Práctica 2.5. Sockets

Objetivos

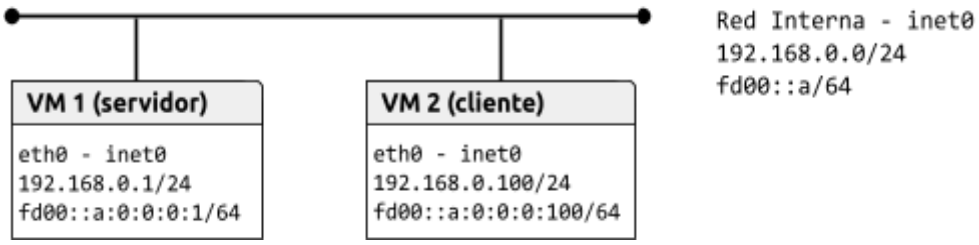
En esta práctica, nos familiarizaremos con la interfaz de programación de sockets como base para la programación de aplicaciones basadas en red, poniendo de manifiesto las diferencias de programación entre los protocolos UDP y TCP. Además, aprenderemos a programar aplicaciones independientes de la familia de protocolos de red (IPv4 o IPv6) utilizados.

Contenidos

Preparación del entorno de la práctica
Gestión de direcciones
Protocolo UDP - Servidor de hora
Protocolo TCP - Servidor de eco

Preparación del entorno de la práctica

Configuraremos la topología de red que se muestra en la figura. Como en prácticas anteriores construiremos la topología con la herramienta `vtopol`. Antes de comenzar la práctica, configurar los interfaces de red como se indica en la figura y comprobar la conectividad entre las máquinas.



Nota: Observar que las VMs tienen un interfaz de red con pila dual IPv6 - IPv4.

Gestión de direcciones

El uso del API BSD requiere la manipulación de direcciones de red y la traducción de estas entre las tres representaciones básicas: nombre de dominio, dirección IP (versión 4 y 6) y binario (para incluirla en la cabecera del datagrama IP).

Ejercicio 1. Escribir un programa que obtenga todas las posibles direcciones con las que se podría crear un socket asociado a un host dado como primer argumento del programa. Para cada dirección, mostrar la IP numérica, la familia de protocolos y tipo de socket. Comprobar el resultado para:

- Una dirección IPv4 válida (ej. "`147.96.1.9`").
- Una dirección IPv6 válida (ej. "`fd00::a:0:0:0:1`").
- Un nombre de dominio válido (ej. "`www.google.com`").
- Un nombre en `/etc/hosts` válido (ej. "`localhost`").
- Una dirección o nombre incorrectos en cualquiera de los casos anteriores.

El programa se implementará usando `getaddrinfo(3)` para obtener la lista de posibles direcciones de socket (**`struct`** `sockaddr`). Cada dirección se imprimirá en su valor numérico, usando `getnameinfo(3)` con el *flag* `NI_NUMERICHOST`, así como la familia de direcciones y el tipo de socket.

Nota: Para probar el comportamiento con DNS, realizar este ejercicio en la máquina física.

Ejemplos:

```
# Las familias 2 y 10 son AF_INET y AF_INET6, respectivamente (ver socket.h)
# Los tipos 1, 2, 3 son SOCK_STREAM, SOCK_DGRAM y SOCK_RAW, respectivamente
> ./gai www.google.com
66.102.1.147 2      1
66.102.1.147 2      2
66.102.1.147 2      3
2a00:1450:400c:c06::67    10     1
2a00:1450:400c:c06::67    10     2
2a00:1450:400c:c06::67    10     3
> ./gai localhost
::1    10     1
::1    10     2
::1    10     3
127.0.0.1    2      1
127.0.0.1    2      2
127.0.0.1    2      3
> ./gai ::1
::1    10     1
::1    10     2
::1    10     3
> ./gai 1::3::4
Error getaddrinfo(): Name or service not known
> ./gai noexiste.ucm.es
Error getaddrinfo(): Name or service not known
```

```
#include <sys/socket.h>
#include <netdb.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char ** argv){
   if(argc < 2) {
      printf("Olvidaste incluir la direccion.\n");
      return 1;
   }

   struct addrinfo * result;
   char host[NI_MAXHOST];
   char serv[NI_MAXSERV];

   if(getaddrinfo(argv[1], NULL, NULL, &result)){
      perror("No se pudo obtener las estructuras addrinfo \n.");
      return 1;
   }

   while(result != NULL){
      getnameinfo(result->ai_addr, result->ai_addrlen, host, NI_MAXHOST, serv, NI_MAXSERV,
NI_NUMERICHOST | NI_NUMERICSERV);
      printf("%s %i %i\n", host, result->ai_family, result->ai_socktype);
      result = result->ai_next;
   }
```

```
  return 0;
}
```

