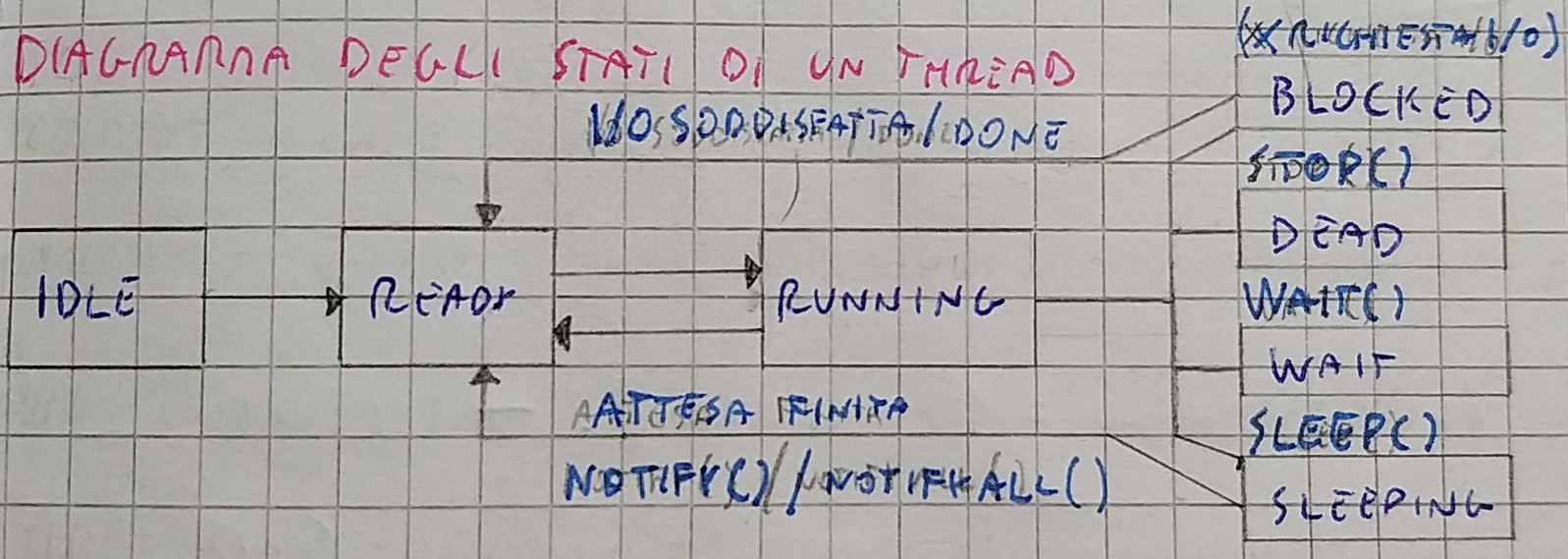
THREAD

Sono detti processi leggeri (LWP) perché per evolversi non necessitano di risorse, ma solo della CPU.

TCB (Thread Control Block)

* TID (Thread Identifier) --> Come il PID ma per i thread, li identifica univocamente.
* PC --> Un Program Counter/Instruction Pointer per sapere quale è la prossima istruzione de eseguire.
* Registri.
* Stato --> idle/ready/running/blocked/dead/wait/sleeping.
* Contesto nel quale è salvato quando inutilizzato.
* Stack.
* Spazio di memoria per variabili locali.
* Pointer a PCB del padre.

Diagramma degli stati di un thread



User Level Threads

Vantaggi

* Non necessita di fare chiamate al kernel --> + efficiente

Svantaggi

* Ogni volta che c’è system call per accedere a dispositivo I/O il thread (e quindi il processo) è sospeso.
* Impossibile il parallelismo fisico in architetture multiprocessore.

Kernel Level Threads

Vantaggi

* Schedulati in modo autonomo (se 1 è sospeso gli altri continuano).
* Parallelismo fisico possibile.
* Kernel può essere sfruttato come un sistema multithread.

Svantaggi

* Equiparati e gestiti come processi (quindi deve gestire contemporaneamente processi e thread).

Fork

Con il C si possono scrivere programmi fatti da insiemi di processi sequenziali asincroni interagenti, quindi è possibile collaudare processi paralleli che indifferentemente dall’ordine in cui essi vengono eseguiti, dato che non cambierà i risultati in output (detti con ordinamento parziale). La macchina parallela astratta che c’è nei sistemi monoproc, è:

* Macchina astratta,
* Software multiprogrammazione,
* Software sincronizzazione,
* Nucleo (esecutore del programma in linguaggio concorrente avente meccanismi di sync per app multiprog).

Con la fork() si possono generare gerarchie di processo con alcuni costrutti principali:

* fork – join,
* cobegin – coend.

Funzione che crea un processo figlio identico al padre. In entrambi ritorna un parametro int: nel padre ritorna il PID del processo appena creato, mentre nel figlio ritorna 0.

Terminazione di un processo

Quando un processo termina restituisce un exit status, solitamente un int a 16 bit.

Terminazione volontaria

Quando ha eseguito tutte le sue istruzioni oppure quando viene chiamata la funzione exit().

Byte meno significativo di status == 0 --> Byte più significativo == exit status.

Exit status < 256.

Terminazione involontaria

Quando cerca di eseguire operazioni non permesse o viene interrotto.

Byte meno significativo di status descrive il segnale che ha fatto terminare il processo e il suo valore è forzato dal SO.

Se exit status > 256, va diviso per 256 per ottenere valori dei 2 byte separati.

PID padre e figlio

Ogni figlio ricorda il PID del padre o *parent PID*, detto anche PPID. Con *getpid()* si ottiene il PID proprio del processo che ha eseguito l’istruzione, mentre con *getppid()* si ottiene il PID del padre che processo. Con *ps -e* vedo tutti i processi in esecuzione e i relativi PID.

*Sleep()* permette di “addormentare” il processo per un tot di secondi, va inserita per evitare che il padre termini prima del figlio lasciandolo orfano. Avendo n fork, il numero totale dei processi (padre compreso) sarà 2n; con la *sleep* non avviene alcuna sincronizzazione, ma si ritarda l’esecuzione dei processi in modo che terminino dopo i figli.

