Práctica 9: Comunicación entre procesos. Sockets

Esta práctica se ha realizado en una máquina virtual corriendo el sistema operativo Ubuntu 15.04 LTS.

Apartado 1: Llamadas al API para la gestión de direcciones

El uso del API BSD requiere la manipulación de direcciones de red, y traducción de estas entre las tres representaciones básicas: el nombre de dominio (e.j. “www.google.com”, la dirección IP (tanto versión 4, e.j. “173.194.34.212”, como 6 e.j. “2a00:1450:4003:801::1012” ) y binario (que finalmente se envían en la red como campo dirección origen en la cabecera del datagrama IP, por ejemplo).

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

● Host identificado por su dirección IPv4 en formato punto (ej. “147.96.1.9”)

● Host identificado por su dirección IPv6 en formato punto (ej. “fd00::a:0:0:0:1”)

● Host identificado por su FQDN (ej. “www.google.com”)

● Host identificado por su nombre en /etc/hosts (ej. “localhost”)

● Una dirección incorrecta en cualquiera de los casos anteriores

El programa se implementará usando la función getaddrinfo(3) para obtener la lista de posibles conexiones (struct sockaddr \*) . Cada dirección, protocolo y tipo de socket se imprimirá en su valor numérico usando la función getnameinfo(3) y flags NI\_NUMERICHOST.

**NOTA:** Para probar el comportamiento de las funciones y el servicio DNS, realizar este ejercicio en el host anfitrión.

|  |
| --- |
| **Ejemplo de Ejecución Ejercicio 1** |
| *# El protocolo 2 es AF\_INET. Los protocolos 1,2,3 corresponden a*  *# SOCK\_STREAM, SOCK\_DGRAM y SOCK\_RAW, respectivamente; ver socket.h*  **>./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  *# El protocolo 10 es AF\_INET6, ver socket.h*  **> ./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 noexite.ucm.es**  Error: Name or service not known |

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

**int main (int argc, char \*\*argv) {  
 struct addrinfo hints;  
 struct addrinfo \*res, \*ptr;**

**memset(&hints, 0, sizeof(struct addrinfo));**

**hints.ai\_flags = AI\_PASSIVE;  
 hints.ai\_family = AF\_UNSPEC;**

**int rc = getaddrinfo(argv[1], NULL, &hints, &res);**

**if(rc != 0) {  
 printf("Error getaddrinfo: %s\n", gai\_strerror(rc));  
 exit (-1);  
 }**

**for(ptr = res; ptr != NULL; ptr = ptr->ai\_next) {  
 char host[NI\_MAXHOST];  
 getnameinfo(ptr->ai\_addr, ptr->ai\_addrlen, host, NI\_MAXHOST, NULL, 0, NI\_NUMERICHOST);  
 printf("%s\t%i\t%i\n", host, ptr->ai\_family, ptr->ai\_socktype);  
 }**

**freeaddrinfo(res);**

**return 0;**

**}**

Apartado 2: Protocolo UDP - Servidor de hora

***Ejercicio 1.*** Usando como base el servidor estudiado en clase, escribir un servidor que use el protocolo UDP, de forma que

● La dirección y puerto son el primer y segundo argumento del programa. Las direcciones pueden expresarse en cualquier formato, esto es nombre de host, notación punto. Además el servidor debe funcionar con direcciones IPv4 e IPv6 .

**Nota:** Dado que el servidor puede funcionar con direcciones IPv4 e IPv6, hay que usar una estructura que permita acomodar cualquiera de ellas, por ejemplo en recvfrom(3). El API BSD define el tipo sockaddr\_storage para estas situaciones.

● El servidor recibirá un comando (codificado en un carácter), de forma que: 't' devuelva la hora, 'd' la fecha y 'q' termina el proceso servidor.

● En cada mensaje el servidor debe imprimir el nombre y puerto del cliente, usar la función getnameinfo(3).

