UNIVERSIDAD DE COSTA RICA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS DE LA COMPUTACIÓN E INFORMÁTICA

CI-0123 Proyecto integrador de Redes de Comunicación de Datos y Sistemas Operativos

Funcionalidad e implementación final del servidor y tenedor para la recepción de solicitudes de clientes

Grupo 01 - Equipo 1:
Aarón Meza B74787
Aarón Santana C27373
Amanda Rodríguez C36725

Prof. Francisco Arroyo
Prof. Tracy Hernández

Ciclo 1 - 2025

Descripción general

En este proyecto, se implementó un servidor de figuras que atiende solicitudes de un cliente a través de un tenedor, el cual se encarga de traducir y reenviar los mensajes correspondientes. El servidor desconoce de dónde se conecta el cliente y nunca tendrá una comunicación directa con este, sino que aguardará aislado en su isla hasta obtener mensajes de un tenedor. El tenedor, por su parte, tiene conocimiento de la localización de los servidores activos y además le permite a un cliente, ya sea en su isla o en la red pública, hacerle pedidos a él.

A continuación, se describen más a detalle las funcionalidades de cada uno.

Servidor

Cuando se ejecuta el servidor, lo primero que hace es distribuir su trabajo en dos hilos diferentes, uno que se encargará de mantenerse escuchando por broadcasts de los tenedores y otro que enviará un broadcast avisando el despertar del servidor presente, tras lo cual se quedará escuchando por conexiones de tenedores.

- **Alertas de despertar y apagado**

  Al principio, el servidor envía un único broadcast a todas las islas anunciando su despertar. Sin embargo, este anuncio también se hará cada vez que el hilo que escucha broadcasts recibe uno en su puerto de escucha (1234), de manera que le anuncia únicamente al tenedor que le notificó:

  `BEGIN/ON/SERVIDOR/{IP}/{PORT}/END`

  Donde PORT es el puerto en el cual el servidor escuchará conexiones TCP.

  Si el servidor se apaga abruptamente, a través de un signal enviará otro broadcast anunciando lo ocurrido:

  `BEGIN/OFF/SERVIDOR/{IP}/{PORT}/END`

  Todas estas interacciones son por medio de comunicación no confiable, es decir, UDP.

- **Manejo de solicitudes de tenedores**

  Otro hilo se encarga de escuchar a conexiones TCP de tenedores en un puerto efímero de selección. Un proceso independiente se asigna para manejar las solicitudes de cada tenedor y se encargará de retornar la resolución, ya sea una figura específica o el directorio completo:

BEGIN/OK/{CONTENT}/END

Para esto, el servidor accede a su sistema de archivos (FS) para poder retornar el directorio o el contenido de la figura solicitada (de tenerla almacenada).

Tenedor

Al igual que el servidor, un hilo del tenedor envía un único broadcast de su despertar a las demás islas, mientras que otro hilo se queda escuchando por conexiones de clientes para poder manejarlas y un tercer hilo escucha por broadcasts de servidores.

- **Alerta de despertar**

  En este caso, solo se hace una alerta de despertar a través de un broadcast al iniciar la ejecución.

  BEGIN/ON/TENEDOR/{IP}/{PORT}/END

- **Mapeo de servidores activos**

  Cuando el hilo que escucha broadcasts recibe un nuevo mensaje UDP en su puerto (4321), de ser el mensaje de anuncio, el tenedor obtendrá la información respectiva del servidor anunciado, lo guardará en su ServerManager y de inmediato le solicitará sus figuras almacenadas para añadirlas a su manager. Si, al contrario, recibe el anuncio de desconexión, elimina dicho servidor del manager.

- **Recepción de conexiones de clientes**

  A través de un puerto efímero, el tenedor aguarda por conexiones TCP provenientes de clientes, ya sea de la red privada de su isla o desde la red pública del laboratorio. Por cada cliente, el tenedor crea un nuevo proceso que decodifica el mensaje en HTTP para traducirlo a PIGP.

Clases complementarias

Ambos programas cuentan con algunas clases o estructuras complementarias para facilitar la ejecución de sus tareas

- **Message**

  Maneja los distintos formatos de mensaje propuestos en el protocolo grupal, ya sea desde el servidor al tenedor o viceversa.
- **ProtocolParser**

  Recibe y maneja mensajes del formato del protocolo, de forma que sea posible tomar tokens seleccionados de este.
- **HttpParser**

  Usado por el tenedor, descifra mensajes de HTTP generados por el cliente para enviarlos en formato PIGP a los servidores.
- **Router**

  Facilita al tenedor el mapeo de servidores activos de los que se le ha notificado, guardándolos en un vector de parámetros y un mapa que vincula métodos y Handlers.
- **ForkRoutes**

  Maneja el router del tenedor para agregar o identificar las distintas rutas utilizables en el navegador, ya sea hacia los servidores, una figura en particular, el directorio.
- **ServerManager**

  Para el tenedor, facilita el mapeo de servidores activos de los que se le ha notificado, manejando una lista de máximo 32 servidores para localizar los ya identificados, agregar uno nuevo o eliminar aquel que se ha desconectado.

File System (versión final)

El FS se implementó con un bitmap pues ofrece la facilidad de saber cuáles bloques están libres sin tener que realizar más de una lectura y escritura a almacenamiento. De esta forma, se espera lograr que el sistema sea más eficiente y su lógica más sencilla. Como el archivo se definió de 2048 bloques, basta con un bloque para poder almacenar el bitmap. Como tiene 256 bytes, eso equivale a 2048 bits, siendo una coincidencia casi perfecta. Como el bloque 0 es para bloques nulos y el bloque 1 ya estaba reservado para el directorio, se le asignó el bloque 2.

El sistema de archivos está conformado por las siguientes clases complementarias:

- **Interface**

  Es una clase para poder probar el file system desde la terminal.

- **FileSystem**

  Es la clase principal del programa. Se encarga de integrar todas las funcionalidades de las clases para proveer un interfaz que permite agregar archivos, eliminarlos y poder acceder tanto al directorio como a los archivos.

- **Directory**

  Administra la metadata de los archivos en un lugar. Como el directorio crece dinámicamente, la clase se encarga de hacer crecer y disminuir el directorio. Tiene acceso al bitmap para poder solicitar y liberar bloques y tiene acceso al FileHandler para poder leer y escribir de esos bloques.

- **Bitmap**

  Para poder llevar un registro de los bloques que están ocupados y los que están libres, se agregó un bitmap. El bitmap tiene un puntero al FileHandler, ya que requiere poder leer del archivo el bitmap actual y poder actualizarlo (escribir).

- **FileHandler**

  La clase encargada de la manipulación del archivo. Provee un tipo de interfaz para realizar lecturas y escrituras en el archivo por bloques.