



**INSTITUTO TECNOLÓGICO DE COSTA RICA.**

**ESCUELA DE COMPUTACIÓN.**

**Principios de Sistemas Operativos.**

**Proyecto 3.**

**Elaborado por:**

**Diego José Masís Quirós - 2020403956.**

**Kevin Jose Rodriguez Lanuza - 2016112117.**

**II SEMESTRE 2023.**

## **Introducción.**

En el contexto actual, donde la conectividad a Internet no siempre está garantizada y la necesidad de mantener datos actualizados entre sistemas es crucial, surge la iniciativa de desarrollar una herramienta de sincronización de archivos sin conexión. Este proyecto busca superar los desafíos asociados con la transferencia de datos entre máquinas en entornos donde la conectividad a la red no es constante, proporcionando una solución eficiente y versátil para la gestión de archivos en entornos diversos.

La herramienta se ha concebido con el objetivo central de facilitar la sincronización bidireccional de directorios entre dos máquinas de manera eficaz y segura. A través del uso de sockets, se establece una comunicación directa que permite la transferencia de datos de manera rápida y fiable. La singularidad de esta aplicación radica en su capacidad para operar tanto como servidor como cliente, adaptándose a diferentes escenarios y facilitando la configuración según las necesidades particulares de los usuarios.

Al abordar las limitaciones de la conectividad intermitente, la herramienta no solo garantiza la actualización constante de archivos, sino que también ha sido diseñada para gestionar situaciones de desconexión de manera elegante, retomando la sincronización de forma automática cuando se restablece la conexión. Esto asegura la continuidad operativa incluso en entornos donde la estabilidad de la red no es un hecho garantizado.

Además de su enfoque en la conectividad sin conexión, la herramienta se destaca por sus funcionalidades avanzadas, como la

gestión de conflictos y un historial detallado de sincronización. Estos elementos combinados hacen de esta aplicación una solución integral y adaptable para la sincronización de archivos, brindando a los usuarios un control preciso sobre sus datos y una herramienta confiable para mantener la coherencia en ambientes de trabajo diversos y desafiantes. En las secciones siguientes, exploraremos más a fondo las características clave que hacen de esta herramienta una opción esencial para la gestión eficiente de archivos en entornos sin conexión a Internet.

## **Descripción del problema.**

Un problema que presentan los diferentes sistemas de sincronización de archivos (como DropBox, LiveDrive y SugarSync) es que solo funcionan conectados al Internet. Desde antes existían aplicaciones sencillas que permitían sincronizar máquinas en una red sin conectarse a la Web (por ejemplo, una laptop y una computadora de escritorio).

En este proyecto se pretende desarrollar una pequeña herramienta que permita la sincronización de un directorio entre dos máquinas.

## **Programa sincronizador.**

El programa sincronizador se debe ejecutar en dos máquinas al mismo tiempo. En la primer máquina el programa recibirá como parámetro únicamente el nombre del directorio a sincronizar y quedará esperando conexiones en el puerto 8889.

En la otra máquina se debe pasar como parámetros el nombre del directorio local a sincronizar y el IP de la primer máquina. Al iniciarse el programa deberá identificar los archivos en su directorio local, almacenando su tamaño, hora y fecha de la última actualización. Debe comparar dicha información con un listado de su corrida anterior, y determinar cuáles archivos han cambiado. Si algún archivo ha cambiado, se eliminó, o se creó, se debe enviar esa información a la otra máquina para que realice los cambios adecuados. Dentro de dichos cambios está solicitar el archivo actualizado a la otra máquina.

- Una vez que se realiza la sincronización, los programas que corren en las máquinas deben terminar. La siguiente vez que se desee sincronizar se debe volver a ejecutar el programa en ambas máquinas.
- Note que NO se deben escribir dos programas diferentes. Es el mismo programa que actúa como cliente o servidor dependiendo de los parámetros que se le pasen.
- Debe utilizar sockets para que se comuniquen los programas. Además, debe definir un formato y protocolo adecuado para enviar y recibir las solicitudes de archivos.
- Debe tomar en cuenta la posibilidad que un archivo haya sido modificado en ambas máquinas, provocando el caso de “copias en conflicto”. Aquí lo mejor es dejar las dos copias pero con nombres ligeramente diferentes.

