Lezione 4 (20-02-25)

Protection and Security

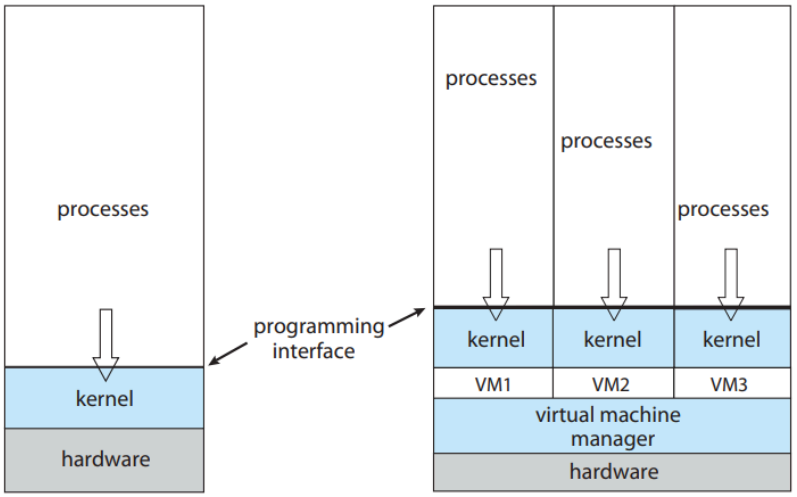
L'accesso ai dati deve osservare alcune regole. Vi sono meccanismi che assicurano che i file, i segmenti di memoria, la CPU e le altre risorse possano essere manipolati solo dai processi che abbiano ottenuto apposita autorizzazione dal SO.
Per **protezione** intendiamo ciascun meccanismo di controllo dell'accesso alle risorse possedute da un elaboratore, da parte di processi o utenti. La protezione può migliorare l'affidabilità rilevando errori nascosti alle interfacce tra i componenti del sottosistema e può prevenire la contaminazione di un sottosistema sano a opera di un altro sottosistema infetto.
Nonostante la protezione, un sistema può comunque essere esposto agli accessi abusivi, rischiando malfunzionamenti.
È compito della **sicurezza** difendere il sistema da attacchi provenienti dall'interno o dall'esterno (es. virus e worm, attacchi denial-of-service). Per alcuni sistemi l'attività di prevenzione da alcuni di questi attacchi è considerata una funzione del sistema, altri delegano la difesa a regole operative o programmi aggiuntivi.
Protezione e sicurezza presuppongono che il sistema sia in grado di distinguere tra tutti i propri utenti. Nella maggior parte dei SO è disponibile un elenco di nomi degli utenti e dei loro identificatori (user id). Sono id numerici che identificano univocamente l'utente, questo id è associato a tutti i processi e i thread del soggetto in questione.
In alcune circostanze è preferibile distinguere tra gruppi di utenti, ad esempio quando dobbiamo gestire l'accesso ad un file.
Un utente può far parte di uno o più gruppi e l'identificatore di gruppo è incluso in tutti i processi e i thread a esso relativi.
Durante un normale utilizzo del sistema sono sufficienti un id utente e un id del gruppo, talvolta potrebbe servire ***scalare i privilegi***, ovvero ottenere permessi ausiliari per certe attività.

Virtualization

La **virtualizzazione**:

- è una tecnica che permette di astrarre l'hardware di un singolo computer in diversi ambienti di esecuzione.
- Un utente di una macchina virtuale può passare da un SO a un altro esattamente come può commutare tra processi in esecuzione contemporaneamente in un singolo SO.
- permette ai SO di funzionare come applicazione all'interno di altri SO.
  In senso lato, possiamo dire che appartiene a una tipologia di software che include anche l'**emulazione**, tecnica utilizzata quando il tipo di CPU origine è diverso dal tipo di CPU destinazione. L'emulazione ha un costo molto elevato, poiché ogni istruzione che può essere eseguita nativamente sul sistema sorgente deve essere tradotta nell'equivalente funzione da eseguire sul sistema di destinazione e ciò richiede spesso l'utilizzo di diverse istruzioni di destinazione. Se le due CPU hanno prestazioni simili, allora la CPU di destinazione non ha abbastanza potenza extra per compensare l'overhead dell'emulazione, di conseguenza il codice emulato viene eseguito molto più lentamente rispetto al codice nativo.
  Con la virtualizzazione invece un SO compilato per una particolare architettura viene eseguito all'interno di un altro SO progettato per la stessa CPU.

