PRÁCTICA 4

PASO DE MENSAJES

Introducción

Cuando los procesos interactúan unos con otros pueden necesitar intercambiar información. Uno de los métodos posibles para conseguir esto es el paso de mensajes. Además de permitir la comunicación, este método permite la sincronización mediante la exclusión mutua entre procesos y se presta muy bien para ser implementados en sistemas distribuidos así como en sistemas multiprocesador y monoprocesador de memoria compartida.

Existen diversas variantes en la implementación del paso de mensajes. En este capítulo se utilizará el sistema de paso de mensajes que emplea el sistema operativo Unix.

Conceptos básicos

Un método para direccionar los mensajes es mediante lo que se conoce como *direccionamiento indirecto*. Este método consiste en utilizar unas estructuras de datos compartidas formadas por colas que pueden guardar los mensajes temporalmente. Estas colas se denominan generalmente *buzones* (*sockets*). De este modo, para que dos procesos se comuniquen, uno envía mensajes al buzón apropiado y el otro los recoge del buzón. Cada buzón tiene colas para enviar o recibir datos.

La relación entre emisores y receptores puede ser uno a uno, de muchos a uno, de uno a muchos o de muchos a muchos. Una relación uno a uno, permite que se establezca un enlace privado de comunicaciones entre dos procesos. Una relación de muchos a uno resulta útil para interacciones cliente/servidor, donde un proceso ofrece un servicio a un conjunto de procesos.

Esta última relación ofrece una gran ventaja por su adaptación a las redes de computadoras ya que los procesos clientes o servidor pueden ejecutarse en el mismo host o pueden encontrarse en hosts distintos; pero en cualquier caso, se debe saber en qué host se encuentra el proceso servidor.

Un proceso servidor necesita realizar varias operaciones para poder aceptar peticiones por parte de los procesos cliente. Algunas de estas operaciones son: obtener la dirección del host y puerto de comunicaciones local, crear el buzón y enlazar la dirección a éste, escuchar solicitudes, aceptarlas y mantener una comunicación con el cliente. Estudiemos cada una de estas operaciones más detenidamente.

Obtener la dirección del host local

La dirección internet se obtiene mediante la función *gethostbyname* que busca en el archivo */etc/hosts* a partir del nombre oficial. Su sintaxis es la siguiente:

```
struct hostent *gethostbyname(const char *name);
```

El argumento *name* especifica el nombre del host del cual se va a obtener la dirección. La función devuelve un puntero a una estructura llamada *hostent* que contiene la dirección requerida. Su formato es el siguiente:

```
struct hostent {
    char *h_name; /* nombre oficial */
    char **h_alias; /* lista de alias */
    int h_addrtype; /* tipo de dirección */
    int h_length; /*longitud de la dirección */
    char **h_addr_list; /*lista de direcciones */
    long h_addr; /* dirección IP */
};
```

Obtener la dirección del puerto local

La función *getservbyname* permite obtener el número del puerto local de un servicio a partir del nombre del servicio. Para ello busca en el archivo */etc/services*. Su sintaxis es la siguiente:

```
struct servent *getservbyname(const char *name, const char *protocol);
```

Los argumentos que se le pasan a la función son el nombre del servicio (*name*) y la familia de protocolos (*protocol*). La función devuelve un puntero a una estructura (*servent*) que contiene el número del puerto requerido. Su formato es el siguiente:

```
struct servent {
    char *s_name; /* nombre oficial del servicio */
    char **s_aliases; /* lista de alias */
    int s_port; /* número del puerto */
    char s_proto; /* familia de protocolos */
    char **h_addr_list; /* lista de direcciones */
};
```

Crear el buzón

Para poder crear los buzones se utiliza la función *socket*. Su sintaxis es la siguiente:

```
int socket(int domain, int type, int protocol);
```

El argumento *domain* especifica la familia de protocolos. Algunos de los valores posibles son AF_INET, AF_UNIX, AF_X25, etc. Utilizaremos en nuestro estudio el dominio AF_INET que corresponde al dominio Internet. *type* especifica el tipo de comunicación. Algunos de los valores posibles son SOCK_STREAM, SOCK_DGRAM, SOCK_RAW, etc). Para el dominio Internet debemos escoger SOCK_STREAM; y por último, *protocol* especifica el protocolo específico que se va a usar. La función devuelve un entero que identifica al buzón recién creado.

