Facultad de Ingeniería Universidad de Buenos Aires

75.59 Técnicas de Programación Concurrente I

Trabajo Práctico N°2

Integrantes:

Padrón	Nombre	Email
87991	Ramonda, Juan Pablo	juanpr@gmail.com
89542	Hurtado, Pablo Nicolás	hurtado.pani@gmail.com
89579	Torres Feyuk, Nicolás R. Ezequiel	ezequiel.torresfeyuk@gmail.com

Índice

1. Introducción

El presente trabajo consiste en el desarrollo de una aplicación *concurrente*. El objetivo de la misma es permitir el almacenamiento de datos centralizado y la posterior recuperación de los clientes.

Debido a que la materia apunta a obtener conocimientos sobre los mecanismos de concurrencia vistos hasta el momento en la cátedra, el proyecto debe correr en una única computadora y funcionar bajo un ambiente *Unix/Linux* dado que los mecanismos vistos son los implementados por *System V*.

La implementación de la aplicación consiste en un esquema *Cliente-Servidor*. El servidor se encuentra escuchando peticiones del cliente. Los clientes deben conectarse al servidor, y luego de esto pueden utilizar los servicios provistos por el mismo.

2. Modo de Operación

2.1. Cómo compilar y correr la aplicación

Para poder correr la aplicación correctamente, lo primero que debe realizarse es compilar el programa. Para esto se ha provisto de un *Makefile* que se encagará de realizar el proceso de compilación.

Para compilar la aplicación servidor se debe ingresar el siguiente comando:

```
$ make server
```

Para compilar la aplicación cliente se debe ingresar el siguiente comando:

```
$ make client
```

En el caso de que desee compilar todas las aplicaciones juntas puede hacerlo por medio de alguno de los siguientes comandos:

```
$ make server client
$ make all
```

2.2. Casos de Uso

Como bien se dijo, la aplicación está compuesta por un *Servidor* que escucha peticiones y *Clientes* que realizan las mismas. De esta forma, el *Servidor* es un proceso que no tiene interacción con el usuario. En cambio, el *Cliente* si tiene interacción con el usuario. La aplicación *Cliente* despliega un menú con las tareas que puede realizar el mismo. A continuación se explica el modo de uso de cada una de las opciones del menú:

- **1-Leer un Registro:** Mediante esta opción, el cliente le indica al servidor qu/'e registro desea recuperar de la base de datos.

  Precondición: El servidor se encuentra corriendo.

  - El usuario solicita la lectura de un registro.
  - El programa cliente pregunta el número de registro deseado.

- El usuario ingresa el número de registro.
- El programa cliente lo envía al servidor.
- El servidor recibe el pedido, valida el número pedido y responde enviando el registro solicitado o error (E1).
- El cliente recibe la respuesta y muestra el registro al usuario.

E1) El número de registro es inválido. Corresponde a un registro inexistente en la base de datos.

- El programa cliente muestra un mensaje de error y vuelve al menú de selección de operación.

- **2-Agregar un registro a la Base de Datos:** Mediante esta opción, el cliente le indica al servidor que desea agregar un registro. Precondición: El servidor se encuentra corriendo.

  - El usuario solicita agregar un registro a la base de datos.
  - El programa cliente pregunta el campo nombre del nuevo registro.
  - El usuario ingresa el nombre.
  - El programa cliente valida el tama no del nombre (E1).
  - El programa cliente pregunta el campo dirección del nuevo registro.
  - El usuario ingresa la dirección.
  - El cliente valida el tama no de la dirección (E1).
  - El programa cliente pregunta el campo iteléfono del nuevo registro.
  - El usuario ingresa el teléfono.
  - El cliente valida el tama no del teléfono. (E1).
  - El cliente envía el registro a agregar al servidor.
  - El servidor recibe el mensaje, agrega el registro a la base de datos, y envía al cliente la respuesta.
  - El cliente recibe el mensaje y notifica al usuario del éxito de la operación.
  - El cliente muestra el menú principal.

  E1) El tamaño del campo es inválido.

