Sistemi centralizzati

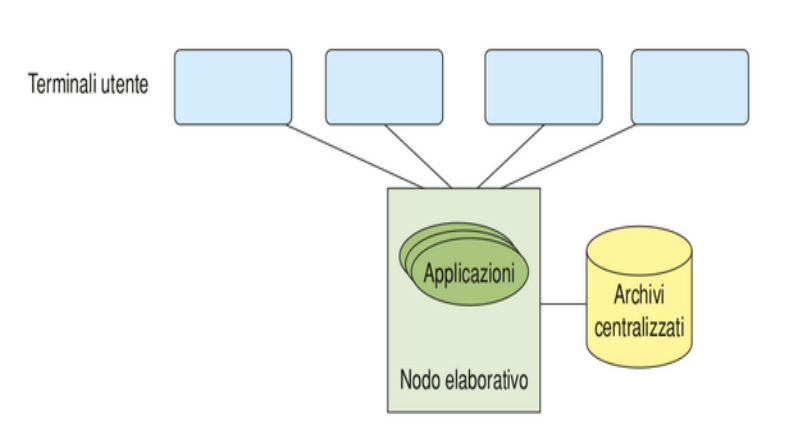
Dati e programmi sono nello stesso posto

Sistemi distribuiti

Dati e programmi possono essere in luoghi diversi (anche geograficamente)

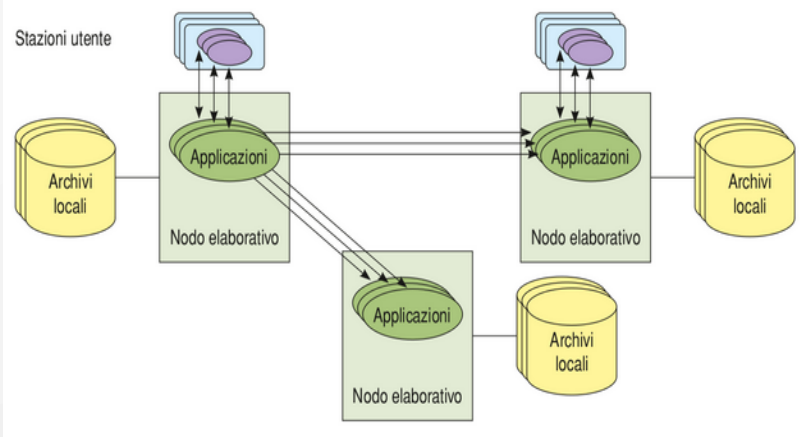
Sistemi centralizzati

Dati e programmi sono nello stesso posto



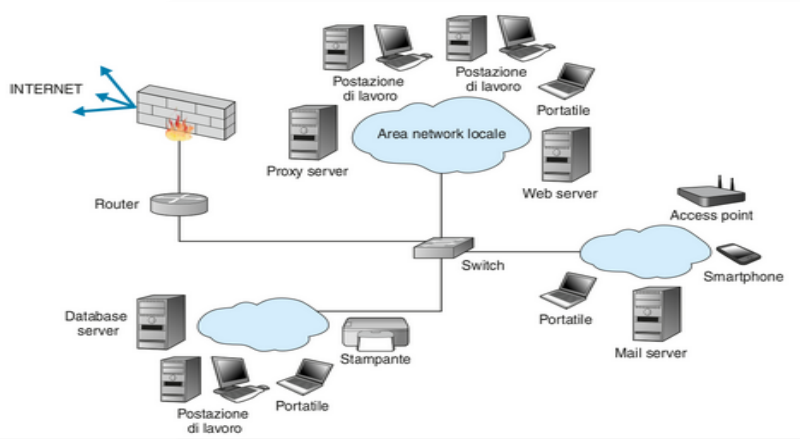
Sistemi distribuiti

Dati e programmi possono essere in luoghi diversi (anche geograficamente)

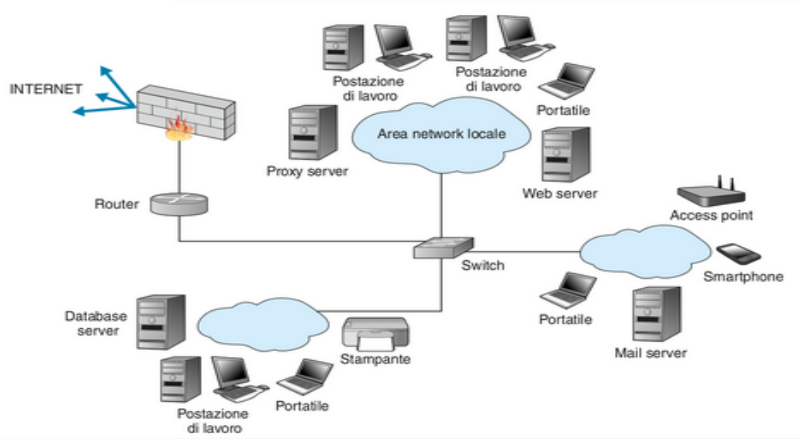


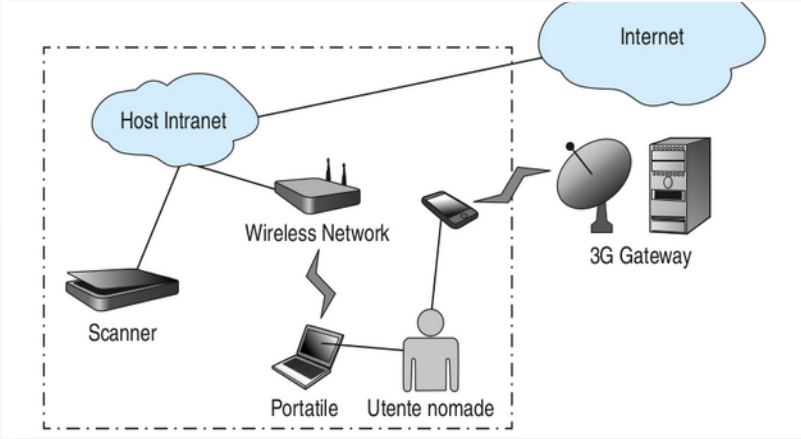
ESEMPI

ESEMPI

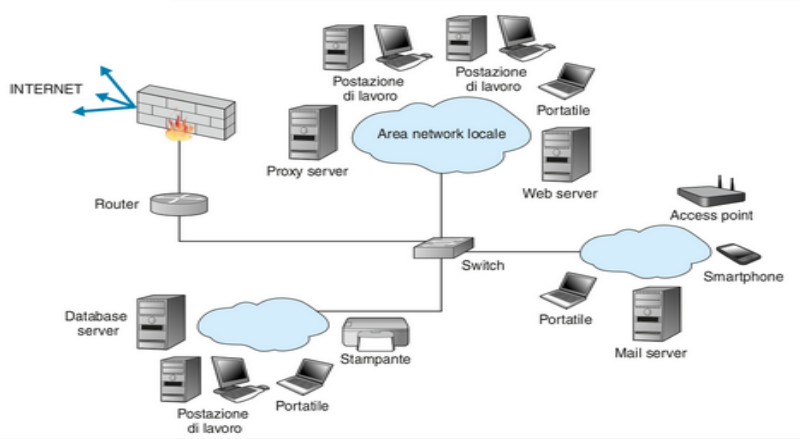

INTERNET
Postazione di lavoro
Postazione di lavoro
Portatile
Area network locale
Proxy server
Web server
Router
Access point
Switch
Smartphone
Database server
Portatile
Stampante
Mail server
Postazione di lavoro
Portatile

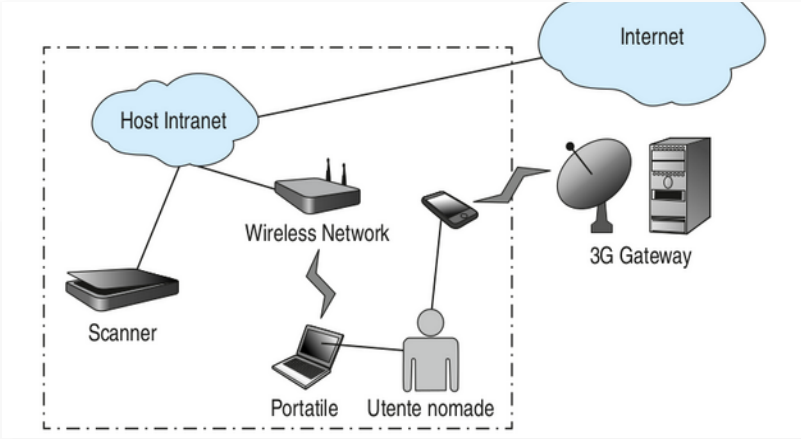
ESEMPI

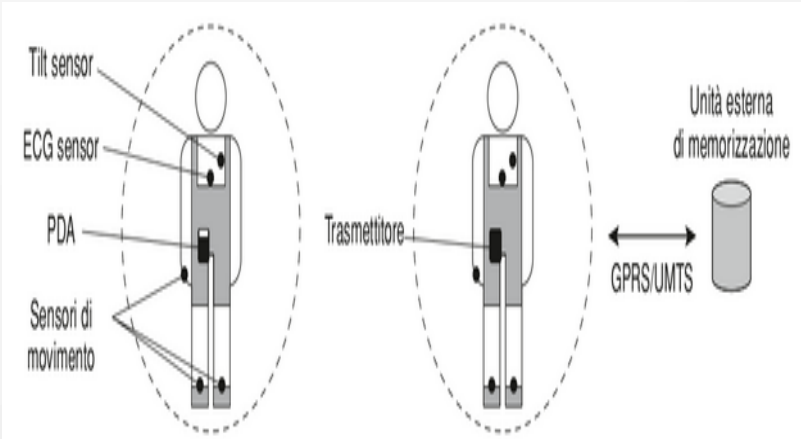

INTERNET
Postazione di lavoro
Postazione di lavoro
Portatile
Area network locale
Proxy server
Web server
Router
Access point
Switch
Smartphone
Portatile
Database server
Stampante
Mail server
Postazione di lavoro
Portatile


Internet
Host Intranet
Wireless Network
3G Gateway
Scanner
Portatile
Utente nomade

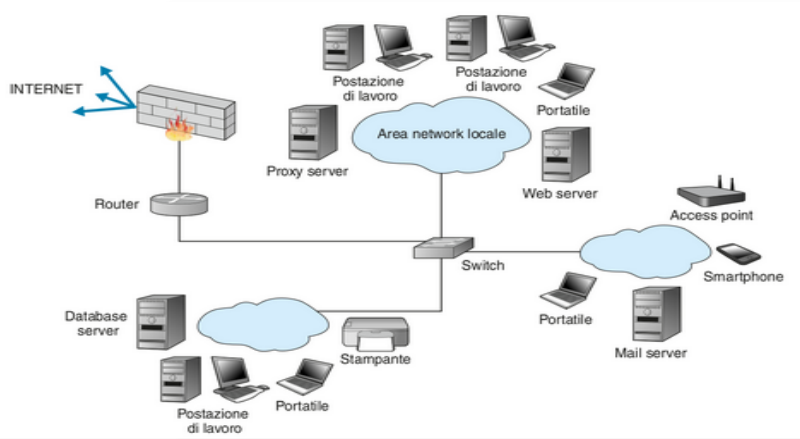
ESEMPI

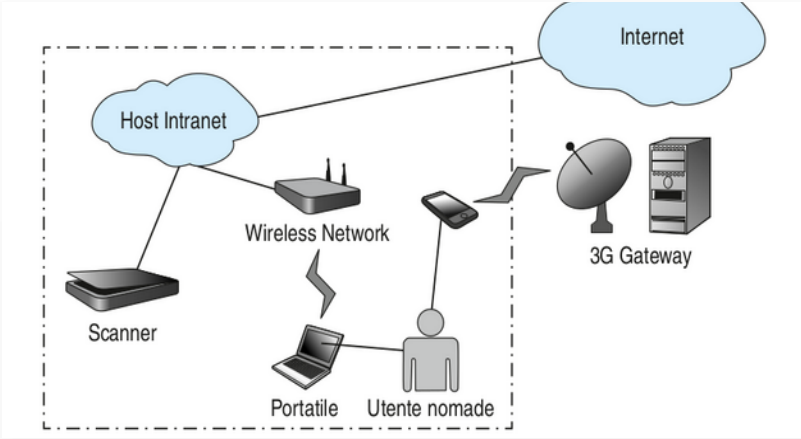

INTERNET
Postazione di lavoro
Postazione di lavoro
Portatile
Area network locale
Proxy server
Web server
Router
Access point
Switch
Smartphone
Database server
Portatile
Stampante
Mail server
Postazione di lavoro
Portatile

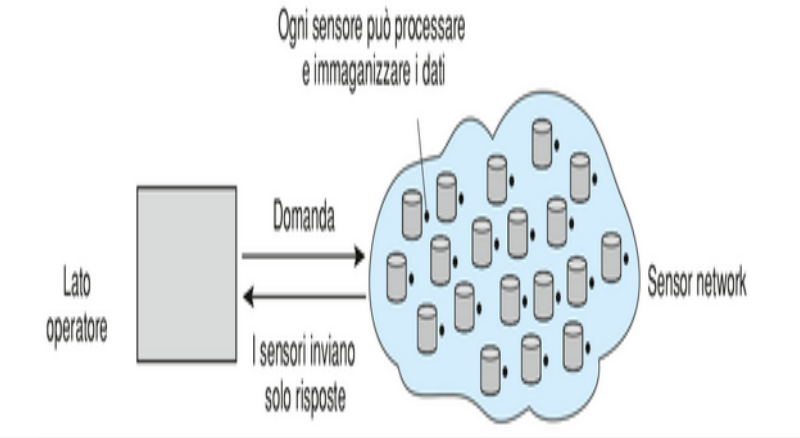

Internet
Host Intranet
Wireless Network
3G Gateway
Scanner
Portatile
Utente nomade

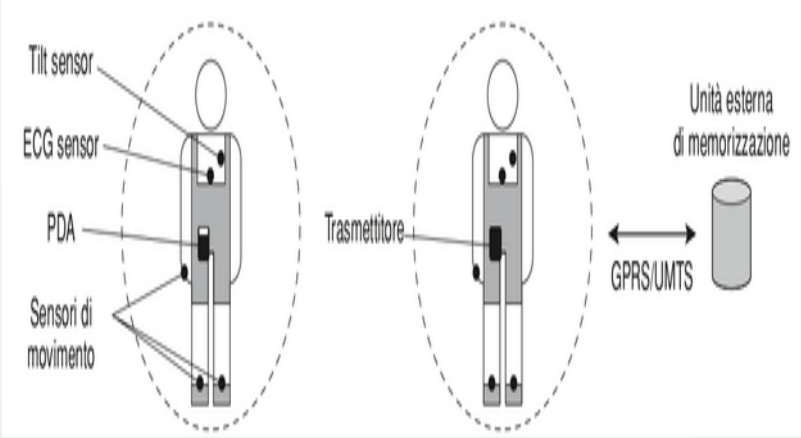

Tilt sensor
ECG sensor
PDA
Sensori di movimento
Trasmettitore
Unità esterna di memorizzazione
GPRS/UMTS

ESEMPI


INTERNET
Postazione di lavoro
Postazione di lavoro
Portatile
Area network locale
Proxy server
Web server
Router
Access point
Switch
Smartphone
Database server
Portatile
Stampante
Mail server
Postazione di lavoro
Portatile


Internet
Host Intranet
Wireless Network
3G Gateway
Scanner
Portatile
Utente nomade


Ogni sensore può processare e immaganizzare i dati
Domanda
Lato operatore
Sensor network
I sensori inviano solo risposte


Tilt sensor
ECG sensor
PDA
Sensori di movimento
Trasmettitore
Unità esterna di memorizzazione
GPRS/UMTS

Dal punto di vista del calcolo

Sistemi centralizzati

- Flusso strettamente sequenziale
- Unica memoria
- Programmazione semplice
  - La difficoltà sta nel progetto degli algoritmi

Dal punto di vista del calcolo

Sistemi centralizzati

- Flusso strettamente sequenziale
- Unica memoria
- Programmazione semplice
  - La difficoltà sta nel progetto degli algoritmi

Sistemi distribuiti

- Flusso non strettamente sequenziale
- Memorie multiple
- Programmazione complessa
  - Algoritmi + Comunicazione dei dati

Perché?

Ci sono dei limiti fisici alle prestazioni

Posso oltrepassare questi limiti usando più componenti

- Sistemi Multicore
- Cluster di computer
- Reti di computer

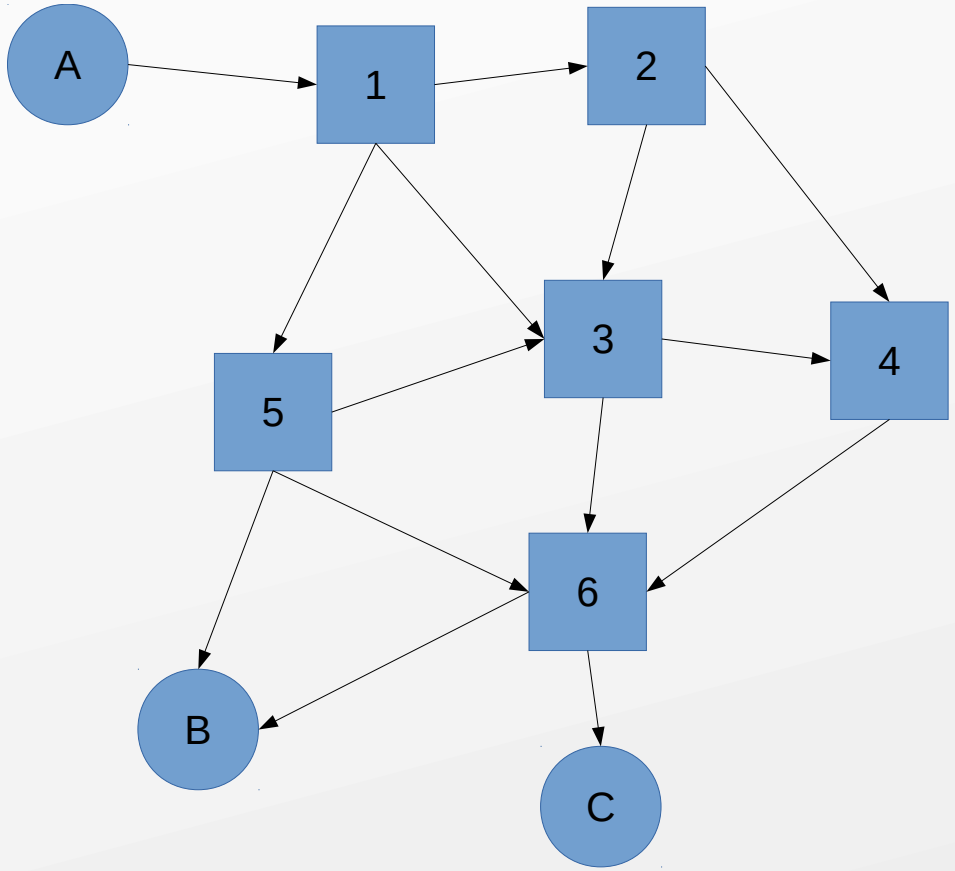
INTEGRAZIONE + PRESTAZIONI + SCALABILITÀ

Perché?

Utilizzare più risorse mi permette di gestire le criticità

- Unità di backup
- Macchine sostitutive
- Potenza di calcolo su richiesta

AFFIDABILITÀ + TOLLERANZA AI GUASTI

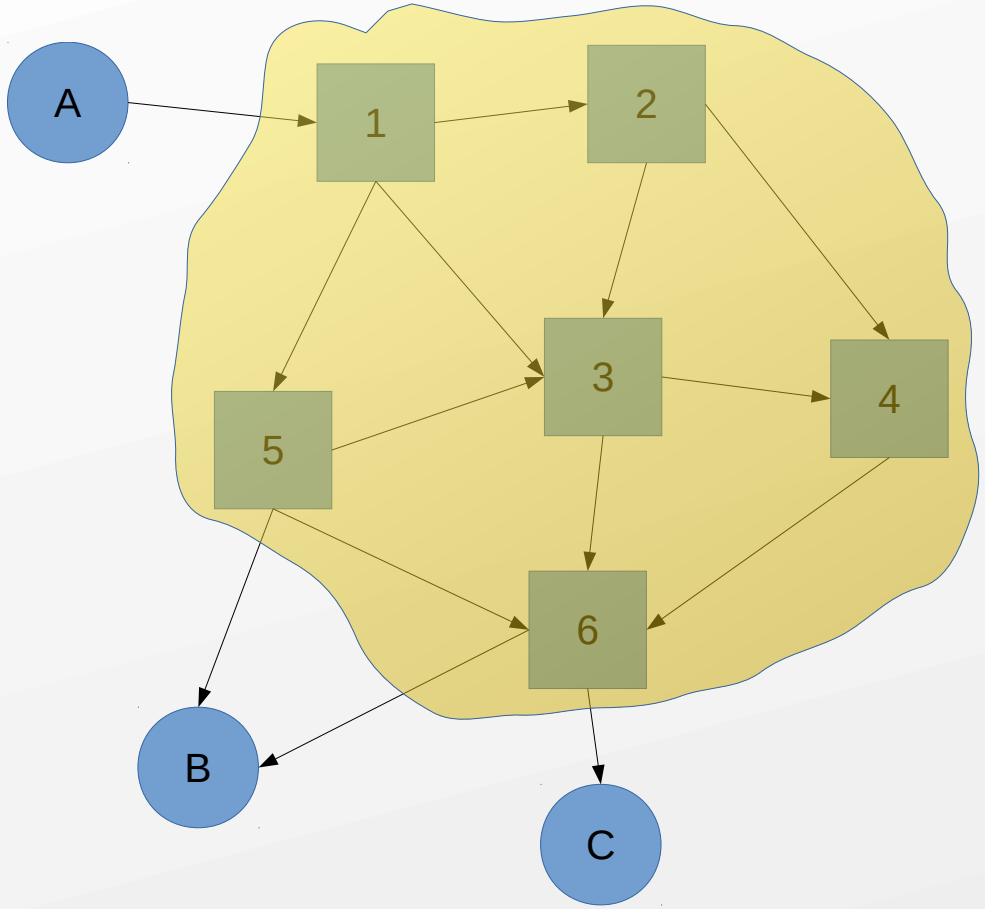


Caratteristiche

Gli utenti non devono sapere come è implementato il sistema sia dal punto di vista HW che SW

Accedono alle risorse chiamandole per "nome" ma non sanno né dove sono né come sono configurate

TRASPARENZA

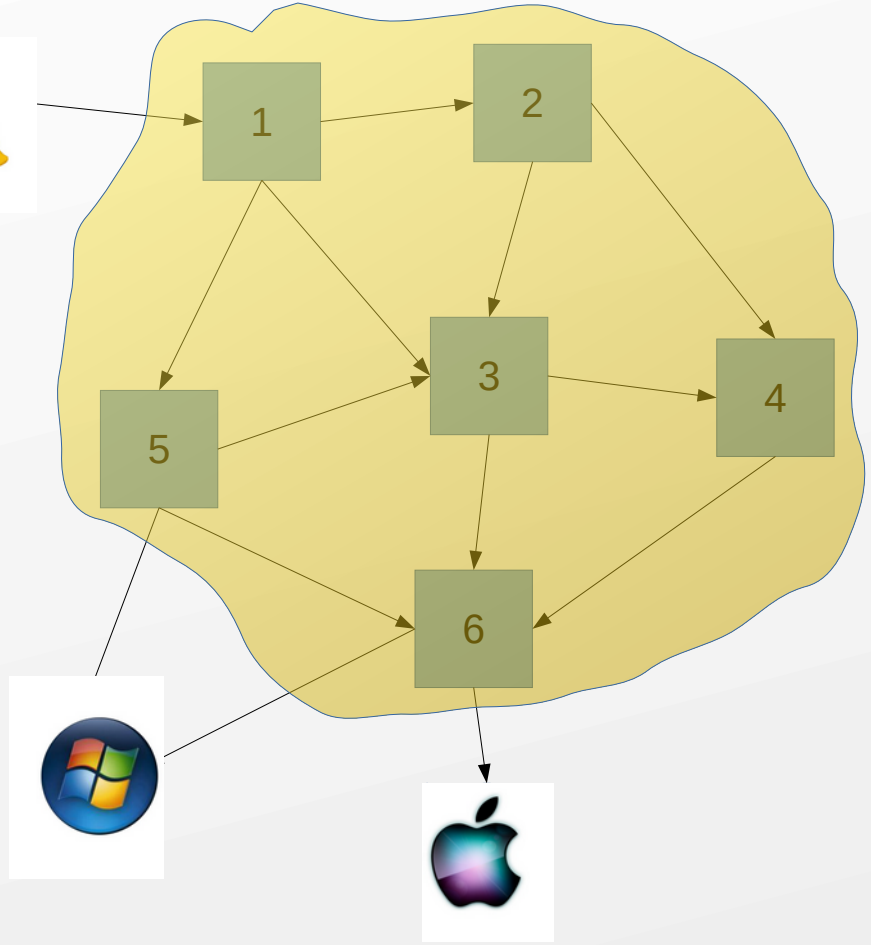


Caratteristiche

Deve essere possibile la comunicazione fra componenti diversi sia HW che SW

Un sistema distribuito è, in genere, molto eterogeneo. Macchine con caratteristiche HW diverse, sistemi operativi diversi, luoghi geografici diversi

