



ARQUITECTURA DE REDES Laboratorio

Práctica 2: **“Ejercicios de aplicación de Sockets en C”**

1. OBJETIVOS.

El objetivo de esta práctica es que el alumno llegue a conocer los principales conceptos relacionados con la comunicación de procesos usando Sockets TCP.

2. ACTIVIDADES.

- El alumno deberá estudiar los ejercicios cuyo código se adjunta.
- El alumno deberá diseñar, compilar y depurar los ejercicios propuestos utilizando el *Lenguaje C* sobre el S.O Linux.

3. EJERCICIOS.

Ejercicio 1: Resolución de los nombres de equipos.

En Linux esta información se puede obtener ejecutando el comando: *nslookup www.xxxx.yy* y nos devolverá su dirección IP. En programación se utilizarán las funciones *gethostbyname* y *gethostbyaddr*, para obtener dicha información.

El siguiente programa obtiene una estructura (*struct hostent*) de un equipo a partir de su dirección IP en notación de punto, y extrae de ella el nombre para mostrarlo en pantalla. Por ejemplo si ejecutamos:

Ejemplo1: `>./resolucion 195.53.213.16`

El resultado obtenido debería ser: `>195.53.213.16 www.iberia.es`

Ejemplo2: `>./resolucion 212.128.64.12`

El resultado obtenido debería ser: `>212.128.64.12 intranet.uah.es`

```
/*resolucion.c */

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

int main(int argc, const char *argv[])
{
    u_long addr;
    struct hostent *hp;
    char **p;

    if (argc != 2)
    {
        printf("Uso: %s direccion IP\n",argv[0]);
        exit(1);
    }

    /*Cambia del formato notación de punto a formato binario*/
    if ((addr=inet_addr(argv[1])) == -1 )
    {
        printf("La direccion IP tiene que estar en notacion x.x.x.x \n");
        exit(2);
    }

    /*Obtiene una estructura hostent con la información del host dado en binario.*/
    hp=gethostbyaddr( (char *)&addr, sizeof(addr),AF_INET);

    if (hp==NULL)
    {
        printf("No se pude encontrar la informacion sobre el equipo %s\n", argv[1]);
        exit(3);
    }
}
```

```
for (p=hp->h_addr_list; *p != 0; p++)
{
    struct in_addr in;
    memcpy(&in.s_addr, *p, sizeof(in.s_addr));
    /*Pasa el binario de la tabla a in.s_addr porque esa estructura la necesita
    inet_ntoa, para pasarla a formato notación de punto */

    printf("La dirección IP (%s) corresponde a %s \n", inet_ntoa(in),hp->h_name);
}
exit(0);
}
```

Práctica 1: Resolución de nombres

Crear un programa (*num_ip.c*) que tomando como argumento de entrada el nombre del equipo, escriba por pantalla la/s dirección/es IP que tiene asignada/s y su/s posibles alias.

Ejercicio 2: Conexiones en el dominio de Internet.

En el siguiente ejemplo, el programa *cliente.c* va a crear un socket que se encargará de realizar una conexión con el servidor, le enviará un mensaje (*DATA*) y terminará (conexión incluida).

El programa *servidor.c* creará un socket que se encargará de calcular una dirección asociándosela a *sock*, se conectará al cliente con un nuevo socket (*msgsock*) que será a través del que realice el intercambio de información. Una vez terminada la conexión esperará la llegada de una nueva.

Nota: El S.O. asigna dinámicamente un puerto si se intenta establecer una conexión y no ha habido un `bind()` previo. El puerto del cliente no aparece explícitamente en ninguna parte pero es necesario, ya que una conexión TCP precisa de dos puertos.

```

/* cliente.c */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#define SERV_ADDR (IPPORT_RESERVED+1)
#define DATA "###---###---###-----***-----###---###---###"

int main (int argc, char *argv[])
{
    int sock;
    struct sockaddr_in server;
    struct hostent *hp;
    if (argc<2)
    {
        printf("Uso: %s como nombre de host\n",argv[0]);
        exit(1);
    }
    sock=socket(AF_INET,SOCK_STREAM,0);

    if (sock<0)
    {
        perror("No se ha podido crear el socket");
        exit(1);
    }
    server.sin_family =AF_INET;
    hp=gethostbyname(argv[1]);

