Programming
language

Curso: R1042

Docente: Ing. Nahuel Gonzalez
Ayudantes:
Sr. Pose, Fernando
Sr. Demski, Andrés
Ing. Spataro, Hector

Introducción a Sockets
Lenguaje C

R1042

Universidad Tecnológica Nacional
Facultad Regional Buenos Aires

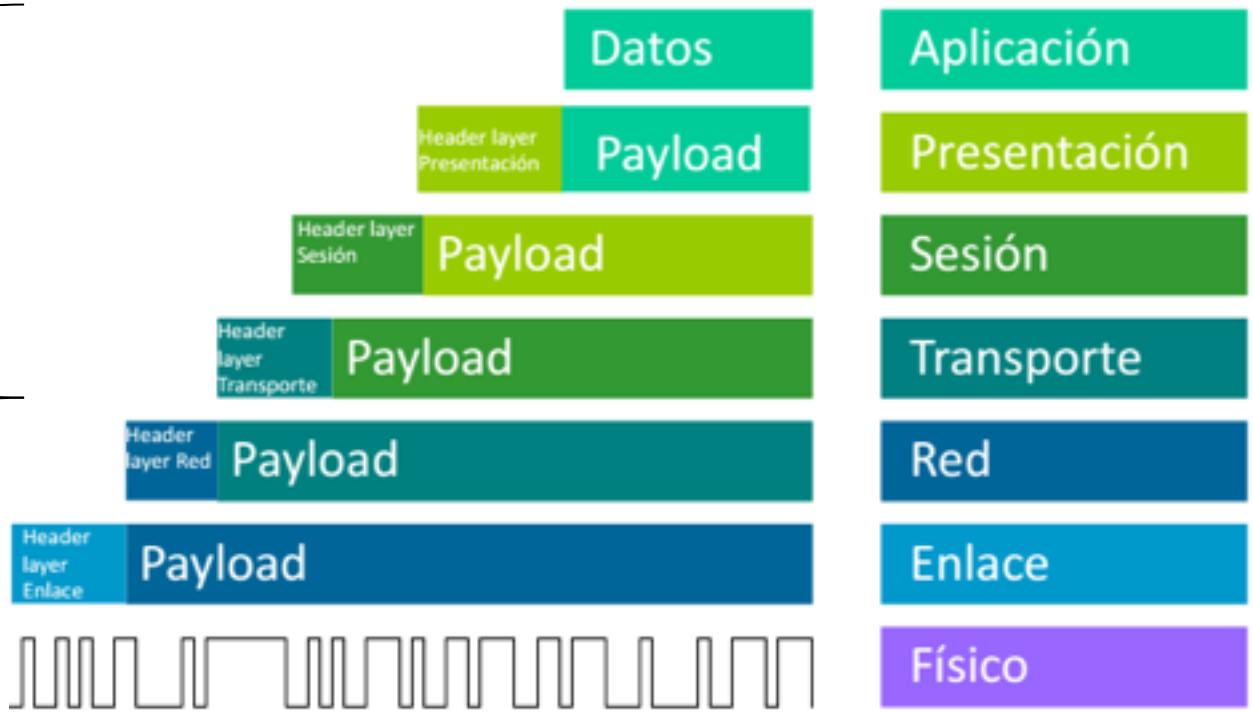
Año 2017
V.2.2

Programming
language

Modelo OSI

Informática I

Técnicas Digitales III



Modelo OSI

Capa Física: Conexión a nivel físico.

Capa de Enlace: Identificación de PC dentro de la red local mediante MAC

Capa de Red: Identificación de PC en internet mediante IP

Capa de Transporte: Permite la conectividad extremo a extremo. TCP y UDP.

Sesión: Comunicación entre dispositivos de la red.

Presentación: Encriptar y desencripta los datos de las capas anteriores.

Aplicación: Capa que interactúa con el usuario.

Dirección MAC e IP

Dirección MAC:

- ✓ Es un número de 6 bytes.
- ✓ Es única, cada PC tiene su dirección.
- ✓ Es comprada por el fabricante.

Ejemplo: 08:00:20:00:00:03

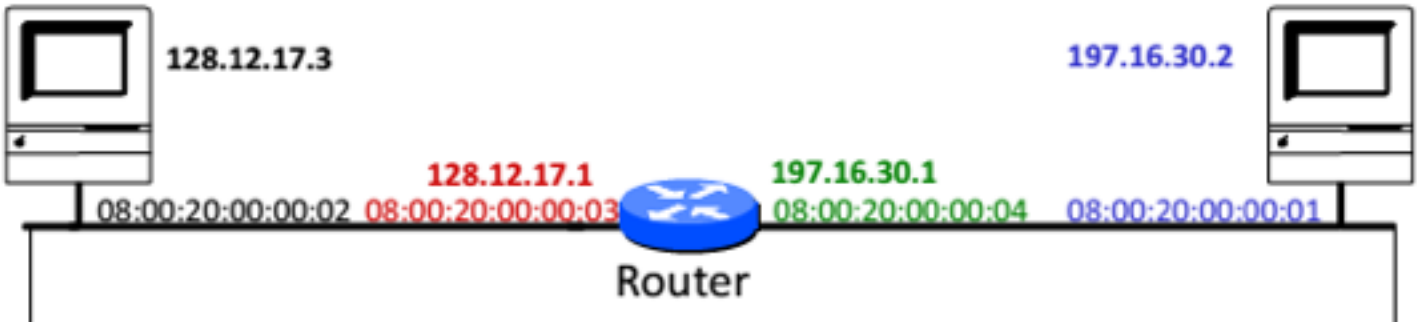
Dirección IP:

- ✓ Esun número compuesto por 4 decimales.
- ✓ Rango de 0 a 255, separada por puntos.
- ✓ Cada red que conforma internet tiene su rango de direcciones y esta no se repite.

Ejemplo: 128.12.17.3

Programming
language

Comunicación entre redes LAN



Programming
language

Estructura de la Presentación:

- ❑ Problemática a Resolver.
- ❑ Introducción a Sockets.
- ❑ Resolución de programa servidor.
- ❑Programando “mi” servidor.
- ❑ Resolución de programa cliente.
- ❑ Programando “mi” cliente.
- ❑ Compilación, linkeo y ejecución.
- ❑ Otros datos de interés.
- ❑ Servidor multiusuario.
- ❑ Ejemplo de función select()
- ❑ Bibliografía.