Elaborazione concorrente

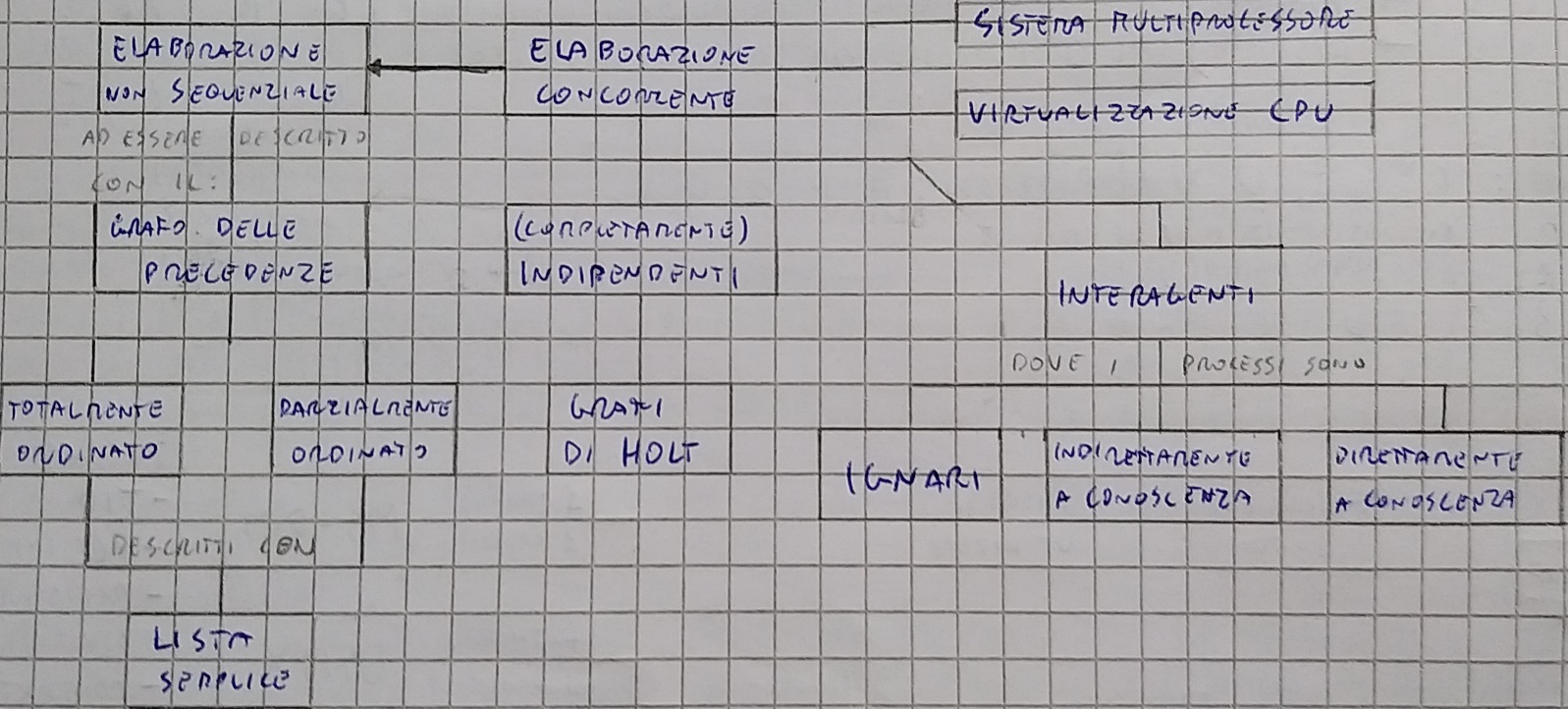
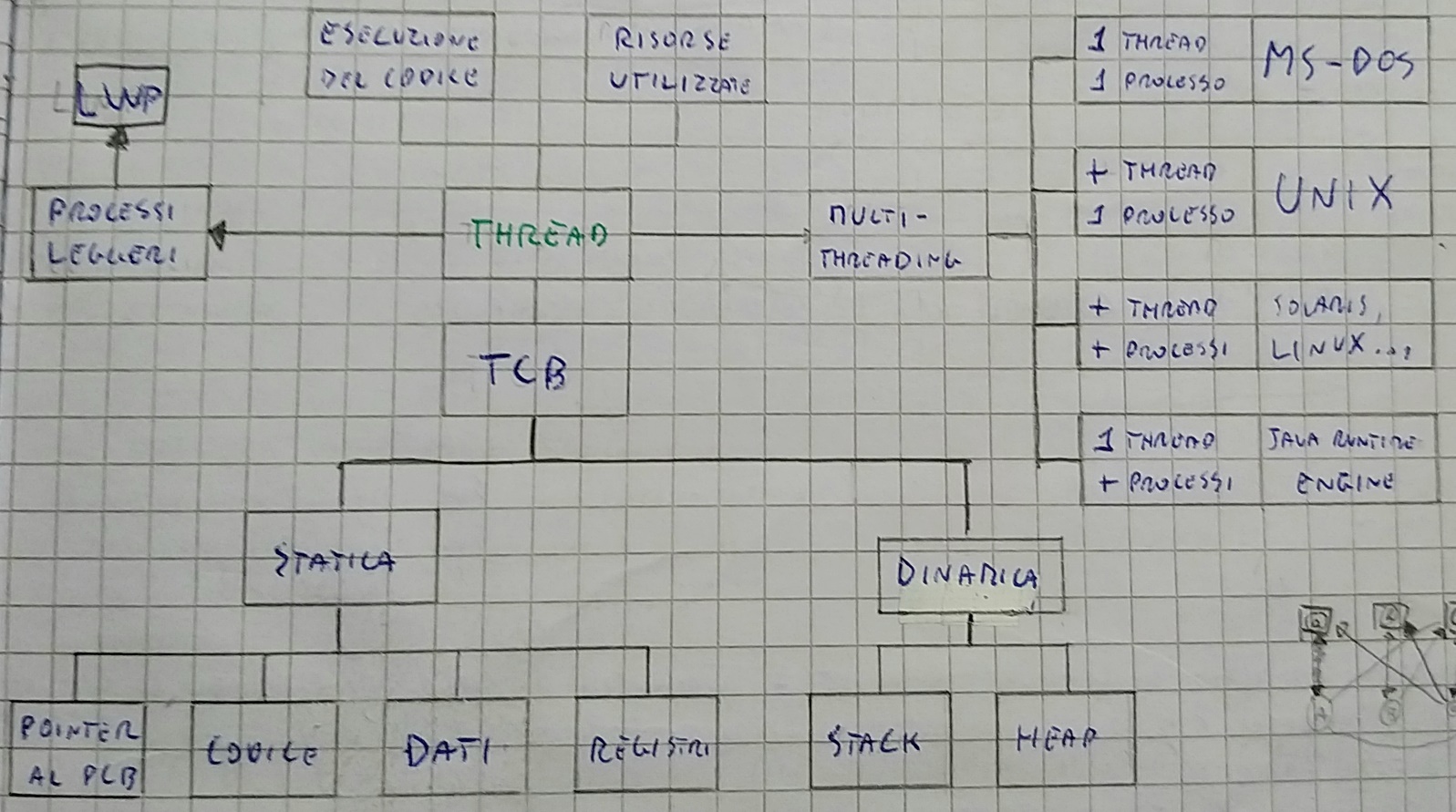


Diagramma thread



Background e Foreground

Sono processi di servizio che forniscono servizi ai processi di foreground. (Tipo timer per app principale C#).

MODELLI DI INTERAZIONE

Modelli di interazione

Individuabili anche in macchine monoprocessore basate sul modello di von Neumann, ovvero macchine SISD.

- A memoria comune (ambiente globale)

Memoria comune per tutti i processi. Usato per macchine monoprocessore multitasking. Complesso da implementare in sistemi distribuiti. Rischio problemi quando 2 processi richiedono accesso a stessa risorsa condivisa.

- A scambio di messaggi

Usata in sistemi distribuiti (+ macchine ma in luoghi diversi) e macchine comunicano con messaggi. Utilizzate le funzioni Send(dest, mess) e Recieve(source, mess). Mess è composto da: ***Header****(dest, source, type, priority, length…)* e **Messaggio** (string). Queste possono essere funzioni **bloccanti (sincrone)** o **non** **bloccanti (asincrone)**.

|  |  |  |
| --- | --- | --- |
| Funzione | **Bloccante/Sincrona** | **Non bloccante/Asincrona** |
| **Send** | Quando processo che la esegue si blocca finché non confermato che mess è ricevuto. | Dopo mess, prosegue esecuzione. |
| **Recieve** | Quando processo si ferma in attesa di un messaggio dal mittente. | Non attende mess: 1) mess inviato prima di Recieve() (riceve se fa Recieve), 2) mess inviato dopo Recieve() (perdita mess) |

**Rendez-vous** --> solo per comunicazione **sincrona/bloccante**

Incontro tra 2 o + processi in un punto specifico del codice nel quale questi si scambiano messaggi tramite un canale.

- **Stretto**: Si limita alla trasmissione del messaggio da mittente a destinatario.

- **Esteso**: Una volta inviato il messaggio, il mittente aspetta una risposta (*ack*) da parte del destinatario.

**Comunicazione**

- **Asimmetrica**: **Mittente** **nomina** **esplicitamente** **destinatario**, ma destinatario **non** nomina esplicitamente mittente.

- **Simmetrica**: Entrambi si nominano esplicitamente.

**Indirizzamento**

- **diretto:**

P0 (in pc 0) fa la Send direttamente a P1 (pc 1) che fa la Recieve e riceve il mess direttamente.

- **indiretto:**

P0 fa la Send in una **coda**, dove possono venir salvati multipli mess. P1 con Recieve prende messaggi 1 per volta.

Gestori (o allocatori)

Segmenti di codice che gestiscono tutte le richieste fatte da diversi processi che richiedono la risorsa a cui è assegnato; quindi questo assegna o nega la risorsa in base a stato e se sta già venendo usata.

Compiti del gestore:

- Tenere aggiornato lo stato di allocazione della risorsa,

- Fornire a processi che richiedono risorsa i meccanismi per accedervi,

- Implementare la strategia di allocazione della risorsa definendo a quale processo e per quanto tempo.

