EXPLICACION PRACTICA 2 SO2 2010/2011
20031707-H JOSE MANUEL MORATO ESCANDELL

1. ENUNCIADO DE LA PRACTICA

Las arquitecturas cliente/servidor son actualmente muy utilizadas en el desarrollo de aplicaciones distribuidas. En esta práctica se propone implementar una arquitectura de este tipo mediante colas de mensajes. Básicamente, se trata de desarrollar un programa que acepta peticiones por parte de otros programas; aquel que atiende las peticiones es el servidor, y el que realiza las peticiones es el cliente.

Si se quiere realizar una arquitectura cliente/servidor mediante colas de mensajes, se necesitan dos colas de mensajes:

- Cola Cliente - Servidor (CCS) para canalizar el paso de la información del cliente al servidor.
- Cola Servidor - Cliente (CSC) para canalizar el paso de la información del servidor al cliente.

De esta manera, el servidor no gestionará su propia información y los clientes no escucharán sus propios mensajes. Esto es similar a realizar una comunicación bidireccional mediante el uso de tuberías.

Para poder desarrollar el sistema compuesto por dos procesos independientes (el servidor y el cliente), éstos deben compartir el tipo de datos que van a enviar/recibir por la cola de mensajes, el nombre de las colas y el conjunto de órdenes y estados. Esto configura el nivel de aplicación del protocolo de comunicación. Los mensajes que se intercambian deben incluir un tipo que permita identificar qué proceso es el que se ha conectado al servidor y realiza las peticiones, así como qué proceso debe recibir la información que proporciona el servidor. Este tipo va a ser el PID del proceso.

Teniendo en cuenta esto, se pretende desarrollar una aplicación con una estructura de cliente/servidor que gestione un listado de datos sobre los alumnos/as de una clase.

2. EXPLICACIÓN

Las colas de mensajes son uno de los mecanismos de comunicación entre procesos que se introdujo en el UNIX System V. Una cola es una estructura de datos gestionada por el núcleo y donde podrán escribir varios procesos. Los datos que se escriben en la cola deben tener formato de mensaje y son tratados como un todo indivisible.

La cola se gestiona como un mecanismo con disciplina de hilera, donde el primer mensaje que entra es el primero que sale. Para dotar de más flexibilidad a las colas, se pueden hacer peticiones de lectura para extraer un mensaje de un tipo determinado, con lo que se rompe la gestión en hilera, aunque se sigue manteniendo para todos aquellos mensajes que son de un mismo tipo. Esta clasificación de los mensajes por tipos permite distinguir cuáles son los mensajes que van destinados a cada uno de los procesos lectores. También se pueden hacer operaciones de lectura sin especificar el tipo de mensaje, con lo que se fuerza a extraer el primer mensaje que haya en la cola sin importar el tipo del mismo.

La llamada que permite crear una cola de mensajes es msgget y su declaración es:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget (key_t key, int msgflag);
```

si la llamada se ejecuta satisfactoriamente la función devuelve el identificador de la cola creada, en caso contrario devuelve -1 y en errno estará el código del error producido.

El parámetro key es una variable o constante de tipo key_t que usaremos para acceder a los mecanismos IPC previamente reservados o para reservar otros nuevos. Esta llave de los mecanismos IPC se puede obtener mediante la llamada a ftok que tiene la siguiente declaración:

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok (char *path, char id);
```

si el fichero que se indica en path no es accesible para el proceso, bien porque no existe, bien porque sus permisos lo impide, ftok devolverá el valor (key_t) – 1, indicando así que se ha producido un error en la creación de la llave. El parámetro id es un carácter que identifica el proyecto; para un mismo fichero, ftok devolverá diferentes llaves para distintos valores de id.

El segundo parámetro de la llamada a msgget, msgflag, es una máscara de bits que indica el modo de adquisición del identificador. Si el bit IPC_CREAT está activo, la facilidad IPC se creará en el supuesto de que otro proceso no la haya creado antes. Si los bits IPC_CREAT e IPC_EXCL están activos simultáneamente, la llamada falla en el caso de que la cola ya esté creada. Los nueve bits menos significativos de msgflag indican los permisos de la cola de mensajes.

