Internet Applications (Client-Server Concept, Use of Protocol Ports, Socket API, DNS, E-mail, TELNET, FTP)

Funzionalità

Livello di trasporto e livelli sottostanti
- Comunicazione base
- Disponibilità

Livello di applicazione
- Astrazioni
  - Files
  - Servizi
  - Databases
- Nomi

Aspetti

Network
– Trasferisce bits
– Opera sulle richieste delle applicazioni

L'applicazione determina
– Cosa inviare
– Quando inviare
– Dove inviare
– Significato dei bits

Aspetti importanti

Un sistema Internet fornisce un servizio base di comunicazione; il protocollo SW da solo non può iniziare un contatto con, od accettare un contatto da, un altro computer. Infatti 2 programmi applicativi devono partecipare in ogni comunicazione: una applicazione inizia la comunicazione e l'altra la accetta.

Architettura di comunicazione Del paradigma Client-Server

Una applicazione
– Inizia la sua esecuzione prima
– Aspetta passivamente in una posizione predefinita

L'altra applicazione
– Inizia l'esecuzione dopo
– Contatta attivamente l'altro programma

– E' usato da tutte le applicazioni di rete
– Il programma passivo è chiamato Server
– Quello attivo Client

Comunicazioni Internet

Tutte le applicazioni Internet usano una forma di comunicazione chiamata Paradigma Client-Server. Una applicazione server attende passivamente di essere contattata, mentre quella client inizia attivamente la comunicazione.

Caratteristiche di un client

– Programma applicativo qualsiasi
– Diventa temporaneamente client
– Può fare anche altre cose
– Viene chiamato dall' utente
– Gira localmente sul computer dell' utente
– Inizia il contatto con il server
– Contatta un server alla volta

Caratteristiche di un server

– Programma Specializzato, con privilegi
– Dedicato a fornire un servizio
– Può gestire più client remoti contemporaneamente
– Richiamato automaticamente alla partenza del sistema
– Viene eseguito continuativamente
– Richiede computer e sistemi operativi potenti
– Aspetta passivamente di essere contattato dal client
– Accetta richieste da un qualsiasi client

Definizioni

Server
– Un programma eseguibile che accetta contatti attraverso la rete

Server-class computer
– Hardware sufficiente ad eseguire il server

N.B.
Il termine server viene spesso riferito al computer

Flusso dei dati

Direzioni
– Solo dal Client verso il Server
– Solo dal Server verso il Client
– In entrambe le direzioni

Il protocollo applicativo definisce il flusso

Scema classico
– Client manda la richiesta
– Il server risponde
Clients e servers sono programmi applicativi che operano a livello 5 dello Stack TCP/IP

Funzionamento Server

Server opera come le altre applicazioni
– Usa la CPU per eseguire le istruzioni
– Esegue operazioni di I/O
L' attesa di richieste dalla rete non richiede CPU

N.B.
Quindi il server usa la CPU solo quando riceve richieste, quindi abbiamo bisogno di HW potente solo per gestire molte richieste.

Possiamo avere più server sullo stesso computer.
– Ogni server offre un servizio
– Ogni server può gestire più client

Riconoscimento di un servizio

Si basa sul numero di porta usata da un protocollo, ed ogni servizio ha un unico numero di porta *P*.

Server
– Informa il SO che usa la porta *P*
– Aspetta che arrivino le richieste

Client
– Esegue le richieste
– Manda le richieste sulla porta *P* del server

N.B.
I protocolli di trasporto assegnano ad ogni servizio un'identificativo di porta univoco. Un server deve conoscere il suo numero di porta quando inizia l'esecuzione. Un client deve specificare l'identificativo quando chiede al protocollo di trasporto di contattare un server. Il trasporto sul server usa questo identificativo per dirigere le richieste che arrivano al Server applicativo giusto.

