

>> Ejemplo 1 – Sockets TCP/IP

Los sockets de Unix pueden ser usados para comunicación entre dos procesos que se encuentran en la misma computadora. Los sockets basados Internet pueden ser usados para tener comunicación entre diferentes computadoras a través de una red y se componen de dos partes: una computadora y un puerto. Esta información es almacenada en una estructura cuya sintaxis se muestra a continuación:

```
struct sockaddr_in {
    short int sin_family;    /* AF_INET */
    uint16_t sin_port;      /* Port number */
    struct in_addr sin_addr; /* IP address */
};
```

sin_port es el puerto al que se va a conectar
sin_addr es la dirección IP.

Los parámetros antes descritos deben de estar en representación binaria. Dado que los humanos pueden aprender más fácilmente una palabra que una serie de números, se hace uso del Domain Name Service (DNS), pero las máquinas entienden números en representación binaria, por lo que tenemos que transformar el nombre en su representación numérica. La función *gethostbyname()* transforma el nombre en una dirección IP. Una vez que se tiene la IP, ésta se tiene que transformar a su representación binaria debido a que *sin_addr* lo necesita de esa manera; además de la IP, también el puerto se debe de cambiar a su representación binaria. Para facilitar este proceso se tienen las siguientes funciones:

```
char *inet_ntoa(struct in_addr addr);
char *inet_aton(const char *ddaddr, struct in_addr *ipaddr);
```

inet_ntoa para convertir de binario a decimal.

inet_aton para convertir de decimal a binario.

El siguiente código crea un socket a un sitio de Internet para extraer su información por medio del comando GET del protocolo HTTP.

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

/* Print the contents of the home page for the server's socket.
```

Return an indication of success. */

```
void get_home_page (int socket_fd)
{
    char buffer[10000];
    ssize_t number_characters_read;

    /* Send the HTTP GET command for the home page. */
    sprintf (buffer, "GET /\n");
    write (socket_fd, buffer, strlen (buffer));

    /* Read from the socket. The call to read may not
    return all the data at one time, so keep
    trying until we run out. */
    while (1) {
        number_characters_read = read (socket_fd, buffer, 10000);
        if (number_characters_read == 0)
            return;

        /* Write the data to standard output. */
        fwrite (buffer, sizeof (char), number_characters_read, stdout);
    }
}

int main (int argc, char* const argv[]){

    int socket_fd;
    struct sockaddr_in name;
    struct hostent* hostinfo;

    /* Create the socket. */
    socket_fd = socket (PF_INET, SOCK_STREAM, 0);

    /* Store the server's name in the socket address. */
    name.sin_family = AF_INET;

    /* Convert from strings to numbers. */
    hostinfo = gethostbyname (argv[1]);

    if (hostinfo == NULL)
        return 1;
    else
        name.sin_addr = *((struct in_addr *) hostinfo->h_addr);

    /* Web servers use port 80. */
    name.sin_port = htons (80);

    /* Connect to the Web server */
    if (connect (socket_fd, (struct sockaddr *)&name, sizeof (struct sockaddr_in)) == -1) {
        perror ("connect");
        return 1;
    }

    /* Retrieve the server's home page. */
    get_home_page (socket_fd);
```

```
return 0;
```

```
}
```

Código 5 Extrayendo información de una página por el método GET (mkget.c)

La función `gethostbyname()` regresa un apuntador *hostinfo* apunta a una estructura llamada *hostent* que se describe a continuación.

```
struct hostent
{
    char *h_name;      /* Nombre del host*/
    char **h_aliases;  /* Arreglo de nombres alternos del host*/
    int  h_addrtype;   /*AF_INET*/
    int  h_length;     /*Tamaño de la dirección en bytes*/
    char **h_addr_list; /* Arreglo de direcciones de internet*/
};
#define h_addr h_addr_list[0]
```



```
[root@vesta CLinux]# ./exe www.google.com
HTTP/1.0 302 Found
Location: http://www.google.com.mx/
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Set-Cookie: PREF=ID=3c95dad4dbbe4a15:FF=0:TM=1330395119:LM=1330395119:S=QYFjq79e7msNKYuM; expires=Thu, 27-Feb-2014 02:11:59 GMT; path=/; domain=.google.com
Date: Tue, 28 Feb 2012 02:11:59 GMT
Server: gws
Content-Length: 222
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.com.mx/">here</A>.
</BODY></HTML>
```

El siguiente código se encarga de obtener las direcciones IP asociadas a un dominio.

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

```

#include <arpa/inet.h>

int main(int argc, char *argv[ ])
{
    struct hostent *h;

    /* error check the command line */
    if(argc != 2)
    {
        fprintf(stderr, "Usage: %s <domain_name>\n", argv[0]);
        exit(1);
    }

