T4
Accés a les aplicacions i continguts de la xarxa

Xarxes de Computadors i Aplicacions

PAU ARTIGAS, DAVID CARRERA i JORDI TORRES
Departament d'Arquitectura de Computadors
UPC, setembre - 2009

Accés a les aplicacions i continguts de la Xarxa

1. Visió general
2. Sockets en TCP
3. Sockets en UDP
4. Consideracions sobre sockets
5. Les crides de sockets
   - Realització d'un client TCP
   - Realització d'un servidor TCP
   - Realització d'una comunicació UDP
6. Invocació Remota

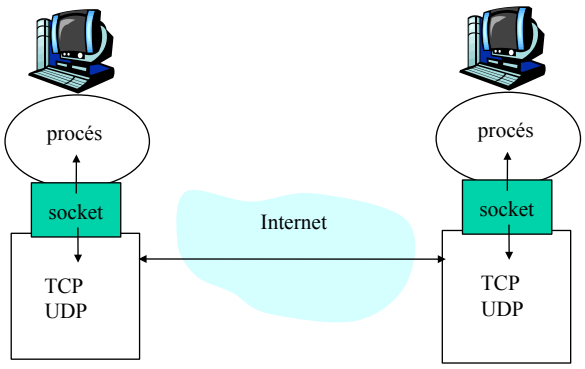
Annex: Sockets en Java

Computer Networking: A Top Down Approach Featuring the Internet,
2nd edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2002.

1. Visió general

- El nivell de transport (TCP o UDP) permet al nivell aplicació establir canals de comunicació entre dos hosts: **sockets**
- Aquests canals permeten fer lectures i escriptures
- Les dades que s'escriuen en un socket són rebudes a l'altra banda del canal de comunicació

1. Visió general



1. Visió general

- Cada procés que es vol comunicar crea el seu socket
- Cal diferenciar dos tipus de comunicacions:
  - Orientades a la connexió (TCP)
  - No orientades a la connexió (UDP)
- Les dos funcionen amb sockets, però el procediment de comunicació és diferent
- Els sockets es creen dinàmicament

1. Visió general

- Comunicacions orientades a la connexió (TCP):
  - Fiabilitat en la recepció de missatges
  - Es tallen els missatges en paquets
  - Manté l'ordre dels paquests
  - Utilitza un checksums per detectar errors
  - Hi ha una negocicació d'establiment de connexió inicial (handshake)

1. Visió general

- Comunicacions no orientades a la connexió (UDP):
  - No es fragmenten els missatges
  - Cap sincronització (ni control d'ordre)
  - En cas d'error, el missatge és retransmès des del nivell aplicació
  - Cap confirmació
  - No hi ha una negocicació d'establiment de connexió inicial (handshake)

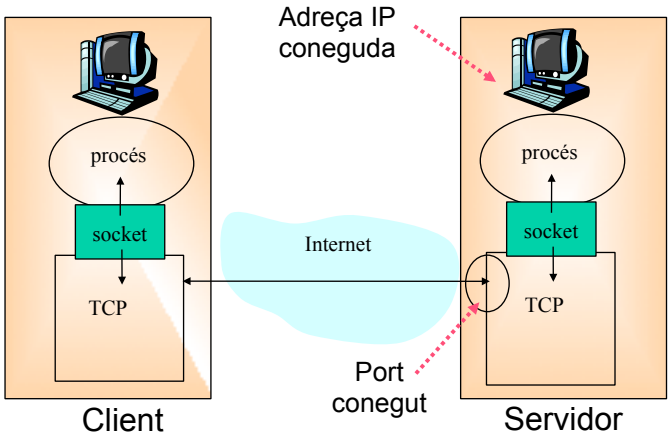
2. Sockets en TCP

- La interfície de sockets orientada a la connexió distingeix entre el rol de client i el de servidor
- Cada socket va associat a un identificador únic dins del host, anomenat **port**
- El port és l'entitat des d'on i cap a on s'estableix la connexió

2. Sockets en TCP

- El procés que té el rol de servidor, crea un socket associat a un port per on espera rebre peticions dels clients
- El procés que té el rol de client, coneix el número de port al que es vol connectar, a més de l'adreça IP de la màquina servidora (una adreça única a Internet)

