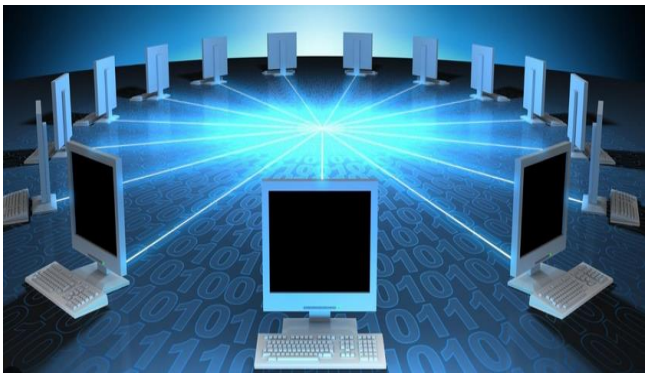
Appunti Sistemi Distribuiti

Domande e Risposte

Capitolo 1 – Introduzione ai Sistemi Distribuiti (definizione, caratteristiche e problematiche)

1) Cosa si intende per sistema distribuito?

Un sistema si dice distribuito se tale sistema è costituito da un insieme di applicazioni logicamente indipendenti che collaborano per raggiungere obiettivi comuni attraverso un'infrastruttura di comunicazione hardware e software. In altri termini è un insieme di calcolatori indipendenti che appaiono all'utente come un singolo calcolatore.



2) Quali sono le caratteristiche dei sistemi distribuiti?

I sistemi distribuiti hanno le seguenti caratteristiche:

→ mancanza di memoria condivisa e la comunicazione tra processi avviene mediante scambio di messaggi (message passing);

→ autonomia tra i componenti;

→ esecuzione concorrente;

→ mancanza di clock globale e il coordinamento avviene mediante scambio di messaggi;

→ indipendenza dei fallimenti.

3) Cosa si intende per trasparenza referenziale?

La trasparenza referenziale è la proprietà principale dei sistemi distribuiti che consiste nel considerare i fallimenti, i ritardi e le locazioni trasparenti in modo da mostrare all'utente l'unicità del sistema.

4) Che ruolo possiede l'architettura software?

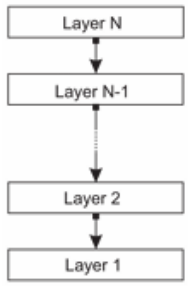
L'architettura software ha il ruolo di definire la struttura del sistema, le interfacce tra i componenti e i pattern di interazione. I sistemi distribuiti possono essere organizzati secondo diversi stili architetturali: a strati

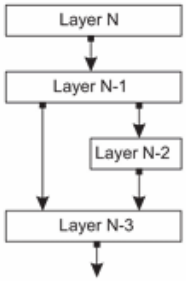
(layered), a livelli (tier), basato sugli oggetti, centrate sui dati o basati su eventi.

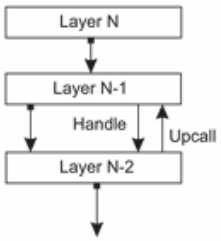
5) Cosa si intende per architettura stratificata?

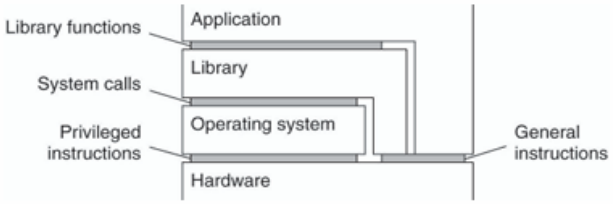
Si dice architettura stratificata un software organizzato in strati, ovvero costruiti uno sopra l'altro, e ogni strato è un set di sottosistemi. I livelli più alti rappresentano applicazioni più specifiche, mentre quelli più in basso sono più generali.

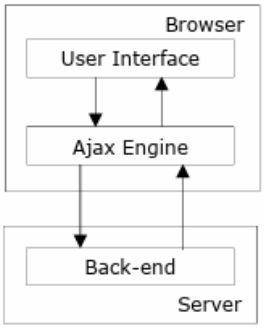
Esempio: Sistemi Operativi, Middleware e Protocollo ISO/OSI per reti di computer.











6) Cosa si intendono per sistemi operativi distribuiti?

Un sistema operativo viene definito distribuito se tale sistema è in grado di comunicare con altri sistemi indipendenti mediante l'infrastruttura di rete per raggiungere un obiettivo comune. Esistono tre tipologie di sistemi distribuiti con obiettivi diversi:

→ Network Operating System (NOS): governato da applicazioni, ovvero l'applicazione decide dove può essere eseguita in maniera autonoma;

→ Distribuited Operating System (DOS): trasparente alle applicazioni, ovvero il sistema decide dove eseguire l'applicazione in maniera autonoma;

→ Middleware: layer aggiuntivo che aggiunge trasparenza.

7) Cosa risulta necessario considerare riguardo la migrazione?

Essendoci diverse macchine che collaborano, la collaborazione deve essere esplicitata anche nella migrazione, che può avvenire in tre modi diversi:

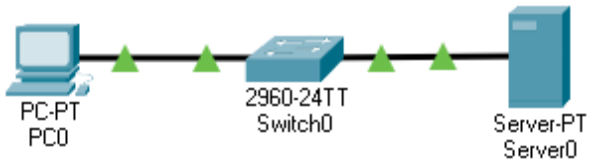
→ dei dati di un file o di porzioni di file;
→ della computazione, ossia trasferire la computazione piuttosto che i dati;
→ del processo, ossia eseguire un processo o parti di un processo in diversi siti.

8) Quali sono i servizi offerti dal middleware?

Il middleware è un software che si colloca tra le applicazioni e le componenti sottostanti che permette l'interazione di software distribuito. Implementa servizi per renderli trasparenti alle applicazioni e ci sono alcuni servizi base come quelli:
→ naming;
→ trasparenza di accesso;
→ persistenza, ovvero uno storage;
→ transazioni distribuite, con relativa consistenza in lettura e scrittura;
→ sicurezza con modelli per proteggere dati e servizi.

9) Modello Client – Server: definizione e caratteristiche

Il modello client – server è uno dei modelli di sistemi distribuiti in cui il client invia una request al server e quest'ultimo elabora la richiesta e fornisce una response al client quando la risposta è pronta. L'accesso al server può avvenire direttamente, a server multipli o mediante proxy server.



10) Quali sono i problemi fondamentali di un sistema distribuito?

Solitamente un sistema distribuito deve interfacciarsi con quattro problemi principali:
→ naming: identificare la controparte della comunicazione;
→ access point: come raggiungere un processo o una risorsa remota;
→ protocol: come vengono scambiati i messaggi tra i partecipanti alla comunicazione;
→ content of a message: come comprendere il contenuto di un messaggio, concordando la sintassi e la semantica.

11) Quali sono le tipologie di trasparenza?

Esistono otto tipologie di trasparenza:
→ naming: i nomi simbolici sono usati per identificare risorse che non sono parte dei sistemi distribuiti;

→ access transparency: nasconde le differenze nella rappresentazione dei dati e su come delle risorse remote o locali sono accessibili;
→ location transparency: nasconde dove una risorsa è localizzata nella rete;
→ mobility transparency: detta anche relocation, nascondere lo spostamento di una risorsa in altre locazioni mentre questa è in uso;
→ migration transparency: nasconde lo spostamento di una risorsa, intesa come migrazione;
→ replication transparency: nasconde la replicazione di una risorsa;
→ concurrency transparency: nascondere la condivisione di una risorsa tra più utenti indipendenti;
→ failure transparency: nasconde un fallimento e il recovery di una risorsa;
→ persistance transparency: nasconde se una risorsa è volatile o permanente.