    /* get the host info */
    if((h=gethostbyname(argv[1])) == NULL)
    {
        perror("gethostbyname(): ");
        exit(1);
    }
    else
        printf("gethostbyname() is OK.\n");

    printf("The host name is: %s\n", h->h_name);
    printf("The IP Address is: %s\n", inet_ntoa(*(struct in_addr *)h->h_addr));
    printf("The address length is: %d\n", h->h_length);

    printf("Sniffing other names...sniff...sniff...sniff...\n");
    int j = 0;
    do
    {
        printf("An alias #%d is: %s\n", j, h->h_aliases[j]);
        j++;
    }
    while(h->h_aliases[j] != NULL);

    printf("Sniffing other IPs...sniff....sniff...sniff...\n");
    int i = 0;
    do
    {
        printf("Address #%i is: %s\n", i, inet_ntoa(*(struct in_addr *)h->h_addr_list[i]));
        i++;
    }
    while(h->h_addr_list[i] != NULL);
    return 0;
}

```

Código 6 Extrae las direcciones Ip de un dominio (ipaddr.c)

En el código `h->h_addr` es un `char*` e `inet_ntoa()` requiere una estructura de tipo `in_addr`. Es por esta razón que se hace un cast a una estructura `in_addr` y después se hace la desreferencia para obtener los datos.

```
[root@vesta CLinux]# ./exe www.google.com
gethostbyname() is OK.
The host name is: www.l.google.com
The IP Address is: 74.125.227.144
The address length is: 4
Sniffing other names...sniff...sniff...sniff...
An alias #0 is: www.google.com
Sniffing other IPs...sniff...sniff...sniff...
Address #0 is: 74.125.227.144
Address #1 is: 74.125.227.147
Address #2 is: 74.125.227.148
Address #3 is: 74.125.227.145
Address #4 is: 74.125.227.146
```

El siguiente código muestra la interacción de un servidor con uno o varios clientes. El servidor va a estar escuchando por el puerto 3490, esperando a que algún cliente se conecte. La función *sigaction()* es responsable de eliminar a los procesos zombis que se crean por el uso de *fork()* para la creación de procesos hijos.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>

/* the port users will be connecting to */
#define MYPORT 3490
/* how many pending connections queue will hold */
#define BACKLOG 10

void sigchld_handler(int s)
{
    while(wait(NULL) > 0);
}

int main(int argc, char *argv[ ])
{
    /* listen on sock_fd, new connection on new_fd */
    int sockfd, new_fd;
    /* my address information */
    struct sockaddr_in my_addr;
    /* connector's address information */
    struct sockaddr_in their_addr;
    int sin_size;
    struct sigaction sa;
    int yes = 1;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
```

```

{
    perror("Server-socket() error lol!");
    exit(1);
}
else
    printf("Server-socket() sockfd is OK...\n");

if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1)
{
    perror("Server-setsockopt() error lol!");
    exit(1);
}
else
    printf("Server-setsockopt is OK...\n");

/* host byte order */
my_addr.sin_family = AF_INET;
/* short, network byte order */
my_addr.sin_port = htons(MYPORT);
/* automatically fill with my IP */
my_addr.sin_addr.s_addr = INADDR_ANY;

printf("Server-Using %s and port %d...\n", inet_ntoa(my_addr.sin_addr), MYPORT);

/* zero the rest of the struct */
memset(&(my_addr.sin_zero), '\0', 8);

if(bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1)
{
    perror("Server-bind() error");
    exit(1);
}
else
    printf("Server-bind() is OK...\n");

if(listen(sockfd, BACKLOG) == -1)
{
    perror("Server-listen() error");
    exit(1);
}
printf("Server-listen() is OK...Listening...\n");

/* clean all the dead processes */
sa.sa_handler = sigchld_handler;
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;

if(sigaction(SIGCHLD, &sa, NULL) == -1)
{
    perror("Server-sigaction() error");
    exit(1);
}
else
    printf("Server-sigaction() is OK...\n");

/* accept() loop */

```

```

while(1)
{
    sin_size = sizeof(struct sockaddr_in);
    if((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -1)
    {
        perror("Server-accept() error");
        continue;
    }
    else
        printf("Server-accept() is OK...\n");
    printf("Server-new socket, new_fd is OK...\n");
    printf("Server: Got connection from %s\n", inet_ntoa(their_addr.sin_addr));

    /* this is the child process */
    if(!fork())
    {
        /* child doesn't need the listener */
        close(sockfd);

        if(send(new_fd, "This is a test string from server!\n", 37, 0) == -1)
            perror("Server-send() error lol!");
        close(new_fd);
        exit(0);
    }
    else
        printf("Server-send is OK...!\n");

    /* parent doesn't need this */
    close(new_fd);
    printf("Server-new socket, new_fd closed successfully...\n");
}
return 0;
}

```

Código 7 Servidor, utilizando sockets TCP/IP (serverstream.c)

```

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

// the port client will be connecting to
#define PORT 3490
// max number of bytes we can get at once
#define MAXDATASIZE 300

int main(int argc, char *argv[])
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct hostent *he;