Caratteristiche

Formalmente:
– I numeri di porta sono semplici interi
– Ogni server potrebbe usare un qualsiasi numero di porta

In pratica
– I numeri di porta sono usati per identificare i vari servizi
– È necessaria una numerazione uniforme:
  – Per permettere ad un qualsiasi client di contattare un server su una qualsiasi macchina
  – Per evitare di dover avere un sistema di directory
– Numeri di porta
  – Uniformi in Internet
  – Definiti secondo convenzione

Server sequenziali e concorrenti

Un programma **sequenziale** ha un singolo thread di controllo, ed è una configurazione tipica della maggior parte dei programmi
Un programma **concorrente** ha molteplici thread di controllo, la sua esecuzione avviene in parallelo, ma è estremamente difficile da creare.

Un server sequenziale, detto anche iterativo, può eseguire una richiesta alla volta, quindi un client deve attendere che il server abbia completato di soddisfare le precedenti richieste.
Un server concorrente, invece può gestire più richieste alla volta, quindi non comporta attese per i client, infatti crea un nuovo thread di controllo per gestire ogni richiesta ed il client deve solo attendere che venga eseguita la sua richiesta.
N.B.: *La concorrenza è quindi fondamentale per i server, infatti consente che molteplici client possano ottenere un dato servizio senza dover attendere che vengano completate le richieste precedenti. Il thread principale del server crea un nuovo thread del servizio per gestire ogni client.*

Porte di protocollo e Server concorrenti

Problema:
– Un numero di porta assegnato ad ogni servizio
– Server concorrenti comportano che ci siano diversi thread
– Client e server devono interagire
– I messaggi inviati alla porta del server devono essere affidati al thread corretto.

Soluzione:
Usare le informazioni del client per inviare i pacchetti al thread corretto.
TCP usa 4 parametri per identificare la connessione
– Indirizzo IP del Server
– Numero di porta di protocollo del Server
– Indirizzo IP del Client
– Numero di porta di protocollo del Client

Demultiplexing in un server concorrente

Il protocollo di trasporto assegna un identificativo ad ogni client come per ogni servizio. Il Protocollo sul server usa la combinazione degli identificatori del server e del client per scegliere la giusta copia (Thread) del server concorrente.

Un server può usare
– Connectionless transport (UDP)
– Connection-oriented transport (TCP)
– Entrambi per un singolo servizio

Server particolari possono:
– offrire più servizi, di solito piuttosto semplici ed usare quindi diversi numeri di porta, per ogni servizio
– tenere una interazione con un client per giorni e/o ore
– mandare una breve risposta e chiudere l' interazione
– fare operazioni di I/O sul computer locale
– diventare un client per un altro servizio

Interazioni con il Protocollo

Client e server usano i protocolli di trasporto
– Protocollo software è dentro il SO
– Le applicazioni sono fuori dal SO
Il meccanismo necessario per collegare i 2 sono chiamati ***Application Program Interface (API)***

Application Program Interface

È parte del sistema operativo e permette alle applicazioni di usare i protocolli e definisce:
– Operazioni permesse
– Argomenti per ogni operazione

Socket API

È stato inizialmente progettato per UNIX BSD da usare con i protocolli TCP/IP ed ora è uno standard industriale disponibile su molti sistemi operativi.

Socket

– È un' astrazione del SO (non HW)
– vengono creati dinamicamente
– Persiste solo mentre gira l' applicazione
– Riferito da un descrittore che è:
  – un intero piccolo
  – uno per ogni socket attivo
  – usato in tutte le operazioni sul socket
  – generato dal sistema operativo quando viene creato il socket
  – serve solo all' applicazione che lo possiede
  – su UNIX è integrato con i descrittori dei files

Creazione del Socket

L' applicazione chiama una funzione *socket*

```
sdesc = socket(protofamily, type, proto)
```

Il SO restituisce il descrittore per il socket che è valido finchè l' applicazione lo chiude od esce.