Risorse

Qualunque oggetto, fisico o logico, di cui un processo necessita per portare a termine il suo compito.

|  |  |  |
| --- | --- | --- |
| Risorse | **Dedicate** (visibili sempre da 1 processo) | **Condivise** (visibili sempre da + processi) |
| **Statiche** | **Private** | **Comuni** |
| **Dinamiche** | **Comuni** | **Comuni** |

**Statiche**: gestore = programmatore, perché definite prima che programma starta.

**Dedicate**: gestore = SO, perché utilizzo esclusivo di 1 solo processo e programmatore non serve.

**Condivise**: gestore = ?, programmatore definisce solo regole di visibilità (pubbliche o private) e modalità di sync.

Anche un’**istanza di struttura dati allocata in memoria comune** è una **risorsa**.

Dedicate --> sincronizzazione non necessaria.

Condivise --> sincronizzazione necessaria, assicurarsi che accessi eseguiti in maniera **non divisibile** (mutex).

Competizione

Interazione tra processi PREVEDIBILE e INDESIDERATA.

- risorse **condivise statiche**: competizione al momento di accesso a risorsa

- risorse **dedicate dinamiche**: competizione al momento della richiesta di utilizzo

Soluzione: sincronizzazione **indiretta** o **implicita**.

Cooperazione

Interazione tra processi PREVEDIBILE e DESIDERATA (insita in logica del programma).

- risorse **condivise statiche**: logica di **produttore e consumatore** (produttore scrive, consumatore legge da risorsa)

- risorse **dedicate dinamiche**: logica di **lettore e scrittore**, (cooperazione c’è quando gestore **sa** che processi cooperano, quindi non cancella risultato prodotto in risorsa da P1 cosicché P2 possa usarlo.

Soluzione: sincronizzazione **diretta** o **esplicita**.

Interferenza

Interazione tra processi IMPREVEDIBILE e INDESIDERATA. Causata da:

- competizione tra + processi che utilizzano risorse condivise senza opportune autorizzazioni,

- errata soluzione per problema di competizione e cooperazione.

Errori in programmi concorrenti

Oltre a correttezza **logica** dei programmi, verificare anche quella **temporale**, infatti se in programmi sequenziali i risultati variavano solo in base a un cambiamento di input, ora con programmazione concorrente, se non uso tecniche di **sincronizzazione** adatte, risultati di un’elaborazione rischiano di variare anche solo in **base al tempo** (tipo quale processo viene eseguito prima). Un obiettivo di questa programmazione è **sfruttare al max la CPU**, quindi non posso inserire **ritardi** **in** **esecuzione** (cmq non garantita la correttezza dell’esecuzione) ma è una tecnica inaccettabile.

Vincoli

- Determinismo: i risultati devono essere uguali per ogni esecuzione,

- Timing costraints: i risultati vanno prodotti entro certi limiti di tempo (***deadlines***).

Caratteristiche

1) Irriproducibili, ovvero possono verificarsi con alcune sequenze e non con altre;

2) Indeterminati, ovvero esito ed effetti dipendono dalla sequenza;

3) Latenti, ovvero possono presentarsi soltanto con sequenze rare;

4) Difficili da verificare testare, poiché tecniche di verifica e testing si basano sulla riproducibilità del comportamento.

Overlapping

Quando più processi vengono eseguiti **contemporaneamente** e la loro esecuzione si **sovrappone** **nel** **tempo** (ciò grazie a CPU multiple di **sistemi** **multiprocessore**). Questo è detto **parallelismo fisico o reale**.

Interleaving

Quando (in macchine **monoprocessore**), dato che non è possibile il parallelismo fisico, i processi vengono eseguiti **alternati nel tempo** ma con velocità tali da dare l’impressione di avere multiprocessore. Ciò si presenta come un’alternanza casuale nell’esecuzione di processi in base alle politiche di **scheduling**. Questo è parallelismo **virtuale**.

Interleaving e overlapping possono essere **combinati** in presenza di + macchine multitasking.

Condizioni di Bernstein

1) rango(Ia) ∩ rango(Ib) = ∅

2) rango(Ia) ∩ dominio(Ib) = ∅

3) dominio(Ia) ∩ rango(Ib) = ∅

Dominio

È 1 o + aree di memoria di un’istruzione o procedura (variabili il cui valore è usato per operazioni e altro…)

Rango

È il contenuto 1 o + aree di memoria di un’istruzione o procedura (variabili il cui contenuto è modificato/sovrascritto)

Sezione critica

Sequenza di istruzioni tramite la quale un processo accede e modifica un insieme di variabili condivise.

Mutua esclusione

Modalità di accesso a risorse condivise per cui, ad ogni istante, al max 1 processo per volta può accedervi.

Secondo la sua regola, una risorsa o è libera o è assegnata a 1 solo processo.

Regole

- Nessuna coppia di processi può trovarsi **simultaneamente** nella sezione critica;

- L'accesso alla regione critica **non** deve essere **regolato** da alcuna **assunzione** **temporale** o dal **n°** di **CPU**;

- Nessun processo in esecuzione al di fuori della regione critica può **bloccare** un processo che prova a entrarvi;

- Nessun processo deve attendere **indefinitamente** per poter accedere alla regione critica.

*Busy* *waiting*

È una soluzione adottabile al fine di prevenire l’accesso a sezioni critiche (risorse già assegnate), che consiste nel far aspettare i processi che provano ad accedervi finché la risorsa non viene rilasciata.

*Deadlock* (o blocco multiplo/stallo)

Quando 2 o + processi rimangono in attesa di eventi che non accadranno mai a causa di condizioni cicliche nel possesso e richiesta di risorse.

(Quando un processo sta usando una risorsa e ci mette troppo tempo perché è molto lento, quindi quello che aspetta va in *deadlock*, ovvero attende lunghi periodi finché il processo che sta usando al risorsa non la rilascia).

*Starvation* (o blocco individuale)

Un processo “muore di fame”, ovvero, rimane in attesa di un evento che non accadrà mai, quindi non può completare la sua esecuzione, e dato che non può terminare, consuma risorse.

(Processi con priorità + alta continuano ad occupare una risorsa lasciando quello con – priorità in attesa infinita).

*Safety*

Condizione che deve sempre verificarsi x buona esecuzione di un processo o che non deve mai verificarsi per la sicurezza del sistema operativo.

Questa proprietà è garantita dai meccanismi di **sincronizzazione** e sancisce che i processi non devono **mai** **interferire** nell’accesso a risorse condivise in modo che le risorse si trovino sempre in stato **consistente**.

(Un programma rispetta la condizione di *safety* quando non permette a dei processi di accedere a una risorsa in uso)

*Liveness*

Sancisce la necessità che i processi vengano **portati a termine** (prima o poi) al fine di evitare il *deadlock*.

*Fairness* (o equità)

S.O. deve **sempre** mandare in esecuzione **ogni processo** (facendoli evolvere tutti + o – per lo stesso tempo) soddisfacendo (prima o poi) tutte le richieste di esecuzione così da **non causare** **starvation**.

Comunicazione tra processi

Comunicazione con struct Buffer condiviso: P0 scrive dati e P1 legge dati. Questi 2 non possono usare buffer contemporaneamente, quindi si usa un bool **StatoBuffer**. True: P0 sta scrivendo | False: risorsa libera.

**Problema**: **busy**-**waiting**, se P0 scrive, P1 consuma memoria continuando a testare il while.

**Semafori**

Sono variabili contenenti: un **contatore** int e una **coda** **associata**. Tramite essi i processi comunicano scambiandosi **segnali**. Solo 2 funzioni: **Wait**() --> riceve i segnali di un semaforo. **Signal**() --> invia segnale e contatore lo conta. Quando un semaforo non riceve segnali (perché non arrivano) il processo va in **wait** e spostato in coda associata.

Con contatore a 1 --> Mutua esclusione tra processi. Con contatore a 0 --> sincronizzazione tra processi.

Tecnica produttore-consumatore

Tecnica usata quando 1 o + processi (produttori) producono dati e li inseriscono in una coda/buffer condivisi, mentre altri processi (consumatori) accedono alla coda/buffer per prelevare i dati prodotti dai produttori.

Sincronizzazione: mutua esclusione --> solo 1 processo (non importa se produttore o consumatore) può leggere o scrivere per volta nella coda/buffer.

Tecnica lettore-scrittore

Tecnica usata quando si ha una risorsa condivisa (tipo database) che può essere letta da + processi (lettori) ma può essere scritta solo da 1 processo per volta (scrittore).

Sincronizzazione: mutex? --> lettori possono leggere come e quando vogliono, ma quando uno scrittore deve scrivere, solo lui ha accesso alla risorsa finché non finisce.