Para enviar y recibir mensajes de la cola utilizaremos las llamadas msgsnd y msgrcv respectivamente, cuya declaración se explica en las lineas posteriores:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd (int msqid, void *msgp, int msgsz, int msgflg);
int msgrcv (int msqid, void *msgp, int msgsz, long msgtyp, int msgflg);
```

en ambas llamadas, msqid es el identificador de la cola sobre la que queremos trabajar; msgp apunta a la memoria intermedia con los datos que vamos a enviar o recibir, la composición de esta memoria la define el usuario, sólo es obligatorio que el primer campo sea de tipo long y se utiliza para identificar el tipo de mensaje, no existen tipos definidos por el sistema, por tanto, la clasificación de los mensajes depende del programador; msgsz es el tamaño, en bytes, del mensaje que queremos enviar o recibir, en este tamaño no se incluyen los bytes que ocupa el campo long ¨tipo del mensaje¨; msgtyp sólo aparece en la llamada de lectura y especifica el tipo del mensaje que queremos leer, puede tomar los siguientes valores:

- msgtyp = 0, leer el primer mensaje que haya en la cola.
- msgtyp > 0, leer el primer mensaje de tipo msgtyp que haya en la cola.
- msgtyp < 0, leer el primer mensaje que cumpla que su tipo es menor o igual al valor absoluto de msgtyp y a la vez sea el más pequeño de los que hay.

El parametro msgflg es un mapa de bits y tiene distintos significados según aparezca en la llamada msgsnd o msgrcv. Para la llamada msgsnd (escritura en la cola), cuando la cola está llena:

- Si el bit IPC_NOWAIT está activo, la llamada devuelve el control inmediatamente y retorna el valor -1.
- Si el bit IPC_NOWAIT no está activo, el proceso suspende su ejecución hasta que haya espacio libre en la cola

Para la llamada msgrcv (lectura de la cola), cuando no hay ningún mensaje del tipo especificado:

- Si el bit IPC_NOWAIT está activo, la llamada devuelve el control inmediatamente y retorna el valor -1.
- Si el bit IPC_NOWAIT no esta activo, el proceso suspende su ejecución en espera de que

haya un mensaje del tipo deseado.

La llamada para leer y modificar la información estadística y de control de una cola es msgctl:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl (int msqid, int cmd, struct msqid_ds *buf);
```

el parámetro msqid es un identificador creado mediante una llamada previa a msgget e indica la cola sobre la que vamos a actuar; cmd es el tipo de operación que queremos llevar a cabo y sus posibles valores son:

- IPC_STAT, lee el estado de la estructura de control de la cola y lo devuelve a través de la zona de memoria apuntada por buf.
- IPC_SET, inicializa los campos de la estructura de control de la cola. El valor de estos campos se toma de la estructura apuntada por buf.
- IPC_RMID, borra del sistema la cola de mensajes identificada por msgqid. Si la cola está siendo usada por otros procesos, el borrado no se hace efectivo hasta que todos los procesos terminan su ejecución.

Después de leer y entender la teoría, paso a explicar en las siguientes secciones cada uno de los archivos que componen el sistema.

2.1 comun.h

Tal y como se recomienda en el enunciado de la práctica, en el archivo comun.h he declarado las constantes y estructuras comunes que se utilizarán tanto en el programa cliente como en el servidor.

Las primeras lineas, como también se recomienda en la ayuda 1 de la práctica, declaran una definición condicional para evitar las inclusiones repetitivas y a la vez agilizar el proceso de compilación.

```
#ifndef _COMUN_
#define _COMUN_
.
.
.
#endif
```

Las siguientes 4 constantes se utilizarán para identificar fácilmente las posibles ordenes que pueden intercambiar cliente y servidor:

```
#define PEDIR_DATOS 1
#define INSERTAR_DATO 2
#define FIN_DE_COMUNICACION 3
#define ERROR 4
```

A continuación, definimos el archivo que se utilizará para la creación de las colas y las claves asociadas a las colas cliente-servidor y servidor-cliente:

```
#define FICHERO ".colas"
#define CLAVE_CCS 'C'
#define CLAVE_CSC 'S'
```

Acto seguido, declaramos la estructura alumno que contendrá todos los datos requeridos para un alumno en cuestión:

```
struct alumno
{
      int alumnoID;
      char nombre[50];
      char apellido1[50];
      char apellido2[50];
      float nota;
};
```

Esta estructura formará parte de los mensajes que intercambiarán el cliente y el servidor:

```
struct mensaje
{
      long pid;
      int orden;
      struct alumno datos;
};
```

como podemos ver la estructura del mensaje contiene un primer campo de tipo long, pid, que se utilizará como identificador del tipo de mensaje, tal como requiere el núcleo de UNIX para trabajar con las colas de mensajes. También se define un campo, orden, para identificar que comando se esta enviando (PEDIR_DATOS, INSERTAR_DATO, FIN_DE_COMUNICACION o ERROR). Finalmente, definimos el campo datos de tipo alumno que contendrá los datos del alumno en las ordenes que sea necesario.

