Sockets tipo UNIX

MSc. Ivan A. Escobar

Creación socket: *socket()*

int socket(familia, tipo, protocolo)

Regresa un valor entero
Parecido descriptor de archivos: descriptor socket *sockfd*

int familia
familia de direcciones:
AF_UNIX protocolos internos UNIX
AF_INET protocolo de internet
AF_NS protocolo de la XEROX NS
AF_IMPLIK IMP link layer (interface Multiple Layer)

int tipo
tipo de socket:

SOCKET_STREAM	socket tipo stream
SOCKET_DGRAM	socket tipo datagrama
SOCKET_RAW	socket tipo raw
SOCKET_SEQPAKET	socket paquete secuencial
SOCKET_RDM	realible delivered message socket (no implementado)

int protocolo
tipo de protocolo a utilizar
O: el sistema elige su protocolo.

Protocolos para diversos tipos y familias

	AF_UNIX	*AF_INET*	*AF_NS*
SOCK_STREAM	yes	TCP	SPP
SOCK_DGRAM	yes	UDP	IDP
SOCK_RAW		IP	yes
SOCK_SEQPACKET			SPP

yes: validos pero sin handy acronyms.
vacías: no implementadas

Ejemplo:

```
        :
int sock;
        :
sock = socket(AF_INET, SOCK_STREAM, 0);
if  (sock < 0) {
    fprintf(stderr, “Error en la creacion del socket \n”);
    exit(1);
}
```

Familias direcciones sockets

1. Unix domain addres

```
struct sockaddr_un {
   short     sun_family;
   char      sun_path[108];
}
```

2. Internet domain addres

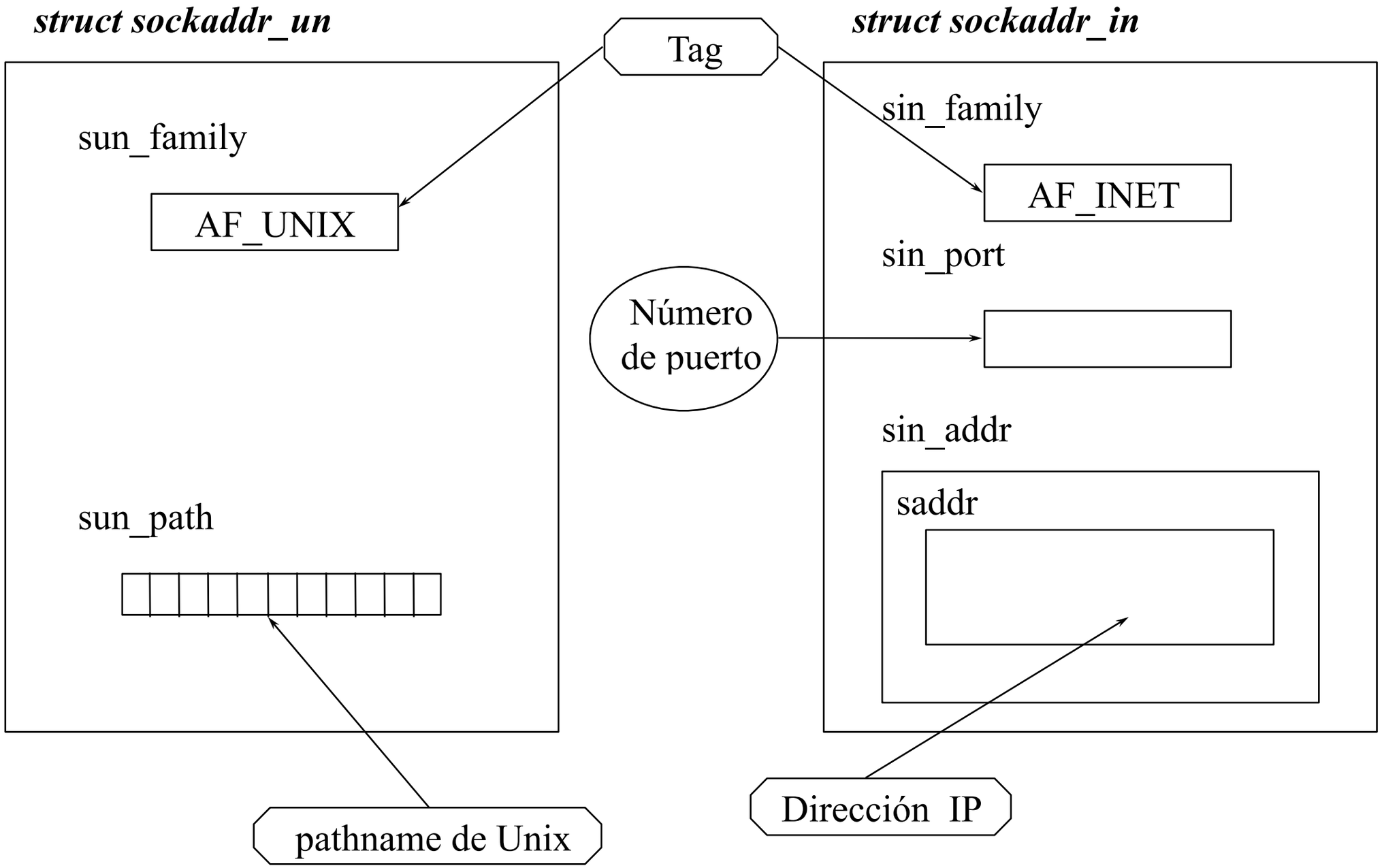
```
struct sockaddr_in {
   short              sin_family;
   u_short            sin_port;
   struct in_addr     sin_addr;
   char               sin_zero;
}

struct in_addr {
   u_long    s_addr;
}
```

