

Parte I: Programación Distribuida con Sockets

Contenido

En la presente practica se continuara con la comunicación entre procesos, la diferencia con los métodos anteriores es que tendremos la posibilidad de comunicar procesos en un entorno de red, esto se logra por medio de la programación con socket.

Objetivo Especifico

- a) Desarrollar aplicaciones distribuidas cliente-servidor utilizando socket.
- b) Utilizar las funciones del API Posix orientadas a la implementacion de aplicaciones distribuidas utilizando sockets.

Introduccion Teorica

Socket

Un socket se define como un descriptor de un canal de comunicación por medio del cual un proceso puede enviar o recibir información. Siempre es necesario crear un socket tanto en el cliente como en el servidor. Se basa en los mecanismos de E/S de archivos en UNIX. Aparecen por primera vez en la distribución 4.1c de Berkeley Software Distribution.

Para establecer las comunicaciones entre procesos utilizando sockets, es necesario definir primero la familia o dominio del mismo. Este especifica el formato de las direcciones que se podrán dar a los sockets y los diferentes protocolos soportados por la comunicación entre sockets. Todo dominio posee su propia estructura especialmente creada conocida como estructura de dirección socket.

La mayoría de las funciones relacionadas con sockets requieren un puntero a una estructura de dirección socket como argumento. Cada conjunto de protocolos (dominio) define su propia estructura de dirección socket. Los nombres de dichas estructuras inician con las letras `sockaddr_` y finalizan con un sufijo único para cada conjunto de protocolos. En el caso de UNIX/LINUX, estas estructuras pueden ser pasadas tanto de un proceso al kernel (núcleo) como del kernel a un proceso.

Las estructuras de dirección socket son pasadas siempre por referencia cuando se pasan como argumento a cualquiera de las funciones relacionadas con sockets. Sin embargo, la estructura puede variar dependiendo del conjunto de protocolos usados ya que por cada dominio existente, se tiene una estructura de dirección socket específica para ese dominio. Con esto se tendría el inconveniente de necesitar un conjunto de funciones dedicadas a manejar solamente un tipo de estructura de dirección socket, lo que resultaría impráctico. Es por ello lo que dichas funciones deben ser capaces de gestionar un formato estándar en lugar de adaptarse a un tipo de estructura de dirección socket específica.

Material y Equipo

- a) Sistema operativo Linux
- b) Compilador gcc
- c) Guía de laboratorio

Procedimiento

Aplicación echo distribuida con sockets

La aplicación cliente-servidor siguiente utiliza sockets para la transferencia de información. Desde el programa cliente se envía una cadena al programa servidor, cuando el servidor recibe la cadena la reenvía al cliente para que este la imprima en pantalla a manera de "eco".

A este programa cliente solo se le debe proporcionar la dirección IP del servidor en notación de puntos y posteriormente la(s) cadena(s) respectiva(s) para realizar el eco.

Digitar el código del programa cliente y del programa servidor, compilarlos y ejecutar la aplicación.

Código archivo include (archivo eco.h)

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

#define PORT_TCP_SERV 6500
#define PORT_TCP_CLI 6501
#define MAXLINEA 100

#define error(mesg) { perror(mesg); exit(1); }
```

Código del programa cliente (archivo ecoudpcli.c)

```
#include "eco.h"

main(int argc, char *argv[])
{
    int sockfd, n;
    struct sockaddr_in dir_cli, dir_serv;
    char lineaenv[MAXLINEA], linearec[MAXLINEA];

    bzero((char*) &dir_serv, sizeof(dir_serv));

    dir_serv.sin_family = AF_INET;
    dir_serv.sin_addr.s_addr = inet_addr(argv[1]);
    dir_serv.sin_port = htons(PORT_TCP_SERV);

    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        error("cliente: no se puede abrir socket");

    if(connect(sockfd, (struct sockaddr *)&dir_serv, sizeof(dir_serv)) < 0)
        error("cliente: error funcion connect");

    while((fgets(lineaenv, MAXLINEA, stdin) != NULL) &&
           (strlen(lineaenv) != 1)){
        n = strlen(lineaenv);
```

3 Sistemas Operativos, Guía 6

