ROMA TRE UNIVERSITÀ DEGLI STUDI
Corso di Laurea in Matematica
Dipartimento di Matematica e Fisica

Sistemi per l'elaborazione delle informazioni

3. Sistemi operativi

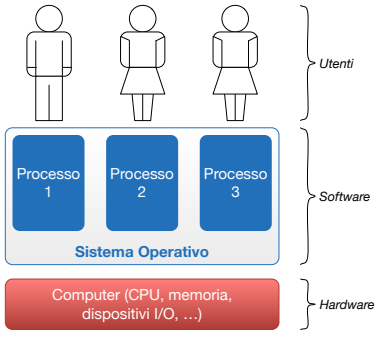
Dispense del corso IN530 a.a. 2019/2020

prof. Marco Liverani

Premessa

- Il **sistema operativo** è il **software di base** che sovrintende al funzionamento dell'intero computer, all'esecuzione dei programmi, all'interazione con gli utenti
- È lo strato software che **si colloca tra l'hardware della macchina e il software applicativo** utilizzato dall'utente e **fornisce un'astrazione dell'hardware ai programmi software**

- Il sistema operativo si occupa della
  - gestione delle componenti hardware
  - esecuzione dei programmi
  - interazione con le periferiche
  - interazione con gli utenti
- È il sistema operativo a definire la **modalità operativa** del computer
- È il sistema operativo a definire la **modalità di utilizzo** del computer da parte degli utenti



- Alcuni esempi: Microsoft Windows, GNU Linux, Apple OS X, Sun Solaris, HP-UX, IBM AIX, IBM Z-OS, Google Android, Apple iOS, Chrome OS, FreeBSD, NetBSD, Plan9, GNU Hurd, ...

Caratteristiche di un sistema operativo

Il sistema operativo definisce la modalità operativa della macchina:

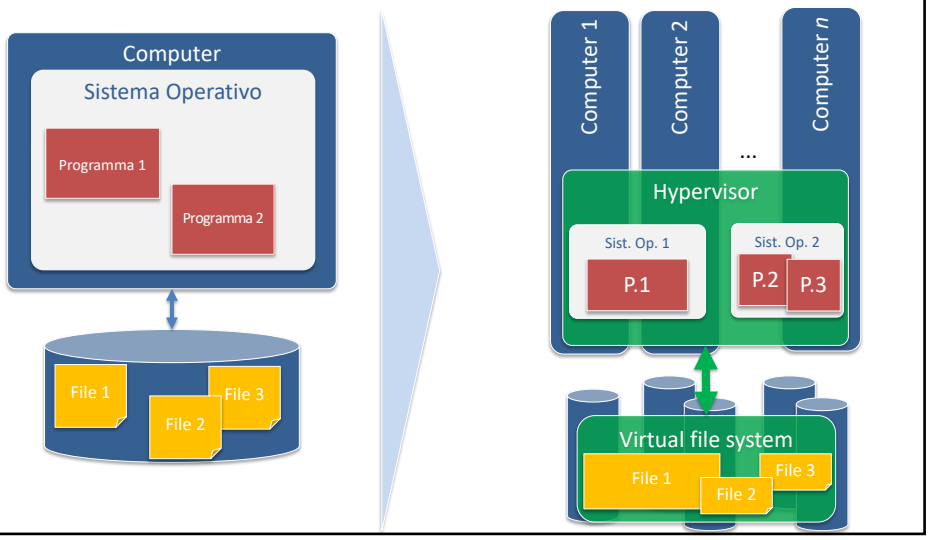
- **sistemi operativi *batch***: il computer esegue programmi in modalità non interattiva, l'utente carica dati e programma e solo al termine dell'elaborazione ottiene un output
- **sistemi operativi *time sharing* interattivi**: la macchina esegue più programmi dedicando a turno a ciascun programma una parte del tempo CPU; l'utente interagisce con il programma durante la sua esecuzione, fornendo input e ottenendo output anche durante l'esecuzione
- **sistemi operativi *real time***: la macchina è dedicata a manovrare degli apparati (es.: un braccio robotizzato, un tornio a controllo numerico, ecc.) e il sistema operativo garantisce l'esecuzione del programma in un tempo predefinito, senza latenze o possibilità di deterioramento delle prestazioni
- **sistemi operativi *embedded***: il computer ed il sistema operativo sono integrati in un apparato hardware (es.: un'automobile) e ne controllano il funzionamento di alcune componenti
- **sistemi operativi *hypervisor***: il sistema operativo consente la ripartizione delle risorse hardware in più macchine virtuali di cui presenta un'astrazione per poter eseguire, come programmi, altri sistemi operativi ospiti

A volte sistemi operativi *time sharing* contengono componenti di virtualizzazione (*hypervisor*) e consentono un partizionamento del computer in più macchine virtuali (es.: partizioni IBM Z/OS, zone Sun Solaris, ecc.)

