegato dalla CPU per eseguire le istruzioni **non** di sistema di programma).
- **sys :** il tempo di CPU di sistema (il tempo impiegato dalla CPU per eseguire le istruzioni del sistema di programma).

real = user + sys + waiting
waiting = I/O waiting time + idle time

SINTASSI :
time [opzioni] [- -] comando [arg1 [arg2] ...]
ES : time sort file.txt > file_ordinato.txt

COMANDO	*DESCRIZIONE*
ls	visualizza il contenuto della directory corrente.
man [comando]	visualizza il manuale di istruzione di *comando* si torna al terminale con il comando **q**
touch nome file { invio }	aggiorna la data del file o se il file non esiste ne crea uno vuoto
mkdir nome directory { invio }	crea una directory denominandola *nome directory*
mkdir - p work/completed/2001 { invio }	crea la directory 2001 e tutte le directory superiori se non esistono
cd nome directory { invio }	cambia directory
cd { invio }	rende come corrente la directory *home*
pwd { invio }	visualizza il PATH in cui si trova
ls -R { invio }	visualizza il contenuto della directory e delle sotto directory.
cp file_sorgente file_destinazione { invio }	vopia il *file_sorgente* in *file_destinazione*
mv file_sorgente file_destinazione { invio }	rinomina o sposta il *file_sorgente* in *file_destinazione*
ls -a	visualizza i file nascosti, quelli in cui il primo carattere del nome è un punto (.)
find percorso -name nome_file { invio }	visualizza tutti ifile che si trovaano sotto *percorso* aventi *nome_file*
find /tmp -size -10 000k { invio }	visualizza i file memorizzati sotto */tmp* aventi dimensione maggiore di 10 000 K
find /home -user topolino { invio }	visualliza i file memorizzati sotto /home di proprietà di topolino
which programma { invio }	visualizza il percorso completo di dove si trova il comando

COMANDO	*DESCRIZIONE*
grep nome { invio }	filtra le righe che contengono la parola *nome*
more programma { invio }	interrompe la visualizzazione quando si riempo lo schermo ed attende la pressione di un tasto per proseguire
clear { invio }	ripulisce il teminale
ls -l [file]	permette di vedere i vari permessi
chmod { permission } { path }	permette di cambiare i permessi
chown { new owner } { file } ***chown { new owner : new griup } { file }***	cambia il proprietario di un file cambia proprietario ed associa un gruppo al file
su < parameters > username	l'utente *username* diventa root e torna allo stato normale con il comando **exit**
sudo < parameters > username	si assumono gli stessi privilegi del root
echo	permette l'interazione con il sistema variabile: • **$SHELL** : restituisce tutte le stringhe che nelle righe successive vengono riportate dopo *echo* • **$DE** : indica gli ambienti usati dal desktop • **$PATH** : contiene tutte le directory in cui il proprio sistema operativo ha eseguoro dei file • **$HOME** : contiene il path della home directory • **$PWD** : contiene il path della directory in cui si sta lavorando
cat filename	visualizza il contenuto del filename
cat filename1 filename2 filename3 { invio }	visualizza la concatenazione dei tre file
cat filename > newfile { invio }	copia il contenuto del filename in newfile (**sovrascrivendolo**)
cat filename >> newfile { invio }	appende il contenuto del filename alla fine del contenuto del newfile
cat - > newfile { invio }	tutto quello che digito da tastiera viene inserito all'interno di newfile
cat -n filename { invio }	nello stampare il contenuto del filename vengono numerate le righe
head	stampa le prime 10 linee, numero che può essere modificato, modificando il comando
tail	stampa le ultime 10 linee, numero che può essere modificato modificando il comando
sort	ordinerà l'input
nl	numera le righe
wc	comando che conta le parole
cut	permette di estrarre dei contenuti dal testo

COMANDO	DESCRIZIONE
time	tempo impiegato per l'esecuzione di un programma
alias ***alias nome_alias = ' comando '***	comando di shell che permette di definire altri comandi
du ***du parametri file***	misura l'uso del disco da parte di un file • **du <file>** restituisce il numero di blocchi • **du -h <file>** restituisce il numero di blocchi in termine di byte usati • **du -sh <dir>** restituisce la somma degli usi dei vari file contenuti nella directory.
df ***df parametri file***	misura l'uso del disco e lo spazio libero da parte di una directory • **df -h <dir>** restituisce il numero di blocchi in termine di byte usati
tar **tar cvf <archive> <files or directories>** **tar xfv <archive>**	genera file utili all'archiviazione/ backup • **cvf** crea • **xfv estrae**

Modifiche del contenuto

sed **sed actions files**	permette di ***editare*** il testo. Se non si specifa un comando verrà restituito il file senza modifiche

Operatore	Nome	Effetto
`[indirizzo]/p`	*print*	Visualizza [l'indirizzo specificato]
`[indirizzo]/d`	*delete*	Cancella [l'indirizzo specificato]
`s/modello1/modello2`	*substitute*	Sostituisce in ogni riga la prima occorrenza della stringa modello1 con la stringa modello2
`[indirizzo]/s/modello1/modello2`	*substitute*	Sostituisce, in tutte le righe specificate in indirizzo, la prima occorrenza della stringa modello1 con la stringa modello2
`[indirizzo]/y/modello1/modello2`	*transform*	sostituisce tutti i caratteri della stringa modello1 con i corrispondenti caratteri della stringa modello2, in tutte le righe specificate da indirizzo (equivalente di tr)
`g`	*global*	Agisce su tutte le verifiche d'occorrenza di ogni riga di input controllata

```
sed '4,$d' /etc/passwd
```

- *stampa a video soltanto le prime 3 righe del file `/etc/passwd`: `d` è il comando di cancellazione che elimina dall'output tutte le righe a partire dalla quarta (`$` sta per l'ultima riga del file); quindi l'azione richiede di cancellare le ultime righe del file a partire dalla quarta.*

```
> sed '4,$d' /etc/passwd
##
# User Database
#
```

```
sed 3q /etc/passwd
```

- *stesso effetto del precedente comando: in questo caso `sed` esce dopo aver elaborato la terza riga (`3q`); l'azione consiste quindi di elaborare tutte le linee del file fino alla terza.*

```
> sed 3q /etc/passwd
##
# User Database
#
```

```
sed /sh/y/:0/_%/ /etc/passwd
```