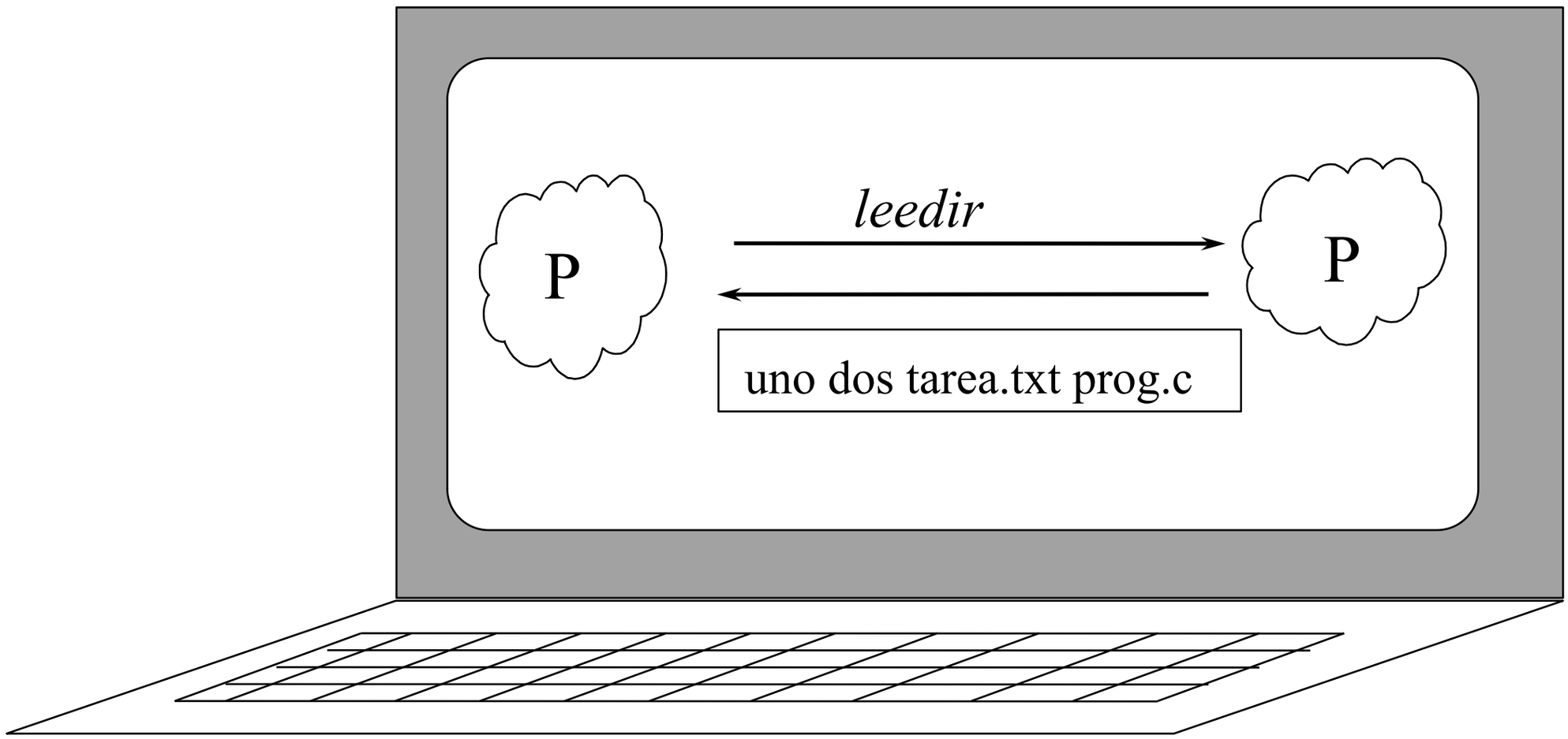
3. Generic socket addres

```
struct sockaddr{
   u_short    sa_family;
   char       sa_data[14];
}
```

