Programación en Internet
(Sockets)

Programación con Sockets

Objetivo: aprender como construir aplicaciones cliente/servidor que se comunican utilizando sockets

Socket API

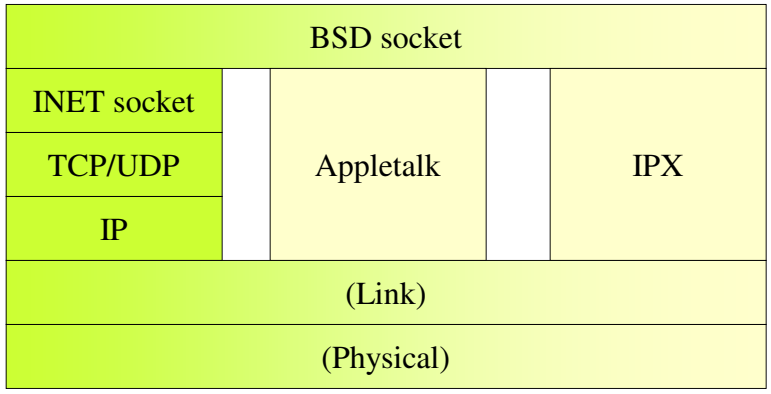
- introducida en el BSD4.1 UNIX, 1981
- explicitamente creada, utilizada, por aplicaciones paradigma cliente/servidor
- dos tipos de servicio de transporte via el API de socket:
  - datagrama no fiable
  - flujo de bytes fiable

socket

Una interfaz de aplicación creada, tenida y controlada por el OS (una "puerta") dentro de la cual los procesos pueden enviar y recibir mensajes a/desde otro proceso (remoto o local).

Sockets en UNIX

- Una forma de comunicarse con otro proceso usando descriptores de ficheros estándares (recordar: todo en UNIX es un fichero)
-



3

Struct sockaddr

- Almacena información de la dirección del socket

```
struct sockaddr {
    unsigned short sa_family;       // address family, AF_xxx
    char sa_data[14];               // 14 bytes of protocol address
};
```

- Familia de direcciones soportadas (*include/linux/socket.h*)
  - UNIX — Unix domain sockets
  - INET — TCP/IP
  - AX25 — Amateur radio
  - IPX — Novell IPX
  - APPLETALK — Appletalk
  - X25 — X.25
- Mas; unas 24 en total

- Para INET, sa_family = AF_INET

Network Byte Order vs. Host Byte Order

- Existen dos formas de ordenar los bytes:
  - El más significativo primero: Network Byte Order o "Big-Endian Byte Order"
  - El menos significativo primero: Host Byte Order o “Little-Endian Byte Order"
- Se puede convertir de un orden a otro usando las funciones:
  - htons() : Host to Network short
  - htonl() : Host to Network long
  - ntohs() : Network to Host short
  - ntohl() : Network to Host long
- Siempre debe convertirse los datos a Network Byte Order antes de enviarlos por la red

5

Struct sockaddr_in

- Sockaddr para TCP / IP

```
struct sockaddr_in {
    short int sin_family;           // Address family
    unsigned short int sin_port;    // Port number
    struct in_addr sin_addr;                // Internet address
    unsigned char sin_zero[8];      // Same size as struct sockaddr
};
```

- A un puntero a **struct sockaddr_in** puede hacérsele casting a **struct sockaddr** y viceversa
- *sin_family* se corresponde con *sa_family* en **struct sockaddr**
- *sin_port* y *sin_addr* deben estar en Network Byte Order

6

struct in_addr sin_addr

- Dirección Internet (dirección IP)

```
struct in_addr {
    unsigned long s_addr;  // that's a 32-bit long, or 4 bytes
};
```

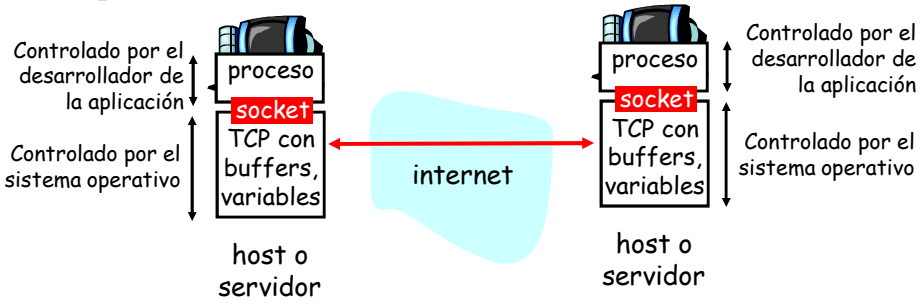
- De esta forma si se declara *ina* del tipo type struct sockaddr_in, entonces *ina.sin_addr.s_addr* referencia la dirección IP de 4 bytes (en Network Byte Order).