- *sostituisce in tutte le righe che contengono la stringa* `sh` *il carattere* `:` *con il carattere* `_` *ed il carattere* `0` *con il carattere* `%`

```
> cat /etc/passwd
...
root:*:0:0:System Administrator:/var/root:/bin/sh
...
> sed /sh/y/:0/_%/ /etc/passwd
...
root:*_%_%_System Administrator:_/var/root:_/bin/sh
...
```

- *cancella l'ottava riga dell'input* `/etc/passwd`

```
sed 8d /etc/passwd
```

- *cancella tutte le righe vuote dell'input* `/etc/passwd`

```
sed '/^$/d' /etc/passwd
```

- *cancella dall'inizio dell'input* `/etc/passwd` *fino alla prima riga vuota compresa*

```
sed '1,/^$/d' /etc/passwd
```

- *Visualizza solo le righe in cui è presente "Jones" (con l'opzione* `-n`*) dell'input* `/etc/passwd`

```
sed -n /Jones/p /etc/passwd
```

- *Cancella tutti gli spazi che si trovano alla fine di ogni riga dell'input* `/etc/passwd`

```
sed 's/ *$//' /etc/passwd
```

- *Sostituisce con "Linux" la prima occorrenza di "Windows" trovata in ogni riga dell'input* `/etc/passwd`

```
sed 's/Windows/Linux' /etc/passwd
```

Altri esempi nel pdf della lezione 22 ultime slides.

Comando AWK

AWK **awk 'script' nomefile** **awk -f fileprogramma nomefile**	filtro generico del file di testo che permette di trovare sequenze di caratteri di file di testo e di effettuare una serie di azioni sulle linee corrispondenti AWK elabora il file nomefile secondo le istruzioni contenute in script o in fileprogramma.

Fondamenti del comando AWK :

- Quando il pattern è soddisfatto viene eseguita l'azione.
  I pattern possono essere:
  - Semplici espressioni regolari racchiuse tra **/**
  - Un espressione logica o ancora le espressioni **begin** ed **end** (vengono ritenute vere rispettivamente prima di incominciare a leggere i file in input e dopo averlo esaminato).
- Suddivisione del file di testo in campi (fields) e linee (records)

- *Una istruzione AWK appartiene ai seguenti tipi:*
  - *Assegnazione:* `var = exp` *dove* `exp` *calcola il valore di un'espressione e lo assegna alla variabile* `var` *(es:* `doppio = pluto * 2`*).*
  - *Statement if:* `if (exp) statement1 [else statement2]` *dove se* `exp` *è diverso da zero viene eseguito* `statement1`*, altrimenti* `statement2`*.*
  - *Ciclo while:* `while (exp) statement` *dove* `statemen` *viene eseguito finchè* `exp` *continua ad avere un valore diverso da zero.*

- *Una istruzione AWK appartiene ai seguenti tipi:*
  - *Ciclo for:* `for(exp1;exp2;exp3) statement` *dove* `exp1` *è eseguita al momento dell'inizializzazione del ciclo,* `exp3` *viene eseguita all'inizio di ogni ciclo e* `exp2` *fa si che si esca dal ciclo quando diventa falsa.*
  - *Ciclo for in:* `for(var in arrayname)statement` *simile al ciclo for della shell, fa si che alla variabile* `var` *vengano assegnati ad uno ad uno i valori contenuti nel vettore (unidimensionale)* `arrayname`*.*
  - *Stampa:* `print exp,[exp,...,exp]` *in cui ogni espressione* `exp` *viene calcolata e stampata nello standard output. I valori delle varie* `exp` *saranno distanziati dal carattere contenuto nella variabile* `OFS` *che di default è lo spazio. Se* `print` *viene usata senza* `exp` *viene eseguita la* `print $0`*.*

- *esistono poi le istruzioni:*
  - *break: esce dal ciclo* `while` *o* `for` *attivo.*
  - *continue: fa partire l'iterazione seguente del ciclo* `while` *o* `for` *ignorando le istruzioni rimanenti del ciclo*
  - *next: salta le istruzioni rimanenti del programma* AWK
  - *exit: fa terminare immediatamente* AWK

- `cat elenco.txt| awk '/Luca/ {print $3}'`
  - *stampa il terzo campo di tutte le righe del file* `elenco.txt` *che contengono la parola* Luca *(stampa una riga vuota se il terzo campo della riga è vuoto)*
- `cat elenco.txt| awk '/Luca/ {print}'`
  - *stampa tutte le righe del file elenco.txt che contengono la parola Luca (print equivale a* `print $0`*).*
- `awk 'BEGIN{FS=":"} ($2 == "OPE") {print $3,$1}' /etc/passw`
  - *stampa lo username e l'UID di tutti gli utenti del sistema che sono senza password.*
- `awk '/main()/{print FILENAME}' *.c`
  - *Stampa il nome di tutti i file con estensione* `.c` *che contengono la funzione* `main()`.

- *Oltre alle variabili FS e OFS, in* AWK *esistono altre variabili che vengono aggiornate automaticamente durante l'elaborazione del file in input:*
  - *NF: Numero dei campi della riga correntemente elaborata.*
  - *NR: Numero della riga correntemente elaborata*
  - *FILENAME: Nome del file correntemente elaborato. Questa variabile è indefinita all'interno del blocco BEGIN e contiene "-" se non sono specificati file nella linea di comando*
- `cat`
  - `awk '{print}'` *stampa l'intero file (si ricorda che* `print` *e* `print $0` *sono identiche)*
- `cat -n`
  - `awk '{print NR,$0}'` *stampa l'intero file includendo i numeri di riga*
- `wc -l`
  - `awk 'END {print NR}'` *stampa il numero di righe del file*

Lezione 9 dalla slides 27 sono presenti altri esercizi

COMANDI AWK IN SINTESI

Operatore	Descrizione
`(…)`	Raggruppamento
`$`	Riferimento a campi
`++ --`	Incremento e decremento, sia prefisso che postfisso
`^`	Elevamento a potenza
`+ - !`	Più e meno unari, e negazione logica
`* / %`	Moltiplicazione, divisione e resto
`spazio`	Concatenazione di stringhe.