Por lo tanto, y para finalizar el archivo comun.h, definimos la constante LON_MENSAJE:

```
#define LON_MENSAJE  sizeof(int) + sizeof(struct alumno)
```

como vemos, esta constante sera la suma del tamaño de un dato de tipo entero mas el tamaño de la estructura alumno. Esto es así, por que no se debe tener en consideración el campo del tipo de mensaje a la hora de enviar o recibir mensajes de una cola de mensajes.

2.2 servidor.c

En primer lugar, encontramos la inclusión de las librerías que utilizamos para el programa:

- stdio.h: donde encontramos las llamadas principales relacionadas con la entrada/salida
- stdlib.h: donde se encuentran algunas funciones comunes propuestas por el estandard ANSI; entre ellas la función exit()
- sys/types.h: donde se definen muchos de los tipos utilizados en el núcleo de UNIX.
- sys/ipc.h: donde se definen estructuras y constantes comunes para los mecanismo de iPC
- sys/msg.h: donde se definen constantes y estructuras especificas para el manejo de colas de mensajes.
- comun.h: El archivo de protocolo común explicado anteriormente.

Después de la inclusión de los archivos de cabecera, definimos una constante, DB, que identificará el archivo que se debe usar como base de datos para los alumnos.

A continuación se puede ver el prototipo de las funciones que se utilizarán en el programa y, justo después, las variables globales que utilizaremos (creo que los nombres son auto-explicativos así que no me extenderé explicando para que se utiliza cada una).

En la próxima línea de código encontramos la función main, que sirve como punto de entrada de un programa en c y toma como argumentos:

- argc: Número de parámetros con el que se ha llamado al programa
- argv: Array de cadenas que contiene los parámetros de entrada al programa

Con una visión global de la función main, podemos comprobar que esta función se encargará de truncar o crear el archivo utilizado para las colas de mensajes, inicializar las variables globales que serán utilizadas en el programa y finalmente entrar en un bucle infinito a la espera de recibir mensajes desde la cola clientes-servidor.

Cada vez que se reciba una orden desde el cliente, el servidor llamará a la función correspondiente para su tratamiento:

- PEDIR_DATOS => PedirDatos(struct mensaje mensajeRcv): Cuando un cliente solicite el listado de los datos guardados en la base de datos, el servidor abrirá el fichero de la base de datos en modo lectura (en caso de error informará de la situación por la consola y enviara el mensaje de ERROR al cliente) y a continuación leerá el contenido del archivo y enviará un mensaje al cliente por cada alumno que contiene la base de datos. Finalmente, enviará la orden FIN_DE_COMUNICACION que indicará al cliente que todos los datos han sido enviados correctamente.
- INSERTAR_DATO => InsertarDatos(struct mensaje mensajeRcv): Si el cliente desea insertar un nuevo registro a la base de datos, el servidor abrirá el archivo en modo añadir (en caso de error informará de la situación por la consola y enviara el mensaje de ERROR al cliente) y a continuación escribirá los datos de la estructura alumno que se encuentran en el campo datos del mensaje recibido mediante la llamada a fwrite. Finalmente, enviará las ordenes de INSERTAR_DATOS y FIN_DE_COMUNICACION al cliente que le indicarán que los datos han sido insertados correctamente.
- FIN_DE_COMUNICACION => CerrarServidor(): Esta función es la encargada de borrar las colas de mensajes mediante la llamada a msgctl y finalizar la ejecución del proceso servidor.

Como se puede observar en el código, me he ayudado de la función EnviarCSCMsg para enviar mensajes a la cola de cliente-servidor. Esta función simplemente declara una estructura de tipo mensaje, la inicializa con los datos pasados como argumento a la función y los enviá a la cola cliente-servidor mediante la llamada a msgsnd. En caso de que que se produzca un error en el envío del mensaje se informará por pantalla de la causa del error.