Istruzioni (primitiva) atomiche

Sono delle istruzioni che non possono essere interrotte durante la loro esecuzione, ed essendo **non** **interrompibili** vanno programmate come **sezioni** **critiche**. (Tipo Wait e Signal dei semafori)

WEB & HTTP

Server web

Il server web è un’**applicazione in esecuzione su un** pc **server** ch**e fornisce contenuti web** richiesti, è chiamato anche **Daemon HTTP** o **HTTPd**. Questo è costantemente in attesa di richieste da client, e se arriva una, cerca di evaderla il più velocemente possibile cercando e restituendo il file richiesto se trovato. In ambienti Unix/Linux il Daemon si può **duplicare** con la fork(), mandando il figlio a gestire una determinata richiesta e poi muore.

HTTP

(Le app web lavorano con il protocollo **HTTP**, che stabilisce le regole di comunicazione tra client e server. Questo trasmette i dati in formato testuale (ASCII) con i protocolli IP (liv 3) e TCP (liv 4), il cui compito è di instaurare e mantenere la comunicazione.)

La comunicazione inizia con una richiesta di connessione **TCP** con **porta** **80** (o **443** se **HTTPS**).

**URI** (Uniform Resource Identifier)

**Identifica univocamente una risorsa** tramite uno schema universale e una sintassi generica.

**URL** (Uniform Resource Locator)

(Sottoinsieme di URI) **identifica univocamente l’indirizzo web di una risorsa**, tipo pagina HTML. Metaforicamente una stringa URL risponde alle domande: “***Come****?*”, “***A chi****?*”, “***Che cosa****?*” così:

|  |  |
| --- | --- |
| 1. **Protocollo** usato per la richiesta (*Come?*) 2. **Host** a cui fare la richiesta (*A chi?*) 3. **File** richiesto (*Che cosa?*) | Forma:  [**protocollo**]://[**host**]/[**file**]  **http**://**www.gildown.it**/**interroghiamo.html** |

HTTP Requests

Fatte per **accedere a una risorsa** **identificata da un URI** **presente su un server** in Internet. Sono **stringhe ASCII** così:

|  |  |  |
| --- | --- | --- |
| [**metodo**] [**URI**] [**versione**]  [**NomiCampi**]:[**value**]  [**body**] | POST / HTTP/1.1  HOST: [www.gildown.it](http://www.gildown.it) …  … |  |

**Metodo**: GET, POST… + **Versione** **HTTP** + **Metainformazioni** con **valori** + **body**.

HTTP Responses

Anche queste consistono di stringhe di testo ASCII:

|  |  |  |
| --- | --- | --- |
| [**versione**] [**status**] [**desc**]  [**NomiCampi**]:[**value**]  [**body**] | HTTP/1.1 200 OK  HOST: [www.gildown.it](http://www.gildown.it) …  … |  |

**Versione HTTP** + **status code** + **descrizione status code** + **Metainformazioni** con **valori** + **body**.

Status Code

* + **200** – **299** = **Success**,
  + **300** – **399** = **Redirect**,
  + **400** – **499** = **Client Error**,
  + **500** – **599** = **Server Error**.

Passaggio di parametri

Client passa parametri al server nell’URL con il protocollo HTTP, in formato: [**URL**]?[**NomePar**]=[**ValorePar**]&…

GET

**Invia dati in formato ASCII aggiungendo i parametri all’URL** come suddetto. Si può fare digitando i parametri giusti nell’URL oppure compilando un form e facendo submit (così appaiono in automatico).

Col GET il server assegna i parametri dopo il “?” a una variabile chiamata **QUERY\_STRING**, poi il **GET** **non** **passa** **body** e la stringa è visibile e ha una **lunghezza max** di **2048 char**.

POST

I parametri sono passati col body della richiesta, quindi non visibili nell’URL e senza limiti.

Caratteristiche

* + **HTTP non ha memoria** (*stateless*), perché le **richieste sono uniche per il server** e **non salvano dati di navigazione** (risolto con **sessione** e **cookies**),
  + **HTTP è asimmetrico** (*pull-based*), ovvero **solo il client chiama il server,**
  + **HTTP è nato per trasmettere pagine web**,
  + **HTTP non è sicuro**, ma **HTTPS** usa **SSL** (Secure Socket Layer) che **crittografa** e **protegge** i **dati** tra client e server.

EVOLUZIONE DELLE RETI

Rete

**Insieme di sistemi** (da almeno 2 o + dispositivi) **in comunicazione tra loro e che si scambiano dati.**

Modello Mainframe-Terminal

Le prime erano costose, sviluppate in università e in aziende, e il 1° obiettivo era **condividere** **poche** **risorse** **tra** + **persone**. L'architettura prevedeva:

* + **mainframe** potente
  + semplici **terminali** (associati al mainframe e inutilizzabili se disconnessi da esso)

In seguito si è passati alle moderne reti di computer, fatte da tanti pc che erano:

* + **autonomi** (Non ci sono dipendenze tra i vari pc, quindi se 1 muore l'altro continua ad andare.)
  + **interconnessi** (Capaci di scambiarsi informazioni)

Rete di computer

**Insieme di 2 o + dispositivi in grado di trasmettere dati tra loro**

Problematiche

Elettroniche

Serve stabilire come verranno collegati fisicamente i sistemi (cavi/no cavi, fibra/rame/wi-fi...) in base a vari fattori (tipo n sistemi da connettere, distanza tra sistemi...)

Informatiche

Serve un OS in grado di sfruttare l'hardware

Telematiche

Serve tener conto che potrebbero già esserci reti di trasmissione di info sfruttabili sul territorio (tipo rete telefonica)

Tutto ciò va risolto rendendo **irrilevante/ininfluente la distanza tra i sistemi** da connettere.

Vantaggi

* + **Condivisione delle risorse**
  + **Miglior rapporto prestazioni/costo**
  + **Scalabilità** (investimenti mirati per migliorare prestazioni complessive nei sistemi usati)
  + **Maggior Fault Tolerance** (affidabilità rispetto ai guasti)

Dimensione e nomi

* **LAN** (Local Area Network) --> con cavi | **WLAN** (Wireless Local Area Network) --> con wifi

Piccole reti grande da ufficio a edificio per condividere dati tra utenti con:

- Alta velocità di trasmissione (essendo piccole)

- Sicurezza + easy

* **WAN** (Wide Area Network)

Dimensioni estese da città a pianeta

Internetworking

**Interconnessioni di reti diverse** (tipo LAN con WAN o 2 WLAN...)

RETI CLIENT-SERVER E P2P

Tipi di reti

In condivisione c'è sempre 1 pc che mette a disposizione roba e un altro che riceve.

Client/Server

Dei pc (**server**) che **mettono a disposizione delle risorse** o offrono servizi **ad** altri pc (**client**).

Se client non può diventare server e viceversa, il server è detto **server dedicato**.

In altri casi (come in **> parte delle reti locali**) **non vi è** una **distinzione** predefinita **tra client e server**, perché un pc può condividere dati e al contempo usare quelli di un altro pc.

*In reti grandi spesso pc + potente è il server, detto host (distribuisce servizi e risorse a rete) ma pc obsoleti possono fare da server*

P2P

Tutti i **pc** sono **sullo stesso livello** e condividono risorse comuni. Qui tutti i pc sono sia client sia server.

Messaggi

**Insieme di caratteri e dati che devono essere trasferiti da un sistema a un altro**.

*(Quindi un insieme di info organizzate in modo da formare un'entità trasmissibile tra 2 sistemi di rete)*

Pacchetti

Per favorire la trasmissione, i messaggi sono suddivisi in **pacchetti** (perché un mess piccolo è + semplice che trovi una linea che abbia spazio per lui e può lasciare spazio ad altri mess)

I pacchetti hanno anche info riguardanti: il **destinatario** e **l'ordine di ricomposizione** una volta giunti a destinazione.