Protocolo UDP - Servidor de hora

Ejercicio 2. Escribir un servidor UDP de hora de forma que:

- La dirección y el puerto son el primer y segundo argumento del programa. Las direcciones pueden expresarse en cualquier formato (nombre de host, notación de punto…). Además, el servidor debe funcionar con direcciones IPv4 e IPv6 .
- El servidor recibirá un comando (codificado en un carácter), de forma que `'t'` devuelva la hora, `'d'` devuelve la fecha y `'q'` termina el proceso servidor.
- En cada mensaje el servidor debe imprimir el nombre y puerto del cliente, usar `getnameinfo(3)`.

Probar el funcionamiento del servidor con la herramienta Netcat (comando `nc` o `ncat`) como cliente.

Nota: Dado que el servidor puede funcionar con direcciones IPv4 e IPv6, hay que usar **`struct`** `sockaddr_storage` para acomodar cualquiera de ellas, por ejemplo, en `recvfrom(2)`.

Ejemplo:

```
$ ./time_server :: 3000
2 bytes de ::FFFF:192.168.0.100:58772
2 bytes de ::FFFF:192.168.0.100:58772
2 bytes de ::FFFF:192.168.0.100:58772
Comando X no soportado
2 bytes de ::FFFF:192.168.0.100:58772
Saliendo...
$
```

```
$ nc -u 192.168.0.1 3000
t
10:30:08 PMd
2014-01-14X
q
^C
$
```

Nota: El servidor no envía `'\n'`, por lo que se muestra la respuesta y el siguiente comando (en negrita en el ejemplo) en la misma línea.

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>
#include <time.h>
#include <sys/time.h>

#define BUF_SIZE 500

int main(int argc, char **argv){
        bool fin = 0;

        struct addrinfo hints;
```

```
struct addrinfo *result, *rp;
int sfd, s;
struct sockaddr_storage addr;
socklen_t addr_len;
ssize_t nread;
char buf[BUF_SIZE];

if (argc != 3) {
printf("Error: Necesito [dir] [port].\n");
return 1;
}

memset(&hints, 0, sizeof(struct addrinfo));
hints.ai_family = AF_UNSPEC;    /* Allow IPv4 or IPv6 */
hints.ai_socktype = SOCK_DGRAM; /* Datagram socket */
hints.ai_flags = 0;
hints.ai_protocol = 0;          /* Any protocol */
hints.ai_canonname = NULL;
hints.ai_addr = NULL;
hints.ai_next = NULL;

if (getaddrinfo(argv[1], argv[2], &hints, &result)) { /*(dir,puerto,hints,result)*/
perror("No funciono getaddrinf.\n");
return 1;
}

/* getaddrinfo() returns a list of address structures.
Try each address until we successfully bind(2).
If socket(2) (or bind(2)) fails, we (close the socket
and) try the next address. */

for (rp = result; rp != NULL; rp = rp->ai_next) {
sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol); /*Socket File Descriptor*/
if (sfd == -1)
continue;

if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
break;              /* Success */

close(sfd);
}

if (rp == NULL) {           /* No address succeeded */
printf("No se pudo bindear el socket.\n");
return 1;
}

freeaddrinfo(result);           /* No longer needed */

/* Read datagrams and echo them back to sender */

while (!fin) {
addr_len = sizeof(struct sockaddr_storage);

nread = recvfrom(sfd, buf, BUF_SIZE, 0, (struct sockaddr *) &addr, &addr_len);
if (nread == -1)
continue;           /* Ignore failed request */
```