2.3 cliente.c

En primer lugar, encontramos la inclusión de las librerías que utilizamos para el programa:

- stdio.h: donde encontramos las llamadas principales relacionadas con la entrada/salida
- stdlib.h: donde se encuentran algunas funciones comunes propuestas por el estandard ANSI; entre ellas la función exit()
- sys/types.h: donde se definen muchos de los tipos utilizados en el núcleo de UNIX.
- sys/ipc.h: donde se definen estructuras y constantes comunes para los mecanismo de iPC
- sys/msg.h: donde se definen constantes y estructuras especificas para el manejo de colas de mensajes.
- strings.h: donde se encuentran diversas utilidades para el procesamiento de cadenas de caracteres.
- comun.h: El archivo de protocolo común explicado anteriormente.

A continuación se puede ver el prototipo de las funciones que se utilizarán en el programa y, justo después, las variables globales que utilizaremos (creo que los nombres son auto-explicativos así que no me extenderé explicando para que se utiliza cada una).

En la próxima línea de código encontramos la función main, que sirve como punto de entrada de un programa en c y toma como argumentos:

- argc: Número de parámetros con el que se ha llamado al programa
- argv: Array de cadenas que contiene los parámetros de entrada al programa

Al iniciar la ejecución del programa, inicializamos las variables que almacenarán las llaves del mecanismo IPC y los identificadores de las colas cliente-servidor y servidor-cliente a su valores de error. Esto nos indicará que la conexión con el servidor no ha sido aún iniciada o que fue finalizada.

A continuación, guardamos el valor del pid del proceso en la variable currentPid que la utilizaremos en el campo pid de los mensajes que enviaremos al servidor para identificar la procedencia del mensaje.

Justo después, imprimimos el mensaje de bienvenida al llamar a la función ImprimirMensaje() y esperamos la primera entrada del usuario llamando a la función RecivirEntrada(). La función RecivirEntrada() muestra un mensaje que las posibles opciones del programa y llama a la función scanf que espera la entrada de usuario y la almacena en la variable “entrada”.

Si el usuario inserta la opción “s”, romperá el bucle de recepción de entradas, imprimirá el mensaje de salida y terminará la ejecución del proceso.

En caso de que inserte la opción “i”, “p” o “c” se ejecutará la función InsertarDatos(), PedirDatos() o CerrarServidor() respectivamente. En caso de insertar cualquier otra opción se imprimirá el mensaje “Opción incorrecta”.

Estas tres funciones,InsertarDatos(), PedirDatos() y CerrarServidor(), empiezan comprobando si se ha creado o si se puede crear una conexión al servidor llamando a la función Conectar(). La función Conectar() comprueba los valores de los identificadores de las colas cliente-servidor y servidor-cliente y en el caso de que se encuentren en su valores por error, intentará obtener de nuevo la conexión al servidor al obtener los identificadores validos para las colas de mensajes mediante la llamada a msgget. La función devuelve 1 en caso de obtener identificadores válidos o 0 en caso contrario.

La función InsertarDatos() , después de comprobar la conexión con el servidor, preparará el mensaje de INSERTAR_DATOS preguntando por los datos del alumno que se desea insertar al usuario del programa cliente. Después, enviará los datos recogidos a la cola cliente-servidor utilizando la llamada a msgsnd y esperará la respuesta del servidor llamando a la función ProcesarMensajes(int ordenEnviada) que será explicada mas adelante.

La función PedirDatos() enviará el mensaje PEDIR_DATOS al servidor y esperará la respuesta de este llamando también a la función ProcesarMensajes(int ordenEnviada).

Por último la función CerrarServidor() enviará el mensaje FIN_DE_COMUNICACION a la cola cliente-servidor, reinicializará los valores de las variables de los identificadores de las colas e imprimirá el mensaje “Servidor cerrado”.

La función ProcesarMensajes, que es llamada desde InsertarDatos y PedirDatos, es la encargada de procesar la respuesta del servidor después de haberse enviado una petición a este. Para ello, crea un bucle infinito que solo se rompe cuando se recibe un mensaje desde el servidor, a través de la cola servidor-cliente, de tipo FIN_DE_COMUNICACION, ERROR o un tipo de mensaje desconocido. En caso de que se reciba un mensaje de tipo PEDIR_DATOS, se imprimirá en pantalla un mensaje con los datos correspondiente al alumno almacenado en mensajeRcv.datos. Si se recibe el mensaje INSERTAR_DATO, se tendrá la confirmación de que la petición de insertar datos ha sido procesada correctamente por el servidor. En ambos casos se esperará al mensaje FIN_DE_COMUNICACION para salir de la función.