7

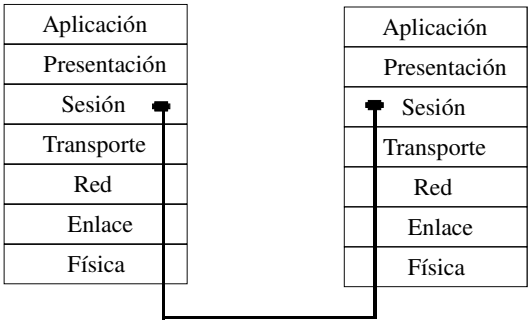
Programación de Sockets utilizando TCP en Java

Socket: una puerta entre procesos de aplicación y el protocolo de transporte extremo a extremo (UCP o TCP)

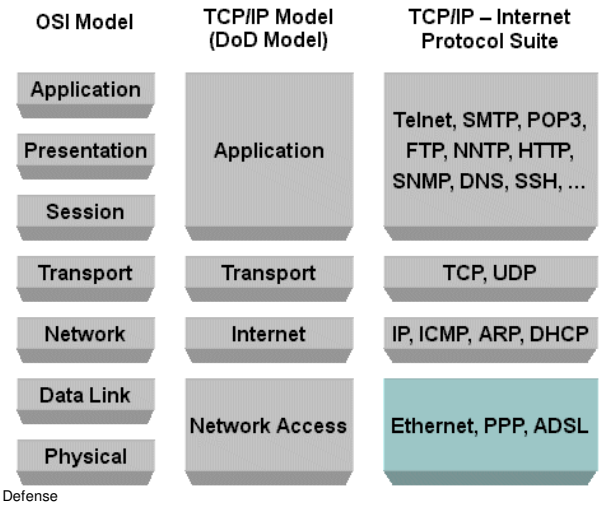
Servicio TCP: trasnferencia fiable de **bytes** desde un proceso a otro



Sockets



OSI Model, DoD Model and TCP/IP Protocol Suit



DoD: Department of Defense

Programación de Sockets utilizando TCP

El cliente debe contactar al servidor

- El proceso servidor debe correr primero
- El servidor debe haber creado algún socket (puerta) que reciba conexiones de clientes

El cliente contacta al servidor mediante:

- Crear un socket TCP local al cliente
- Especificar la dir. IP, el puerto del proceso servidor

- Cuando el cliente crea un socket: el cliente TCP establece una conexión al servidor TCP
- Cuando el servidor es contactado por un cliente, el servidor TCP crea un nuevo socket para que el proceso servidor se comunique con el cliente permite que el servidor hable con múltiples clientes

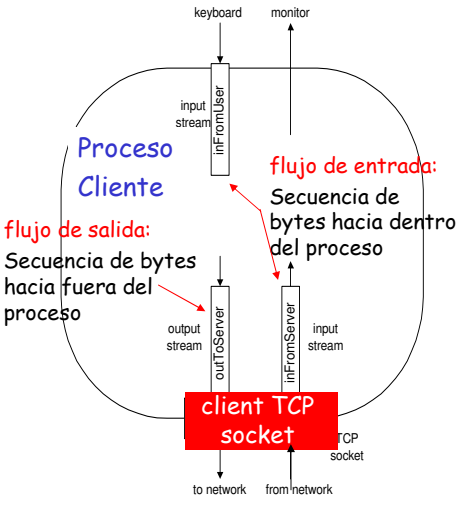
Punto de vista de la aplicación

El TCP provee de una transferencia de bytes ("pipe") fiable y en orden entre el cliente y el servidor

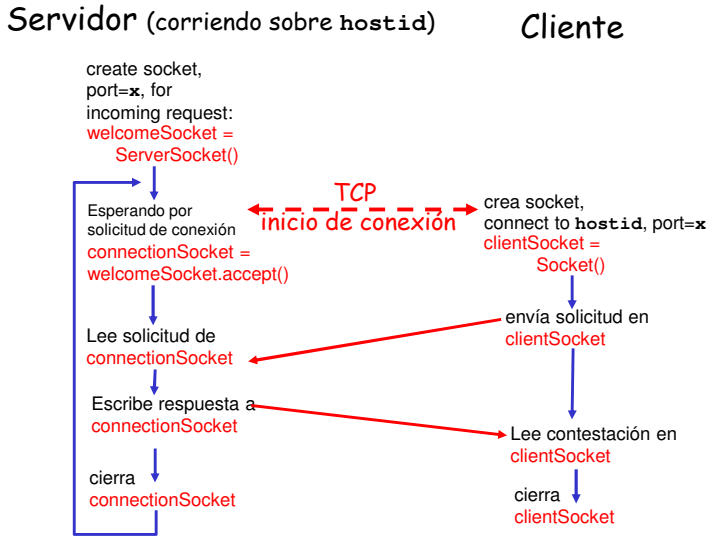
Programación de Sockets utilizando TCP

Ejemplo aplicación cliente-servidor:

- El cliente lee una línea de la entrada estándar (**inFromUser** stream) , y la envía al servidor vía el socket (**outToServer** stream)
- El servidor lee la línea del desde el socket
- El servidor convierte la línea en mayusculas y la envía al cliente
- El cliente lee e imprime la línea modificada desde el socket (**inFromServer** stream)



Interacción cliente/servidor con Socket TCP



Programación de Sockets *con UDP*

UDP: sin "conexión" entre el cliente y el servidor

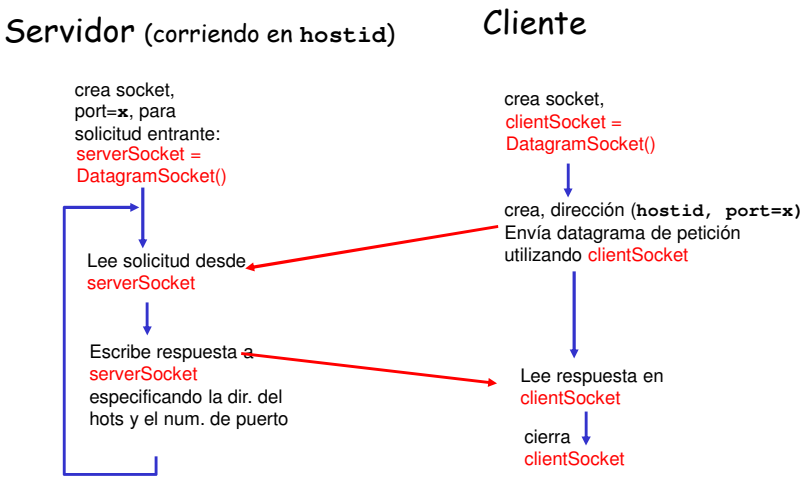
- sin handshaking (saludo)
- El emisor asocia explícitamente la dir. IP y el puerto del destinatario
- El servidor debe extraer del datagrama recibido, la dir. IP y el puerto del emisor

UDP: los datos transmitidos pueden ser recibidos fuera de orden, o se pueden perder

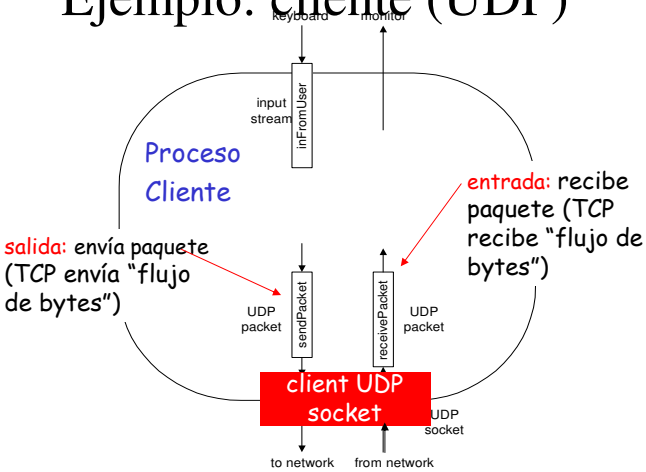
Punto de vista de la aplicación

UDP provee una transferencia no fiable de grupos de bytes (datagramas) entre el cliente y el servidor

Interacción cliente/servidor de sockets UDP



Ejemplo: cliente (UDP)



Sockets

- Protocolos de Transporte:
  - XNS
  - TCP/IP
  - UNIX
- Tipos de Servicios
  - Datagramas: **SOCK_DGRAM**
  - Circuitos Virtuales:
    - **SOCK_STREAM**
    - **SOCK_SECPACKED**
  - Especiales: **SOCK_RAW**

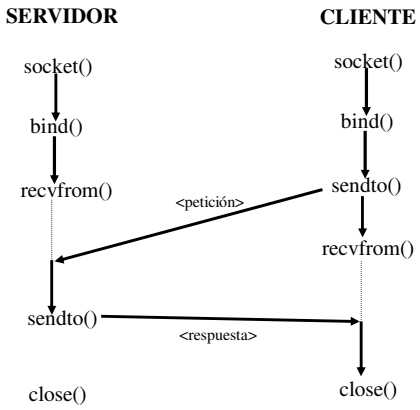
Sockets

- Un socket está caracterizado por 5 parámetros :
  - Una familia de protocolos
  - La dirección IP de la máquina local
  - El puerto de la capa de transporte
  - La dirección IP de la máquina remota
  - El puerto de la capa de transporte remota
- Los cinco parámetros arriba mencionados son necesarios para establecer una comunicación.