Comparando familias



Ejemplo uso sockets familia Unix:



Los dos procesos en la misma máquina

Lectura local de un directorio
archivo: lee_dir.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <errno.h>


lee_dir(dir)
    char *dir;
{
    DIR *dirp;
    struct dirent *direntp;


    /* abriendo el directorio */
    dirp = opendir(dir);
    if (dirp == NULL)
        return((int)NULL);


    /* copiando los nombres archivos en el buffer dir */
     dir[0] = '\0'
     while ( (direntp= readdir(dirp)) != NULL)
         sprintf(dir, "%s%s\n", dir, direntp->d_name);


    /* regresando el resultado */
       closedir(dirp);
       return((int)dir);
}
```

Versión orientada conexión con sockets tipo Unix

Código del Servidor

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

#define  DIRSIZE   8192              /* tamaño máximo parámetro entrada/salida*/

main()
{
        char  dir[DIRSIZE];          /* parámetro entrada y salida */
        int     sd,   sd_actual;     /* descriptores de sockets */
        struct sockaddr_un   sin;    /*  direcciones socket */
```

```
/* obtención de un socket tipo AF_UNIX   */
        if ( (sd = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
                perror("socket");
                exit(1);
        }

/* llenando los campos de la estructura de dirección unix   */
        strcpy(sin.sun_path, "./socket");
        sin.sun_family = AF_UNIX;

/* asociando el socket al archivo ./socket */
        if ( bind(sd, (struc sockaddr *)&sin, sizeof(sin)) == -1 ) {
                perror("bind");
                exit(1);
        }

/* ponerse a escuchar a través del socket */
        if ( listen(sd, 5) == -1 ) {
                perror("listen");
                exit(1);
        }
```

```
/* esperando que un cliente solicite un servicio */
        if ( (sd_actual = accept(sd, 0, 0)) == -1) {
                perror("accept");
                exit(1);
        }

/* tomar un mensaje del cliente */
        if ( read(sd_actual, dir, sizeof(dir) ) == -1) {
                perror("read");
                exit(1);
        }

/* realizando servicio solicitado: leyendo el directorio */
        lee_dir(dir);
```

```
/* enviando la respuesta del servicio */
        if ( write(sd_actual, dir, sizeof(dir) ) == -1) {
                perror("write");
                exit(1);
        }

/* cerrar los dos sockets */
        close(sd_actual);   close(sd);

/* no olvidar realizar un poco de limpieza */
        unlink("./socket");

}
```