funzione	Descrizione
`atan2(y, x)`	l'arcotangente di y/x in radianti
`sin(expr)` `cos(expr)`	seno e coseno di expr (si aspetta radianti).
`exp(expr)`	esponenziale
`int(expr)`	troncamento ad intero
`log(expr)`	logaritmo naturale
`rand()`	fornisce un numero casuale tra 0 ed 1.
`sqrt(expr)`	radice quadrata

Operatore	Descrizione
`< >` `<= >=` `!= ==`	I ben noti operatori di relazione
`~ !~`	Controllo di conformità ("match") tra regular expression, e controllo di non conformità
`in`	Controllo di appartenenza ad un vettore
`&& ||`	AND e OR logici
`+ - !`	Più e meno unari, e negazione logica
`+= -= *= /= %= ^=`	Assegnamento con operatore

funzione	Descrizione
`getline`	Setta `$0` leggendo la linea successiva; setta anche `NF, NR, FNR`
`getline <file`	Come sopra, ma legge da file
`getline var`	Setta `var` leggendo la linea successiva; setta `NR, FNR`
`getline var <file`	Come sopra, ma legge da file
`gensub(r, s, h [, t])`	cerca nella stringa obiettivo `t` corrispondenze con la regular expression `r`. Se `h` è una stringa che inizia con `g` o `G`, tutte le corrispondenze con `r` sono sostituite con `s`; altrimenti, `h` è un numero che indica la particolare corrispondenza con `r` che si vuole sostituire. Se `t` non è specificata, al suo posto è usato `$0`.

funzione	Descrizione
`gsub(r, s [, t])`	per ogni sottostringa conforme alla regular expression `r` nella stringa `t`, sostituisce la stringa `s`, e restituisce il numero di sostituzioni. Se `t` non è specificata, al suo posto è usato `$0`
`index(s, t)`	trova l'indice posizionale della stringa `t` nella stringa `s`, o restituisce `0` se `t` non è presente.
`length([s])`	la lunghezza della stringa `s`, oppure la lunghezza di `$0` se `s` non è specificata.
`match(s, r)`	trova la posizione in `s` del tratto che si conforma alla regular expression `r`, oppure `0` se non ci sono conformità.
`split(s, a [, r])`	spezza la stringa `s` nel vettore `a` utilizzando il metodo di separazione descritto dalla regular expression `r`, e restituisce il numero di campi. Se `r` è omessa, il separatore utilizzato è `FS`

funzione	Descrizione
`sprintf(fmt, expr-list)`	stampa fin nodo finisci expr-list secondo il formato fmt, e restituisce la stringa risultante
`sub(r, s [, t])`	come gsub(), ma è sostituita solo la prima sottostringa trovata.
`substr(s, i [, n])`	restituisce la sottostringa di `s` di `n` caratteri al più che inizia nella posizione `i`. Se `n` è omesso, è usato il resto di `s`.
`tolower(str)`	restituisce una copia della stringa str, con tutti i caratteri maiuscoli tradotti nei minuscoli corrispondenti. I caratteri non alfabetici restano invariati.
`toupper(str)`	restituisce una copia della stringa str, con tutti i caratteri minuscoli tradotti nei maiuscoli corrispondenti. I caratteri non alfabetici restano invariati.

funzione	Descrizione
`systime()`	restituisce la data e l'ora correnti, espresse come numero di secondi trascorsi da una certa data convenzionale (la mezzanotte del 1/1/1970 sui sistemi POSIX).
`strftime([format [, timestamp]])`	Applica il formato format a timestamp.

ARCHITETTURA DI UN SISTEMA OPERATIVO

La progettazione di un sistema operativo deve tener conto di dvierse caratteristiche:

- efficenza
- manutenibilità
- espansibilità
- **modularità** sistemi **molto efficenti** sono **poco moldulari**, sistemi **molto modulari** sono p**ochi efficenti,**

queste caratterisiche rappresentano un **trade-off**

E' possibile dividere i Sistemi Operativi in due grandi famiglie a seconda della loro struttura:

1. **Sistemi con struttura semplice :** la loro struttura non è progettata a priori e possono essere dscritti come una collezione di procedure, ognuna delle quali può richiamarne un'altra. Tipicamente sono sistemi operativi semplici e limitati he hanno subito un evoluzione al di là dello scopo originario.
   Osservazioni : le interfacce a livelli non sono ben separate e un programma sbagliato può mandare in crash l'intero sistema.

2. **Sistemi con struttura a strati :** presentano un insieme di strati, ogni strato si basa sullo strato precedente e offre servizi allo strato superiore. Il vantaggio principale è la modularità e con sistemi con struttura a strati vengono semplificate le fasi di *implementazione, debbuging, ristrutturazione del sistema.*
   Osservazioni (*problematiche*) : tendono ad essere meno efficenti (in quanto vanno in **overhead**) , occorre studiare attentamente i layer.
   Le seguenti problematiche hanno portato ad avere dei Sistemi con struttura a strati ma con meno strati.

Spesso queste caratteristiche rappresentano un ***trade-off.***

Kernel

Esistono quattro diversi tipo di Kernel :

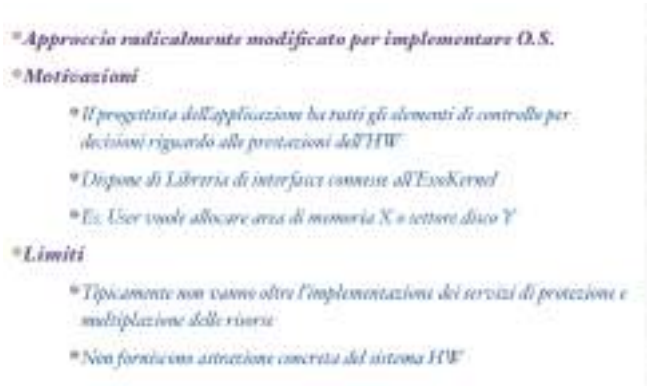
1. **Kernel monolitico** un aggregato unico di procedure

di gestione mutamente coordinate e astrazioni dell'hardware.

