DESARROLLO

>> Ejemplo 1 – Sockets locales

Los sockets que conectan procesos en la misma computadora pueden usar los espacios de nombres representados por *PF_LOCAL* y *PF_UNIX*. El nombre del socket es especificado en una estructura llamada *sckaddr_un* definida en <sys/un.h> cuya sintaxis se muestra a continuación:

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

Sólo los procesos que se están ejecutando en la misma computadora se pueden comunicar por sockets con espacios de nombres locales. Debido a que los sockets locales residen en el sistema de archivos, un socket local es listado como un archivo. Para remover a un socket se utiliza la llamada a *unlink()*.

El siguiente código muestra un servidor que está escuchando peticiones de los clientes. Cuando llega un mensaje por parte del cliente, el servidor lo imprime en pantalla y espera por más mensajes.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>


int server (int client_socket)
{
    while (1) {
        int length;
        char* text;

        /* First, read the length of the text message from the socket. If
        read returns zero, the client closed the connection. */
        if (read (client_socket, &length, sizeof (length)) == 0)
            return 0;

        /* Allocate a buffer to hold the text. */
        text = (char*) malloc (length);

        /* Read the text itself, and print it. */
        read (client_socket, text, length);
        printf ("%s\n", text);
```

```
            /* If the client sent the message "quit," we're all done. */
            if (!strcmp (text, "quit"))
                return 1;
            else{
                /* Free the buffer. */
                free (text);
            }
        }
}

int main (int argc, char* const argv[])
{
    int socket_fd;          /*file descriptor for socket*/
    struct sockaddr_un name;  /*server socket structure*/
    int client_sent_quit_message;
    const char* const socket_name = argv[1];

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

    /* Indicate that this is a server. */
    name.sun_family = AF_LOCAL;
    strcpy (name.sun_path, socket_name);
    bind (socket_fd, (struct sockaddr *)&name, SUN_LEN (&name));

    /* Listen for connections. */
    listen (socket_fd, 5);

    /* Repeatedly accept connections, spinning off one server() to deal
    with each client. Continue until a client sends a "quit" message. */
    do {
        struct sockaddr_un client_name;
        socklen_t client_name_len;
        int client_socket_fd;

        /* Accept a connection. */
        client_socket_fd = accept (socket_fd, (struct sockaddr *)&client_name, &client_name_len);

        /* Handle the connection. */
        client_sent_quit_message = server (client_socket_fd);

        /* Close our end of the connection. */
        close (client_socket_fd);
    }while (!client_sent_quit_message);

    /* Remove the socket file. */
    close (socket_fd);
    unlink (socket_name);
    return 0;
}
```

Código 1 Servidor, imprime los mensajes del cliente (socket-server.c)

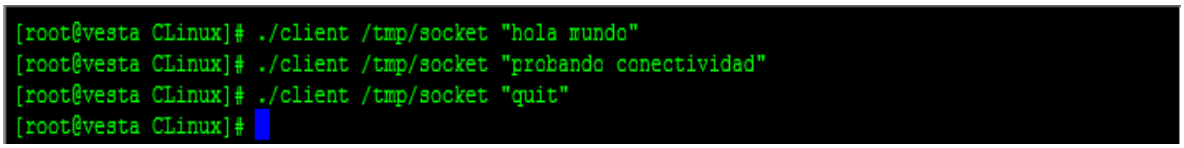
```c
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

/* Write TEXT to the socket given by file descriptor SOCKET_FD. */
void write_text (int socket_fd, const char* text)
{
	/* Write the number of bytes in the string, including
	NUL-termination. */
	int length = strlen (text) + 1;
	write (socket_fd, &length, sizeof (length));

	/* Write the string. */
	write (socket_fd, text, length);
}

int main (int argc, char* const argv[])
{
	const char* const socket_name = argv[1];
	const char* const message = argv[2];
	int socket_fd;
	struct sockaddr_un name;

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

	/* Store the server's name in the socket address. */
	name.sun_family = AF_LOCAL;
	strcpy (name.sun_path, socket_name);

	/* Connect the socket. */
	connect (socket_fd, (struct sockaddr *)&name, SUN_LEN (&name));

	/* Write the text on the command line to the socket. */
	write_text (socket_fd, message);
	close (socket_fd);
	return 0;
}
```

Código 2 Cliente, manda un mensaje al servidor (socket-cliente.c)

Antes de que el cliente mande el mensaje de texto, manda el tamaño del mismo para crear un buffer con el tamaño apropiado para poder almacenar el texto antes de leerlo del socket.

La siguiente imagen muestra la salida del servidor y los mensajes que recibió por parte del cliente a través del socket.

La siguiente imagen muestra los mensajes que el cliente manda al servidor a través del socket.



Los siguientes códigos muestran la interacción de un cliente y un servidor, el cliente le manda un mensaje al servidor, éste lo obtiene y se lo manda de nuevo al cliente.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

#define SOCK_PATH "echo_socket"

int main(void)
{
	int s, s2, t, len;
	struct sockaddr_un local, remote;
	char str[100];

	if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
		perror("socket");
		exit(1);
	}

	local.sun_family = AF_UNIX;
	strcpy(local.sun_path, SOCK_PATH);
	unlink(local.sun_path);
	len = strlen(local.sun_path) + sizeof(local.sun_family);
	if (bind(s, (struct sockaddr *)&local, len) == -1) {
		perror("bind");
		exit(1);
	}

	if (listen(s, 5) == -1) {
		perror("listen");
```

```
                exit(1);
        }

        for(;;) {
                int done, n;
                printf("Waiting for a connection...\n");
                t = sizeof(remote);
                if ((s2 = accept(s, (struct sockaddr *)&remote, &t)) == -1) {
                        perror("accept");
                        exit(1);
                }

                printf("Connected.\n");

                done = 0;
                do {
                        n = recv(s2, str, 100, 0);
                        if (n <= 0) {
                                if (n < 0) perror("recv");
                                done = 1;
                        }

                        if (!done)
                                if (send(s2, str, n, 0) < 0) {
                                        perror("send");
                                        done = 1;
                                }
                } while (!done);

                close(s2);
        }

        return 0;
}
```

Código 3 Servidor echo (echos.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

#define SOCK_PATH "echo_socket"

int main(void)
{
   int s, t, len;
   struct sockaddr_un remote;
   char str[100];

   if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
      perror("socket");
      exit(1);
   }
```

```
    printf("Trying to connect...\n");

    remote.sun_family = AF_UNIX;
    strcpy(remote.sun_path, SOCK_PATH);
    len = strlen(remote.sun_path) + sizeof(remote.sun_family);
    if (connect(s, (struct sockaddr *)&remote, len) == -1) {
        perror("connect");
        exit(1);
    }

    printf("Connected.\n");

    while(printf("> "), fgets(str, 100, stdin), !feof(stdin)) {
        if (send(s, str, strlen(str), 0) == -1) {
            perror("send");
            exit(1);
        }

        if ((t=recv(s, str, 100, 0)) > 0) {
            str[t] = '\0';
            printf("echo> %s", str);
        } else {
            if (t < 0) perror("recv");
            else printf("Server closed connection\n");
            exit(1);
        }
    }

    close(s);

    return 0;
}
```

Código 4 Cliente echo (echoc.c)

La siguiente imagen muestra que el servidor está esperando a que se conecte un cliente para devolver la cadena que éste le mande.

La siguiente imagen muestra cómo el cliente manda mensaje al servidor y éste se los reenvía al cliente que los imprime en pantalla.