```
        char host[NI_MAXHOST], service[NI_MAXSERV];

        s = getnameinfo((struct sockaddr *) &addr,
                addr_len, host, NI_MAXHOST,
                service, NI_MAXSERV, NI_NUMERICSERV);
        if (s == 0)
        printf("Received %zd bytes from %s:%s\n",
                nread, host, service);
        else
        fprintf(stderr, "getnameinfo: %s\n", gai_strerror(s));

        char q = buf[0];
        switch(q){
        case 't': {
        time_t tim = time(NULL);
        tm *t = localtime(&tim);
        sprintf(buf, "%i:%i:%i\n", t->tm_hour, t->tm_min, t->tm_sec);
        break;
        }
        case 'd': {
        time_t tim = time(NULL);
        tm *t = localtime(&tim);
        sprintf(buf, "%i-%i-%i\n", t->tm_mday, t->tm_mon + 1, t->tm_year + 1900);
        break;
        }
        case 'q': {
        fin = 1;
        printf("Cerrando servidor...\n");
        sprintf(buf, "Cerrando servidor...\n");
        break;
        }
        default: {
        printf("Error [%c] no es un caracter reconocido.\n", q);
        sprintf(buf, "Error [%c] no es un caracter reconocido.\n", q);
        break;
        }
        }

        if (sendto(sfd, buf, strlen(buf), 0, (struct sockaddr *) &addr, addr_len) != strlen(buf))
        fprintf(stderr, "Error enviando respuesta.\n");


        }
        freeaddrinfo(rp);
        free(buf);
        printf("Servidor cerrado.\n");
}
```

Ejercicio 3. Escribir el cliente para el servidor de hora. El cliente recibirá como argumentos la dirección del servidor, el puerto del servidor y el comando. Por ejemplo, para solicitar la hora, `./time_client 192.128.0.1 3000 t`.

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

#define BUF_SIZE 255

int main(int argc, char ** argv){

	if(argc < 3){
	printf("Error: Usage DIR PORT COMMAND.\n");
	return 1;
	}

	addrinfo hints;
	addrinfo * result, * rp;
	sockaddr_storage s_addr;
	socklen_t s_addrlen = sizeof(s_addr);
	int sfd = 0, rcv = 0;
	char buf[BUF_SIZE];

	#pragma region ConexionCS

	/* Filtros */
	memset(&hints, 0, sizeof(addrinfo));
	hints.ai_family = AF_UNSPEC;    /* Allow IPv4 or IPv6 */
	hints.ai_socktype = SOCK_DGRAM; /* Datagram socket */
	hints.ai_flags = 0;
	hints.ai_protocol = 0;          /* Any protocol */

	/* Direccion legible a dir maquina */
	if(getaddrinfo(argv[1], argv[2], &hints, &result)){
	printf("No se pudo obtener addrinfo.\n");
	return 1;
	}

	/* Iteramos sobre result para obtener el socketfiledescriptor */
	for(rp = result; rp != NULL; rp = rp->ai_next){
	sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
	if(sfd == -1)
	continue;           /* Si sfd no valido intentamos siguiente */
	if(connect(sfd, rp->ai_addr, rp->ai_addrlen) != -1)
	break;              /* Si conseguimos conectar salimos del bucle*/
	close(sfd);
	}

	if(rp == NULL){ /* No se pudo abrir el socket */
	perror("No se pudo abrir el socket.\n");
	return 1;
	}

	freeaddrinfo(result);
```

```
        #pragma endregion ConexionCS

        #pragma region Intercambio

        if(sendto(sfd, argv[3], 1+1, 0, rp->ai_addr, rp->ai_addrlen) == -1){
        printf("No se pudo enviar comando.\n");
        }

        rcv = recvfrom(sfd, buf, BUF_SIZE, 0, (sockaddr *) &s_addr, &s_addrlen);
        buf[rcv] = '\0';

        #pragma endregion Intercambio

        printf("%s\n", buf);
        close(sfd);

        return 0;
}
```