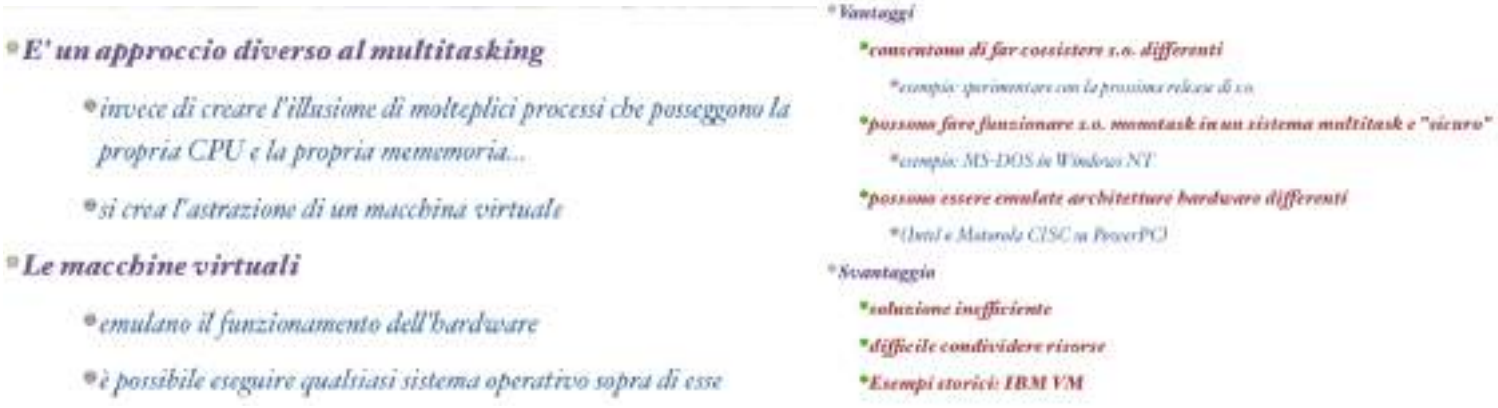
2. **Micro Kernel** semplice astrazioni dell'hardware gestite da un kernel minimale, basate su un paradigma client/server e primitive di message passing, cioè i messaggi tra i vari processi vengono smistati. Un microKernel deve fornire: le funzionalità minime di gestione dei processi e della memoria e meccanismi di comunicazione per permettere ai processi clienti di chiedere servizi ai processi serventi.

3. **Kernel ibridi** simili ai micro kernel ma con componenti eseguite in kernel space per questioni di maggiore efficenza

4. **Exo Kernel** non forniscono livelli di astrazioni dell'hardware ma forniscono librerie che mettono a contatto diretto le applicazioni con l'hardware.



Macchine virtuali



Progettazione di un sistema operativo :

1. **Definizione del problema :** definire gli obiettivi del sistema che si vogliono conseguire, definire i ***costraint*** entro cui si opera
2. **La progettazione sarà influenzata da :** al livello più basso dal sistema hardware con cui si va ad operare, al livello più alto dalle applicazioni che devono essere eseguite dal sistema operativo.

Uno stesso sistema operativo viene proposto per architetture hardware diverse, i tipici parametri usati sono:

1. Tipo di CPU usata
2. Quantità di memoria centrale
3. Periferiche usate
4. Parametri numerci di vario tipo

E si possono usare due diversi metodi : la rigenerazione del kernel con i nuovi parametri o prevedere la gestione di moduli aggiuntivi collegati durante il boot.

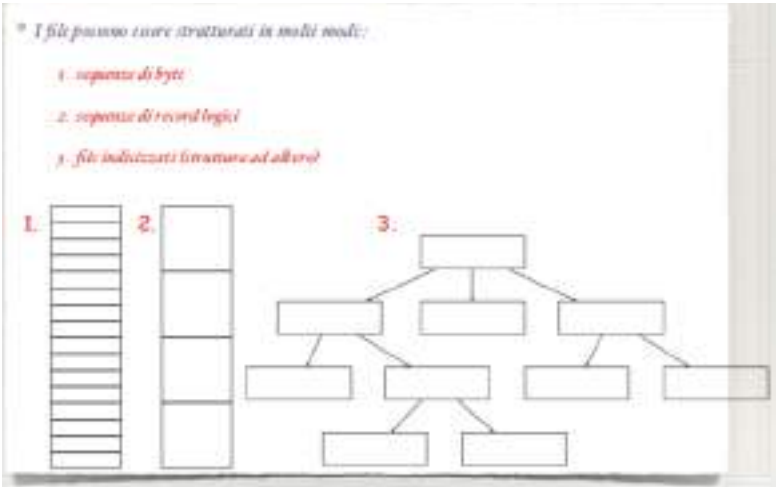
File System

Compito del file system è quello di astrarre la complessità di utilizzo dei diversi media, proponendo un'interfaccia per i sistemi di memorizzazione. Dal punto di vista dell'utente il file system è composto da due elementi:

1. **File :** unità logica di memorizzazione. Un file è l'entità atomica di assegnazione/gestione della memoria secondaria, è una collezione di informazioni correlate e fornisce una vista logica uniforme ad informazioni correlate. Ogni file ha:
   - **Nome :** una stringa di caratteri che permette agli utenti ed al sistema operativo di identificare un particolare file nel file system.
   - **Tipo :** necessario in alcuni sistemi per identificare il tipo di file.
   - **Locazione e dimensione :** informazioni sul posizionamento del file in memoria secondaria.
   - **Data & ora :** informazioni relative al tempo di creazione ed ultima modifica del file.
   - **Informazioni sulla proprietà :** utenti, gruppi etc. Utilizzato per *accounting* e *autorizzazione.*
   - **Attributi di protezione :** informazioni di acceso per verificare chi è autorizzato a eseguire operazioni sui file.
   - **Altri attributi :** flag, informazioni di locking, etc.

Esistono tre tecniche principali per identificare il tipo di un file:

- Meccanismo delle estensioni
- Utilizzo di un attributo *tipo* associato al file directory
- **Magic number**



I file in un File System si distinguono tra : *file regolari, directory (* file di sistema per mantenere la struttura del file system *) , file speciali a blocchi* (usati per modellare dispositivi I/O come i dischi) , *file speciali a caratteri (* usati per mdoellare device I/O seriali come terminali, stampanti e reti).