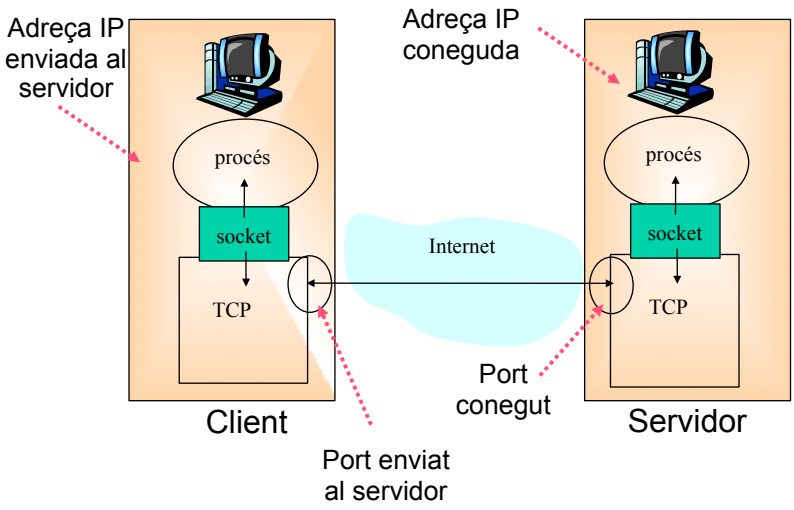
2. Sockets en TCP



2. Sockets en TCP

- El port que utilitza el client per a connectar-se al servidor s'assigna dinàmicament quan es crea el socket
- Durant el procés d'establiment de connexió, el port i l'adreça IP del client són enviats al servidor per a que aquest li pugui enviar les respostes

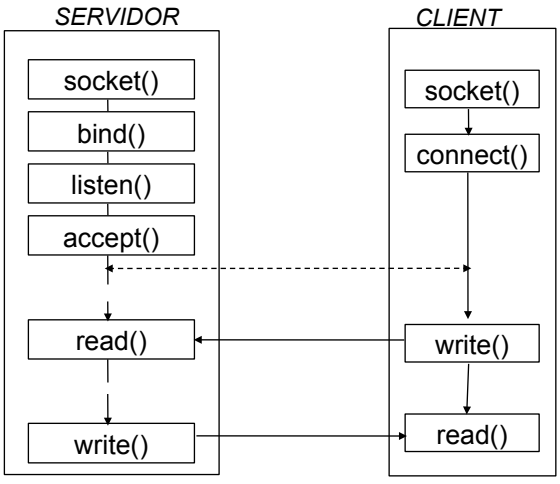
2. Sockets en TCP



2. Sockets en TCP

- Les dades que s'envien pel socket TCP es fragmenten en segments
- La comunicació es fa en forma **d'stream:**
  - Flux de dades (d'entrada o sortida a un procés)
  - Es reben en el socket destinació d'una comunicació en el mateix ordre en què han estat escrits en el socket origen

2. Sockets en TCP



2. Sockets en TCP

- Passos per a crear un servidor:
  - Crear el socket
  - Vincular el socket a un port (conegut)
  - Preparar el socket per a rebre connexions
  - Acceptar una petició de connexió
  - Comunicar-se (*)
  - Tancar el socket

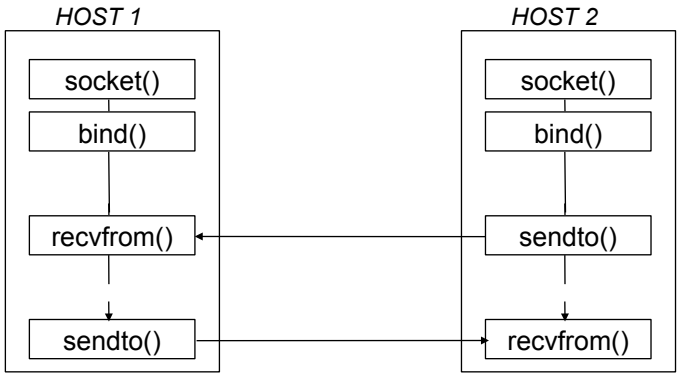
2. Sockets en TCP

- Passos per a crear un client:
  - Crear el socket
  - Sol·licitar la connexió al sevidor
    - Indicant l'adreça IP del servidor i el port al que es vol establir la connexió
  - Comunicar-se (*)
  - Tancar el socket

3. Sockets en UDP

- No existeix el concepte de connexió entre els dos equips
- Cada paquet que s'envia (datagrama) és idependent dels anteriors i no es fragmenta
- No hi ha garanties sobre la recepció ni sobre l'ordre de recepció
- Els errors (de recepció o d'ordre) els ha de detectar i solventar l'aplicació

3. Sockets en UDP

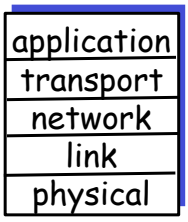


3. Sockets en UDP

- Passos per a realitzar una comunicació:
  - Crear el socket
  - Vincular el socket a un port
  - Comunicar-se (*)
    - Indicant l'adreça IP i el port de destí de la comunicació
  - Tancar el socket

4. Consideracions sobre sockets

- Els sockets no estan vinculats a TCP o UDP
- Són mecanismes de comunicació genèrics que s'adapten al nivell 4 de la torre OSI
  - Locals a la màquina (UNIX)
  - Entre hosts (Internet)
  - ...
- Usats per internament en els sistemes operatius (pipes)



4. Consideracions sobre sockets

- Poden ser:
  - Actius: quan inicien la comunicació
  - Passius: quan esperen la inicialització d'una comunicació
- Són locals al procés que els crea, però el port al que estan vinculats és únic a la màquina
- Des del punt de vista de les operacions d'E/S del sistema, són un *file descriptor*