**Nota:** Probar el funcionamiento del servidor con el comando netcat (nc)

**#include <sys/types.h>**

**#include <sys/socket.h>**

**#include <stdio.h>**

**#include <errno.h>**

**#include <arpa/inet.h>**

**#include <netinet/in.h>**

**#include <netdb.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/time.h>**

**#include <unistd.h>**

**#include <time.h>**

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

**if (argc < 3) {**

**printf("Use: ./ej1 host port\n");**

**return -1;**

**}**

**struct addrinfo hints;**

**hints.ai\_family = AF\_UNSPEC;**

**hints.ai\_socktype = SOCK\_DGRAM;**

**hints.ai\_flags = AI\_PASSIVE;**

**struct addrinfo \*list;**

**int result = getaddrinfo(argv[1], argv[2], &hints, &list);**

**if (result != 0) {**

**printf("Error while gettinf info: %s\n", gai\_strerror(result));**

**return -1;**

**}**

**int on = 1;**

**int off = 0;**

**int sd = socket(list->ai\_family, SOCK\_DGRAM, list->ai\_protocol);**

**if (sd == -1) {**

**perror("Unable to open the socket");**

**return -1;**

**}**

**setsockopt(sd, SOL\_SOCKET, SO\_REUSEADDR, (void \*)&on, sizeof(int));**

**setsockopt(sd, IPPROTO\_IPV6, IPV6\_V6ONLY, (void \*)&off, sizeof(int));**

**if (bind(sd, list->ai\_addr, list->ai\_addrlen)) {**

**perror("Unable to bind");**

**return -1;**

**}**

**freeaddrinfo(list);**

**char buf[2] = "";**

**struct sockaddr\_storage input\_addr;**

**socklen\_t input\_len = sizeof(input\_addr);**

**while (buf[0] != 'q') {**

**int readsize = recvfrom(sd, &buf, 2\*sizeof(char), 0, (struct sockaddr \*) &input\_addr, &input\_len);**

**if (readsize == -1) {**

**perror("Error ocurred while reciving");**

**close(sd);**

**return -1;**

**}**

**else if (readsize == 0) {**

**printf("Client has performed a shutdown\n");**

**buf[0] = 'q';**

**}**

**else {**

**char hostname[NI\_MAXHOST];**

**char port[NI\_MAXSERV];**

**if (getnameinfo((struct sockaddr \*)&input\_addr, input\_len, hostname, NI\_MAXHOST, port, NI\_MAXSERV, 0)) {**

**perror("Error getting info from name");**

**return -1;**

**}**

**printf("Recibí %i bytes de %s:%s\n", readsize, hostname, port);**

**if (buf[0] == 'd') {**

**time\_t t;**

**struct tm \*loctime;**

**char buffer[256];**

**t = time(NULL);**

**loctime = localtime(&t);**

**strftime(buffer,256,"%Y-%m-%d", loctime);**

**int writesize = sendto(sd, buffer, strlen(buffer), 0, (struct sockaddr \*)&input\_addr, input\_len);**

**if (writesize == -1) {**

**perror("Error while sending back to client");**

**return -1;**

**}**

**} else if(buf[0] == 't') {**

**time\_t t;**

**struct tm \*loctime;**

**char buffer[256];**

**t = time(NULL);**

**loctime = localtime(&t);**

**strftime(buffer,256,"%r", loctime);**

**int writesize = sendto(sd, buffer, strlen(buffer), 0, (struct sockaddr \*)&input\_addr, input\_len);**

**if (writesize == -1) {**

**perror("Error while sending back to client");**

**return -1;**

**}**

**} else if (buf[0] == 'q')**

**printf("Comando de finalización recibido, saliendo…\n");**

**else**

**printf("Comando no soportado %s", buf);**

**}**

**}**

**close(sd);**

**return 0;**

**}**

***Ejercicio 2****.* Escribir el cliente para el servidor de hora, similar al funcionamiento del comando nc. El cliente tendrá por parámetros la dirección y puerto del servidor y el comando, ej. ./hora 127.0.0.1 3000 t, para solicitar la hora.