La struttura fisica è divisa in blocchi detti anche ***record fisici.*** Per una gestione efficente occorre risolvere il problema del packing dei record ligoci nei record fisici :

- **Record Fisico :** multiplo di blocco, unità di interscambio col livello di libreria
- **Recordo Logico :** unità di informazione vista dal livello applicativo

I metodi di accesso ai file sono : **sequenziale**, **accesso diretto, indicizzato (** si ha una tabella di corrispondenza chiave - posizione , il file viene memorizzato su disco con un metodo molto efficiente ma dispendioso).

2. **Directory :** un'insieme di informazione per orgsnizzare e fornire informazioni sui file che compongono un file system. La struttura di una directory:
   - **A livello singolo :** tutti i file sono elencati su un'unica lista lineare, ciascuno con il proprio nome (motivo per cui i nomi dei file devono essere unici) , Si comincia ad avere una gestione onerosa all'aumentare dei file.
   - **A due livelli :** una *Root Directory* contiene una *USer File Directory (UFD)* per ciascun utente di sistema. L'utente registrato può vdere solo la propria UFD (Le UFD di altri solo se esplicitamente autirizzato). I file sono locazzati tramite percordo (**path name**) , i programmi di sistema inoltre vengono copiati su tutte le UFD oppure posti in una directory di sistema condivisa e lì localizzati medianti cammini di ricerca predefiniti (**search path**). La struttua a due livelli ha un efficienza di ricerca e una libertà di denominazione, ma non si ha la libertà di raggruppamento.
   - **Ad albero :** ogni file è contenuto in una directory univoca.
   - **A grafo aciclico :** un file può essere contenuto in due o più directory ed esiste un'unica copia del file suddetto (ogni modifica al file è visibile in entrambe le directory). Abbiamo due implementazioni possibili :
     - **Link simbolici :** viene creato un tipo speciale di directory entry che contiene un riferimento al file in questione. Quando viene fatto un riferimento al file si cerca nella directory , si scopre che si tratta di un link, viene risolto il link.
     - **Hard link :** le informazioni relative al file vengono copiate en entrambe le directory. Non è necessario una doppia ricerca nel file system, ed è impossibile distinguere la copia dall'origine.

     Una struttura a grafo diretto aciclico è più flessibile di un albero, ma crea tutta una serie di problemi nuovi, non tutti i file system sono basati su DAG. Per l'implementazione è necessario usare la tecnica dei *i - node* che devono contenere un contatore di riferimento
   - **A grafo**

Un disco può essere diviso in una o più porzioni, porzioni indipendenti del disco che possono essere ospitare file system distinti. Il primo settore dei dischi è il cosidetto **master boot record (MBC)** ed è usato per fare il boot del sistema, per contenere la *partition table* e per contenere l'indicazione della partizione atttiva. Ogni partizione inizia con un boot block, il MBR carica il boot block della partizione attiva e lo esegue, il boot block carica il sistema operativo e lo esegue, l'organizzazione del resto della partizione dipende dal file system.

- **Problema dell'Allocazione :**

L'hardware e il driver del disco forniscono accesso al disco visto come un insieme di blocchi dati di dimensione fissa.

- **Allocazione contigua :**

I file sono memorizzati in sequenze contigue di blocchi di dischi.
Vantaggi : non è necessario utilizzare strutture dati per collegare i blocchi, l'accesso sequenziale efficente, l'accesso diretto è altrettanto efficente
Svantaggi : si ripropongono tutte le problematiche dell'allocazione contigua in memoria centrale; la frammentazione esterna e la politica di scelta dell'area di blocchi liberi da usare per allocare spazi per un file.

- **Allocazione concatenata :** ogni file è costituito da una lista concatenata di blocchi, ogni blocco contiene un puntatore al blocco successivo. Il descrittore del file contiene i puntatori al primo e all'ultimo elemento della lsita.

Vantaggi : risolve il problema della frammentazione esterna, l'accesso sequenziale è efficente.
Svantaggi : l'accesso diretto è inefficente, progressivamente l'efficenza globale del file system degrada (i blocch isono disseminati nel disco) , la dimensione utile di un blocco non è una potenza di due, se il blocco è piccolo l'overhead per i puntatori può essere rilevante.

- **Allocazione basata su FAT :** invece di usare parte del blocco per contenere il puntatore al blocco successivo si crea una tabella unica con un elemento per blocco. La FAT in sè mantiene la traccia delle aree del disco diponibili e di quelle già usate dai file e dalle directory.

Vantaggi : i blocchi dati sono interamente dedicati ai dati
Svantaggi : la scansione richeide anche la lettura della FAT, aumentando così il numero di accessi al disco.

- **Allocazione indicizzata :** l'elenco dei blocchi che compongono un file viene memorizzato in un blocco indice. Per accedere ad un file si carica in memoria la sua area indice e si usano i puntatori contenuti.

Vantaggi : risolve il problema della frammentazione esterna, è efficiente per l'accesso diretto, il blocco indice deve essere caricato in memoria solo quando il file è aperto.
Svantaggi : la dimensione del blocco indice determina l'ampiezzza massima del file, usano blocchi indici troppo grandi comportano un notevole spreco di spazio.

Il problema del trade - off viene risolto **concatenando i blocchi indici** : l'ultimo elemento del blocco indice non punta al blocco dati ma al blocco indice successivo e si ripropone il problema per l'accesso diretto a file di grandi dimensioni. Un'altra soluzione è **l'indice multilivello.**

Una directory è un file speciale contenente infomrazione su file contenuti nella directory.
Una directory è suddivisa in un certo numero di **directory entry.** Ogni directory entry deve permettere di accedere a tutte le informazioni necessarie per gestire il file : nome, attributi, informazioni di allocazione.

Risorse

Un sistema di elaborazione è composto da un insieme di risorse da assegnare ai processi presenti. I processi competono nell'accesso alla risorse che possono essere suddivise in classi, le risorse ppartenenti alla stessa classe sono *equivalenti.* Le risorse di una classe vengono dette **istanze** della classe. Il numero di risorse in una classe viene detto **moleplicità** del tipo di risorsa.
Un processo non può richidere una specifica risorsa ma solo una risorsa di una specifica classe.
Assegnazione delle risorse :

- **Risorse ad assegnazione statica :** avviene al momento dellac reazione del processo e rimane valida fino alla terminazione.
- **Risorse ad assegnazione dinamica :** i processi richiedono le risorse durante la loro esistenza, le utilizzano una volta ottenute e le rilasciano quando non più necessarie.