Enlazar dirección al buzón

La función *bind* enlaza una dirección a un buzón. Su sintaxis es:

```
int bind(int s, const struct sockaddr *name, int namelen);
```

Los argumentos que se le pasan son el identificador del buzón (*s*) que va a ser enlazado, un puntero a una estructura de dirección (*name*) y el tamaño de dicha estructura (*namelen*). El formato de una estructura de dirección es el siguiente:

```
struct sockaddr_in {
    short sin_family; /* familia de dirección */
    short sin_port; /* número de puerto */
    long sin_addr; /* dirección IP */
    . . .
};
```

Escuchar solicitudes

La función *listen* solamente indica que la aplicación desea aceptar solicitudes de conexión. Su sintaxis es la siguiente:

```
int listen (int s, int max);
```

Los argumentos que se le pasan son el identificador del buzón (*s*) y el número máximo de solicitudes pendientes en la cola mediante el parámetro *max*.

Aceptar solicitudes

La función *accept* obtiene la primera conexión de la cola de conexiones pendientes, crea un nuevo buzón con las mismas propiedades que tiene el buzón que se utiliza para aceptar solicitudes y asigna un nuevo descriptor para el buzón recién creado. Su sintaxis es:

```
int accept (int s, struct sockaddr_in *addr, int *len);
```

Los argumentos que se le pasan a la función son el identificador de buzón (*s*), un puntero a una estructura de dirección (*addr*) y un puntero a un entero que contiene el tamaño de dicha estructura (*len*). La función devuelve el nuevo descriptor de buzón.

Enviar/recibir información

La función *send* se utiliza para enviar datos a un buzón conectado. Su sintaxis es:

```
int send(int s, const char *msg, int len, int flags);
```

Los argumentos que se le pasan son el identificador del buzón (*s*), un puntero al buffer desde donde se van a enviar los datos (*msg*), la longitud del buffer de recepción (*len*) y por último se le pasa el parámetro *flags* para especificar unas opciones que permiten, entre otras cosas, ser usados por programas de diagnóstico por ejemplo.

La función *recv* se utiliza para leer datos desde un socket conectado. Su sintaxis es:

int recv(int s, char *buf, int len, int flags);

Los argumentos que se le pasan son el identificador del buzón devuelto por la función accept (*s*), un puntero al buffer donde se van a recibir los datos (*buf*) y la longitud del buffer de recepción (*len*). El parámetro *flags* tiene el mismo significado que en *send*.

Un proceso cliente necesita realizar algunas de las operaciones anteriormente indicadas y otras que desarrollamos a continuación para poder solicitar servicios por parte de un proceso servidor:

Obtener la dirección remota

Se utiliza la función *gethostbyname*, de forma similar a como se emplea en el proceso servidor.

Obtener la dirección del puerto remoto

Se utiliza la función *getservbyname*, de forma similar a como se emplea en el proceso servidor.

Crear un buzón

Se utiliza la función *socket* para crear buzones, de forma similar a como se emplea en el proceso servidor.

Conectar con el servidor

La función *connect* permite a un cliente iniciar una conexión con el buzón de un servidor. Su sintaxis es:

int connect(int s, struct sockaddr *name, int namelen);

Los argumentos que se le pasan a la función son el identificador del buzón (*s*), un puntero a una estructura de dirección (*name*) y un puntero a un entero que contiene el tamaño de dicha estructura (*namelen*).

Enviar/recibir información

Se utilizarán las funciones *send* o *recv* según sea el flujo de información que interese. Se emplea de forma similar a como se ha visto para el proceso servidor.

En los programas 1 y 2 podemos ver un ejemplo que nos permite comprobar cómo se utilizan las llamadas al sistema que permiten crear un modelo cliente/servidor mediante buzones.

Programa 1: Proceso cliente para la implementación del paso de mensajes

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

main ()
{
    /*declaración de variables y estructuras de datos */
    int retcode, sock;
    struct sockaddr_in name;
    struct servent *servrec;
    struct hostent *hostrec;
    char buf[40], host[30];
    int flag=0;

    /* Obtener host remoto del proceso servidor */
    if ((hostrec=gethostbyname("almanzora"))==NULL)
```

```
        error (1);

    /* Obtener puerto remoto del servicio */
    if ((servrec=getservbyname("servicio1","tcp"))==NULL)
        error (2);

    /* Crear un socket */
    if ((sock=socket(AF_INET, SOCK_STREAM, 0))==-l)
        error (3);

    /* Actualizar nombre de socket */
    name.sin_port = servrec->s_port;
    name.sin_family = hostrec->h_addrtype;
    name.sin_addr.s_addr = INADDR_ANY;