```

```

// connector's address information
struct sockaddr_in their_addr;

// if no command line argument supplied
if(argc != 2)
{
    fprintf(stderr, "Client-Usage: %s the_client_hostname\n", argv[0]);
    // just exit
    exit(1);
}

// get the host info
if((he=gethostbyname(argv[1])) == NULL)
{
    perror("gethostbyname()");
    exit(1);
}
else
    printf("Client-The remote host is: %s\n", argv[1]);

if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{
    perror("socket()");
    exit(1);
}
Código 8 Cliente, utilizando sockets TCP/IP (clientstream.c)
else
    printf("Client-The socket() sockfd is OK...\n");

// host byte order
their_addr.sin_family = AF_INET;
// short, network byte order
printf("Server-Using %s and port %d...\n", argv[1], PORT);
their_addr.sin_port = htons(PORT);
their_addr.sin_addr = *((struct in_addr *)he->h_addr);
// zero the rest of the struct
memset(&(their_addr.sin_zero), '\0', 8);

if(connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
{
    perror("connect()");
    exit(1);
}
else
    printf("Client-The connect() is OK...\n");


if((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1)
{
    perror("recv()");
    exit(1);
}
else
    printf("Client-The recv() is OK...\n");

buf[numbytes] = '\0';
printf("Client-Received: %s", buf);

```

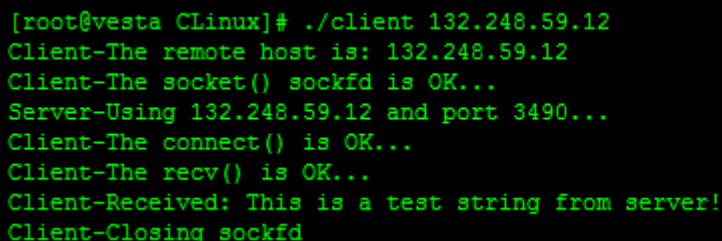
```
printf("Client-Closing sockfd\n");
close(sockfd);
return 0;
}
```

La siguiente imagen muestra la salida del servidor, cuando recibe una petición por parte del cliente manda una cadena al cliente diciendo: "This is a test string from server". Esta cadena es impresa en la salida del cliente que solicitó la conexión. Mientras no se termine el proceso del servidor, éste seguirá esperando clientes que se conecten a él.

A terminal window with a black background and green text. The text shows the execution of a server program. It starts with a prompt [root@vesta CLinux]# ./server. The output includes status messages for socket creation, binding to 0.0.0.0 on port 3490, and listening. It then shows two successful connections from 132.248.59.12 and 132.248.59.43, each followed by a successful send operation. The last line shows a successful close of the socket for the second connection.

```
[root@vesta CLinux]# ./server
Server-socket() sockfd is OK...
Server-setsockopt is OK...
Server-Using 0.0.0.0 and port 3490...
Server-bind() is OK...
Server-listen() is OK...Listening...
Server-sigaction() is OK...
Server-accept() is OK...
Server-new socket, new_fd is OK...
Server: Got connection from 132.248.59.12
Server-send is OK...!
Server-new socket, new_fd closed successfully...
Server-accept() is OK...
Server-new socket, new_fd is OK...
Server: Got connection from 132.248.59.43
Server-send is OK...!
Server-new socket, new_fd closed successfully...
```

Las siguientes imágenes muestran las salidas de dos diferentes clientes que se conectaron al servidor y que recibieron la cadena por parte de éste.

A terminal window with a black background and green text. The text shows the execution of a client program. It starts with a prompt [root@vesta CLinux]# ./client 132.248.59.12. The output includes status messages for socket creation, connecting to 132.248.59.12 on port 3490, receiving the test string "This is a test string from server!", and finally closing the socket.

```
[root@vesta CLinux]# ./client 132.248.59.12
Client-The remote host is: 132.248.59.12
Client-The socket() sockfd is OK...
Server-Using 132.248.59.12 and port 3490...
Client-The connect() is OK...
Client-The recv() is OK...
Client-Received: This is a test string from server!
Client-Closing sockfd
```

```
[root@lestat tmp]# ./client 132.248.59.12
Client-The remote host is: 132.248.59.12
Client-The socket() sockfd is OK...
Server-Using 132.248.59.12 and port 3490...
Client-The connect() is OK...
Client-The recv() is OK...
Client-Received: This is a test string from server!
Client-Closing sockfd
```

4. CUESTIONARIO

- [1] Hacer que el servidor imprima lo que recibe del cliente.
- [2] Hacer que el servidor soporte más de un cliente.

5. BIBLIOGRAFÍA

- [1] Francisco Manuel Márquez García. “Unix: programación avanzada”. Editorial RA-MA
- [2] Richard Stevens. “Advanced Unix programming”. Editorial Addison Wesley
- [3] Kurt Wall. “Linux Programming by Example”. Editorial QUE
- [4] <http://mermaja.act.uji.es/docencia/ii22/teoria/TraspasTema2.pdf>
- [5] <http://blog.txipinet.com/2006/11/05/48-curso-de-programacion-en-c-para-gnu-linux-vii/>
- [6] <http://www.dlsi.ua.es/asignaturas/sid/JSockets.pdf>
- [7] <http://www.tenouk.com/cnlinuxsockettutorials.html>