APERTURA



Problemi

Cambia l'approccio alla programmazione

- Algoritmi + Protocolli

Aumenta la complessità

- Bisogna gestire più risorse fisiche

Aumenta il grado di sicurezza richiesto

- Bisogna gestire gli accessi e la sicurezza dei dati

Aumentano le risorse HW necessarie

- Bisogna prevedere meccanismi di recupero ripristino (ridondanza)

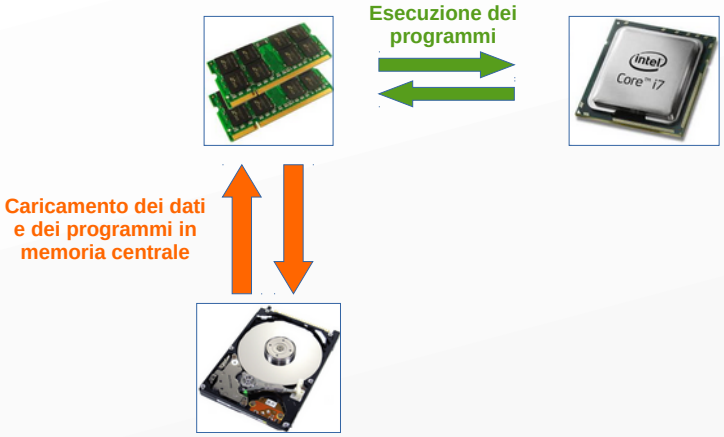
Architetture distribuite hardware

Tassonomia di Flynn

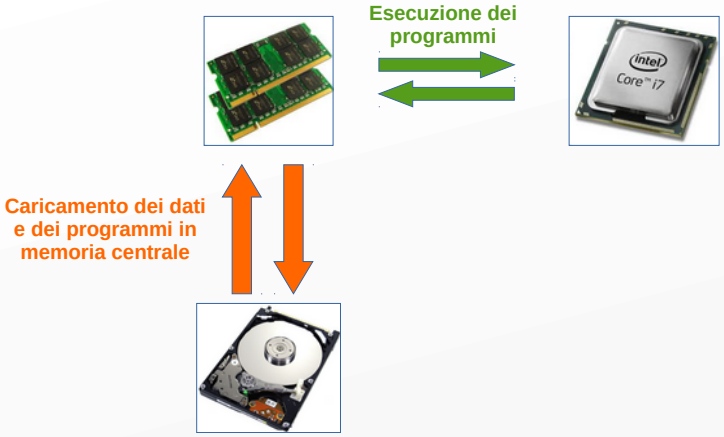
Sviluppata per classificare i sistemi di calcolo in base al rapporto fra i dati in ingresso e le operazioni che su di essi vengono svolte

	Dati singoli	Dati multipli
Istruzioni singole	SISD	SIMD
Istruzioni multiple	MISD	MIMD

SISD


Esecuzione dei programmi
Caricamento dei dati e dei programmi in memoria centrale
intel
Core i7

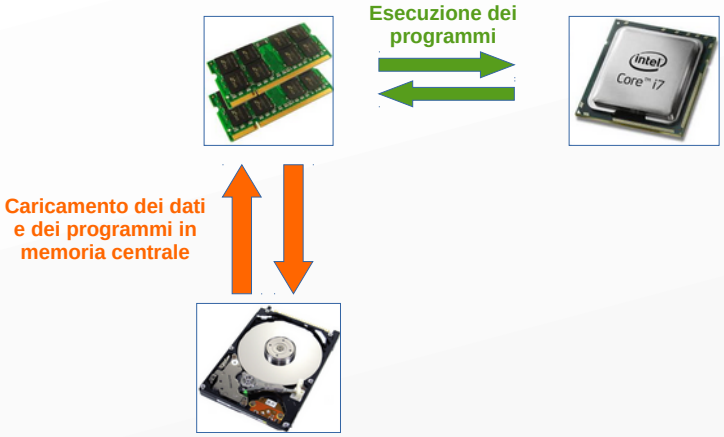
SISD

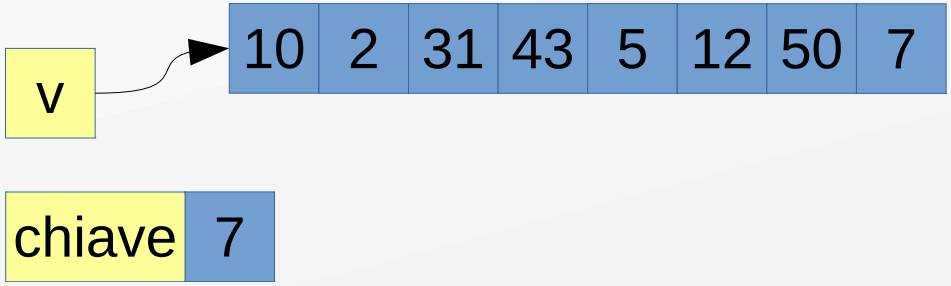


Esempio: ricerca di un elemento

```
for(int i = 0; i < N; i++)
    if(v[i] == chiave) return i;
```

SISD

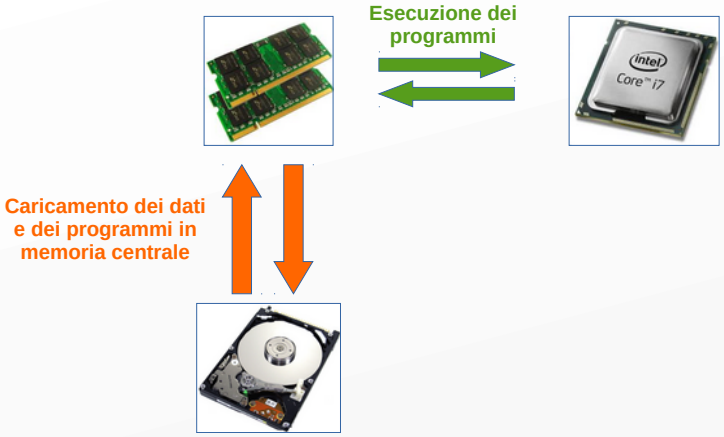


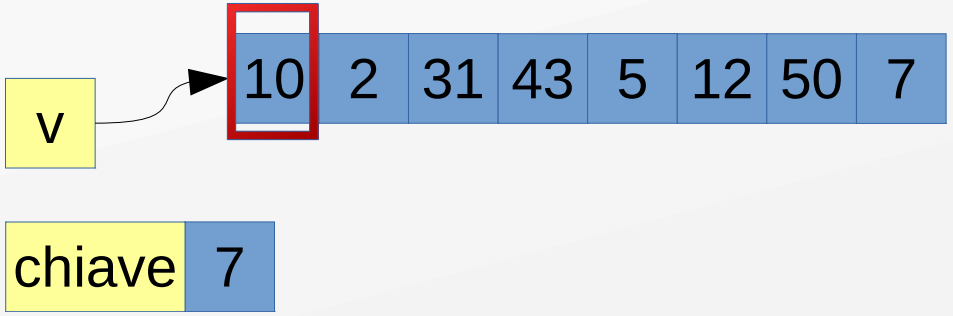


Esempio: ricerca di un elemento

```
for(int i = 0; i < N; i++)
    if(v[i] == chiave) return i;
```

SISD



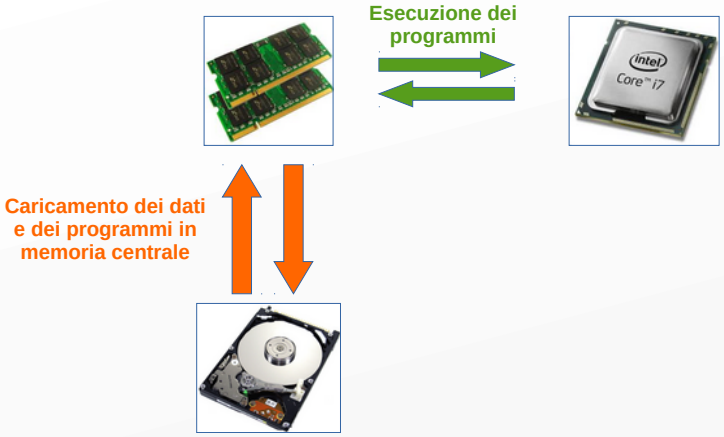


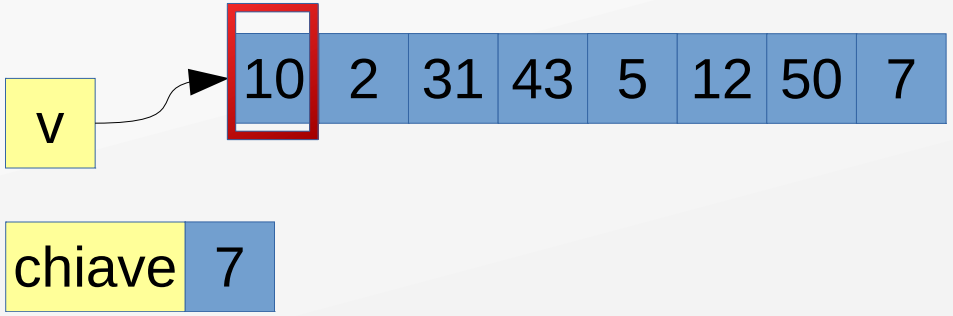
Esempio: ricerca di un elemento

```
for(int i = 0; i < N; i++)
    if(v[i] == chiave) return i;
```

chiave 7

SISD



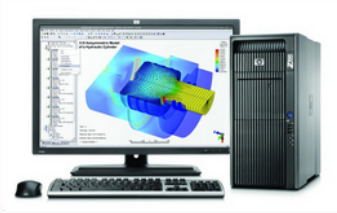


Esempio: ricerca di un elemento

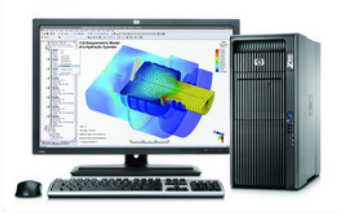
```
for(int i = 0; i < N; i++)
    if(v[i] == chiave) return i;
```

chiave 7

SIMD

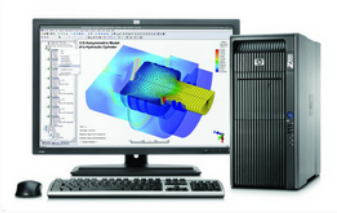


SIMD



Esempio; ricerca di un elemento

```
if(v[i] == chiave) return i;
```



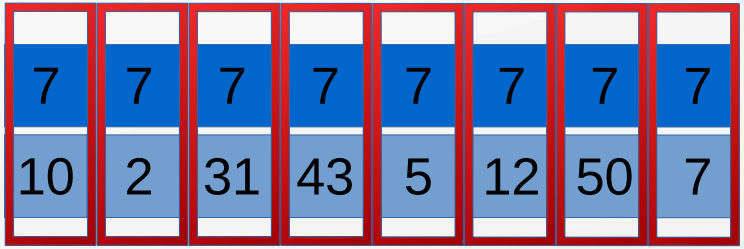
7	7	7	7	7	7	7	7
10	2	31	43	5	12	50	7

Esempio; ricerca di un elemento

```
if(v[i] == chiave) return i;
```

chiave	7

SIMD



Esempio; ricerca di un elemento

```
if(v[i] == chiave) return i;
```

chiave 7

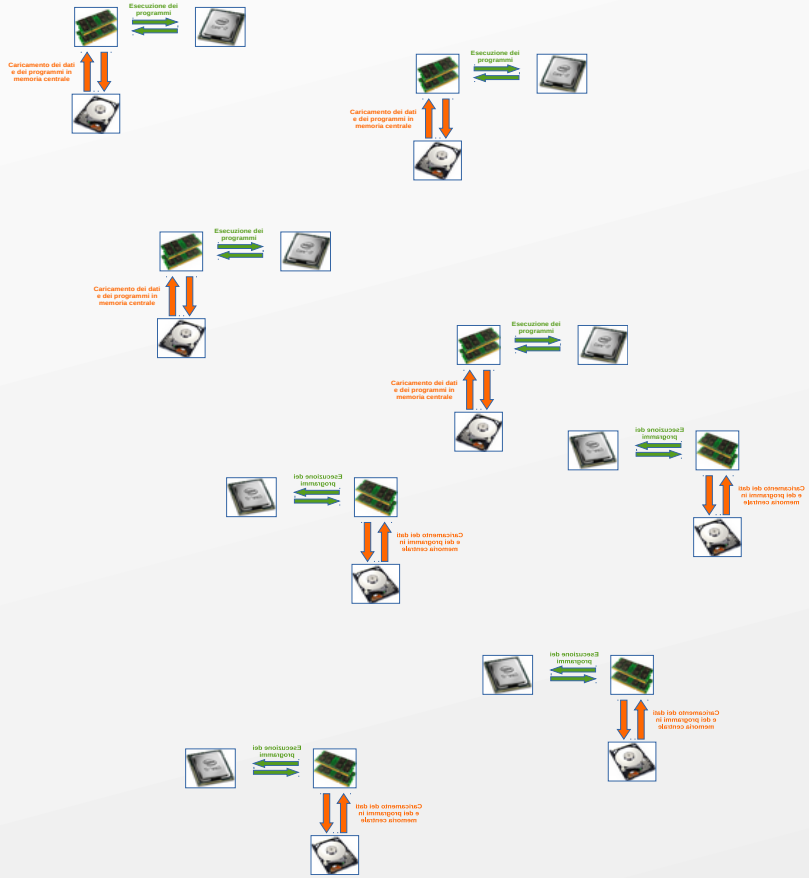
MIMD

MIMD

Esempio: Supercomputing

- Ogni processore ha una propria memoria ed esegue codice indipendente
- I processori vengono “affittati” per svolgere contemporaneamente diversi tipi di calcolo
- I processori possono cooperare per portare a termine le operazioni

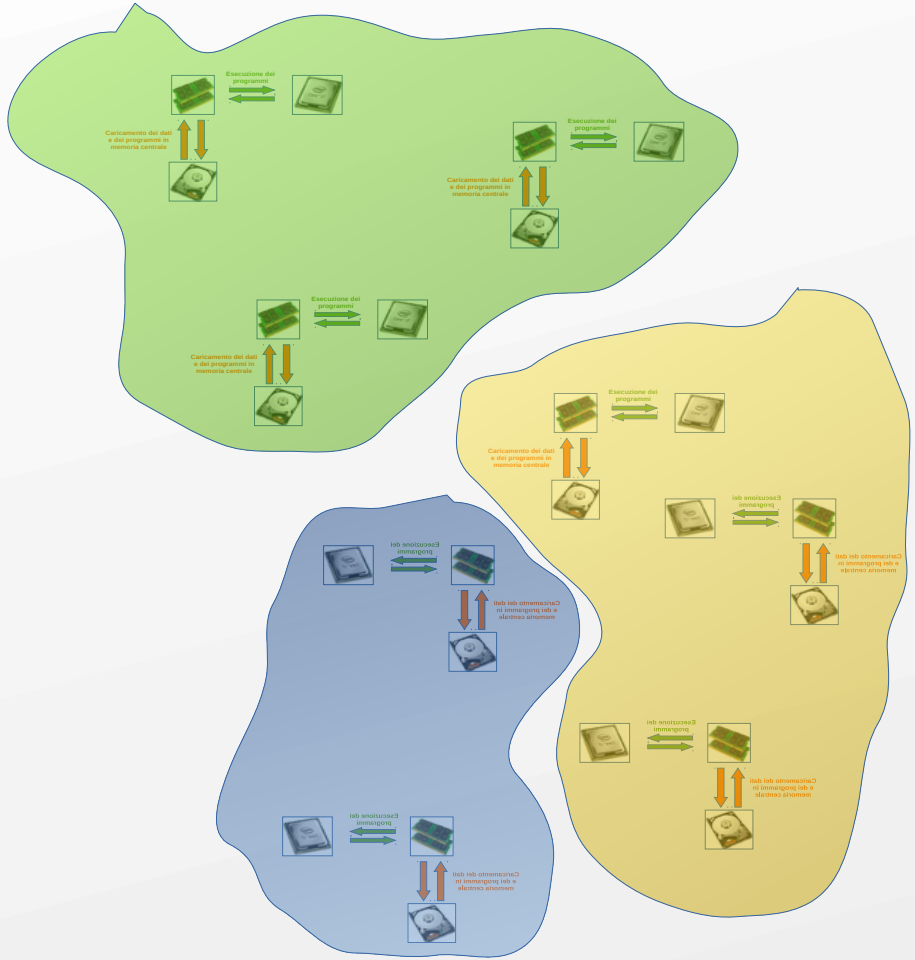
MIMD



Esempio: Supercomputing

- Ogni processore ha una propria memoria ed esegue codice indipendente
- I processori vengono “affittati” per svolgere contemporaneamente diversi tipi di calcolo
- I processori possono cooperare per portare a termine le operazioni

MIMD

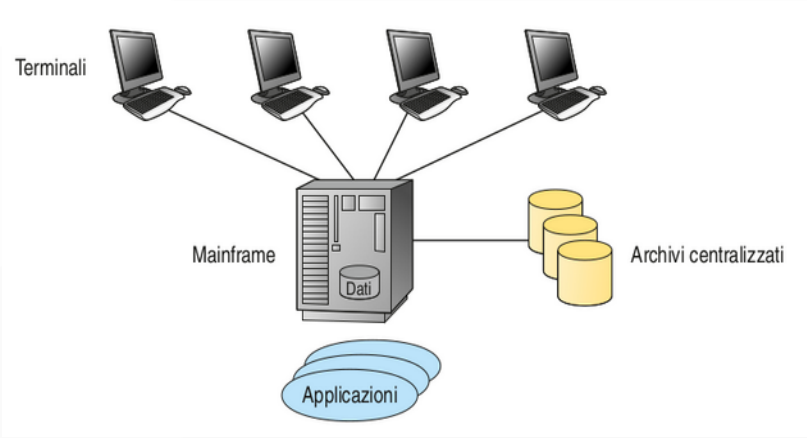


Esempio: Supercomputing

- Ogni processore ha una propria memoria ed esegue codice indipendente
- I processori vengono “affittati” per svolgere contemporaneamente diversi tipi di calcolo
- I processori possono cooperare per portare a termine le operazioni

Architetture distribuite software

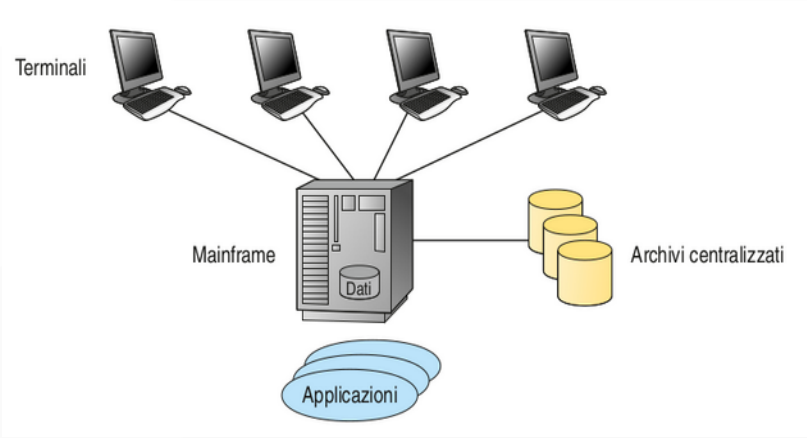
Architetture distribuite software

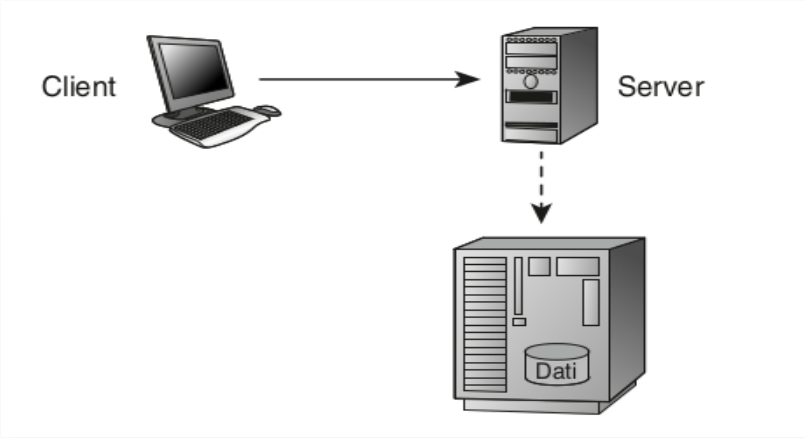


Terminali remoti

I terminali non hanno potenza di calcolo, servono solo per interfacciarsi alla macchina principale (Mainframe)

Architetture distribuite software





Terminali remoti

I terminali non hanno potenza di calcolo, servono solo per interfacciarsi alla macchina principale (Mainframe)

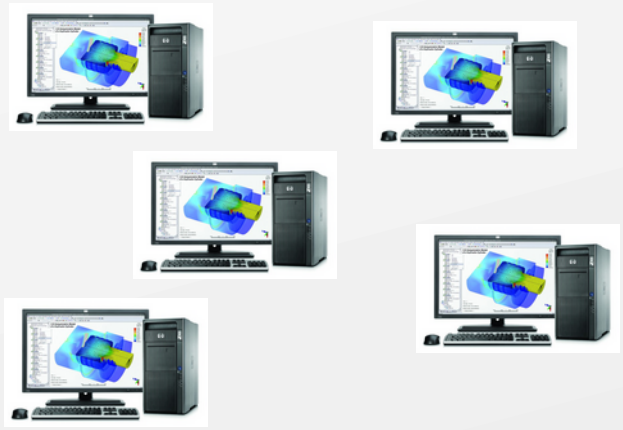
Client – Server

I terminali hanno potenza di calcolo ma cooperano con una entità intermedia (Server)

Architetture distribuite software

Cooperativa

Si applicano i principi della programmazione orientata agli oggetti. Si standardizzano le modalità di richiesta/fornitura di un servizio in modo da poter utilizzare lo stesso framework su piattaforme diverse.



Architetture distribuite software

WEB-centric

È un ritorno al passato. Terminali con poca potenza di calcolo (smartphone) richiedono servizi che vengono svolti sul server (esempio: riconoscimento vocale)

Cooperativa

Si applicano i principi della programmazione orientata agli oggetti. Si standardizzano le modalità di richiesta/fornitura di un servizio in modo da poter utilizzare lo stesso framework su piattaforme diverse.

