Proyecto I

Facultad de Ciencias Exactas y Naturales

Escuela de Informática

EIF-508: Sistemas Distribuidos

Juan Pablo Gutiérrez García

Andrés Dalolio Aguilar

Claribel Bermúdez Rojas

Profesor: Eddy Ramírez

II Ciclo 2018

Introducción

Una de las aplicaciones más comunes en sistemas distribuidos consiste en implementar un DSM (Sistema de Memoria Distribuida por sus siglas en inglés), el cual permite que múltiples páginas sean modificadas simultáneamente por varios sitios, con el fin de agilizar un proceso paralelizable que requiere de mucha memoria y que puede verse beneficiado del trabajo conjunto de un sistema distribuido. Por lo que este sistema pretende implementar una solución que ejemplifica lo dicho anteriormente.

Casos de uso

Los casos deseados para la funcionalidad de la aplicación son:

- Quiero leer y soy dueño

```
Intentando leer pagina #3.
Leyendo mi pagina #3 en su version 2.
```

- Quiero leer, no soy dueño pero tengo copia

```
Intentando leer pagina #3.
Leyendo pagina #3 (copia) en su version 8.
```

- Quiero leer, no soy dueño ni tengo copia
    - Le pido la solicitud al servidor y el servidor es el dueño de la página

```
Intentando leer pagina #2.
Conectado con el servidor.
Adquiriendo copia y leyendo pagina #2 en su version 0.
```

    - Le pido la solicitud al servidor pero el servidor no es dueño de la página, entonces el servidor me dice quien es el dueño de la página y se la pido para leer

```
Intentando leer pagina #1.
Conectado con el servidor.
Debo pedirsela a otro cliente pedir.
Debo conectarme con 127.0.0.1:6091 para pedirle pagina a leer.
Conectado con 127.0.0.1:6091
Adquiriendo copia y leyendo pagina #1 en su version 1.
```

- Quiero escribir y soy dueño

```
Intentando escribir pagina #2 .
Escribiendo mi pagina #2 en su version 14, nueva version 15 .
El cliente 127.0.0.1:2015 debe borrar su copia.
Conectado con 127.0.0.1:2015 .
```

- Quiero escribir pero no soy dueño

    - Le pido la solicitud al servidor y el servidor es el dueño de la página, entonces se la pido para escribir, por lo que ahora me convierto en dueño de la página

```
Intentando escribir pagina #1 .
Conectado con el servidor.
Conectado con 127.0.0.1:6093
Escribiendo en pagina #1 en su version: 1 nueva version: 2 .
```

    - Le pido la solicitud al servidor pero el servidor no es dueño de la página, entonces el servidor me dice quien es el dueño de la página y se la pido para
      escribir, por lo que ahora me convierto en dueño de la página

```
Intentando escribir pagina #3 .
Conectado con el servidor.
Conectado con 127.0.0.1:6099
Escribiendo en pagina #3 en su version: 3 nueva version: 4 .
```

Una vez que se termina el servidor, este debe mostrar la última versión de cada una de las páginas de las cuales los clientes son dueños.

```
Exit: terminando procesos.
Pagina #1 en su version 12.
Pagina #2 en su version 15.
Pagina #3 en su version 13.
Pagina #4 en su version 0.
```

Desarrollo

Para el desarrollo de la aplicación se crearon dos archivos programados en C: el servidor y el cliente. Para ejecutar la aplicación primero se necesita levantar el servidor el cual utiliza un archivo de configuración donde se indica la cantidad de páginas que tendrá el servidor a disposición y el puerto de trabajo. Luego, se pueden levantar la cantidad de clientes necesarios para conectarse con el servidor donde cada uno de esos clientes también cuenta con un archivo de configuración que contiene la Ip del servidor, el puerto de conexión, el rango de páginas con las que trabajará el cliente, una media exponencial que se utiliza para calcular la frecuencia con la que cada cliente hará una solicitud para leer o escribir una página y una probabilidad de lectura con la cual se genera un valor aleatorio para que el cliente tome la decisión de escribir o leer la página a solicitar.

Ambos archivos utilizan diferentes librerías de C pero principalmente sys/socket.h la cual se encarga de la comunicación (envío y recepción de mensajes) entre las diferentes instancias, ya sea cliente-servidor o cliente-cliente y pthread.h que es la encargada de crear threads para ejecutar procesos de manera asíncrona sin afectar la funcionalidad principal de la aplicación.

El servidor cuenta con dos threads base, uno para atender clientes cuando estos se comuniquen en caso que quieran solicitar leer o escribir alguna página y otro thread que siempre está esperando a que el usuario digite el carácter 'E’ para detener la ejecución del servidor y a su vez todos lo clientes que estén conectados a él. El thread encargado de atender clientes, a su vez crea otro thread para atender un cliente en especifico y así evitar poner en cola a los demás clientes sino que atender a cada uno de manera inmediata. Para atender a cada cliente se implementó un método que abarca los diferentes casos de leer, escribir y si el servidor es propietario o no de la página que se solicita. Además, es importante mencionar que se añadió una bandera a cada una de las página para que si ésta página se está ateniendo para cualquier caso, se bloquea por un momento hasta que el servidor la haya desocupado y con esto evitar diferentes el manejo del mismo recurso al mismo tiempo.