```
        if(write(sockfd, lineaenv, strlen(lineaenv)) == -1)
            error("cliente: error funcion write");

        n = read(sockfd, linearec, sizeof(linearec));

        if(n < 0)
            error("cliente: error funcion read");

        linearec[n] = 0;
        fputs(linearec, stdout);
    }

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

Código del programa servidor (archivo ecoudpsrv.c)

```
#include "eco.h"

main(int argc, char *argv[])
{
    int sockfd, n, newsockfd, len_cli;
    struct sockaddr_in dir_cli, dir_serv;
    char linea[MAXLINEA];

    bzero((char*) &dir_serv, sizeof(dir_serv));

    dir_serv.sin_family = AF_INET;
    dir_serv.sin_addr.s_addr = htonl(INADDR_ANY);
    dir_serv.sin_port = htons(PORT_TCP_SERV);

    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        error("servidor: no se puede crear socket");

    if(bind(sockfd, (struct sockaddr *) &dir_serv, sizeof(dir_serv)) < 0)
        error("servidor: no se puede asociar la dir local");

    listen(sockfd, 5);

    for( ; ; ){

        len_cli = sizeof(dir_cli);
        newsockfd = accept(sockfd, (struct sockaddr *)&dir_cli, &len_cli);

        if(newsockfd < 0) error("servidor: error funcion accept");

        while((n = read(newsockfd, linea, MAXLINEA)) != 0)
        {
            if(n < 0) error("servidor: error funcion read");

            if(n != write(newsockfd, linea, n)) error("servidor: error funcion
                                                                    write");
        }
        close(newsockfd);
    }
}
```

```
usuario@usuario-desktop: ~/walter.sanchez/sistemas.operativos

usuario@usuario-desktop:~/walter.sanchez/sistemas.operativos$ ./ecosrv&
[1] 3410
usuario@usuario-desktop:~/walter.sanchez/sistemas.operativos$ ./ecocli 127.0.0.1
Esta es una prueba Sistemas Operativos
Esta es una prueba Sistemas Operativos
Que facil es la programacion con sockets
Que facil es la programacion con sockets
```

Desde el programa cliente se envía una cadena al programa servidor, cuando el servidor recibe la cadena la reenvía al cliente para que este la imprima en pantalla a manera de “eco”.

Aplicación daytime distribuida con sockets

La aplicación cliente-servidor siguiente utiliza sockets para la transferencia de información. El programa cliente solicita al servidor el servicio daytime, el servidor responde al cliente con la fecha y hora actual del servidor.

A este programa cliente solo se le debe proporcionar la dirección IP del servidor en notación de puntos.

Digitar el código del programa cliente y del programa servidor, compilarlos y ejecutar la aplicación.

Código archivo include (archivo daytime.h)

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>

#define PORT_UDP_SERV 3480
#define PORT_UDP_CLI 3481
#define MAXLINEA 100

#define error(mesg){ perror(mesg); exit(1); }
```

Código del programa cliente (archivo daytimeudcli.c)

```
#include "daytime.h"

main(int argc, char *argv[])
{
    int sockfd, n;
    struct sockaddr_in dir_cli, dir_serv;
    char buffer[MAXLINEA] = "0";

    bzero((char*) &dir_cli, sizeof(dir_cli));

    if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        error(" cliente: no se puede abrir socket");

    dir_cli.sin_family = AF_INET;
    dir_cli.sin_addr.s_addr = INADDR_ANY;
```

5 Sistemas Operativos, Guía 6

```
dir_cli.sin_port = htons(PORT_UDP_CLI);

if(bind(sockfd,(struct sockaddr *) &dir_cli, sizeof(dir_cli)) == -1)
    error(" cliente: no se puede asociar la dir local");

bzero((char *) &dir_serv, sizeof(dir_serv));

dir_serv.sin_family = AF_INET;
dir_serv.sin_addr.s_addr = inet_addr(argv[1]);
dir_serv.sin_port = htons(PORT_UDP_SERV);

if(sendto(sockfd, buffer, MAXLINEA, 0, (struct sockaddr *) &dir_serv, sizeof(dir_serv)) == -1)
    error("cli: error en el sendto");

n = recvfrom(sockfd, buffer, MAXLINEA, 0, (struct sockaddr *) 0, (int *) 0);

if(n < 0)
    error("cli: error funcion recvfrom");

buffer[n] = 0;
fputs(buffer, stdout);