  - El programa cliente muestra un mensaje de error y solicita nuevamente el ingreso del valor del campo.
  - El usuario debe ingresar el valor del campo hasta que la validación sea correcta.

- **3-Modificar un registro:** Mediante esta opción, se permite la modificación de un registro de la base de datos.

  - El usuario solicita la modificación de un registro.
  - El cliente pregunta el número de registro a modificar.
  - El usuario ingresa el número.

- El cliente pregunta el valor del campo nombre del registro.
- El usuario ingresa el nombre.
- El programa cliente valida el tamaño del nombre (E1).
- El cliente pregunta el valor del campo dirección del registro
- El usuario ingresa la dirección.
- El programa cliente valida eltama no del campo dirección (E1).
- El cliente pregunta el valor del campo teléfono del registro
- El usuario ingresa el teléfono.
- El programa cliente valida el tama no del campo teléfono (E1).
- El cliente notifica al usuario el resultado de la operación.
- El cliente muestra el menú principal.

  E1) El tama no del campo es inválido.
  - El programa cliente muestra un mensaje de error y solicita nuevamente el ingreso del valor del campo.
  - El usuario debe ingresar el valor del campo hasta que la validación sea correcta.

- **4-Eliminar un registro de la Base de Datos:** Mediante esta opción, el cliente puede eliminar un registro de la base de datos.

  Precondición: El servidor se encuentra corriendo.

  - El usuario solicita al programa cliente la eliminación de un registro.
  - El programa cliente solicita el número de registro que se desea eliminar.
  - El programa cliente envÃa la petición de eliminación al servidor.
  - El servidor realiza la petición y envía el resultado al cliente.
  - El cliente recibe la respuesta del servidor y la muestra al usuario (E1).
  - El cliente regresa al menú principal.

  E1) El número de registro ingresado no existe.

  - El cliente notifica el error al usuario.
  - El cliente regresa al menú principal.

- **5-Salir de la Aplicación:** Mediante esta opción, el cliente abandona la aplicación.

  Precondición: El servidor se encuentra corriendo.

  - El usuario solicita salir del programa cliente.
  - Finaliza el programa cliente.

3. Análisis de la Solución

3.1. División de la Aplicación en Procesos

Debido a la naturaleza de la solución adoptada, se pueden distinguir fácilmente los siguientes procesos:

- **Servidor:** Este proceso se encarga de escuchar las peticiones los clientes. Mientras ningún usuario le envíe ningún mensaje, el mismo se bloquea. Cuando un cliente le envía algún mensaje, despierta al mismo y este se encarga de responderle al usuario.
- **Cliente:** Al abrir un usuario una aplicación Cliente, se crea un nuevo proceso el cual puede comunicarse con el servidor. El Cliente envía peticiones al servidor esperando una respuesta del mismo.

3.2. Comunicación entre Procesos

A continuación se detalla como es la comunicación entre los procesos según los casos de uso:

- **Leer Registro de la Base de Datos:**
  - El servidor y un cliente se comunican.
  - El cliente envía un mensaje al servidor para leer un registro.
  - El servidor recibe el mensaje y contesta al cliente con el registro solicitado o un error.
- **Agregar Registro al Servidor:**
  - El servidor y un cliente se comunican.
  - El cliente envía un mensaje al servidor para agregar un registro a la base de datos, junto con el registro a agregar.
  - El servidor recibe el mensaje y contesta con el resultado de la operación.
- **Modificar un Registro:**
  - El servidor y un cliente se comunican.
  - El cliente envía un mensaje al servidor pidiendomodificar un registro, con el registro modificado.
  - El servidor recibe el mensaje y contesta al cliente con el resultado de la operación.
- **Eliminar un registro de la base de datos:**
  - El servidor y un cliente se comunican.
  - El cliente envía un mensaje al servidor pidiéndo la eliminación de un registro de la base de datos junto con el número de registro a eliminar.
  - El servidor le responde con el resultado de la operación.

3.3. Problemas Conocidos de Concurrencia en el Trabajo

Respecto a las transferencias, se podría comparar con el problema de *Consumidor Productor*, con la excepción de que habría un solo consumidor y un solo productor.
Sin embargo, se tienen los mismos problemas que en el modelo, ya que el lector debe esperar a que el escritor "produzca" y el escritor debe esperar a que el lector "consuma".