Problemática a Resolver - Aplicación

Se desea desarrollar la implementación de un **cliente/servidor** con las características que se detallan a continuación

✓El cliente escribirá datos que enviará al servidor.

✓ El servidor debe recibir los datos enviados por el cliente conectado y luego imprimirlos en pantalla.

Programming
language

Introducción a Socket (1)

Nosotros necesitamos que…

“Hola Mundo”

Sistema Operativo

Placa de Red

Programming
language

Introducción a Socket (1)

Nosotros necesitamos que…

“Hola Mundo”

“Hola Mundo”

Sistema Operativo

Placa de Red

Sistema Operativo

Placa de Red

¿Cómo lo hacemos?

Programming
language

Introducción a Socket (2)

¡SOCKET!

¿SOCKET?

Programming
language

Introducción a Socket (3)

…EN LINUX TODO…

…TODO ES UN ARCHIVO

Introducción a Socket (4)

La consola es un archivo

El teclado es un archivo

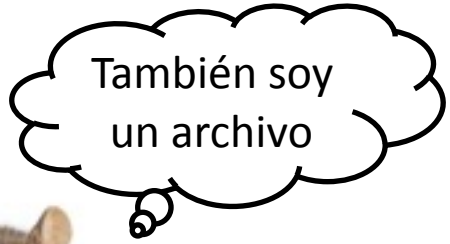
La pantalla es un archivo

Introducción a Socket (4)

La consola es un archivo

El teclado es un archivo

La pantalla es un archivo



Introducción a Socket (6)

Pero este es un archivo **especial**

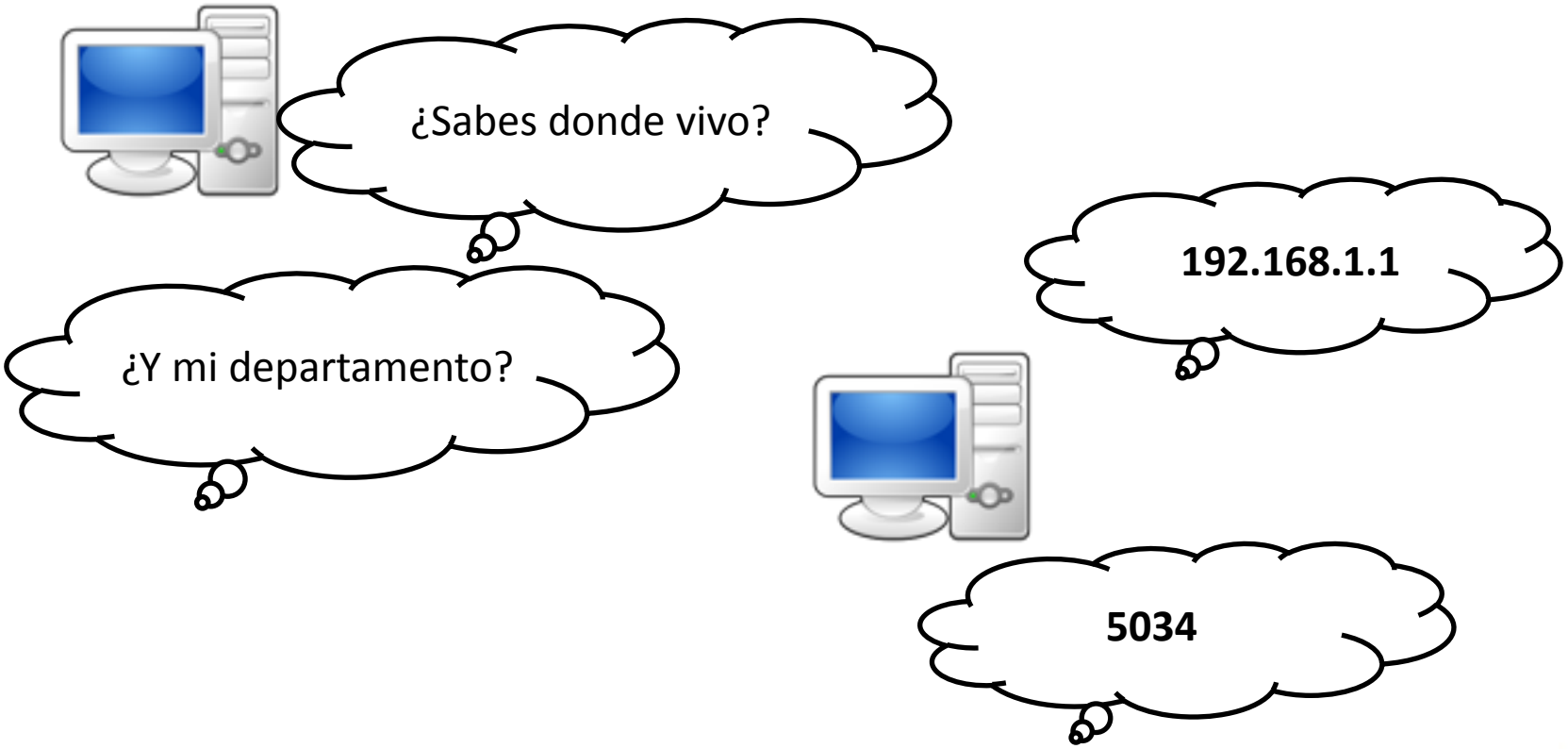
Programming
language

Introducción a Socket (7)

Además de ser un descriptor de archivo “Bidireccional”

Necesita conectarse a algún lado

Introducción a Socket (8)



¿PREGUNTAS?

Recordando la situación propuesta

Deseamos desarrollar la implementación de un **cliente/servidor** con las siguientes características:

✓El cliente escribirá datos que enviará al servidor.

✓ El servidor debe recibir los datos enviados por el cliente conectado y luego imprimirlos en pantalla.