## Definición de estructuras de datos.

La representación de archivos en esta aplicación se realiza mediante una estructura designada como “Archivo”. Cada instancia de esta estructura encapsula información vital, incluyendo el nombre del archivo, su tamaño, la marca temporal de la última modificación y un indicador de presencia en el directorio del mismo. La inclusión de estos atributos fundamentales facilita una representación completa y detallada de cada archivo, proporcionando la base necesaria para llevar a cabo una sincronización precisa y eficiente entre directorios. El nombre del archivo sirve como identificador único, mientras que el tamaño y la marca temporal permiten evaluar cambios y actualizaciones. El indicador de presencia en el directorio asegura la coherencia entre las máquinas, garantizando una sincronización que refleje con precisión el estado actual de los archivos en ambos extremos.

```
// DEFINICIÓN DE ESTRUCTURAS.  
  
struct Archivo{  
    char nombre_archivo[256];  
    long tamano;  
    long ultima_modificacion;  
    int presente_en_directorio;  
};
```

## Descripción detallada y explicación de los componentes principales del servidor/cliente.

El código se organiza en diferentes secciones para facilitar la manipulación e interacción de los diferentes componentes:

### Importación de librerías.

En esta sección se utiliza la librería estándar de C y se incluyen librerías específicas para sistemas Windows y Unix.

### Definición de constantes globales.

```
// DEFINICIÓN DE CONSTANTES GLOBALES.  
  
#define BUFFER_MAXIMO 131072 // 128 KB  
#define ARCHIVOS_MAXIMO 500
```

En esta sección se encuentran las siguientes constantes:

- **BUFFER\_MAXIMO:** Encargada de definir la longitud máxima de los buffers encargados del almacenamiento de bytes relacionados a la transferencia y recepción de datos mediante sockets.
- **ARCHIVOS\_MAXIMO:** Encargada de determinar la cantidad máxima de archivos que el programa es capaz de manejar en ciertas funciones del programa.

### Definición de funciones.

En esta sección se encuentran todas las funciones que el programa necesita para su adecuado funcionamiento. Las funciones que se encuentran definidas en esta sección son las siguientes:

### ❖ generar\_registro

```
void generar_registro(const char *ruta_directorio, const char *nombre_archivo){ ...  
}
```

Esta función recibe como parámetros la ruta de un directorio y el nombre de un archivo y es la responsable de generar un registro binario (con el nombre suministrado) con la información relevante de los archivos presentes en un directorio de la ruta suministrada. Para efectos de este proyecto, el archivo binario generado se llama “registro.bin” y contiene el nombre, tamaño, registro de última modificación y un identificador de presencia de todos los archivos registrados.

### ❖ imprimir

```
/* void imprimir(const char *registro){ ...  
} */
```

Esta función recibe como parámetro el nombre de un archivo binario generado por la función “generar\_registro” y es la responsable de leer su contenido y mostrarlo en consola.

Ya que esta función fue diseñada únicamente con el propósito de ser utilizada en la fase de desarrollo del programa con el fin de corroborar la coherencia de los diferentes registros que se generaban, se tomó la decisión de comentarla en el código final para evitar código muerto y en caso de que fuera necesario utilizarla en un futuro.

### ❖ borrar\_archivo\_de\_registro

```
void borrar_archivo_de_registro(const char *registro, const char *nombre_archivo){ ...  
}
```

Esta función recibirá como parámetros el nombre de un archivo binario que funge como registro y el nombre de un archivo cuya información puede o no estar registrada en el archivo binario.

La función buscará entre todos los registros del archivo binario, y si encuentra una coincidencia con el nombre del archivo suministrado, entonces borrará toda la información de ese registro coincidente.

### ❖ **borrar\_archivos\_existentes\_de\_registro**

```
void borrar_archivos_existentes_de_registro(const char *ruta_directorio, const char *registro_binario){ ...  
}
```

Esta función recibirá como parámetros la ruta de un directorio y el nombre de un archivo binario que funge como registro.

La función se encargará de analizar qué archivos se encuentran presentes dentro del directorio en la ruta suministrada, y una vez que ha recolectado los nombres de dichos archivos, procederá a borrarlos de los registros del archivo binario haciendo uso de la función “borrar\_archivo\_de\_registro”.

En el contexto del proyecto, el trabajo de esta función sirve para detectar mediante el registro si hubo eliminación de archivos por parte del usuario entre ejecuciones del programa, ya que si la función borra del registro los archivos que únicamente los archivos que aún se encuentran presentes en el directorio, significa que en el registro únicamente queda grabada la información de los archivos que fueron borrados por parte del usuario, por lo tanto haciendo uso posteriormente de otras funciones, el registro modificado puede enviarse a la otra computadora para que los archivos en el directorio remoto también puedan ser borrados.