12) Discutere la separazione tra politica e meccanismo dei sistemi distribuiti

Un sistema distribuito dovrebbe essere composto da componenti indipendenti logicamente, ovvero ogni componente dovrebbe essere in grado di fornire autonomamente un servizio o eseguire un'operazione. Ogni componente collabora con altri componenti per fornire servizi più complicati ed eseguire operazioni complesse. Questi obiettivi possono essere raggiunti separando le politiche dai meccanismi. Il meccanismo è rappresentato dai servizi offerti dai componenti mentre la politica ne definisce il comportamento. Maggiore è il grado di separazione tra politica e meccanismo, più risulta necessario fornire meccanismi più complessi, portando ad una gestione più complessa del sistema. Bisogna quindi trovare un giusto bilanciamento.

13) Cosa si intende per protocollo?

Si dice protocollo un insieme di regole che definisce il formato, l'ordine di invio e la ricezione dei messaggi tra dispositivi, il tipo dei dati e le azioni da eseguire quando si riceve un messaggio.
Esempio: le applicazioni su TCP/IP si scambiano stream di byte di lunghezza infinita (il meccanismo) che possono essere segmentati in messaggi (la politica) definiti da un protocollo condiviso.

Capitolo 2 – Stream Oriented Comunication

14) Quali sono le caratteristiche dei servizi TCP e UDP?

Il servizio TCP ha le seguenti caratteristiche:
→ orientato alla connessione;
→ trasporto affidabile;

→ controllo di flusso e congestione;
→ non offre garanzia di banda e ritardi minimi;
→ ogni messaggio viene suddiviso in segmenti e numerato per garantire il riordinamento dei dati e controllare duplicazioni e perdite;
→ protocollo favorito per i sistemi distribuiti.
Il servizio UDP è non affidabile tra processo mittente e destinatario dato che:
→ non offre connessioni;
→ non affidabile;
→ non viene effettuato il controllo di flusso;
→ non viene effettuato il controllo di congestione;
→ possibili ritardi e perdita di banda.
Tale protocollo può essere utilizzato per applicazioni che tollerano perdite parziali a vantaggio delle prestazioni. Ogni messaggio (flusso di byte) viene scomposto in segmenti e li invia uno per volta.

15) Quali i problemi fondamentali per entrambi i protocolli?

In entrambi i servizi (TCP e UDP), ci sono problemi fondamentali che devono essere affrontati per definire una corretta comunicazione. Tali problemi sono:
→ gestione del ciclo di vita di client e server: risulta necessario determinare la modalità di avvio e terminazione del client e del server;
→ identificazione e accesso: il client deve avere determinate informazioni per poter riconoscere e accedere ad un server e l'indirizzo del server può essere:
a) direttamente fornito nel codice;
b) fornito dall'utente;
c) richiesto con un server DNS;
d) attraverso protocolli di identificazione;
→ ripartizione dei compiti: spesso tra client e server, purché abbiano ruoli separati, è necessario definire le task del client e del server.

16) Cosa si intende per socket?

Si dice socket un'astrazione software progettata per utilizzare delle API standard e condivise per la trasmissione e la ricezione di dati attraverso una rete oppure come meccanismo di IPC. Sono particolari canali per la comunicazione tra processi che non condividono memoria, per esempio perché risiedono su macchine diverse. Per poter permettere la connessione o l'invio dei dati al processo destinatario, il mittente deve conoscere il nodo destinatario (indirizzo di rete) e la porta a cui connettersi. La comunicazione via socket

avviene mediante flussi di byte tramite tradizionali chiamate di sistema `read()` e `write()`. Entrambe le call utilizzano buffer per garantire flessibilità.

17) Quali sono le chiamate di sistema che permettono la gestione della comunicazione mediante socket (oltre `read()` e `write()`)?

Tali chiamate di sistema sono:

→ `Socket`: crea un nuovo endpoint di comunicazione;
→ `Bind`: blinda un indirizzo locale a un socket;
→ `Listen`: annuncia che il socket è pronto alla connessione;
→ `Accept`: blocca fintantoché una connessione non viene accettata;
→ `Connect`: tentativo di stabilire una connessione;
→ `Close`: rilascia la connessione.

18) Come avviene la comunicazione tra client e server mediante socket?

Il server crea una socket collegata alla well-known port, che identifica il servizio fornito, dedicata a ricevere richieste di connessione. Con la primitiva `accept()`, il server crea una nuova socket, cioè un nuovo canale, dedicato alla comunicazione con il client. Un altro processo elabora le richieste del client che si è connesso, mentre il processo principale continua ad accettare nuove richieste.

19) Descrivere le caratteristiche dei metodi `read()` e `write()`

Dato che la comunicazione via socket avviene mediante flussi di byte, o stream, in questo strato non vi è il concetto di messaggio e quindi il processo di lettura e scrittura devono avvenire mediante un numero arbitrario di byte. Si devono, quindi, prevedere cicli di lettura che termineranno in base alla dimensione dei messaggi come stabilito dal formato del protocollo applicativo in uso.

20) Descrivere l'utilizzo delle socket in Java: descrizione e caratteristiche

Java definisce alcune classi che costituiscono un'interfaccia ad oggetti alle system call. I principali packages sono `java.net.ServerSocket` e `java.net.Socket`. Tali funzionalità sono:

1 – Costruttori

→ `public Socket(String host, int port)`: crea un socket connesso alla specifica porta, in un determinato host;
→ `public ServerSocket(int port)`: crea un server socket connesso alla specifica porta. La coda massima consentita è di default di 50 richieste di connessione.

2 – Gestione connessioni

→ `public void bind(SocketAddress bindport)`: blinda un socket ad un indirizzo locale;
→ `public void connect(SocketAddress endpoint, int timeout)`: connette il socket al server, con uno specifico valore di timeout;
→ `public void close()`: chiude il socket;
→ `public Socket accept() throws IOException`: rimane in ascolto e accetta una connessione. Restituisce il nuovo socket. Il metodo è bloccante fintantoché non viene effettuata una connessione.

3 – Gestione flussi

→ `public InputStream getInputStream() throws IOException`: restituisce lo stream di input legato al socket.
→ `public OutputStream getOutputStream() throws IOException`: restituisce l'output strem legato al socket per scrivere bytes.

21) Cosa risulta necessario considerare durante la progettazione di un'applicazione

Per quanto riguarda il lato client, l'architettura è concettualmente più semplice, ossia un'applicazione che utilizza una socket anziché un altro canale di I/O. Non ci sono particolari problemi di sicurezza. Per quanto riguarda il lato server si prevede un'architettura generale in cui viene creata una socket con una porta nota per accettare le richieste di connessione. Si prevede un ciclo infinito in cui:
→ attesa/accettazione di una richiesta di connessione ad un client;
→ ciclo lettura – esecuzione, invio risposta fino al termine della conversazione, stabilita spesso dal client;
→ chiusura connessione.

22) Quali sono le tipologie di architetture dei server?