```

```
if (hp==0)
{
    fprintf(stderr, "%s: No conozco ese computador\n",argv[1]);
    exit(2);
}
memcpy((char *)&server.sin_addr, (char *)hp->h_addr, hp->h_length);
server.sin_port = htons (SERV_ADDR);

if (connect(sock, (struct sockaddr *)&server, sizeof(server))<0)
{
    perror("La conexion no sido aceptada");
    exit(1);
}

if (write(sock,DATA, strlen(DATA)+1)<0)
    perror("No he podido escribir el mensaje");

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

```
/* servidor.c */
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#define STDOUT 1
#define SERV_ADDR (IPPORT_RESERVED+1)

int main()
{
    int rval;
    int sock,length,msgsock;
    struct sockaddr_in server;
    char buf[1024];

    sock=socket(AF_INET, SOCK_STREAM,0);

    if (sock<0)
    {
        perror("No hay socket de escucha");
        exit(1);
    }

    server.sin_family=AF_INET;
    server.sin_addr.s_addr=htonl(INADDR_ANY);
    server.sin_port = htons (SERV_ADDR);

    if (bind(sock,(struct sockaddr *)&server, sizeof(server))<0)
    {
        perror("Direccion no asignada");
        exit(1);
    }

    listen(sock,1);
    while (1)
    {
        /*Estará bloqueado esperando petición de conexión*/
        msgsock = accept(sock, (struct sockaddr *)0, (int *) 0);

        if (msgsock== -1)
            perror("Conexion no aceptada");
        else
            do
            {
                /*Me dispongo a leer datos de la conexión*/
                memset(buf,0,sizeof(buf));
                rval=read(msgsock,buf,1024);
            }
```

```
    if (rval<0)
        perror("Mensaje no leído");
    else
        write(STDOUT,buf,rval);
    }
    while (rval>0);

    printf("\nConexion cerrada\n");
    close(msgsock);
}
exit(0);
}
```

----- -----

Para su funcionamiento:

- Ejecutar: `>./servidor &` (comprobar con el comando **ps** que el servidor esta funcionando)
- Ejecutar: `>./cliente localhost`

Práctica 2: Identificación de clientes.

A partir de los programas anteriores, modificar *servidor.c* para que durante su ejecución imprima por pantalla los datos de los clientes conectados a él:

Número de puerto ; Dirección IP ; Nombre del equipo (correspondiente a esa dirección IP).

Ejercicio 3: Servidor de eco secuencial.

El siguiente ejemplo es un servidor secuencial de eco (*servidor_eco.c*), que simplemente retransmite al cliente los datos que éste le ha enviado. El cliente es un programa que envía al servidor todo lo que lee de la entrada estándar e imprime lo que éste le retransmite.

Si se ejecutan varios clientes a la vez, debido al carácter secuencial del servidor y al modelo de mantenimiento de conexión, cada cliente debe esperar a que terminen los clientes previos para obtener servicio.

```
/* servidor_eco.c */
```

```
/*Nota: Se ha incluido invocaciones al mandato netstat que permiten observar la evolución en el estado de los sockets involucrados.*/
```

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

```
#define TAM 64
```

```
void traza_estado(const char *mensaje)
{
    printf("\n----- %s ----- \n", mensaje);
    system("netstat -at | head -2 | tail -1");
    system("netstat -at | grep 56789");
    printf("----- \n\n");
}
```

```
int main(int argc, char *argv[])
{
    int s, s_conec, leído;
    unsigned int tam_dir;
    struct sockaddr_in dir, dir_cliente;
    char buf[TAM];
    int opcion=1;
```

```
if ((s=socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
{
    perror("Error creando socket");
    return 1;
}

/* Para reutilizar puerto inmediatamente */
if (setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &opcion, sizeof(opcion))<0)
{
    perror("Error en setsockopt");
    return 1;
}

dir.sin_addr.s_addr=INADDR_ANY;
dir.sin_port=htons(56789);
dir.sin_family=PF_INET;

if (bind(s, (struct sockaddr *)&dir, sizeof(dir)) < 0)
{
    perror("Error en bind");
    close(s);
    return 1;
}

if (listen(s, 5) < 0)
{
    perror("Error en listen");
    close(s);
    return 1;
}

traza_estado("Despues de listen");

while(1)
{
    tam_dir=sizeof(dir_cliente);

    if ((s_conec=accept(s, (struct sockaddr *)&dir_cliente, &tam_dir))<0)
    {
        perror("Error en accept");
        close(s);
        return 1;
    }

    traza_estado("Despues de accept");

    while((leido=read(s_conec, buf, TAM))>0)
    {
        if (write(s_conec, buf, leido)<0)
        {
            perror("Error en write");
            close(s);
            close(s_conec);
            return 1;
        }
    }

    if (leido<0)
    {
        perror("Error en read");
        close(s);
        close(s_conec);
        return 1;
    }
    close(s_conec);
    traza_estado("Despues de cierre de conexion");
}