**#include <sys/types.h>**

**#include <sys/socket.h>**

**#include <stdio.h>**

**#include <errno.h>**

**#include <arpa/inet.h>**

**#include <netinet/in.h>**

**#include <netdb.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/time.h>**

**#include <unistd.h>**

**#include <time.h>**

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

**if (argc < 4) {**

**printf("Use: ./ej2 host port command\n");**

**return -1;**

**}**

**struct addrinfo hints;**

**hints.ai\_family = AF\_UNSPEC;**

**hints.ai\_socktype = SOCK\_DGRAM;**

**hints.ai\_flags = AI\_PASSIVE;**

**struct addrinfo \*list;**

**int result = getaddrinfo(argv[1], argv[2], &hints, &list);**

**if (result != 0) {**

**printf("Error while gettinf info: %s\n", gai\_strerror(result));**

**return -1;**

**}**

**int on = 1;**

**int off = 0;**

**int sd = socket(list->ai\_family, SOCK\_DGRAM, list->ai\_protocol);**

**if (sd == -1) {**

**perror("Unable to open the socket");**

**return -1;**

**}**

**setsockopt(sd, SOL\_SOCKET, SO\_REUSEADDR, (void \*)&on, sizeof(int));**

**setsockopt(sd, IPPROTO\_IPV6, IPV6\_V6ONLY, (void \*)&off, sizeof(int));**

**if (-1 == sendto(sd, argv[3], strlen(argv[3]) + 1, 0, list->ai\_addr, list->ai\_addrlen)) {**

**perror("Error while sending to server");**

**return -1;**

**}**

**char buf[257];**

**ssize\_t readsize = recvfrom(sd, buf, 256, 0, list->ai\_addr, &(list->ai\_addrlen));**

**if (readsize == -1) {**

**perror("Error while recieving to server");**

**return -1;**

**}**

**buf[readsize] = '\0';**

**printf("%s\n", buf);**

**freeaddrinfo(list);**

**printf("Command %s sent to %s : %s\n", argv[3], argv[1], argv[2]);**

**close(sd);**

**return 0;**

**}**

***Ejercicio 3****.* Modificar el servidor para que además de poder recibir comandos por red, los pueda recibir directamente del terminal (leyendo dos caracteres, comando + „\n‟, de la entrada estándar). Multiplexar el uso de ambos flujos de datos usando la función select(2).