Reti C/S (Client-Server)

Quando si stabilisce una comunicazione tra client e server, inizia l'erogazione del servizio richiesto, con 2 possibilità:

Esecuzione lato client o locale

**Il programma è trasmesso dal server, caricato in RAM ed eseguito dal client stesso**

1) *Client*: invia richiesta x prog

2) *Server*: elabora e invia il prog

3) *Client*: riceve e usa il prog per avere risultati

Esecuzione lato server o remota

**Il programma è eseguito sul server che trasmette i dati al client**

1) *Client*: invia richiesta per una elaborazione

2) *Server*: elabora richiesta, la esegue, risponde con il dato

3) *Client*: riceve il dato

Vantaggi esecuzione lato server

* **unica installazione** --> prog installato solo su server e non su tutti i client
* **aggiornamenti e manutenzione + easy** --> siccome è su 1 macchina, e lo stesso livello di update tra tutti i sistemi
* **diminuzione costi** --> di licenze
* **> sicurezza di dati** --> solo 1 backup da fare

Cloud

**Insieme di risorse, applicazioni e servizi disponibili e distribuiti su tutta la rete**. (vantaggi di prima sono alla base del cloud computing)

Reti P2P (+ usata in reti locali dove ogni pc condivide dati)

Non vi è distinzione tra client e server, bensì i pc di una rete sono detti **nodi** (ovvero, **ogni pc è sia client sia server**)

Un pc è client quando riceve dati e server quando li fornisce.

Esempio comune è il file-sharing tramite internet., fatto con programmi tipo eMule o bitTorrent, dove un pc si connette a un altro per scaricare dei file che lui mette a disposizione.

Vantaggi

* **semplice da gestire** (dato che non necessita amministrazione)

Svantaggi

* **meno sicura nel controllo di accessi e utenti** (dato che non c'è amministrazione)

Differenze

Client/Server

* **L'accesso** alle risorse è **gestito** **da** un **database** **di** **sicurezza** centralizzato
* **Whitelist** del server **con** **utenti** **e** rispettive **password**
* Gli **utenti** possono **accedere** **solo** a **determinate** **risorse** a cui sono autorizzati
* **Pochi** **problemi** **al** **ridimensionamento**

P2P

* Implementano una **rete overlay astratta** a **livello applicazione** sulla topologia fisica
* Idea di base è **condividere** **risorse** in **modo** **economico**
* Sologli **utenti** finali possono **controllare l'accesso** alle **risorse** (con password in punti di condivisione che creano)
* **< prestazioni al ridimensionamento**

LIVELLI DI RETE

Modello ISO/OSI (progettuale)

1) Fisico

Definisce funzioni della **connessione fisica**, da **elettronica** a **tecnica**. Gestisce **hardware**.

2) Datalink

Riguarda i **collegamenti dati da 1 pc a 1 altro della rete**. Controlla **sequenze di bit**, **recupero/ritrasmissione mess errati**.

3) Rete

Qui **suddivisi e riassemblati i pacchetti**. **Sceglie** una **strada** tra le disponibili tramite i router. Protocollo + usato: **IP**.

4) Trasporto

**Gestisce** **trasmissione** dei **pacchetti** (tra strati di sessioni di macchine diverse). **Identifica dest**, **apre e chiude connessioni** con pc corrispondente, **check errors** o **info flowing speed.**

5) Sessione

**Gestisce la sync dei dati da visualizzare**. **Instaura sessioni** (collegamenti logici e diretti tra 2 interlocutori) e ne organizza il dialogo.

6) Presentazione

**Decodifica in modo particolare le info** che vi arrivano **in modo da farle visualizzare** **nei** dispositivi di **output.**

7) Applicazione

**Gestisce** programmi **applicativi** + view di progs di login, file transfer, posta elettronica...

Per questa pila, i dati vanno **da alto verso** il **basso** nel **trasmettitore** e **da basso ad alto** nel **ricevente**.

**Scendendo** si **aggiungono** **dati** al mess (**incapsulamento**) e **salendo** il **contrario** (**deincapsulamento**)

Modello TCP/IP (pratico)

*(accesso (1 e 2) - rete - trasporto - applicazione (5, 6 e 7))*

Alla base dell'interconnessione delle reti, usa **IP** (**Internet Protocol**, liv 3) e **TCP** (**Transfer Control Protocol**, liv 4, ***connection-oriented***) ma anche **UDP** (**User Datagram Protocol**, liv 4, ***connectionless***).

Livello rete TCP/IP

**Instrada** i **pacchetti** **dalla sorgente alla destinazione lungo il percorso migliore** (diverso dal datalink, che muove info da un estremo del link all'altro). Percorsi diversi se ci sono reti interconnesse.

Compiti:

* conoscere topologia della rete
* scegliere percorso migliore
* gestire flusso dati e congestioni
* gestire problematiche riguardo a presenza di reti diverse

PROTOCOLLI

Protocollo

**Insiemi di regole e convenzioni usate nel dialogo tra livelli.**

TCP

Il TCP è un protocollo livello **trasporto** che fornisce supporto a altri protocolli liv applicativo (HTTP, FTP, SMTP, SSH…).

È importante perché **permette** di **operare** **contemporaneamente** **con applicativi distinti**. (Tipo utenti in internet possono usare + app internet, non solo TCP ma anche UDP)

Ciò avviene grazie alle **porte**, numeri interi di 16 bit (0-65535). (nelle trasmissioni viene dopo l'IP destinatario separato da ":", tipo: 192.168.0.1:80)

È **orientato alla connessione** (stabilire connessione prima di trasferimento) al contrario di **UDP**. (*Both use IP*)

Porte

**Identificano univocamente l'applicazione a cui instradare i pacchetti**, soprattutto in caso ci siano tante applicazioni che vanno riconosciute in rete. Divise in:

Well-Known (0 – 1023)

Riservate per servizi e app standard dei server, e settate di default sui client per essi. (client non dovrebbero usarle)

Registrate (1024 – 49151)

Usate per servizi privati, il client le usa per farci quello che vuole

Dinamiche (49152 – 65535)

Libere per essere assegnate dinamicamente da processi applicativi, usate per servizi passivi, tipo P2P.

Porte di default

TCP 80

Abbiamo 2 pc, 1 client e 1 server. Il server attende sempre per 1° una richiesta dal client, ma questo deve sapere il suo IP e la porta del servizio che deve usare. Il server non può usare una porta casuale per quel servizio, se no il client come lo raggiunge? Per questo si è definita la porta TCP 80 come predefinita per accedere a un qualsiasi server HTTP.

|  |  |  |  |
| --- | --- | --- | --- |
| TCP 25 | TCP 21 | UDP 53 | TCP 443 |
| Server di posta SMTP | Server FTP | Server DNS | HTTPs |

SOCKET

Perché?

Per lo scambio di messaggi tra processi su host diversi, serve un modo per i processi di identificare il destinatario, quindi le applicazioni in rete vanno riconosciute in qualche modo; ciò grazie al port address.

Port address

**N° che identifica** le **porte logiche** (su 2 byte, da 0 a 65535) e **individua** un **canale** da usare **per la comunicazione**.

I **n° di porta logica** sono **univocamente assegnati al relativo protocollo** (porta TCP != porta UDP, anche se a volte usata una unica per entrambi). La sola **porta** **non** **rende** una **connessione** **univoca**, perché processi di host diversi potrebbero ascoltare tutti sulla stessa. Per questo **la** **porta viene combinata con l’indirizzo IP** (socket).

**Socket**

È scritto così -->“**[IP]:[port]**”. La coppia è:

**IP** --> che identifica l’host avente il processo destinatario con cui comunicare.

**Port** --> identifica la porta usata dal processo dell’host destinatario.

Permette di comunicare in rete con la pila TCP/IP (mess esce da socket mittente e arriva a socket dest).

In un server TCP abbiamo 2 tipi di socket:

* 1 per accettare le connessioni (condiviso) che si chiama *connection socket*,
* 1 per funzioni send() e receive() (non condiviso) detto *data socket*.

Socket & client/server

In rete ci sono client e server. Il **server** (o meglio **processi server** che girano **su un host che erogano servizi ad altri processi** in rete) ha più controllo perché lui **crea i socket**. Più client possono comunicare sullo stesso socket.

Un processo **client deve conoscere il socket del server** (con eventuale richiesta), mentre il **server deve acquisire le info sull’host client**, come il suo **n° di porta** per inviare le risposte. Passaggi:

1. **Inizializzazione** dei **processi** (si **crea un socket**),
2. **Client** **effettua** la **richiesta di collegamento** (nel socket ci sono delle **code** con le **richieste** in logica **FIFO**),
3. Il server accetta la richiesta e stabilisce il collegamento (**se esaurisco la coda**, il **server** **rischia di crashare** per troppe richieste, quindi **controlla** il **socket** e **vede quante** **ne** **arrivano** **e da quali IP**. Se ne arrivano troppe da un IP magari può essere un DDoS e prendere contromisure).