Architetture distribuite software

Distribuita

Ogni macchina agisce in maniera indipendente e funge sia da fornitore che da richiedente di servizi (esempio: P2P)

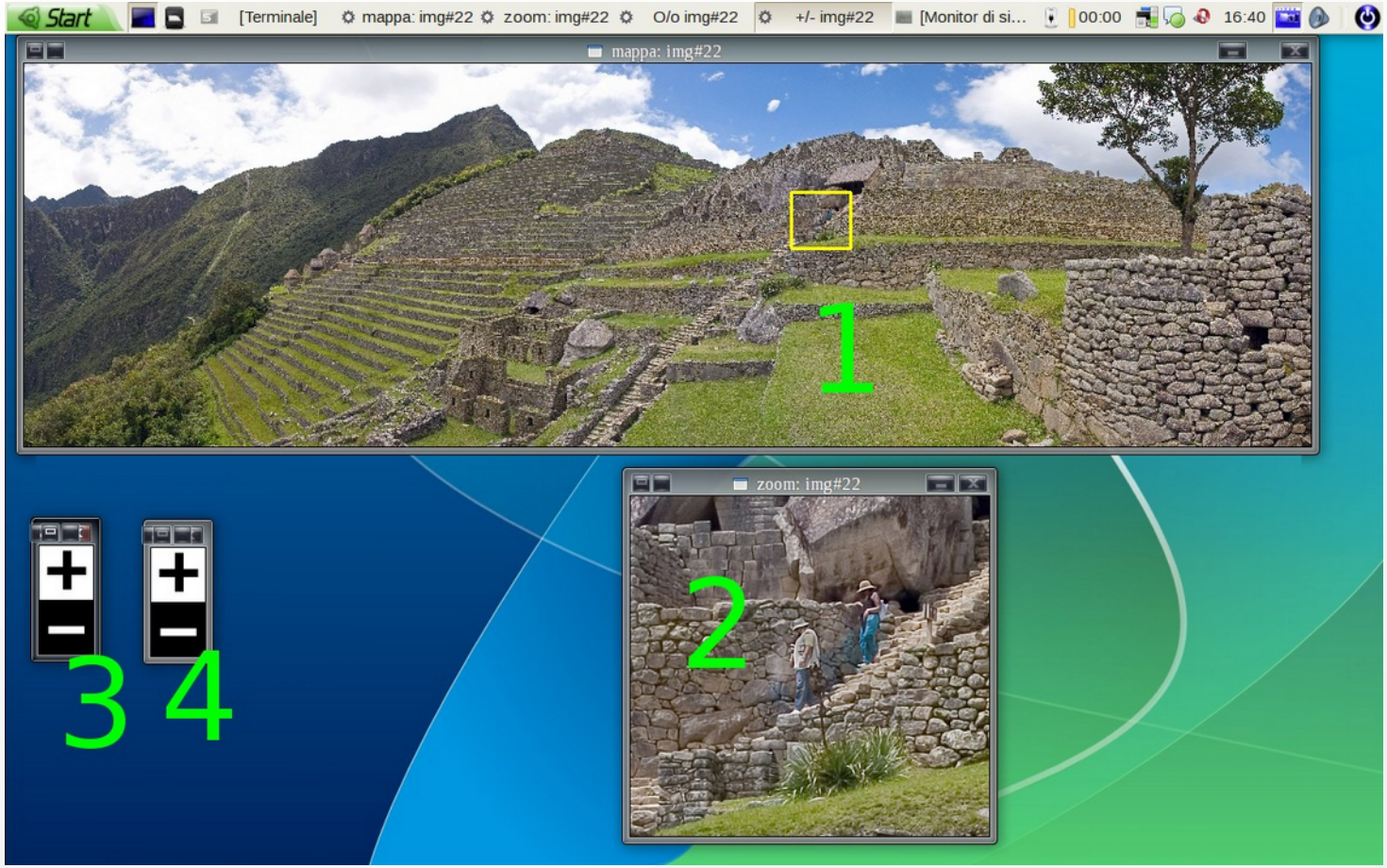
WEB-centric

È un ritorno al passato. Terminali con poca potenza di calcolo (smartphone) richiedono servizi che vengono svolti sul server (esempio: riconoscimento vocale)

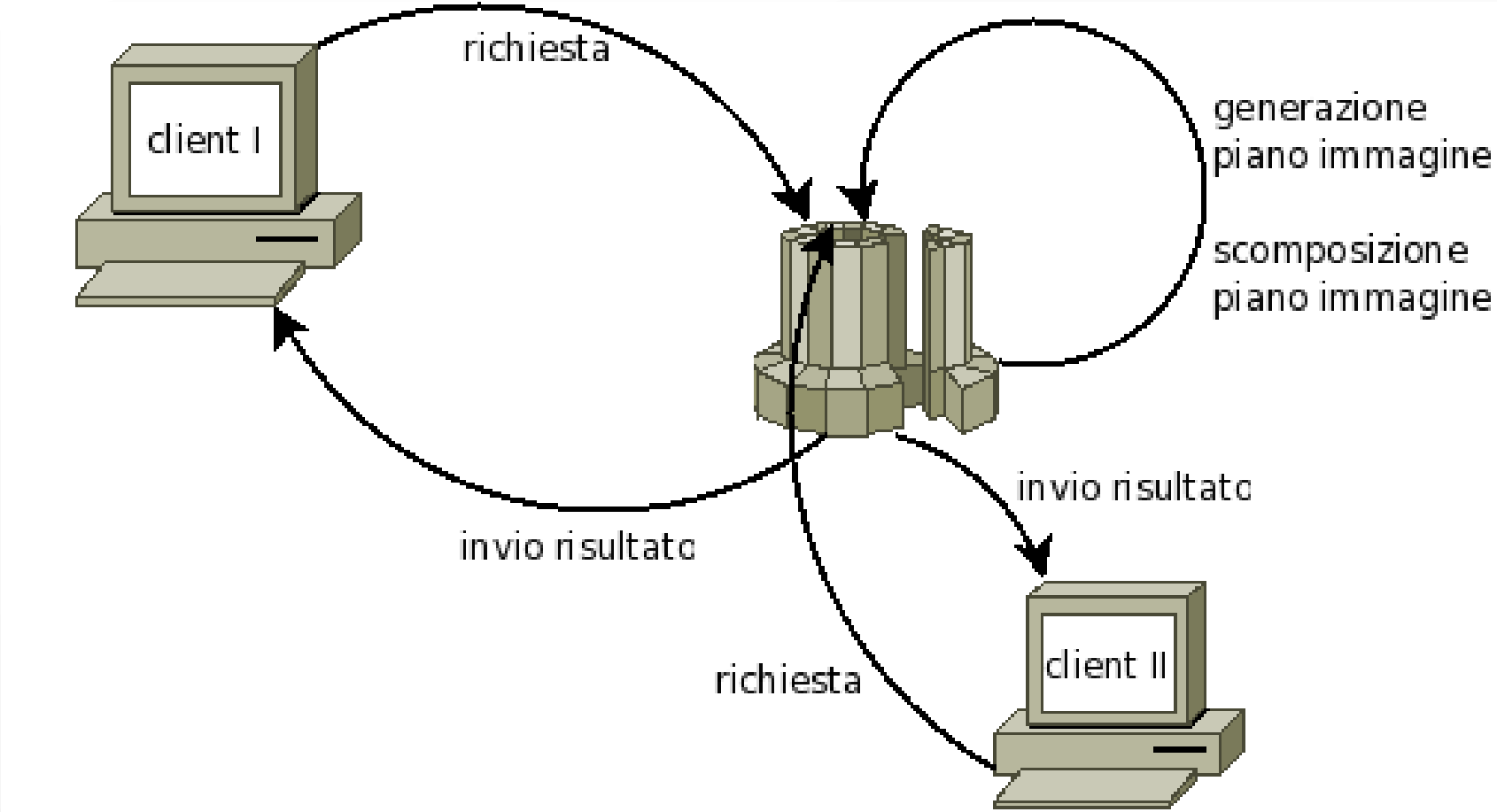
Cooperativa

Si applicano i principi della programmazione orientata agli oggetti. Si standardizzano le modalità di richiesta/fornitura di un servizio in modo da poter utilizzare lo stesso framework su piattaforme diverse.

Esempio applicazione distribuita



Esempio applicazione distribuita



Esempio codice distribuito (programmazione del server)

```
#include "../headers/mioSocketServer.h"
#include "../../common.h"

int main(int argc, char** argv)
{
    MioSocketServer mss(10000);

    if(argv[1] != NULL)
    {   printf("[scomposizione immagine %s...]\n", argv[1]);
        printf("******************************\n");
        int o = NUM_ORIZZONTALI;
        int v = NUM_VERTICALI;
        int p = NUM_STEP_ZOOM;
        int l = NUM_LIVELLI_ZOOM;
        mss.separaImmagine(argv[1], o/*orizzontale*/, v/*verticale*/, l/*livelli*/, p/*passo*/);

        printf("******************************\n");
        printf("[fatto]\n");

    }
    printf("[avvio server...");

    mss.connetti();
    mss.ascolta();
    printf("fatto.]\n");
    mss.esegui();
}
```

Esempio codice distribuito (programmazione del server)

```
#include "../headers/mioSocketServer.h"
#include "../../common.h"

int main(int argc, char** argv)
{
    MioSocketServer mss(1

    if(argv[1] != NULL)
    {   printf("[scomposi
        printf("*********
        int o = NUM_ORIZZ
        int v = NUM_VERTI
        int p = NUM_STEP_
        int l = NUM_LIVEL
        mss.separaImmagir

        printf("*********
        printf("[fatto]\r

    }
    printf("[avvio server

    mss.connetti();
    mss.ascolta();
    printf("fatto.]\n");
    mss.esegui();
}
```

```
int MioSocketServer::esegui()
{
    int connessioni = 0;
    while (1)
    {
        printf("in attesa di richiesta... ");
        unsigned int clientlen = sizeof(echoclient);

        /* Wait for client connection */
        if ((clientsock = accept(serversock, (struct sockaddr *) &echoclient, &clientlen)) < 0)
        {
            printf("Failed to accept client connection [%i] \n", clientsock);
            return -1;
        }

        /* Gestione delle richieste*/
        if(clientsock > 0)
        {
            connessioni++;
            #ifdef linux
                ScambioFileSendFile(clientsock);
            #endif

            #ifdef _WIN32
                ScambioFile(clientsock);
            #endif

        }
        close(clientsock);
    }
    return 0;
}
```

Esempio codice distribuito (programmazione del server)

```
#include "../headers/mioSocketServer.h"
#include "../../common.h"

int main(int argc, char** argv)
{
    MioSocketServer mss(1

    if(argv[1] != NULL)
    {   printf("[scomposi
        printf("*********
        int o = NUM_ORIZZ
        int v = NUM_VERTI
        int p = NUM_STEP_
        int l = NUM_LIVEL
        mss.separaImmagir

        printf("*********
        printf("[fatto]\r

    }
    printf("[avvio server

    mss.connetti();
    mss.ascolta();
    printf("fatto.]\n");
    mss.esegui();
}
```

```
int MioSocketServer::esegui()
{
    int connessioni = 0;
    while (1)
    {
        printf("in attesa di richiesta... ");
        unsigned int clientlen = sizeof(echoclient);

        /* Wait for client connection */
        if ((clientsock = accept(serversock, (struct sockaddr *)
        {
            printf("Failed to accept client connection [%i] \n",
            return -1;
        }

        /* Gestione delle richieste*/
        if(clientsock > 0)
        {
            connessioni++;
            #ifdef linux
                ScambioFileSendFile(clientsock);
            #endif

            #ifdef _WIN32
                ScambioFile(clientsock);
            #endif

        }
        close(clientsock);
    }
    return 0;
}
```

```
    // allocate memory to contain the whole file:
    buffer = new char [lSize];
    if (buffer == NULL)
    {   fputs ("Memory error",stderr);
        exit (2);
    }

    // copy the whole file into the buffer:
    result = fread (buffer,1,lSize,pFile);

    //send file content to client byte per byte
    int s = 0;
    int step = 1024;
    int tot = 0;
    while(tot!=lSize)
    {
         s = send(sock, &buffer[tot], step, 0);
         tot += s;
    }/*
    for(unsigned int i = 0; i < lSize; i+=step)
    {   send(sock, &buffer[i], step, 0);
        s++;
    }*/
    fclose(pFile);
    //printf("inviati : %i bytes\n", s);
    if (result != lSize)
    {   fputs ("Reading error",stderr);
        exit (3);
    }

    delete[] buffer;
}
```

Esempio codice distribuito (programmazione del client)

```
#include "../headers/visualizzatore.h"
#include "../../common.h"

int main(int argc, char** argv)
{   printf("visualizzatore...\n");

    Visualizzatore v(argv[1], NUM ORIZZONTALI/*oriz*/, NUM VERTICALI/*vert*/, NUM LIVELLI ZOOM/*numliv*/, 3);
    printf("visuaizzatore creato\n");
    v.avvia();
    printf("visuaizzatore avviato\n");
    v.ferma();
    printf("visuaizzatore fermato\n");

}
```

Esempio codice distribuito (programmazione del client)

```
#include "../headers/visualizzatore.h"
#include "../../common.h"

int main(int argc, char** argv)
{   printf("visualizzatore...\n");

    Visualizzatore v(a
    printf("visuaizza
    v.avvia();
    printf("visuaizza
    v.ferma();
    printf("visuaizza

}
```

```
void Visualizzatore:: avvia()
{
    Visualizzatore *v = this;
    int X = 50;

    cvNamedWindow(titoloFinestraMappa); cvMoveWindow(titoloFinestraMappa, X, 10);
    cvNamedWindow(titoloFinestraZoom); cvMoveWindow(titoloFinestraZoom, X , 10);
    cvNamedWindow(titoloControlloROICostante); cvMoveWindow(titoloControlloROICostante, X + 60, mappa->height + 100);
    cvNamedWindow(titoloControlloAreaCostante); cvMoveWindow(titoloControlloAreaCostante, X, mappa->height + 100);

    cvSetMouseCallback(titoloFinestraMappa,selezionaPorzioneMultiImage, v);         //gestione della selezione della roi sulla mappa di navi
    cvSetMouseCallback(titoloFinestraZoom,spostaPorzioneMultiImage, v);             //gestione dello spostamento della roi sull'immagine ing
    cvSetMouseCallback(titoloControlloROICostante,zoomMultiImageRoiCostante, v);    //gestione dei livelli di zoom
    cvSetMouseCallback(titoloControlloAreaCostante,zoomMultiImageAreaCostante, v);  //gestione dei livelli di zoom

    char controlloZoom[255];
    strcpy (controlloZoom, cartellaImmaginiClient);
    strcat (controlloZoom, "controlloZoom.bmp");
    IplImage *controllo = cvLoadImage(controlloZoom);

    char controlloZoom2[255];
    strcpy (controlloZoom2, cartellaImmaginiClient);
    strcat (controlloZoom2, "controlloZoom2.bmp");
    IplImage *controllo2 = cvLoadImage(controlloZoom2);

    cvShowImage(titoloControlloAreaCostante, controllo);
    cvShowImage(titoloControlloROICostante, controllo2);
    cvShowImage(titoloFinestraMappa, mappa);

    cvReleaseImage(&controllo);
    while( cvWaitKey() != 'q'){};
}
```

Esempio codice distribuito (programmazione del client)

```
#include "../headers/visualizzatore.h"
#include "../../common.h"

int main(int argc, char** argv)
{   printf("visualizzatore...\n");

    Visualizzatore v(a
    printf("visuaizza
    v.avvia();
    printf("visuaizza
    v.ferma();
    printf("visuaizza

}
```

```
void Visualizzatore:: avvia()
{
    Visualizzatore *v = this;
    int X = 50;

    cvNamedWindow(titoloFinestraMappa); cvMoveWindow(titoloFinestraMappa, X, 10);
    cvNamedWindow(titoloFinestraZoom); cvMoveWindow(titoloFinestraZoom, X , 10);
    cvNamedWindow(titoloControlloROICostante); cvMoveWindow(titoloControlloROICostante, X + 60, mappa->height + 100);
    cvNamedWindow(titoloControlloAreaCostante); cvMoveWindow(titoloControlloAreaCostante, X, mappa->height + 100);

    cvSetMouseCallback(titoloFinestraMappa,selezionaPorzioneMultiImage, v);         //gestione della selezione della roi sulla mappa di navi
    cvSetMouseCallback(titoloFinestraZoom,spostaPorzioneMultiImage, v);             //gestione dello spostamento della roi sull'immagine ing
    cvSetMouseCallback(titoloControlloROICostante,zoomMultiImageRoiCostante, v);    //gestione dei livelli di zoom
    cvSetMouseCallback(titoloControlloAreaCostante,zoomMultiImageAreaCostante, v); //gestione dei livelli di zoom

    char controlloZoom[255];
    strcpy (controlloZoom, cartellaImmaginiClient);
    strcat (controlloZoom, "controlloZoom.bmp");
    IplImage *controllo = cvLoadImage(controlloZoom);

    char controlloZoom2[255];
    strcpy (controlloZoom2, cartellaImmaginiClient);
    strcat (controlloZoom2, "controlloZoom2.bmp");
    IplImage *controllo2 = cvLoadImage(controlloZoom2);

    cvShowImage(titoloControlloAreaCostante, controllo);
    cvShowImage(titoloControlloROICostante, controllo2);
    cvShowImage(titoloFinestraMappa, mappa);

    cvReleaseImage(&controllo);
    while( cvWaitKey() != 'q'){};
}
```

```
//richiesta file al server
MioSocketClient msc(indirizzoServer, porta);
msc.connetti();
msc.scambioFile(nome, nomeFileClient);
msc.disconnetti();

//caricamento immagine locale
temp = cvLoadImage(nomeFileClient);
```

Esempio codice distribuito

```
int MioSocketServer::esegui()
{
    int connessioni = 0;
    while (1)
    {
        printf("in attesa di richiesta... ");
        unsigned int clientlen = sizeof(echoclient);

        /* Wait for client connection */
        if ((clientsock = accept(serversock, (struct sockaddr *) &echoclient, &clientlen)) < 0)
        {
            printf("Failed to accept client connection [%i] \n", clientsock);
            return -1;
        }

        /* Gestione delle richieste*/
        if(clientsock > 0)
        {
            connessioni++;
            #ifdef linux
                ScambioFileSendFile(clientsock);
            #endif

            #ifdef _WIN32
                ScambioFile(clientsock);
            #endif

        }
        close(clientsock);
    }
    return 0;
}
```

```
//richiesta file al server
MioSocketClient msc(indirizzoServer, porta);
msc.connetti();
msc.scambioFile(nome, nomeFileClient);
msc.disconnetti();

//caricamento immagine locale
temp = cvLoadImage(nomeFileClient);
```

Esempio codice distribuito

```
int MioSocketClient::scambioFile(const char* nomeFileRichiesto, const  char* nomeFileLocale)
{
    char nome[512];
    strcpy(nome, nomeFileRichiesto);

    //send filename
    int scambiati = send(sock, nome, 512, 0);

    //receive file dimensions
    char dimensione[8];
    scambiati = recv(sock, dimensione, 8, 0);
    long int dim = *((long int*)dimensione);

    /*ricezione file [funzionante]*/
    //allocate buffer
    char* file = new char[dim];

    //riceve l'intero file in memoria
    int totale = 0;
    while(totale != dim)
    {
        totale += recv(sock, &file[totale], CHUNK, 0);
        //printf("ricevuto %i di %i\n", totale, dim);
    }

    //scrive l'intero file su disco
    FILE* pFile = fopen ( nomeFileLocale , "wb" );
    fwrite (file , 1 , totale , pFile );
    fclose (pFile);
    delete[] file;

    /************************* */
    return 1;
```

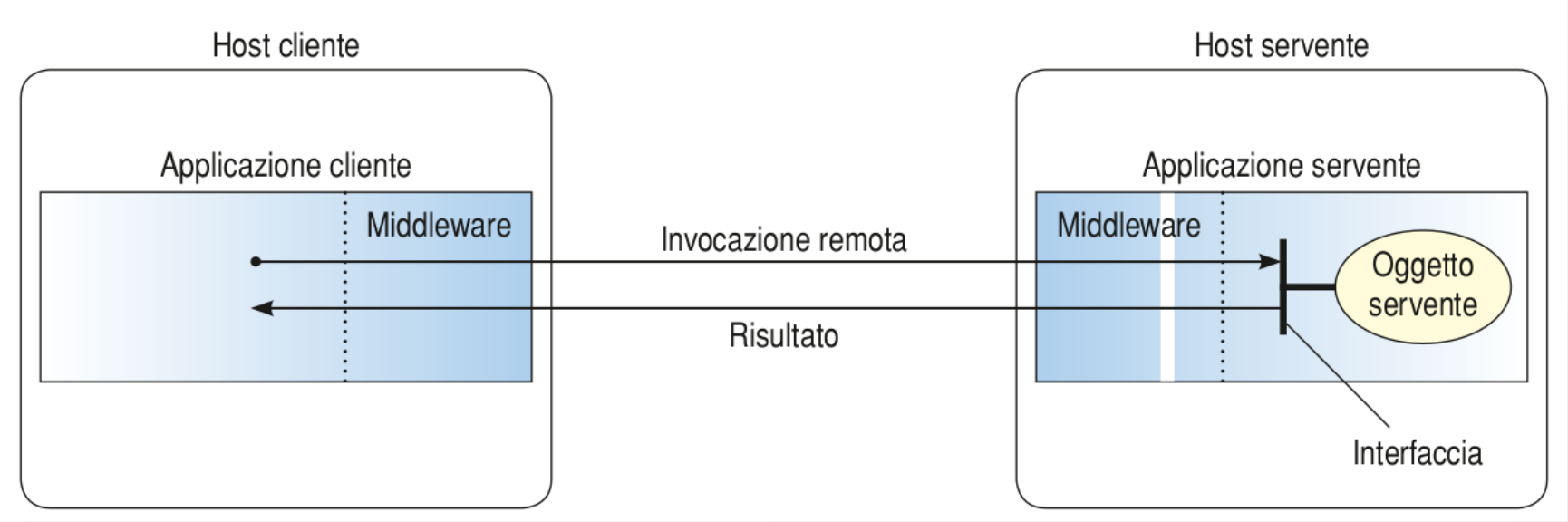
```
void MioSocketServer::ScambioFile(int sock)
{
    char nome[512];

    //wait for file name
    recv(sock, nome, 512, 0);

    //try to load the file
    FILE * pFile;
    unsigned long lSize;
    char * buffer;
    size_t result;
    char file[512];

    //nella versione definitiva bisogna implementare il collegamento ad un database
    //per recuperare il file a partire da una chiave di ricerca
    strcpy(file, nome);
    pFile = fopen ( file , "rb" );
    if (pFile==NULL)
    {   printf("\nil file [%s] non esiste\n", nome);
        exit (1);
    }
    printf("\nrichiesto ed inviato\n %s", nome);
    printf("\n");
    // obtain file size:
    fseek (pFile , 0 , SEEK_END);
    lSize = ftell (pFile);
    rewind (pFile);

    //send file dimension to client
    char* lSizeChar = (char*)&lSize;
    send(sock, lSizeChar, 4, 0);
```

Modello Client/Server

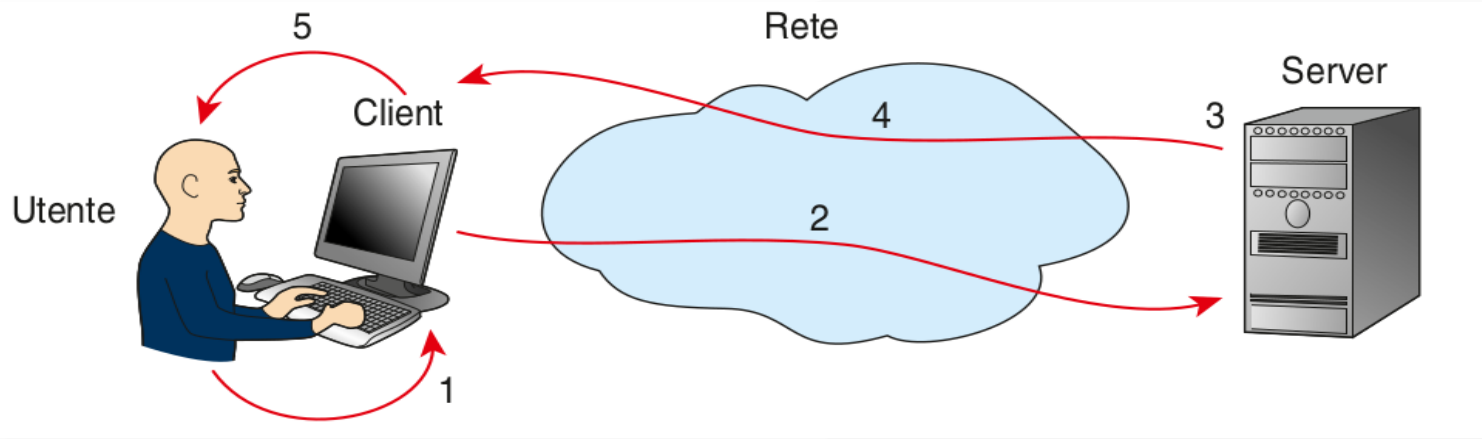
Separazione fra l'entità che richiede un servizio e l'entità che fornisce un servizio

È una separazione logica, riguarda cioè i processi più che le macchine

La stessa macchina può ospitare processi server e processi client



Modello Client/Server



Schema standard del modello client/server

(1)Richiesta del client

(2)Trasmissione della richiesta via rete

(3)Elaborazione della richiesta sul server

(4)Trasmissione della risposta via rete

(5)Elaborazione della risposta sul client

Le basi comuni delle architetture distribuite

La comunicazione fra due processi residenti su macchine diverse può avvenire solo se

1. Le due macchine sono connesse da un canale di comunicazione fisico (rete di comunicazione)
2. Su questo canale fisico è possibile identificare i due processi in maniera univoca
3. Esiste uno standard per lo scambio dei dati

Le basi comuni delle architetture distribuite – SOCKET

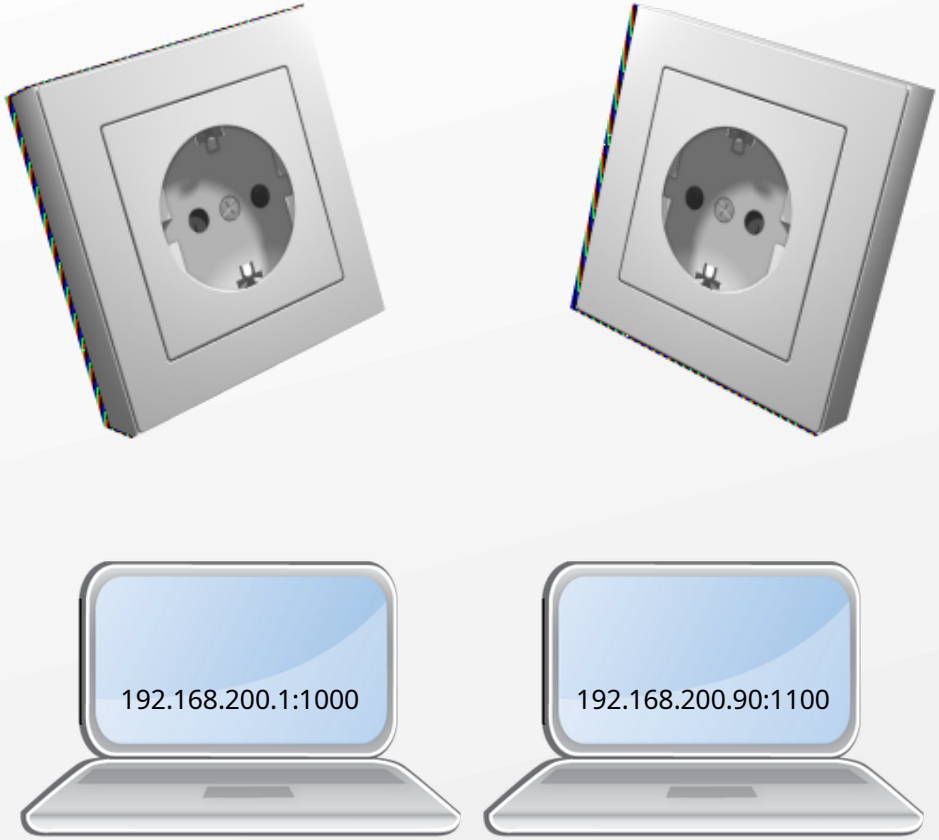
Un *socket* è un terminale di un canale di comunicazione.

