Trabajo Práctico 1
Sistemas Operativos

Especificaciones del sistema cliente-servidor

Alumnos:

di Tada, Teresa (Legajo: 52354)

Rey, Juan Pablo (Legajo: 50265)

Fecha de entrega:

31 de Marzo de 2014

Introducción

El trabajo práctico consta de la realización de múltiples implementaciones de un sistema cliente-servidor a definir utilizando distintos mecanismos IPC. El sistema cliente-servidor elegido es un sistema de manejo de stock rudimentario, utilizando como ejemplo en el presente informe el stock de una librería.

Implementaciones a realizarse:

1. Clientes con acceso directo a la base de datos, con file control como mecanismo de exclusión.
2. Base de datos gestionada por un servidor, quien proporciona a los clientes los servicios necesarios:
   (a) Clientes locales, los cuales correrán en la misma máquina que el servidor.
   (b) Clientes externos, los cuales correrán máquinas diferentes a la del servidor.

Prestaciones del sistema

El sistema permitirá a los usuarios:

- *Ver* todos los datos de productos existentes en el stock
- *Agregar* nuevos productos al stock
- *Remover* productos del stock, haciéndolos inválidos desde ese momento
- *Depositar* cantidades de productos existentes al stock
- *Sustraer* cantidades de productos existentes del stock

A su vez, se encargará de mantener consistencia en los datos del stock a todo momento, manteniendo una exclusión mutua de clientes para evitar problemas de concurrencia.

Especificación del funcionamiento del servidor y de los clientes

Cliente

El cliente constará de funciones con las cuales manejará el stock: mostrar, agregar, remover, depositar y sustraer un producto. Todas recibirán el nombre del producto como parámetro, puesto que se lo optó por sobre un id por cuestiones de simplicidad. Los prototipos de dichas funciones pueden apreciarse debajo, en la estructura de *client.h*:

Algorithm 1 Estructura de *client.h*

```
void show_product(char * prodname);
void add_product(char * prodname, int quantity);
void remove_product(char * prodname);
void deposit_product(char * prodname, int quantity);
void take_product(char * prodname, int quantity);
```

Servidor

El servidor, por otra parte, ofrecerá los servicios respectivos para obtener los datos de un producto, para insertar o modificar un producto y para removerlo -como se aclaró previamente, el servidor, en las implementaciones en las que esté presente, será el único en contactarse con la base de datos.

Todas las funciones del servidor recibirán el id del cliente que realizó el llamado, un mensaje genérico y su dimensión, y en todos los casos retornarán códigos de tipo *srv_ret_code*, los cuales indicarán si el procesamiento del request se hizo de manera adecuada o si hubo algún problema (por ejemplo, si el mensaje recibido fue inválido, o si el producto solicitado es inexistente).

Algorithm 2 Estructura de *server.h*

```
srv_ret_code srv_write_product(int fromId, void * msg, int msglen);
srv_ret_code srv_get_product(int fromId, void * msg, int msglen);
srv_ret_code srv_remove_product(int fromId, void * msg, int msglen);
```

Comunicación entre cliente y servidor

La comunicación entre partes en los casos en los que el servidor esté presente se hará a través de las funciones *ipc_send* e *ipc_recv*. Estas utilizan los identificadores de los procesos para conocer quién envió o va a recibir la señal (en el caso de *ipc_recv*, por no ser estrictamente necesario saber quién envió el pedido, se optó por que el usuario deba enviar el fromid en el cuerpo del mensaje de serle necesario).

Algorithm 3 Estructura de *communicator.h*

int ipc_init(int fromid);

int ipc_send(int fromid, int toid, void * buf, int len);

int ipc_recv(void * buf, int len); int ipc_close(int fromid);

int ipc_close(int fromid);

Base de datos

La base de datos serán archivos con el nombre del producto y la cantidad de stock.

Algorithm 4 Estructura de *productDB.h*

db_ret_code db_init();

db_ret_code db_save_product(Product product);

db_ret_code db_get_product_by_name(char * name, Product * productp);

db_ret_code db_update_product(Product product);

db_ret_code db_delete_product(char * name);

Exclusión mutua

La exclusión mutua se garantiza a través del lockeo, como se requiere en la consigna (a través del uso de *fcntl()* y advisory locking) que se realiza con un módulo de exclusión mutua. En las implementaciones con servidor, esta será garantizada por el servidor -quien llamará de todas formas a este módulo si le es necesario.

