Proyecto 3

Principios de Sistemas operativos. Profesor Armando Arce Orozco. 29/10/2023

1st David Achoy Yakimova
Ingeniería en Computación
Instituto Tecnológico de Costa Rica
2020053336
achoy@estudiantec.cr

2nd Earl Alvarado Cabrera
Ingeniería en Computación
Instituto Tecnológico de Costa Rica
2020035739
earlalvarado@estudiantec.cr

***Abstract*—Este documento introduce el proyecto del curso de Principios de Sistemas Operativos en el Instituto Tecnológico de Costa Rica, cuyo objetivo es desarrollar una herramienta de sincronización de archivos para redes locales. A diferencia de soluciones comunes como DropBox o LiveDrive, que dependen de una conexión a Internet, nuestro enfoque permite la sincronización directa entre dos máquinas dentro de una red local. El programa sincronizador se ejecuta en ambas máquinas, utilizando sockets para la comunicación y un protocolo específico para la transferencia de archivos. Este proyecto aborda el desafío de las "copias en conflicto" cuando un archivo se modifica en ambas máquinas, manteniendo ambas versiones con nombres distintos para asegurar la integridad de los datos. El proyecto enfatiza en la sincronización eficiente y segura de archivos en un entorno Unix, utilizando el lenguaje C y sin depender de bibliotecas externas, alineándose con las restricciones académicas del curso. Se espera que este proyecto sirva como un modelo de referencia para la sincronización de archivos en entornos de red local y demuestre la aplicabilidad de los principios de sistemas operativos en un contexto práctico.**

***Index Terms*—file, Unix, command line, tar, simple tar, compress**

I. Introduccion

II. Descripción del Problema y Solución Implementada

A. Problema

El problema central de este proyecto es la creación de una herramienta de sincronización de archivos en redes locales que funcione independientemente de una conexión a Internet. A diferencia de soluciones existentes como DropBox, nuestro enfoque se centra en sincronizar archivos entre dos máquinas en una red cerrada. El reto incluye gestionar de manera eficiente la sincronización, manejar casos de "copias en conflicto" cuando un archivo se modifica en ambas máquinas, y asegurar la integridad y actualización de los datos sin depender de servicios externos o conexión a la web.

B. Solución Implementada

Para abordar este problema, se ha desarrollado el programa de cliente-servidor. La solución implementada consta de varios componentes que cumplen con los requerimientos del profesor:

1) **Funcionalidad de Enviar Archivos**: Se ha implementado la funcionalidad de enviar los archivos que el cliente tiene pero el servidor no.
2) **Funcionalidad de Modificar Archivos**: Se ha implementado la funcionalidad de modificar los archivos que han sido modificados en el cliente, pero no en el servidor.
3) **Funcionalidad de Borrar Archivos**: Se ha implementado la funcionalidad de borrar los archivos que han sido borrados en el cliente, pero no en el servidor.

En resumen, la solución implementada aborda de manera eficiente la la sincronizacion de archivos entre dos maquinas.

III. Descripción del problema

A. Instrucciones brindadas por el profesor

El profesor ha establecido directrices claras para la realización de este proyecto. Los estudiantes deben desarrollar una herramienta de sincronización de archivos para redes locales que no dependa de una conexión a Internet. El programa debe ejecutarse en dos máquinas simultáneamente, actuando como cliente o servidor según los parámetros ingresados. Se requiere el uso de sockets para la comunicación y la implementación de un protocolo para el intercambio de archivos. Además, es crucial manejar adecuadamente las "copias en conflicto" de archivos modificados en ambas máquinas. El proyecto debe ser realizado en grupos de máximo dos personas, empleando el lenguaje C y ambiente Unix, sin utilizar bibliotecas externas. La entrega final incluirá el código fuente, binarios generados y documentación en formato PDF.

IV. Funciones Implementadas

A. *Enviar Archivos al servidor y inicializar sockets*

```
void startClient (
    const char *directoryPath,
    FileInfo *files,
    int fileCount
    )
```

Esta función crea un inicia el socket y al mismo tiempo le envia al servidor la lista de los archivos que tiene el cliente con su respectivo estado.

B. Leer directorio

```
void readDirectory(
    const char *directoryPath,
    FileInfo **files,
    int *fileCount) {
```

La funcion se encarga de leer todos los archivos del directorio y meterlos en una lista.

C. leer archivo de estado

```
void readStateFile(
    const char *stateFileName,
    FileInfo **previousFiles,
    int *previousFileCount) {
```

La funcion se encarga de leer todos los archivos del archivo de estados y meterlos en una lista.