4. Consideracions sobre sockets

- Les dades es poden ordenar en memòria en ordres diferents:
  - Big endian / Little endian
- Les dades també s'han d'ordenar en la xarxa
- Cal que l'ordre en què es posen a la xarxa i en memòria siguin consistents
  - htons, ntohs, htonl i ntohl

5. Les crides de sockets

1. Realització d'un client TCP
2. Realització d'un servidor TCP
3. Realització d'una comunicació UDP

5.1. Realització d’un client TCP

1.Creació d'un socket
2. Adreces
3. Petició de connexió
4. Exemple de codi client

5.1.1 Creació d'un socket

```
#include <sys/types.h>
#include <sys/socket.h>
int s = socket(int domini,
               int tipus,
               int protocol);
```

- La crida retorna un descriptor de fitxer que identifica el socket creat
- Domini: (família de protocol) depèn de l'entorn on es trobi treballant l'aplicació. Aquest paràmetre descriu la naturalesa o tipus de xarxa sobre la qual es treballa.
  - AF_UNIX o PF_UNIX Protocols interns d'UNIX
  - AF_INET o PF_INET Protocols DARPA d'Internet
  - AF_NS o PF_NS Protocols Xerox NS

5.1.1 Creació d'un socket

- Tipus: indica el tipus de socket que volem crear:
  - SOCK_STREAM (Stream socket): Aquests sockets són orientats a la connexió, és a dir, proporcionen un flux de dades bidireccional i fiable (sense pèrdues i sense duplicació d'informació). Les dades es reben amb el mateix ordre amb què van ser enviades.
  - SOCK_DGRAM (Datagram socket): Aquests sockets són no orientats a la connexió, és a dir, els missatges es fragmenten en datagrames de mida fixa, i normalment petita. Els datagrames no tenen perquè ser rebuts en el mateix ordre en què es van enviar. El flux de datagrames no és fiable (poden produir-se pèrdues i duplicacions de datagrames) i és bidireccional.
- Protocol: Se sol posar un 0 deixant al sistema la feina de triar el protocol més adient amb el tipus i el domini del socket indicats.

5.1.2 Adreces

- Una adreça és una estructura de dades que s'ha d'omplir manualment abans d'assignar-la al socket. El format d'una adreça varia considerablement depenent del domini al qual pertany:
  - AF_INET

```
struct sockaddr_in {
    short          sin_family;         /* AF_INET */
    u_short   sin_port;           /* port */
    struct in_addr      sin_addr; /* @IP */
    char      sin_zero[8];        /* es posa a 0*/
};
```

  - AF_UNIX

```
struct sockaddr_un {
    short sun_family;           /* AF_UNIX */
    char sun_path[108];         /* pathname */
};
```

5.1.3 Petició de connexió

```
#include <sys/types.h>
#include <sys/socket.h>
int connect (int s,
             struct sockaddr *addr,
             int addr_len );
```

- s: descriptor del socket al qual s'ha d'assignar l'adreça *addr
- *addr: adreça del servidor amb qui es vol connectar
- addr_len: longitud de *addr
- resultat: si tot ha anat bé retorna 0. Altrament, retorna -1 i errno codifica l'error.

Conversió d'adreces

- Si ens hi fixem bé, el sistema no espera l'adreça IP en format decimal sinó que espera un u_int. Es pot utilitzar la crida a inet_addr(). Per exemple:

```
adreca.sin_addr.s_addr = inet_addr("147.83.43.10");
```

- També existeix la crida inversa (inet_ntoa()), que a partir d'una adreça del tipus u_int la converteix a cadena decimal

5.1.4 Exemple de codi client

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define SIZE 1024
char buf[SIZE];
#define TIME_PORT 13

int main(argc, argv)
 int argc;
 char *argv[];
{
 int sockfd;
 int nread;
 struct sockaddr_in serv_addr;

if (argc != 2) {
 fprintf(stderr, "%s IPaddr\n", argv[0]);
 exit(1);
 }
```

5.1.4 Exemple de codi client

```
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0))<0) {
        perror(NULL);
        exit(2);
 }

serv_addr.sin_family = AF_INET;
 serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
 serv_addr.sin_port = htons(TIME_PORT);
 if (connect(sockfd,
                 (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {

    perror(NULL);
    exit(3);
 }
 nread = read(sockfd, buf,  SIZE);
 printf("%s\n", buf);
 close(sockfd);
}
```

5.2. Realització d'un servidor TCP

1. Ressolució de noms
2. Vinculació a un port
3. Mode escolta
4. Acceptar connexions
5. Exemple de codi servidor

5.2.1 Ressolució de noms

- gethostbyname() retorna certa informació, entre la qual en destaca l'adreça, a partir del nom de la màquina,

```
char *host = "www.ac.upc.es";
struct hostent *hp;
hp = gethostbyname(host);
```

- Aquesta crida retorna una estructura del tipus struct hostent, definida a <netdb.h>:

```
struct hostent
{
  char *h_name;                 /* Official name of host.  */
  char **h_aliases;             /* Alias list.  */
  int h_addrtype;               /* Host address type.  */
  int h_length;                 /* Length of address.  */
  char **h_addr_list;           /* List of addresses */
  #define h_addr  h_addr_list[0]  /* Address */
};
```

5.2.1 Ressolució de noms

- Ús de gethostbyname()

```
struct hostent* host;
host = gethostbyname ("www.ac.upc.edu");
if (host)
      serv_addr.sin_family = AF_INET;
      memcpy(&serv_addr.sin_addr.s_addr, host->h_addr, host->h_length);
      serv_addr.sin_port = htons(80);
}
```

5.2.1 Ressolució de noms

- Quan no coneixem el nom de la màquina on estem executant: gethostname() retorna el nom oficial de la màquina sobre la qual s'està executant

```
#include <unistd.h>
int err= gethostname(char *nom_host, size_t nom_hostlon);
```

  - *nom_host: nom oficial de la màquina on s'ha executat la crida
  - nom_hostlon: longitut de *nom_host

5.2.2 Vinculació a un port

```
int err = bind(int s, struct sockaddr *adr, int adrlon);
```

  - s: descriptor del socket al qual assignem l'adreça
  - adr: adreça assignada, d'acord amb el domini indicat a la crida socket(), però fent una conversió de tipus a struct sockaddr. Per exemple, declarat al domini d'AF_INET la variable struct sockaddr_in adrec tenim:

```
err = bind( s, (struct sockaddr *) &adrec, sizeof(adrec) )
```

  - adrlon: longitud de l'adreça *adr

5.2.3 Mode escolta

```
int error = listen (int s, int maxcua);
```

- s: descriptor del socket que rep les peticions de connexió
- maxcua: nombre màxim permès de peticions de connexió que esperen ser ateses. No pot superar el valor indicat per la constant SOMAXCONN, que es troba a <sys/socket.h>

5.2.4 Acceptar connexions

```
int snou = accept (int s, struct sockaddr *client_adr,
   int *client_adrlon);
```

- s: descriptor del socket que rep les peticions de connexió
- *client_adr: adreça del client amb qui s'ha establert la connexió (l'omple el sistema)
- *client_adrlon: longitud de *client_adr. És un paràmetre d'e/s: abans de la crida s'ha d'inicialitzar amb l'espai que ocupa una estructura del tipus de *client_adr. (sizeof(*client_adr)). Quan torna de la crida conté la grandària real de *client_adr
- snou: **retorna el descriptor del nou socket creat per a aquesta connexió acceptada**. Té les mateixes característiques que el socket s.

5.2.4 Acceptar connexions

- Comentaris
  - Si no ens interessés obtenir l'adreça del client, al camp client_adr posaríem un NULL.
  - Si no hi ha cap petició de connexió pendent i el socket no té activat el mode O_NDELAY (no bloquejant), la crida accept() es bloqueja fins que arribi alguna petició de connexió. Si té activat el mode O_NDELAY, llavors la crida accept() acaba immediatament i retorna un error.
  - El procés servidor té un socket per on pot anar rebent les peticions de connexió de tots els clients i un socket més per cada connexió que ha anat acceptant. És mitjançant cada un d'aquests nous sockets per on es fan les transferències entre el servidor i els clients. El socket inicial s, creat amb la crida socket(), només s'usa per gestionar la cua de peticions de connexió al servidor.

5.2.5 Exemple de codi servidor

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(argc,argv)
   int argc;
   char *argv[];
          {
           int sockfd, client_sockfd;
           int nread, len;
           struct sockaddr_in serv_addr,
           client_addr;
           time_t t;

           if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
           perror(NULL);
           exit(2);
           }
```

5.2.5 Exemple de codi servidor

```
 serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(TIME_PORT);

if (bind(sockfd,
        (struct sockaddr *)&serv_addr, sizeof(serv_addr))<0)
 {
   perror(NULL);
   exit(2);
 }
listen(sockfd, 5);
for (;;) {
    len = sizeof(struct sockaddr_in);
   client_sockfd = accept(sockfd,
                (struct sockaddr *)&client_addr, &len);
   time(&t);
   sprintf(buf, "%s", asctime(localtime(t)));
   len = strlen(buf) + 1;
   write(client_sockfd, buf, len);
   close(client_sockfd);
}
}
```

5.3. Realització d'una comunicació UDP

1. Crides equivalents a les de TCP
2. Crides addicionals a les de TCP
3. Exemple d'una comunicació UDP

5.3.1 Crides equivalents a les de TCP

- socket

```
int s = socket(int domini,
               int tipus,
               int protocol);
```

  - Camp tipus: SOCK_DGRAM

- bind

```
int err = bind(int s,
                  struct sockaddr   *adr,
               int adrlon);
```