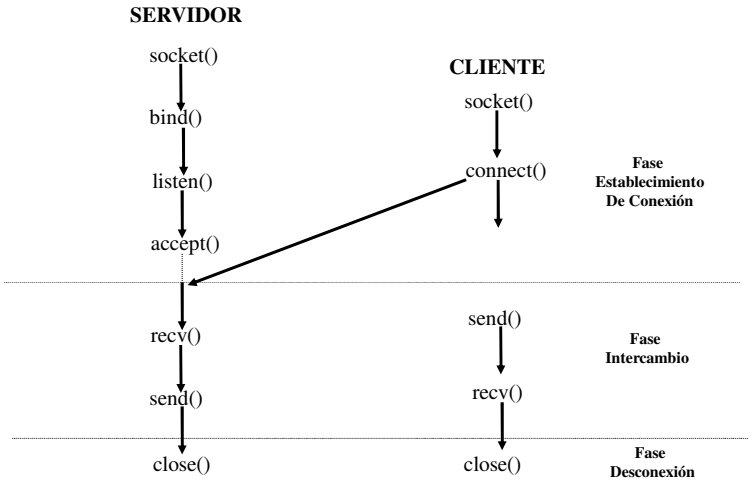
Sockets: Llamados al Sistema

socket	Crea un socket del tipo especificado (Stream, Datagrama, o Especial).
bind	Asocia una dirección IP, un puerto de la capa de transporte y una familia de protocolos.
listen	Crea una cola de peticiones de conexión.
accept	Acepta una petición de conexión solicitada por un cliente. En caso de no existir se duerme en espera de una. Al salir crea un socket completo.
connect	Invita a establecer conexión con un socket remoto perteneciente a un servidor .
send/write	Envía datos a través del socket (circuitos virtuales)
recv/read	Recibe datos a través del socket (circuitos virtuales)
sendto	Envía datos a través del socket (datagramas)
recvfrom	Recibe datos a través del socket (datagramas)
select	Revisa el estado de varios sockets abiertos para lectura ó escritura
close	Termina la conexión en un socket.

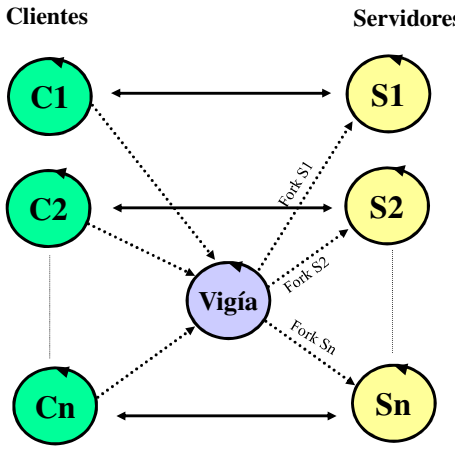
Sockets: Escenario Típico de Comunicación (Serv. Sin conexión)



Sockets: Escenario Típico de Comunicación (Serv. Orientado a Conexión)

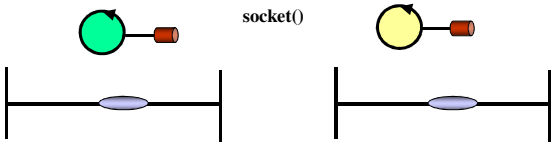


Sockets: Servidor Concurrente

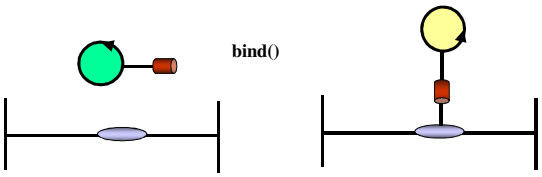


Sockets: Servidor Concurrente

(a) Cliente y Servidor solicitan un socket al sistema

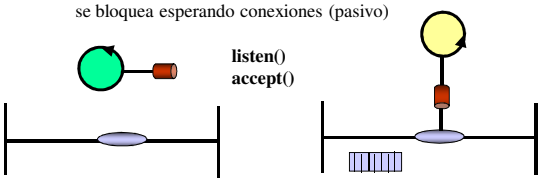


(b) El Servidor se asocia a un puerto conocido

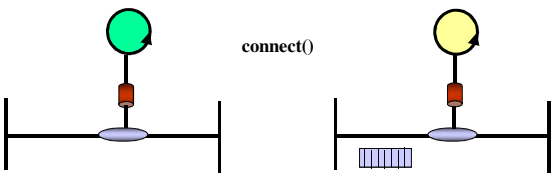


Sockets: Servidor Concurrente

(c) El Servidor reserva cola de espera e indica que está listo y se bloquea esperando conexiones (pasivo)