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

Código del programa servidor (archivo daytimeudpsrv.c)

```
#include "daytime.h"

main(int argc, char *argv[])
{
    int sockfd, n, len_cli;
    struct sockaddr_in dir_cli, dir_serv;
    char buffer[MAXLINEA];
    time_t ticks;

    bzero((char*) &dir_serv, sizeof(dir_serv));

    dir_serv.sin_family = AF_INET;
    dir_serv.sin_addr.s_addr = INADDR_ANY;
    dir_serv.sin_port = htons(PORT_UDP_SERV);

    if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        error("servidor: no se puede crear socket");

    if(bind(sockfd, (struct sockaddr *) &dir_serv, sizeof(dir_serv)) < 0)
        error("servidor: no se puede asociar la dir local");

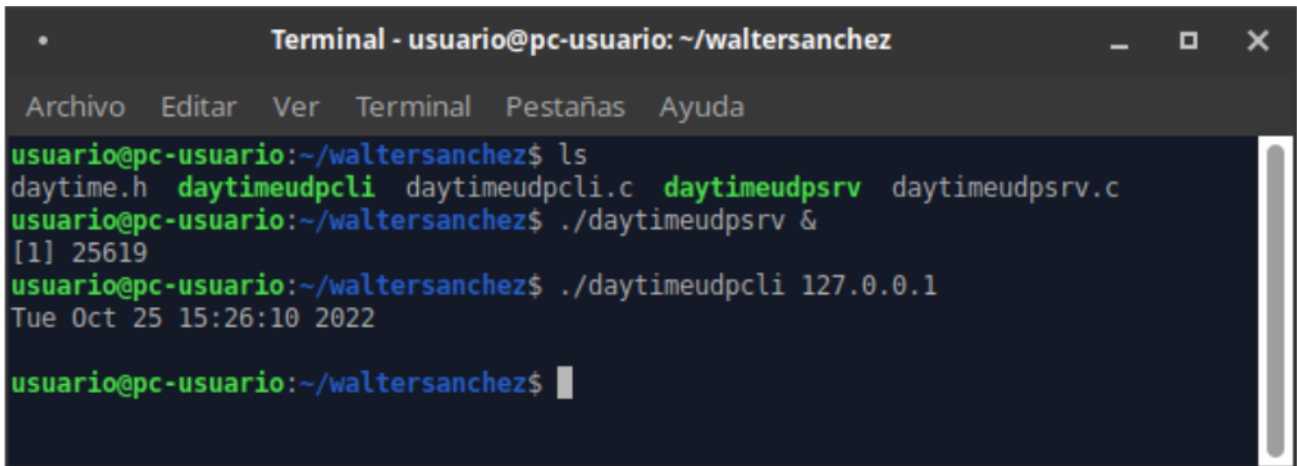
    for(;;){

        len_cli = sizeof(dir_cli);

        n = recvfrom(sockfd, buffer, MAXLINEA, 0, (struct sockaddr *) &dir_cli, &len_cli);

        ticks = time(NULL);
```

```
    snprintf(buffer, sizeof(buffer), "%s\r\n", ctime(&ticks));  
    if(sendto(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *) &dir_cli, sizeof(dir_cli)) <  
0)        error("servidor: error funcion sendto");  
    }  
}
```



```
Terminal - usuario@pc-usuario: ~/waltersanchez  
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda  
usuario@pc-usuario:~/waltersanchez$ ls  
daytime.h  daytimeudpcli  daytimeudpcli.c  daytimeudpsrv  daytimeudpsrv.c  
usuario@pc-usuario:~/waltersanchez$ ./daytimeudpsrv &  
[1] 25619  
usuario@pc-usuario:~/waltersanchez$ ./daytimeudpcli 127.0.0.1  
Tue Oct 25 15:26:10 2022  
usuario@pc-usuario:~/waltersanchez$
```

Facultad: Ingeniería

Escuela: Computación

Asignatura: Sistemas Operativos

Parte II: Gestión de archivos y directorios

Contenido

En esta practica se abordan las funciones mas ampliamente utilizadas del API POSIX para la gestión de archivos y directorios.

Objetivo Especifico

- a) Comprender las funciones mas ampliamente utilizadas del API POSIX para la gestión de archivos y directorios.

Introduccion Teorica

Desde el punto de vista de los usuarios y las aplicaciones, los ficheros y directorios son los elementos centrales del sistema, debido a que en estos se guarda tanto la información como los programas que gestionan esa información.

Un fichero es un tipo abstracto de datos, por lo tanto, para que este completamente definido, es necesario definir las operaciones que pueden ejecutarse sobre un objeto fichero. En general, los sistemas operativos proporcionan operaciones para crear un fichero, almacenar información en el y recuperar dicha información.

En función de lo anterior, el API POSIX de los sistemas tipo UNIX como GNU/Linux provee una interfaz para la programación y gestión de archivos y directorios, esto se resume en un conjunto de funciones que realizan tareas comunes de manejo de archivos.