Respecto a la comunicación entre cliente y servidor, la comunicación también se parece al problema de *Cosumidor Productor*, con la excepción de que puede haber muchos productores y un solo consumidor. En el caso de las respuestas del servidor a cada cliente, solo hay un productor y un consumidor.
En este caso, el lector siempre debe esperar a que el escritor "produzca".

Respecto a la lectura del archivo para la transferencia, se podría comparar con el modelo de *Lectores Escritores*, aunque en la aplicación se pueden tener varios lectores para el archivo a transferir y un solo escritor para el archivo nuevo.
Aunque, análogamente al caso anterior, se pueden presentar los problemas característicos de este modelo, es decir, podría darse el caso de que se desee modificar de forma externa al archivo de lectura mientras está siendo leido. Para el archivo de escritura, podría ser que se quiera leer o escribir el archivo mientras se está escribiendo.

3.4. Mecanismos de Concurrencia Utilizados

A continuación se exhiben los mecanismos de concurrencia utilizados en la comunicación entre los procesos:

3.4.1. Envío de Mensajes entre Servidor y Cliente

Los clientes se comunican con el servidor enviando mensajes con la tarea que desean realizar. Dado que el servidor debe responder al cliente sobre el estado de la operación realizada, los mecanismos posibles para establecer esta comunicación podrían ser *Fifos* o *Memoria Compartida*. No se podría haber utilizado *pipes* dado que el cliente debería ser hijo del servidor, cosa que no ocurre.

Se decidió utilizar *Fifos* para establecer esta comunicación dado que los mensajes que envía el cliente tienen distintos tamaños según la tarea a realizar, y en algunos casos como a la hora de compartir archivos, varían según la longitud del path del archivo compartido.

De esta forma, el servidor posee una *Fifo* para recibir mensajes del cliente y cada cliente posee una *Fifo* para escribir mensajes al servidor. Con esta implementación surge un inconveniente: Una *Fifo* es un mecanismo de concurrencia unidireccional que asegura el sincronismo entre procesos. Sin embargo, dado que todos los clientes conectados poseen la *Fifo* de lectura del servidor abierta, ya no existe sincronismo entre los procesos.

Para solucionar este problema, se agrega un semáforo. Este semáforo se encarga de bloquear al servidor mientras que no haya algún cliente que intente escribir algún mensaje. El cliente que desea escribir un mensaje incrementa el semáforo y despierta al servidor para procesar el mensaje.

Los mensajes que el servidor envía a cada cliente son entregados también por medio de *Fifos*. En este caso, el servidor posee una *Fifo* por cada cliente que se encuentre conectado, por lo cual no existen problemas de sincronismo dado que este mecanismo sólo es compartido entre el servidor y un cliente particular.

3.4.2. Transferencia de Archivos

Para realizar la transferencia de archivos se podía elegir nuevamente entre *Fifos* y *Memoria Compartida*. En este caso, se eligió el mecanismo de concurrencia de *Memoria Compartida*. El motivo de esta elección es que de esta forma el archivo compartido se escribe en memoria y no en un archivo en disco como se lo hace utilizando *Fifos*. De esta forma, se mejora el rendimiento de la aplicación.

Dado que el tamaño del archivo puede llegar a ser muy grande, el archivo se escribe de forma particionada en la *Memoria Compartida*.

Para asegurar la sincronización entre escrituras y lecturas, se emplean dos semáforos. Un semáforo bloquea la lectura de la *Memoria*. Esto se requiere en caso de que otro proceso se encuentre escribiendo la misma. Al terminar de escribir, el proceso desbloquea la *Memoria Compartida* y el proceso que deseaba realizar una lectura se libera para poder completarla. El otro semáforo bloquea la escritura de la *Memoria*. Esto se requiere en caso de que un proceso se encuentre leyendo la memoria a la hora de que otro quiera escribirla. En este caso, el proceso que escribe se suspende hasta que el proceso que lee incremente el semáforo, lo cual se da cuando termina de leer.