D. crear nombre de estado

```
char *generateStateFileName(
    const char *directoryPath);
```

La Funcion se encarga de crear el nombre para el archico de estado

E. procesar los mensajes en el servidor

```
void processClientChanges(
    int socket,
    FileInfo *serverFiles,
    int serverFileCount,
    const char *directoryPath)
```

La funcion se encarga de leer los mensajes que le llegan del cliente y procesar los archivos, en este caso los añade si no estan, los elimina o los modifica

F. Comparar los archivos del directorio con los del archivo de estado

```
void compareAndUpdateFileStates(
    FileInfo **currentFiles,
    int *currentFileCount,
    FileInfo *previousFiles,
    int previousFileCount) {
```

La funcion se encarga de leer los archvios del directorio y los archivos del archivo de estados y compararlos para saber en que trabajar.

G. procesar los mensajes en el servidor

```
void processClientChanges(
    int socket,
    FileInfo *serverFiles,
    int serverFileCount,
    const char *directoryPath)
```

La funcion se encarga de leer los mensajes que le llegan del cliente y procesar los archivos, en este caso los añade si no estan, los elimina o los modifica

H. Iniciar servidor

```
void startServer(
    const char *directoryPath,
    FileInfo *serverFiles,
    int serverFileCount) {
```

La funcion de iniciar el servidor.

V. Estructuras de Datos Utilizadas

A continuación, se presentan las principales estructuras de datos utilizadas:

A. FileInfo

```
typedef struct {
    char filename[256];
    off_t size;
    time_t mod_time;
    char status[32];
} FileInfo;
```

La estructura `FileInfo` almacena información sobre los archivos esta incluye nombre, tamaño, ultima modificacion y su estado

VI. Análisis de resultados de pruebas

A. Pruebas Realizadas

1) Prueba enviar archivos nuevos:

Objetivo: Comprobar que el programa pueda pasar archivos nuevos del al cliente al servidor
Resultado Esperado: El servidor y el cliente se actualizan con los mismos archivos
Resultado Obtenido: La sincronizacion ocurre con exito:

2) Prueba enviar archivos modificados:

Objetivo: Comprobar que el programa pueda pasar archivos modificados del al cliente al servidor
Resultado Esperado: El servidor y el cliente se actualizan con los mismos archivos
Resultado Obtenido: La sincronizacion ocurre con exito:

3) Prueba enviar archivos eliminados:

Objetivo: Comprobar que el programa pueda le diga al servidor que archivos borro el cliente
Resultado Esperado: El servidor elimina los archivos del directorio
Resultado Obtenido: La sincronizacion ocurre con exito:

VII. Conclusiones

Basado en el análisis del programa de sincronizacion.

1) **Dualidad Cliente-Servidor**: La implementación de un único programa que actúa tanto como cliente como servidor, dependiendo de los parámetros ingresados, es un diseño innovador que simplifica la ejecución y mantenimiento del software.
2) **Manejo Eficiente de Sockets**: La utilización de sockets para la comunicación entre máquinas asegura una transferencia de datos eficiente y fiable, crucial para la sincronización de archivos.
3) **Gestión de Conflictos de Archivos**: El programa maneja adecuadamente los casos de conflicto cuando un archivo es modificado en ambas máquinas, generando copias con nombres ligeramente distintos para evitar pérdida de información.
4) **Comparación y Actualización Efectivas**: La rutina de comparación de archivos basada en tamaño, fecha y hora de última modificación garantiza una sincronización precisa y actualizada de los archivos entre las dos máquinas.
5) **Uso de Protocolos y Formatos Estandarizados**: La definición y uso de un protocolo claro y formatos adecuados para la transmisión de datos asegura la interoperabilidad y la eficiencia en la comunicación.
6) **Aporte al Aprendizaje y Desarrollo de Habilidades**: Este proyecto ha contribuido significativamente al entendimiento práctico de conceptos de sistemas operativos, programación en C, y manejo de redes en un entorno Unix, fortaleciendo las competencias de los estudiantes en estas áreas cruciales.

En conclusión, esta herramienta de sincronización de archivos no solo cumple con los objetivos establecidos sino que también proporciona una base sólida para futuras investigaciones y desarrollos en el campo de la sincronización de archivos en entornos de red local. Su diseño innovador y la implementación efectiva de conceptos técnicos destacan su importancia como un proyecto educativo y como un producto de software funcional.

References