Tipi di richieste :

- **Richiesta singola :** si riferisce ad una singola risorsa di una classe definita
- **Richiesta multipla :** si riferisce ad una o più classi, e per ogni classe, ad una o più risorse deve essere soddisfatta integralmente.

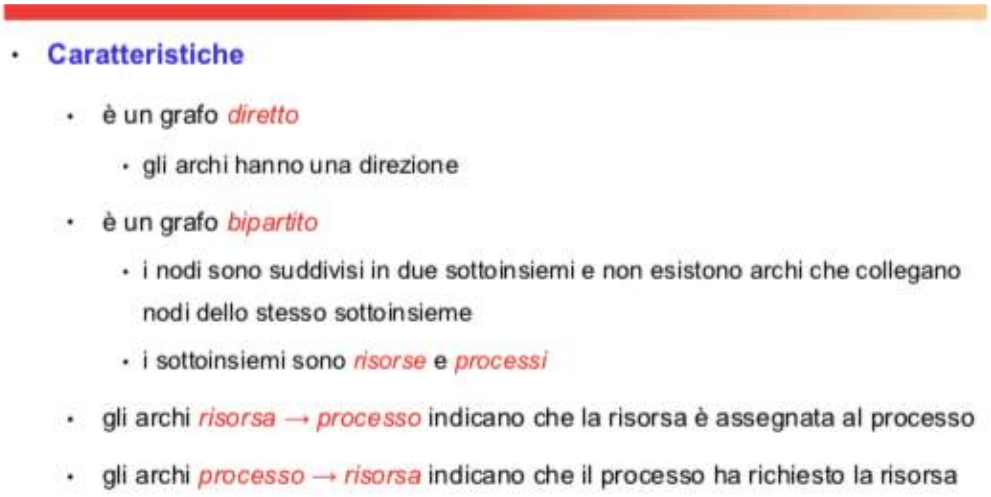
Le risorse si possono distinguere in :

- **Risorse seriali :** una singola risorsa non può essere assegnata a più processi contemporaneamente. Ne sono un esempio i processi.
- **Risorse non seriali :** ne sono un esempio i file di solo lettura.

- **Risorse prerilascibile :** se la funzione di gestione può sottrarla ad un processo prima che questo l'abbia effettivamente rilasciata, in altre parole una risorsa è prerilascibile se il suo stato non si modifica durante l'utilizzo oppure se il suo stato può essere facilmente salvato e ripristinato. Il processo che subisce il **prerilascio** deve sospendersi. La risorsa prerilasciata sarà successivamente restituita al processo.
- **Risorse NON prerilascibile:** la funzione di gestione non può sottrarle al processo al quale sono assegnate, sono non prerilascibili le risorse il cui stato non può essere salvato e ripristinato.

I **deadlock** impediscono ai processi di terminare correttamente, le risorse bloccate in deadlock non possono essere usate da altri processi. Le condizioni per avere un deadlock:
- **Mutua esclusione :** le risorse coinvolte devono essere seriali
- **Assenza di prerilascio :** le risorse coinvolte non possono essere prerilasciate, ovvero devono essere rilasciate volontariamente dai processi che le controllano.
- **Richiesta bloccanti :** le richieste devono essere bloccanti, e un processo che ha già ottenuto risorse può chiederne ancora.
- **Attesa circolare :** esiste una sequenza di processi $P_0, P_1, ..., P_n$, tali per cui P_0 attende una risorsa controllata da P_1, P_1 attende una risorsa controllata da P_2, ..., e P_n attende una risorsa controllata da P_0

Grafo di Holt



L'insieme delle risorse è **partizionato** in classi e gli archi di richiesta sono diretti alla classe e non alla singola risorsa. I processi sono rappresentati da cerchi, le classi vengono rappresentate da contenitori rettangolari e le risorse sono punti all'interno delle classi.

Metodi di gestione dei deadlock :
- **Deadlock detection and recovery:** permette al sistema di entrare in stati deadlock, usando un algoritmo per rilevare questo stato ed eventualmente eseguire un'azione di recovery. Mantiene aggiornato il grado di Holt, registrando su di esso tutte le assegnazioni e le richieste di risorse. Nel deadlock recovery tutti i processi coinvolti vengono temrinati, viene elimanto un processo alla volta, fino a quando il deadlock non scompare. Una risorsa viene sottratta ad uno dei processi coinvolti nel deadlock.
- **Deadlock prevetion / avoidance :** impedire al sistema di entrare in uno stato di deadlock
- **Ostrich algoritm** ignorare il problema del tutto

Teorema :
Se le risorse sono ad accesso mutualmente esclusivo, seriali e non prerilasciabili. Lo stato è di deadlock se e solo se il grafo di Holt contiene un ciclo. Lo stato non è di deadlock se e solo se il grafo di Holt è completamente riducibile. Esiste una sequenza di passi di riduzione che elimina tutti gli archi del grafo.
Allora il grafo rappresenta uno stato di deadlock se e solo se esiste un knot.

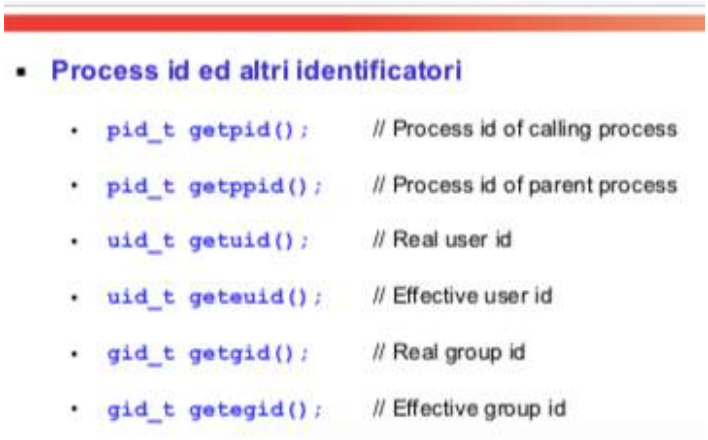
Un grafo di Holt si dice riducibile se esiste almeno un nodo processo con solo archi entranti.
Lo stato dei processi viene perodicamente salvato su disco (*checkpoint)* , in caso di deadlock, si ripristina *(rollback)* uno o più processiad uno stato precedente, fino a quando il deadlock non scompare. Per evitare il deadlock si elimina una delle quattro condizioni del deadlock, nel seguente modo il deadlock viene eliminato strutturalmente.
Prima di assegnare una risorsa ad un processo, si controlla se l'operazione può portare al pericolo di deadlock, in quest'ultimo caso l'operazione viene ritardata.