A proposito dei sistemi Hypervisor

Sistemi tradizionali: 1 computer, 1 sistema operativo, più programmi, capacità di archiviazione file limitata alla dimensione dello storage

Sistemi virtualizzati: *n* computer controllati da un unico hypervisor che alloca le risorse comuni (virtual CPU, virtual memory, virtual network, virtual file system) alle macchine virtuali. La capacità delle macchine può superare quella di un singolo computer fisico



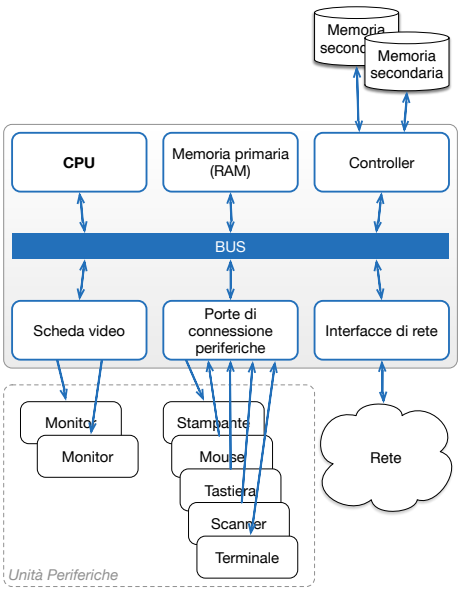
Caratteristiche di un sistema operativo

I sistemi operativi possono essere

- **mono-task** (mono-programmazione): il sistema operativo consente l'esecuzione di un solo processo per volta; un secondo processo può essere eseguito solo quando il precedente ha terminato il proprio lavoro (es.: Microsoft MS-DOS, Apple System 1-5)
- **multi-task** (multi-programmazione): il sistema operativo consente l'esecuzione di più processi contemporaneamente, dedicando a ciascuno a turno un po' del tempo di elaborazione (es.: Microsoft Windows, Apple System 6-9, Apple OS X, GNU Linux, Sun Solaris, ...)
  - ***multi-tasking cooperativo***: è una versione poco efficiente di *multi-tasking* in cui è il programma (e non il sistema operativo) a stabilire quando rilasciare la CPU per l'esecuzione di istruzioni di altri programmi
- **multi-threading**: un processo viene suddiviso in più *thread* distinti, eseguiti contemporaneamente sulla stessa CPU, in modo da migliorare le performance del programma utilizzando più a fondo la CPU; il multithreading deve essere supportato da un hardware (CPU) che lo renda possibile (es.: Microsoft Windows NT/2000/..., GNU Linux, Sun Solaris, ...)
- **mono-utente**: il sistema operativo non possiede il concetto di utente e non distingue l'utente che utilizza il computer (es.: Microsoft MS-DOS, Microsoft Windows 95/98, Apple System 1-9)
- **multi-utente**: il sistema operativo possiede il concetto di utente ed assegna a ciascun utente file sulla memoria secondaria e processi in esecuzione (es.: Microsoft Windows NT/2000/..., GNU Linux, Sun Solaris, ...)

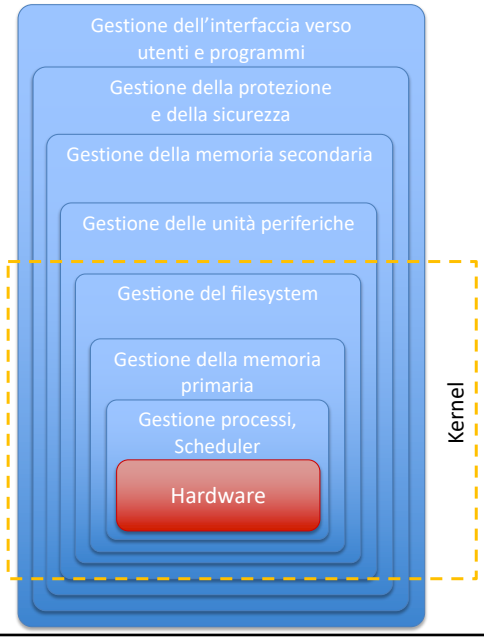
Componenti di un computer

- Un computer è un sistema formato da diverse componenti: una unità centrale connessa con un insieme di unità periferiche
- Il sistema operativo garantisce il funzionamento di tutte queste componenti elettroniche, elettromagnetiche e meccaniche, coordinando la comunicazione tra le diverse unità
- Tra le interfacce di rete:
  - Ethernet, wi-fi, GSM, ecc.
- Tra le porte di connessione:
  - parallela, seriale, USB, Bluetooth, Thunderbolt, ecc.



Struttura di un sistema operativo

1. Gestione dei **processi**, *scheduler*
2. Gestione della **memoria primaria**
3. Gestione del **filesystem**
4. Gestione delle **periferiche** e dell'input/output
5. Gestione della **memoria secondaria**
6. Gestione della **protezione** e della **sicurezza**
7. Gestione delle **interfacce verso l'utente** e i **programmi applicativi**