**#include <sys/types.h>**

**#include <sys/socket.h>**

**#include <stdio.h>**

**#include <errno.h>**

**#include <arpa/inet.h>**

**#include <netinet/in.h>**

**#include <netdb.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/time.h>**

**#include <unistd.h>**

**#include <time.h>**

**#include <sys/time.h>**

**#include <unistd.h>**

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

**if (argc < 3) {**

**printf("Use: ./ej1 host port\n");**

**return -1;**

**}**

**struct addrinfo hints;**

**hints.ai\_family = AF\_UNSPEC;**

**hints.ai\_socktype = SOCK\_DGRAM;**

**hints.ai\_protocol = 0;**

**hints.ai\_flags = AI\_PASSIVE;**

**hints.ai\_addr = NULL;**

**hints.ai\_canonname = NULL;**

**hints.ai\_next = NULL;**

**struct addrinfo \*list;**

**int result = getaddrinfo(argv[1], argv[2], &hints, &list);**

**if (result != 0) {**

**printf("Error while gettinf info: %s\n", gai\_strerror(result));**

**return -1;**

**}**

**int on = 1;**

**int off = 0;**

**int sd = socket(list->ai\_family, SOCK\_DGRAM, list->ai\_protocol);**

**if (sd == -1) {**

**perror("Unable to open the socket");**

**return -1;**

**}**

**setsockopt(sd, SOL\_SOCKET, SO\_REUSEADDR, (void \*)&on, sizeof(int));**

**setsockopt(sd, IPPROTO\_IPV6, IPV6\_V6ONLY, (void \*)&off, sizeof(int));**

**if (bind(sd, list->ai\_addr, list->ai\_addrlen)) {**

**perror("Unable to bind");**

**return -1;**

**}**

**freeaddrinfo(list);**

**char buf[2] = "";**

**struct sockaddr\_storage input\_addr;**

**socklen\_t input\_len = sizeof(input\_addr);**

**fd\_set set;**

**int readsize;**

**while (buf[0] != 'q') {**

**FD\_ZERO(&set);**

**FD\_SET(0, &set);**

**FD\_SET(sd, &set);**

**if (-1 == select(sd + 1, &set, NULL, NULL, NULL)) {**

**perror("Error while waiting for data");**

**return -1;**

**}**

**if (FD\_ISSET(sd, &set))**

**readsize = recvfrom(sd, &buf, 2\*sizeof(char), 0, (struct sockaddr\*) &input\_addr, &input\_len);**

**else if (FD\_ISSET(0, &set))**

**readsize = read(0, &buf, 2\*sizeof(char));**

**if (readsize == -1) {**

**perror("Error ocurred while reciving");**

**close(sd);**

**return -1;**

**}**

**else if (readsize == 0) {**

**printf("Client has performed a shutdown\n");**

**buf[0] = 'q';**

**}**

**else {**

**if (FD\_ISSET(sd, &set)) {**

**char hostname[NI\_MAXHOST] = "";**

**char port[NI\_MAXSERV] = "";**

**if (getnameinfo((struct sockaddr\*) &input\_addr, input\_len, hostname, NI\_MAXHOST, port, NI\_MAXSERV, 0)) {**

**perror("Error getting info from name");**

**return -1;**

**}**

**printf("Recibí %i bytes de %s:%s\n", readsize, hostname, port);**

**}**

**if (buf[0] == 'd') {**

**time\_t t;**

**struct tm \*loctime;**

**char buffer[256];**

**t = time(NULL);**

**loctime = localtime(&t);**

**strftime(buffer,256,"%Y-%m-%d", loctime);**

**if (FD\_ISSET(0, &set))**

**printf("%s\n", buffer);**

**else if (FD\_ISSET(sd, &set)) {**

**int writesize = sendto(sd, buffer, strlen(buffer), 0, (struct sockaddr \*)&input\_addr, input\_len);**

**if (writesize == -1) {**

**perror("Error while sending back to client");**

**return -1;**

**}**

**}**

**} else if (buf[0] == 't') {**

**time\_t t;**

**struct tm \*loctime;**

**char buffer[256];**

**t = time(NULL);**

**loctime = localtime(&t);**

**strftime(buffer,256,"%r", loctime);**

**if (FD\_ISSET(0, &set))**

**printf("%s\n", buffer);**

**else if (FD\_ISSET(sd, &set)) {**

**int writesize = sendto(sd, buffer, strlen(buffer), 0, (struct sockaddr \*)&input\_addr, input\_len);**

**if (writesize == -1) {**

**perror("Error while sending back to client");**

**return -1;**

**}**

**}**

**}**

**else if (buf[0] == 'q')**

**printf("Comando de finalización recibido, saliendo…\n");**

**else**

**printf("Comando no soportado %s", buf);**

**}**

**}**

**close(sd);**

**return 0;**

**}**

***Ejercicio 4*** *(Opcional)***.** Convertir el servidor UDP en *multi-thread*. Una vez asociado el socket a la dirección después de la llamada a bind(2), realizar la llamada recvfrom en threads diferentes de forma que cada uno atenderá una conexión de un cliente distinto.

Apartado 3: Protocolo TCP - Servidor echo

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 1.*** Servidor echo que escuche en un puerto y dirección dada IPv4 - IPv6. Utilizando sockets sobre TCP, crear un servidor de eco que escuche por conexiones entrantes en un puerto dado. 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 el comando netcat (*nc*) como cliente. Comprobar qué sucede si varios clientes intentan conectar al mismo tiempo.

|  |
| --- |
| **Ejemplo ejercicio 1** |
| **Servidor**:  $ ./echo\_tcp\_server1 2222  **Cliente**:  $ nc ::1 2222  Hola  Hola  Qué tal  Qué tal  ^C  $ **Servidor**:  Conexión desde ::1 53456  Conexión terminada |