Algoritmo del banchiere :
Un banchiere desidera *condividre un capitale* con un numero prefissato di clienti. Ogni cliente specifica in anticipo la sua necessità massima di dinaro (che ovviamente non deve superare il capitale del banchiere). I clienti fanno due tipi di transazioni : **richieste di prestito & restituzioni.**
Il denaro prestato ad ogni cliente non può mai eccedere la necessità massima specificata a priori.
Ogni cliente può fare richieste multiple, al massimo si può avere un importo specificato. Una volta che le richieste sono state accolte e il denaro è stato ottenuto deve garantire la restituzione in un tempo finito.
Il banchiere deve essere in ogni istante in grado di soddisfare tutte le richieste dei clienti o concedendo immediatamente il prestito oppure facendo loro aspettare la disponibilità del denaro in un tempo finito.

Le similutidini fra banchieri e Sistemi Operativi è la seguente :
- Il denaro sono le risorse.
- Il sistema le deve allocare ai processi senza che si possa verificare deadlock.
- Le definizioni viste fino a questo punto riguardano il caso teorico elementare di un sistema avente **un'unica classe di risorse.**

Controllo di processo

Ogni processo ha un identificatore univoco, usato spesso per creare altri identificatori e grantire unicità.



La **bufferizzazione** è una delle caratteristiche principlai dell'interfaccia degli stream; lo scopo è quello di ridurre al minimo il numero di system call eseguite nelle operazioni di input/output. I caratteri che vengono scritti su di uno stream normalmente vengono accumulati in un buffer e poi trasmessi in blocco tutte le volte che il buffer viene riempito, in maniera asincrona rispetto alla scrittura. Lo standard definisce tre distinte modalità in cui può essere eseguita la bufferizzazione.
- **Unbuffered :** in questo caso non c'è bufferizzazione ed i caratteri vengono trasmessi direttamente al file non appena possibile.
- **Line buffered :** in questo caso i caratteri vengono normalmente trasmessi ai file in blocchiogni volta che viene incontrato un carattere di newline.
- **Fully buffered :** in questo caso i caratteri vengono trasmessi da e verso il file in blocchi di dimensione opportuna.,

Segnali

I segnali sono ***interrumpt software*** a livello di proesso. Permettono la gestione di eventi asincroni che interrompono il normale funzionamento di un processo. Ogni segnale ha un identificatore. Identificatori di segnali sono ***nomi simbolici*** che iniziano con i tre caratteri **SIG.** I nomi simbolici corrispondono ad un intero positivo . La gestione avviene tramite ***signal handler.***
I segnali possono essere legati a diverse cause:

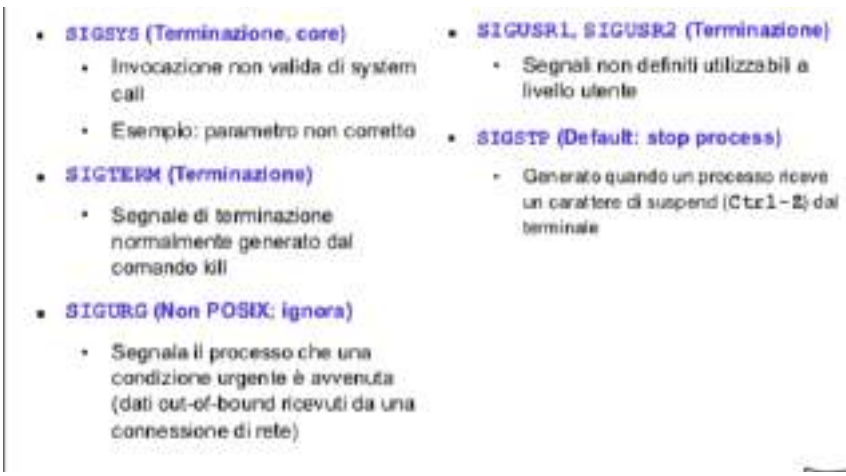
- **Kernel** : verifica la presenza di segnali pendenti per un processo quando questo passa da kernel a user mode, o quando abbandona lo stato sleep o vi entra.
- **Altri processi**
- **Interrupt espliciti dell'utente**

Un segnale può essere gestito con un certo ritardo, e non si tiene conto del numero di segnali dello stesso tipo pendenti.
Quando un processo riceve un segnale può reagire in uno dei seguenti modi:

- **Ignore :** Il segnale viene ignorato e non ha alcun effetto (*SIGKILL* e *SIGSTOP* non possono essere ignorati)
- **Catch :** esegue un **signal handler,** una funzione che gestisce l'evento associato al segnale :
  - Sospende il flusso di esecuzione corrente.
  - Esegue il signal handler.
  - Riprende il flusso di controllo originario quando il signal handler termina.

La reazione ad un segnale può essere definita dal programmatore o può essere quella di default stabilita dal kernel. Ogni segnale ha un'azione di default che può essere:

- **Segnale scartato**
- **Terminazione processo** (generando un file core - ***dump*** - o senza generare un file core - ***quit*** -)
- **Sospensione processo** (suspend) .
- **Riprende l'esecuzione del processo** (resume) .



E' possibile inviare un segnale ad un processo in **foreground** premendo **<Ctrl-c>** o **<Ctrl-z>** dalla tastiera. Qundo il driver riconosce che è stato premuto uno dei due comandi viene inviato un segnale **SIGINT (SIGSTP)** a tutti i processi nel gruppo del processo in foreground.

- **SIGCHLD :** inviato al padre da un figlio che termina.
- **SIGSTOP :** sospensione da dentro un programma.
- **SIGCONT :** riprende l'esecuzione di un programma dopo una sospensione.

unsigned int alarm (unsigned int count)

Istruisce il Kernel a spedire il sengale **SIGALARM** al processo invocante dopo **count** secondi :

- Un eventuale **alarm()** già schedulato viene schedulato.
- Se **count è 0** non schedula nessun nuovo **allarm()** , e cancella eventualmente quello già schedulato.
- Restituisce il numero di secondi rimanenti dell'invio dell'arme, oppure **0** se non è schedulato nessun **allarm().**

L'allarme è inviato dopo almeno **count** secondi, ma il meccanismo di scheduling può ritardare ulteriormente la ricezione del segnale.