Le architetture dei server vengono distinte in quattro tipologie:
→ iterativi: soddisfano una richiesta alla volta;
→ concorrenti processo singolo: simulano la presenza di un server dedicato;
→ concorrenti multi-processo: creano server dedicati;
→ concorrenti multi-thread: creano thread dedicati.

23) Quale metodologia viene utilizzata per effettuare operazioni I/O in modo non bloccante?

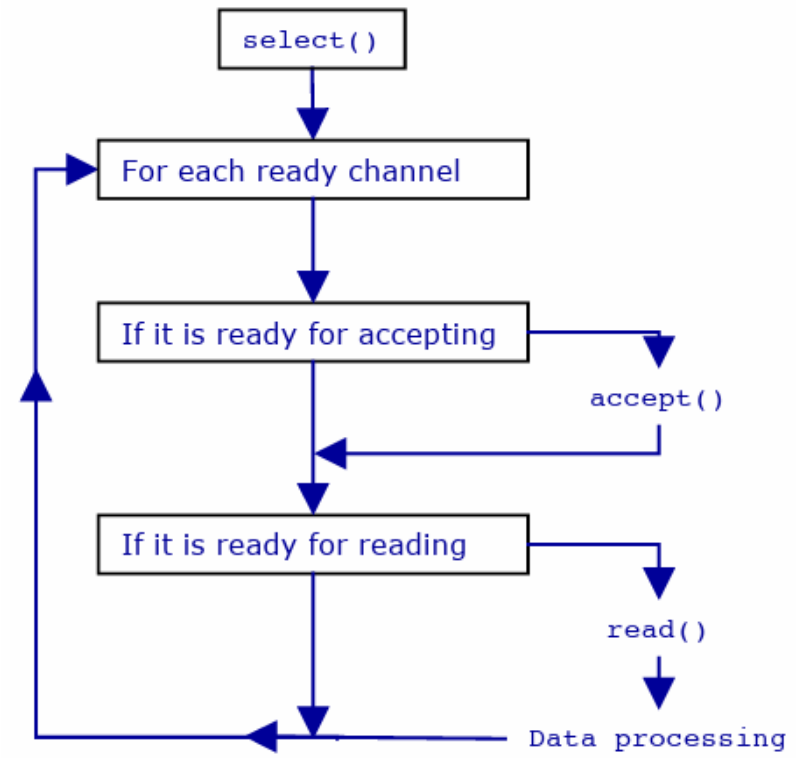
Per leggere in modo non bloccante, prima di fare un'operazione di I/O, serve saper se il canale è pronto o meno e la system call `select()` ha proprio questo compito. In Java vengono utilizzati i channel, che mettono il processo chiamante in sleep e restituisce 1 se è possibile effettuare tale operazione, 0 altrimenti. Le interazioni possono essere effettuate in due modi:
→ con i canali di rete: implementati dalle classi `ServerSocketChannel`, `SocketChannel`, `DatagramChannel`;
→ con i selector: possono essere gestite in modo più efficiente delle socket definite in java.net e permettono di gestire più canali in multiplex.

24) Server Iterativo: descrizione e caratteristiche

Al momento di una richiesta di connessione, il server crea una socket temporanea per stabilire una connessione diretta con il client. Le eventuali ulteriori richieste per il server verranno accodate alla porta nota per essere successivamente soddisfatte. Semplice da progettare ma viene servito un solo client alla volta bloccando l'evoluzione di altri client.

25) Server Concorrente: descrizione e caratteristiche

Un server concorrente può gestire più connessioni client, attraverso la possibilità di simulare più socket utilizzando un solo processo, oppure creando processi slave.
Il comportamento di tale server è sintetizzato nel seguente schema:



26) Server Multi processo: descrizione e caratteristiche

Il server multi processo è un server concorrente che crea nuovi processi slave attraverso le `fork()`, che crea un processo figlio come clone del padre che eredita i canali di comunicazione ed esegue lo stesso codice. Il codice deve prevedere che il padre chiuda la socket per la conversazione con il client e che il figlio chiuda a sua volta la socket per l'accettazione di nuove connessioni. Ha la struttura identica di quello iterativo in quanto ogni server gestisce ogni client.

27) Differenza tra server monoprocesso e multiprocesso

Un server si dice monoprocesso se è iterativo o concorrente, quindi:
→ gli utenti condividono lo stesso spazio di lavoro;
→ adatto ad applicazioni cooperative che prevedono la modifica dello stato, ovvero può avvenire lettura e scrittura.
Un server multiprocesso ha le seguenti caratteristiche:
→ ogni utente ha uno spazio di lavoro autonomo;
→ adatto ad applicazioni cooperative che non modificano lo stato del server, ovvero si effettua solo la lettura;
→ adatto anche ad applicazioni autonome che modificano uno spazio di lavoro proprio, in cui si effettuano lettura e scrittura.

Capitolo 3 – Concorrenza e Programmazione Multithreading

28) Definizione di concorrenza e distinzione tra concorrenza e parallelismo

Si dice concorrenza la contemporaneità di esecuzione di parti diverse di uno stesso programma, ossia la capacità di far progredire due o più attività in modo simultaneo. Si dice parallelismo la capacità di eseguire più di una attività contemporaneamente.

29) Caratteristica principale della legge di Amdhal

La legge di Amdahl fornisce un guadagno in termini di performance causata dall'aggiunta di core ad un'applicazione che ha componenti sia sequenziali che parallele.

30) Cosa si intende per programmazione concorrente?

Si dice programmazione concorrente un paradigma di programmazione in cui vengono implementati programmi contenenti più flussi di esecuzione. Viene utilizzato per sfruttare gli attuali processori multicore e per evitare di

bloccare l'intera esecuzione di un'applicazione a causa dell'attesa del completamento di un'azione di I/O.

31) Cos'è un processo?

Un processo è un'entità attiva astratta definita dal sistema operativo allo scopo di eseguire un programma.

32) Differenza tra programma e processo

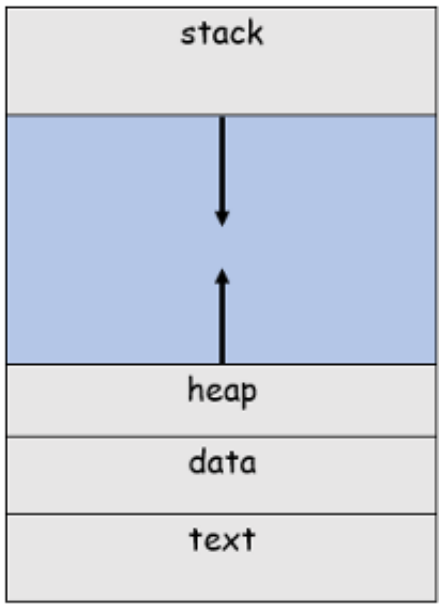
Un programma è un'entità passiva (un insieme di istruzioni, tipicamente contenuto in un file sorgente o eseguibile), mentre il processo è un'entità attiva (è un esecutore di un programma, o un programma in esecuzione).

33) Struttura di un processo: descrivere la sua struttura

Un processo è composto da diverse parti:
→ lo stato dei registri del processore che esegue il programma, incluso il program counter;
→ lo stato della immagine del processo, ossia della regione di memoria centrale usata dal programma;
→ le risorse del sistema operativo in uso al programma (files, lock…);
→ diverse informazioni sullo stato del processo per il sistema operativo.
Attenzione: I processi distinti hanno immagini distinte! Due processi operano su zone di memoria centrale separate!

34) Cosa si intende per spazio di indirizzamento del processo?