5.3.2 Crides addicionals a les de TCP

```
#include <sys/types.h>
#include <sys/socket.h>

int res = sendto  (int s, char *buffer, int
  num_bytes, int flags,struct sockaddr *adr_desti,
  int adr_destilon);

int res = recvfrom(int s, char *buffer, int
  num_bytes, int flags,struct sockaddr
  *adr_origen, int *adr_origenlon);
```

5.3.2 Crides addicionals a les de TCP

- s: descriptor on s'escriu i d'on es llegeix
- *buffer: buffer d'on es poden escriure/llegir les dades
- num_bytes: núm. de bytes que s'han d'escriure/ llegir al/del *buffer
- flags: pot valer 0 o bé una combinació de diversos valors
- *adr_desti: adreça on s'han d'enviar les dades
- adr_destilon: longitud de *adr_desti

5.3.2 Crides addicionals a les de TCP

- *adr_origen: retorna l'adreça de qui va enviar les dades
- *adr_origenlon: retorna la longitud de *adr_origen. És un paràmetre d'entrada/ sortida: abans de la crida s'ha d'inicialitzar amb l'espai que ocupa una estructura del tipus de *adr_origen (*adr_origenlon = sizeof(*adr_origen)). Quan torna de la crida conté la grandària real de *adr_origen
- res: si tot ha anat bé retorna la longitud de les dades que han estat escrites/ llegides. recvfrom retorna la longitud del datagrama rebut. Si la transmissió ha anat malament, retorna -1 i errno codifica l'error

5.3.3 Exemple d'una comunicació UDP

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>

#define SIZE 1400
char buf[SIZE];

#define TIME_PORT 13

int main(int argc, char *argv[])
{
 int sockfd;
 int nread;
 struct sockaddr_in serv_addr,
 client_addr;
 int len;
```

5.3.3 Exemple d'una comunicació UDP

```
if (argc != 2) {
fprintf(stderr, "error: %s \n", argv[0]); exit(1);
 }

 if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0))  < 0) {
 perror(NULL); exit(2);
 }
 client_addr.sin_family = AF_INET;
 client_addr.sin_addr.s_addr = htonl(INADDR_ANY);
 client_addr.sin_port = htons(0);

 serv_addr.sin_family = AF_INET;
 serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
 serv_addr.sin_port =  htons(TIME_PORT);
```

5.3.3 Exemple d'una comunicació UDP

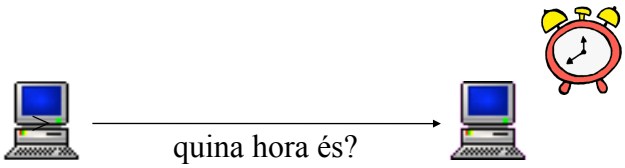
```
if (bind(sockfd, (struct sockaddr *)&client_addr,
        sizeof(client_addr)) < 0) {
 perror(NULL);
 close(sockfd);
 exit(3);
 }
len = sizeof(serv_addr);
 sendto(sockfd, buf, 1, 0, (struct sockaddr *)&serv_addr, len);

nread = recvfrom(sockfd, buf, SIZE, 0, (struct sockaddr *)&client_addr,
   &len);
 write(1, buf, nread);