Ejercicio 4. Modificar el servidor para que, además de poder recibir comandos por red, los pueda recibir directamente por el terminal, leyendo dos caracteres (el comando y `'\n'`) de la entrada estándar. Multiplexar el uso de ambos canales usando `select(2)`.

```
/**
 * @file ej2504.cpp
 *
 * @author Enrique
 * @brief Esquema Servidor para el cliente ej2503.cpp modificado
 * @version 0.1
 * @date 2021-12-17
 *
 * @copyright Copyright (c) 2021
 *
 */

#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>
#include <time.h>
#include <sys/time.h>

#define BUF_SIZE        255
#define CONSOLE         0
#define CLIENT          1
#define NONE            -1
```

```
int main(int argc, char **argv){

	if (argc != 3) {
	printf("Error: Usage DIR PORT.\n");
	return 1;
	}

	addrinfo hints,*result, *rp;
	sockaddr_storage addr;
	socklen_t addr_len;
	ssize_t nread;
	bool fin = 0;
	int sfd, s;
	char buf[BUF_SIZE];

	#pragma region ConexionSC

	memset(&hints, 0, sizeof(struct addrinfo));
	hints.ai_family = AF_UNSPEC;    /* Allow IPv4 or IPv6 */
	hints.ai_socktype = SOCK_DGRAM; /* Datagram socket */
	hints.ai_flags = 0;
	hints.ai_protocol = 0;          /* Any protocol */
	hints.ai_canonname = NULL;
	hints.ai_addr = NULL;
	hints.ai_next = NULL;

	if (getaddrinfo(argv[1], argv[2], &hints, &result)) { /*(dir,puerto,hints,result)*/
	perror("No funciono getaddrinf.\n");
	return 1;
	}

	/* getaddrinfo() returns a list of address structures.
	Try each address until we successfully bind(2).
	If socket(2) (or bind(2)) fails, we (close the socket
	and) try the next address. */

	for (rp = result; rp != NULL; rp = rp->ai_next) {
	sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol); /*Socket File Descriptor*/
	if (sfd == -1)
	continue;

	if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
	break;              /* Success */

	close(sfd);
	}

	if (rp == NULL) {           /* No address succeeded */
	printf("No se pudo bindear el socket.\n");
	return 1;
	}

	freeaddrinfo(result);           /* No longer needed */

	#pragma endregion ConexionCS
```

```
#pragma region BucleLectura

fd_set fds; /* Para usar select */
int selected;
struct timeval timeout;
timeout.tv_sec = 20;
timeout.tv_usec = 0;
int dest = NONE;

while (!fin) {

FD_ZERO(&fds);
FD_SET(sfd, &fds);
FD_SET(0, &fds);            /* Entrada estandar */


selected = select(sfd+1,&fds, NULL, NULL, &timeout);

if(selected == -1) continue;

if(FD_ISSET(sfd, &fds)){
addr_len = sizeof(struct sockaddr_storage);

nread = recvfrom(sfd, buf, BUF_SIZE, 0, (struct sockaddr *) &addr, &addr_len);
if (nread == -1)
continue;           /* Ignore failed request */

char host[NI_MAXHOST], service[NI_MAXSERV];

s = getnameinfo((struct sockaddr *) &addr,
            addr_len, host, NI_MAXHOST,
            service, NI_MAXSERV, NI_NUMERICSERV);
if (s == 0)
printf("Received %zd bytes from %s:%s\n",
            nread, host, service);
else
fprintf(stderr, "getnameinfo: %s\n", gai_strerror(s));

dest = CLIENT;
}
else if(FD_ISSET(0, &fds)){
read(0, buf, 2);
dest = CONSOLE;
}

char q = buf[0];
switch(q){
case 't': {
time_t tim = time(NULL);
tm *t = localtime(&tim);
sprintf(buf, "%i:%i:%i\n", t->tm_hour, t->tm_min, t->tm_sec);
break;
}
case 'd': {
time_t tim = time(NULL);
tm *t = localtime(&tim);
```

```
            sprintf(buf, "%i-%i-%i\n", t->tm_mday, t->tm_mon + 1, t->tm_year + 1900);
            break;
            }
            case 'q': {
            fin = 1;;
            sprintf(buf, "Cerrando servidor...\n");
            break;
            }
            default: {
            sprintf(buf, "Error [%c] no es un caracter reconocido.\n", q);
            break;
            }
            }

            if(dest == CONSOLE){
            printf("%s",buf);
            }
            else if(dest == CLIENT){
            if (sendto(sfd, buf, strlen(buf), 0, (sockaddr *) &addr, addr_len) != strlen(buf))
            fprintf(stderr, "Error enviando respuesta.\n");
            }

            dest = NONE;
            }

            #pragma endregion BucleLectura
            printf("Servidor cerrado.\n");
            close(sfd);
}
```