Lo spazio di indirizzamento del processo è l'intervallo degli indirizzi di memoria occupati dall'immagine del processo.



35) Da cosa è costituita l'immagine del processo?

L'immagine di un processo di norma contiene:
→ text section: contenente il codice macchina del programma;
→ data section: contenente le variabili globali;
→ heap: contenente la memoria allocata dinamicamente durante l'esecuzione;
→ stack di chiamate: contenente parametri, variabili locali, return address delle varie procedure che vengono invocate durante l'esecuzione del programma.

La text section e data section hanno dimensioni costanti, mentre heap e stack variano durante la vita del processo.

36) Quali sono gli stati di un processo?

Durante l'esecuzione, un processo cambia più volte stato. Gli stati possibili di un processo sono:
→ new: il processo è creato, ma non ancora ammesso all'esecuzione;
→ ready: il processo può essere eseguito ed è in attesa che gli sia assegnata una CPU;
→ waiting: il processo non può essere eseguito perché è in attesa che si verifichi qualche evento;
→ terminated: il processo ha terminato l'esecuzione.

37) Cos'è il context switching?

Si dice context switching una situazione che si verifica quando la CPU deve passare dall'esecuzione di un processo a quella di un altro processo. Tale azione viene effettuata dal dispatcher.

38) Che cosa si intende per multitasking?

Il termine multitasking si indica la capacità di un software di eseguire più programmi contemporaneamente.

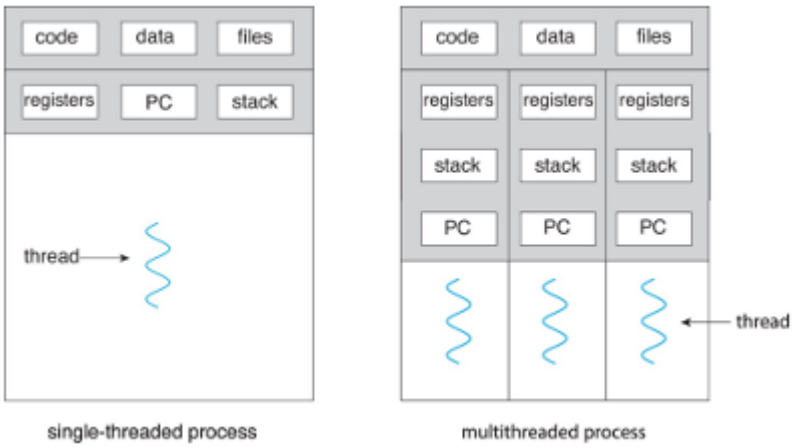
39) Che cosa si intende per multithreading?

Il termine multithreading si indica la capacità di un software di eseguire più thread contemporaneamente.

40) Che cosa si intende per thread?

Si dice thread l'unità di base di utilizzo della CPU caratterizzata da identificatore, PC, stack e registri. I thread di uno stesso processo condividono la memoria globale (data), la memoria contenente il codice (code) e le risorse ottenute dal sistema operativo (ad esempio i file aperti). Ogni thread di uno stesso processo però deve avere un proprio stack, altrimenti le chiamate a

subroutine di un thread interferirebbero con quelle di un altro thread concorrente.



41) Che cosa si intende per thread a livello utente?

Si dice thread a livello utente la tipologia di thread disponibile nello spazio utente dei processi, cioè quello offerto dalle librerie thread ai processi.

42) Che cosa si intende per thread a livello kernel?

Si dice thread a livello kernel la tipologia di thread implementata nativamente dal kernel, ovvero quella tipologia di thread utilizzata per strutturare il kernel stesso in maniera concorrente. Tali thread vengono utilizzati dalle librerie di thread per implementare i thread a livello utente di un certo processo.

Vi sono tre tipologie di thread:

→ molti a uno (N:1): i thread a livello utente di un certo processo sono implementati su un solo thread a livello del kernel, è bloccante ma utilizzabile su ogni SO;

→ uno a uno (1:1): ogni thread a livello utente è implementato su un singolo, distinto thread a livello kernel, aumenta la concorrenza e il parallelismo, ma anche l'overhead;

→ molti a molti (N:N): i thread a livello utente di un certo processo sono implementati su un insieme di thread a livello del kernel possibilmente inferiore di numero, e l'associazione thread utente/ thread kernel è dinamica. Risulta complessa da implementare.

43) Che cosa si intende per LWP?

Un LWP è l'interfaccia offerta dal kernel alle librerie dei thread per usare i thread del kernel. Ogni LWP è un oggetto astratto associato staticamente dal kernel ad esattamente un thread del kernel. Può essere usato dalla libreria dei thread per aggiornare le (minime) informazioni relative al contesto del

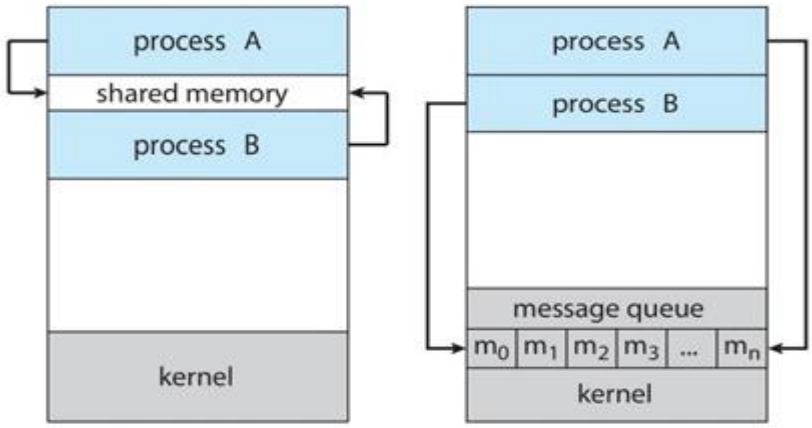
processo utente e tale metodologia consente di assumere migliori decisioni di scheduling.

44) Comunicazione Inter Processo: IPC

La comunicazione inter processo (IPC) permette lo scambio di informazioni e dati tra più processi in evoluzione. Per permettere ciò il sistema operativo deve fornire primitive di comunicazione inter processo (IPC). Vi sono due tipologie di primitive:

→ memoria condivisa: in cui viene stabilita una zona di memoria condivisa tra i processi che intendono comunicare, la comunicazione è controllata dai processi che comunicano, non dal sistema operativo e vi è la sincronizzazione tra i processi;

→ message passing: in cui i processi si comunicano e sincronizzano tra di loro.



45) Qual è il modo principale per creare ed eseguire thread in Java?

Il modo più semplice per creare ed eseguire un thread in Java è:

1. estendere la classe `java.lang.Thread` che contiene un metodo `run()` vuoto;
2. riscrivere, ovvero effettuare un override, del metodo `run()` nella sottoclasse;
3. il codice eseguito dal thread è incluso nel metodo e nei metodi invocati direttamente o indirettamente dallo stesso e il codice verrà eseguito in parallelo a quello di altri thread;
4. creare un'istanza della sottoclasse;
5. richiamare il metodo `start()` che crea l'istanza e fa anche partire il thread.

Il metodo `run()` viene invocato automaticamente e i due thread, creante e creato, saranno eseguiti in modo concorrente ed indipendente.