Resolución – Programa Servidor (1)

Primero deberemos conseguir un Socket

Socket();

Programming
language

Resolución – Programa Servidor (2)

La función **socket()** se utiliza para conseguir el descriptor de fichero. Función:

```
int socket(int domain,int type,int protocol);
```

Siendo los argumentos de la función:

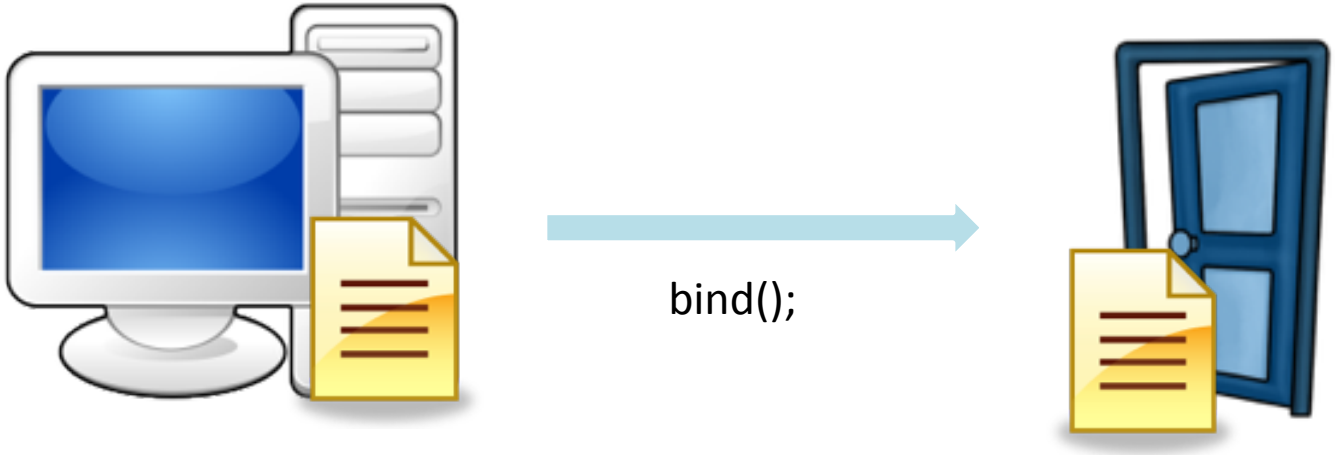
- domain – AF_INET. Es el protocolo de internet ARPA (sockets de redes de UNIX)
- type – SOCK_STREAM. Tipo de socket. Usan protocolo TCP y son los que utilizamos en informática I.
- protocol – 0. De esta forma el socket() elige el protocolo correcto en función del type.

La función retorna el descriptor de archivo (socket) o -1 en caso de error

Resolución – Programa Servidor (3)

Este socket no esta totalmente definido

¡Debemos asociarlo a un puerto y una IP!



Programming
language

Resolución – Programa Servidor (4)

La función **bind()** se utiliza para asociar al socket con [IP][puerto] de la máquina local

```
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

Siendo los argumentos de la función:

- Sockfd – Es el descriptor de fichero que devolvió socket()
- My_addr – Puntero a estructura que contiene la dirección propia del servidor
- Addrlen – sizeof(struct sockaddr) tamaño de la estructura de datos del servidor

La función retorna -1 en caso de error

Programming
language

Resolución – Programa Servidor (4)

La función **bind()** se utiliza para asociar al socket con [IP][puerto] de la máquina local

```
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

Siendo los argumentos de la función:

- Sockfd – Es el descriptor de fichero que devolvió socket()
- My_addr – Puntero a estructura que contiene la dirección propia del servidor
- Addrlen – sizeof(struct sockaddr) tamaño de la estructura de datos del servidor

La función retorna -1 en caso de error
(error producido por ejemplo por puerto ocupado)

Resolución – Programa Servidor (6)

¿Y si quiero usar el primer puerto desocupado?

Resolución – Programa Servidor (6)

¿Y si quiero usar el primer puerto desocupado?

¡Con una estructura do-while y un if lo podemos solucionar!

Programming
language

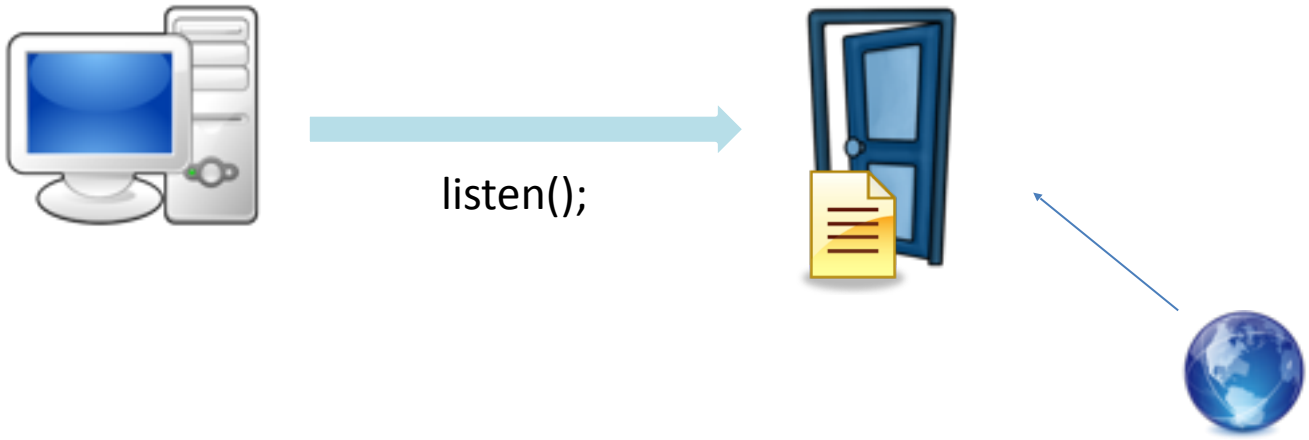
Resolución – Programa Servidor (7)

```
do{
   var_bin = bind(socket, (struct sockaddr*) &my_addr, sizeof(my_addr);
   if(var_bin == -1){
      condicion = 1;
      nuevo_port++;
      modificar_sockaddr(nuevo_port); //Acá debo modificar my_addr.
   }
   else{
      condicion = 0;
   }
}while(condicion);
```

Finalmente el puerto queda ocupado

Resolución – Programa Servidor (8)

Luego lo ponemos a escuchar conexiones entrantes



Programming
language

Resolución – Programa Servidor (9)

La función **listen()** se utiliza para poner el servidor a la espera de conexiones. Función:

```
int listen(int sockfd,int backlog);
```

Siendo los argumentos de la función:

- Sockfd – Es el descriptor de fichero que devolvió socket()
- Backlog – Número de conexiones permitidas en la cola de entrada.

La función retorna -1 en caso de error

Nota: Las conexiones de entrada (clientes) esperan en la cola de entrada hasta que son aceptadas, **backlog** es el límite de conexiones que puede permanecer a esta cola.