    /* Contactar con el servidor */
    if ((retcode=connect(sock,&name,sizeof(name)))==-1)
        error (4);

    /* Enviar información al servidor */
    strcpy (buf, "Esto es un mensaje");
    if ((retcode=send(sock,buf,sizeof(buf),flag))==-1)
        error (5);

    /* Cerrar conexión */
    if (close (sock)==-1) error (6);
} /* fin de main */

error (n)
    int n;
{
    switch (n) {
        case 1: printf ("Error en gethostbyname \n"); break;
        case 2: printf ("Error en getservbyname \n"); break;
        case 3: printf ("Error en socket \n"); break;
        case 4: printf ("Error en connect \n"); break;
        case 5: printf ("Error en send \n"); break;
        case 6: printf ("Error en close \n"); break;
    } /* fin de switch */
    exit (1);
} /* fin de error */
```

Programa 2: Proceso servidor para la implementación del paso de mensajes

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

main ()
{
    /* declaración de variables y estructuras de datos */
    char host[30];
    struct hostent *hostrec;
    struct servent *servrec;
    struct sockaddr_in name, addr;
    int retcode, sock, newsock;
    int len=0;
    char buf[40];

    /* Obtener nombre del host local */
```

```
    if ((hostrec=gethostbyname("almanzora"))==NULL)
        error (1);

    /* Obtener puerto local */
    if ((servrec=getservbyname("servicio1","tcp"))==NULL)
        error (2);

    /* Crear un socket */
    if ((sock=socket(AF_INET, SOCK_STREAM, 0))==-1)
        error (3);

    /* Actualizar nombre de socket */
    name.sin_port = servrec -> s_port;
    name.sin_family = hostrec -> h_addrtype;
    name.sin_addr.s_addr = INADDR_ANY;

    /* Enlazar nombre al socket */
    if ((retcode=bind(sock,&name,sizeof(name)))==-1) {
        close (sock);
        error (4);
    }

    /* Escuchar solicitudes */
    if ((retcode=listen(sock, 1)) == -1) error (5);

    /* Aceptar solicitudes */
    memset (&addr, 0, sizeof (addr));
    len = sizeof (addr);
    if ((newsock=accept(sock, &addr, &len)) == -1)
        error (6);

    /* Recibir un mensaje */
    if (retcode=recv(newsock,buf,sizeof(buf),0)==-1)
        error (7);

    /* Visualizar informacion transferida */
    printf (" %s \n", buf);

    /* Cerrar socket */
    if (close (newsock) == -1) error (8);
} /* fin de main */

error (n)
    int n;
{
    switch (n) {
        case 1: printf ("Error en gethostbyname\n"); break;
        case 2: printf ("Error en getservbyname\n"); break;
        case 3: printf ("Error en socket\n"); break;
        case 4: printf ("Error en bind\n"); break;
        case 5: printf ("Error en listen\n"); break;
        case 6: printf ("Error en accept\n"); break;
        case 7: printf ("Error en recv\n"); break;
        case 8: printf ("Error en close\n"); break;
    } /* fin de switch */
    exit (1);
} /* fin de error */
```

Ejercicio

Crear un programa que simule el juego de adivinar una frase (juego del ahorcado). Para ello, se creará un proceso cliente y otro proceso servidor. El proceso servidor solicitará a un usuario una frase y, posteriormente, recibirá del proceso cliente letras para comprobar si se encuentran en la frase. En caso afirmativo, el servidor informa al cliente de las posiciones donde se encuentra la letra; en caso negativo, lo que le envía es el número de errores que lleva cometidos. El

programa termina cuando se cometen 5 errores o cuando se adivina la frase completa. Limitar la frase para que no tenga más de 40 caracteres (incluyendo los espacios en blanco).

Información complementaria

En el capítulo 11 de [BACH86] y el 3 de [ANDL90] podemos encontrar información sobre la estructura de buzones implementada en Unix. [RIWE92] es una obra dedicada exclusivamente a la programación de aplicaciones de red. En este libro podemos encontrar toda la información necesaria para programar en los distintos dominios mediante buzones y con otras estructuras más avanzadas. En el capítulo 7 de [ROCH85] podemos encontrar información de otro método de implementación de mensajes mediante FIFOs. El capítulo 12 de [ROBB96] también desarrolla la comunicación cliente/servidor mediante diversos mecanismos de comunicación, entre los que se encuentran los buzones.

Otra fuente de información aparece en la ayuda en línea (*man*) de las llamadas al sistema estudiadas.