**#include <sys/types.h>**

**#include <sys/socket.h>**

**#include <stdio.h>**

**#include <errno.h>**

**#include <arpa/inet.h>**

**#include <netinet/in.h>**

**#include <netdb.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/time.h>**

**#include <unistd.h>**

**#include <time.h>**

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

**if (argc < 2) {**

**printf("Use: ./ej1 port\n");**

**return -1;**

**}**

**struct addrinfo hints;**

**hints.ai\_family = AF\_UNSPEC;**

**hints.ai\_socktype = SOCK\_STREAM;**

**hints.ai\_protocol = 0;**

**hints.ai\_flags = AI\_PASSIVE;**

**hints.ai\_addr = NULL;**

**hints.ai\_canonname = NULL;**

**hints.ai\_next = NULL;**

**struct addrinfo \*list;**

**int result = getaddrinfo("::", argv[1], &hints, &list);**

**if (result != 0) {**

**printf("Error while gettinf info: %s\n", gai\_strerror(result));**

**return -1;**

**}**

**int on = 1;**

**int off = 0;**

**int sd = socket(list->ai\_family, SOCK\_STREAM, list->ai\_protocol);**

**if (sd == -1) {**

**perror("Unable to open the socket");**

**return -1;**

**}**

**setsockopt(sd, SOL\_SOCKET, SO\_REUSEADDR, (void \*)&on, sizeof(int));**

**setsockopt(sd, IPPROTO\_IPV6, IPV6\_V6ONLY, (void \*)&off, sizeof(int));**

**if (bind(sd, list->ai\_addr, list->ai\_addrlen)) {**

**perror("Unable to bind");**

**close(sd);**

**return -1;**

**}**

**freeaddrinfo(list);**

**char buf[257] = "";**

**struct sockaddr\_storage input\_addr;**

**socklen\_t input\_len = sizeof(input\_addr);**

**if (-1 == listen(sd, 10)) {**

**perror("Error while listening");**

**return -1;**

**}**

**while (1) {**

**int asock = accept(sd, (struct sockaddr\*)&input\_addr, &input\_len);**

**if (asock == -1) {**

**perror("Error while accepting connection");**

**return -1;**

**}**

**char hostname[NI\_MAXHOST];**

**char port[NI\_MAXSERV];**

**if (getnameinfo((struct sockaddr\*)&input\_addr, input\_len, hostname, NI\_MAXHOST, port, NI\_MAXSERV, 0)) {**

**perror("Error getting info from name");**

**return -1;**

**}**

**printf("Connection from %s : %s\n", hostname, port);**

**ssize\_t readsize;**

**do {**

**readsize = recv(asock, buf, 256\*sizeof(char), 0);**

**if (readsize == -1) {**

**perror("Error ocurred while reciving");**

**close(sd);**

**return -1;**

**}**

**else if (readsize == 0) {**

**printf("Client has performed a shutdown\n");**

**}**

**else {**

**buf[readsize] = '\0';**

**ssize\_t writesize = send(asock, buf, (readsize + 1)\*sizeof(char), 0);**

**if (writesize == -1) {**

**perror("Error while sending back to client");**

**return -1;**

**}**

**}**

**} while (readsize > 0);**

**close(asock);**

**}**

**close(sd);**

**return 0;**

**}**

***Ejercicio 2****.* Escribir el cliente para conectarse con el servidor del ejercicio 1. El cliente debe tomar la dirección y el puerto del servidor desde la línea de órdenes (pasados como parámetros) 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 la letra 'Q' como único carácter de una línea, el cliente cerrará la conexión con el servidor.

|  |
| --- |
| **Ejemplo ejercicio 2** |
| **Servidor**:  $ ./echo\_tcp\_server1 2222  **Cliente**:  $ ./echo tcp\_server ::1 2222  Hola  Hola  Q  $  **Servidor**:  Recibida conexión desde ::1 53445  Conexión terminada  $ |