Programming
language

Resolución – Programa Servidor (10)

Es SUMAMENTE importante destacar:

- La función listen() NO es bloqueante.
- Listen() Solo avisa que el sistema debe ponerse a escuchar.
- Las conexiones que escucha las encola.
- El máximo de conexiones a encolar es determinado por Backlog

¡No confundir máximo de conexiones a encolar con máximo de conexiones que transaccionan!

Resolución – Programa Servidor (11)

Ejemplo:

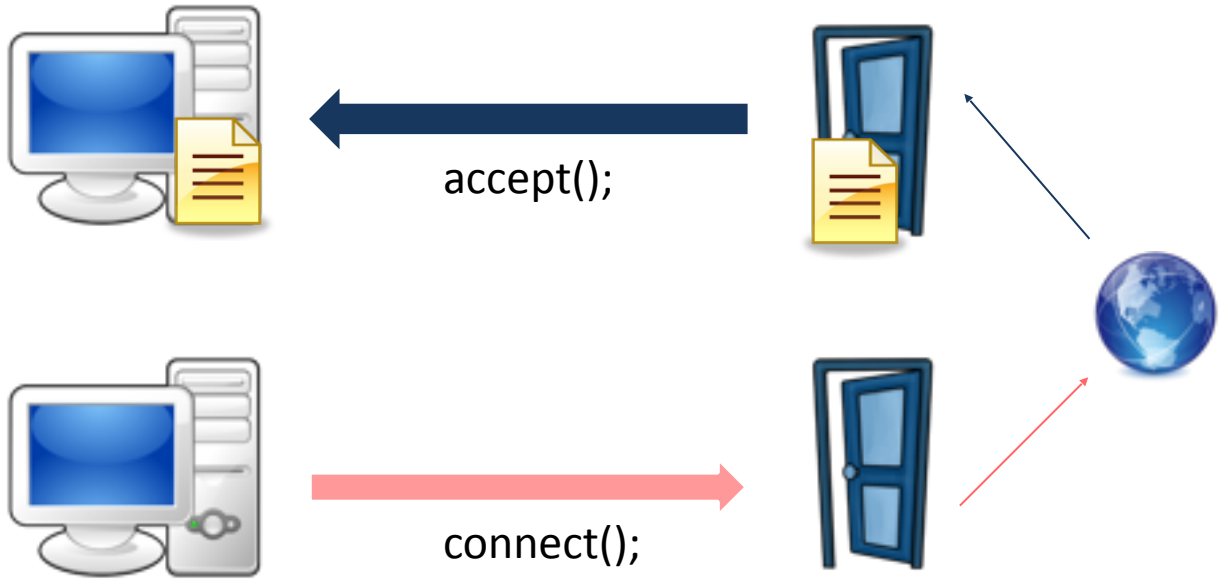
Suponemos a Backlog = 10

Si desde el servidor “escuchamos” una nueva conexión la misma no la encola

Luego de comenzar a realizar transacciones con el socket que escuchamos la cola se decrementa y podemos escuchar una nueva conexión

Resolución – Programa Servidor (12)

Hasta que finalmente… ¡Establecimos una comunicación!



Programming
language

Resolución – Programa Servidor (13)

La función **accept()** se utiliza para obtener la conexión pendiente. Función:

```
int accept(int sockfd, void *addr, int *addrlen);
```

Siendo los argumentos de la función:

- Sockfd – Es el descriptor de fichero que devolvió socket()
- addr – Puntero a estructura struct sockaddr_in local. Es donde se guarda la información de la conexión entrante
- addrlen – sizeof(struct sockaddr_in) contiene el tamaño de la estructura addr

La función retorna -1 en caso de error

Resolución – Programa Servidor (14)

La función **accept()** es bloqueante, pero…

¿Cuándo se bloquea?

Resolución – Programa Servidor (14)

La función **accept()** es bloqueante, pero…

¿Cuándo se bloquea?

¡Se bloquea cuando no hay conexiones en la cola para aceptar!

Resolución – Programa Servidor (15)

Es de suma importancia notar que la función **accept()** devolverá un descriptor de fichero de socket nuevo a utilizarse en la conexión para enviar y recibir datos.

El descriptor de fichero original, devuelto por **socket()**, continuará “escuchando” en el puerto.

Resolución – Programa Servidor (16)

¿Cómo enviamos datos?

Programming language

Resolución – Programa Servidor (17)

Las funciones **recv()** y **send()** se utilizan para comunicarse a través de los sockets de flujo. Funciones:

```
int send(int sockfd, const void *msg, int len, int flags);
int recv(int sockfd, void *buf, int len, unsigned int flags);
```

Siendo los argumentos de las funciones:

- Sockfd – Es el descriptor de fichero que devolvió accept()
- buf – Puntero a los datos que se desean enviar o donde se desea recibir
- Len – Longitud de los datos de buf en bytes
- Flags – Se le asigna 0

Las funciones retornan el número de bytes que leyeron/enviaron o -1 en caso de error

Resolución – Programa Servidor (18)

Finalmente, cerramos la puerta

close();

Programming
language

Resolución – Programa Servidor (19)

La función **close()** se utiliza para cerrar la conexión. Función:

```
int close(int sockfd);
```

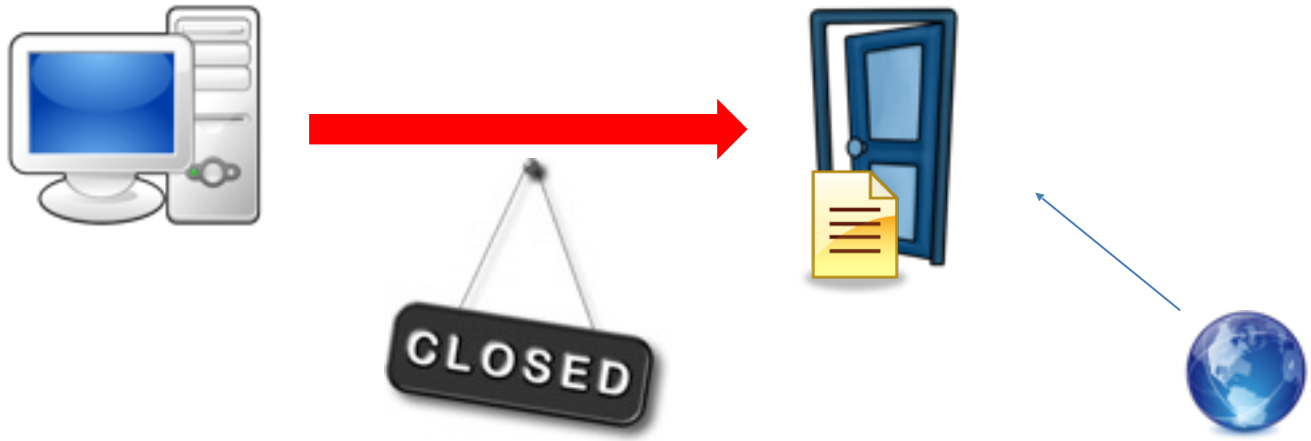
Siendo el argumento de la función :

- Sockfd – Descriptor de fichero que devolvió accept() y se desea cerrar

La función retorna -1 en caso de error y 0 en caso de éxito

Resolución – Programa Servidor (20)

¡Fin de la conexión!