Letteralmente un socket è una piastrina a cui possiamo collegare un componente.

Nei sistemi elettrici, un socket è la presa della corrente: è il terminale della rete elettrica che porta l'energia dalla centrale al dispositivo che vogliamo alimentare.

Nei sistemi informatici, un socket è una coppia <IP, PORTA>: è l'identificatore della rete che permette ad un processo di comunicare

Occhio: Ogni processo definisce un proprio socket. L'indirizzo IP è quello della macchina dove è attivo il processo, la porta viene decisa dal processo.



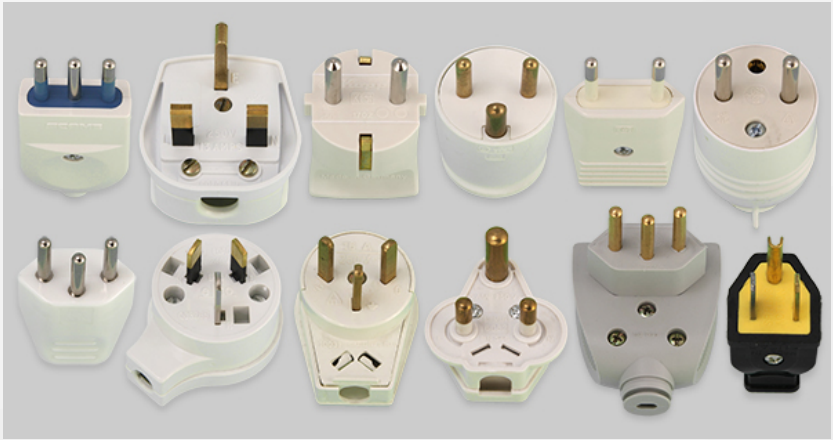
Le basi comuni delle architetture distribuite – SOCKET

I passi per impostare un socket sono i seguenti

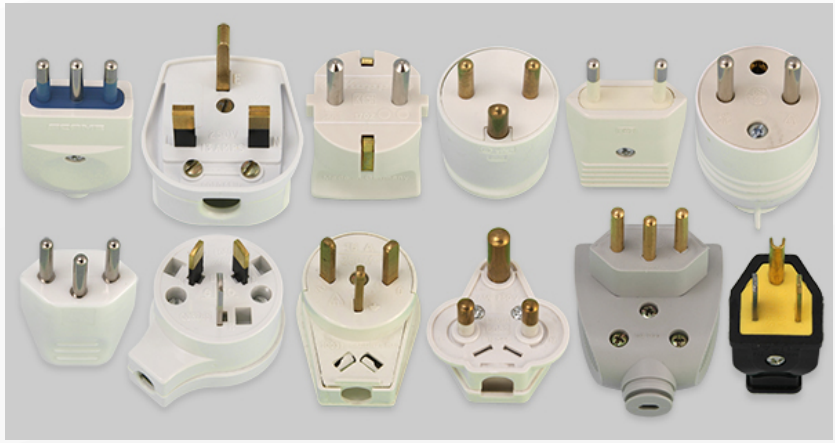
- Creazione del socket
- Collegamento del socket
- Ascolto

Una volta che il socket è impostato, è pronto a ricevere una richiesta di connessione. Quando questa viene effettuata

- La connessione viene accettata
- Può iniziare lo scambio dei messaggi e l'elaborazione



Le basi comuni delle architetture distribuite – SOCKET



Server

Il processo che vuole mettere a disposizione dei servizi decide che tipo di connessione utilizzare.

Una volta creato un socket, il processo server si mette in attesa delle connessione

Client

Il processo che vuole utilizzare dei servizi deve “adattarsi” a ciò che viene deciso dal server.

Per utilizzare un servizio, deve usare un socket compatibile con quello del server.

SOCKET e Linguaggi di programmazione: C/C++

```c
int main(int argc, char *argv[])
{
        int serversock;
        struct sockaddr_in server_socket;

        printf("Sto creando il socket...\n");
                serversock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        printf("fatto.\n");

        printf("Sto impostando i parametri...\n");
                memset(&server_socket, 0, sizeof(server_socket));
                server_socket.sin_family = AF_INET;
                server_socket.sin_addr.s_addr = htonl(INADDR_ANY);
                server_socket.sin_port = htons(50000);
        printf("fatto.\n");

        printf("Sto impostando il collegamento\n");
                bind(serversock, (struct sockaddr *) &server_socket, sizeof(server_socket));
        printf("fatto.\n");

        printf("Sono in attesa delle richieste dei client...\n");
                listen(serversock, 5);

        struct sockaddr_in client_socket;
        unsigned int clientlen = sizeof(client_socket);

        int clientsock = accept(serversock, (struct sockaddr *) &client_socket, &clientlen);

        printf("Connesso.\n");

        printf("Chiusura connessione ..\n");
                close(serversock);
        printf("fatto.\n");
}
```

```c
int main(int argc, char *argv[])
{
        int clientsock;
        struct sockaddr_in server_socket;

        printf("Sto creando il socket...\n");
                clientsock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        printf("fatto.\n");

        printf("Sto impostando i parametri...\n");
                memset(&server_socket, 0, sizeof(server_socket));
                server_socket.sin_family = AF_INET;
                server_socket.sin_addr.s_addr = inet_addr("127.0.0.1");
                server_socket.sin_port = htons(50000);
        printf("fatto.\n");

        printf("Connessione al server...\n");
                connect(clientsock, (struct sockaddr *) &server_socket, sizeof(server_socket));
        printf("fatto.\n");

        write(clientsock,"Hello Socket!",strlen("Hello Socket!"));

        printf("Chiusura connessione ..\n");
                close(clientsock);
        printf("fatto.\n");
}
```

Osservazioni

Nel codice non sono gestiti gli errori

Nel caso di programmazione con i socket ciò non deve accadere

Non c'è scambio di dati

L'esempio appena visto permette semplicemente di connettere due processi

Al client viene assegnato un identificatore

Quando la connessione viene accettata, il server associa un canale di comunicazione dedicato al client

Client e server hanno vita “limitata”

Una volta che il server ha fornito il servizio al client, la sua esecuzione termina e nessun altro client è in grado di connettersi

Un esempio con scambio dati

```c
int main(int argc, char *argv[])
{
        int serversock;
        struct sockaddr_in server_socket;

        printf("Sto creando il socket...\n");
                serversock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        printf("fatto.\n");

        printf("Sto impostando i parametri...\n");
                memset(&server_socket, 0, sizeof(server_socket));
                server_socket.sin_family = AF_INET;
                server_socket.sin_addr.s_addr = htonl(INADDR_ANY);
                server_socket.sin_port = htons(50000);
        printf("fatto.\n");

        printf("Sto impostando il collegamento\n");
                bind(serversock, (struct sockaddr *) &server_socket, sizeof(server_socket));
        printf("fatto.\n");

        printf("Sono in attesa delle richieste dei client...\n");
                listen(serversock, 5);

        struct sockaddr_in client_socket;
        unsigned int clientlen = sizeof(client_socket);

        int clientsock = accept(serversock, (struct sockaddr *) &client_socket, &clientlen);

        printf("Connesso.\n");

        char buffer[256];

        read(clientsock,buffer,255);
        printf("%s\n",buffer); printf("\n");
        write(clientsock,buffer,strlen(buffer));

        printf("Chiusura connessione...\n");
                close(serversock);
        printf("fatto.\n");
}
```

```c
int main(int argc, char *argv[])
{
        int clientsock;
        struct sockaddr_in server_socket;

        printf("Sto creando il socket...\n");
                clientsock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        printf("fatto.\n");

        printf("Sto impostando i parametri...\n");
                memset(&server_socket, 0, sizeof(server_socket));
                server_socket.sin_family = AF_INET;
                server_socket.sin_addr.s_addr = inet_addr("127.0.0.1");
                server_socket.sin_port = htons(50000);
        printf("fatto.\n");

        printf("Connessione al server...\n");
                connect(clientsock, (struct sockaddr *) &server_socket, sizeof(server_socket));
        printf("fatto.\n");

        write(clientsock,"Hello Socket!",strlen("Hello Socket!"));

        char buffer[256];
        read(clientsock,buffer,255);
        printf("%s\n",buffer);

        printf("Chiusura connessione...\n");
                close(clientsock);
        printf("fatto.\n");
}
```

Le funzioni per lo scambio dei dati

recv(socket, buffer, numero_byte, 0)

Si mette in attesa sul *socket* e riceve *numero_byte* dati

Nell'esempio che segue

- si dichiara un buffer di 1024 elementi
- si ricevono 3 byte di dati

```
char buffer[1024];
int effettivi = recv(clientsock,&buffer[0],1);
int effettivi = recv(clientsock,&buffer[1],1);
int effettivi = recv(clientsock,&buffer[2],1);
```

send(socket, buffer, numero_byte, 0)

Invia sul *socket numero_byte* dati

Nell'esempio che segue

- si dichiara un buffer di 1024 elementi
- si inviano i primi tre byte

```
char buffer[1024];
int effettivi = send(clientsock,&buffer[0],1);
int effettivi = send(clientsock,&buffer[1],1);
int effettivi = send(clientsock,&buffer[2],1);
```

I/O nei sistemi distribuiti: multithreading e sincronizzazione

Istruzioni non bloccanti

Il programma avvia l'operazione di I/O e prosegue nella sua elaborazione. In questo caso l'utente deve farsi carico di controllare che la ricezione o l'invio siano andati a buon fine

Esempio

Server	Client	
recv()		
send()		
fa_altro()		fai_altro()
check_point()	check_point()	

Istruzioni bloccanti

Il programma si arresta in attesa della ricezione/invio di un dato

Quando si è in presenza di istruzioni bloccanti bisogna fare attenzione alle problematiche di deadlock e starvation.

Esempio

Server	Client	
recv()		recv()
send()	send()	

Per ovviare a questo problema si possono utilizzare due thread distinti che si occupino rispettivamente di ricevere e inviare dati.

Osservazione

Per la visualizzazione dell'I/O distribuito su schermo, può non essere sufficiente utilizzare le funzioni di base (cin, cout, scanf, printf, Syste,out.println, etc...). Lo schermo diventa una risorsa condivisa e come tale bisogna gestirla. Ciò può avvenire in modo implicito (ad esempio in Java, usando le interfacce grafiche) o esplicito (ad esempio in C/C++ usando le librerie ncurses).

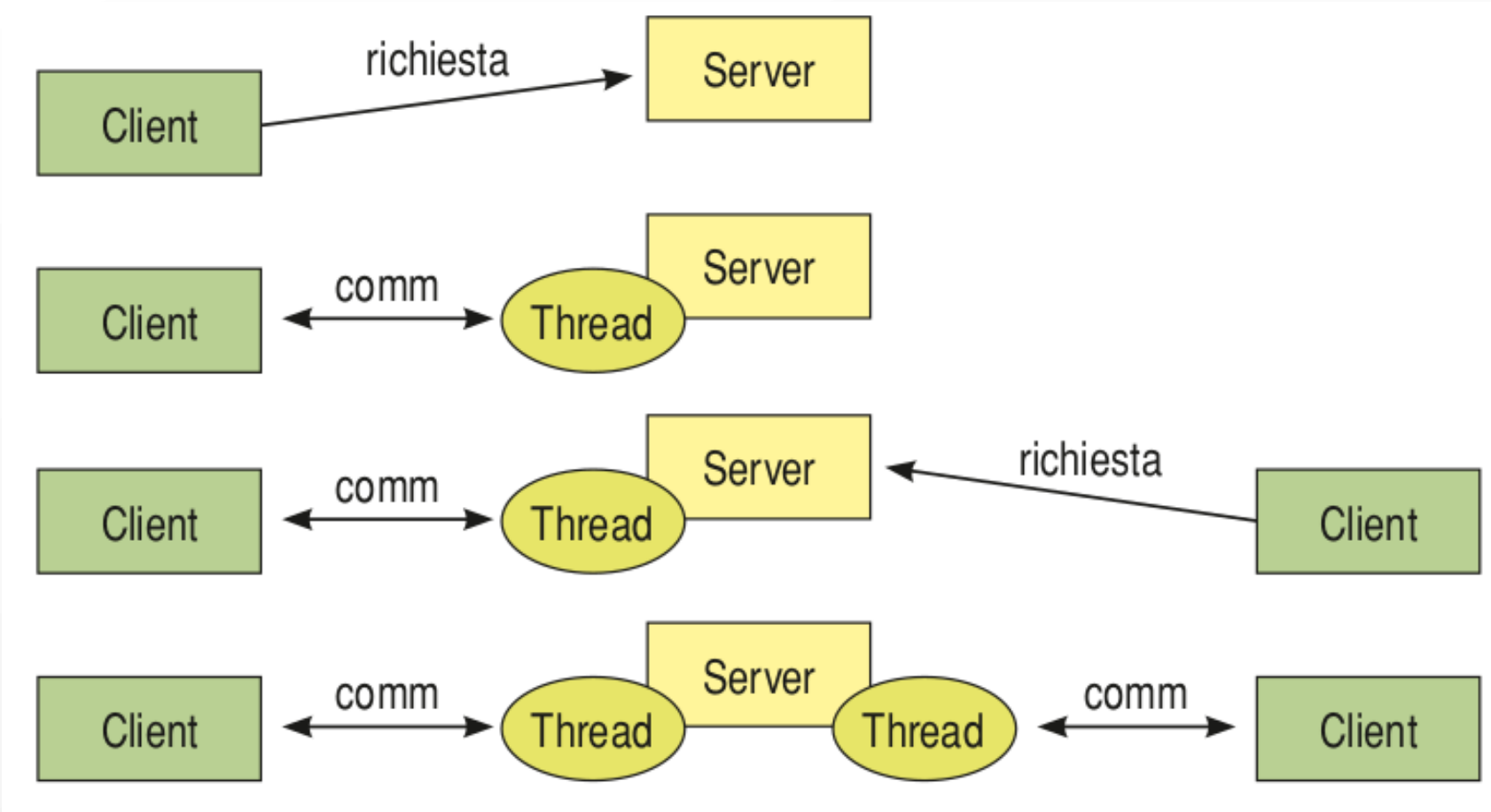
I/O nei sistemi distribuiti: Esempio di thread per la ricezione in C++

```
void *ricevi(void *threadid)
{


    while(1)
    {
        //riceve un stringa
        char buffer[512];
        int num caratteri = recv(sock , buffer , 511 , 0);
        buffer[num caratteri] = '\0';

        //ottiene la mutua esclusione e visualizza il messaggio sulla linea corrente
        pthread mutex lock(&mutexout);
            mvprintw(row++,1,"> %s",buffer);
            //riporta il cursore sulla linea di input
            move(LINES - 2,5);
            refresh();
        pthread mutex unlock(&mutexout);
    }
}
```

I/O nei sistemi distribuiti: Esempio di thread per l'invio in C++

```
void *invia(void *threadid)
{
        while(1)
        {
            char mesg[]="####";
            char str[512];

            //ottiene la mutua esclusione e visualizza il prompt
            pthread_mutex_lock(&mutexout);
                mvprintw(LINES - 2,0,"%s",mesg);
            pthread_mutex_unlock(&mutexout);

            //rimane in attesa dell'input
            getstr(str);
            str[strlen(str)] = '\0';

            //invia la stringa al server
            send(sock , str , strlen(str) , 0);

            //ottiene la mutua esclusione e visualizza il messaggio inviato sulla linea corrente
            pthread_mutex_lock(&mutexout);
                mvprintw(row++,1,"%s",str);
                move(LINES - 2,0);
                //cancella i caratteri inseriti precedentemente
                clrtoeol();
            pthread_mutex_unlock(&mutexout);
        }
}
```

Organizzazione del calcolo nei sistemi distribuiti



Esercizi

- **Modificare il client in modo che tenti di collegarsi indefinitamente al server interrogando la porta 50000**
- **Modificare il client in modo che il messaggio inviato al server venga introdotto da tastiera**
- **Modificare il server in modo che quando finisca l'interazione con un client sia in grado di accettare una nuova connessione**
- **Modificare il client in modo tale che sia possibile specificare indirizzo IP e numero di porta del server**
- **Modificare il server in modo tale che sia possibile decidere il numero di porta su cui mettersi in ascolto**
- **Modificare client e server in modo che si realizzi una chat a due in cui**
  - L'utente sulla macchina dove è attivo il server possa inviare messaggi da tastiera
  - L'utente sulla macchina dove è attivo il client possa inviare messaggi da tastiera
  - Il messaggio inviato dal client venga visualizzato su entrambe le macchine
  - Il messaggio inviato dal server venga visualizzato su entrambe le macchine
  - N.B: supporre che il primo invio venga effettuato dal client e che l'utente lato server risponda solo se gli viene inviato il messaggio
  - Osservazioni sulla sincronizzazione

Esercizio

Scrivere un server ed un client che simulino una rete ad anello

1)Il server è attivo su tutte le macchine alla porta 50000

2)Il messaggio è una stringa in cui compare l'indirizzo IP del destinatario, quello del mittente ed un messaggio di testo

3)Un client può inviare un messaggio solo alla macchina alla sua destra

4)Se la macchina che riceve il messaggio è la destinataria del messaggio, invia un messaggio di acknowledgement alla macchina mittente, secondo la regola 2, altrimenti passa il messaggio alla macchina alla sua destra

Esercizio

Scrivere un server ed un client che implementino un sistema di chat basato su server centrale

1) Il server è attivo su una singola macchina alla porta 50000
2) Il messaggio è una struttura in cui compare l'indirizzo IP del destinatario ed un messaggio di testo
3) Un client può inviare un messaggio solo al server
4) Quando un server riceve il messaggio lo invia alla macchina specificata nell'indirizzo IP

Esercizio

Scrivere un server che permetta di simulare un router

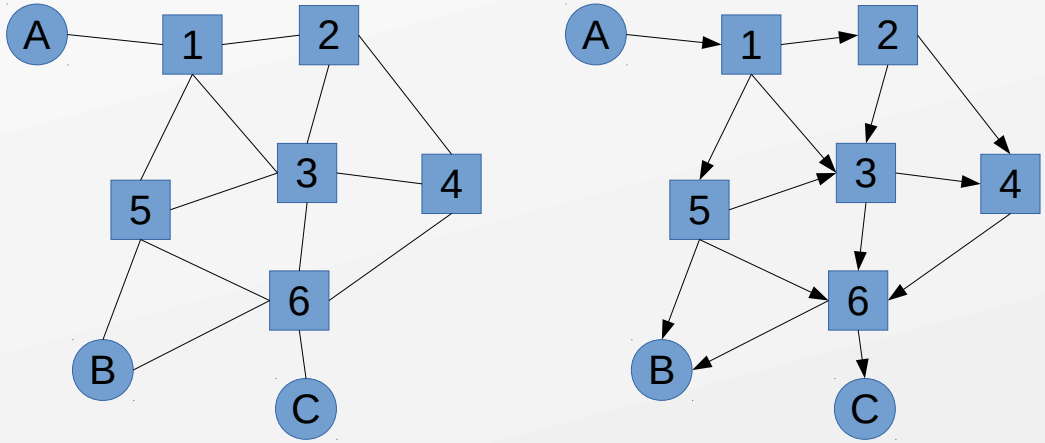
Il server è attivo su una singola macchina alla porta 50000

Il messaggio è una struttura in cui compare l'indirizzo IP del destinatario ed un messaggio di testo

Un client può inviare un messaggio a qualunque macchina

Quando una macchina riceve un messaggio lo reindirizza sulla base di una tabella di routing in essa memorizzata.

Si suppone che la rete sia organizzata mediante routing statico e che non sia applicato nessun algoritmo di ottimizzazione, come riportato nell'esempio sotto



Esercizio

Scrivere un client ed un server che permettano di inviare dati cifrati utilizzando un semplice algoritmo di crittografia

- Il client chiede in input una stringa e la codifica
- Invia la stringa al server che la decodifica e la rimanda indietro
- Se la decodifica è corretta il client visualizza un messaggio di conferma e continua lo scambio dei dati
- Altrimenti visualizza un messaggio di allarme e interrompe la connessione
- Esempio
  - G→ H, I→ J, O→ T, V → M, A→ 4, N → X, S → P, C→ 9, V→ R, E→ F, L→ B, ' ' → W, etc…
  - Client: giovanni scavello
  - Risposta del server HJTM4XXJWP94RFBBT
  - Client: ok, proseguo
  - Client: giovanni scavello
  - Risposta del server HJTM4JXJWP94RFBBT
  - Client: dati corrotti, interrompo il collegamento

Livelli e strati

Stackoverflow

originale su

http://stackoverflow.com/questions/120438/whats-the-difference-between-layers-and-tiers

Livelli

Si riferiscono alla separazione logica del codice.

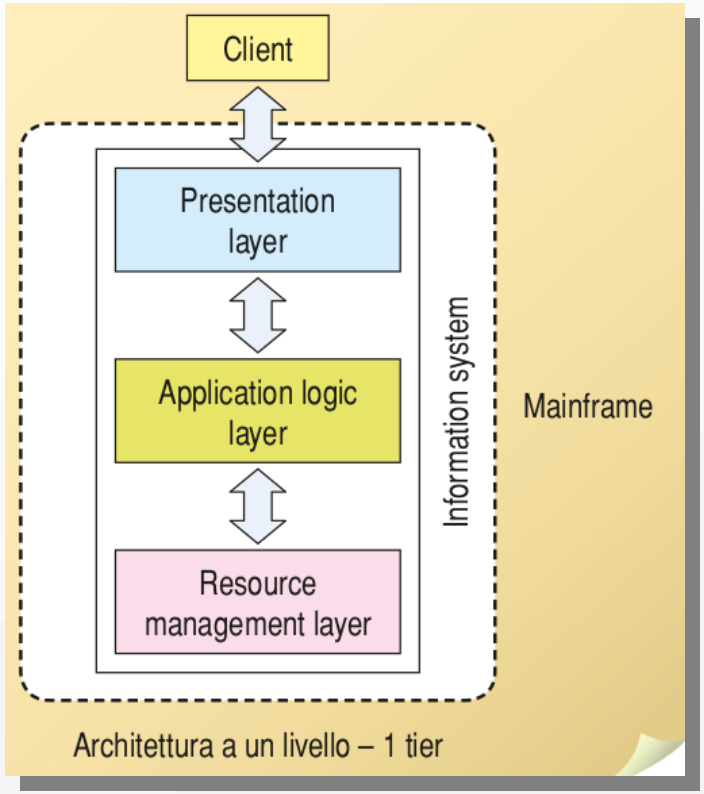
I livelli logici ci aiutano nell'oraganizzare meglio il codice. Ad esempio un'applicazione può avere i seguenti tre livelli: Presentazione – Logica applicativa – Accesso ai dati. I tre livelli costituiscono progetti a se stanti, possono consistere di 3 o più progetti (a seconda della complessità, ndt). Quando si compilano i progetti si ottengono i rispettivi codici oggetto (dll o so). In questo caso avremo 3 librerie dinamiche

Strati

Si riferiscono alla separazione fisica delle macchine.