Ejercicio 5. Convertir el servidor UDP en multi-proceso siguiendo el patrón *pre-fork*. Una vez asociado el socket a la dirección local con `bind(2)`, crear varios procesos que llamen a `recvfrom(2)` de forma que cada uno atenderá un mensaje de forma concurrente. Imprimir el PID del proceso servidor para comprobarlo. Para terminar el servidor, enviar la señal `SIGTERM` al grupo de procesos.

```
/**
 * @file ej2505.cpp
 *
 * @author Enrique
 * @brief Esquema Servidor para el cliente ej2503.cpp modificado2
 * @version 0.1
 * @date 2021-12-17
 *
 * @copyright Copyright (c) 2021
 *
 */

#include <sys/types.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>
#include <time.h>
#include <sys/time.h>
#include <sys/wait.h>

#define BUF_SIZE        255
#define N_PROC          2


int main(int argc, char **argv){

        if (argc != 3) {
        printf("Error: Usage DIR PORT.\n");
        return 1;
        }

        addrinfo hints,*result, *rp;
        sockaddr_storage addr;
        socklen_t addr_len = sizeof(sockaddr_storage);
        ssize_t nread;
        bool fin = 0;
        int sfd, s;
        char buf[BUF_SIZE];

        #pragma region ConexionSC

        memset(&hints, 0, sizeof(struct addrinfo));
        hints.ai_family = AF_UNSPEC;    /* Allow IPv4 or IPv6 */
        hints.ai_socktype = SOCK_DGRAM; /* Datagram socket */
        hints.ai_flags = 0;
        hints.ai_protocol = 0;                  /* Any protocol */
        hints.ai_canonname = NULL;
        hints.ai_addr = NULL;
        hints.ai_next = NULL;

        if (getaddrinfo(argv[1], argv[2], &hints, &result)) { /*(dir,puerto,hints,result)*/
        perror("No funciono getaddrinf.\n");
        return 1;
        }

        /* getaddrinfo() returns a list of address structures.
        Try each address until we successfully bind(2).
        If socket(2) (or bind(2)) fails, we (close the socket
        and) try the next address. */

        for (rp = result; rp != NULL; rp = rp->ai_next) {
        sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol); /*Socket File Descriptor*/
        if (sfd == -1)
        continue;

        if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
        break;                  /* Success */
```

```
close(sfd);
}

if (rp == NULL) {               /* No address succeeded */
printf("No se pudo bindear el socket.\n");
return 1;
}

freeaddrinfo(result);                   /* No longer needed */

#pragma endregion ConexionCS

pid_t pid;
for(int i = 0; i < N_PROC; ++i){
pid = fork();
if(pid <= 0) break;
}

switch (pid){
case -1:{
perror("Error forking.\n");
return 1;
break;
}

case 0:{
#pragma region BucleLectura

while (!fin) {

nread = recvfrom(sfd, buf, BUF_SIZE, 0, (sockaddr *) &addr, &addr_len);
if (nread == -1)
        continue;           /* Ignore failed request */

char host[NI_MAXHOST], service[NI_MAXSERV];

s = getnameinfo((sockaddr *) &addr,
        addr_len, host, NI_MAXHOST,
        service, NI_MAXSERV, NI_NUMERICSERV);
if (s == 0)
        printf("Received %zd bytes from %s:%s\n",
        nread, host, service);
else
        fprintf(stderr, "getnameinfo: %s\n", gai_strerror(s));

char q = buf[0];
switch(q){
        case 't': {
        time_t tim = time(NULL);
        tm *t = localtime(&tim);
        sprintf(buf, "%i:%i:%i\n", t->tm_hour, t->tm_min, t->tm_sec);
        break;
        }
        case 'd': {
        time_t tim = time(NULL);
        tm *t = localtime(&tim);
        sprintf(buf, "%i-%i-%i\n", t->tm_mday, t->tm_mon + 1, t->tm_year + 1900);
```

```
                    break;
                    }
                    case 'q': {
                    fin = 1;
                    sprintf(buf, "Cerrando servidor...\n");
                    break;
                    }
                    default: {
                    printf("Error [%c] no es un caracter reconocido.\n", q);
                    sprintf(buf, "Error [%c] no es un caracter reconocido.\n", q);
                    break;
                    }
            }

            if (sendto(sfd, buf, strlen(buf), 0, (struct sockaddr *) &addr, addr_len) != strlen(buf))
                    fprintf(stderr, "Error enviando respuesta.\n");

            if (fin){
                    close(sfd);
                    printf("Cerrando servidor\n");
                    kill(0, SIGTERM);
            }

            }

            #pragma endregion BucleLectura
            break;
            }

            default:break;

            }

            return 0;
}
```