L'ordine con cui ogni thread eseguirà le proprie istruzioni è noto, ma l'ordine

globale in cui le istruzioni saranno eseguite è indeterminato, ossia non deterministico.

```java
public class HelloThread extends Thread {
    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new HelloThread()).start();
    }
}
```

```
C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\SecondoSemestre\Sistemi_Distribuiti\Esempi\Thread>java HelloThread
Hello from a thread!
```

46) Qual è un modo alternativo per creare ed eseguire thread in Java?

Dato che Java non consente l'ereditarietà multipla, un modo alternativo di realizzare un thread è implementare solamente il metodo `run()`, ovvero implementare l'interfaccia `Runnable`, avendo la possibilità di estendere un'altra classe base, introducendo flessibilità. Per creare thread usando l'interfaccia `java.lang.Runnable` bisogna:

1. definire una classe, implementazione di `Runnable`, dotata di un metodo `run()` significativo;
2. creare un'istanza di questa classe;
3. istanziare un nuovo thread, passando al costruttore l'istanza della classe.

```java
public class RunnableExample implements Runnable{
    public void run() {
        System.out.println("Hello!");
    }

    public static void main(String arg[]){
        RunnableExample re = new RunnableExample();
        Thread t1 = new Thread(re);
        t1.start();
    }
}
```

```
C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\SecondoSemestre\Sistemi_Distribuiti\Esempi\Thread>javac RunnableExample.java

C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\SecondoSemestre\Sistemi_Distribuiti\Esempi\Thread>java RunnableExample
Hello!
```

47) Come vengono terminati i thread in Java?

Un thread in Java viene terminato una volta che il metodo `run()` ritorna. Una volta terminato, il thread non può essere rieseguito.

48) Cosa indica il predicato isAlive()?

Il predicato isAlive() valuta se il thread sia stato fatto partire e allo stesso tempo non sia stato fatto terminare.

49) Cosa succede se si tenta di avviare più volte un thread?

Viene generata l'eccezione `IllegalThreadStateException`.

50) Quale scopo ha il metodo BusyLoop()?

Lo scopo di tale metodo è quello di rallentare l'esecuzione dei thread. Generalmente non è però una buona soluzione, perché si sprecano cicli di processore ed è possibile che quando si giunge al momento in cui il thread deve uscire dal ciclo, un secondo thread non abbia accesso alla CPU, con la conseguenza che esce in ritardo rispetto al momento desiderato.

```
private void BusyLoop() {
    long start = System.currentTimeMills();
    long stop = stratTime + 60000;

    while(System.currentTimeMills() < stop)
        continue;
}
```

51) Quali sono le due tipologie di cancellazione di thread?

Vi sono due approcci di cancellazione dei thread:

→ Cancellazione asincrona: il thread che riceve la cancellazione viene terminato immediatamente. Il vantaggio di questo approccio è la nessuna necessità di controllare periodicamente se ci sono richieste di cancellazione pendenti.

→ Cancellazione differita: un thread che supporta la cancellazione differita deve controllare periodicamente se esiste una richiesta di cancellazione pendente, e in tal caso terminare la propria esecuzione. Il vantaggio di questo approccio è che dal momento che un thread controlla il momento della propria cancellazione, può effettuare una terminazione ordinata.

In Java, la cancellazione avviene attraverso il metodo `interrupt()` che setta un flag di interruzione nel thread di destinazione e ritorna. Il thread che riceve l'interruzione non viene effettivamente interrotto, ma solo quando viene controllato il flag il thread sa di dover terminare. Se un thread è già in stato di pausa e riceve un `interrupt()`, la JVM lancia l'eccezione `InterruptedException`.

```java
public class ThreadSleep extends Thread {
    public ThreadSleep(String s) {
        super(s);
    }

    public void run() {
        for (int i=0; i<5; ++i) {
            System.out.println(getName() + ": in esecuzione.");
            try {
                Thread.sleep(200);
            }
            catch (InterruptedException e) {
                System.err.println(getName() + ": interrotto");
                break;
            }
        }
        System.err.println(getName() + ": finito");
    }
}
```

52) Cosa si intende per threading implicito?

Meccanismo che delega la creazione e gestione dei thread ai compilatori e alle librerie. L'obiettivo è di permettere agli sviluppatori di ragionare in termini di task da compiere, che vengono poi mappati sui thread in maniera automatica.

53) Cosa si intende per thread pool?

È uno degli approcci impliciti di threading, in cui viene creato un certo numero di thread, organizzati in gruppo, che attendono di rispondere ad una richiesta di lavoro. È l'approccio utilizzato per la gestione del ciclo di vita delle servlet.
Ha diversi vantaggi:
→ più rapido che creare il thread all'arrivo della richiesta;
→ separa il task da svolgere dalla meccanica della sua creazione, permettendo diverse strategie di esecuzione.
Un altro metodo di threading implicito, che suddivide un main thread in diverse task è il Fork – Join.

54) Quali sono le tre modalità di interazione tra agenti concorrenti?

Esistono tre modalità di interazione:
→ cooperazione: interazioni prevedibili e desiderate, la loro presenza è necessaria per la logica del programma e avviene tramite scambio di

informazioni, anche semplici come segnali;
→ competizione: gli agenti competono per accedere ad una risorsa condivisa, con la necessità di politiche di accesso alla risorsa;
→ interferenze: interazioni non prevedibili e non desiderate, principalmente errori di programmazione, spesso dipendenti dalle tempistiche e non facilmente riproducibili.

55) Cosa sono i meccanismi di sincronizzazione?

Sono i meccanismi che permettono di controllare l'ordine relativo delle varie attività dei processi/thread. Questi meccanismi vengono messi a disposizione dal sistema operativo nel caso di processi. In Java ritroviamo le stesse soluzioni implementate a livello di JVM per i thread.

56) Cosa si intende per meccanismo di barriera?

Si dice barriera un meccanismo di sincronizzazione, il quale prevede che i thread vengono messi progressivamente in attesa che una condizione globale non venga soddisfatta. Tale meccanismo viene implementato mediante un contatore condiviso che conta il numero di processi che devono ancora terminare e decrementa ogni volta che un thread termina il suo task.

57) Cosa si intende per race condition?

La race condition è un fenomeno che si verifica nei sistemi concorrenti quando, in presenza di una sequenza di processi multipli, il risultato finale dell'esecuzione dei processi dipende dalla temporizzazione o dalla sequenza con cui vengono eseguiti. Una semplice soluzione al problema consiste nel fornire accesso mutuamente esclusivo alla risorsa, logicamente non interrompibile.

58) Cosa si intende per sezione critica?

La sezione critica è un blocco di codice che può essere eseguito da un solo thread alla volta. Per realizzare questo tipo di operazione risulta necessario di un'operazione non atomica e non interrompibile `test_and_set(B)` in cui dato B, un valore booleano, restituisce il valore originario della cella di memoria puntata e imposta a **true** il valore della stessa.

59) Cos'è il lock?

Il lock è una variabile logica booleana manipolabile atomicamente. Quando un thread acquisisce un lock, gli altri thread che lo richiedono si bloccano fintantoché il thread che lo detiene non lo rilascia. Per questo motivo che ogni lock deve avere e gestire una coda di thread in attesa.

60) Cosa si intende per semaforo?

Il semaforo è una variabile intera che può essere manipolata solo attraverso operazioni atomiche `acquire()`, che aspetta che la variabile sia maggiore di 0, e quindi la decrementa e `release()`, che invece incrementa in condizionalmente la variabile. Anch'esso deve avere una coda di attesa.