Un VMM (*Virtual Machine Manager*) permette all'utente di installare su un computer portatile o desktop più sistemi operativi e di eseguire applicazioni scritte per SO diversi da quello installato nativamente.



Distributed Systems

Per **sistema distribuito** si intende un insieme di elaboratori fisicamente separati e con caratteristiche spesso eterogenee, interconnessi da una rete per consentire agli utenti l'accesso alle varie risorse die singoli sistemi. L'accesso a una risorsa condivisa aumenta la velocità di calcolo, la funzionalità, la disponibilità dei dati e il grado di affidabilità.
Una rete si può considerare un canale di comunicazione tra due o più sistemi. Differiscono per i protocolli usati, per le distanze tra i nodi e per il mezzo attraverso il quale avviene la comunicazione; il più diffuso è il **TCP/IP**.
Le reti si classificano secondo le distanze tra i loro nodi:

- **LAN** (*Local Area Network*), rete locale: collega nodi all'interno della stessa stanza (distanza breve);
- **WAN** (*Wide Area Network*), rete geografica: si estende a gruppi di edifici, città o al territorio, possono funzionare con uno o più protocolli (distanza lunga);
- **MAN** (*Metropolitan Area Network*), reti metropolitane: possono collegare gli edifici all'interno di una città (es. wi-fi) (distanza media);
- **PAN** (*Personal Area Network*), reti personali: ad esempio, tra un telefono e le cuffie bluetooth (distanza molto breve).

Protocollo := un protocollo di rete è un insieme di regole che definiscono come i dispositivi comunicano tra loro all'interno di una rete.

TCP/IP := (*Transmission Control Protocol / Internet Protocol*) è il protocollo standard usato per comunicare su internet. Due parti principali:

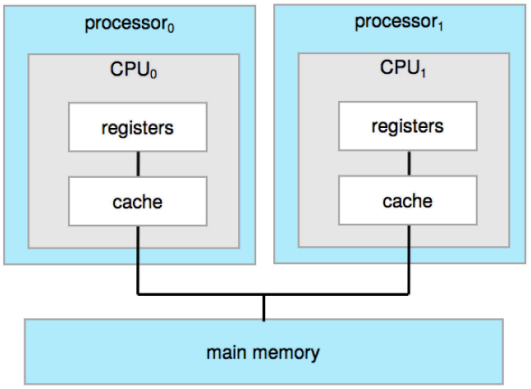
- **IP** → si occupa dell'instradamento dei pacchetti di dati tra i dispositivi sulla rete, assegnando un indirizzo univoco a ogni nodo
- **TCP** → garantisce che i dati vengano inviati correttamente e nell'ordine giusto

Nodo := un nodo in una rete è qualsiasi dispositivo connesso.
Distanza tra nodi := indica quanto sono lontani fisicamente o logicamente i dispositivi connessi.