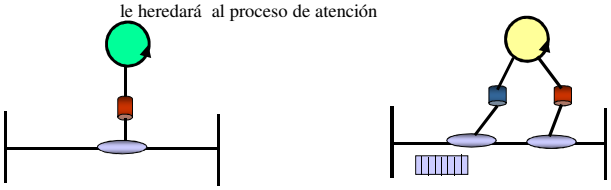


(d) El Cliente solicita conexión a la dirección bien conocida del Servidor. El sistema le asigna un puerto libre.

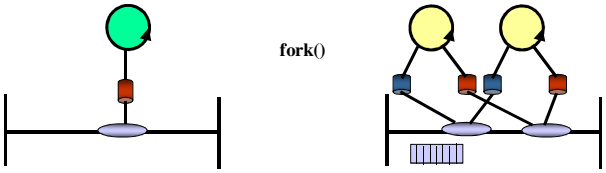


Sockets: Servidor Concurrente

(e) Al aceptar la conexión, el Servidor crea un nuevo socket que le heredará al proceso de atención

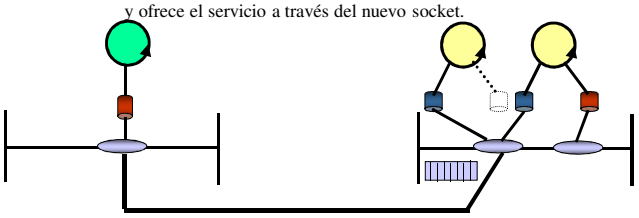


(f) El Servidor crea un nuevo proceso que atenderá la petición

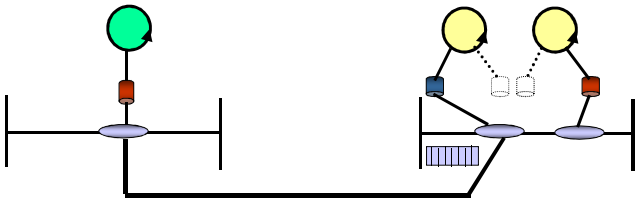


Sockets: Servidor Concurrente

(g) El proceso hijo cierra el socket de petición de servicio y ofrece el servicio a través del nuevo socket.



(h) Por su parte, el padre cierra el nuevo socket y regresa a esperar una nueva conexión.



Anexo: Ejemplos de programas con Sockets

En este anexo se presentan ejemplos de programas cliente-servidor. Los servicios son:

- Servicio de eco con transformación de minúsculas a mayúsculas
  - Con sockets stream
  - Con sockets datagrama

El servidor con sockets stream es concurrente; la aplicación está formada por los archivos: srvTcp.c y clTcp.c

La aplicación con sockets datagrama está formada por los archivos: srvUdp.c y clUdp.c

Cliente (Sockets Stream): clTCP.c

```
// Cliente (Socket Stream)
// Envia una cadena al servidor y se la devuelve en mayusculas
// Victor J. Sosa Sosa
// Lincar con -lxnet en la compilacion : gcc cliente.c -o cl -lxnet

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <sys/errno.h>
#include <netdb.h>
#define MAX_LINE 120
extern int errno;

main(int argc,char *argv[])
{
  char buf[MAX_LINE];
  struct sockaddr_in fsock, sname;
  struct hostent *hent; /* estructura que guarda el llamado a gethostbyname */
  int s, len;
  if (argc!=2){
      printf("USO: cliente nombreMaquina_del_Servidor\n");
      exit(1);
   }
  if(!(hent = gethostbyname(argv[1])) ){
       perror("GETHOSTBYNAME: ");
       exit(0);
   }
  if((s=socket(AF_INET,SOCK_STREAM,0)) < 0) {
       perror("SOCKET: ");
       exit(0);
   }
```

Cliente (Sockets Stream): clTCP.c

```
 fsock.sin_family = AF_INET;
  fsock.sin_addr.s_addr = *(long *) hent->h_addr; /* direccion IP de Maq. Remota */
  fsock.sin_port = htons(4400); /* puerto de la maq. remota en formato BIGENDIAN */
  if(connect(s,(struct sockaddr *)&fsock, sizeof(struct sockaddr_in)) == -1){
       perror("CONNECT: ");
       close(s);
       exit(0);
   }
  printf("Arranca el Programa Cliente !!!... Pulse q para salir\n");
  while(1){
       printf("Dame una cadena.: ");
       fgets(buf,MAX_LINE,stdin);
       if( send(s,buf,strlen(buf),0) < strlen(buf) ){
            perror("SEND: ");
            break;
         }
       if( buf[0]=='q' || buf[0] == 'Q'){
            printf("Terminamos....\n");
            close(s);
            exit(0);
         }
       printf("Me detengo a recibir la respuesta del servidor...\n");
       if( (len=recv(s,buf,MAX_LINE-1,0))<= 0 ){
            perror("RECV: ");
            close(s);
            exit(0);
         }
       buf[len] = '\0';
       printf("Respuesta..: %s\n\n",buf);
   }
}
```