Funzionalità del Socket

Il Socket è completamente generale può essere usato per molte operazioni:

– Dal client
– Dal server
– Con un CO transport protocol
– Con un CL transport protocol
– Per inviare, ricevere dati

Funzioni del Socket

Close: chiude e rilascia il socket
Bind: specifica la porta e l' indirizzo IP per un socket, con *INADDR_ANY*.
Listen: prepara il socket ad accettare le connessioni (usata dal server)
Accept: aspetta la prossima connessione e restituisce un nuovo socket (usata dal server)
Connect: specifica l' indirizzo per UDP oppure crea una connessione TCP (usata dal server)
*Sen*d, *sendt*o, e *sndmsg:* trasferisce i dati in uscita dall' applicazione
*Rec*v, *recvfro*m, e *recvmsg:* trasferisce i dati in ingresso all' applicazione

I due scopi della funzione *Connect*

La funzione connect, che è chiamata dai clients ha 2 usi. Con un trasporto CO connect *stabilisce la connessione ad uno specifico server. Con un trasporto CL registra l' indirizzo del server nel socket, permettendo al client di di mandare molti messaggi allo stesso server senza specificare l' indirizzo di destinazione in ogni messaggio.*

Funzioni di supporto al Socket

Esistono ulteriori funzioni di supporto ed utilità che vengono implementate come chiamate di libreria.

Gethostbyname: dato il nome di dominio del sistema (es. opac.unica.it) restituisce l' indirizzo IP associato

Getprotobyname: dato il nome del protocollo (es. tcp o udp) restituisce il numero del protocollo associato

Esempio di servizio

Scopo: conta quante volte è stato chiamato e rende un messaggio ASCII
Basato su un protocollo CO
Esecuzione sequenziale (non concorrente)

Esempio di Client

– Apre una connessione TCP al server
– Riceve un testo e lo stampa
– Chiude la connessione

Esempio di Server

– Crea un socket e passa in modalità passiva
– Ciclo iterativo:
  – Accetta una nuova connessione e crea un nuovo socket
  – Incrementa il contatore e manda il messaggio di testo
  – Chiude il socket per la connessione

NB
Il socket principale rimane aperto ed il Server non esce mai

Chiamate di funzione tra Client e Server

SERVER
getprotobyname
socket
bind
listen
ciclo: (accept
send
close)

CLIENT
gethostbyname
getprotobyname
socket
connect
ciclo: (recv)
close

NB
il Client chiude il socket dopo averlo usato, mentre il server non chiude mai il socket originale

Implementazione C del Client

Argomenti del programma: Host e porta di protocollo (entrambi opzionali)
NB: ci sono alcune incompatibilità tra le varie implementazioni del socket: Unix e Microsoft (si usa il costrutto C #ifdef)

```
/* client.c - code for example client program that uses TCP */
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#endif
#include <stdio.h>
#include <string.h>
#define PROTOPORT 5193 /* default protocol port number */
extern int errno;
char localhost[] = "localhost"; /* default host name */
/*------------------------------------------------------------------------*
Program: client
*
* Purpose: allocate a socket, connect to a server, and print all output
* Syntax: client [ host [port] ]
*
* host - name of a computer on which server is executing
* port - protocol port number server is using
*
* Note: Both arguments are optional. If no host name is specified,
```

```
* the client uses "localhost"; if no protocol port is
* specified, the client uses the default given by PROTOPORT.
*
*------------------------------------------------------------------------*/
main(argc, argv)
int argc;
char *argv[];
{
struct hostent *ptrh; /* pointer to a host table entry */
struct protoent *ptrp; /* pointer to a protocol table entry */
struct sockaddr_in sad; /* structure to hold an IP address */
int sd; /* socket descriptor */
int port; /* protocol port number */
char *host; /* pointer to host name */
int n; /* number of characters read */
char buf[1000]; /* buffer for data from the server */
#ifdef WIN32
WSADATA wsaData;
WSAStartup(0x0101, &wsaData);
#endif
memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
sad.sin_family = AF_INET; /* set family to Internet */
/* Check command-line argument for protocol port and extract */
/* port number if one is specified. Otherwise, use the default */
/* port value given by constant PROTOPORT */
if (argc > 2) { /* if protocol port specified */
port = atoi(argv[2]); /* convert to binary */
} else {
port = PROTOPORT; /* use default port number */
}
if (port > 0) /* test for legal value */
sad.sin_port = htons((u_short)port);
else { /* print error message and exit */
fprintf(stderr,"bad port number %s\n",argv[2]);
exit(1);
}
/* Check host argument and assign host name. */
if (argc > 1) {
```



```
host = argv[1]; /* if host argument specified */
} else {
host = localhost;
}
/* Convert host name to equivalent IP address and copy to sad. */
ptrh = gethostbyname(host);
if ( ((char *)ptrh) == NULL ) {
fprintf(stderr,"invalid host: %s\n", host);
exit(1);
}
memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
/* Map TCP transport protocol name to protocol number. */
if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
fprintf(stderr, "cannot map \"tcp\" to protocol number");
exit(1);
}
/* Create a socket. */
sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
fprintf(stderr, "socket creation failed\n");
exit(1);
}
/* Connect the socket to the specified server. */
if (connect(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
fprintf(stderr,"connect failed\n");
exit(1);
}
/* Repeatedly read data from socket and write to user's screen. */
n = recv(sd, buf, sizeof(buf), 0);
while (n > 0) {
write(1,buf,n);
n = recv(sd, buf, sizeof(buf), 0);
}
/* Close the socket. */
closesocket(sd);
/* Terminate the client program gracefully. */
exit(0);
}
```