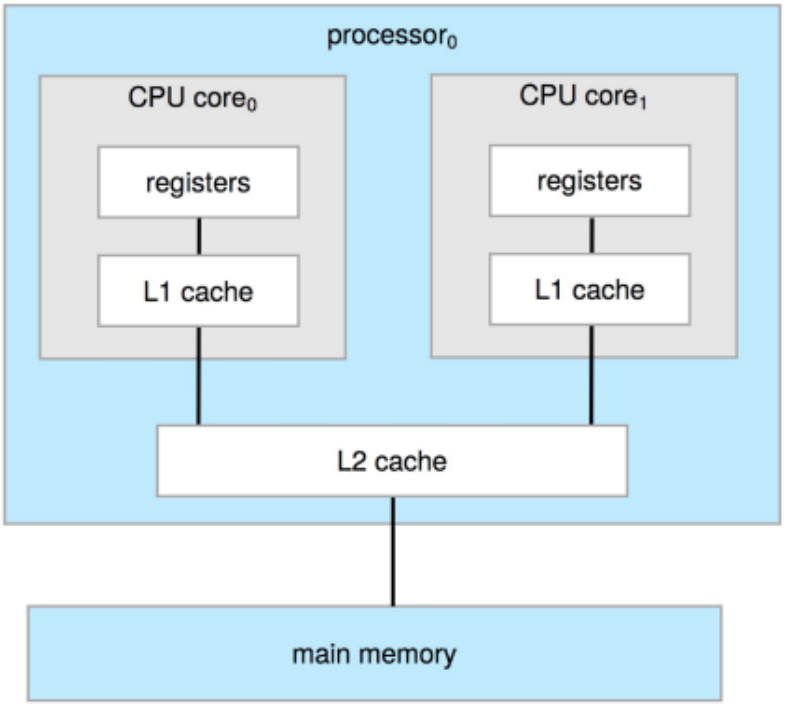
Computer-System Architecture

Fino a qualche anno fa molti sistemi usavano un singolo processore. Oggi si usano più processori, detti *multiprocessori*, che possono fare più operazioni contemporaneamente e si migliorano le prestazioni.

- **multiprocessore *asimmetrico***: c'è un singolo processore che coordina tutti gli altri processori.
- **multiprocessore *simmetrico***: tutti i processori sono allo stesso livello
  Architettura multiprocessore simmetrico:

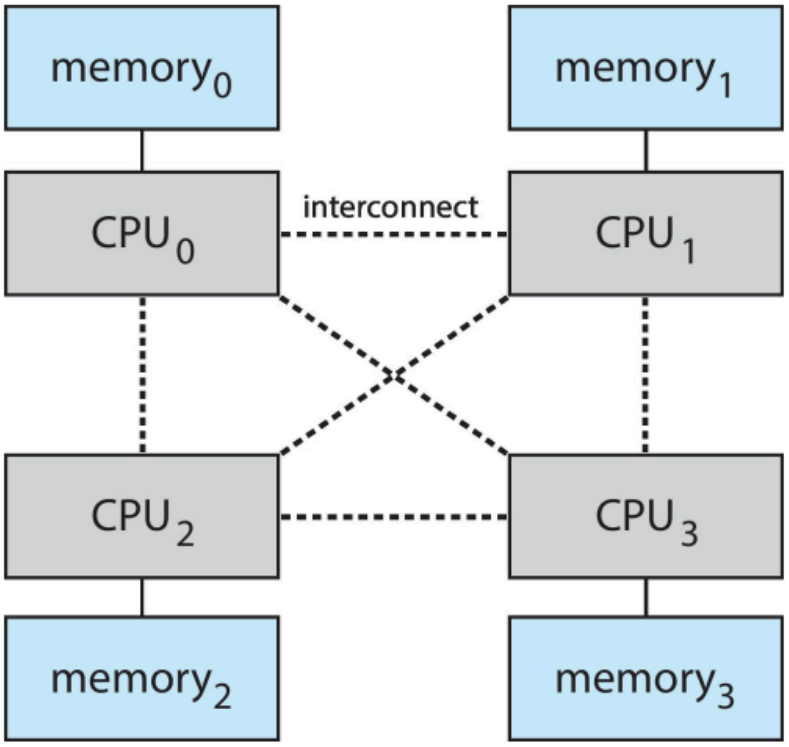


  Notiamo che ciascuna CPU ha il proprio set di registri e una cache privata (locale), tuttavia condividono la memoria fisica sul bus di sistema.
  La definizione di multiprocessore si è evoluta nel tempo e ora comprende anche i sistemi **multicore**, in cui più unità di calcolo (core) risiedono su un unico chip. Possono essere più efficienti rispetto a più chip dotati di una singola unità di calcolo, perché la comunicazione all'interno di un singolo chip è più veloce rispetto a quella tra un chip e un altro.
  Architettura dual-core, con due unità sullo stesso chip:



  Un approccio alternativo è quello di fornire a ciascuna CPU la propria memoria locale accessibile per mezzo di un bus locale piccolo e veloce. Le CPU sono collegate da un'interconnessione di sistema condivisa, in modo che tutte le CPU condividano uno spazio di indirizzamento fisico. Si tratta dell'accesso non uniforme alla memoria, o **numa**, il vantaggio è che quando una CPU accede alla sua memoria locale, non solo è veloce, ma non vi è alcun conflitto sull'interconnessione di

sistema.