(Nel caso di tre livelli, ndt) A seconda di come verrà implementata l'applicazione, si possono avere da uno a tre strati. Siccome abbiamo generato 3 librerie dinamiche, se eseguiamo il codice sulla stessa macchina, abbiamo un solo strato fisico ma tre livelli logici. Se invece decidiamo di eseguire le tre librerie su tre macchine diverse abbiamo tre livelli fisici e tre livelli logici

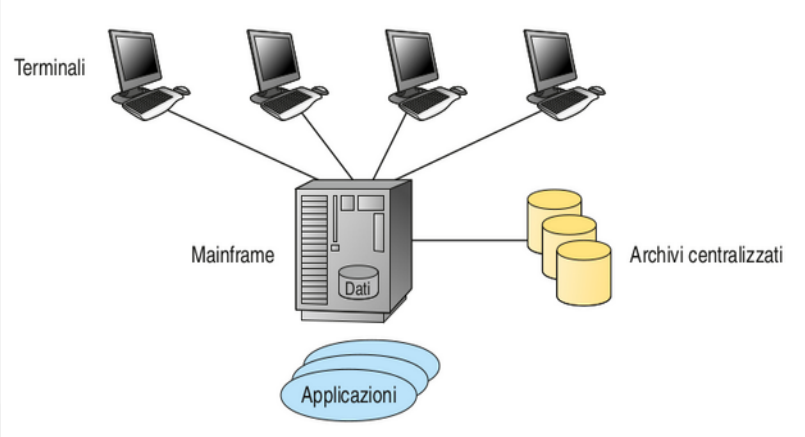
Livelli e strati



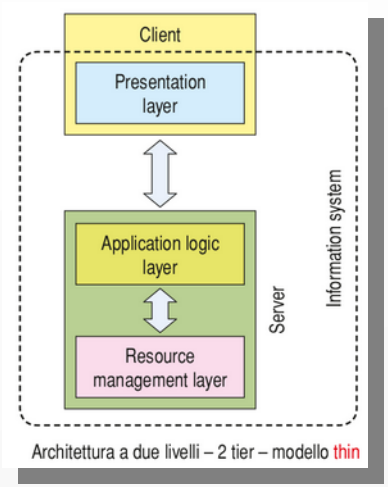
Architettura centralizzata

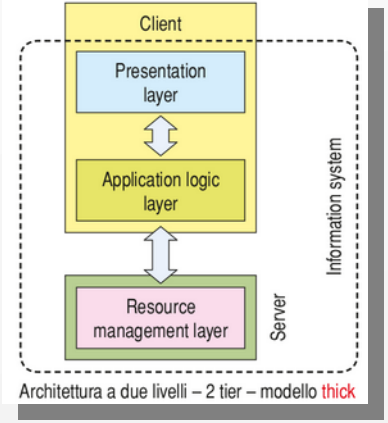
Il client funge da semplice visualizzatore, si suppone poca potenza di calcolo.

3 livelli → 1 strato



Livelli e strati



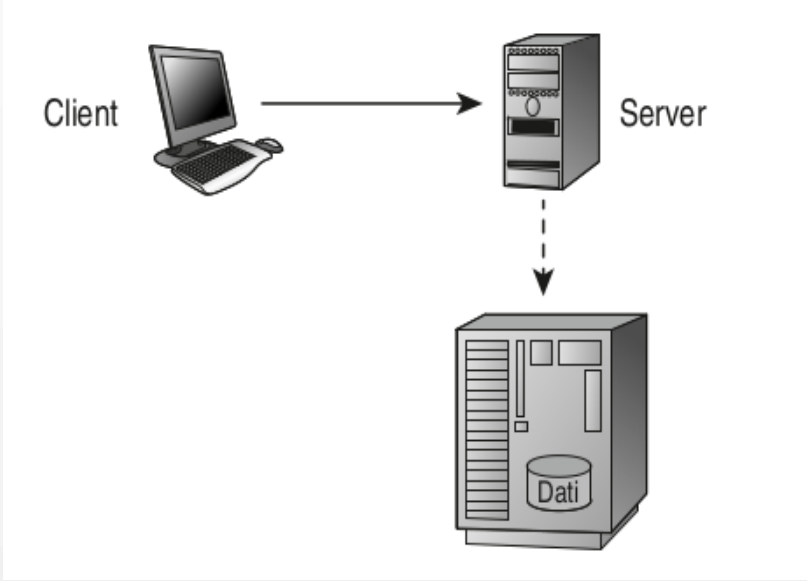


Architettura distribuita

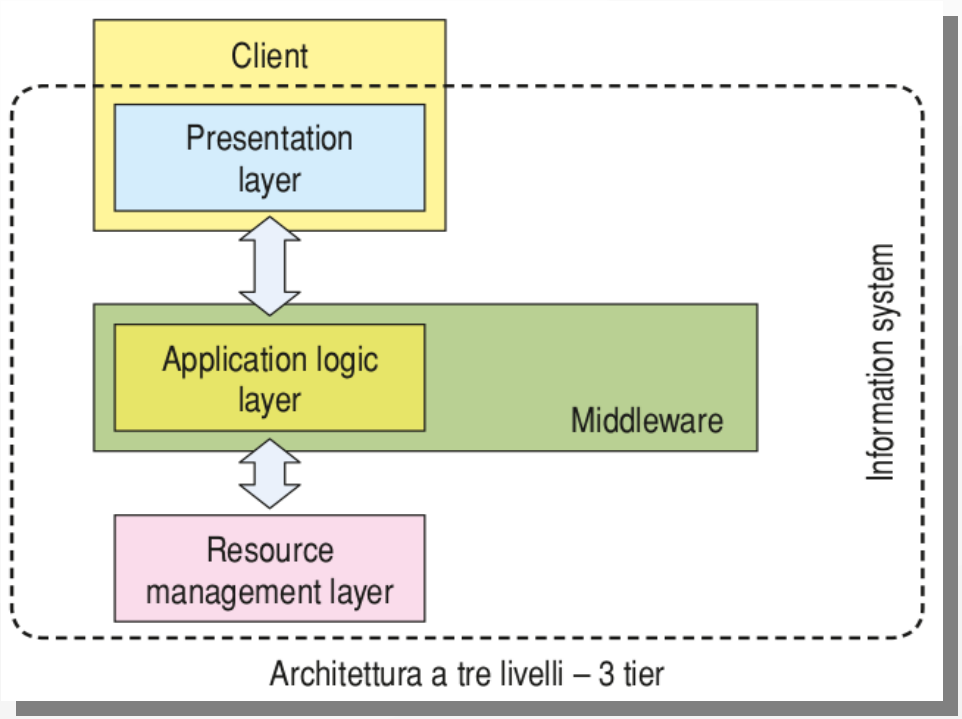
Al client possono essere assegnati compiti di calcolo in base alla potenza disponibile

3 livelli → 3 strati

3 livelli → 2 strati



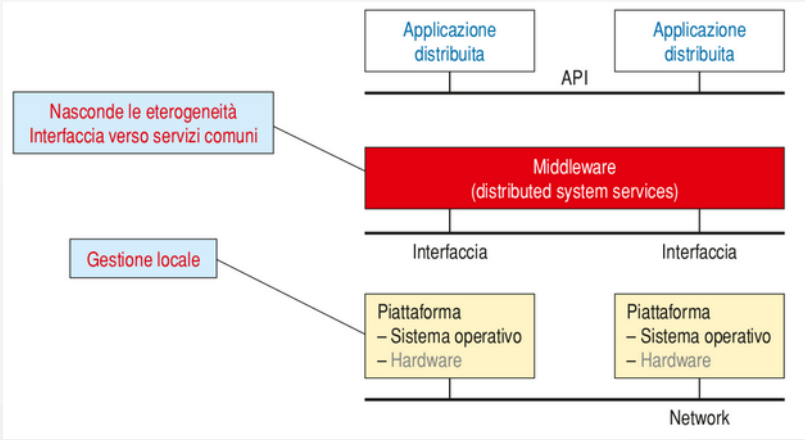
Livelli e strati



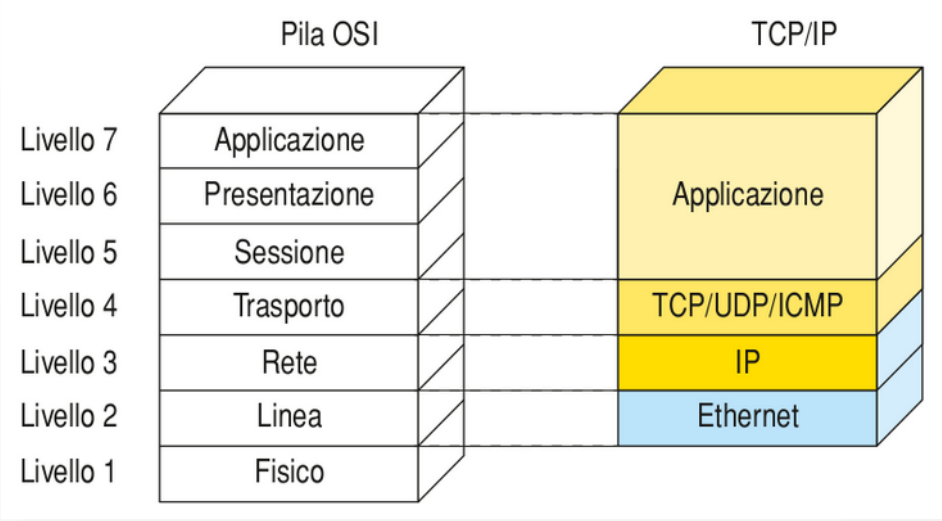
Middleware

Viene introdotto un ulteriore livello che ingloba il livello applicativo del sistema.

Si tratta di un insieme di servizi software che sta tra le applicazioni e il sistema operativo (Es: servizi di virtualizzazione delle risorse)



ISO/OSI – TCP/IP



Osservazione

Il livello di applicazione è lo strato protocollare che mette a disposizione i protocolli mediante i quali le applicazioni possono comunicare tra host remoti presenti sulla rete.

Applicazioni di rete

In generale una applicazione di rete è costituita da un insieme di programmi che vengono eseguiti su due o più computer contemporaneamente: questi operano interagendo tra loro utilizzando delle risorse comuni, accedendo cioè concorrentemente agli archivi (database), mediante la rete di comunicazione che li connette.

ARCHITETTURE SOFTWARE DI RETE

Meccanismi

Forniscono gli strumenti mediante i quali è possibile svolgere una determinata funzione.

Esempio

socket, semafori, timer, interrupt

ARCHITETTURE SOFTWARE DI RETE

Meccanismi

Forniscono gli strumenti mediante i quali è possibile svolgere una determinata funzione.

Esempio

socket, semafori, timer, interrupt

Politiche

Decidono come gli strumenti vengono utilizzati.

Esempio

protocollo del socket, mutua esclusione, quanto di tempo per il cambio di contesto, gestione delle interruzioni

ARCHITETTURE SOFTWARE DI RETE

Meccanismi

Forniscono gli strumenti mediante i quali è possibile svolgere una determinata funzione.

Esempio

socket, semafori, timer, interrupt

Politiche

Decidono come gli strumenti vengono utilizzati.

Esempio

protocollo del socket, mutua esclusione, quanto di tempo per il cambio di contesto, gestione delle interruzioni

Architetture

In generale, i meccanismi dovrebbero essere indipendenti dalle politiche in quanto queste ultime possono essere cambiate molto più spesso.

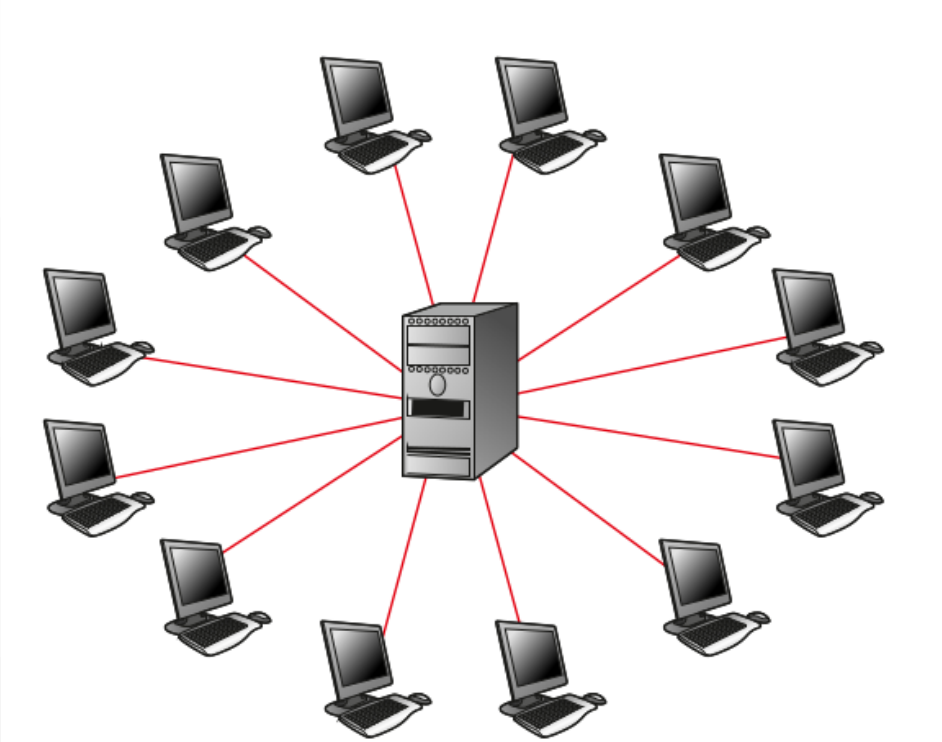
Esempio

Unix: mette a disposizione un insieme di primitive che possono essere utilizzate in svariati modi, a seconda dei parametri che vengono impostati

Windows/Mac OS: le primitive non sono separate dal resto del sistema ma sono integrate in esso. Ciò toglie flessibilità al sistema ma gli conferisce più uniformità grafica

Organizzazione generale del sistema. Stabilisce i principi sui quali si baserà lo sviluppo concreto dell'applicazione.

ARCHITETTURE SOFTWARE DI RETE



Client/Server

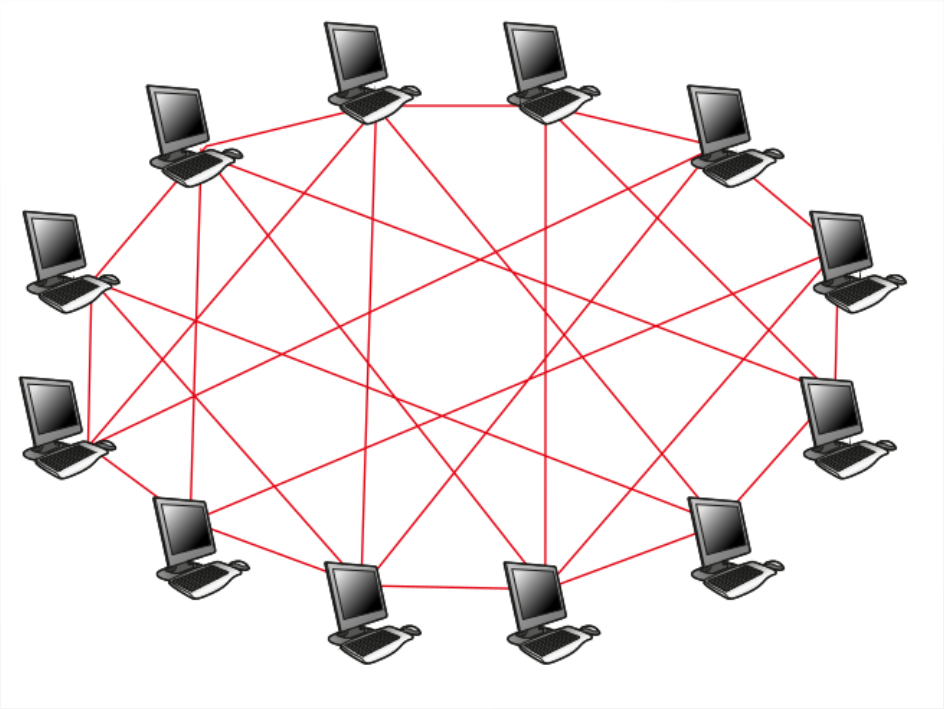
L'host del server deve sempre essere attivo e deve poter essere raggiunto dagli host client

La comunicazione fra due client avviene sempre tramite l'intermediazione del server

Più client possono comunicare direttamente con il server

Separazione netta fra chi fornisce il servizio e chi lo richiede. Gestione delle richieste

ARCHITETTURE SOFTWARE DI RETE



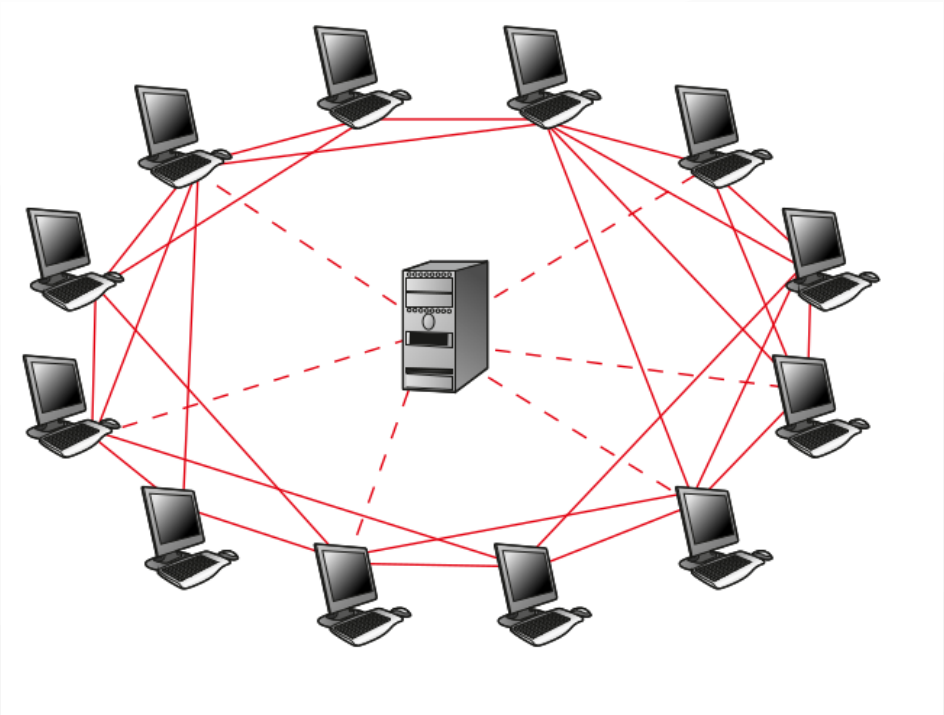
P2P decentralizzato

Ogni host ha una vita autonoma

Gli host possono comunicare direttamente fra di loro a seconda delle esigenze del momento, condividendo tutte le risorse a loro disposizione

Gestione delle risorse, gestione dei collegamenti

ARCHITETTURE SOFTWARE DI RETE



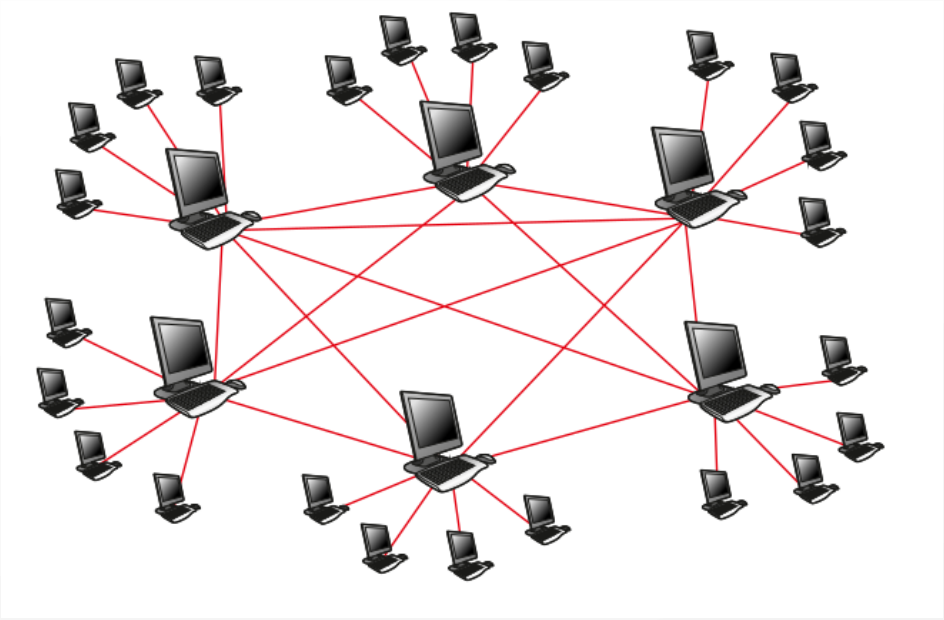
P2P centralizzato

Il server mantiene informazioni sulle risorse dei partecipanti alla rete, deve quindi essere raggiungibile come nel caso Client/Server

Una volta fatta la richiesta di una risorsa, i due host vengono messi in contatto e scambiano informazioni in maniera diretta

Gestione delle tabelle dei partecipanti, aggiornamento delle informazioni sulle risorse (indicizzazione)

ARCHITETTURE SOFTWARE DI RETE



P2P ibrido

Le funzionalità di indicizzazione sono affidate a dei nodi intermedi che si occupano di mantenere in vita l'intero sistema

Una parte degli host agisce da client.

Una parte agisce sia da client (per quanto riguarda il reperimento delle informazioni) che da server (rispondendo alle richieste dei client e facendo da intermediario)

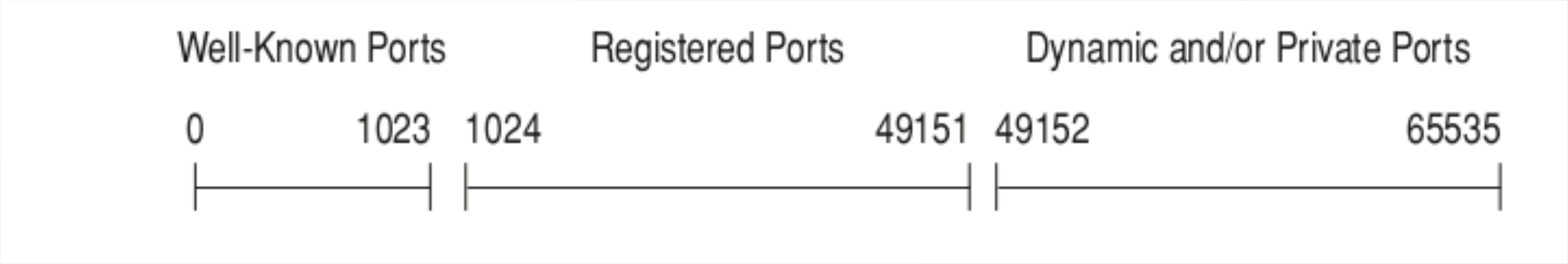
Approfondimento sui socket

Numeri di porta e porte riservate

Nonostante a livello teorico la scelta delle porte possa essere arbitraria, nella pratica esistono alcune porte che sono dedicate a particolari servizi

In tal modo lo sviluppo delle applicazioni che utilizzano un dato protocollo può fare affidamento su un numero di porta standard

Porta logica	Protocollo di rete
7	ECHO
21/tcp	FTP (file transfer protocol)
22/tcp	SSH (Secure SHell)
23/tcp	TELNET
25/tcp	SMTP (Simple Mail Transfer Protocol)
42	WINS (Windows Internet Naming Service)
53	DNS (Domain Name Service)
80/tcp	HTTP (Hyper Textual Transfer Protocol)
110/tcp	POP3 (Post Office Protocol, v3)
143/tcp	IMAP (Internet Message Access Protocol)
161	SNMP (Simple Network Management Protocol)
389/tcp	LDAP (Lightweight Directory Access Protocol)
443/tcp	HTTPS (Secure HTTP)



Approfondimento sui socket

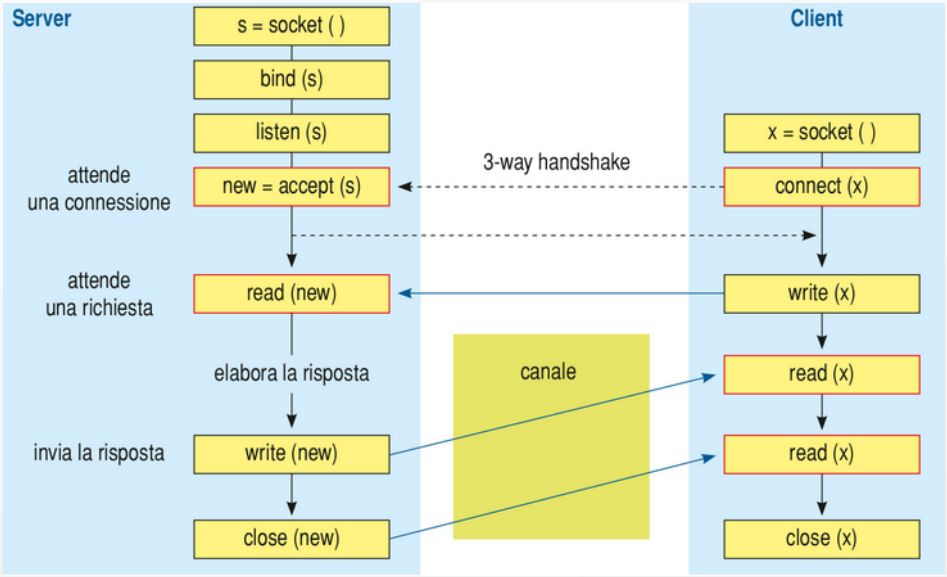
Socket TCP

Utilizzano il protocollo TCP/IP per lo scambio dei dati

La connessione è affidabile

Il protocollo del livello inferiore è più complesso (quindi più lento) ma consente la consegna dei messaggi in modo sicuro