 close(sockfd);
 exit(0);
}
```

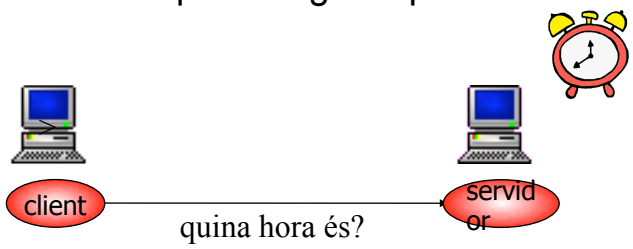
6. Invocació remota

- Volem executar un mètode en un equip remot:



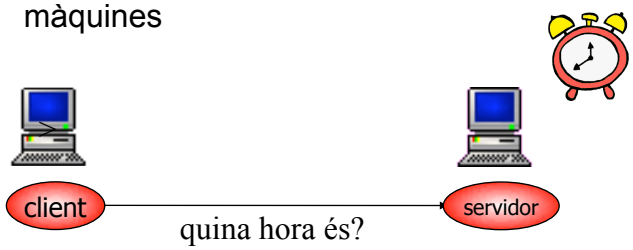
6. Invocació remota

- Necessitem:
  - Un programa que s'executi a la màquina client i que faci la petició
  - Un programa que s'executi en la màquina servidora i que atengui la petició



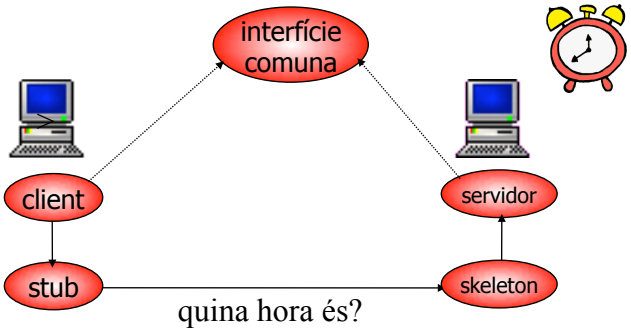
6. Invocació remota

- Volem:
  - Poder executar aquesta invocació
    - sense saber quina màquina ho proporciona (només sabent el nom del mètode a executar)
    - sense tenir en compte la xarxa que uneix les màquines



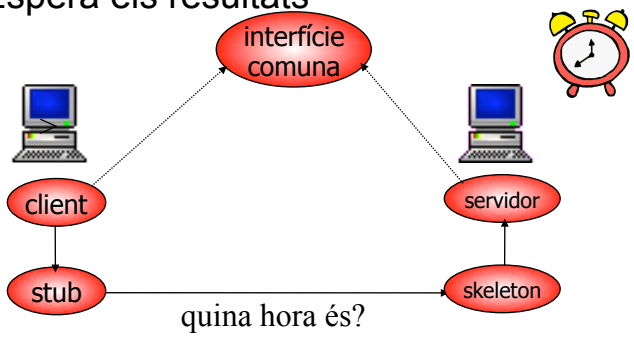
6. Invocació remota

- Requisit:
  - Tenir una interfície del mètode a invocar
  - Tenir un codi que localitzi la màquina servidora i en gestioni l'accés per xarxa de manera transparent



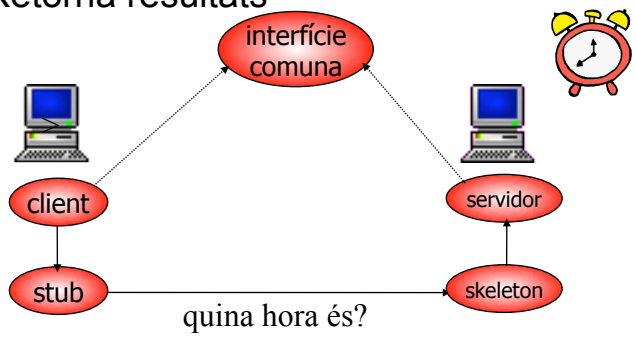
6. Invocació remota

- Stub:
  - Rep una crida local
  - Empaqueta els paràmetres
  - Els envia al destí
  - Espera els resultats



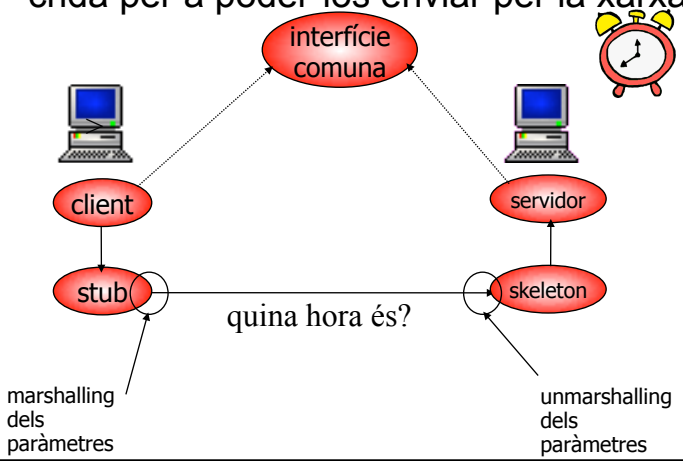
6. Invocació remota

- Skeleton:
  - Rep els paràmetres remots
  - Desempaqueta els paràmetres
  - Fa una crida local
  - Retorna resultats



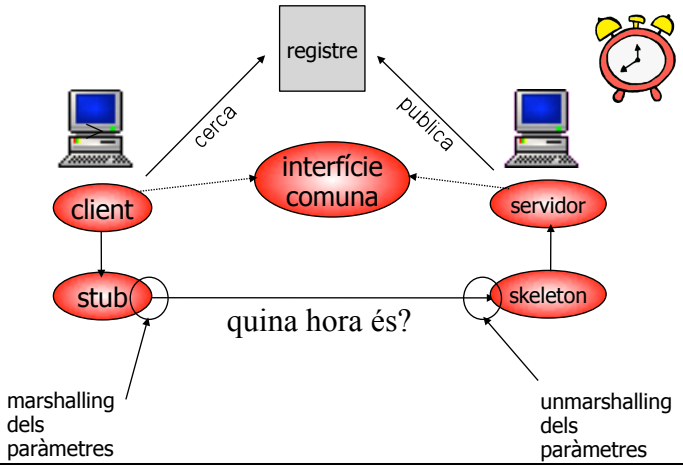
6. Invocació remota

- Marshalling:
  - Procés d'empaquetar els paràmetres d'una crida per a poder-los enviar per la xarxa



6. Invocació remota

- Localització:
  - Com trobar un mètode que es vol invocar?
    - Registre públic



6. Invocació remota

- Plataformes de computació distribuida:
  - Sun RPC (+XDR)
  - RMI
  - SOAP (i XML-RPC)
  - CORBA
  - DCOM
  - J2EE
  - .NET
  - Web Services
  - ...

ANNEX: SOCKETS A JAVA

- A Java també tenim la possibilitat de comunicar màquines diferents mitjançant sockets a l'estil *UNIX*.
- Aquesta comunicació l'aconsegueix a partir d'una funcionalitat encapsulada en bàsicament 4 classes:
  - *Socket*,
  - *ServerSocket*,
  - *DatagramSocket*
  - *SocketImpl*.

Les classes

- *Socket* :

  Objecte bàsic en una comunicació Internet. Serveix per establir comunicacions fiables orientades a la connexió (protocol TCP) entre programes executant-se en nodes a la xarxa Internet.

- *ServerSocket:*

  Classe encarregada de fer el *bind*, *listen* i *accept* pel servidor. El *bind* s'estableix en el moment de creació de la classe, passant-li el número de port sobre l'adreça de la màquina en què es troba. Al constructor també se li pot passar la longitud màxima de la cua d'espera de connexions per fer *accept* (paràmetre del listen en els sockets d'*UNIX*). En fer un *accept* retorna una instància de la classe *Socket* per on es durà a terme la comunicació amb el client.

Les classes

- *DatagramSocket*:

  Classe encarregada de realitzar comunicacions no fiables, no orientades a la connexió (protocol UDP). La unitat d'enviament és un paquet, simbolitzat en una instància de l'objecte *DatagramPacket*.

- *SocketImpl*:

  Classe abstracta per poder definir qualsevol tipus de comunicació. És a dir, podem crear una subclasse de *SocketImpl*, redefinint els seus serveis per tenir un control de la comunicació; per exemple, si volguéssim implementar un *firewall*, hauríem de redefinir el mètode *accept()* afegint els controls de seguretat necessaris.

  Les dues primeres classes "implementen" la classe *SocketImpl*.

Les funcionalitats

- Sockets
  - constructors:
    - *public* ***Socket****(InetAddress address, int port);*
    - *public* ***Socket****(InetAddress address, int port, boolean stream);*
    - *public* ***Socket****(String host, int port);*
    - *public* ***Socket****(String host, int port, boolean stream);*

    - Cada un dels constructors anteriors crea un socket a l'adreça i port donats.
    - Els paràmetres són els següents:

      - address: Adreça Inet de la màquina.

      - port: Port de la màquina al qual s'associarà el servei.

      - host: El nom del host.

      - stream: Si val cert crea un socket de tipus STREAM, si no, tipus DATAGRAM.

Les funcionalitats