Clustered Systems

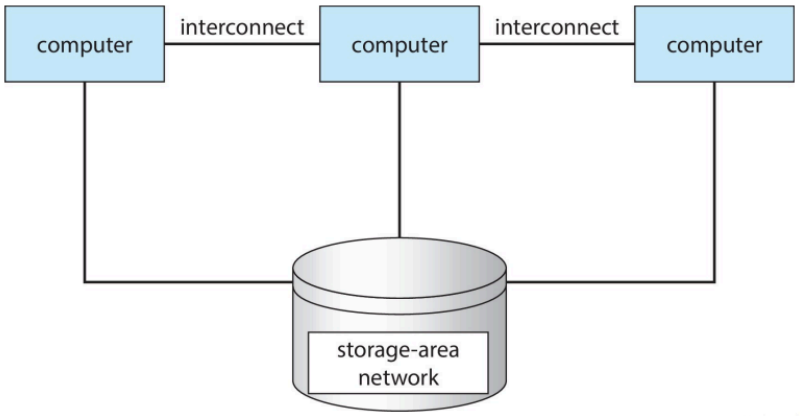
I sistemi cluster funzionano in modo simile ai sistemi multiprocessore, ma coinvolgono più sistemi indipendenti che collaborano tra loro.

- Solitamente condividono lo storage attraverso una Storage-Area Network (SAN), che è una rete specializzata per l'archiviazione centralizzata e veloce dei dati.
- Offrono un servizio ad alta disponibilità (high-availability, HA), il che significa che il sistema può continuare a funzionare anche se uno o più nodi si guastano.
  Tipologie di clustering:

1. **Clustering asimmetrico (*Asymmetric clustering*)**
   - un nodo è in modalità di **hot-standby**, ossia è inattivo fino a quando il nodo principale non si guasta;
   - quando il nodo principale fallisce, quello in standby prende il suo posto, garantendo la continuità del servizio;
   - è usato spesso nei sistemi mission-critical dove è fondamentale la ridondanza (es. sistemi bancari).
2. **Clustering simmetrico (Symmetric clustering)**
   - più nodi eseguono applicazioni contemporaneamente;
   - ogni nodo monitora lo stato degli altri, in modo che, se uno fallisce, gli altri possano intervenire e distribuire il carico;
   - offre maggiore efficienza rispetto al clustering asimmetrico, poichè tutte le macchine vengono utilizzate attivamente.
   Clustering per High-Performance Computing (HPC)

- Alcuni cluster non servono solo per ridondanza e affidabilità, ma per calcolo ad alte prestazioni (HPC).

- In questi sistemi, le applicazioni devono essere progettate per parallelizzare i processi, cioè dividere il lavoro tra più nodi per accelerare i calcoli.
  Distributed Lock Manager (DLM)
- In alcuni cluster, esiste un Distributed Lock Manager per gestire le operazioni concorrenti.
- Serve a evitare conflitti, per esempio quando più nodi vogliono accedere e modificare la stessa risorsa contemporaneamente.
- Funziona bloccando temporaneamente l'accesso a una risorsa fino a quando un'operazione non è completa.



Ambienti d'elaborazione

Sistemi tradizionali:

- sono macchine stand-alone (autonome) e general-purpose.
- tuttavia, la distinzione sta diventando meno netta perché la maggior parte dei sistemi è connessa al altri, ad esempio tramite internet.
  Esempi:
  - portali web → consentono l'accesso a sistemi interni via browser
  - network computers (thin clients) → dispositivi con risorse limitate che dipendono da server remoti per l'elaborazione.
  - computer mobili → dispositivi come laptop e smartphone connessi tramite reti wireless.
  L'interconnessione sta diventando onnipresente, al punto che anche i sistemi domestici utilizzano firewall per proteggere i computer dagli attacchi proveniente da internet.