¿PREGUNTAS?

Programming
language

¿Y si quisiera conectarme de dos clientes?

Solución propuesta:

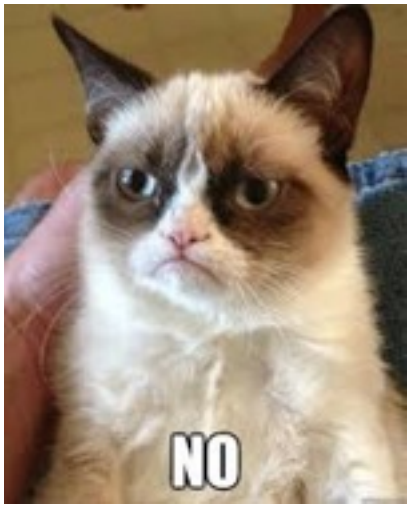
1. socket();
2. socket();
3. bind();
4. bind();
5. listen();
6. listen();
7. accept();
8. accept();

Programming language

¿Y si quisiera conectarme de dos clientes?

Solución propuesta:

1. socket();
2. socket();
3. bind();
4. bind();
5. listen();
6. listen();
7. accept();
8. accept();



El primer accept() me bloquea mi proceso y el segundo accept() puede no enterarse de la conexión correspondiente!

Programming
language

Resumiendo…

Los prototipos de las funciones que serán de utilidad en el programa fuente de la aplicación servidor serán:

```
int socket(int domain,int type,int protocol);
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
int listen(int sockfd,int backlog);
int accept(int sockfd, void *addr, int *addrlen);
int recv(int sockfd, void *buf, int len, unsigned int flags);
int send(int sockfd, const void *msg, int len, int flags);
int close(int sockfd);
```

Programming
language

Programando “mi” servidor (1)

Programando “mi” servidor (2)


```
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/types.h>
```

Cuando se desea trabajar con sockets, tanto en el programa fuente del cliente como en el servidor, **se deberán** incluir los archivos de cabecera que se detallan.

Programming language

Programando “mi” servidor (3)

La función main de un servidor contiene entonces:

Inicialización de variables

```
int listener;     //Socket que recibe las conexiones
int cliente;      //Socket que se conectó
int addrlen=sizeof(struct sockaddr_in); //Tamaño necesario para accept()
struct sockaddr_in datosServer, datosCliente; //Datos del cliente y servidor
char buffer[MAXBUFFER]; //Buffer para recibir mensajes
int clienteConectado; //Bandera de conexión de cliente
int sizeMensaje; //Tamaño del mensaje recibido
```

Obtención del descriptor de fichero

```
// Creo el socket
listener = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP) ;
```

Relleno de la estructura tipo “sockaddr” con los datos del servidor

```
datosServer.sin_family = AF_INET; //Familia de direcciones
datosServer.sin_addr.s_addr = INADDR_ANY; //Dirección de internet. Localhost
datosServer.sin_port = htons(PUERTO); //Número de puerto.
memset(datosServer.sin_zero,0,8);     //Se rellena a 0 para preservar tamaño
                                      //original de struct sockaddr
```

Programando “mi” servidor (4)

Enlazado del socket con la ip:puerto

```
bind(listener, (struct sockaddr*) &datosServer, sizeof datosServer);
```

“Encendido” del servidor

```
listen(listener,MAXCONEXIONES);
printf("Server conectado \n");
```

Programming language

Programamando “mi” servidor (5)

Y finalmente, una vez inicializado el servidor, continua un bucle infinito, por ejemplo **while(1)**.

```
while(1){

   printf("Esperando conexiones\n");

   //Acepto una conexion
   cliente = accept(listener, &datosCliente, &addrlen);
   clienteConectado = 1;

   //Muestro la IP del cliente, la obtengo de datosCliente
   printf("Se conecto alguien desde la ip: %s\n", inet_ntoa((struct in_addr)datosCliente.sin_addr));

   while(clienteConectado){
      printf("Esperando mensaje\n");

      //Recibo mensaje del cliente
      sizeMensaje = recv(cliente, buffer, MAXBUFFER,0);

      if(sizeMensaje>0){
         //imprimo mensaje
         buffer[sizeMensaje]='\0'; //Por las dudas pongo el \0, me protejo si no envía
         printf("Mensaje recibido: \"%s\"\n",buffer);
      }else clienteConectado = 0;
   }
   printf("Se desconecto el cliente\n");

   //Cierro el socket del cliente
   close(cliente);
}
return 0;
}
```