Material y Equipo

- a) Sistema operativo Linux
- b) Compilador gcc
- c) Guía de laboratorio

Procedimiento

Ejemplo 1. Gestión de archivos. El programa copiar.c hace una copia de un fichero a partir de otro ya existente.

Programa copiar.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

#define TAMANYO_ALM 1024
main (int argc, char **argv)
{
    int    fd_ent, fd_sal;
    char   almacen[TAMANYO_ALM];
    int    n_read;

    if (argc != 3) {
        fprintf (stderr, "Uso: copiar origen destino \n");
        exit(-1);
    }

    fd_ent = open (argv[1], O_RDONLY);
    if (fd_ent < 0)
    {
        perror ("open");
        exit (-1);
    }

    fd_sal = creat (argv[2], 0644);
    if (fd_sal < 0)
    {
        perror ("creat");
        exit (-1);
    }

    if (flock (fd_ent, LOCK_SH) == -1) {
        perror ("flock origen"); close(fd_ent); close(fd_sal);
        exit (-1);
    }

    if (flock (fd_sal, LOCK_EX) == -1) {
        perror ("flock destino");
        flock (fd_ent, LOCK_UN);
        close(fd_ent); close(fd_sal);
        exit (-1);
    }

    while ((n_read = read (fd_ent, almacen, TAMANYO_ALM)) > 0 )
    {
        if (write (fd_sal, almacen, n_read) < n_read)
        {
            perror ("write");
            close (fd_ent);
            close (fd_sal);
        }
    }
}
```



```

        exit (-1);
    }
}
if (n_read < 0)
{
    perror ("read");
    close (fd_ent);
    close (fd_sal);
    exit (-1);
}

flock (fd_ent, LOCK_UN);
flock (fd_sal, LOCK_UN);

close (fd_ent);
close (fd_sal);
exit (0);
}

```

```

Terminal - usuario@pc-usuario: ~/waltersanchez
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
usuario@pc-usuario:~/waltersanchez$ ls
copiar.c  copiar.c  muestra.txt
usuario@pc-usuario:~/waltersanchez$ cat muestra.txt
Un socket se define como un descriptor de un canal de comunicación por medio del cual un proceso puede enviar o recibir información. Siempre es necesario crear un socket tanto en el cliente como en el servidor. Se basa en los mecanismos de E/S de archivos en UNIX. Aparecen por primera vez en la distribución 4.1c de Berkeley Software Distribution.
usuario@pc-usuario:~/waltersanchez$ ./copiar muestra.txt copiamuestra.txt
usuario@pc-usuario:~/waltersanchez$ cat copiamuestra.txt
Un socket se define como un descriptor de un canal de comunicación por medio del cual un proceso puede enviar o recibir información. Siempre es necesario crear un socket tanto en el cliente como en el servidor. Se basa en los mecanismos de E/S de archivos en UNIX. Aparecen por primera vez en la distribución 4.1c de Berkeley Software Distribution.
usuario@pc-usuario:~/waltersanchez$

```

Ejemplo 2. Gestión de directorios, ls2.c implementa la funcionamiento básica del comando ls, accediendo a la lista de accesos del directorio por medio de la función readdir.

Archivo ls2.c

```

#include <sys/types.h>
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <error.h>

#define TAMANYO_ALM 1024

void main (int argc, char **argv)
{
    DIR *dirp;
    struct dirent *dp;
    char almacen[TAMANYO_ALM];