Implementazione C del Server

Argomenti del programma: Host e porta di protocollo (entrambi opzionali)

```
/* server.c - code for example server program that uses TCP */
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#endif
#include <stdio.h>
#include <string.h>
#define PROTOPORT 5193 /* default protocol port number */
#define QLEN 6 /* size of request queue */
int visits = 0; /* counts client connections */
/*------------------------------------------------------------------------*
Program: server
*
* Purpose: allocate a socket and then repeatedly execute the following:
* (1) wait for the next connection from a client
* (2) send a short message to the client
* (3) close the connection
* (4) go back to step (1)
* Syntax: server [ port ]
*
* port - protocol port number to use
*
* Note: The port argument is optional. If no port is specified,
* the server uses the default given by PROTOPORT.
*
```



```
*------------------------------------------------------------------------*/
main(argc, argv)
int argc;
char *argv[];
{
struct hostent *ptrh; /* pointer to a host table entry */
struct protoent *ptrp; /* pointer to a protocol table entry */
struct sockaddr_in sad; /* structure to hold server's address */
struct sockaddr_in cad; /* structure to hold client's address */
int sd, sd2; /* socket descriptors */
int port; /* protocol port number */
int alen; /* length of address */
char buf[1000]; /* buffer for string the server sends */
#ifdef WIN32
WSADATA wsaData;
WSAStartup(0x0101, &wsaData);
#endif
memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
sad.sin_family = AF_INET; /* set family to Internet */
sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address */
/* Check command-line argument for protocol port and extract */
/* port number if one is specified. Otherwise, use the default */
/* port value given by constant PROTOPORT */
if (argc > 1) { /* if argument specified */
port = atoi(argv[1]); /* convert argument to binary */
} else {
port = PROTOPORT; /* use default port number */
}
if (port > 0) /* test for illegal value */
sad.sin_port = htons((u_short)port);
else { /* print error message and exit */
fprintf(stderr,"bad port number %s\n",argv[1]);
exit(1);
}
/* Map TCP transport protocol name to protocol number */
if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
fprintf(stderr, "cannot map \"tcp\" to protocol number");
exit(1);
```

```
}
/* Create a socket */
sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
fprintf(stderr, "socket creation failed\n");
exit(1);
}
/* Bind a local address to the socket */
if (bind(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
fprintf(stderr,"bind failed\n");
exit(1);
}
/* Specify size of request queue */
if (listen(sd, QLEN) < 0) {
fprintf(stderr,"listen failed\n");
exit(1);
}
/* Main server loop - accept and handle requests */
while (1) {
alen = sizeof(cad);
if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0) {
fprintf(stderr, "accept failed\n");
exit(1);
}
visits++;
sprintf(buf,"This server has been contacted %d time%s\n",
visits,visits==1?".":"s.");
send(sd2,buf,strlen(buf),0);
closesocket(sd2);
}
}
```


Interfaccia di streaming

Sorgente:
– Chiama *send* ripetutamente
– Specifica il numero di ottetti per chiamata

TCP
– Divide stream in segmenti

Destinatario:
– Chiama *recv* ripetutamente
– Riceve uno o più ottetti per chiamata
– Conto di zero significa "end of file"