Versión orientada conexión con sockets tipo Unix
Código del Cliente

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

#define  DIRSIZE   8192                    /* tamaño máximo parámetro entrada/salida */


main(argc,  argv)
int argc; char *argv[];
{
        char  dir[DIRSIZE]; /* paramétro entrada y salida */
        int     sd;                       /* descriptor del socket */
        struct sockaddr_un   sin;         /*  direcciones socket */
```

```
/* verificando número parámetros entrada */
        if (argc != 2) {
                fprintf(stderr,"Error, uso: %s <directorio> \n",argv[0]);
                exit(1);
        }

/* obtención de un socket tipo AF_UNIX  (igual que el servidor) */
        if ( (sd = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
                perror("socket");
                exit(1);
        }

/* llenando los campos de la estructura de dirección unix, mismos datos servidor */
        strcpy(sin.sun_path, "./socket");
        sin.sun_family = AF_UNIX;

/* conectandose  al archivo ./socket */
        if ( connect(sd, (struct sockaddr *)&sin, sizeof(sin)) == -1) {
                perror("connect");
                exit(1);
        }
```

```
/* enviar mensaje al servidor */
        strcpy(dir, argv[1]);
        if ( write(sd, dir, sizeof(dir)) == -1) {
                perror("write");
                exit(1);
        }

/* esperar por la respuesta */
        if ( read(sd, dir, sizeof(dir) ) == -1) {
                perror("read");
                exit(1);
        }

/* imprimir los resultados y cerrando la conexion del socket */
        printf("%s \n", dir);
        close(sd);
}
```

Versión “resumida” del servidor

```
1   main()
2   {
3           sd = socket(AF_UNIX, SOCK_STREAM, 0);
4           strcpy(sin.sun_path, “./socket”);
5           sin.sun_family = AF_UNIX;
6           bind(sd, (struc sockaddr *)&sin, sizeof(sin));
7           listen(sd, 5);
8           sd_actual = accept(sd, 0, 0);
9           read(sd_actual, dir, sizeof(dir) );
10          lee_dir(dir);
11          write(sd_actual, dir, sizeof(dir));
12          close(sd_actual);
13          close(sd);
14          unlink(“./socket”);
15  }
```

Versión servidor serial

```
1   main()
2   {
3           sd = socket(AF_UNIX, SOCK_STREAM, 0);
4           strcpy(sin.sun_path, "./socket");
5           sin.sun_family = AF_UNIX;
6           bind(sd, (struc sockaddr *)&sin, sizeof(sin));
7           listen(sd, 5);
8           while (1) {
9                   sd_actual = accept(sd, 0, 0);
10                  read(sd_actual, dir, sizeof(dir) );
11                  lee_dir(dir);
12                  write(sd_actual, dir, sizeof(dir));
13                  close(sd_actual);
14          }
15          close(sd);
16          unlink("./socket");
17  }
```

Versión servidor padre con regreso de valores numéricos

```
main()
{
    sd = socket(AF_UNIX, SOCK_STREAM, 0);
    strcpy(sin.sun_path, "./socket");
    sin.sun_family = AF_UNIX;
    bind(sd, (struc sockaddr *)&sin, sizeof(sin));
    listen(sd, 5);
    while (1) {
            sd_actual = accept(sd, 0, 0);
            if ( fork() == 0 ) {
                    atencion(sd_actual);
                    exit(0);
            }
    }
   close(sd);
   unlink("./socket");
}
```

```
atencion(int sock)
{
    int pid;
    char res[8], pet[20];

    read(sock, pet, sizeof(pet) );
    if  (pet[0] == 'a')
        pid = getpid();
    else
        pid = getppid();
    sprintf(res,"%d", pid);
    write(sock, res, sizeof(res));
    close(sock);
}
```