61) Come viene implementata in Java la mutua esclusione?

In Java la mutua esclusione viene implementata associando ad un metodo la keyword **synchronized** e associa un lock intrinseco ad ogni oggetto che abbia almento un metodo **synchronized**. Quando un thread T1 sia in esecuzione all'interno di un metodo **synchronized** fa sì che altri thread che richiedano l'esecuzione dello stesso o un altro metodo **synchronized** vengano messi in attesa che T1 completi l'esecuzione del metodo.

```
public synchronized void deposito(float cifra) {
    saldo += cifra;
}

public synchronized void prelievo(float cifra) {
    if (saldo > cifra)
        saldo -= cifra;
}
```

62) Cos'è il monitor?

Il monitor è una primitiva a più alto livello rispetto a lock e semafori, di tipo astratto, le cui variabili interne sono accessibili solo da un insieme di procedure esposte dal monitor stesso. Solo un processo/thread alla volta può essere attivo nel monitor. All'interno, vi possono essere delle variabili di tipo condizione, su cui è possibile effettuare operazioni di:

→ `x.wait()`: mette in stato di attesa il processo corrente, e lo forza a lasciare il monitor;

→ `x.signal()`: rende ready un processo che aveva invocato il metodo precedente, se esiste.

In Java, ogni Object è un monitor, e quindi i metodi wait, notify, notifyAll sono presenti in ogni classe, però la gestione dei meccanismi di accesso condizionato che utilizzano questi metodi sono completamente a carico del programmatore.

63) Come viene risolto il problema del Produttore – Consumatore?

Il problema del Produttore – Consumatore consiste nell'assicurare che:

→ il produttore non cerchi di inserire nuovi dati quando il buffer è pieno;

→ il consumatore non cerchi di estrarre dati quando il buffer è vuoto.

Per risolvere tale problema risulta necessario trovare un meccanismo di controllo sulla risorsa che garantisca che certe condizioni siano verificate. Dal punto di vista implementativo:
→ se il buffer è pieno, non inserire nulla e aspetta, il produttore deve quindi sospendere la propria esecuzione fintantoché il buffer è pieno;
→ se il buffer è vuoto, non provare a cancellare niente e aspettare, ovvero il consumatore si sospende fintantoché il buffer è vuoto.

64) Quando un'operazione viene definita atomica?

Un'operazione è atomica quando tutte le sub-operazioni che la compongono verranno eseguite senza possibilità di interruzione da parte di un altro thread.

65) Quando si verifica il problema della visibilità?

Il problema della visibilità si crea quando due thread sono in esecuzione su core diversi, perché le variabili condivise vengono tenute in cache per ragioni di performance. Si può ovviare al problema della visibilità utilizzando variabili volatile.

66) Che problemi presenta il locking?

Il processo di locking viene detto pessimistico, infatti se la contesa non è frequente, nella maggior parte dei casi la richiesta e l'esecuzione di un lock non è necessaria ed aggiunge overhead.
Risulta utile considerare che:
→ se un thread non riesce ad acquisire un lock viene sospeso;
→ risvegliare un thread presenta un costo;
→ un thread in attesa non può eseguire nessuna operazione;
→ un thread a bassa priorità può bloccare thread che ne hanno una più alta (priority inversion), infatti quando un thread acquisisce un lock nessun altro thread che ha bisogno di quel lock può proseguire.

67) Cos'è il CAS (Compare and Swap)?

Il CAS è un'istruzione atomica utilizzata nel multithreading per ottenere la sincronizzazione. Confronta il contenuto di una locazione di memoria con un dato valore e, solo se sono gli stessi, modifica il contenuto di quella locazione di memoria ad un nuovo dato valore.

68) Quali sono le operazioni che vengono effettuate con le variabili atomiche?

Java dispone di 12 classi contenitori per implementare le variabili atomiche, su cui si possono effettuare operazioni di:

→ `int get()`: restituisce il valore corrente della variabile
→ `void set(int newValue)`: aggiorna il valore corrente della variabile
→ `int getAndSet(int newValue)`: aggiorna atomicamente il valore corrente della variabile e restituisce il valore contenuto precedentemente
→ `boolean compareAndSet(int expected, int update)`: aggiorna atomicamente il valore della variabile update se il valore corrente della variabile è uguale a expected;
→ `int getAndIncrement()`: incrementa atomicamente il valore corrente della variabile e restituisce il valore contenuto precedentemente.

69) Cosa si intende per liveness?

Si dice liveness il fatto che un programma sia sempre in grado di progredire correttamente evitando problemi. A tal fine i meccanismi di sincronizzazione devono fare in modo che un processo non aspetti che una risorsa venga rilasciata oppure che tutti i processi si "blocchino" in attesa di eventi che non possono verificarsi. Garantisce che prima o poi tutti i processi entrino in uno stato corretto e progrediscano verso il completamento.

70) Cosa si intende per deadlock?

Si dice deadlock l'evento in cui due o più processi rimangono in attesa di eventi che non potranno verificarsi a causa di condizioni cicliche nel possesso e nella richiesta di risorse.
Ci sono quattro condizioni necessarie affinché si verifichi un deadlock:
→ mutua esclusione: solo un'attività concorrente (thread o processo) per volta può utilizzare una risorsa;
→ hold and wait: attività concorrenti che sono in possesso di una risorsa possono richiederne altre senza rilasciare la prima;
→ no preemption sulle risorse: una risorsa può essere rilasciata solo volontariamente da un'attività concorrente;
→ attesa circolare: deve esistere una possibile catena circolare di attività concorrenti e di richieste di accesso a risorse concorrenti tale che ogni attività mantiene bloccate delle risorse che contemporaneamente vengono richieste dai attività successive.
Ci sono due modalità per prevenire un deadlock:
→ deadlock prevention: il deadlock può essere evitato se si fa in modo che almeno una delle quattro condizioni richieste per deadlock non si verifichi mai.

→ deadlock removal: non si previene il deadlock, ma lo si risolve quando ci si accorge che è avvenuto.

71) Cosa si intende per starvation?

Si dice starvation il fenomeno che si verifica quando un processo rimane in attesa di un evento che non accadrà mai e quindi non può portare a termine il proprio lavoro.

72) Cosa si intende per livelock?

Si dice livelock il continuo tentativo di un'azione che fallirà sempre.

73) Cosa consiste il problema dei 5 filosofi a cena?

Il problema consiste nello sviluppo di un algoritmo che impedisca situazioni di deadlock o starvation. Se tutti i filosofi prendessero la forchetta di sinistra, si raggiungerebbe un deadlock, e la stessa cosa vale per qualsiasi permutazione di questo insieme di azioni. Vi sono alcune soluzioni a tale problema:
→ soluzione di Dijkstra: questa soluzione al problema, spezza la simmetria che causa l'attesa circolare e viene considerato l'ordine tra le forchette e si forza a prendere prima quella con identificativo minore;
→ soluzione basata sull'attesa circolare: se un filosofo ha acquisito una forchetta e non è riuscito a procurarsi anche la seconda, può ipotizzare che si stia creando la condizione per un deadlock, quindi rilascia la forchetta in suo possesso e attende un po' di tempo prima di riprovare a impossessarsi delle forchette;
→ soluzione basata su rimozione di hold and wait: soluzione basata su rimozione di hold and wait, ovvero asta che ogni filosofo prenda (con una operazione atomica) entrambi le forchette se disponibili, e aspetti se invece non sono disponibili. Ci vuole un mediatore che osservi lo stato delle forchette e agisca di conseguenza.