- Controllo del flusso
- Controllo dell'errore
- Ricezione dei pacchetti nell'ordine di invio
- Frame di dimensioni più grandi



Approfondimento sui socket

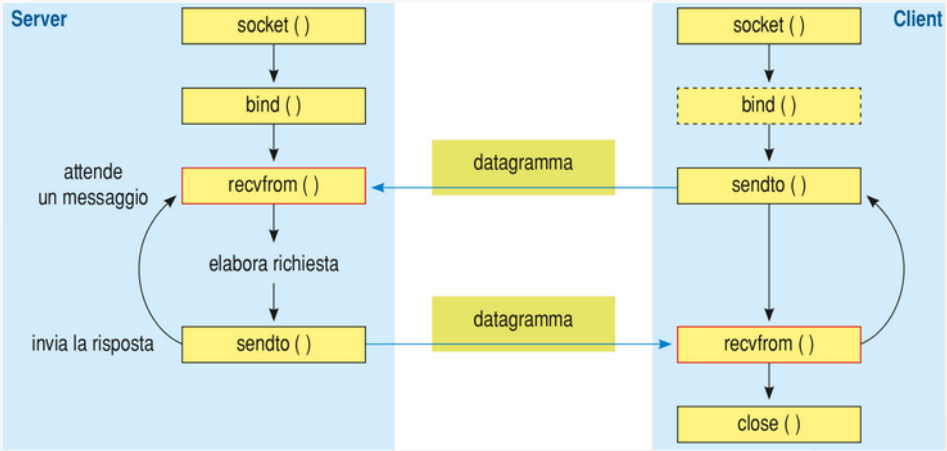
Socket UDP

Utilizzano il protocollo UDP/IP per lo scambio dei dati

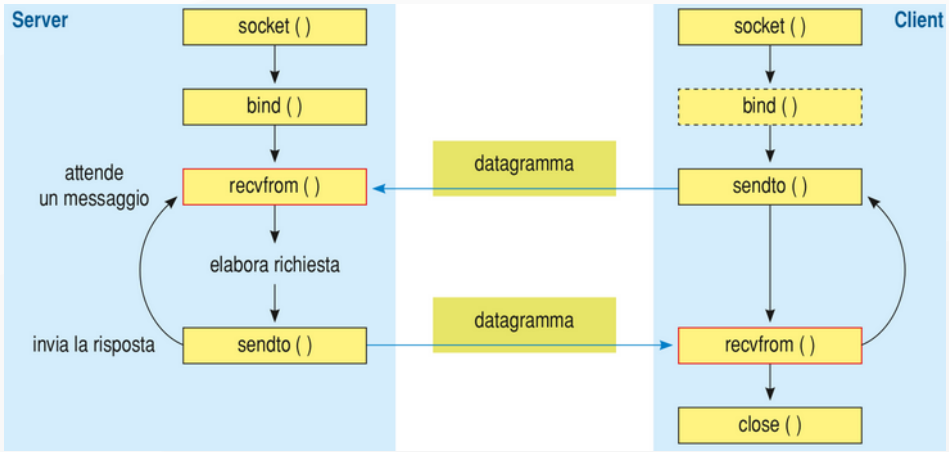
La connessione non è affidabile

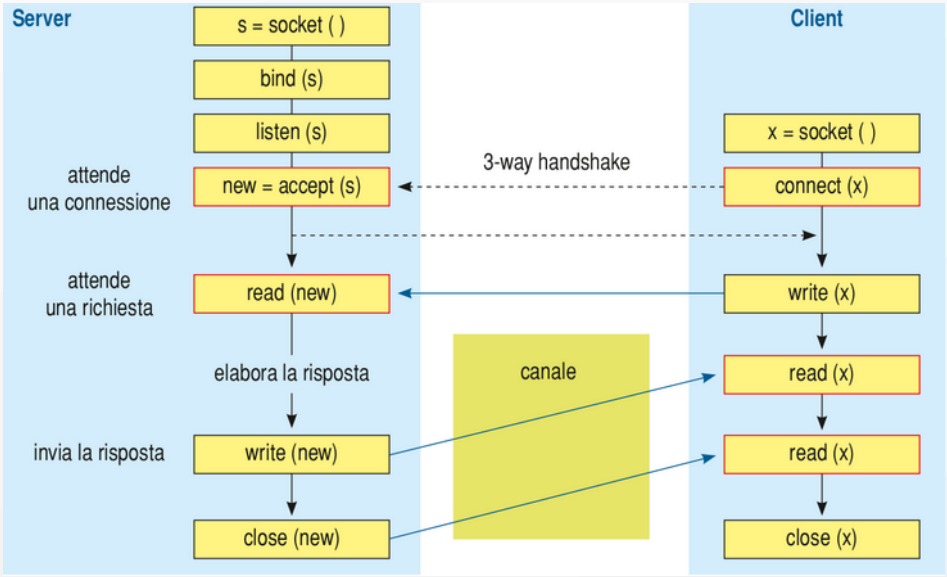
Il protocollo del livello inferiore è più semplice (quindi più veloce) ma non consente la consegna dei messaggi in modo sicuro

- Invio alla “cieca”
- Nessun controllo dell'errore
- Possibile ricezione dei pacchetti non in ordine
- Frame di dimensioni più piccole



TCP VS UDP (I)


Server
socket ( )
bind ( )
attende
un messaggio
recvfrom ( )
elabora richiesta
invia la risposta
sendto ( )
datagramma
datagramma
socket ( )
bind ( )
sendto ( )
recvfrom ( )
close ( )
Client


Server
s = socket ( )
bind (s)
listen (s)
attende
una connessione
new = accept (s)
3-way handshake
attende
una richiesta
read (new)
elabora la risposta
invia la risposta
write (new)
close (new)
canale
Client
x = socket ( )
connect (x)
write (x)
read (x)
read (x)
close (x)

UDP Socket

```c
int server()
{
        int serversock;
        struct sockaddr_in server_socket, from;
        char buffer[256];

        printf("Sto creando il socket...\n");
                serversock= socket(AF_INET, SOCK_DGRAM, 0);
        printf("Fatto.\n");

        printf("Sto impostando i parametri...\n");
                memset(&server_socket, 0, sizeof(server_socket));
                server_socket.sin_family = AF_INET;
                server_socket.sin_addr.s_addr = htonl(INADDR_ANY);
                server_socket.sin_port = htons(50000);
        printf("fatto.\n");

        printf("Sto impostando il collegamento...\n");
                bind(serversock,(struct sockaddr *)&server_socket,sizeof(server_socket));
        printf("fatto.\n");


        unsigned int fromlen = sizeof(struct sockaddr_in);
        recvfrom(serversock,buffer,256,0,(struct sockaddr *)&from, &fromlen);
        printf("RICEVUTO: %s\n", buffer);
        sendto(serversock,"Got your message\n",17, 0,(struct sockaddr *)&from,sizeof(sockaddr_in));


        return 0;

}
```

```c
int client()
{
        int clientsock;
        struct sockaddr_in server_socket, from;

        printf("Sto creando il socket...\n");
                clientsock= socket(AF_INET, SOCK_DGRAM, 0);
        printf("Fatto.\n");

        printf("Sto impostando i parametri...\n");
                memset(&server_socket, 0, sizeof(server_socket));
                server_socket.sin_family = AF_INET;
                server_socket.sin_addr.s_addr = inet_addr("127.0.0.1");
                server_socket.sin_port = htons(50000);
        printf("fatto.\n");

        char msg[] = "Hello Socket UDP";

        sendto(clientsock,msg, strlen(msg),0,(const struct sockaddr *)&server_socket,sizeof(sockaddr_in));

        char buffer[256];
        unsigned int dim = sizeof(sockaddr_in);
        recvfrom(clientsock,buffer,256,0,(struct sockaddr *)&from, &dim);
        printf("RICEVUTO: %s\n", buffer);

        close(clientsock);
    return 0;
}
```

TCP VS UDP (II)

```c
int server()
{
        int serversock;
        struct sockaddr_in server_socket, from;
        char buffer[256];

        printf("Sto creando il socket...\n");
                serversock= socket(AF_INET, SOCK_DGRAM, 0);
        printf("Fatto.\n");

        printf("Sto impostando i parametri...\n");
                memset(&server_socket, 0, sizeof(server_socket));
                server_socket.sin_family = AF_INET;
                server_socket.sin_addr.s_addr = htonl(INADDR_ANY);
                server_socket.sin_port = htons(50000);
        printf("fatto.\n");

        printf("Sto impostando il collegamento...\n");
                bind(serversock,(struct sockaddr *)&server_socket,sizeof(server_socket));
        printf("fatto.\n");


        unsigned int fromlen = sizeof(struct sockaddr_in);
        recvfrom(serversock,buffer,256,0,(struct sockaddr *)&from, &fromlen);
        printf("RICEVUTO: %s\n", buffer);
        sendto(serversock,"Got your message\n",17, 0,(struct sockaddr *)&from,sizeof(sockaddr_in));


        return 0;

}
```

```c
int main(int argc, char *argv[])
{
        int serversock;
        struct sockaddr_in server_socket;

        printf("Sto creando il socket...\n");
                serversock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        printf("fatto.\n");

        printf("Sto impostando i parametri...\n");
                memset(&server_socket, 0, sizeof(server_socket));
                server_socket.sin_family = AF_INET;
                server_socket.sin_addr.s_addr = htonl(INADDR_ANY);
                server_socket.sin_port = htons(50000);
        printf("fatto.\n");

        printf("Sto impostando il collegamento\n");
                bind(serversock, (struct sockaddr *) &server_socket, sizeof(server_socket));
        printf("fatto.\n");

        printf("Sono in attesa delle richieste dei client...\n");
                listen(serversock, 5);

        struct sockaddr_in client_socket;
        unsigned int clientlen = sizeof(client_socket);

        int clientsock = accept(serversock, (struct sockaddr *) &client_socket, &clientlen);

        printf("Connesso.\n");

        printf("Chiusura connessione ..\n");
                close(serversock);
        printf("fatto.\n");
}
```

TCP VS UDP (III)

```c
int client()
{
        int clientsock;
        struct sockaddr_in server_socket, from;

        printf("Sto creando il socket...\n");
                clientsock= socket(AF_INET, SOCK_DGRAM, 0);
        printf("Fatto.\n");

        printf("Sto impostando i parametri...\n");
                memset(&server_socket, 0, sizeof(server_socket));
                server_socket.sin_family = AF_INET;
                server_socket.sin_addr.s_addr = inet_addr("127.0.0.1");
                server_socket.sin_port = htons(50000);
        printf("fatto.\n");

        char msg[] = "Hello Socket UDP";

        sendto(clientsock,msg, strlen(msg),0,(const struct sockaddr *)&server_socket,sizeof(sockaddr_in));

        char buffer[256];
        unsigned int dim = sizeof(sockaddr_in);
        recvfrom(clientsock,buffer,256,0,(struct sockaddr *)&from, &dim);
        printf("RICEVUTO: %s\n", buffer);

        close(clientsock);
   return 0;
}
```

```c
int main(int argc, char *argv[])
{
        int clientsock;
        struct sockaddr_in server_socket;

        printf("Sto creando il socket...\n");
                clientsock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        printf("fatto.\n");

        printf("Sto impostando i parametri...\n");
                memset(&server_socket, 0, sizeof(server_socket));
                server_socket.sin_family = AF_INET;
                server_socket.sin_addr.s_addr = inet_addr("127.0.0.1");
                server_socket.sin_port = htons(50000);
        printf("fatto.\n");

        printf("Connessione al server...\n");
                connect(clientsock, (struct sockaddr *) &server_socket, sizeof(server_socket));
        printf("fatto.\n");

        write(clientsock,"Hello Socket!",strlen("Hello Socket!"));

        printf("Chiusura connessione ..\n");
                close(clientsock);
        printf("fatto.\n");
}
```

Esercizio

Una stazione meteorologica mette a disposizione un dispositivo per inviare ad una centrale i dati relativi a temperatura (T), umidità (U) e velocità del vento (V). I dati vengono inviati ogni 10 secondi dalla stazione meteorologica.

Implementare:

Un client che metta a disposizione due funzioni

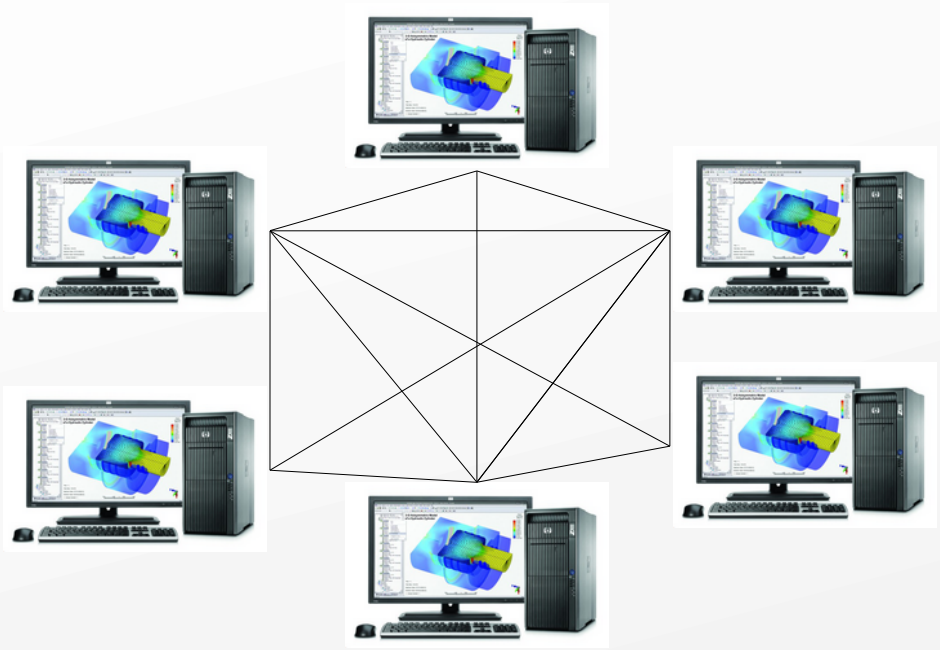
- Inviare i dati al server (utilizzare la generazione di numeri casuali per simulare i dati raccolti dai sensori)
- Scaricare i dati relativi ad un giorno

Un server che metta a disposizione due funzioni

- Raccogliere i dati e memorizzarli a fine giornata in un file di testo. Il file di testo ha come nome DATI-<data del giorno>.txt
- Inviare su richiesta del client un file con i dati di un determinato giorno

Utilizzare in modo opportuno i protocolli TCP/UDP per le diverse funzioni.

Sistemi distribuiti e formato dei dati



Interoperabilità

Per poter garantire lo scambio dei dati fra sistemi diversi è necessario che i dati siano interpretabili da tutti i componenti del sistema

In un sistema distribuito possono essere presenti

- Diverse architetture software
- Diverse architetture Hardware

Lo scambio dei dati non può avvenire a basso livello

È necessario

- Un modello per la creazione e interpretazione dei dati
- Un approccio che offra flessibilità

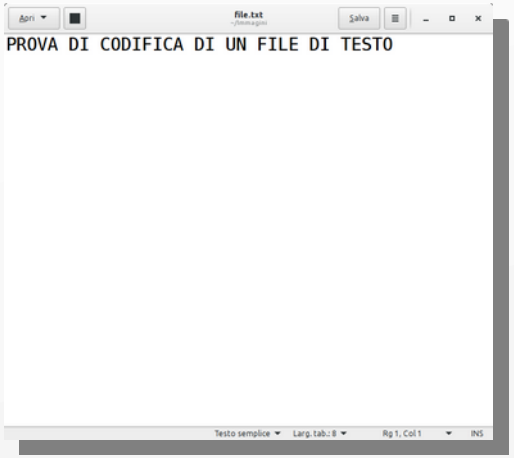
Dati: Contenuto VS visualizzazione

Esempio: File di testo

Viene memorizzato secondo una data codifica (ASCII, UNICODE, etc…)

Quando viene aperto in un editor di testo, posso scegliere il tipo di carattere e la dimensione con cui lavorare.

La scelta del tipo di carattere dell'editor non ha effetti sul contenuto del file. Rende solo più comoda la visualizzazione del contenuto da parte dell'utente.




```
giovanni@debian: ~/Immagini
File  Modifica  Visualizza  Cerca  Terminale  Aiuto
giovanni@debian:~/Immagini$ hd file.txt
00000000  50 52 4f 56 41 20 44 49  20 43 4f 44 49 46 49 43  |PROVA DI CODIFIC|
00000010  41 20 44 49 20 55 4e 20  46 49 4c 45 20 44 49 20  |A DI UN FILE DI |
00000020  54 45 53 54 4f 0a                                 |TESTO.|
00000026
giovanni@debian:~/Immagini$
```

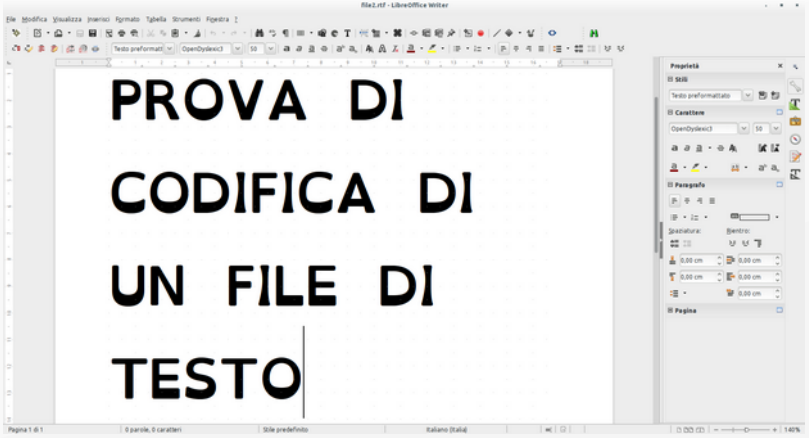
Dati: Contenuto + Visualizzazione

Esempio: documento di testo

Viene memorizzato secondo una data codifica (ASCII, UNICODE, etc…)

Quando viene aperto in un elaboratore di testi, posso scegliere il tipo di carattere e la dimensione.

La scelta del tipo di carattere ha effetti sul contenuto del file.



giovanni@debian: ~/Immagini

File Modifica Visualizza Cerca Terminale Aiuto

```
{\rtf1\ansi\deff3\adeflang1025
{\fonttbl{\f0\froman\fprq2\fcharset0 Times New Roman;}{\f1\froman\fprq2\fcharset2 Symbol;}{\f2\fswiss\fprq2\fcharset0 Arial;}{\f3\froman\fprq2\fcharset0 Liberation Serif{\*\falt Times New Roman};}{\f4\froman\fprq2\fcharset0 Liberation Sans{\*\falt Arial};}{\f5\froman\fprq2\fcharset0 Liberation Mono{\*\falt Courier New};}{\f6\fnil\fprq2\fcharset128 OpenDyslexic3;}{\f7\fnil\fprq2\fcharset0 Liberation Mono{\*\falt Courier New};}{\f8\fnil\fprq2\fcharset0 FreeSans;}{\f9\fnil\fprq2\fcharset0 Liberation Serif{\*\falt Times New Roman};}}
{\colortbl;\red0\green0\blue0;\red0\green0\blue255;\red0\green255\blue255;\red0\green255\blue0;\red255\green0\blue255;\red255\green0\blue0;\red255\green255\blue0;\red255\green255\blue255;\red0\green0\blue128;\red0\green128\blue128;\red0\green128\blue0;\red128\green0\blue128;\red128\green0\blue0;\red128\green128\blue0;\red128\green128\blue128;\red192\green192\blue192;}
{\stylesheet{\s0\snext0\ql\nowidctlpar\hyphpar0\ltrpar\cf1\dbch\af8\langfe1081\dbch\af9\afs24\kerning1\alang1081\loch\f3\fs24\lang1040 Normal;}
{\s15\sbasedon0\snext16\ql\nowidctlpar\hyphpar0\sb240\sa120\keepn\ltrpar\cf1\dbch\af8\langfe1081\dbch\af9\afs24\loch\f4\fs28\lang1040 Titolo;}
{\s16\sbasedon0\snext16\sl288\slmult1\ql\nowidctlpar\hyphpar0\sb0\sa140\ltrpar\cf1\dbch\af8\langfe1081\dbch\af9\afs24\loch\f3\fs24\lang1040 Corpo del testo;}
{\s17\sbasedon16\snext17\sl288\slmult1\ql\nowidctlpar\hyphpar0\sb0\sa140\ltrpar\cf1\dbch\af8\langfe1081\dbch\af9\afs24\loch\f3\fs24\lang1040 Elenco;}
{\s18\sbasedon0\snext18\ql\nowidctlpar\hyphpar0\sb120\sa120\ltrpar\cf1\i\dbch\af8\langfe1081\dbch\af9\afs24\loch\f3\fs24\lang1040 Didascalia;}
{\s19\sbasedon0\snext19\ql\nowidctlpar\hyphpar0\ltrpar\cf1\dbch\af8\langfe1081\dbch\af9\afs24\loch\f3\fs24\lang1040 Indice;}
{\s20\sbasedon0\snext20\ql\nowidctlpar\hyphpar0\sb0\sa0\ltrpar\cf1\dbch\af7\langfe1081\dbch\af9\afs24\loch\f5\fs20\lang1040 Testo preformattato;}
}{\*\generator LibreOffice/5.0.4.2$Linux_X86_64 LibreOffice_project/00m0$Build-2}{\info{\creatim\yr0\mo0\dy0\hr0\min0}{\revtim\yr2016\mo1\dy22\hr15\min30}{\printim\yr0\mo0\dy0\hr0\min0}}\deftab709
\hyphauto0\viewscale140
{\*\pgdsctbl
{\pgdsc0\pgdscuse451\pgwsxn11906\pghsxn16838\marglsxn1134\margrsxn1134\margtsxn1134\margbsxn1134\pgdscnxt0 Stile predefinito;}}
\formshade{\*\pgdscno0}\paperh16838\paperw11906\margl1134\margr1134\margt1134\margb1134\sectd\sbknone\sectunlocked1\pgndec\pgwsxn11906\pghsxn16838\marglsxn1134\margrsxn1134\margtsxn1134\margbsxn1134\ftnbj\ftnstart1\ftnrstcont\ftnnar\aenddoc\aftnrstcont\aftnstart1\aftnnrlc
{\*\ftnsep\chftnsep}\pgndec\pard\plain \s20\ql\nowidctlpar\hyphpar0\sb0\sa0\ltrpar\cf1\dbch\af7\langfe1081\dbch\af9\afs24\loch\f5\fs20\lang1040\sb0\sa0{\afs100\rtlch \ltrch\loch\fs100\loch\f6
PROVA DI CODIFICA DI UN FILE DI TESTO}
\par }giovanni@debian:~/Immagini$
```

Sistemi distribuiti e formato dei dati

Linguaggi di programmazione VS linguaggi di markup

Un linguaggio di programmazione definisce le istruzioni che devono essere eseguite da un calcolatore tramite opportune parole chiave

Un linguaggio di markup descrive le proprietà di un documento tramite opportune etichette

Metalinguaggio

Linguaggi che servono a descrivere altri linguaggi

In altre parole, insieme di regole che permettono di creare una descrizione dei dati

Un metalinguaggio definisce regole ma non specifica contenuti

xml

Dal libro di testo

L’ eXtensible Markup Language (XML) non è l’ennesimo linguaggio di markup né l’evoluzione HTML.

È un meta-linguaggio di markup, cioè un linguaggio che permette di definire altri linguaggi di markup.

XML non ha tag predefiniti e non serve né per programmare né per definire pagine Web.

È costituito da un insieme standard di regole sintattiche per modellare la struttura di documenti e dati.

Un file xml è un file di testo in cui devono essere rispettate determinate regole sintattiche

In teoria...

Formato dei tag

- Tag di apertura
- Tag di chiusura
- L'ultimo tag aperto deve essere il primo chiuso

Contenuto del documento

- Il testo compreso fra due tag costituisce il contenuto del documento
- All'interno di un tag si possono specificare attributi e valori per gli stessi

Elemento

- È tutto ciò che va dal tag iniziale al tag finale e può avere diversi tipi di contenuto: può essere formato da altri elementi, può avere un contenuto misto oppure semplice e può anche essere vuoto. Un elemento può contenere attributi.