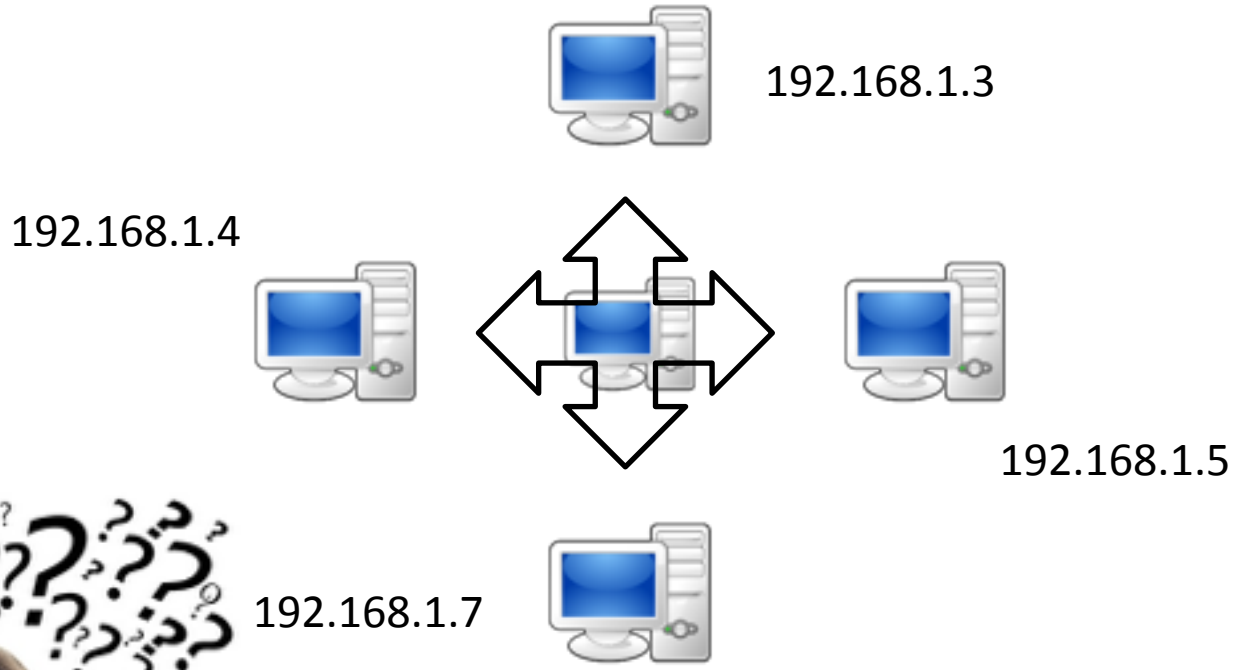
Resolución – Programa cliente (1)

Primero deberemos conseguir un Socket

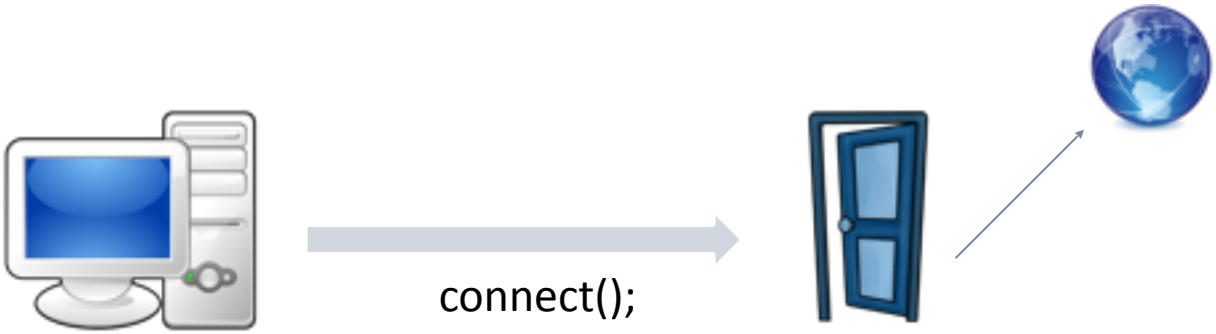
Socket();

Resolución – Programa cliente (2)

¿A dónde vamos?



Resolución – Programa cliente (3)



Programming language

Resolución – Programa Cliente (4)

A continuación se detalla la función **connect()** dado que las restantes funciones ya fueron detalladas y mencionadas en el “programa servidor”.

La función **connect()** se utilizan para conectar el cliente con una máquina remota. Función:

```
int connect(int sockfd, struct sockaddr *serv_addr , int addrlen);
```

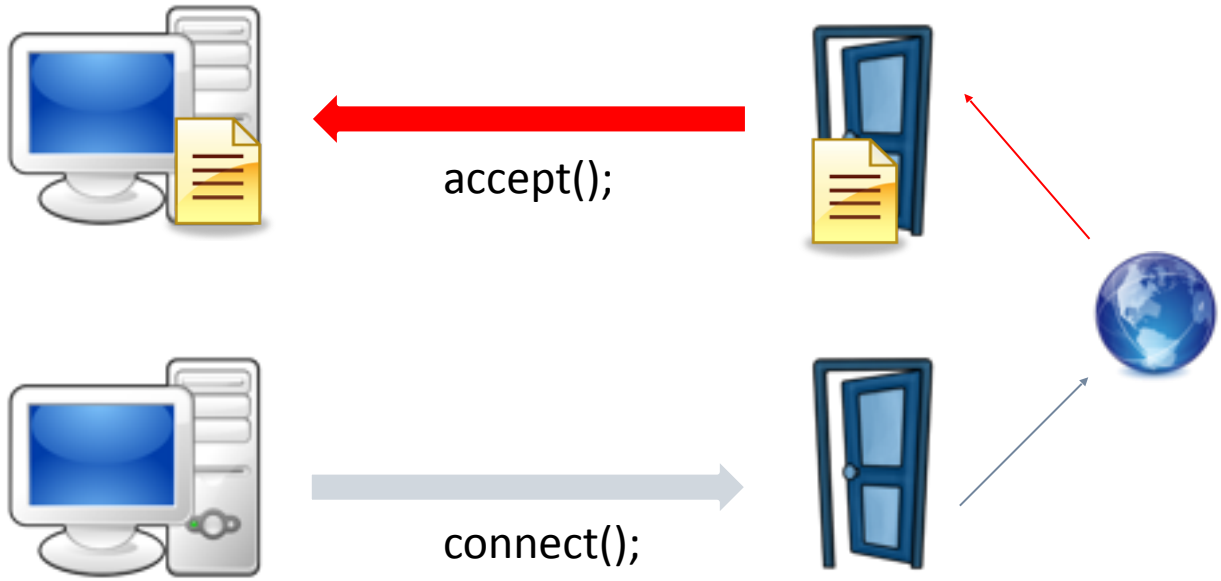
Siendo los argumentos de la función:

- Sockfd – Es el descriptor de fichero que devolvió socket()
- serv_addr – Puntero a estructura que contiene ip:puerto de destino (servidor)
- Addrlen – sizeof(struct sockaddr) contiene el tamaño de la estructura addr

La función retorna -1 en caso de error
Fd en caso de éxito

Resolución – Programa cliente (5)

¡Toquemos el timbre y esperemos que contesten..!