2/2

Servidor Concurrente (Sockets Stream): srvTcp.c

```
// Recibe una cadena del Cliente  y se la devuelve en mayusculas
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define MAX_LINE 120
extern int errno;

main()
{
  struct sockaddr_in lsock,fsock, sname;
  int s, ss;
  int len,i;
  char buf[MAX_LINE];

  if((s=socket(AF_INET,SOCK_STREAM,0)) < 0) {
       perror("SOCKET: ");
       exit(0);
   }
  lsock.sin_family = AF_INET;
  lsock.sin_port = htons(4400); /* puerto para dar el servicio */
  lsock.sin_addr.S_un.S_addr = 0; /* direccion IP de mi maquina servidora */
  if(bind(s,(struct sockaddr *)&lsock, sizeof(struct sockaddr_in)) < 0 ){
       perror("BIND: ");
       exit(1);
   }
  if(listen(s,3)<0){
       perror("LISTEN: ");
       exit(1);
   }
```

1/2

srvTcp.c

```
while(1){
       len = sizeof(struct sockaddr_in);   /* &len: entra y sale el tamano del socket esperado */
       if((ss=accept(s,(struct sockaddr *)&fsock, &len)) < 0){
            perror("ACCEPT: ");
            continue;
         }
        printf("Conexion en el socket %d (antes %d)\n",ss, s);
        if (fork() == 0) {
             /* Aqui se ejecuta el proceso hijo */
             /* Cierra el socket incompleto */
             /* se dedica a atender la conexion con el socket completo */
           close(s);
           while(1){
            if((len=recv(ss,buf,MAX_LINE-1,0))<=0){
                 perror("RECV: "); /* Si len==0 entonces el cliente cerro la conexion */
                 exit(1);
              }
            for(i=0; i<len; i++) { /* Despliega y transforma a Mayusculas */
                 putchar(buf[i]);
                 if(buf[i] >= 'a' && buf[i] <= 'z')
                         buf[i] = 'A' + (buf[i] - 'a');
              }
            putchar('\n');
            if(buf[0] == 'Q' || buf[0] == 'q'){
                  printf("Termina el servicio por decision del Cliente\n");
                  close(ss);
                  exit(0); /* el proceso hijo se mata */
              }
            if(send(ss,buf,len,0) < len) /* responde al cliente */
                  perror("SEND: ");
          } /*while */
         } /* if fork */
         else /* Aqui continua el proceso vigia para aceptar otra conexion */
            close(ss);  /* el padre cierra el socket completo que dejo al hijo */
    } /*while*/
}
```

2/2

Cliente (Sockets Datagrama): clUDP.c

```
/#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <sys/errno.h>
#include <netdb.h>
#include <string.h>
#define MAX_LINE 120
extern int errno;

main(int argc,char *argv[])
{
  char buf[MAX_LINE];
  struct sockaddr_in fsock;
  struct hostent *hent; /* estructura que guarda el llamado a gethostbyname */
  int s, len, lenS;
  if (argc!=2){
      printf("USO: cliente nombreMaquina_del_Servidor\n");
      exit(1);
   }
  if(!(hent = gethostbyname(argv[1])) ){
       perror("GETHOSTBYNAME: ");
       exit(0);
   }
  lenS=sizeof(struct sockaddr_in);
  if((s=socket(AF_INET,SOCK_DGRAM,0)) < 0) {
       perror("SOCKET: ");
       exit(0);
   }
  fsock.sin_family = AF_INET;
  fsock.sin_addr.s_addr = *(long *) hent->h_addr; /* direccion IP de Maq. Remota */
  fsock.sin_port = htons(4400); /* puerto de la maq. remota en formato BIGENDIAN */
```