- Sockets
  - serveis:
    - *public void **close**();* Tanca el socket.
    - *public InetAddress **getInetAddress**();* Torna l'adreça Inet a la que està connectat el socket.
    - *public InputStream **getInputStream**();* Torna el "canal" de lectura del socket.
    - *public int **getLocalPort**();* Torna el número de port local de la màquina sobre el qual s'ha fet el *bind*.
    - *public OutputStream **getOutputStream**();* Torna el "canal" d'escriptura del socket.
    - *public int **getPort**();* Torna el número de port al qual s'ha connectat el socket.

Les funcionalitats

- ServerSocket
  - constructors:
    - *public **ServerSocket**(int port);*
    - *public **ServerSocket**(int port, int count);*

    - Defineixen un socket de servidor amb:
      - port: Número de port a associar el servei. Si val 0, el tria sol.
      - count: Llargada màxima de la cua d'espera d'accept. el primer constructor és 50.

Les funcionalitats

- ServerSocket
  - serveis:
    - *public Socket **accept**();*

      Es bloqueja fins que arriba una nova connexió. Retorna un nou Socket per on es durà a terme la comunicació. S arriba una connexió nova que s'ha d'encuar a la cua d'accept i aquesta està plena, serà refusada.
    - *public void **close**();* Tanca el socket.
    - *public InetAddress **getInetAddress**();* Torna l'adreça Inet a la qual està connectat el socket.
    - *public int **getLocalPort**();* Torna el número de port local de la màquina sobre el qual s'ha fet el *bind*.

Les funcionalitats

- DatagramSocket
  - constructors:
    - public **DatagramSocket**();
    - public **DatagramSocket**(int port);

    - Creen un socket de tipus DATAGRAM, sobre la màquina local.
    - *Port* indica el port sobre el qual es pot fer el *bind*. En el primer cas en tria un de lliure.

Les funcionalitats

- DatagramSocket
  - serveis:
    - public void **close**(); Tanca el socket.
    - protected void **finalize**(); Allibera els recursos utilitzats pel socket una vegada no hagi ningú que l'utilitzi.
    - public int **getLocalPort**(); Torna el número de port local de la màquina sobre el qual s'ha fet el *bind*.
    - public void **receive**(DatagramPacket p); Llegeix un paquet del socket. El paquet conté, a més de les dades i la mida, l'adreça IP i el port de la màquina del qual l'ha enviat. És una crida bloquejant.
    - public void **send**(DatagramPacket p); Envia un paquet pel socket. El paquet ha de contenir l'adreça IP i el port de la màquina destí, així com les dades i la mida d'aquestes.

Les funcionalitats

- SocketImpl
  - constructor:
    - *public **SocketImpl**();*
  - camps:
    - *protected InetAddress **address**;*
    - *protected FileDescriptor **fd**;*
    - *protected int **localport**;*
    - *protected int **port**;*

Les funcionalitats

- SocketImpl
  - serveis: *(Els serveis abstract són els que s'han de redefinir)*
    - *protected abstract void **accept**(SocketImpl s);*
    - *protected abstract int **available**();*
    - *protected abstract void **bind**(InetAddress host, int port);*
    - *protected abstract void **close**();*
    - *protected abstract void **connect**(InetAddress address, int port);*
    - *protected abstract void **connect**(String host, int port);*

Les funcionalitats

- SocketImpl
  - serveis: *(continuació)*
    - *protected abstract void **create**(boolean stream);*
    - *protected FileDescriptor **getFileDescriptor**();*
    - *protected InetAddress **getInetAddress**();*
    - *protected abstract InputStream **getInputStream**();*
    - *protected int **getLocalPort**();*
    - *protected abstract OutputStream **getOutputStream**();*
    - *protected int **getPort**();*
    - *protected abstract void **listen**(int count);*

Exemple

- L'exemple següent mostra la realització d'un servidor i d'un client de telnet en mode orientat a la connexió.
- Cada vegada que el servidor rep una connexió, engega un thread que s'encarrega de servir la petició pel socket que torna l'*accept*.

TelnetServer.java