Algorithm 5 Estructura de *bdx.h*

db_ret_code dbx_save_product(Product product);

db_ret_code dbx_get_product_by_name(char * name, Product * productp);

db_ret_code dbx_update_product(Product product);

db_ret_code dbx_delete_product(char * name);

Funcionamiento general de los sistemas

Caso sin servidor

Los clientes se comunican directamente con la base de datos generando esta la exclusión mutua con advisory lockers y la función file control.

Una secuencia de uso posible sería, por ejemplo:

1. Se crea una instancia del cliente. Al crearse, esta inicializa el depósito llamando a *db_ init()* -de haber ya sido inicializada la base de datos no habrá problema.

2. Se decide sustraer un lápiz del depósito

3. Se llama a la función *take_ product()* con el nombre "lapiz" y la cantidad deseada.

4. Esta llama al módulo de exclusión mutua.

5. El módulo de exclusión mutua llama a db_get_product_by_name().

6. Se modifica la cantidad en el Product (se le resta uno).

7. Se llama a la función *db_ update_ product* con el Product modificado como parámetro.

Con clientes y servidor, conectados mediante diferentes IPCs

1. Se crea una instancia del servidor que queda iniciada y en espera de una petición de algún cliente (el cual conocerá donde esta el servidor para poder contactarse con el).

2. Varios clientes envían diferentes peticiones (por ejemplo, *take_ product(...)*) las cuales llamarán al módulo de comunicación para enviar sus respectivos mensajes al servidor.

3. Como la impleme

4. ntación base del cliente-servidor no será concurrente, el servidor, -quien estará conectado al módulo de comunicación mediante *recv(...)*- recibirá uno de los mensajes y lo atenderá, mientras que fallarán todos los demás. Para atender el mensaje, el servidor hará las llamadas correspondientes a la base de datos.

Implementación

Locks y File Control.

En el módulo de exclusión mutua se genera una estructura flock con el id del proceso cliente. Se lee la respuesta de abrir la sección de la base de datos que se desea leer con O_RDWR y se ataja el error que puede existir. Se hace lo mismo con una llamada a file control seteando un advisory lock con el modo deseado. Ahí se llama a la base de datos, y una vez que se vuelve se utiliza el file control para sacar el lock existente.

FIFO

Se inicializa un FIFO desde el server como solo lectura al iniciarlo con un nombre pactado con anterioridad. Cuando los clientes deseen leer o escribir crearán el fifo con el mismo nombre pero modos RDWR o WRONLY según corresponda. Luego escribirán o leerán.

Semáforo y memoria compartida

Cliente y servidor comparten una función de alocación de memoria compartida que se maneja con un semáforo. La memoria compartida se lee por polling y se gestiona la lectura escritura a través del semáforo. Cuando un cliente quiere leer, la variable del semáforo se modifica y si es cero se bloquea el cliente. Al finalizar el pedido se modifica la variable nuevamente de manera inversa.

Cola de mensajes

Se abre una cola de lectura para el servidor al iniciarlo. Cuando el cliente quiere conectarse con el servidor, abre dos colas una para leer la respuesta del servidor y otra para escribir.

Señales y Archivos

Las señales se utilizan para la comunicación entre el cliente y el servidor de manera que cualquiera de los dos se despierta solamente cuando le llega una señal. Una vez despierto, lee de un archivo preacordado el mensaje enviado por el otro y actúa de manera acorde.

Para no perder pedidos el servidor debe enmascar las señales en cuanto se reciba una.

Sockets

El servidor tiene una IP declarada y un puerto (los clientes lo conocen para comunicarse con el servidor). Para comenzar el servidor abre un socket y el cliente que desee conectarse abre un socket propio e intenta conectarse. El servidor debe aceptar la conexión y desde allí se comportan como si fueran dos pipes con write y read. Al finalizar el servidor cierra la conexión y al cliente le llega todo lo que ocurrió.