Programming
language

Resumiendo…

Los prototipos de las funciones que nos podrán ser de utilidad en el programa fuente de la aplicación cliente serán:

```
int socket(int domain,int type,int protocol) ;
int connect(int sockfd, struct sockaddr *serv_addr , int addrlen);
int send(int sockfd, const void *msg, int len,int flags);
int recv(int sockfd, void *buf, int len,unsigned int flags);
int close(int sockfd);
```

Programming language

Programando “mi” cliente (1)

La función main de un cliente entonces contiene:

Inicialización de variables

```
char buffer[MAXBUFFER]; //Buffer donde guardo el dato a enviar
int socketCliente;      //Socket con el cual me conecto
struct sockaddr_in datosServer; //Datos del servidor
int enviados; //Tamaño del mensaje enviado
```

Obtención del descriptor de fichero

```
// Creo el socket
socketCliente = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP) ;
```

Relleno de la estructura tipo “sockaddr” con los datos del servidor

```
// Lleno la estructura con la info del server
datosServer.sin_family = AF_INET;  //Falia de direcciones
datosServer.sin_addr.s_addr = INADDR_ANY; //Dirección de internet Localhost
datosServer.sin_port = htons(PUERTO); //Número de puerto
memset(datosServer.sin_zero,0,8); //Se rellena a 0 para preservar tamaño
                                  //original de struct sockaddr
```

Programming
language

Programando “mi” cliente (2)

Realizo la conexión con el “servidor”

```
//Me conecto al server
connect(socketCliente, (struct sockaddr *) &datosServer , sizeof(struct sockaddr));
printf("Conectado\nPara desconectarse escriba \"salir\"\n");
```

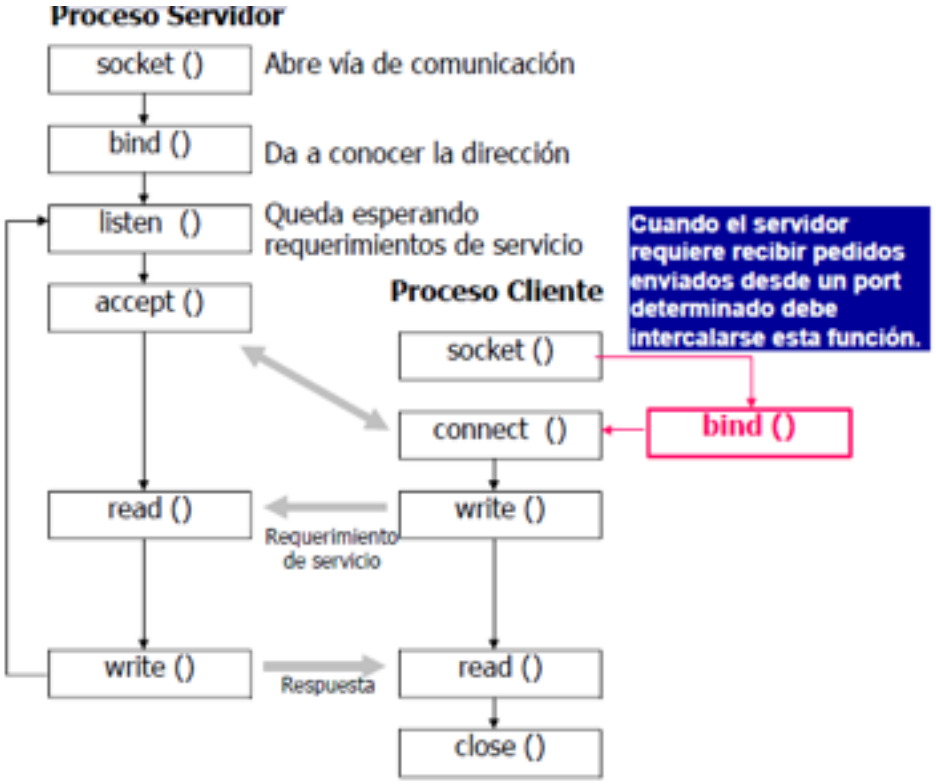
Programming language

Programando “mi” cliente (3)

Y finalmente, una vez inicializado el cliente y conectado, se termina con un bucle **while(1)** el cual termina cuando desconecto mi cliente del servidor (en este caso ingresando “salir”)

```
while(1){

   printf("Escriba mensaje a enviar> ");
   fgets(buffer,MAXBUFFER,stdin);
   buffer[strlen(buffer)-1]='\0';

   if(strcmp(buffer,"salir")==0) break;

   enviados = send(socketCliente, buffer, strlen(buffer),0);

   if(enviados>0){
      printf("Enviado\n");
   }else{
      printf("Error servidor\n");
      break;
   }
}
close(socketCliente);
printf("Desconectado\n");
return 0;
}
```

RESUMEN


Proceso Servidor
socket ()
Abre vía de comunicación
bind ()
Da a conocer la dirección
listen ()
Queda esperando requerimientos de servicio
accept ()
Proceso Cliente
socket ()
connect ()
bind ()
Cuando el servidor requiere recibir pedidos enviados desde un port determinado debe intercalarse esta función.
read ()
write ()
Requerimiento de servicio
write ()
Respuesta
read ()
close ()

Programming
language

Compilación – Linkeo y Ejecución

A la hora de ejecutar el cliente/servidor es importante ejecutar en primera instancia el servidor, luego el cliente, de otra forma el cliente no podrá conectarse y la función connect() retornará -1

Programming language

Otros datos de interés (1)

Struct sockaddr:

```
struct sockaddr{
   unsigned short sa_family;   //Familia de direcciones, AF_XXX
   char           sa_data[14]; // bytes de la dirección del protocolo
};
```

Donde “sa_family” será AF_INET y “sa_data” contendrá una dirección y número de puerto de destino para el socket.

Dado que “struct sockaddr” requiere de una compleja manipulación, programadores, definieron una estructura paralela para trabajar más cómodamente: “Sturct sockaddr_in” (la que nosotros utilizamos)

```
struct sockaddr_int{
   short int          sin_family;   //Familia de direcciones, AF_INET
   unsigned short int sin_port;     //Número de puerto
   struct in_addr     sin_addr;     //Direccipon de Internet
   unsigned char      sin_zero[8];  //Relleno para preservar el tamaño de
                                    //original struct sockaddr
}
```

Programming
language

Otros datos de interés (2)

¿Cómo levanto Puerto desde un archivo config.txt?