```
import java.net.ServerSocket;
import java.net.Socket;
import java.io.IOException;

public class TelnetServer extends Thread {
    protected ServerSocket server_listening_socket;

    public static void fail(Exception exception, String msg) {
          System.err.println(msg + "." + exception);
          System.exit(1);
    }

    public TelnetServer() {
          try {
                    server_listening_socket = new ServerSocket(23, 5);
          } catch (IOException ioexception)
                    fail(ioexception, "No puc iniciar el servidor de telnet");
          System.out.println("Servidor de telnet iniciat");
          this.start(); //inicia el thread per rebre peticions
    }
```

TelnetServer.java (cont.)

```
// codi que executa el thread que realitza un bucle infinit en espera // de
peticions de connexió
public void run() {
      try {
           while(true) {
                Socket client_socket= server_listening_socket.accept();
                TelnetConnection connection =
                          new TelnetConnection (client_socket);
            }
      }catch (IOException e)
                fail(e, "Error al bucle d'escoltar connexions");
}


// Codi principal del programa servidor
public static void main(String[] args) {
      new TelnetServer();
}
```

TelnetConnection.java

```
import java.net.Socket;
import java.io.IOException;
import java.io.DataInputStream;
import java.io.PrintStream;

class TelnetConnection extends Thread {
    protected Socket telnet_client;
    protected DataInputStream from_client;
    protected PrintStream to_client;

    public TelnetConnection (Socket client_socket) {
          telnet_client = client_socket;
          try {
          from_client=new DataInputStream(telnet_client.getInputStream());
          to_client=new PrintStream(telnet_client.getOutputStream());
          }catch (IOException e) {
                    //en cas d'error tancar la connexió
                    try telnet_client.close();
                    catch (IOException anIOException);
                    System.err.println("Error obtenint canals del socket"+e);
                    return;
          }
          this.start(); // iniciar el thread
    }
```

TelnetConnection.java (cont.)

```
    public void run() {
          String login_id; String password; int len;
          try {
              for(;;) {
                    to_client.println("Login: "); //demana el login
                    login_id=from_client.readLine();
                    if(login_id==null) break;
                    to_client.println("Password: "); //demana el password
                    password=from_client.readLine();
                    if(password==null) break;
                    //Ara s'hauria de comprovar les dades
                    //Una vegada comprovades, s'envia un missatge
                    to_client.println("Entri la comanda: ");
                    //fer el servei
                    break;
                }
          }catch(IOException e);
          finally try telnet_client.close(); catch (IOException ioexception);
    }
}
```

TelnetClient.java

```
import java.net.Socket;
import java.io.IOException;
import java.io.DataInputStream;
import java.io.PrintStream;

public class TelnetClient {

    public static void main(String [] args) {
          Socket s=null;

          try {
                    s= new Socket("alabi_fddi.upc.es", 23);
                    DataInputStream server_in=
                              new DataInputStream(s.getInputStream());
                    PrintStream server_out=
                              new PrintStream(s.getOutputStream());

                    DataInputStream input=new DataInputStream(System.in);
```

TelnetClient.java (cont.)

```
                String line;

                while(true) {
                        line=server.in.readLine();
                        if(line==null) break;
                        System.out.println(line);

                        line=input.readLine();
                        if(line==null) break;
                        server_out.println(line);
                }
        }catch (IOException e) System.out.println(e);
        finally {
                try if(s != null) s.close(); catch (IOException e2);
        }
    }
}
```