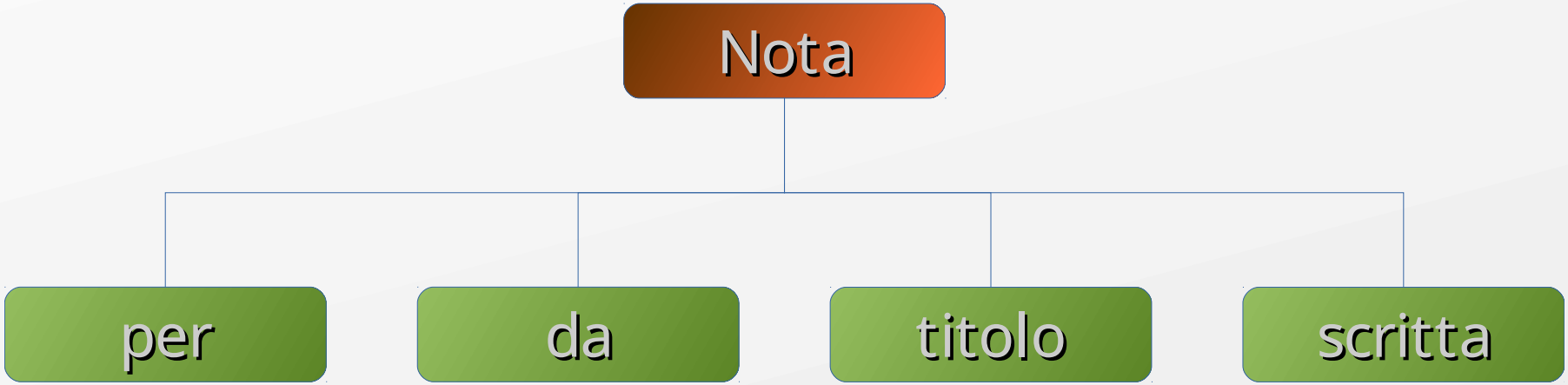
XML e rappresentazione dei dati

… e in pratica

```
<nota>
    <per>Classe VB</per>
    <da data="1-1-1900">Giovanni Scavello</da>
    <titolo>Esempio XML</titolo>
    <scritta>Ecco un esempio di un documento XML</scritta>
</nota>
```

La rappresentazione ad albero è utile perché in questo modo si possono analizzare i documenti in maniera standard utilizzando le stesse tecniche algoritmiche che si usano in fase di compilazione: analisi lessicale, analisi sintattica.

A differenza delle procedure di compilazione, tuttavia, non viene analizzato il "valore" dei tag in quanto questi ultimi possono essere definiti a piacere a patto che venga rispettata la sintassi.



Utilizzo di xml

Programmazione lato server

Viene utilizzato per la definizione del deployment descriptor, cioè del file web.xml che viene utilizzato per informare un server in cui è installata un'applicazione J2EE riguardo alla configurazione dell'applicazione stessa.

In esso possono essere definiti parametri di inizializzazione, mappature dei percorsi e, in generale, tutto ciò che contiene informazioni importanti per la corretta esecuzione dell'applicazione.

Esempio

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <servlet>
   <servlet-name>HelloWorldServlet</servlet-name>
   <servlet-class>org.test.servlet.HelloWorldServel</servlet-
class>
  </servlet>
  <servlet-mapping>
   <servlet-name>HelloWorldServlet</servlet-name>
   <url-pattern>/HelloWorldServel</url-pattern>
  </servlet-mapping>
  <session-config>
   <session-timeout>30</session-timeout>
  </session-config>
</web-app>
```

Utilizzo di xml

Database

```
<?xml version="1.0" standalone="yes"?>
<DATABASE>

<UTENTI>
  <USERNAME>mortadelo</USERNAME>
  <PASSWORD>*******</USERNAME>
</UTENTI>
<UTENTI>
  <USERNAME>filemon</USERNAME>
  <PASSWORD>*******</PASSWORD>
</UTENTI>

<GRUPPI>
  <NOME>God</NomeCampo1>
  <PERMESSI>everithing</NomeCampo1>
</GRUPPI>
 <GRUPPI>
  <NOME>Slave</NomeCampo1>
  <PERMESSI>Just Obey</NomeCampo1>
 </GRUPPI>

</DATABASE>
```

UTENTI	
USERNAME	PASSWORD
Mortadelo	********
Filemón	++++++++++

GRUPPI	
NOME	PERMESSI
God	Everithing
Slave	Just obey

Xml: contenuto VS visualizzazione

```xml
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>

<food>
<name>Belgian Waffles</name>
<price>$5.95</price>
<description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
<calories>650</calories>
</food>

<food>
<name>Strawberry Belgian Waffles</name>
<price>$7.95</price>
<description>Light Belgian waffles covered with strawberries and whipped cream</description>
<calories>900</calories>
</food>

<food>
<name>Berry-Berry Belgian Waffles</name>
<price>$8.95</price>
<description>Light Belgian waffles covered with an assortment of fresh berries and whipped cream</description>
<calories>900</calories>
</food>

<food>
<name>French Toast</name>
<price>$4.50</price>
<description>Thick slices made from our homemade sourdough bread</description>
<calories>600</calories>
</food>

<food>
<name>Homestyle Breakfast</name>
<price>$6.95</price>
<description>Two eggs, bacon or sausage, toast, and our ever-popular hash browns</description>
<calories>950</calories>
</food>

</breakfast_menu>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">
<xsl:for-each select="breakfast_menu/food">
  <div style="background-color:teal;color:white;padding:4px">
    <span style="font-weight:bold"><xsl:value-of select="name"/> - </span>
    <xsl:value-of select="price"/>
    </div>
  <div style="margin-left:20px;margin-bottom:1em;font-size:10pt">
    <p>
    <xsl:value-of select="description"/>
    <span style="font-style:italic"> (<xsl:value-of select="calories"/> calories per serving)</span>
    </p>
  </div>
</xsl:for-each>
</body>
</html>
```

Belgian Waffles - $5.95

two of our famous Belgian Waffles with plenty of real maple syrup *(650 calories per serving)*

Strawberry Belgian Waffles - $7.95

light Belgian waffles covered with strawberries and whipped cream *(900 calories per serving)*

Berry-Berry Belgian Waffles - $8.95

light Belgian waffles covered with an assortment of fresh berries and whipped cream *(900 calories per serving)*

French Toast - $4.50

thick slices made from our homemade sourdough bread *(600 calories per serving)*

Homestyle Breakfast - $6.95

two eggs, bacon or sausage, toast, and our ever-popular hash browns *(950 calories per serving)*

Conclusioni

Pro

- Separazione fra contenuto e visualizzazione
- Portabilità dei dati
- Basato su tag e non su posizione
- Flessibilità
- File di testo

Contro

- Ridondanza
- File di testo

Osservazioni

L'XML serve a definire tipi di dati complessi indipendenti dalla piattaforma e dall'applicazione.

Per poter utilizzare un documento xml è necessario utilizzare un linguaggio di programmazione che metta a disposizione apposite librerie per l'analisi dei file di testo

Il componente principale di tali librerie è il parser che permette di identificare gli elementi presenti in un documento.

Alcuni esempi si trovano qui

http://www.mkyong.com/tutorials/java-xml-tutorials/

SAX VS DOM

SAX

***S**imple **A**pi for **X**ml*

Il parser SAX presenta ogni nodo del documento XML in sequenza. Così facendo, quando viene processato un nodo bisogna aver già memorizzato le informazioni dei nodi analizzati precedentemente e ritenuti rilevanti. Inoltre non si hanno informazioni sui nodi seguenti.

Il parser SAX usa poca memoria ed è utile in quei casi in cui il documento da analizzare non necessita elaborazioni complesse (esempio: estrazione di informazioni da un file di configurazione)

SAX VS DOM

SAX

***S**imple **A**pi for **X**ml*

Il parser SAX presenta ogni nodo del documento XML in sequenza. Così facendo, quando viene processato un nodo bisogna aver già memorizzato le informazioni dei nodi analizzati precedentemente e ritenuti rilevanti. Inoltre non si hanno informazioni sui nodi seguenti.

Il parser SAX usa poca memoria ed è utile in quei casi in cui il documento da analizzare non necessita elaborazioni complesse (esempio: estrazione di informazioni da un file di configurazione)

DOM

***D**ocument **O**bject **M**odel*

Il parser DOM carica l'intero documento XML in una struttura di memoria. Il documento diventa così accessibile mediante i metodi di navigazione tipici degli alberi:

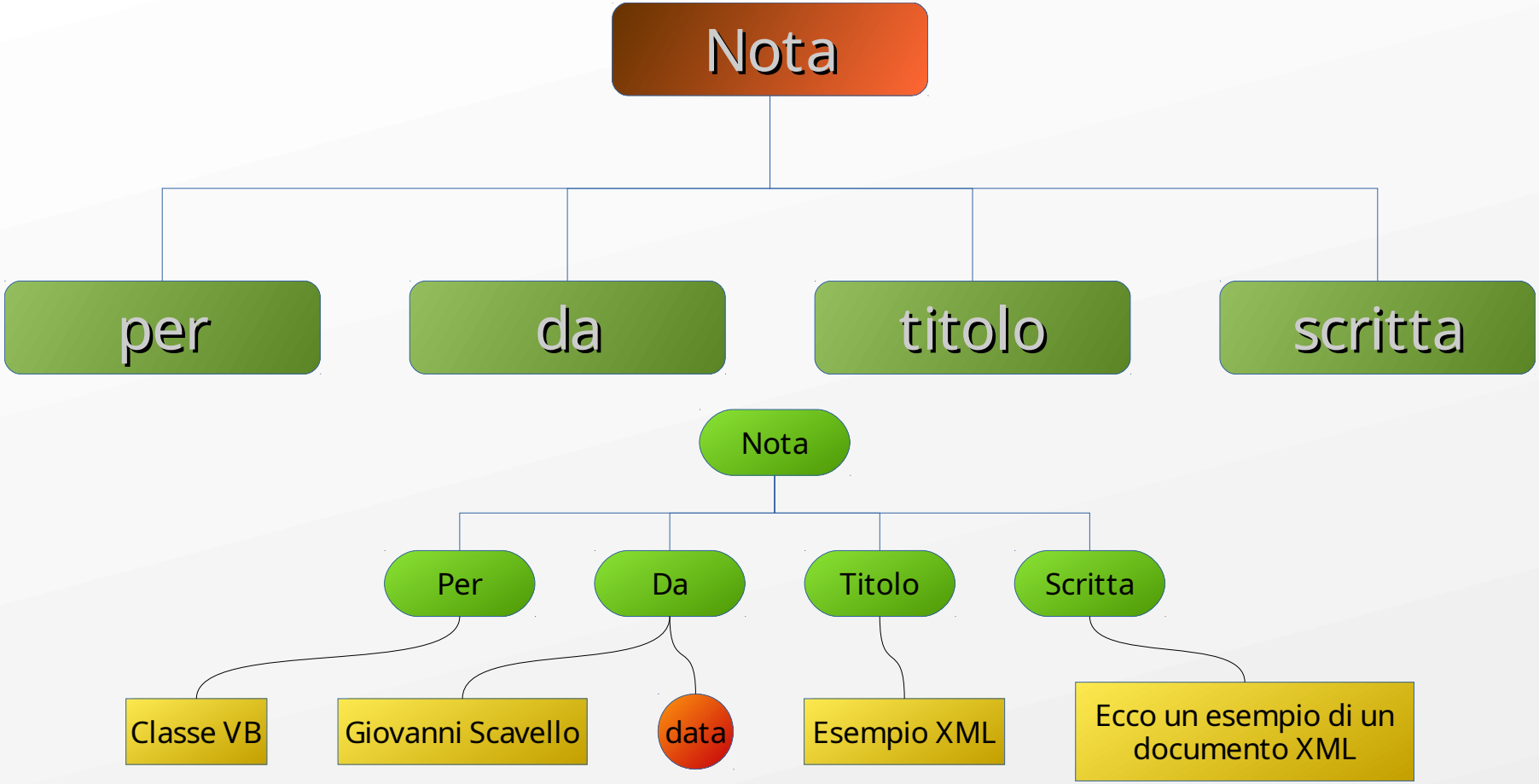
getnode()

getchildren()

Dovendo memorizzare il documento in una struttura dati, l'occupazione di memoria diventa un fattore non trascurabile.

Tuttavia, in questo modo, è possibile eseguire sul documento elaborazioni più complesse della semplice analisi sequenziale

Manipolazione di file XML (DOM)



Manipolazione di file XML in C++ (DOM)

Element

Text or comment

Attribute

```
void print_node(const xmlpp::Node* node, unsigned int indentation = 0)
{

  const xmlpp::ContentNode*     nodeContent = dynamic_cast<const xmlpp::ContentNode*>(node);
  const xmlpp::TextNode*        nodeText    = dynamic_cast<const xmlpp::TextNode*>(node);
  const xmlpp::CommentNode*     nodeComment = dynamic_cast<const xmlpp::CommentNode*>(node);
  const xmlpp::Element*         nodeElement = dynamic_cast<const xmlpp::Element*>(node);

  if(nodeText && nodeText->is_white_space()) //Let's ignore the indenting - you don't always want to do this.
    return;

  //NOME DEL NODO (è il tag dell'XML)
    if(!nodeText && !nodeComment && !node->get_name().empty()) //Let's not say "name: text".
    {
        print_indentation(indentation);
        std::cout << "[" << node->get_name() <<"]" << std::endl;
    }

  //SE IL NODO HA UN CONTENUTO, LO STAMPA
    if(nodeContent)
    {
        print_indentation(indentation);
        std::cout << "{" << nodeContent->get_content() << "}" << std::endl;
    }
      //VISUALIZZA EVENTUALI ATTRIBUTI DEL NODO
    if(nodeElement)
    {
        const xmlpp::Element::AttributeList& attributes = nodeElement->get_attributes();
        for(xmlpp::Element::AttributeList::const_iterator iter = attributes.begin(); iter != attributes.end(); ++iter)
        {
            const xmlpp::Attribute* attribute = *iter;
            print_indentation(indentation);
            std::cout << "-" <<node->get_name() << "." << attribute->get_name() << " = " << attribute->get_value() << std::endl;
        }

    }
    xmlpp::Node::NodeList list = node->get_children();
    for(xmlpp::Node::NodeList::iterator iter = list.begin(); iter != list.end(); ++iter)
    {
      print_node(*iter, indentation + 1); //recursive
    }

}
```

Manipolazione di file XML in Java (DOM)

Element

Text or comment

Attribute

```java
private static void printNote(Node tempNode, int ind)
{

// get node name and value
  if (tempNode.getNodeType() != Node.TEXT_NODE && tempNode.getNodeType() != Node.COMMENT_NODE)
  {
      printspace(ind);
      System.out.println("[" + tempNode.getNodeName() + " ]");
  }

  if (tempNode.getNodeType() == Node.TEXT_NODE && tempNode.getTextContent().trim().length() != 0)
  {   printspace(ind);
      System.out.println("{" + tempNode.getNodeValue() + " }");

  }

  // make sure it's element node.
  if (tempNode.getNodeType() == Node.ELEMENT_NODE)
  {
      if (tempNode.hasAttributes())
      {
          // get attributes names and values
          NamedNodeMap nodeMap = tempNode.getAttributes();

          for (int i = 0; i < nodeMap.getLength(); i++)
          {
              Node node = nodeMap.item(i);
              printspace(ind);
              System.out.println("-" + tempNode.getNodeName() + "." + node.getNodeName() + " = " + node.getNodeValue());
          }

      }

      if (tempNode.hasChildNodes())
      {
          // loop again if has child nodes
          NodeList l = tempNode.getChildNodes();
          for (int count = 0; count < l.getLength(); count++)
              printNote(l.item(count), ind+1);
      }

  }

}
```

C++ VS Java (DOM)

```cpp
void print_node(const xmlpp::Node* node, unsigned int indentation = 0)
{
  const xmlpp::ContentNode*     nodeContent = dynamic_cast<const xmlpp::ContentNode*>(node);
  const xmlpp::TextNode*        nodeText    = dynamic_cast<const xmlpp::TextNode*>(node);
  const xmlpp::CommentNode*     nodeComment = dynamic_cast<const xmlpp::CommentNode*>(node);
  const xmlpp::Element*         nodeElement = dynamic_cast<const xmlpp::Element*>(node);

  if(nodeText && nodeText->is_white_space()) //Let's ignore the indenting - you don't always want to do this.
    return;

  //NOME DEL NODO (è il tag dell'XML)
    if(!nodeText && !nodeComment && !node->get_name().empty()) //Let's not say "name: text".
    {
        print_indentation(indentation);
        std::cout << "[" << node->get_name() <<"]" << std::endl;
    }
  //SE IL NODO HA UN CONTENUTO, LO STAMPA
    if(nodeContent)
    {
        print_indentation(indentation);
        std::cout << "{" << nodeContent->get_content() << "}" << std::endl;
    }
      //VISUALIZZA EVENTUALI ATTRIBUTI DEL NODO
    if(nodeElement)
    {
        const xmlpp::Element::AttributeList& attributes = nodeElement->get_attributes();
        for(xmlpp::Element::AttributeList::const_iterator iter = attributes.begin(); iter != attributes.end(); ++iter)
        {
            const xmlpp::Attribute* attribute = *iter;
            print_indentation(indentation);
            std::cout << "-" <<node->get_name() << "." << attribute->get_name() << " = " << attribute->get_value() << std::endl;
        }
    }
    xmlpp::Node::NodeList list = node->get_children();
    for(xmlpp::Node::NodeList::iterator iter = list.begin(); iter != list.end(); ++iter)
    {
      print_node(*iter, indentation + 1); //recursive
    }
}
```

```java
private static void printNote(Node tempNode, int ind)
{

// get node name and value
  if (tempNode.getNodeType() != Node.TEXT_NODE && tempNode.getNodeType() != Node.COMMENT_NODE)
  {
      printspace(ind);
      System.out.println("[" + tempNode.getNodeName() + " ]");
  }

  if (tempNode.getNodeType() == Node.TEXT_NODE && tempNode.getTextContent().trim().length() != 0)
  {   printspace(ind);
      System.out.println("{" + tempNode.getNodeValue() + " }");

  }

  // make sure it's element node.
  if (tempNode.getNodeType() == Node.ELEMENT_NODE)
  {
      if (tempNode.hasAttributes())
      {
          // get attributes names and values
          NamedNodeMap nodeMap = tempNode.getAttributes();

          for (int i = 0; i < nodeMap.getLength(); i++)
          {
              Node node = nodeMap.item(i);
              printspace(ind);
              System.out.println("-" + tempNode.getNodeName() + "." + node.getNodeName() + " = " + node.getNodeValue());
          }

      }

      if (tempNode.hasChildNodes())
      {
          // loop again if has child nodes
          NodeList l = tempNode.getChildNodes();
          for (int count = 0; count < l.getLength(); count++)
              printNote(l.item(count), ind+1);
      }