Struttura di un sistema operativo

- Il **Kernel** costituisce il nucleo fondamentale del sistema operativo: sul kernel si appoggiano le altre funzioni del sistema operativo stesso
- In base alla struttura interna del sistema operativo e del suo kernel si distinguono tre macro-tipologie differenti:
  - **Sistemi operativi a kernel monolitico**: implementa una completa astrazione del computer su cui gira il sistema operativo stesso (es.: GNU Linux, FreeBSD, ...)
  - **Sistemi operativi a micro-kernel**: implementano nel kernel solo le funzioni essenziali, demandando ad altri programmi di sistema (*device driver*, *server*) l'implementazione delle altre funzioni (Minix, IBM AIX, ...)
  - **Sistemi operativi ibridi**: implementano nel kernel diverse funzioni del sistema operativo, ma al momento dell'avvio possono caricare dei moduli aggiuntivi che arricchiscono e integrano le funzioni offerte dal kernel

Gestione dei processi / Scheduler

- Un **processo** è un'istanza in esecuzione di un **programma**
- In un computer con sistema operativo *multi-task* possono essere eseguiti più processi contemporaneamente (di programmi diversi o di uno stesso programma)
- La gestione dei processi da parte del sistema operativo consiste nelle seguenti attività:
  - creazione e terminazione di un processo
  - sospensione e ripristino dell'esecuzione di un processo
  - sincronizzazione e comunicazione tra processi in esecuzione (IPC: *inter-process communication*)
  - gestione del blocco di un processo (*dead lock*)
- I processi possono eseguire delle **chiamate a funzioni di sistema** per ottenere dei servizi dal sistema operativo relative alla gestione dei processi stessi:
  - esecuzione di altri processi (*exec*)
  - replica del processo in esecuzione (*fork*)
  - invio di segnali da un processo ad un altro (*wait, signal*)
  - terminazione di un processo (*kill/terminate*)

Gestione dei processi / Scheduler

Esempio di chiamate di sistema per replicazione del processo in esecuzione (**fork**):

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(void) {
  int pid=0, i;
  pid = fork();
  if (pid == 0) {
    for(i=0; i < 10; i++) {
      printf("Figlio: %d\n", i);
      sleep(1);
    }
    _exit(0);
  } else if (pid > 0) {
    for(i=0; i < 10; i++) {
      printf("Padre: %d\n", i);
      sleep(1);
    }
  } else {
    fprintf(stderr, "Errore nel fork");
    exit(1);
  }
  return 0;
}
```

Esegue una copia del processo attuale: alla variabile `pid` viene assegnato il *process-id* del processo figlio creato da `fork()`

Se **`pid=0`** allora il processo è il "**figlio**" e viene eseguita questa parte di programma

Se **`pid>0`** allora il processo è il "**padre**" e viene eseguita quest'altra parte di programma: `pid` è il *process-id* del processo figlio creato da `fork()`

Gestione dei processi / Scheduler

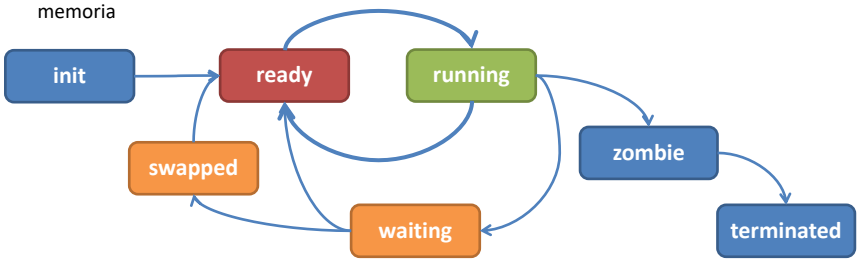
- Mediante la *system call* "**fork**" il sistema operativo, a partire da un processo iniziale di avvio del sistema stesso, è in grado di eseguire, in base alla sua configurazione, altri processi in cascata
- Si viene quindi a creare una **gerarchia ad albero** "padre/figlio" tra i processi eseguiti sul computer: la radice è il processo di inizializzazione del sistema (su UNIX: "`init`")
- Ogni processo è identificato da un **PID** (process ID), da un **PPID** (parent process ID) e da uno **USERID** (codice identificativo dell'utente che sta eseguendo il processo)

```
marco@home ~$ pstree
init─┬─auditd───{auditd}
     ├─crond
     ├─httpd───3*[httpd]
     ├─2*[mingetty]
     ├─rsyslogd───3*[{rsyslogd}]
     ├─2*[sendmail]
     ├─sshd───sshd───sshd───bash───pstree
     ├─udevd───2*[udevd]
     └─vsftpd
```