Mobile Computing:
Dispositivi come smartphone e tablet rientrano nel mobile computing.
Differenze rispetto ai laptop tradizionali:

1. Funzionalità extra
   - hanno sensori come GPS, giroscopio, accelerometro;
   - consentono lo sviluppo di applicazioni avanzate come la realtà aumentata.
2. Connettività
   - si connettono tramite Wi-Fi (IEEE 802.11) o reti cellulari;
3. Ecosistema software:
   - i principali sistemi operativi mobili sono Apple iOS e Google Android.

Client-Server Computing:
Un tempo esistevano *dumb terminals* che si limitavano a ricevere output da un mainframe. Oggi, questi

sono stati sostituiti da PC intelligenti con capacità di elaborazione proprie.
Ora, molti sistemi operano come server, rispondendo alle richieste dei client.

- *computer-server* → fornisce servizi computazionali (es. database);
- *file-server* → permette ai client di memorizzare e recuperare file.

Peer-to-Peer (P2P) Computing:
È un altro modello di sistema distribuito, diverso dal classico client-server.
Caratteristiche principali:

- non esiste una netta distinzione tra client e server;
- ogni nodo è un peer (pari) che può fungere da client, server o entrambi;
- un nodo deve registrarsi in un servizio centrale o trasmettere richieste sulla rete per scoprire altri peer.
  Esempio: Skype per la comunicazione audio/video decentralizzata.

Cloud Computing:
Il cloud computing fornisce risorse di calcolo, storage e applicazioni come servizio su una rete.
È un'estensione logica della virtualizzazione, perché sfrutta la virtualizzazione per offrire servizi scalabili e on-demand.
Esempio: Amazon EC2 → piattaforma cloud con migliaia di server e milioni di macchine virtuali, dove gli utenti pagano solo per risorse che utilizzano.
Tipologie:

1. *Public Cloud* → disponibile via internet per chiunque paghi il servizio (es. Google Drive);
2. *Private Cloud* → utilizzato esclusivamente da un'azienda per i propri scopi;
3. *Hybrid Cloud* → combinazione di public e private cloud.
   Modelli di servizio nel cloud:

- **SaaS** *Software as a Service* → applicazioni accessibili via internet (es. Google Docs);
- **PaaS** *Platform as a Service* → ambiente software pronto per l'uso (es. database server);
- **IaaS** *Infrastructure as a Service* → server virtuali o storage offerti come servizio (es. Amazon S3).
  Il cloud computing richiede strumenti di gestione avanzati e misure di sicurezza come firewall e load balance per proteggere i dati e distribuire il carico di lavoro tra più applicazioni.

Real-Time Embedded Systems:
I sistemi embedded real-time sono la forma più diffusa di computer.
Caratteristiche:

- Sono dispositivi specializzati con un sistema operativo dedicato o, in alcuni casi, funzionano senza un SO.
- Hanno vincoli temporali fissi: il loro funzionamento è corretto solo se le operazioni vengono completate entro un tempo predefinito.
- Sono sempre più diffusi in applicazioni come automobili, elettrodomestici, dispositivi medici e robotica.

Free and Open-Source Operating Systems

I SO possono essere distribuiti non solo in forma binaria chiusa e proprietaria, ma anche con codice sorgente accessibile.