Protocolo TCP - Servidor de eco

TCP nos ofrece un servicio orientado a conexión y fiable. Una vez creado el socket, debe ponerse en estado LISTEN (apertura pasiva, `listen(2)`) y a continuación quedarse a la espera de conexiones entrantes mediante una llamada `accept(2)`.

Ejercicio 6. Crear un servidor TCP de eco que escuche por conexiones entrantes en una dirección (IPv4 o IPv6) y puerto dados. Cuando reciba una conexión entrante, debe mostrar la dirección y número de puerto del cliente. A partir de ese momento, enviará al cliente todo lo que reciba desde el mismo (eco). Comprobar su funcionamiento empleando la herramienta Netcat como cliente. Comprobar qué sucede si varios clientes intentan conectar al mismo tiempo.

```
#include <sys/types.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>
#include <time.h>
#include <sys/time.h>
#include <sys/wait.h>

#define BUF_M 256
#define MAX_CONNECTION_REQUESTS 2

int main(int argc, char **argv){

        if(argc < 3){
        printf("Error: Usage SERV PORT.\n");
        return 1;
        }

        addrinfo hints, *result, *rp;
        int sfd, clsfd, rcv;

        memset(&hints, 0, sizeof(addrinfo));
        hints.ai_family = AF_UNSPEC;    /* Allow IPv4 or IPv6 */
        hints.ai_socktype = SOCK_STREAM;/* Stream socket */
        hints.ai_flags = 0;
        hints.ai_protocol = 0;                  /* Any protocol */
        hints.ai_canonname = NULL;
        hints.ai_addr = NULL;
        hints.ai_next = NULL;

        if(getaddrinfo(argv[1], argv[2], &hints, &result)){
        perror("Error: Bad SERV/PORT.\n");
        return 1;
        }

        for(rp = result; rp != NULL; rp = rp->ai_next){
        sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
        if (sfd == -1)
        continue;

        if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0){
        break;                  /* Success */
        }

        close(sfd);
        }

        if(rp == NULL){
        perror("Error: could not bind socket.\n");
        return 1;
        }

        freeaddrinfo(result);

        if(listen(sfd, MAX_CONNECTION_REQUESTS)){
        printf("Error: Listen.\n");
```

```
        return 1;
        }

        char host[NI_MAXHOST], serv[NI_MAXSERV], buf[BUF_M];

        sockaddr_storage addr;
        socklen_t addrlen = sizeof(addr);

        while(1){
        clsfd = accept(sfd, (sockaddr*)&addr, &addrlen);
        getnameinfo((sockaddr*)&addr, addrlen, host, NI_MAXHOST,
        serv, NI_MAXSERV, NI_NUMERICHOST | NI_NUMERICSERV);
        printf("Conexión desde %s:%s\n", host, serv);
        while(rcv = recv(clsfd,buf,BUF_M, 0)){
        buf[rcv] = '\0';
        printf("\tRecibido: %s\n", buf);
        send(clsfd, buf, rcv, 0);
        }
        close(clsfd);
        printf("Conexión terminada con %s:%s\n", host,serv);
        }

        return 0;
}
```

Ejemplo:

```
$ ./echo_server :: 2222
Conexión desde fd00::a:0:0:0:100 53456
Conexión terminada
```

```
$ nc -6 fd00::a:0:0:0:1 2222
Hola
Hola
Qué tal
Qué tal
^C
$
```

Ejercicio 7. Escribir el cliente para conectarse con el servidor del ejercicio anterior. El cliente recibirá la dirección y el puerto del servidor como argumentos y, una vez establecida la conexión con el servidor, le enviará lo que el usuario escriba por teclado. Mostrará en la consola la respuesta recibida desde el servidor. Cuando el usuario escriba el carácter `'Q'` como único carácter de una línea, el cliente cerrará la conexión con el servidor y terminará.