74) Problema lettori e scrittori

Si ha una risorsa condivisa che può essere letta da un certo numero thread in contemporanea ma il cui accesso in scrittura deve invece essere esclusivo (per questioni di consistenza del dato). La soluzione prevede che:
→ la lettura non sia interamente sincronizzata, il controllo che non ci siano scrittori attivi e che il numero di lettori sia inferiore al massimo permesso deve essere sincronizzata;

→ la scrittura deve essere invece completamente sincronizzata ma non deve avvenire se anche solo un lettore sta effettuando il suo lavoro.

Capitolo 4 – Message Oriented Communication

75) Definizioni di Browser e Web browser

Si definisce browser l'applicazione per il Web sul lato del client che ha la funzione primaria di interpretare il codice con cui sono espresse le informazioni (pagine web) e visualizzarlo (operazione di rendering) in forma di ipertesto. Un web browser (detto User-Agent) è un programma che consente la navigazione nel Web da parte di un utente.

76) Cosa si intende per pagina Web?

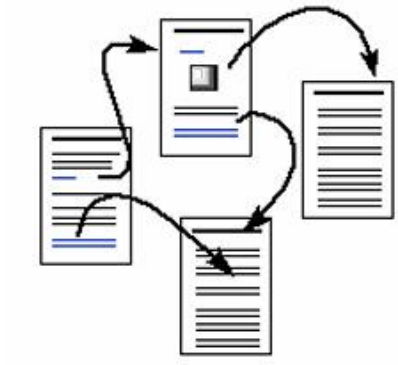
Una pagina Web (web page, o anche documento) è concettualmente un file, cioè una sequenza di dati (in formato digitale) residente in un calcolatore, che è identificato da una URL (indirizzo univoco per la risorsa).
Più precisamente, una pagina web è un file HTML che definisce la struttura e i contenuti della pagina, testo più altri elementi (immagini, elementi grafici, …).

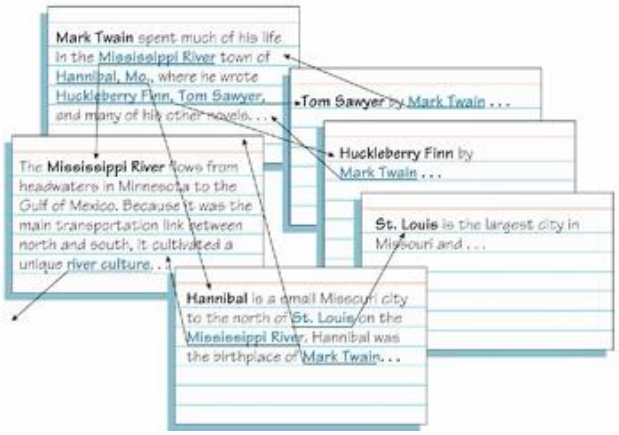
77) Cosa si intende per Server Web?

Un Web Server è una applicazione che si occupa di gestire le risorse (oggetto che può avere rappresentazioni diverse, che possono essere resi disponibili) su un elaboratore e di renderle disponibili ai client.

78) Cosa si intende per ipertesto?

Si dice ipertesto un insieme di testi o pagine che possono essere letti in maniera non sequenziale secondo percorsi definiti dagli hyperlink (detti semplicemente link). Compongono una rete raggiata o variamente incrociata di informazioni organizzate secondo criteri paritetici o gerarchici (esempio menu). Risulta essere il modello fondativo del web.





79) Cosa si intende per URL (Uniform Resource Locator)?

L'indirizzo URL (Uniform Resource Locator) è una stringa di caratteri che identifica e individua una risorsa su Internet. Identifica un oggetto nella rete e specifica il protocollo da usare per ricevere/inviare dati e ha cinque componenti principali:

→ nome del protocollo;
→ indirizzo dell'host;
→ porta del processo (controparte della comunicazione);
→ percorso nell'host;
→ identificatore della risorsa.

protocollo://indirizzo_IP[:porta]/cammino/risorsa RFC 3986

Esempi:

```
ftp://www.adobe.com/dawnload/acroread.exe
http://www.biblio.unimib.it/go/Home/Home-English/Services
http://www.biblio.unimib.it/link/page.jsp?id=47502837
http://www.someSchool.edu/someDept/pic.gif
http://www.someSchool.edu:80/someDept/pic.gif
```

80) Protocollo HTTP: definizione e caratteristiche

Il protocollo HTTP è un protocollo del livello applicativo del modello TCP/IP delle reti di telecomunicazioni considerato stateless, dato che non mantiene informazioni sulle richieste precedenti del client. Tale protocollo ha le seguenti caratteristiche:

→ protocollo di livello applicativo per il Web;
→ usa il modello client/server, in cui il client, rappresentato dal browser, richiede, riceve e "mostra" oggetti Web mentre il server, rappresentato dal Web Server, invia oggetti in risposta alle richieste;
→ il client inizia una connessione TCP (crea una socket) verso il server sulla porta 80 e il server accetta la connessione TCP dal client;
→ vengono scambiati messaggi http tra il browser e il Web server.

81) Quali sono i metodi HTTP?

Metodo HTTP	Descrizione
GET	Restituisce una rappresentazione di una risorsa, includendo eventuali parametri in coda alla URL della risorsa. La sua esecuzione non ha effetti sul server, per questo è considerato safe.

POST	Comunica dei dati da elaborare lato server o crea una nuova risorsa subordinata all'URL indicata. La POST prevede che i dati vengano messi in coda come documento autonomo (body). Ha proprietà di idempotenza ovvero gli effetti collaterali di più richieste identiche sono gli stessi di una singola richiesta.
HEAD	Simile al metodo GET ma viene restituito solo l'head delle pagine web, spesso usato in fase di debugging.
PUT	Richiede che l'entità racchiusa venga archiviata nella Request – URI fornita.
DELETE	Richiede che elimini la risorsa identificata dalla Request – URI.
TRACE	Viene utilizzata per invocare un loopback remoto a livello di applicazione del messaggio di richiesta.
OPTIONS	Rappresenta una richiesta di informazioni sulle opzioni di comunicazione disponibili sulla catena di richiesta/risposta identificata dalla Request – URI.

82) Quali le due tipologie di messaggi HTTP e che struttura hanno?

Esistono due tipi di messaggi, request e response, che hanno però la stessa struttura:

→ start-line obbligatorio: specifica il protocollo ed è necessario lo spazio tra le parti del protocollo con un Carriage – Return Line Feed alla fine.

→ header lines opzionale: formato da coppie nome-valore arbitrarie, sono qualificatori di domanda e risposta e si ha CRLF finale per ogni coppia ed empty line finale per terminare l'header line;

→ payload opzionale: il contenuto vero e proprio che vogliamo mandare.

83) Codici di stato response: cosa e quali sono?

I codici di stato forniscono informazioni relative alla response:

→ 1xx: richiesta ricevuta;

→ 2xx: richiesta ricevuta, compiuta, compresa, accettata e servita;

→ 3xx: azione aggiuntiva deve essere presa;

→ 4xx: la richiesta contiene errori e non può essere compresa;

→ 5xx: il server fallisce l'adempimento a una richiesta apparentemente valida.