```
marco@home ~$ ps -ef
UID        PID  PPID  C STIME TTY          TIME     CMD
root         1     0  0  2014 ?        00:00:23 /sbin/init
root       375     1  0  2014 ?        00:00:00 /sbin/udevd
root      1209     1  0  2014 ?        00:03:28 auditd
root      1229     1  0  2014 ?        00:02:27 /sbin/rsyslogd
root      1291     1  0  2014 ?        00:02:48 /usr/sbin/sshd
root      1303     1  0  2014 ?        00:00:02 /usr/sbin/vsftpd
root      1339     1  0  2014 ?        00:03:32 sendmail: accept
smmsp     1350     1  0  2014 ?        00:00:01 sendmail: Queue
root      1371     1  0  2014 ?        00:00:26 crond
root      1412     1  0  2014 tty1     00:00:00 /sbin/mingetty
root      1414     1  0  2014 tty2     00:00:00 /sbin/mingetty
root      1416   375  0  2014 ?        00:00:00 /sbin/udevd -d
root      2229     1  0  2014 ?        00:06:08 /usr/sbin/httpd
apache    6128  2229  0 Feb22 ?        00:00:03 /usr/sbin/httpd
apache    6129  2229  0 Feb22 ?        00:00:00 /usr/sbin/httpd
root     16640  1291  0 09:55 ?        00:00:00 sshd: marco [pri
marco    16642 16640  0 09:56 ?        00:00:00 sshd: marco@pts/
```

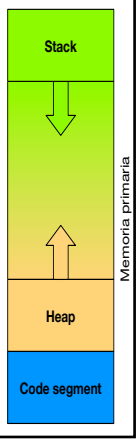
Gestione dei processi / Scheduler

- Ciascun processo durante il suo "ciclo di vita" può trovarsi in uno dei seguenti **stati**:
  - **init**: stato iniziale di caricamento del processo in memoria: viene lanciato in esecuzione il programma e viene creato il processo e allocata la sua memoria
  - **ready**: il processo è pronto per essere eseguito dalla CPU
  - **running**: il processo è in esecuzione da parte della CPU
  - **waiting**: il processo è sospeso in attesa di un evento (es.: la risposta da un *device*)
  - **swapped**: il processo, in attesa di eventi, è stato portato nella memoria virtuale in attesa di essere recuperato nella memoria primaria per essere eseguito
  - **zombie**: il processo ha concluso la sua esecuzione, ma è ancora presente nella memoria (possiede un PID) in attesa che il processo padre lo liberi definitivamente
  - **terminated**: il processo è in corso di terminazione, il sistema operativo lo sta de-allocando dalla memoria



Gestione della memoria primaria

- La **memoria primaria** è costituita dai **registri** della CPU, dalla **memoria cache** della CPU e dalla **memoria RAM** (*random access memory*)
- I compiti del sistema operativo nella gestione della memoria sono:
  - **allocazione e deallocazione della memoria** richiesta dai processi in esecuzione
  - **mantenere separate le porzioni di memoria** destinate a processi diversi su un sistema multi-tasking, evitando conflitti nell'uso della memoria
  - gestire il collegamento tra gli **indirizzi di memoria logici** utilizzati dai processi e la **memoria fisica** della macchina
  - gestire la **paginazione** e la **memoria virtuale**, spostando su memoria secondaria pagine (porzioni) di memoria primaria e caricando dalla memoria secondaria pagine di informazioni da riallocare nella memoria primaria (*swap*)
- Le chiamate di sistema che i processi possono invocare per ottenere servizi di gestione della memoria dal sistema operativo sono:
  - `malloc`, `calloc`, `realloc`, per l'allocazione dinamica di blocchi di memoria;
  - `free` per la disallocazione di blocchi di memoria precedentemente allocati
- La memoria primaria della macchina è una pila costituita da
  - **stack**: cresce dall'alto verso il basso, per allocare variabili richieste dalla chiamata di funzioni
  - **heap**: cresce dal basso verso l'alto, per allocare dinamicamente porzioni di memoria
  - **code segment**: sotto allo heap, contiene le istruzioni in linguaggio macchina del processo (come previsto dal modello di Von Neumann, il programma è nella memoria della macchina



Gestione del filesystem

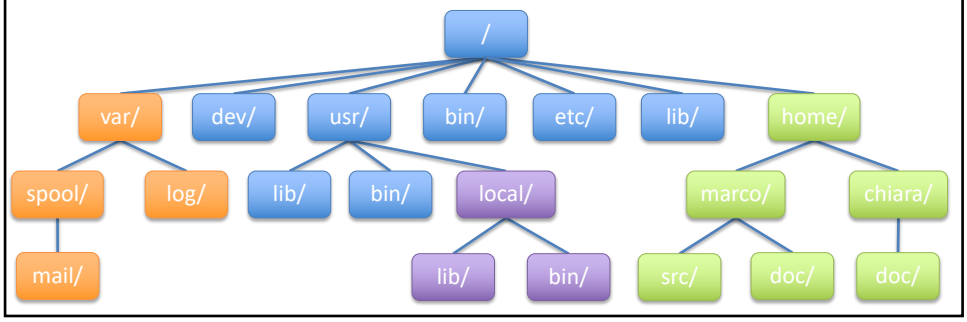
- Il **filesystem** è **un'astrazione del modello** con cui il sistema operativo gestisce i dati sulla memoria secondaria; il modello è indipendente dal tipo e dal numero di dispositivi di memoria secondaria
- L'elemento di base è il **file**, una sequenza di byte memorizzata sulla memoria secondaria, che termina con il simbolo EOF (*end of file*)
- Il filesystem fornisce anche un modello astratto con cui i file sono organizzati sulla memoria secondaria (tipicamente una struttura ad albero di *directory* e *sotto-directory*)
- Il concetto di **directory** ("cartella") è anch'esso un'astrazione: anche una directory è un file; l'inclusione di un file in una directory è realizzato mediante puntatori "padre-figlio"