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

```
/* cliente_eco.c */

#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#define TAM 64

int main(int argc, char *argv[])
{
    int s, leido;
    struct sockaddr_in dir;
    struct hostent *host_info;
    char buf[TAM];

    if ((s=socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
    {
        perror("Error creando socket");
        return 1;
    }

    host_info=gethostbyname("localhost");
    memcpy(&dir.sin_addr.s_addr, host_info->h_addr, host_info->h_length);
    dir.sin_port=htons(56789);
    dir.sin_family=PF_INET;

    if (connect(s, (struct sockaddr *)&dir, sizeof(dir)) < 0)
    {
        perror("Error en la conexion");
        close(s);
        return 1;
    }

    while ((leido=read(0, buf, TAM))>0)
    {
        if (write(s, buf, leido)<0)
        {
            perror("Error en write ");
            close(s);
            return 1;
        }
        if ((leido=read(s, buf, TAM))<0)
        {
            perror("Error en read ");
            close(s);
            return 1;
        }
        write(1, buf, leido);
    }
    return 0;
}
```

----- -----

Para su funcionamiento:

- Ejecutar: >./servidor
- Ejecutar: >./cliente

Práctica 3: Cliente con una sola conexión para servidor de eco secuencial.

Modificar el programa cliente anterior para que este use una sola conexión por cada petición realizada al servidor.

Ejercicio 4: Conversión de información binaria.

En el siguiente ejemplo se muestra el uso de las funciones de conversión de formato de enteros (*htonl*, *ntohl*, etc.) a la hora de transferir en binario datos de este tipo. En el ejercicio, el cliente solicitará al usuario que introduzca dos operandos para enviárselos al servidor, este realizará con ellos una *suma* y devolverá el resultado al cliente para que lo muestre en pantalla.

```
/* servidor.c */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    struct sockaddr_in server_addr, client_addr;
    int c,i,sd, sc, val;
    unsigned int size;
    long int num[2], res;

    sd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    val = 1;
    setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, (char *) &val, sizeof(int));

    bzero((char *)&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(56789);

    bind(sd, (struct sockaddr *)&server_addr, sizeof(server_addr));
    listen(sd, 5);
    size = sizeof(client_addr);

    while (1)
    {
        printf("Esperando conexion\n");
        sc = accept(sd, (struct sockaddr *)&client_addr,&size);

        for(i=0;
            i<2*sizeof(long int)&&(c=read(sc,((char *)num)+i,2*sizeof(long int)-i))>0;
            i+=c); /* recibe la petición */

        res = htonl(ntohl(num[0]) + ntohl(num[1]));
        write(sc, &res, sizeof(long int)); /* se envía el resultado */

        close(sc);
    }
    close (sd);
    return 0;
}

-----

/* cliente.c */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
    int i,c,sd;
    struct sockaddr_in server_addr;
    struct hostent *hp;
    long int num[2], res;
```



```
if (argc!=3)
{
    fprintf(stderr, "Uso: %s primer_sumando segundo_sumando\n", argv[0]);
    return 1;
}
sd = socket(PAF_INET, SOCK_STREAM, IPPROTO_TCP);

bzero((char *)&server_addr, sizeof(server_addr));
hp = gethostbyname ("localhost");

memcpy (&(server_addr.sin_addr), hp->h_addr, hp->h_length);
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(56789);
/* se establece la conexión */
connect(sd, (struct sockaddr *) &server_addr, sizeof(server_addr));

num[0]=htonl(atoi(argv[1]));
num[1]=htonl(atoi(argv[2]));

write(sd, (char *) num, 2 *sizeof(long int));    /*envía la petición*/

for(i=0;
    i<sizeof(long int)&&(c=read(sd, ((char *)&res)+i, sizeof(long int)-i))>0;i+=c);
/* recibe la respuesta */

printf("El resultado es: %d \n", ntohl(res));
close (sd);

return 0;
}
```

Para su funcionamiento:

- Ejecutar: >./servidor
- Ejecutar: >./cliente

Práctica 4: Conversión de valores numéricos.

Realiza las modificaciones oportunas en los programas anteriores para conseguir que se realice una multiplicación de operandos, devolviendo el resultado y mostrándolo por pantalla.

ANEXO

Estructuras para direcciones