Es importante saber que cuando se desee escribir el puerto en la estructura mencionada, **struct sockaddr_in**, se debe utilizar la función **htons()** para ordenar los bytes de forma que primero se encuentre el más significativo, también llamado ordenación de bytes de la red.

```
uint16_t htons (hostshort uint16_t);
```

Ejemplo: Puerto: 1299

```
#define PUERTO 1299
datosServer.sin_port = htons(PUERTO);
```

Programming
language

Otros datos de interés (3)

¿Cómo levanto la dirección de IP desde un archivo config.txt?

Es importante saber que cuando se desee escribir una dirección de IP en la estructura mencionada, **struct sockaddr_in**, dada por cifras y puntos y no se utilice la IP local: INADDR_ANY o **localhost** se debe utilizar la función ***inet_addr("dirección")*** , para convertir la dirección de IP en notación de cifras y puntos en un unsigned long (mismo tipo que en estructura sockaddr_in)

```
in_addr_t inet_addr(const char *cp);
```

Ejemplo: IP: 10.12.110.57

```
struct sockaddr_in datos;
datos.sin_addr.s_addr = inet_addr("10.12.110.57");
```

¿PREGUNTAS?

Servidores Multiusuario (1)

Mejorando nuestro servidor

Utilizamos la función select()

Servidores Multiusuario (2)

Función select():

Int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);

- Nfds – Descriptor de mayor valor + 1
- Readfds – Set de lectura
- Writefds – Set de escritura
- Exceptfds – Set de excepción
- Timeout – Tiempo de bloqueo (ver man)

La función retorna:

- -1 en caso de error
- N descriptor listo
- 0 Time out

Servidores Multiusuario (3)

Tratamiento del set de descriptores:

1. Son vectores de enteros en los que cada bit representa un file descriptor.
2. Cuando se desbloquea por uno de ellos el bit correspondiente cambia a 1.
3. Existen funciones para operar sobre estos bits.

```
1. FD_ZERO(&fdset);          /* Inicializa todos los bits a cero */
2. FD_SET(fd,&fdset);        /* Pone a 1 el bit indicado por fd */
3. FD_CLR(fd,&fdset);        /* Pone a 0 el bit indicado por fd */
4. FD_ISSET(fd, &fdset);     /* Comprueba si el bit indicado por fd está a 1 */
```

Programming
language

Ejemplo de servidor utilizando select() (1)

```
int listener;                          //Socket que recibe las conexiones.
 int cliente;                          //cliente que se conecta.
 struct sockaddr_in datosServer;       //Datos del servidor.
 struct sockaddr_in datosCliente;      //Datos del cliente.
 int clienteConectado;                 //Flag que indica que el cliente se conecto.
 fd_set master;                        //Conjunbto maestro de descriptores de fichero.
 fd_set readset;                       //Conjunto temporal de descriptores de fichero para select()
 int fdmax;                            //Número mayor descriptor de fichero.
 int i;                                //Para el for del select.
 int on = 1;                           //Necesario para setsockopt.
 int addrlen;                          //Necesito para el accept.
 struct datos configuracion;  //IP y PUERTO del servidor.
 int nbytes;                  //Respuesta del servidor
 char respuesta[MAX_DATOS]; //Respuesta del servidor como que el cliente se conecto.
```

Programming
language

Ejemplo de servidor utilizando select() (2)

```
//Cargo ip y puerto en las variables.
datos_server(&configuracion);
//Creo el socket que escucha las conexiones.
listener = socket(AF_INET,SOCK_STREAM,0);
if(listener == -1){
   printf("Error en listener");
   return(-1);
 }
//Lleno la estructura con la información del servidor.
printf("Datos del server\nip:\"%s\"\npuerto:\"%d\"\n",configuracion.ip, configuracion.puerto);
datosServer.sin_family = AF_INET;
datosServer.sin_addr.s_addr = inet_addr(configuracion.ip);
datosServer.sin_port = htons(configuracion.puerto);
memset(datosServer.sin_zero,0,8);
```

Programming
language

Ejemplo de servidor utilizando select() (3)

```
/* Si el server se cierra bruscamente queda ocupado el puerto y se debe
reiniciar el servidor, con setsockp se soluciona. */
if(setsockopt(listener, SOL_SOCKET, SO_REUSEADDR,&on, sizeof(int)) == -1){
     printf("Error en setsockopt");
   return(-1);
}
//Enlazo el socket a la ip:puerto contenida en la estructura datosServer.
if(bind(listener, (struct sockaddr*) &datosServer, sizeof datosServer) == -1){
  printf("Error en bind");
   return(-1);
}
//Pongo el server a la escucha (enciendo el servidor)
if((listen(listener,MAX_CONEXIONES)) == -1){
  printf("Error en listen");
  return(-1);
}
```

Programming
language

Ejemplo de servidor utilizando select() (4)

```
//Aviso que el servidor está conectado.
puts("Servidor conectado");
FD_ZERO(&master);
FD_ZERO(&readset);
//Añado el listener al set maestro.
FD_SET(listener,&master);
//Mayor descriptor de fichero como es único es el máximo.
fdmax = listener;
```

Programming
language

Ejemplo de servidor utilizando select() (5)

```
while(1){              //Bucle principal.
   readset = master;
    if(select(fdmax+1,&readset, NULL, NULL, NULL)== -1){
      printf("Error en select");
      return(-1);
    }
    //Exploro conexiones existentes en busca de datos que leer.
   for(i = 0; i <= fdmax; i++){
      if(FD_ISSET(i,&readset)){
         if(i == listener){     //Tengo conexión para aceptar.
            addrlen = sizeof(datosCliente);
            cliente = accept(listener, (struct sockaddr*) &datosCliente, &addrlen);
            if(cliente == -1){
               printf("Error en accept");
            }
            FD_SET(cliente, &master); //Añado la nueva conexion al conjunto maestro.
            if(cliente > fdmax){     //actualizo el máximo fichero.
               fdmax = cliente;
            }
            printf("La siguiente IP se ha conectado IP: %s\n",inet_ntoa((struct in_addr)datosCliente.sin_addr));
            strcpy(respuesta,"Conexion aceptada");
            send(cliente, &respuesta, sizeof(respuesta), 0);
         }
      }
   }
}
```

Bibliografía

- Guía Beej de Programación en Redes – Brian Beej Hall
- Cómo Programar en C/C++ y Java – Deitel & Deitel 4ta edición
- El Lenguaje de Programación C – Kernighan & Ritchie
- Linux Man Pages
- UTN SO – www.utn.so (Cátedra perteneciente a Ing. en Sist. de Información)

¡ A trabajar !

Continuaremos en
Técnicas Digitales III