- La **denominazione dei file**, il **set di caratteri** con cui possono essere denominati, i **metacaratteri** con cui si indica la collocazione di un file nel modello del filesystem, sono alcuni degli aspetti definiti per ogni specifico modello di filesystem
  - **"`C:\DOCUMENTI\TESI.TEX`"** è un tipico *path* assoluto che identifica univocamente un file dislocato sul disco "C" sul sistema operativo MS-DOS o Microsoft Windows

    NOTA: *non è case sensitive* e viene specificato l'identificativo dell'unità fisica su cui è presente il file
  - **"`~liverani/src/minimumSpanningTree.c`"** è il *path* di un file su un filesystem di un sistema operativo di tipo UNIX

    NOTA: *è case sensitive*, si utilizza la convenzione "`~username`" per identificare la home directory di un utente, il path è indipendente dalla collocazione fisica del file su uno specifico *device*

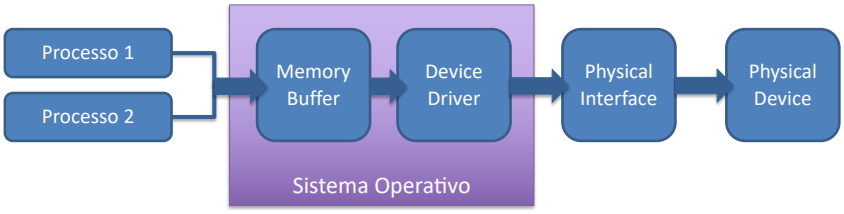
Gestione del filesystem

- I processi possono invocare funzioni di sistema per operare sul filesystem:
  - creazione e cancellazione di file
  - apertura e chiusura di file (`fopen`/`fclose`)
  - lettura e scrittura di file (`fget`/`fread`/`fwrite`/...)
  - impostazione di attributi del file (read only, writable, executable, ...)
- Il filesystem implementa meccanismi di **protezione dei file**, per restringere l'accesso ai soli utenti autorizzati e gestisce una **coda delle richieste di accesso** al file da parte dei processi
- Su un sistema operativo multiutente il filesystem tiene traccia dell'identificativo dell'utente **proprietario del file** e delle regole di accesso al file per gli utenti del sistema



Gestione delle periferiche e dell'input/output

- Il sistema operativo gestisce la comunicazione (in ingresso e in uscita) verso le **unità periferiche**, fornendo un'astrazione e delle funzioni per l'utilizzo di tale canale di comunicazione ai programmi
- Siccome più programmi contemporaneamente possono richiedere l'accesso ad una determinata periferica (es.: l'output sul video di un terminale, l'input da tastiera, l'invio di dati ad una stampante, ecc.) è il sistema operativo a gestire in modo coordinato la **coda di richieste** (serializzazione) evitando conflitti e malfunzionamenti
- Il sistema operativo rende efficiente la comunicazione verso un determinato dispositivo periferico, anche mediante l'utilizzo di un'area di memoria detta ***buffer***, in cui si accodano i dati diretti verso la periferica o provenienti dalla periferica e destinati ai programmi
- L'interazione con le periferiche (*device*, dispositivi) avviene attraverso appositi moduli software detti ***device driver***



Gestione della memoria secondaria

- La memoria secondaria è costituita dai dispositivi di **memorizzazione persistente**, ossia che mantengono memoria delle informazioni registrate anche a macchina spenta
- La memoria secondaria è disponibile in **quantità molto superiore** rispetto alla memoria primaria (almeno 100 volte superiore su un normale personal computer, ad esempio)
- La memoria secondaria, per ragioni fisiche e tecnologiche, ha un **tempo di accesso e di trasferimento dei dati molto più alti** rispetto alla memoria primaria
- Tipici dispositivi di memoria secondaria (o *memoria di massa*): hard disk magnetici, dischi ottici, dispositivi di memoria a stato solido (SSD), flash drive, schede di memoria XD, ecc.
- Operazioni principali svolte dal sistema operativo per conto dei processi attivi:
  - allocazione/deallocazione dello spazio per la memorizzazione di dati (file)
  - gestione dello spazio libero sull'unità di memoria di massa
  - ottimizzazione, serializzazione e scheduling delle operazioni sulla memoria di massa
- La gestione del filesystem e della memoria secondaria sono due funzioni distinte del sistema operativo, anche se sono strettamente collegate
- Di fatto la componente del sistema operativo che si occupa della gestione della memoria secondaria, rende trasparente ai programmi il tipo di dispositivo di memoria utilizzato
  - Ad esempio in linguaggio C si usano le stesse chiamate di sistema per operare su file registrati su dispositivi di tipo differente

Gestione della protezione e della sicurezza

- In un sistema multi-task e multi-utente il sistema operativo si occupa della **protezione delle risorse**: la protezione è diretta a salvaguardare la riservatezza delle risorse dai processi e dagli utenti