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

#define BUF_M 255

int main(int argc, char**argv){
```

```
if(argc < 3){
printf("Error: Usage SERV PORT.\n");
return 1;
}

addrinfo hints, *result, *rp;
int sfd;

memset(&hints, 0, sizeof(addrinfo));
hints.ai_family = AF_UNSPEC;     /* Allow IPv4 or IPv6 */
hints.ai_socktype = SOCK_STREAM;/* Stream socket */
hints.ai_flags = 0;
hints.ai_protocol = 0;                  /* Any protocol */
hints.ai_canonname = NULL;
hints.ai_addr = NULL;
hints.ai_next = NULL;

if(getaddrinfo(argv[1], argv[2], &hints, &result)){
perror("Error: Bad SERV/PORT.\n");
return 1;
}

for(rp = result; rp != NULL; rp = rp->ai_next){
sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
if (sfd == -1)
continue;

if (connect(sfd, rp->ai_addr, rp->ai_addrlen) == 0){
break;                  /* Success */
}

close(sfd);
}

if(rp == NULL){
perror("Error: could not bind socket.\n");
fprintf(stderr, "Maybe server isnt online\n");
return 1;
}

freeaddrinfo(result);

sockaddr_storage addr;
socklen_t addrlen = sizeof(addr);
char buf[BUF_M];

while(1){
int c = read(0, buf, BUF_M);
buf[c] = '\0';
if(buf[0] == 'Q' || buf[0] == 'q'){
close(sfd);
printf("Cerrando conexion...\n");
return 0;
}
send(sfd, buf, c, 0);
c = recv(sfd, buf, BUF_M, 0);
buf[c] = '\0';
```

```
        printf("%s\n",buf);
        }

        return 0;

}
```

Ejemplo:

```
$ ./echo_server :: 2222
Conexión desde fd00::a:0:0:0:100 53445
Conexión terminada
```

```
$ ./echo_client fd00::a:0:0:0:1 2222
Hola
Hola
Q
$
```

Ejercicio 8. Modificar el código del servidor para que acepte varias conexiones simultáneas. Cada petición debe gestionarse en un proceso diferente, siguiendo el patrón *accept-and-fork*. El proceso padre debe cerrar el socket devuelto por `accept(2)`.

```
/**
 * @file ej2508.cpp
 * @author Enrique
 * @brief Servidor TCP con esquema accept-and fork
 * @version 0.1
 * @date 2021-12-18
 *
 * @copyright Copyright (c) 2021
 *
 */


#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>
#include <time.h>
#include <sys/time.h>
#include <sys/wait.h>

#define BUF_M 256
#define MAX_CONNECTION_REQUESTS 2

int main(int argc, char **argv){

        if(argc < 3){
        printf("Error: Usage SERV PORT.\n");
        return 1;
        }

        addrinfo hints, *result, *rp;
```

```
int sfd, clsfd, rcv;

memset(&hints, 0, sizeof(addrinfo));
hints.ai_family = AF_UNSPEC;     /* Allow IPv4 or IPv6 */
hints.ai_socktype = SOCK_STREAM;/* Stream socket */
hints.ai_flags = 0;
hints.ai_protocol = 0;                  /* Any protocol */
hints.ai_canonname = NULL;
hints.ai_addr = NULL;
hints.ai_next = NULL;

if(getaddrinfo(argv[1], argv[2], &hints, &result)){
perror("Error: Bad SERV/PORT.\n");
return 1;
}

for(rp = result; rp != NULL; rp = rp->ai_next){
sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
if (sfd == -1)
continue;

if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0){
break;                /* Success */
}

close(sfd);
}

if(rp == NULL){
perror("Error: could not bind socket.\n");
return 1;
}

freeaddrinfo(result);

if(listen(sfd, MAX_CONNECTION_REQUESTS)){
printf("Error: Listen.\n");
return 1;
}

char host[NI_MAXHOST], serv[NI_MAXSERV], buf[BUF_M];

sockaddr_storage addr;
socklen_t addrlen = sizeof(addr);

while(1){
clsfd = accept(sfd, (sockaddr*)&addr, &addrlen);
pid_t pid = fork();
switch (pid){
case -1:
perror("Error forking.\n");
return 1;
break;

case 0:
getnameinfo((sockaddr*)&addr, addrlen, host, NI_MAXHOST,
            serv, NI_MAXSERV, NI_NUMERICHOST | NI_NUMERICSERV);
```