Il server quando accetta la richiesta realizza un canale virtuale tra il client e un suo nuovo socket, al fine di lasciare libero il primo per ulteriori richieste?????

Processi si scambiano dati con funzioni **read()** e **write()** fino a **chiusura** del **canale** o a chiamata della primitiva **close()**.

Famiglie di socket

Ci sono 2 famiglie di socket (o domini):

* Unix Domain Socket (AF\_UNIX) --> pathname valido nel file system della macchina, permette trasferimento dati tra processi sulla stessa macchina UNIX;
* Internet socket (AF\_INET) --> usato nelle applicazioni per il trasferimento dati tra processi su macchine remote connesse tramite una LAN o internet; specificato da 2 valori:
  + **Indirizzo IP** (32 bit), che individua un unico host Internet,
  + **N° di porta** (16 bit), che specifica una porta dell’host.

Tipi di socket

Ci sono 3 tipi di socket:

* Stream Socket --> **Protocollo TCP/IP** (affidabile con connessione, quindi connessioni punto-punto unicast),
* Datagram Socket --> **Protocollo UDP** (non affidabile, usato anche per multicast),
* Raw Socket --> Usati per realizzare protocolli.

APPLICAZIONI E SISTEMI DISTRIBUITI

Sistema distribuito

Un **insieme di nodi interconnessi** in una rete che **si scambiano messaggi per** **eseguire algoritmi distribuiti**.

Sito = luogo fisico di una macchina.

Nodo = macchina o sistema in un sito (detto anche host).

Client = nodo che consuma risorse da un server.

Server = nodo che fornisce risorse ad altri nodi.

Sistemi distribuiti avanzati

Resource sharing

Condivisione di file in siti remoti, informazioni processate in un db distribuito e uso di dispositivi hw remoti.

Load sharing

Maggiore potenza di calcolo e velocità e load balancing (gestione e suddivisione del carico).

Vantaggi

Reliability --> **Identificazione** di siti in failure, **trasferimento** funzioni da un sito in failure e **ripristino** di esso.

Communication --> Scambio di messaggi.

Flexibility --> Rimpiazzo di mainframe con reti di host e < costi.

Applicazione distribuita

**Applicazione costituita da 2 o + processi eseguiti in parallelo su pc diversi connessi in una rete**.

I processi di una app distribuita cooperano sfruttando i servizi forniti dalla rete e si devono mettere in contatto.

Layer applicativi

1) Livello presentazione o Front End

Gestisce la logica di presentazione, ovvero le modalità di interazione con l'utente (interfacciamento grafico e rendering delle info).

2) Livello di logica applicativa

Gestisce le funzioni da mettere a disposizione all'utente.

3) Livello di accesso ai dati

Gestisce le informazioni ed eventualmente l'accesso ai db.

Questi (livelli software) possono essere installati su vari livelli hardware, detti **Tier**.

Tiers

App può essere configurata come:

Single Tiered

Tutti i **livelli ospitati** **su** **1 host**

Two Tiered

**Livelli divisi tra 1 client** (che ha il livello **Presentazione**) e **1 server** (che ha il **livello** di **accesso** ai **dati**). Il livello di **logica** **applicativa** può stare **o sul client o sul server**.

Three Tiered

**3 macchine dedicate 1 per livello** (client x presentazione e 2 server per altri 2)

Architettura multi-tier = usati 2 o 3 tier come arch di una app (ES: **browser** [*Presentazione*] che fa richieste a un **server** [*Logica applicativa*] che fa richieste a un **DBMS Server** [*Logica accesso ai dati*]) i livelli adiacenti assumono i ruoli di **client e server** (il **+ vicino a utente è sempre client**, mentre il **più lontano sempre server**)

Clients

Thin Client --> quando a **livello utente** c'è **solo** livello **presentazione**

Fat Client --> quando a **livello utente la logica applicativa** **si appoggia su quella di accesso ai dati**

Ends

Backend --> Livello **middleware** che contiene la **combo di logica di accesso ai dati e logica applicativa**

Frontend --> Livello **presentazione**

Sistemi

Centralizzato

Quando **dati e applicazioni sono su un unico nodo elaborativo**. (Tipo architettura a mainframe?)

Distribuito

**Insiemi di applicazioni logicamente indipendenti che cooperano per raggiungere obiettivi comuni tramite** **infrastruttura** di comunicazione **HW** e **SW**

Esso realizza almeno 1 delle seguenti situazioni:

1) Elaborazione distribuita --> **applicazioni cooperanti** distribuite su + nodi elaborativi

2) Base di dati distribuita --> **database** ospitato su + nodi elaborativi

Server Farm

**Insieme** (molto grandi) **di pc interconnessi che condividono app e dati**.

Realizzate secondo 2 principi progettuali:

Cloning

**Su ogni nodo della farm sono installate le stesse app (duplicandole).**

Le richieste sono poi inviate ai vari client tramite un sistema di **load-balancing** (tecnica che distribuisce il carico di elaborazione tra diversi server).

Per cui: **> scalabilità e affidabilità** di arch nel complesso. I sistemi di load-balancing integrano anche **sistemi di** **monitoraggio** che **escludono** automaticamente **cluster non raggiungibili**

RACS (Reliable Array of Cloned Services)

**Insieme di cloni dedicati a svolgere un certo servizio**, in cui **se 1 si guasta**, **un altro** continua a **erogare** il **servizio**.

Presenti in 2 configurazioni:

Shared Nothing

**Dati memorizzati sono duplicati e stanno sul disco di ogni clone.**

**Ottime** **prestazioni** per **app read-only.**

Shared Disk (cluster)

I **cloni condividono un server che gestisce i dischi fissi**.

Partitioning

**Prevede duplicazione di HW o SW**, **ma non dei dati**, **ripartiti tra nodi**.

**Ogni nodo svolge una funzione** **specifica** (tipo ogni nodo assegnato un endpoint di un sito)

**Dati** però **sono su 1 singolo server**, quindi **se esplode**, c'è il **degrado parziale** dei sistemi distribuiti (non tutto il sistema, **ma solo alcune funzionalità**)

Per risolvere questo si usano i:

RAPS (Reliable Array of Partitioned Services)

**Permette di ottenere piena scalabilità e disponibilità del servizio**.

DATACENTER

Datacenter

Un **Datacenter** è lo **spazio fisico che ospita le infrastrutture tecnologiche** usate **per** **elaborazione**, **memorizzazione** e **condivisione** delle **informazioni**.

Dal punto di vista **fisico** un datacenter da: **spazio** e alloggiamenti **per** i **dispositivi**, **energia**, sistemi di **raffreddamento** e **sicurezza fisica** oltre a interconnessioni con operatori per **accesso a internet** e **sicurezza logica**.

Da quello **logico** invece, un datacenter da: servizi come **hosting**, **cloud** privato e/o pubblico, **applicazioni web**, **posta** **elettronica**, gestione **domini** e **altre attività** non necessariamente tipiche di un datacenter.

Tipi

Aziendali (CED) --> **Proprietà** **di un’azienda** e **forniscono** **servizi relativi alla struttura interna**.

Offerti da terzi (hosting, housing, cloud) --> Strutture per **gestione** ederogazionedi **servizi** accessibili **in internet**.

Componenti

Infrastruttura operativa (*white space*)

Ospita i **rack** con i **server**, i sistemi di **storage**, gli **apparati di monitoraggio** e gli **apparati** **per** **trasmettere** dati in rete.

Infrastrutture di supporto

Dispositivi di **backup**, sistemi **ridondanti**, **gruppi di continuità**, impianti di **climatizzazione**, sistemi di **raffreddamento**, impianti **antincendio** e sistema di **distribuzione** dell’**energia**.

Personale operativo

Le figure professionali che gestiscono il centro: **system** **admin**, **tecnici**, **sistemisti**, **programmatori** e **security**.

Obiettivi

* **Mantenere la disponibilità dei dati garantendo continuità di elaborazione**. Quindi serve **monitorare il rischio** e adottare **misure di protezione**, tenere da conto **l’errore umano** e impostare logiche di **controllo degli accessi** oltre a **garantire la protezione** da minacce **dalla** **rete** (tipo malware).
* **Proteggere le informazioni** nello storage e nei sistemi considerando eventuali **attacchi** di **cyber-criminalità** e adottando **misure** internazionali **per** **la** **privacy**, per questo si usano **firewall** e **sistemi anti-intrusione** (IDS).