L'azione di default per **SIGALRM** è di terminare il processo, ma normalmente viene definito un signal handler per il segnale.

System call : int pause();
Questa funzione sospende il processo fin oa quando un sgnale non viene catturato. Ritorna -1 e setta **errno** a **EINTR.**

void (* (signal (int signum, void (*handler) (int))) (int)

Installa un nuovo signal handler, per il segnale con numero **signum.** Restituisce il precendente signal handler associato a signum, se ha successo; altrimento **-1.**

Handler può essere :

- **Indirizzo** di una funzione handler definita dall'utente. La funzione handler ha un argomento intero che rappresenta il numero del segnale. Questo permette di usare lo stesso handler per segnali differenti. Uno dei seguenti valori :
    - **SIGN_IGN :** indica che il segnale dev'essere ignorato
    - **SIGN_DFL :** indica che deve essere usato l'handler di default fornito dal nucleo.

Il nome di una funzione è un valore puntatore a funzione. Una funzione in una dichiarazione di parametro viene interpretata dal compilatore come puntatore. Quindi nel prototipo di **signal()** si può togliere **" * "** …

I segnali **SIGKILL** e **SIGSTP** non sono riprogrammabili.

- Con la creazione di nuovi processi :
  Dopo una **fork(),** il processo figlio eredita le politiche di gestione dei segnali del padre. Se il figlio esegue una **exec()** i segnali precedentemente ignorati continuano ad essere ignorati, oppure i segnali i cui handler erano stati ridefiniti sono ora gestiti dagli handler di default.

- Ad eccezione di **SIGCHLD** i sengali non sono accodati. Quando più segnali dello stesso tipo arrivano *contemporaneamente* solo uno viene tratato.

System call : int kill (pid_t pid , int signo) ;
La funzione **kill** spedisce un segnale ad un processo oppure ad un gruppo di processi.
Argomento pid :

- **pid > 0** spedito al processo identificato da pid.

- **pid == 0** spedito a tutti i processi appartenenti allo stesso gruppo del processo che invoca kill.
- **pid < -1** spedito al gruppo di processi identificati da **| pid |**
- **pid == -1** se il mittente ha per propietario il superuser, invia il segnale a tutti i processi, mittente incluso. Se il mittente non ha per propietario un superuser, invia il segnale a tutti i processi nello stesso gruppo del mittente, con esclusione del mittente.

Argomento **signo** numero di segnale spedito. La funzione **kill** restituisce valore **0** , se invia con successo almeno un segnale altrimenti, restituisce **-1.**

int kill (pid_t pid, int signo) ;

- Il superuser può spedire segnali a chiunque
- I processi mittente e destinatario hanno lo stesso propietario; più precisamente **real** o **effective** del mittente coincide con real o effective uid del destinatario.
- Il processo mittente ha come propietario il **superuser.**
- Se il segnale spedito è null, **kill** esegue i normali meccanismi di controllo errore senza spedire segnali.

System call: int raise (int signo) ;
Spedisce il segnale al processo chiamante.

Il programma **limit.c** permette all'utente di limitare il tempo impiegato da un comando per l'esecuzione. Ha la seguente interfaccia :

limit nsec cmd args

Esegue il comando **cmd** con gli argomenti **args** indicati, dedicandovi al massimo **nsec** secondi. Il programma definisce un handler per il segnale **SIGCHLD.**

Il programma **pulse.c** crea due figli che entrano in un ciclo infinito e mostrano un messaggio ogni secondo.

System call: void abort();
Questa System call spedisce il segnale **SIGABRT** al processo comportamento in caso di:
- **SIG_DFL :** terminazione del processo.
- **SIG_IGN :** NON MMESSO
- **signal handler :** il segnale viene catturato, e il signal handler può eseguire **return** o può invocare **exit** o **_exit.** In entrambi i casi, il processo viene terminato. La normale sequenza di istruzione viene interrotta, vengono eseguite le istruzioni dei signal handler, e quando signal handler ritorna la normale sequenza di istruzioni viene ripresa. Se il segnale viene catturato durante l'esecuzione di una malloc ciò che accede è un **segmentation fault.**

Un segnale è generato per un processo quando accade l'evento associato al segnale. Quando il segnale viene generato, viene settato un flag nel process contrll block del processo. Diciamo che un segnale è *consegnato* ad un processo quando l'azione associata al segnale viene intrapresa. Diciamo che un segnale è *pendente* nell'intervallo di tempo che interccorre tra la generazione del segnale e la consegna. Un processo ha l'opzione di bloccare la consegna di un segnale per cui l'azione di default non è ignore. Se un segnale viene generato per un processo, il segnale rimane *pending* fino a quando
- Il processo sblocca il segnale.
- Il processo cambia l'azione associata al senale ad ignore.

E' possibile ottenere la lista dei segnali pending tramite la funzione *sigpending*

int sigprocmask (int how, sigset_t *set, sigset_t *oset) ;
- **Argomento *oset :*** se è diverso da NULL, al termine della chiamata questa struttura dati conterrà la maschera precedente.

- **Argomento *set :*** se è diverso da NULL, allora l'argomento **how** descrive come la maschera viene modificata.
- **Argomento *how :***
    - **SIG_BLOCK** : blocca i segnali compresi in set.
    - **SIG_UNBLOCL:** sblocca i segnalicompresi in set.
    - **SIG_SETMASK :** set è la nuova maschera.

Pipe

E' un canale di comunicazione che unisce due processi. La più vecchia è la più usata forma di interprocess communication in Unix. Ha delle limitazioni tra cui **half-duplex** (comunicazione in un solo sensso) , è utilizzabile solo tra processi con un **antenato** in comune. E' possibile superare questi limiti con gli **Stream pipe** che sono *full - duplex ,* facendo uso della disciplina **FIFO** che permette di usare più processi : named stream pipe = stream pipe + FIFO .

I pipe in un singolo processo sono completamente inutili, normalmente il processo che chiama pipe chiama **fork**. I descrittori vengono duplicati e creano un canale di comunicazione tra parent e child o viceversa. I canali non usati vengono chiusi