1/2

Cliente(Sockets Datagrama): clTCP.c

```
printf("Arranca el Programa Cliente !!!... Pulse q para salir\n");
  while(1){
       printf("Dame una cadena.: ");
       fgets(buf,MAX_LINE,stdin);
       if( sendto(s,buf,strlen(buf),0, (struct sockaddr *)&fsock, lenS) < strlen(buf) ){
            perror("SENDTO: ");
            break;
         }
       if( buf[0]=='q' || buf[0] == 'Q'){
            printf("Terminamos....\n");
            close(s);
            exit(0);
         }
       printf("Me detengo a recibir la respuesta del servidor...\n");
       if( (len=recvfrom(s,buf,MAX_LINE,0, (struct sockaddr)&fsock, &lenS))<= 0 ){
            perror("RECVFROM: ");
            close(s);
            exit(0);
         }
       buf[len] = '\0';
       printf("Respuesta..: %s\n\n",buf);
   }
```

2/2

Servidor(Sockets Datagrama): srvUDP.c

```
// Recibe una cadena del Cliente  y se la devuelve en mayusculas
// Victor J. Sosa Sosa
// Lincar con -lxnet en la compilacion : gcc servidor.c -o ser -lxnet

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

#define MAX_LINE 120
extern int errno;

main()
{
  struct sockaddr_in lsock,fsock;
  int s;
  int len,i, lenS;
  char buf[MAX_LINE];

  if((s=socket(AF_INET,SOCK_DGRAM,0)) < 0) {
       perror("SOCKET: ");
       exit(0);
   }
  lsock.sin_family = AF_INET;
  lsock.sin_port = htons(4400); /* puerto para dar el servicio */

  lenS=sizeof(struct sockaddr_in);
  if(bind(s,(struct sockaddr *)&lsock, lenS) < 0 ){
       perror("BIND: ");
       exit(1);
   }
```

1/2

Servidor (Sockets Datagrama): srvUDP.c

```
while(1){
      printf("Servidor espera mensaje de algun Cliente\n");

      if((len=recvfrom(s,buf,MAX_LINE,0,&fsock,&lenS))<=0){
         perror("RECVFROM: ");
         exit(1);
      }
      for(i=0; i<len; i++) { /* Despliega y transforma a Mayusculas */
           putchar(buf[i]);
           if(buf[i] >= 'a' && buf[i] <= 'z')
                buf[i] = 'A' + (buf[i] - 'a');
      }
      putchar('\n');
      if(buf[0] == 'Q' || buf[0] == 'q'){
          printf("Un Cliente deja de Comunicarse con su Servidor\n");
      }
      if(sendto(s,buf,len,0,&fsock,lenS) < len) /* responde al cliente */
                perror("SEND: ");
    } /*while */
}
```

2/2

Ejemplo: cliente Java (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

   public static void main(String argv[]) throws Exception
   {
      String sentence;
      String modifiedSentence;

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      Socket clientSocket = new Socket("hostname", 6789);

      DataOutputStream outToServer =
        new DataOutputStream(clientSocket.getOutputStream());
```

Crea flujo de entrada

Crea socket cliente, se conecta al server

Crea flujo de salida asociado al socket

Ejemplo: cliente Java (TCP), cont.

Crea flujo de entrada asociado al socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();

outToServer.writeBytes(sentence + '\n');

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

  }
}
```

Envía una línea al servidor

Lee la línea desde el servidor

Ejemplo: servidor Java (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Crea socket anfitrion en el puerto 6789

Espera, en el socket anfitrion el contacto de parte de un cliente

Crea flujo de entrada asociado al socket

Ejemplo: servidor java (TCP), cont..

Crea flujo de salida asociado al socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

lee línea de entrada desde el socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Escribe línea de salida al socket

```
outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

Fin del ciclo while, regresa al inicio del loop y espera por otra conexión del cliente

Ejemplo: cliente Java(UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
   public static void main(String args[]) throws Exception
   {
```

Create flujo de entrada

```
BufferedReader inFromUser =
  new BufferedReader(new InputStreamReader(System.in));
```

Create socket cliente

```
DatagramSocket clientSocket = new DatagramSocket();
```

Traduce nombre del host a dir. IP utilizando DNS

```
InetAddress IPAddress = InetAddress.getByName("hostname");

byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];

String sentence = inFromUser.readLine();
sendData = sentence.getBytes();
```

Ejemplo: cliente Java (UDP), cont..