Evoluzione

1. **Portare all’esterno dell’azienda i server e lo storage** (*collocation*) rivolgendosi a terzi ed eliminando il CED;
2. Creare **datacenter virtuali** facilmente manipolabili e scalabili **con** la **virtualizzazione** per **ridurre costi e spazi**;
3. **Affidarsi a un fornitore di servizi cloud** (*serverless approach*) che **gestisca** da sé le **problematiche fisiche**.

Rischi

Portando tutti i servizi su un’unica struttura determina un **punto di criticità** detto **SPoF** (**Single Point of Failure**) perché un **guasto** **compromette** **l’intero** **sistema**; e quindi **> il *downtime***, **>** la **ripercussione economica**.

Serve garantire ***High* *Availability*** (alta disponibilità di servizio), con ridondanze, duplicazioni, cluster e repliche datacenter.

Garanzie

***Fault Tolerance***

**Tolleranza ai guasti**, ovvero **evitare l’interruzione dei servizi a fronte di essi**, fatto con diverse tecniche come: **RAID** e la **duplicazione di server** e **apparati di rete**. È complementare insieme alla *disaster recovery*.

***Disaster Recovery***

**Recupero e ripristino di applicazioni e dati in caso di disastro** (di qualsiasi tipo, dal naturale all’errore umano) fatto spesso con **duplicazione** di interi **datacenter**. Determinante è il **tempo di ripristino**.

***Business Continuity***

**Continuità del servizio** che garantisce continuità operativa **anche durante disastri o guasti**.

Tiers

**Tier 1** – Basic Site Infrastructure

**Operatività** = **99.67%** | **Fermo macchina** = **28,8h/anno**

Data center con **gruppo di continuità** con funzioni **minime**, un **unico percorso di distribuzione** e **ridondanze nulle** o **limitate**. Protezione limitata contro eventi fisici.

**Tier 2** – Reduntant Capacity Site Infrastructure

**Operatività** = **99.75%** | **Fermo macchina** = **22h/anno**

Ha **maggiori** capacità **ridondanti**, **migliori** **gruppi** **di** **continuità**, **sistemi** di **raffreddamento** e un **unico percorso di distribuzione non ridondante**. Protezione da eventi fisici migliorata.

**Tier 3** – Concurrently Maintainable Site Infrastructure

**Operatività** = **99.98%** | **Fermo macchina** = **1,6h/anno**

**Aggiunge** componenti di **ridondanza** e **percorsi** di **distribuzione** **indipendenti** **multipli**. Il sito è **contemporaneamente** **mantenibile**, quindi se sospendo qualche componente di capacità, il sito dà ancora servizi mentre lo mantengo. Ha protezione contro la maggior parte degli eventi fisici.

**Tier 4** – Fault Tolerant Site Infrastructure

**Operatività** = **99.99%** | **Fermo macchina** = **0,8h/anno**

Alla manutenzione simultanea aggiunge **l’elaborazione continua** in caso di guasto di componenti (elaborazione influenzata solo se si guastano più componenti identici).

VIRTUALIZZAZIONE

Che cos’è?

È una **tecnica per nascondere e astrarre le caratteristiche fisiche di una risorsa** (pc, server, rete…), risultando in **singoli hw fisici** che **condividono risorse tra** i propri **guest**.

Scopo

Lo scopo della virtualizzazione è quello di **eseguire in contemporanea + istanze di un OS guest in 1 unico host fisico**.

I “**guest”** interagiscono con le risorse dell’host fisico tramite un sw intermedio: l’**hypervisor** o **VMM** (virtual machine monitor).

Vantaggi (per persone)

* Uno **sviluppatore** può eseguire un’app in diversi ambienti senza necessità di + pc fisici,
* Un **system** **admin** può testare uno scenario complesso con + servizi su + host diversi, fattibile con + VM diverse.
* Gli **utenti finali** traggono maggiori benefici:
  + **Aumento dell’affidabilità del sistema**, perché possibile dedicare VM a servizi che non vanno in conflitto e isolare i vari guest cosicché eventuali problemi di una VM non influenzino la stabilità di altre.
  + **Consolidamento dei server**, con la virtualizzazione si eseguono + VM sulla stessa macchina, riducendo il n° dei server necessari per l’erogazione di servizi aziendali di 10 volte o +.
  + **Riduzione dei costi**, meno server significa meno costi energetici, di acquisto e di manutenzione dei server.
  + **Disaster recovery**, un OS guest è facilmente ripristinabile riducendo i tempi di indisponibilità per guasto.
  + **Alta disponibilità**, se ci sono + server fisici con hw compatibili e questi condividono uno storage su cui risiedono delle VM, in caso di failure è possibile spostare l’esecuzione delle VM di un host su un altro.

Vantaggi (per Datacenter)

* Riduzione numero di server da gestire,
* Riduzione spazio fisico necessario,
* Riduzione costi energetici,
* Riduzione costi di manutenzione.

Benefici

* **Esecuzione di app legacy**, virtualizzando OS si possono continuare a usare sw aziendali fatti per sistemi obsoleti.
* **Sviluppo e testing**, è possibile predisporre ambienti di sviluppo/testing senza intaccare l’ambiente principale.

**Riassumendo: - costi, gestione infrastruttura IT + semplice, > affidabilità e > flessibilità.**

Cosa si virtualizza?

Si riferisce in generale all’astrazione delle **risorse di calcolo** (*computing resources*):

Platform virtualization

(Concetto di **VM**) ovvero virtualizzazione di piattaforme hw, poi esteso a storages, namespaces e network resources.

* **Emulation/Simulation**
* **Native/Full virtualization**
* Hardware Enabled Virtualization
* Partial virtualization
* **Paravirtualization**
* **OS-level virtualization**
* Application virtualization

Resource virtualization

(Concetto di **qualità del servizio**) ovvero virtualizzazione di risorse.

* RAID
* SAN
* Channel Bondings
* VPN/NAT
* Multiprocesssor e multi-core
* Cluster e grid computing
* Partitioning

**Hypervisor** (VMM)

È un programma **che crea ed esegue VM** **guest** **su** **una** **macchina** **fisica** **host,** che **astrae l’hw di un pc** e che:

* **Crea e** **controlla** **molti** ambienti di esecuzione (**VM**) diversi e:
  + **Ciascuno** di questi può avere un **proprio** e diverso **OS**
  + **Ciascuno** di questi **crede** **di controllare** **l’intero** sistema **hw**.

Principali hypervisors: VirtualBox, KVM, QEMU, Parallel, Xen, VMWare, Xbox 360 (*gildeldown?*)

Virtual Machine (VM)

**Ambiente di esecuzione creato dall’hypervisor**.

Tipi di hypervisor

Type-0

Soluzione basata sull’hardware (vecchia, sui mainframe).

Type-1 (*Bare-Metal* o *Native*)

Il sw di virtualizzazione è **OS-like**, ovvero si **integra direttamente con il kernel del OS** e si **interfaccia con l’hardware** **direttamente**, consentendo gestione di risorse + efficiente, veloce e stabile per i guest. Tipica nella **virtualizzazione di server** (consolidamento di + server in 1) e per la **distribuzione di OS desktop remoti** per utenti in rete.

Type-2 (*Hosted*)

**L’hypervisor è** una vera e propria **applicazione** che si **gira sull’OS fisico** e **virtualizza** degli **ambienti** **compatibili** con esso a più alto livello, **facendo da tramite** per le operazioni tra guest e host.

**Svantaggio**: essendo un’applicazione, **non ha la stessa efficienza del Type-1**.

**Vantaggio**: **installazione del sw e creazione dei guest senza difficoltà**, spesso **gratis**.

VirtualBox

La **VMM è un processo** che gira nell’host (anche app utente) e il suo **OS** **non sa che si sta virtualizzando** un guest.

**Principali metodologie**

4 sono i principali metodi di virtualizzazione e danno l’illusione di star utilizzando un OS proprio non virtualizzato:

Emulation (emula tutto hw + OS)

L’**hypervisor** **emula l’hw** in modo **standard** **per eseguire l’OS guest** senza modifiche. Il sw di virtualizzazione presenta al OS guest un’architettura hw completa e nota indipendentemente da quella della macchina host.