En el lado del cliente, luego de conectarse con el servidor, se implementó un ciclo que cada cierto tiempo con un sleep de tiempo de la media calculada, se pide una página aleatoria del rango de páginas de trabajo establecidas y un valor también aleatorio calculado de acuerdo a la probabilidad de lectura también definida en el archivo de configuración con al cual se ejecutó el cliente. A Partir de la decisión tomada de

si leer o escribir la página seleccionada, al igual que el servidor, se implementó una función que se encarga de cubrir los diferentes casos (quiero leer y soy dueño, quiero leer y no soy dueño pero tengo una copia, quiero leer y no soy dueño ni tengo copia, quiero escribir y soy dueño, quiero escribir y no soy dueño pero tengo una copia, quiero escribir y no soy dueño ni tengo copia) y realizar las acciones necesarias sobre qué hacer con la página.

Las principales funciones utilizadas en el servidor son:

- void* clienteBorreCopia(char* host, int port) → Esta función se encarga de comunicarse con un cliente correspondiente al host y port de los parámetros el cual tiene una copia de la página donde el servidor sigue siendo el dueño en caso de que esta página haya sido modificada (que otro cliente la haya solicitado para escribirla), por lo que se le debe indicar al cliente que borre la copia que tiene de esa página.

- void* atenderCliente(void* clienteDataParam) → Esta es la función que se encarga de establecer la comunicación con un cliente específico y ejecutar las acciones que correspondientes de acuerdo a la solicitud que envió.

- void* conectarCliente() → Esta función crea el thread para establecer la comunicación entre el servidor y el cliente. El thread trabaja con la función 'atenderCliente' para que cuando esa función reciba solicitudes lo haga de manera asincrónica y no afecte la funcionalidad principal de la aplicación.

- char* clienteVersion(char* host, int port) → Esta función se ejecuta cuando se finaliza el servidor y se comunica con todos los clientes que son dueños de alguna pagina para pedirle a cada uno de ellos la última versión escrita en la página que tienen.

- void* finalizarPrograma() → Esta función es la que ejecuta las acciones necesarias una vez que el usuario digita el carácter 'E' y finaliza el servidor.

- int main(int argc, char **argv) → Es la función principal y es la que inicializa las páginas, las conexiones con los clientes y los threads base.

Las principales funciones utilizadas en el cliente son:

- calcularMedia(float valor) → Calcula para la frecuencia de solicitudes en base al valor dado en el archivo de configuración.
- getPagina(int min, int max) → Retorna la página aleatoria sobre la que se va a trabajar en base al rango de páginas que se establece en el archivo de configuración.
- getLeer(int propabilidad) → Retorna la decisión de leer o escribir en base a la probabilidad establecida en el archivo de configuración.
- clienteBorreCopia(char* host, int port) → Al igual que el servidor, el cliente puede tener una lista de otros clientes que tienen copias de las cuales él es dueño, por lo que en el momento que una de las páginas de las cuales se es dueño es modificada (ya sea que el mismo cliente que es dueño la escriba u otro cliente se la pida para escribirla) se le debería informar a los demás clientes que tienen copia de esa página que la deben borrar.
- serDueno(void* dataPagina) → Esta función se ejecuta en el momento en que un cliente se hace dueño de alguna pagina, y lo que realiza es crear un tipo de mini servidor, donde ahora el cliente se va a convertir en algún tipo de servidor para atender a los otros clientes que puedan llegar a hacerle solicitudes sobre las páginas en las que se es dueño.
- borrarCopia(void* dataPagina) → Esta función se encarga de borrar una página en caso que se lo indiquen.
- int main(int argc, char **argv) → Es la función principal que inicializa las conexiones, establece toda la configuracion y pone a funcionar el ciclo que realiza las solicitudes con las páginas.

Conclusión

En base al desarrollo de la aplicación, la investigación realizada sobre la tecnología utilizada y la búsqueda de la creación de las mejores funciones para resolver el problema, hay diversas maneras de agilizar los procesos que requieren cierto grado de comunicación con algún tipo de servidor.

Entre los problemas encontrados más relevantes podemos mencionar el choque de comunicación entre instancias al querer modificar el mismo archivo, por lo cual se utilizó un mecanismo de bloqueo a dicho archivo para controlar el acceso de los clientes.

Una de las técnicas que se aplicaron en el proyecto fue la implementación de threads para la ejecución asíncrona de procesos lo cual facilita ejecución de diferentes acciones al mismo tiempo sin interrumpir la funcionalidad principal de la aplicación, al igual que la utilización de sockets para la comunicación entre las instancias involucradas.

Además, cabe mencionar que es importante tratar de distribuir la ejecución de los procesos en diversas instancias de ejecución ya que, de la manera en la que implementamos este proyecto logramos convertir los mismos clientes en mini servidores por lo cual estos pueden brindar información a otros clientes liberando la carga de trabajo sobre el servidor principal.