Crea datadagrama con datos-a-enviar, longitud, dir. IP, puerto

Envía el datagrama al servidor

Lee el datagrama desde el servidor

```
                DatagramPacket sendPacket =
                  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

                clientSocket.send(sendPacket);

                DatagramPacket receivePacket =
                   new DatagramPacket(receiveData, receiveData.length);

                clientSocket.receive(receivePacket);

                String modifiedSentence =
                    new String(receivePacket.getData());

                System.out.println("FROM SERVER:" + modifiedSentence);
                clientSocket.close();
                }
            }
```

Ejemplo: servidor Java (UDP)

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
    {
```

Crea el socket datagrama en el puerto 9876

```
    DatagramSocket serverSocket = new DatagramSocket(9876);

    byte[] receiveData = new byte[1024];
    byte[] sendData  = new byte[1024];

    while(true)
      {
```

Crea un espacio para el datagrama recibido

```
      DatagramPacket receivePacket =
        new DatagramPacket(receiveData, receiveData.length);
```

Recibe datagrama

```
      serverSocket.receive(receivePacket);
```

Ejemplo: servidor Java (UDP), cont.

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

                String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress,
                     port);

serverSocket.send(sendPacket);
    }
  }
}
```

Obtiene la dir. IP, el num. Puerto del emisor

Crea datagrama para eníar al cliente

Escribe el datagrama al socket de salida

Fin del ciclo while, regresa al inicio del loop y espera por otro datagrama

Ejemplo: Sockets en Java

Un Cliente

```
import java.io.*;
impoert java.net.*;

public class SocketTest {

 public static void main(String argv[]) {
  try {
   Socket t = new Socket("java.sun.com", 13);
   DataInputStream is =
    new DataInputStream(t.getInputStream());
   boolean more = true;
   while (mnore) {
    String str = is.readLine();
    if (str == null)
     more = false;
    else
     System.out.println(str);
    }
   }
  } catch(IOException e) {
   System.out.println("Error" + e);
  }
 }}
```

Un Server

```
import java.io.*;
import java.net.*;

public class EchoServer {
 public static void main(String argv[]) {
  try {
   ServerSocket s = new ServerSocket(8189);
   Socket incoming = s.accept();
   DataInputStream in =
    new DataInputStream(incoming.getInputStream());
   PrintStream out =
    new PrintStream(incoming.getOutputStream());
   out.println("Hello. Enter BYE to exit");

   boolean done = false;
   while ( ! done) {
    String str = in.readLine();
    if (str == null)
     done = true;
    else {
     out.println("Echo: " + str);
     if (str.trim().equals("BYE"))
      done = true;
    }
   incoming.close();
  } catch(Exception e) {
   System.out.println(e);
  }
 }
}
```

Ejemplo: Sockets in Perl

Un **Cliente** que extrae documentos del Web.
USO: cliente host documento [documento...]

```
!/usr/bin/perl -w
use IO::Socket;
unless (@ARGV > 1) { die "usage: $0 host document ..." }
$host = shift(@ARGV);
$EOL = "\015\012";
$BLANK = $EOL x 2;
foreach $document ( @ARGV ) {
   $remote = IO::Socket::INET->new( Proto => "tcp",
                                    PeerAddr  => $host,
                                    PeerPort  => "http(80)",
                                   );
 unless ($remote) {die "cannot connect to http daemon on $host" }
   $remote->autoflush(1);
   print $remote "GET $document HTTP/1.0" . $BLANK;
   while ( <$remote> ) { print }
   close $remote;
}
```

Un **Servidor** que ejecuta algunos comandos
Y regresa el resultado

```
#!/usr/bin/perl -w
use IO::Socket;
use Net::hostent;              # para version OO de  gethostbyaddr
$PORT = 9000;                  # algun puerto sin usar
$server = IO::Socket::INET->new( Proto     => 'tcp',
                                 LocalPort => $PORT,
                                 Listen    => SOMAXCONN,
                                 Reuse     => 1);

die “no puedo iniciar servidor" unless $server;
print "[Server $0 aceptando clientes]\n";

while ($client = $server->accept()) {
   $client->autoflush(1);
   print $client “Bienvenido a $0; Escribe help para ver comandos.\n";
   $hostinfo = gethostbyaddr($client->peeraddr);
   printf "[Conectadot desde %s]\n", $hostinfo->name || $client->peerhost;
   print $client "Comando? ";
   while (<$client>) {
      next unless /\S/;       # linea en blanco
      if (/quit|exit/i) {            last;       } elsif (/date|time/i) {
         printf $client "%s\n", scalar localtime;       } elsif (/who/i ) {
         print  $client `who 2>&1`;         } elsif (/cookie/i ) {
         print  $client `/usr/games/fortune 2>&1`;       } elsif (/motd/i ) {
        print  $client `cat /etc/motd 2>&1`;       } else {
        print $client "Comandos: quit date who cookie motd\n";
      }
    } continue {
      print $client "Comando? ";
    }
    close $client;}
```

Programación con sockets: referencias

C-language tutorial (audio/slides):

- “Unix Network Programming” (J. Kurose),

http://manic.cs.umass.edu/~amldemo/courseware/intro.

Java-tutorials:

- “All About Sockets” (Sun tutorial), http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html
- “Socket Programming in Java: a tutorial,” http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html