  }

}
```

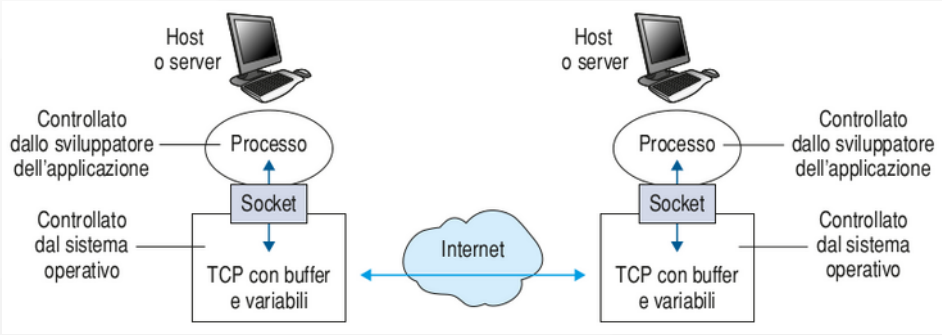
Dai socket al WEB

Socket e Reti

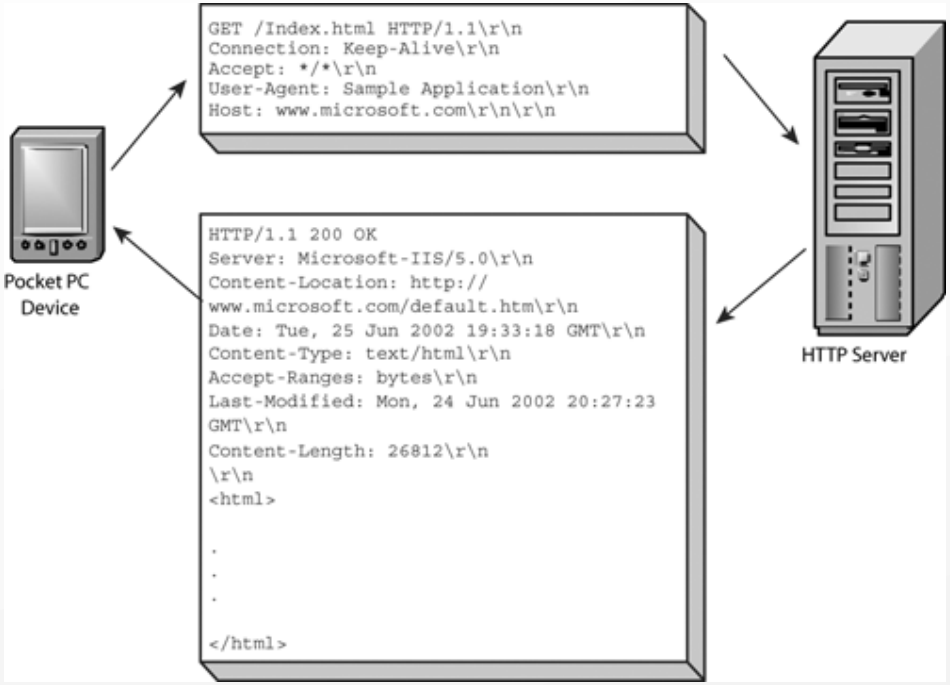
- Strumento di comunicazione di basso livello (livello trasporto)
- Applicazioni gestite dal sistema operativo come un qualsiasi processo
- Lo sviluppatore deve gestire tutti gli aspetti della comunicazione
- Lo scambio dei dati dipende dall'implementazione dell'applicazione

WEB

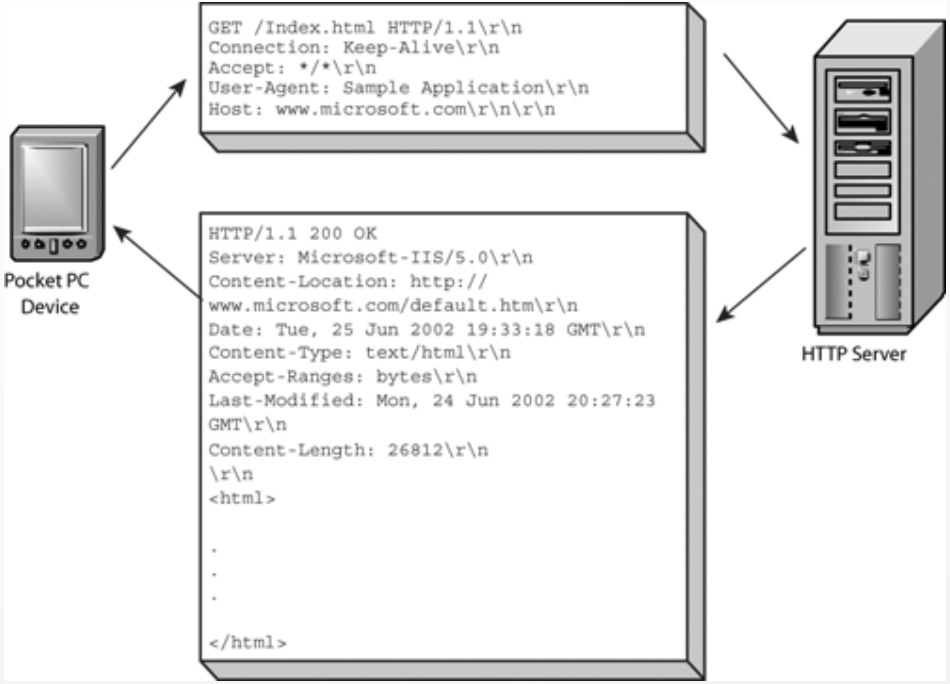
- Strumento di comunicazione di alto livello (protocollo http, livello applicazione)
- Applicazioni gestite da altri programmi (web server)
- Lo sviluppatore gestisce (quasi) solo i contenuti
- Lo scambio dei dati (pagine web) non dipende dall'implementazione (standard W3C)

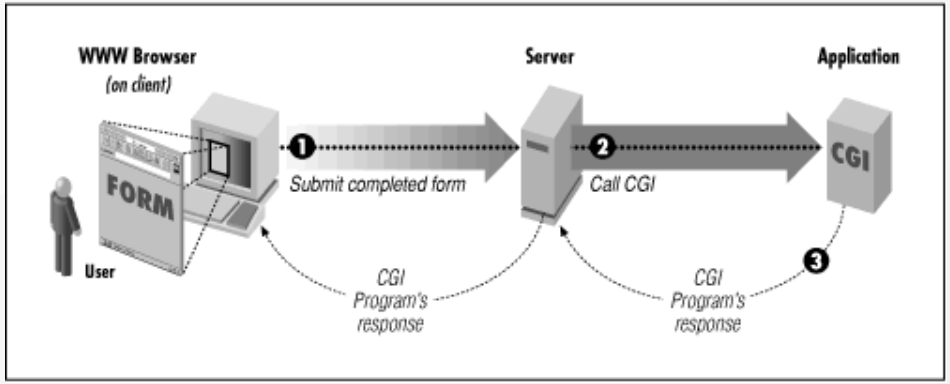


Dal WEB al CGI

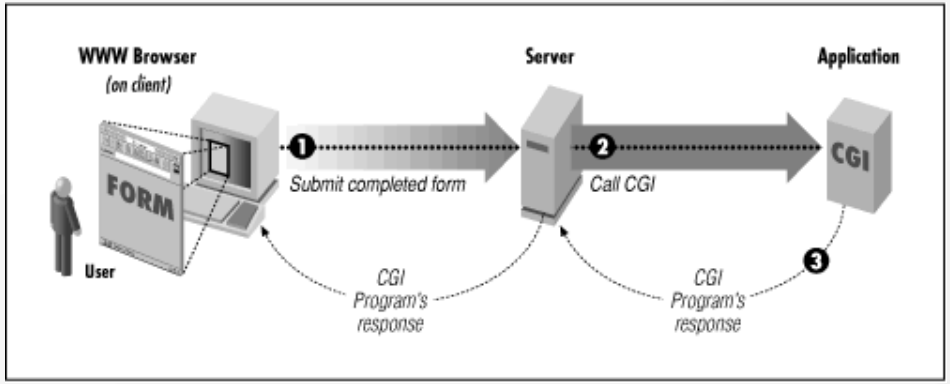


Dal WEB al CGI

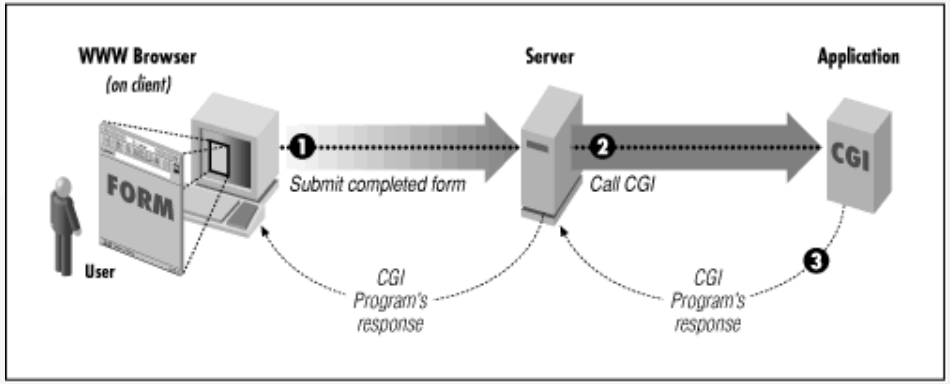




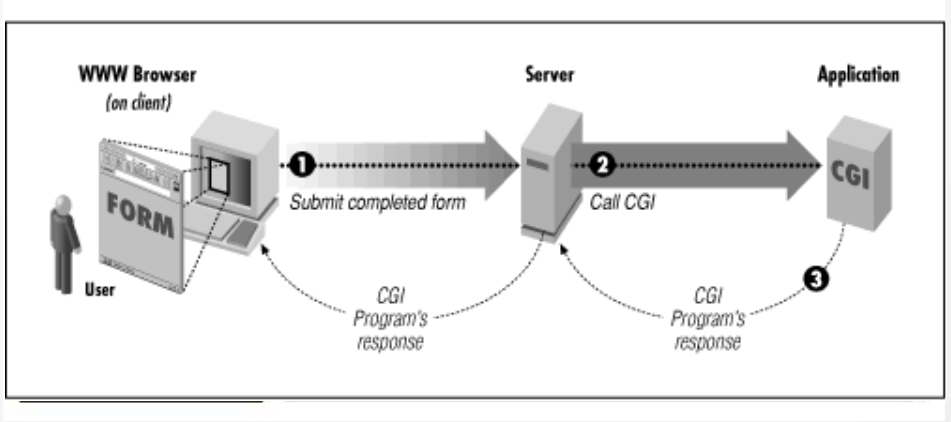
Dal WEB al CGI



Dal WEB al CGI



Dal WEB al CGI



Dal WEB al CGI

La richiesta di un progrmma CGI è molto simile a quella delle normali pagine web. La differenza è che quando un server riconosce che l'indirizzo è un programma CGI, il file non risponde con un documento ma eseguendo un programma. Ad esempio

GET /cgi-bin/welcome.pl HTTP/1.0

Accept: www/source

Accept: text/html

Accept: image/gif

User-Agent: Lynx/2.4 libwww/2.14

From: shishir@bu.edu

Questa GET identifica il file da recuperare come /cgi-bin/welcome.pl. Siccome il server è configurato per riconoscere tutti i file in cgi-bin come dei PROGRAMMI, capisce che il file deve essere ESEGUITO invece che essere trasferito.

L'output generato dal programma potrebbe essere ad esempio il seguente

HTTP/1.0 200 OK

Date: Thursday, 22-February-96 08:28:00 GMT

Server: NCSA/1.4.2

MIME-version: 1.0

Content-type: text/html

Content-length: 2000

<HTML>

<HEAD><TITLE>Welcome to Shishir's WWW Server!</TITLE></HEAD>

<BODY>

<H1>Welcome!</H1>

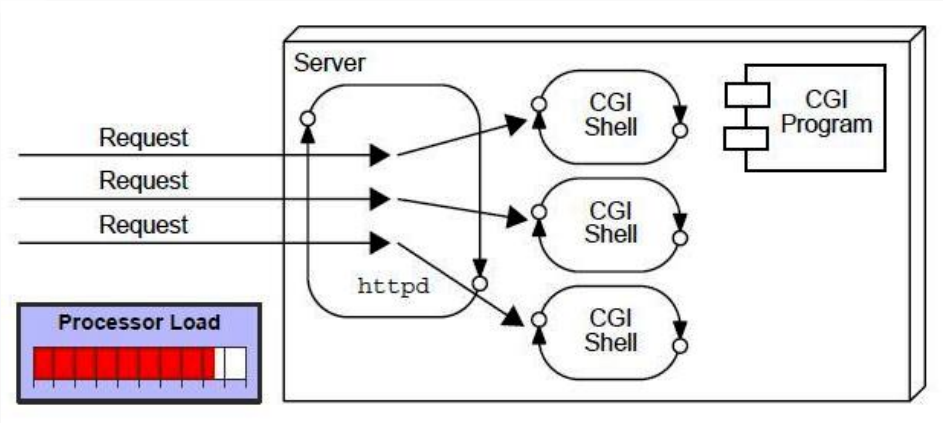
.

.

</BODY>

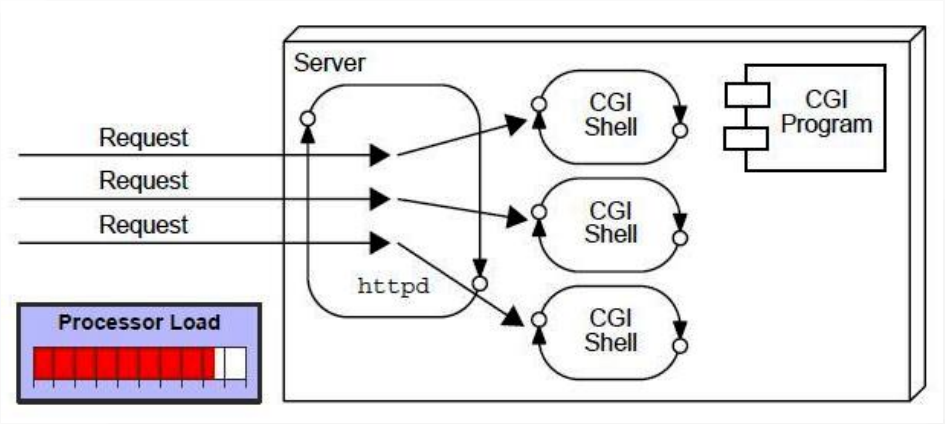
</HTML>

Dal CGI alle Servlet



Dal CGI alle Servlet

Poiché utilizzando CGI è possibile utilizzare un qualsiasi linguaggio di programmazione, anche un programma Java può essere eseguito dal server. Tuttavia eseguire un programma scritto in java ha le stesse limitazioni, per quanto riguarda CGI, di un qualunque altro linuguaggio di programmazione. In particolare

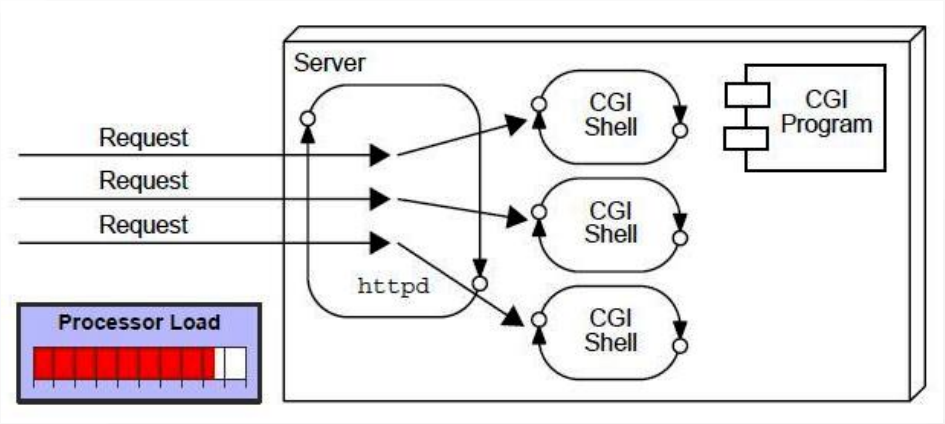


Dal CGI alle Servlet

Poiché utilizzando CGI è possibile utilizzare un qualsiasi linguaggio di programmazione, anche un programma Java può essere eseguito dal server. Tuttavia eseguire un programma scritto in java ha le stesse limitazioni, per quanto riguarda CGI, di un qualunque altro linuguaggio di programmazione. In particolare

Da Wikipedia (https://en.wikipedia.org/wiki/Java_servlet):

- ***Quando viene eseguita una richiesta HTTP viene creato un nuovo processo ogni volta che viene chiamato lo script CGI***
- ***l'overhead associato alla crazione del processo può dominare il carico di lavoro, speciamente quando lo script esegue operazioni relativamente veloci***
- ***La creazione del processo può impiegare più tempo dell'esecuzione dello script.***



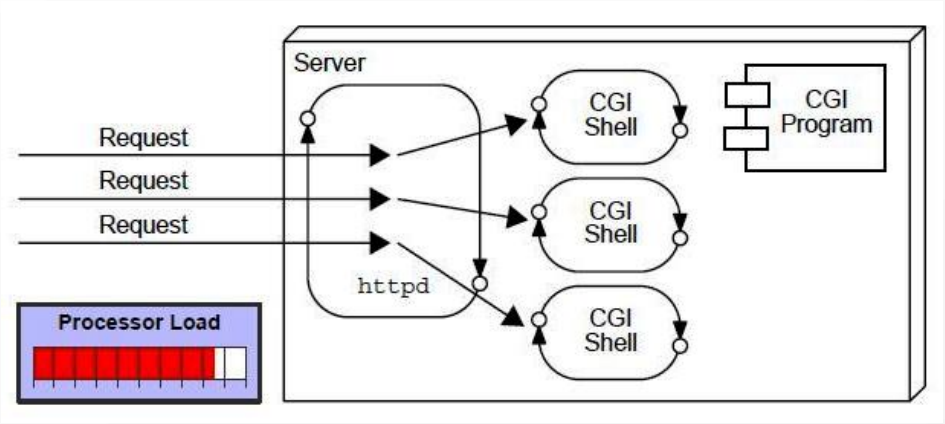
Dal CGI alle Servlet

Poiché utilizzando CGI è possibile utilizzare un qualsiasi linguaggio di programmazione, anche un programma Java può essere eseguito dal server. Tuttavia eseguire un programma scritto in java ha le stesse limitazioni, per quanto riguarda CGI, di un qualunque altro linuguaggio di programmazione. In particolare

Da Wikipedia (https://en.wikipedia.org/wiki/Java_servlet):

- ***Quando viene eseguita una richiesta HTTP viene creato un nuovo processo ogni volta che viene chiamato lo script CGI***
- ***l'overhead associato alla crazione del processo può dominare il carico di lavoro, speciamente quando lo script esegue operazioni relativamente veloci***
- ***La creazione del processo può impiegare più tempo dell'esecuzione dello script.***

Una servlet può essere eseguita da un contenitore in una sandbox.



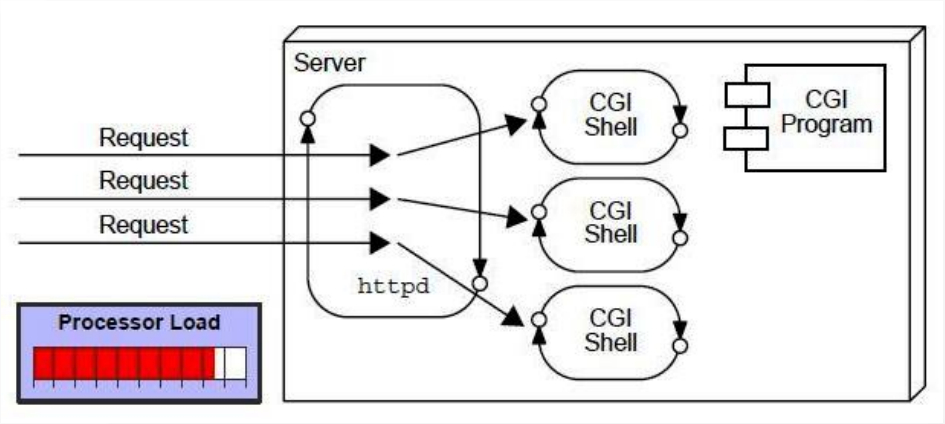
Dal CGI alle Servlet

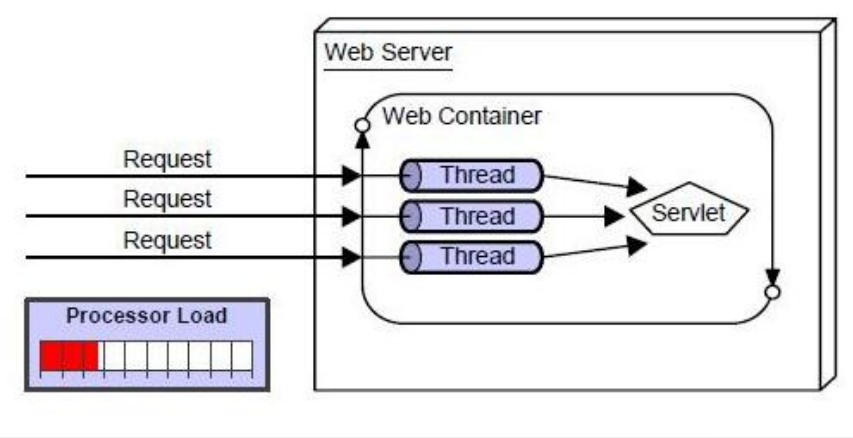
Poiché utilizzando CGI è possibile utilizzare un qualsiasi linguaggio di programmazione, anche un programma Java può essere eseguito dal server. Tuttavia eseguire un programma scritto in java ha le stesse limitazioni, per quanto riguarda CGI, di un qualunque altro linuguaggio di programmazione. In particolare

Da Wikipedia (https://en.wikipedia.org/wiki/Java_servlet):

- ***Quando viene eseguita una richiesta HTTP viene creato un nuovo processo ogni volta che viene chiamato lo script CGI***
- ***l'overhead associato alla crazione del processo può dominare il carico di lavoro, speciamente quando lo script esegue operazioni relativamente veloci***
- ***La creazione del processo può impiegare più tempo dell'esecuzione dello script.***

Una servlet può essere eseguita da un contenitore in una sandbox.





Osservazioni finali

Le Servlet hanno diversi vantaggi rispetto al CGI...

- **Una Servlet non "gira" in un processo separato. Ciò evita l'overhead di creazione di un nuovo processo per ogni richiesta**
- **Una Servlet rimane in memoria fra una richiesta e l'altra**
- **C'è una sola istanza che risponde a tutte le richieste concorrentemente. Ciò permette di risparmiare memoria e di gestire facilmente la persistenza dei dati**
- **Una Servlet può essere eseguita da un Servlet Engine in una Sandbox restrittiva che permette un uso sicuro di Servlet non affidabili e potenzialmente pericolose**

Osservazioni finali

Le Servlet hanno diversi vantaggi rispetto al CGI...

- **Una Servlet non "gira" in un processo separato. Ciò evita l'overhead di creazione di un nuovo processo per ogni richiesta**
- **Una Servlet rimane in memoria fra una richiesta e l'altra**
- **C'è una sola istanza che risponde a tutte le richieste concorrentemente. Ciò permette di risparmiare memoria e di gestire facilmente la persistenza dei dati**
- **Una Servlet può essere eseguita da un Servlet Engine in una Sandbox restrittiva che permette un uso sicuro di Servlet non affidabili e potenzialmente pericolose**

... ma anche qualche svantaggio (dal libro di testo)

Osservazioni finali

Le Servlet hanno diversi vantaggi rispetto al CGI…

- **Una Servlet non "gira" in un processo separato. Ciò evita l'overhead di creazione di un nuovo processo per ogni richiesta**
- **Una Servlet rimane in memoria fra una richiesta e l'altra**
- **C'è una sola istanza che risponde a tutte le richieste concorrentemente. Ciò permette di risparmiare memoria e di gestire facilmente la persistenza dei dati**
- **Una Servlet può essere eseguita da un Servlet Engine in una Sandbox restrittiva che permette un uso sicuro di Servlet non affidabili e potenzialmente pericolose**

… ma anche qualche svantaggio (dal libro di testo)

- **con le servlet si "mescola" la logica della applicazione con la presentazione;**

Osservazioni finali

Le Servlet hanno diversi vantaggi rispetto al CGI…

- **Una Servlet non "gira" in un processo separato. Ciò evita l'overhead di creazione di un nuovo processo per ogni richiesta**
- **Una Servlet rimane in memoria fra una richiesta e l'altra**
- **C'è una sola istanza che risponde a tutte le richieste concorrentemente. Ciò permette di risparmiare memoria e di gestire facilmente la persistenza dei dati**
- **Una Servlet può essere eseguita da un Servlet Engine in una Sandbox restrittiva che permette un uso sicuro di Servlet non affidabili e potenzialmente pericolose**

… ma anche qualche svantaggio (dal libro di testo)

- **con le servlet si "mescola" la logica della applicazione con la presentazione;**
- **se il codice HTML è complesso a volte è "particolarmente oneroso" doverlo "inglobare" all'interno del codice Java per mezzo di istruzioni di println;**

Osservazioni finali

Le Servlet hanno diversi vantaggi rispetto al CGI...

- **Una Servlet non "gira" in un processo separato. Ciò evita l'overhead di creazione di un nuovo processo per ogni richiesta**
- **Una Servlet rimane in memoria fra una richiesta e l'altra**
- **C'è una sola istanza che risponde a tutte le richieste concorrentemente. Ciò permette di risparmiare memoria e di gestire facilmente la persistenza dei dati**
- **Una Servlet può essere eseguita da un Servlet Engine in una Sandbox restrittiva che permette un uso sicuro di Servlet non affidabili e potenzialmente pericolose**

... ma anche qualche svantaggio (dal libro di testo)

- **con le servlet si "mescola" la logica della applicazione con la presentazione;**
- **se il codice HTML è complesso a volte è "particolarmente oneroso" doverlo "inglobare" all'interno del codice Java per mezzo di istruzioni di println;**
- **non è possibile la "prototipazione rapida": ogni modifica del codice richiede una ricompilazione esplicita e ogni aggiunta il cambiamento del file web.xml con il riavvio dell'applicazione o del server.**

Ciclo di vita di una servlet

init()

Metodo della classe madre che serve per impostare i parametri e l'ambiente di esecuzione

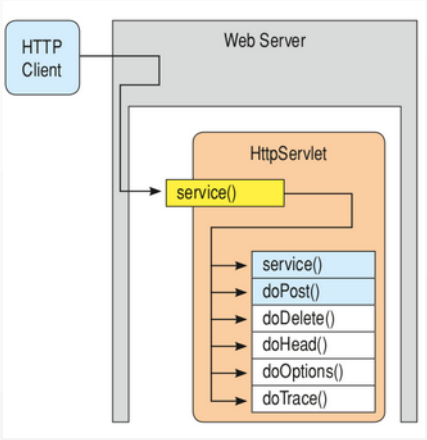
service()

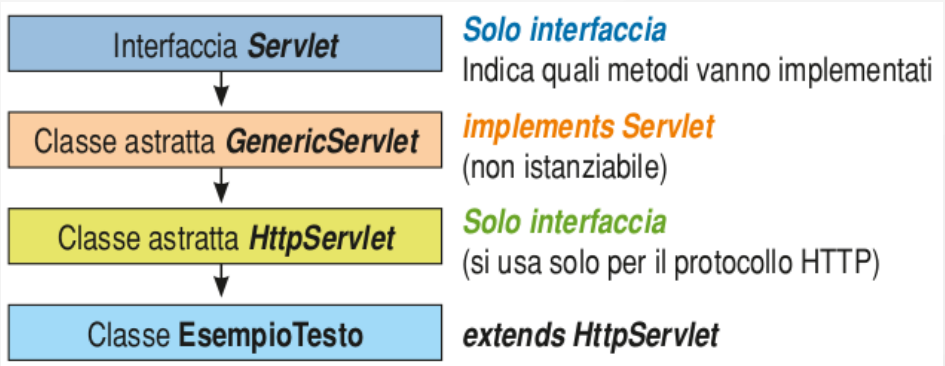
Metodo che implementa le operazioni che devono essere svolte dalla servlet ad ogni sua chiamata

- doGet()
- doPost()
- Vengono chiamati quando si utilizza il protocollo HTTP

destroy()

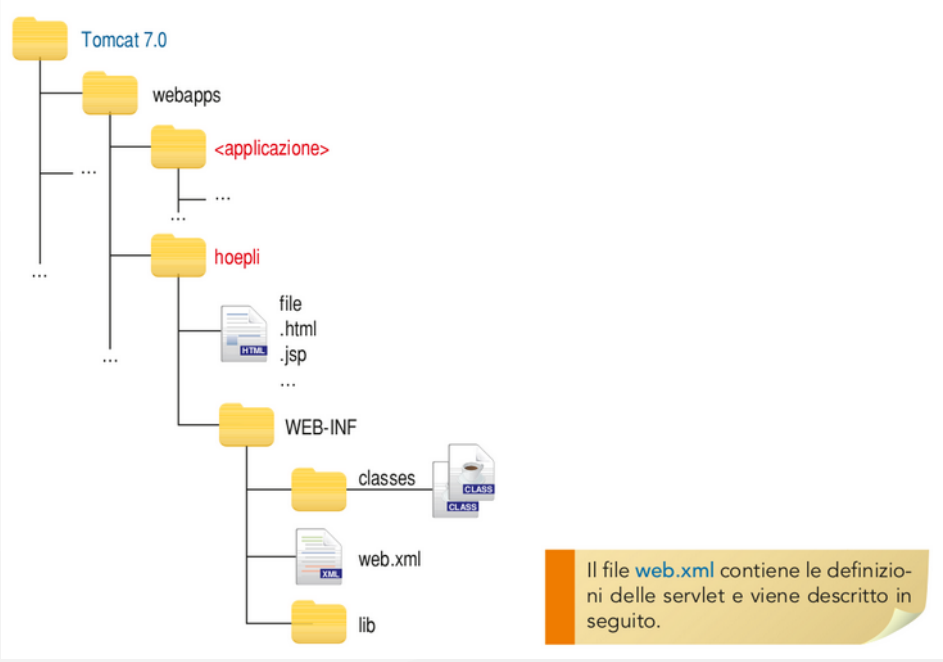
Metodo che si occupa di chiudere le servlet attive in maniera consistente





Primo Esempio

```java
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class primaServlet extends HttpServlet
  {
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse res)
              throws ServletException, IOException
    {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("<HTML><HEAD><TITLE>Hello Client!</TITLE>"+
                  "</HEAD><BODY>Hello Client!</BODY></HTML>");
      out.close();
    }

    public String getServletInfo()
    {
     return "HelloClientServlet 1.0 by Stefan Zeiger";
   }
}
```



riferimenti

http://www.novocode.com/doc/servlet-essentials/chapter1.html

https://www.w3.org/CGI/Overview.html

http://www.oreilly.com/openbook/cgi/index.html