- La protezione è basata su:
  - **autenticazione**: procedura di accertamento dell'identità di un utente
  - **autorizzazione**: procedura di accertamento del diritto di accedere ad una risorsa (in una determinata modalità) da parte di un utente o di un processo
- Le procedure di autorizzazione per i processi sono legate alle autorizzazioni assegnate agli utenti:
  - ogni processo è in esecuzione per conto di un utente
  - il processo eredita le autorizzazioni dell'utente che lo esegue per l'accesso alle risorse del sistema
  - i processi del sistema operativo vengono eseguiti da un utente con il massimo livello di autorizzazione (root, administrator, ecc.)
- I criteri di protezione delle risorse sono basati su regole che mappano le autorizzazioni di accesso alla risorsa con gli utenti del sistema (es.: autorizzazioni di accesso ad un file)
- Per semplificare la mappatura delle autorizzazioni, spesso il sistema operativo consente di aggregare gli utenti in gruppi, mappando poi le autorizzazioni sul gruppo: gli utenti ereditano le autorizzazioni che sono state assegnate ai gruppi di cui fanno parte

Gestione della protezione e della sicurezza

- I sistemi operativi multi-utente implementano procedure di **login** per l'accesso al sistema
- La procedura di login:
  - autentica l'utente sulla base delle credenziali di accesso dichiarate dall'utente stesso (username e password)
  - verifica se l'utente è autorizzato ad accedere al sistema
- La procedura di login è basata su un **repository di credenziali** degli utenti e di definizione dei gruppi di utenti o dei profili autorizzativi:
  - i file `/etc/passwd`, `/etc/shadow`, `/etc/group` in ambiente UNIX
  - il sistema RACF (*resource access control facility*) in ambiente Z/OS sui grandi mainframe IBM
  - directory server LDAP (*lightweight directory access protocol*) per condividere i criteri di autenticazione e autorizzazione tra i computer di uno stesso gruppo/dominio (es.: Microsoft Active Directory, UNIX Network Information System/Yellow Pages, ecc.)
  - altri sistemi di autenticazione e autorizzazione esterni
- NOTA: il meccanismo di autenticazione basato su *username* e *password* non è l'unico, né il più sicuro, ma è certamente il più diffuso
- La password non è mai memorizzata sul sistema:
  - della password sul sistema (file o directory server) viene conservato un *hash* non reversibile;
  - autenticazione: l'utente inserisce la password in chiaro, il sistema produce l'hash della password e lo confronta con quello memorizzato sul repository delle credenziali

Interfaccia verso le applicazioni e gli utenti

- Il sistema operativo offre delle funzioni alle applicazioni per costruire gli strumenti di dialogo ed interazione con l'utente
- *User interface*:
  - **Alfanumerica**: il sistema operativo offre un modello astratto di terminale per presentare le informazioni su uno schermo in grado di visualizzare **caratteri alfabetici e numerici** ed acquisire l'input attraverso una tastiera
  - ***Graphical User Interface*** (GUI): il sistema operativo offre un insieme di funzioni ai programmi con cui possono costruire un'interfaccia utente grafica, basata su elementi quali le **finestre** e le **icone**; l'input avviene anche mediante il *mouse* (o un *trackpad* o un *touch-screen*) e non solo la tastiera
- Alcuni sistemi operativi sono strettamente legati ad una specifica interfaccia utente grafica: in questi casi il sistema operativo non funziona senza la sua Graphical User Interface
  - Esempi: Microsoft Windows, Apple OS X
- Altri sistemi operativi invece prevedono di base solo un'interfaccia utente alfanumerica e implementano l'interfaccia utente grafica come un *add-on* non indispensabile, basato su un modello client/server
  - Esempi: i sistemi operativi UNIX e Linux e il sistema X Window

Interfaccia alfanumerica e terminali

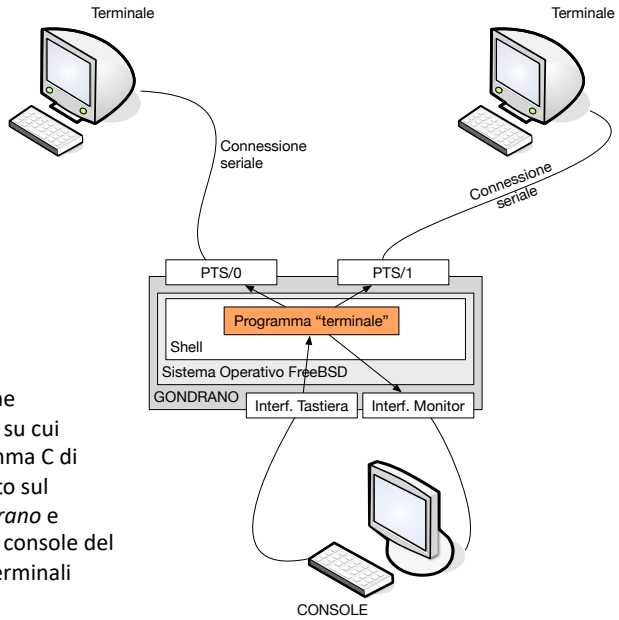
- I terminali alfanumerici vengono originariamente collegati alle porte seriali dei mini-computer o dei mainframe
- Tra i terminali alfanumerici lo standard è il **VT100** della *Digital Equipment Corp.*: aveva un display con cui visualizzare **24 righe** da **80 caratteri** ciascuna e una tastiera
- Il sistema operativo mette a disposizione dei «driver» per utilizzare i terminali come interfacce di I/O
- Il sistema operativo UNIX rende disponibili i terminali come device «`/dev/pts/0`», «`/dev/pts/1`», …
- Esempio:

```
$ echo ''ciao'' > /dev/pts/0
```

- In linguaggio C

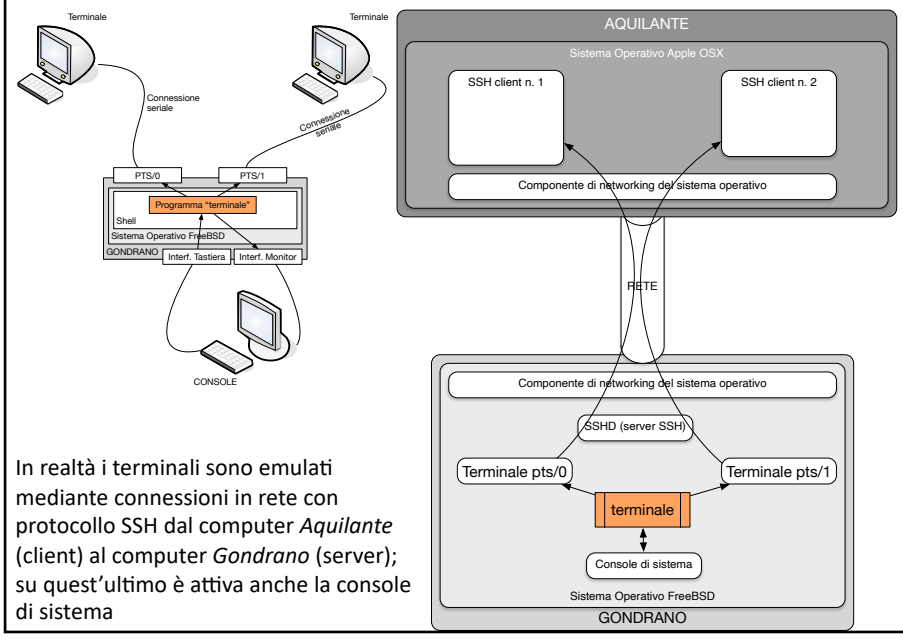
```c
#include <stdlib.h>
#include <stdio.h>