84) Definizione di MIME

Il MIME (Multipurpose Internet Mail Extension) è un protocollo utilizzato per qualificare i dati inviati via internet in 5 sottotipi: image, text, audio, application e video.

85) Perché è stato introdotto l'utilizzo dei cookie?

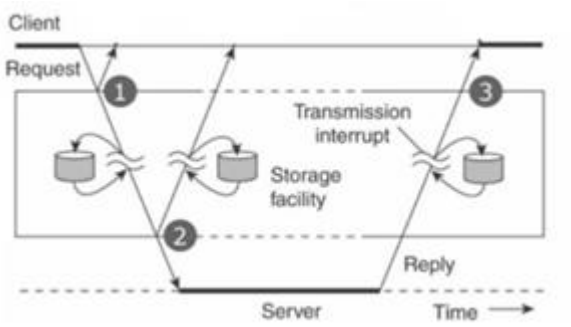
Perché evita stateless ed ha come obiettivo legare più richieste per associare un identificatore alla conversazione. Il server invia un cookie che l'utente presenta in accessi successivi.

86) Come avviene la comunicazione a flusso di messaggio?

Un'applicazione, invia i messaggi come stream di byte al servizio di trasporto, e legge lo stream di byte dal servizio di trasporto e ricostruisce i messaggi. Il servizio UDP scompone lo stream di byte ricevuto in segmenti e invia i segmenti, con una determinata politica, ai servizi network. Il servizio TCP invece scompone e invia come UDP, numera ogni segmento per garantire il riordinamento dei segmenti arrivati, il controllo delle duplicazioni e il controllo delle perdite.

87) Quali sono i tipi di connessione e sincronizzazione?

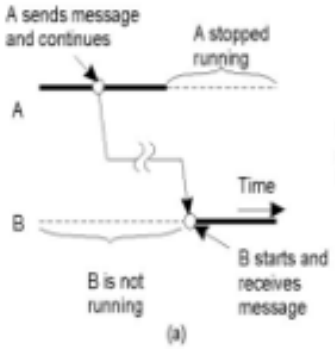
Una connessione può essere:
→ sincrona: due o più interlocutori sono collegati contemporaneamente;
→ asincrona: non richiede il collegamento contemporaneo degli interlocutori alla rete di comunicazione.
Una sincronizzazione può avvenire in 3 punti:
→ request submission: il client invia un messaggio che viene preso in carico dal middleware;
→ request delivery: viene poi trasferito al middleware ricevente;
→ request processing: o passa al server che tratta il messaggio ricevuto.
Questa si divide in 3 ulteriori fasi:
a) ricevuto;
b) letto;
c) processato.
Nelle varie fasi, i buffer utilizzati tracciano i messaggi che permettono la sincronizzazione.



La sincronizzazione può essere:
→ transiente: il destinatario non è connesso e i dati vengono scartati;
→ persistente: il middleware memorizza i dati fino alla consegna del messaggio al destinatario e non è necessario che i processi siano in esecuzione prima e dopo l'invio/ricezione dei messaggi.

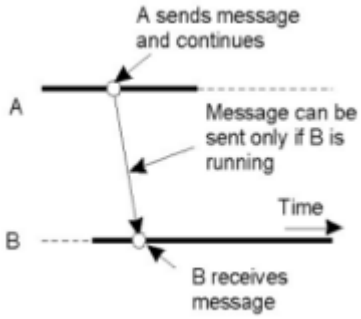
88) Quali sono le tipologie di comunicazione?

Si hanno le seguenti tipologie di comunicazione:
→ comunicazione persistente asincrona: il middleware mantiene il messaggio fin quando il receiver non si connette e riceve il messaggio; B può anche non essere immediatamente attivo e ritorna il controllo prima che ogni altra cosa sia successa, quindi A invia e continua;

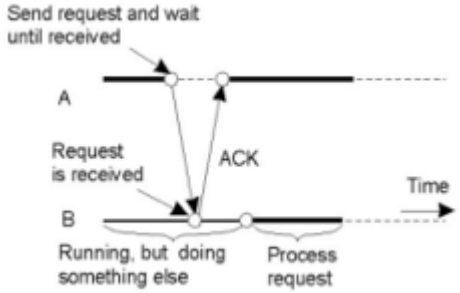


→ comunicazione persistente sincrona: A attende l'accettazione da parte del middleware di B;

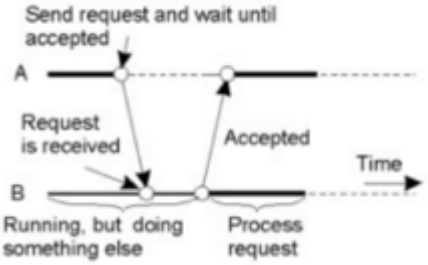
→ comunicazione transiente asincrona: A può inviare solo se B è in run e pronto a ricevere;



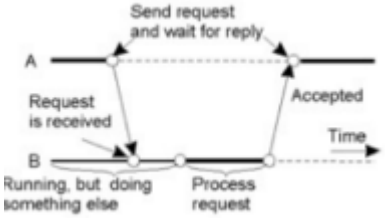
→ comunicazione transiente sincrona basata su ricevuta: B non processa subito, legge solo inviando un ack, e processa più avanti. Fa diventare persistente la comunicazione, se il sistema non lo fa, è il programma che mantiene il dato non un middleware;



→ comunicazione transiente sincrona basata su consegna: uguale al modello precedente, ma deve riceverla e poi leggerla anche prima di inviare un accept;



→ comunicazione transiente sincrona basata su risposta: dopo aver mandato la richiesta aspetta la risposta.



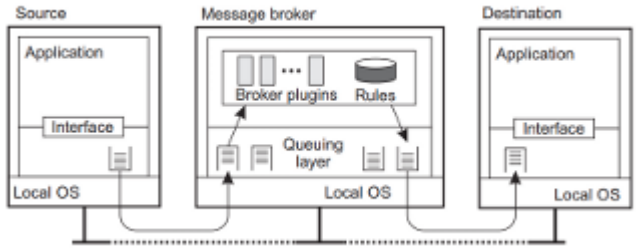
89) Cosa si intende per message – queueing model?

Si dice message – queueing model un meccanismo che offre capacità di archiviazione senza richiedere che il mittente o destinatario siano attivi durante la trasmissione di messaggi. Ci sono 4 modalità, a seconda dello stato di

sender e receiver, che può essere running o passive.
Le quattro primitive sono:
→ put: aggiunge un messaggio ad una coda;
→ get: blocca fino a quando la coda specificata non è vuota e rimuove il primo messaggio;
→ poll: controlla una coda specifica e rimuove il primo messaggio senza bloccare;
→ notify: installa un gestore da chiamare quando un messaggio viene inserito in una coda.

90) Cosa sono il message broker e il PaS Pattern?

Il message broker è un meccanismo che permette il pattern publish e subscribe. Ha il compito di ricevere dati da una sorgente e condividerli a tutti quelli che ascoltano su un dato canale.



Il PaS Pattern è un disaccoppiamento tra publisher e subscriber:
→ publisher: si registrano dichiarando di voler inviare messaggi su un determinato argomento;
→ subscriber: si registrano dichiarando di voler ricevere messaggi su un determinato argomento.
Tra i due fattori si ha un disaccoppiamento, quindi indipendenza e concorrenza, con scalabilità. Il broker può mantenere i dati nel tempo, a seconda di diverse politiche. La MQTT è l'implementazione più diffusa di tale pattern.