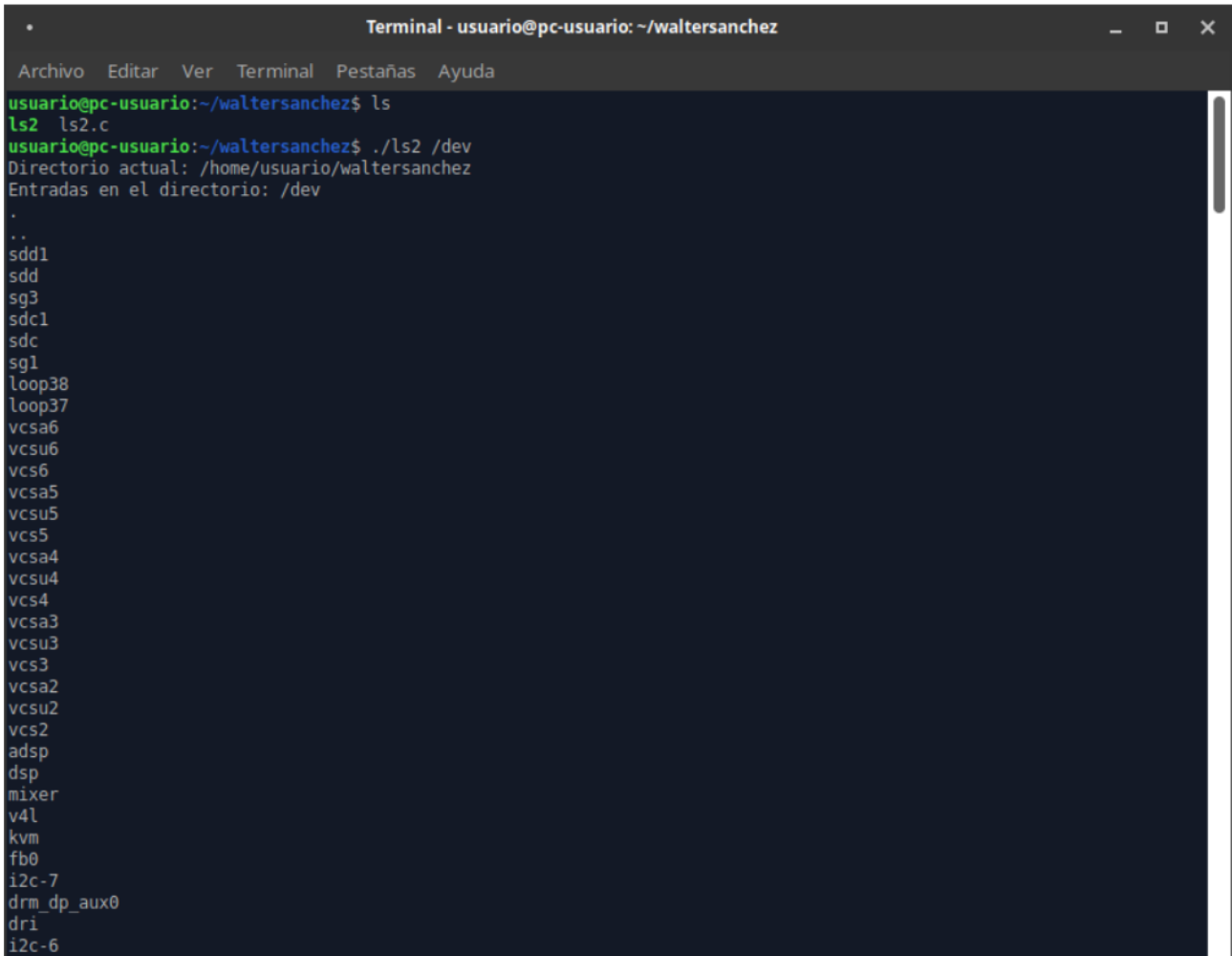
    if (argc < 2) {
        fprintf (stderr, "Uso: ls2 directorio \n");
        exit(1);
    }

    getcwd (almacen, TAMANYO_ALM);
    printf ("Directorio actual: %s \n", almacen);

    dirp = opendir (argv[1]);
    if (dirp == NULL)
    {
        fprintf (stderr, "No se pudo abrir el directorio: %s \n", argv[1]);
        exit (1);
    }
}

```

```
else
{
    printf ("Entradas en el directorio: %s \n", argv[1]);
    while ((dp = readdir(dirp)) != NULL)
        printf ("%s\n", dp->d_name);
    closedir (dirp);
}
exit (0);
}
```



A terminal window titled "Terminal - usuario@pc-usuario: ~/waltersanchez" is shown. The terminal has a menu bar with "Archivo", "Editar", "Ver", "Terminal", "Pestañas", and "Ayuda". The user runs the command `ls2 ls2.c`, and the program outputs the following directory listing for `/dev`:

```
usuario@pc-usuario:~/waltersanchez$ ls
ls2  ls2.c
usuario@pc-usuario:~/waltersanchez$ ./ls2 /dev
Directorio actual: /home/usuario/waltersanchez
Entradas en el directorio: /dev
.
..
sdd1
sdd
sg3
sdc1
sdc
sg1
loop38
loop37
vcsa6
vcsu6
vcs6
vcsa5
vcsu5
vcs5
vcsa4
vcsu4
vcs4
vcsa3
vcsu3
vcs3
vcsa2
vcsu2
vcs2
adsp
dsp
mixer
v4l
kvm
fb0
i2c-7
drm_dp_aux0
dri
i2c-6
```