Limiti

**L’architettura hw** presentata è **standard**, magari non include funzionalità già implementate in hw dell’host.

**È necessario interfacciare** la **CPU**, la **memoria** e l’**I/O** tra sistema host e guest.

Emulazione è difficile perché bisogna emulare sistemi guest con processori di velocità = al processore dell’host. Quindi emulazione fattibile quando ho **+ processori**.

Full Virtualization (virtualizza hw ma con OS compatibile, quindi interfacciato direttamente sul fisico)

Simile a emulation, gli **OS guest** devono essere **compatibili con l’architettura hw dell’host**, quindi **non necessario interfacciare** **CPU**, **memoria** e **I/O** perché **tutto** fatto sull’hw **direttamente**, con > prestazioni.

Di recente Intel e AMD hanno introdotto **VT-x** e **AMD-v**, e la full è fatta tipo da **VirtualBox** e VMware.

Paravirtualization (virtualizza OS e mostra hw modificato a essi, ma lo mostra limitato, quindi + VM per > memoria)

L’hypervisor mostra un’**hw sottostante modificato agli** OS **guest**, ma mantenendone l’architettura. Gli **OS delle VM** **sono modificati** (differenza con la full) (evitano certe system call, per cui prestazioni inferiori di molto poco all’OS non virtualizzato, dato che istruzioni eseguite sulla CPU senza modifiche). (virtualizza - hw fisico, quindi + guest e + VM perché ho + memoria).

OS level virtualization (virtualizza copie dell’OS dell’host sullo stesso hw)

**Non usa hypervisor**, ma si **virtualizza con copie dell’OS dell’host**. I **guest saranno istanze dell’OS dell’host** con un proprio file system, configurazione di rete e applicazioni. [**Vantaggio**: miglior utilizzo di risorse per la condivisione di spazi di memoria siccome guest non necessitano di un kernel privato, ma usano lo stesso dell’host con < utilizzo di memoria fisica]. Non adatto a OS **diversi** sullo stesso host, poca stabilità e isolamento.

Supporto hardware alla virtualizzazione

Aziende produttrici di hw (come Intel e AMD) hanno iniziato a distribuire CPU con supporto di virtualizzazione per diminuire intervento Hypervisor.

Intel

Intel ha implementato (in CPU Xeon e Itanium) le tecnologie **VT-d**, **VT-x** e **VT-i**, che **limitano o eliminano l’intervento** **dell’hypervisor**, eseguendo direttamente e automaticamente operazioni e context switches tra guest e CPU host.

AMD

**AMD-V** è un **insieme di estensioni hw** dell’architettura **x86** **per ridurre o eliminare**, **l’intervento dell’hypervisor**.

La **Direct Connect Architecture** aumenta le prestazioni eliminando il **bottleneck** (ritardi del sistema a causa di parti che rallentano il tutto) del front side bus.

Accesso alla memoria ridotto da un controller nelle CPU che connette i core a un’area di memoria, e da questo esce un bus HyperTransport Link (PCI/PCI-X) che collega le CPU che evitano l’arbitraggio.

Componenti di virtualizzazione

CPU scheduling

Sono possibili **+ CPU virtuali** **che CPU fisiche** (***overcommitment***) e la **VMM** deve **condivide**re **risorse fisiche** **tra** i **guest**. Overcommitment (CPU e memoria)

Allocazione di + CPU o memoria virtuale di quella fisica per poter avere + guest. (rischi di stabilità di sistema) Inoltre c’è:

Thin provisioning (storage)

Allocazione di storage flessibile e ottimizzato dando l’impressione che guest ne abbiano disponibile + di quello fisico.

Memory management

Il **VMM overcommitta memoria tra i guest** e **decide quanta ognuno ne può usare** (anche se il guest pensa che la stia usando tutta) oltre a **fare da sé l’allocazione di pagine di memoria**.

I/O management

Il **VMM dedica dispositivi I/O ai guest** e può fornirgli **device drivers astratti** (mappando le richieste a quelli veri).

Storage management

Il VMM assicura che **ogni guest può accedere** **solo** alla parte di **storage assegnatagli**.

Networking

VMM fornisce: **a ogni guest** **almeno 1 IP**, **routing tra** il **guest** **e** la **rete** e il **NAT**.

RAID

Gestendo la memoria vanno gestite le **ridondanze**, quindi considerare possibilità di guasti o situazioni che mettano in pericolo l’integrità dei dati. Ciò è fatto con il **RAID** (*Reduntant Array of Independent Disks*) che **raggruppa gli HDD in un grande disco logico** (**volume**) suddiviso in **sottovolumi** dedicati ai server. Tipi di RAID più usati:

RAID 1 o *mirroring*

Si configurano **2 dischi con 1 la copia dell’altro**; se 1 si rompe l’altro continua e quando sostituito si ricostruiscono i dati in esso con quelli dell’altro. **Ottime prestazioni** ma consente di sfruttare solo il **50% dello spazio disponibile**.

RAID 5

Si usano **3 o + HDD** e i **dati sono scritti vettorialmente su 2 di essi**, mentre **nel 3° vengono scritti i bit di parità** per ciò che è stato scritto nei primi 2 così che in caso di guasto di 1 disco, le info possono essere **ripristinate** **ricalcolando** **i valori**. Questo presenta un **< spreco di risorse** ma ogni **operazione** **di** **scrittura** **richiede** il **calcolo** dei **bit di parità**.

Virtual Networking

Una rete virtuale è un **insieme di VM in esecuzione su una singola macchina fisica e collegate logicamente tra loro** **così** **che** **possano scambiarsi dati**. Le VM forniscono le funzionalità di **vNIC**, **virtual switches**, **virtual routers**…

Un virtual switch rileva le VM logicamente collegate alle sue porte virtuali e indirizza il traffico; ciò consente di gestire la connettività a livello datalink, con possibilità di impostare **VLAN** multiple secondo lo standard 802.1q.

LACP

Per avere la max disponibilità dei collegamenti, lo switch virtuale può configurare + NIC fisiche associandole a esso in modalità **LACP** (*Link Aggregation Control Protocol*), che rende visibile allo switch le NIC fisiche come un'unica scheda logica e tra queste ripartisce il traffico.

Uplink port

Gli switch virtuali devono potersi connettere alle reti fisiche per comunicare con l’esterno, per cui sono usati alcuni adattatori software (**uplink port** o **pNIC**) associati a quelli fisici (che forniscono connessione tra rete fisica e virtuale).

VM driver

L’OS si interfaccia con i dispositivi sull’hw virtuale installando i relativi driver software.

La virtualizzazione ha permesso di affrontare 2 problemi:

* Consentire di ottenere partizioni sui propri server
* Eseguire applicazioni esistenti su più tipologie e versioni di O.S.

L’ampia applicabilità della virtualizzazione ha contribuito a ridurre il vendor lock-in.

Permette il cloud computing

La VM funziona come un unico file di dati, ed è utilizzabile anche quando spostato da un pc a un altro e usabile anche da + pc.

Quando l’ambiente virtuale fa richiesta all’ambiente fisico, l’hypervisor riporta la richiesta sul fisico, e salva le modifiche nella cache, che vengono fatte in tempo pressoché reale.

Tipi di Virtualizzazione

* Dei dati,
* Dei desktop,
* Dei server,
* Di funzioni di rete,
* .

Virtualizzazione dei dati

I dati distribuiti su più ambienti posson essere consolidati su una sorgente e virtualizzati e clonati per essere disponibili a più dinamicamente in base alla richiesta

Virtualizzazione dei desktop

Diverso dalla virtualizzazione del Os. Consente di distribuire + OS in un’unica macchina, consentendo a un admin centrale di distribuire desktops a diverse macchine fisiche.

Virtualizzazione dei server

Sono pc fatti per elaborazioni ad alto volume. Permette di affiancare ambienti Linux e Windows (virtualizzazione di OS)

Obiettivi:

* Ridurre il costo complessivo dell’hw, siccome ho macchine virtuali
* Aumenta la sicurezza,
* Limitare il tempo impiegato per servizi IT (aggiornamenti…)

Virtualizzazione di funzioni di Rete (NFV)

Separa le funzioni chiave di rete per distribuirle tra gli ambienti. È possibile riunire funzioni specifiche in una nuova rete e assegnarle a un ambiente. (Es, c’è la rete fisica, ma all’interno virtualizzo una rete mia in modo da suddividere carico e controllare accessi meglio). Questa è la base della cittadella in crittografia.