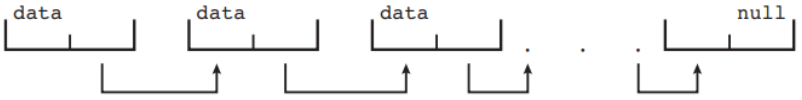
Questa filosofia è contrapposta ai concetti di protezione della copia e Digital Rights Management (DRM), che cercano di limitare l'uso e la distribuzione del software.
Il movimento è iniziato con la Free Software Foundation (FSF), che ha introdotto la licenza GNU General Public License (GPL), basta sul concetto di copyleft, ovvero l'idea che un software libero debba rimanere tale, garantendo che qualsiasi modifica o distribuzione mantenga gli stessi diritti di libertà.
Tuttavia, il software libero e il software open-source non sono esattamente la stessa cosa:

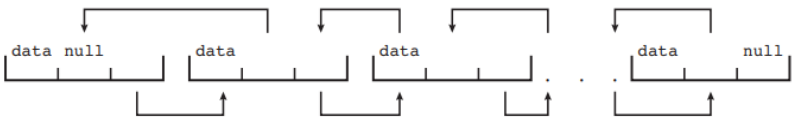
- il software libero si concentra sulla liberta degli utenti di usare, studiare, modificare e condividere il software.
- il software open-source si focalizza più sui vantaggi tecnici e sulla collaborazione nel codice.
  Esempi di SO open-source: GNU/Linux e BSD UNIX (base del kernel di macOS).
  I SO open-source possono essere esplorati in ambienti virtuali, senza doverli installare fisicamente.
  Strumenti comuni: VMware Player e VirtualBox. Essi permettono di eseguire sistemi operativi guest su una macchina host, rendendo più semplice sperimentare con diversi SO senza modificare il sistema principale.

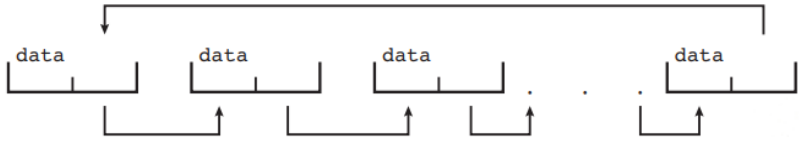
Kernel Data Structures

Liste, Stack e Code

Array:

- struttura dati in cui ogni elemento è direttamente accessibile;
- la memoria principale è costruita come un array.
  Liste:
- i dati sono accessibili solo in un particolare ordine;
- rappresenta una collezione di valori in sequenza;
- ne esistono diverse tipologie:
  - *lista semplicemente concatenata*: ogni elemento punta all'elemento successivo;
    

  - *lista doppiamente concatenata*: ogni elemento contiene riferimenti sia al suo successore sia al suo predecessore;
    

  - *lista circolare*: l'ultimo elemento punta al primo elemento piuttosto che a un valore null.
    

- possono essere utilizzate dagli algoritmi del kernel, ma anche per costruire strutture dati come *code* e *stack*.
  Stack:
- struttura dati ordinata sequenzialmente;;
- tipo di accesso LIFO;

- viene spesso utilizzato dal SO per gestire le chiamate di funzione.
  Coda:
- struttura dati ordinata sequenzialmente;
- tipo di accesso FIFO;
- esempio di utilizzo: documenti inviati a una stampante che vengono solitamente processati nell'ordine in cui sono stati ricevuti.

Alberi

Un albero è una struttura dati utilizzabile per rappresentare i dati in maniera gerarchica. Gli elementi sono strutturato secondo una relazione padre-figlio.

- albero generico: il padre può avere molteplici figli;
- albero binario: il padre può avere solo due figli;
- albero binario di ricerca: come per gli alberi binari, ma con una proprietà aggiuntiva: i figli sono ordinati in modo che $figlio.sinistro <= figlio.destro$.

Funzioni e Mappe Hash

- Una funzione hash riceve dati in input, realizza operazioni numeriche sui dati e restituisce un valore numerico.
- Questo valore numerico può essere utilizzato come indice in una tabella (solitamente un array) per recuperare velocemente il dato.
- Il problema delle collisioni (dati diversi producono lo stesso valore numerico) può essere risolto inserendo nella locazione una lista concatenata contenente tutti gli elementi con lo stesso valore hash.

Bitmap

- Stringa di n caratteri binari utilizzabile per rappresentare lo stato di n elementi.
- Molto efficiente in termini di spazio utilizzato.
- Spesso utilizzate quando si ha la necessità di rappresentare la disponibilità di un gran numero di risorse.