```
		printf("Conexión desde %s:%s\n", host, serv);
		while(rcv = recv(clsfd,buf,BUF_M, 0)){
			buf[rcv] = '\0';
			printf("\tRecibido: %s\n", buf);
			send(clsfd, buf, rcv, 0);
		}
		printf("Conexión terminada con %s:%s\n", host,serv);
		close(clsfd);
		return 0;
		break;

		default:
		close(clsfd);
		break;
		}
		}

		return 0;
}
```

Ejercicio 9. Añadir la lógica necesaria en el servidor para que no quede ningún proceso en estado *zombie*. Para ello, se deberá capturar la señal `SIGCHLD` y obtener la información de estado de los procesos hijos finalizados.

```
/**
 * @file ej2509.cpp
 * @author Enrique
 * @brief Servidor TCP con esquema accept-and fork
 * @version 0.1
 * @date 2021-12-18
 *
 * @copyright Copyright (c) 2021
 *
 */


#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>
#include <time.h>
#include <sys/time.h>
#include <sys/wait.h>

#include <signal.h> /* Sigaction */
```

```
#define BUF_M 256
#define MAX_CONNECTION_REQUESTS 2

void handler(int signal){
	int stat;
	wait(&stat);
	printf("Process ended with result: %i\n", stat);
}

int main(int argc, char **argv){

	if(argc < 3){
	printf("Error: Usage SERV PORT.\n");
	return 1;
	}

	addrinfo hints, *result, *rp;
	int sfd, clsfd, rcv;

	memset(&hints, 0, sizeof(addrinfo));
	hints.ai_family = AF_UNSPEC;    /* Allow IPv4 or IPv6 */
	hints.ai_socktype = SOCK_STREAM;/* Stream socket */
	hints.ai_flags = 0;
	hints.ai_protocol = 0;          /* Any protocol */
	hints.ai_canonname = NULL;
	hints.ai_addr = NULL;
	hints.ai_next = NULL;

	if(getaddrinfo(argv[1], argv[2], &hints, &result)){
	perror("Error: Bad SERV/PORT.\n");
	return 1;
	}

	for(rp = result; rp != NULL; rp = rp->ai_next){
	sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
	if (sfd == -1)
	continue;

	if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0){
	break;              /* Success */
	}

	close(sfd);
	}

	if(rp == NULL){
	perror("Error: could not bind socket.\n");
	return 1;
	}

	freeaddrinfo(result);

	if(listen(sfd, MAX_CONNECTION_REQUESTS)){
	printf("Error: Listen.\n");
	return 1;
	}
```

```
        char host[NI_MAXHOST], serv[NI_MAXSERV], buf[BUF_M];

        sockaddr_storage addr;
        socklen_t addrlen = sizeof(addr);

        struct sigaction act;
        act.sa_handler = handler;

        while(1){
        clsfd = accept(sfd, (sockaddr*)&addr, &addrlen);
        pid_t pid = fork();
        switch (pid){
        case -1:
        perror("Error forking.\n");
        return 1;
        break;

        case 0:
        getnameinfo((sockaddr*)&addr, addrlen, host, NI_MAXHOST,
                serv, NI_MAXSERV, NI_NUMERICHOST | NI_NUMERICSERV);
        printf("Conexión desde %s:%s\n", host, serv);
        while(rcv = recv(clsfd,buf,BUF_M, 0)){
                buf[rcv] = '\0';
                printf("\tRecibido: %s\n", buf);
                send(clsfd, buf, rcv, 0);
        }
        printf("Conexión terminada con %s:%s\n", host,serv);
        close(clsfd);
        kill(getppid(),SIGCHLD);
        return 0;
        break;

        default:
        sigaction(SIGCHLD,&act,NULL);
        close(clsfd);
        break;
        }
        }

        return 0;
}
```