**#include <sys/types.h>**

**#include <sys/socket.h>**

**#include <stdio.h>**

**#include <errno.h>**

**#include <arpa/inet.h>**

**#include <netinet/in.h>**

**#include <netdb.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/time.h>**

**#include <unistd.h>**

**#include <time.h>**

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

**if (argc < 3) {**

**printf("Use: ./ej2 host port\n");**

**return -1;**

**}**

**struct addrinfo hints;**

**hints.ai\_family = AF\_UNSPEC;**

**hints.ai\_socktype = SOCK\_STREAM;**

**hints.ai\_protocol = 0;**

**hints.ai\_flags = AI\_PASSIVE;**

**hints.ai\_addr = NULL;**

**hints.ai\_canonname = NULL;**

**hints.ai\_next = NULL;**

**struct addrinfo \*list;**

**int result = getaddrinfo(argv[1], argv[2], &hints, &list);**

**if (result != 0) {**

**printf("Error while gettinf info: %s\n", gai\_strerror(result));**

**return -1;**

**}**

**int on = 1;**

**int off = 0;**

**int sd = socket(list->ai\_family, SOCK\_STREAM, list->ai\_protocol);**

**if (sd == -1) {**

**perror("Unable to open the socket");**

**return -1;**

**}**

**setsockopt(sd, SOL\_SOCKET, SO\_REUSEADDR, (void \*)&on, sizeof(int));**

**setsockopt(sd, IPPROTO\_IPV6, IPV6\_V6ONLY, (void \*)&off, sizeof(int));**

**if (connect(sd, list->ai\_addr, list->ai\_addrlen)) {**

**perror("Unable to connect");**

**close(sd);**

**return -1;**

**}**

**freeaddrinfo(list);**

**char buf[257] = "";**

**ssize\_t readsize, writesize;**

**while (strcmp("Q\n", buf)) {**

**readsize = read(0, buf, 256\*sizeof(char));**

**if (readsize == -1) {**

**perror("Error ocurred while reading");**

**close(sd);**

**return -1;**

**}**

**buf[readsize] = '\0';**

**writesize = send(sd, buf, (readsize + 1)\*sizeof(char), 0);**

**if (writesize == -1) {**

**perror("Error ocurred while sending");**

**close(sd);**

**return -1;**

**}**

**readsize = recv(sd, buf, 256\*sizeof(char), 0);**

**if (readsize == -1) {**

**perror("Error ocurred while reading");**

**close(sd);**

**return -1;**

**}**

**buf[readsize] = '\0';**

**printf("%s", buf);**

**}**

**close(sd);**

**return 0;**

**}**

***Ejercicio 3****.* Tratar cada petición en un proceso diferente con *fork*(2). Se debe modificar el código del servidor para que acepte varias conexiones simultáneas. Cada conexión se gestionará en un proceso hijo. El proceso padre debe cerrar el socket devuelto por la llamada *accept*(2).

**#include <sys/types.h>**

**#include <sys/socket.h>**

**#include <stdio.h>**

**#include <errno.h>**

**#include <arpa/inet.h>**

**#include <netinet/in.h>**

**#include <netdb.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/time.h>**

**#include <unistd.h>**

**#include <time.h>**

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

**if (argc < 2) {**

**printf("Use: ./ej1 port\n");**

**return -1;**

**}**

**struct addrinfo hints;**

**hints.ai\_family = AF\_UNSPEC;**

**hints.ai\_socktype = SOCK\_STREAM;**

**hints.ai\_protocol = 0;**

**hints.ai\_flags = AI\_PASSIVE;**

**hints.ai\_addr = NULL;**

**hints.ai\_canonname = NULL;**

**hints.ai\_next = NULL;**

**struct addrinfo \*list;**

**int result = getaddrinfo("::", argv[1], &hints, &list);**

**if (result != 0) {**

**printf("Error while gettinf info: %s\n", gai\_strerror(result));**

**return -1;**

**}**

**int on = 1;**

**int off = 0;**

**int sd = socket(list->ai\_family, SOCK\_STREAM, list->ai\_protocol);**

**if (sd == -1) {**

**perror("Unable to open the socket");**

**return -1;**

**}**

**setsockopt(sd, SOL\_SOCKET, SO\_REUSEADDR, (void \*)&on, sizeof(int));**

**setsockopt(sd, IPPROTO\_IPV6, IPV6\_V6ONLY, (void \*)&off, sizeof(int));**

**if (bind(sd, list->ai\_addr, list->ai\_addrlen)) {**

**perror("Unable to bind");**

**close(sd);**

**return -1;**

**}**

**freeaddrinfo(list);**

**char buf[257] = "";**

**struct sockaddr\_storage input\_addr;**

**socklen\_t input\_len = sizeof(input\_addr);**

**if (-1 == listen(sd, 10)) {**

**perror("Error while listening");**

**return -1;**

**}**

**pid\_t pid = getpid();**

**while (pid) {**

**int asock = accept(sd, (struct sockaddr\*)&input\_addr, &input\_len);**

**if (asock == -1) {**

**perror("Error while accepting connection");**

**exit(-1);**

**}**

**char hostname[NI\_MAXHOST];**

**char port[NI\_MAXSERV];**

**if (getnameinfo((struct sockaddr\*)&input\_addr, input\_len, hostname, NI\_MAXHOST, port, NI\_MAXSERV, 0)) {**

**perror("Error getting info from name");**

**exit(-1);**

**}**

**printf("Connection from %s : %s\n", hostname, port);**

**pid = fork();**

**if (pid == -1) {**

**perror("Unable to fork");**

**exit(-1);**

**}**

**if (pid == 0) {**

**ssize\_t readsize;**

**do {**

**readsize = recv(asock, buf, 256\*sizeof(char), 0);**

**if (readsize == -1) {**

**perror("Error ocurred while reciving");**

**close(sd);**

**exit(-1);**

**}**

**else if (readsize == 0) {**

**printf("Client has performed a shutdown\n");**

**}**

**else {**

**buf[readsize] = '\0';**

**ssize\_t writesize = send(asock, buf, (readsize + 1)\*sizeof(char), 0);**

**if (writesize == -1) {**

**perror("Error while sending back to client");**

**exit(-1);**

**}**

**}**

**} while (readsize > 0);**

**}**

**close(asock);**

**}**

**close(sd);**

**exit(0);**

**return 0;**

**}**

***Ejercicio 4****.* Añadir la lógica necesaria para que el servidor sincronice la finalización de los procesos hijo de forma que no quede ningún proceso en estado zombie.