int main(void) {
  FILE *TERM0, *TERM1;
  char stringa[100];

  TERM0 = fopen("/dev/pts/0", "wt");
  TERM1 = fopen("/dev/pts/1", "wt");
  fprintf(TERM0, "\n\nTERMINALE n. 0\n\n");
  fprintf(TERM1, "\n\nTERMINALE n. 1\n\n");

  while (1) {
    printf(">>> ");
    scanf("%s", stringa);
    fprintf(TERM0, "\n--\n%s\n--\n", stringa);
    fprintf(TERM1, "\n--\n%s\n--\n", stringa);
  }
  fclose(TERM0);
  fclose(TERM1);
  return(0);
}
```

Interfaccia alfanumerica e terminali



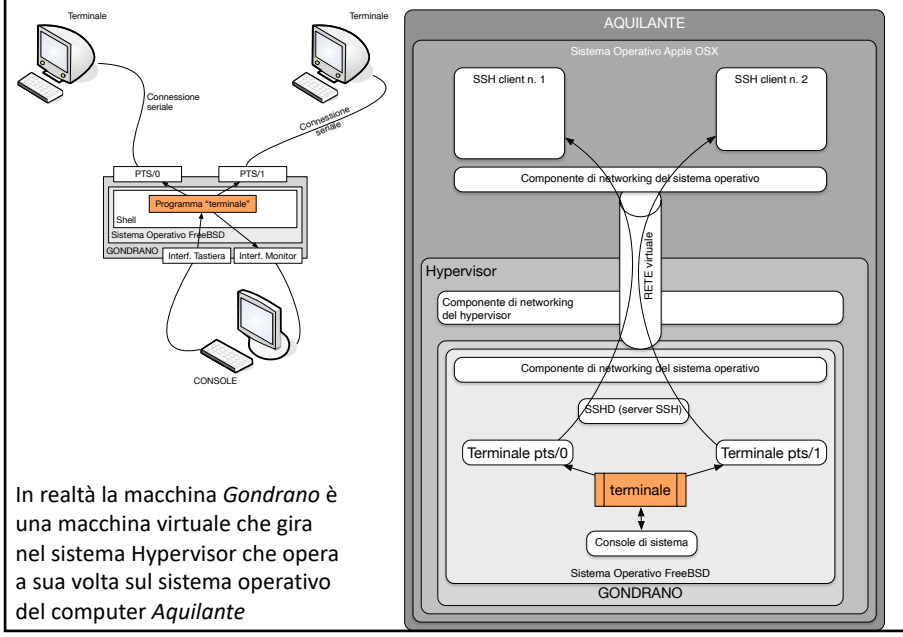
Schematizzazione dell'architettura su cui opera il programma C di esempio eseguito sul computer *Gondrano* e connesso con la console del sistema e due terminali alfanumerici

Interfaccia alfanumerica e terminali



In realtà i terminali sono emulati mediante connessioni in rete con protocollo SSH dal computer *Aquilante* (client) al computer *Gondrano* (server); su quest'ultimo è attiva anche la console di sistema

Interfaccia alfanumerica e terminali



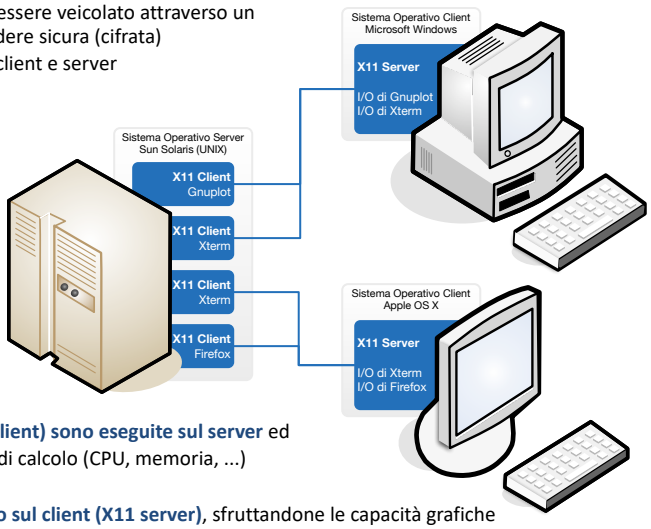
In realtà la macchina *Gondrano* è una macchina virtuale che gira nel sistema Hypervisor che opera a sua volta sul sistema operativo del computer *Aquilante*

Interfaccia verso le applicazioni e gli utenti

- Il sistema **X Window** (X11) è un sistema *client-server* che implementa un'interfaccia utente grafica distribuita:
  - è **portabile**: ne esistono versioni per ogni sistema operativo, non solo UNIX
  - il modello client/server separa l'ambiente di esecuzione dell'applicazione che sfrutta la graphical user interface dall'ambiente in cui la graphical user interface viene utilizzata dall'utente (con display grafico, mouse, tastiera, ecc.)
  - è **indipendente dal window manager** e dal **desktop manager**, l'utility che aiuta l'utente a gestire le finestre sul proprio display grafico con l'uso del mouse
- **Client X11**: l'applicazione che utilizza le API del sistema X11 per acquisire input attraverso mouse e tastiera e produrre output in un ambiente grafico costituito da più finestre visualizzate contemporaneamente
  - Il client X11 spesso è un server su cui girano i programmi lanciati dall'utente attraverso il proprio Terminale grafico X11
- **Server X11**: l'applicazione che accetta connessioni dai client e gestisce le periferiche di input (mouse, tastiera, ...) e di output (uno o più display grafici) connesse al terminale grafico (Terminale X11) dell'utente
  - Il terminale grafico dell'utente può essere anche un personal computer con un suo sistema operativo (Microsoft Windows, Linux, Apple OS X, ...) che esegue il programma Server X11
- Nel sistema X Window il client ed il server X11 **comunicano attraverso la rete**, anche se sono eseguiti sullo stesso computer

Interfaccia verso le applicazioni e gli utenti

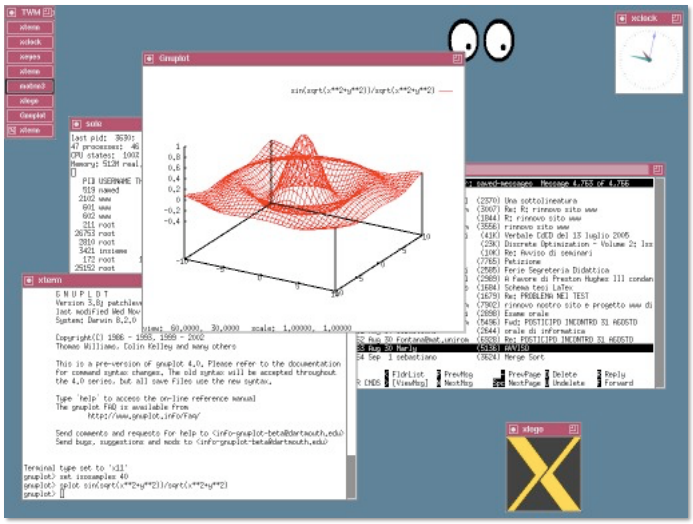
- Il sistema **X Window** implementa il **protocollo X11** per la comunicazione tra **client grafico** (che opera sul server applicativo) e **server grafico** (che opera sulla postazione client dell'utente)
- Il protocollo X11 può essere veicolato attraverso un **"tunnel" SSH** per rendere sicura (cifrata) la comunicazione tra client e server



- Le **applicazioni (X11 client) sono eseguite sul server** ed impegnano le risorse di calcolo (CPU, memoria, ...) del server
- L'**output è visualizzato sul client (X11 server)**, sfruttandone le capacità grafiche

Bibliografia essenziale

1. Stephen R. Bourne, *UNIX System V*, Addison-Wesley, 1990.
2. Brian W. Kernighan, Rob Pike, *The UNIX Programming Environment*, Prentice Hall, 1984
3. Marc J. Rochkind, *Advanced UNIX programming*, Prentice Hall, 1985
4. Andrew S. Tanenbaum, *I moderni sistemi operativi*, terza edizione, Pearson, 2009.



TWM X11 Window Manager

Una schermata del display di una workstation grafica X11 in ambiente UNIX con il window manager TWM