### ❖ existe\_registro

```
bool existe_registro(const char *nombre_archivo){ ...
}
```

Esta función recibe como parámetro el nombre de un archivo que funge como registro binario, y su función es únicamente verificar si el archivo indicado con el nombre del parámetro existe dentro del directorio en el que se encuentra ejecutándose la instancia del programa.

### ❖ borrar\_archivos\_de\_directorio

```
void borrar_archivos_de_directorio(const char *ruta_directorio, const char *registro_archivos_por_borrar){ ...
}
```

Esta función recibe como parámetros la ruta de un directorio y el nombre de un archivo binario que funge como registro, y su función consiste en leer cada nombre de archivo grabado en el registro y borrarlo del directorio especificado en la ruta.

En el contexto del proyecto, el registro que esta función lee, es el registro modificado por la función “borrar\_archivos\_existentes\_de\_registro” y que fué enviado por el otro extremo de la comunicación, así la instancia local del programa sabrá que archivos fueron borrados en la instancia remota y por ende cuáles archivos debe borrar de su propio directorio local.

### ❖ **archivo\_en\_directorio**

```
bool archivo_en_directorio(const char *ruta_directorio, const char *nombre_archivo){ ...
}
```

Esta función recibe como parámetros la ruta de un directorio y el nombre de un archivo y su única función consiste en verificar si dicho archivo especificado existe dentro de la ruta del directorio especificada.

### ❖ **comparar\_informacion\_archivos**

```
int comparar_informacion_archivos(const void *archivo_A, const void *archivo_B){ ...
}
```

Esta función recibe como parámetros el nombre de 2 archivos diferentes y su única función es comparar si ambos archivos son iguales o no.

### ❖ **enviar\_archivo**

```
void enviar_archivo(int socket, const char *nombre_archivo){ ...
}
```

Esta función recibe como parámetros un socket de comunicación y el nombre de un archivo. Su función es enviar dicho archivo especificado a través del socket suministrado hacia la computadora remota con la que se debe mantener comunicación. Básicamente con esta función se envían los archivos del usuario de un extremo a otro de la comunicación, así como los registros de manejo interno que utiliza el programa en diferentes momentos y situaciones.

### ❖ recibir\_archivo

```
void recibir_archivo(int socket, const char *nombre_archivo){ ...
}
```

Esta función recibe como parámetros un socket de comunicación y el nombre de un archivo. Su función es recibir dicho archivo especificado a través del socket suministrado por parte de la computadora remota con la que se debe mantener comunicación. Básicamente con esta función se reciben los archivos del usuario de un extremo a otro de la comunicación, así como los registros de manejo interno que utiliza el programa en diferentes momentos y situaciones.

### ❖ sincronizar\_crear\_actualizar

```
void sincronizar_crear_actualizar(int socket, const char *directorio_local, int es_servidor){ ...
}
```

Esta función recibe como parámetros un socket de comunicación, una ruta de un directorio y un indicador para saber si la función se está ejecutando en el lado del servidor o del cliente.

Esta función es la responsable de sincronizar los eventos de creación o modificación de archivos en los directorios locales y remotos, garantizando que si un archivo es agregado por parte del usuario en alguno de los dos extremos, dicho archivo aparecerá también en el directorio que no lo tenga.

En cuanto a la modificación de archivos y manejo de conflictos, la función garantiza que si un archivo sufre modificaciones por parte del usuario, dichas modificaciones también serán realizadas en el archivo que no las tenga. En caso de que ambos archivos sufran modificaciones

diferentes, se optó por la opción de la permanencia del archivo que tenga la marca de última modificación más reciente.

### ❖ **sincronizar\_borrar**

```
void sincronizar_borrar(const char *ruta_directorio_local, int socket_local, int socket_remoto, int modo){...
```

Esta función recibe como parámetros la ruta de un directorio, los sockets de comunicación local y remoto y un indicador de modo que le hace saber a la función si se está ejecutando desde el lado del servidor o del cliente. Esta función es la responsable de sincronizar el evento de eliminación de archivos en los directorios locales y remotos entre ejecuciones del programa, garantizando que si un archivo es eliminado por parte del usuario en alguno de los dos extremos, dicho archivo se eliminará también en el directorio que aún lo tenga.

En el contexto del programa, esta función se ejecuta únicamente si el “registro.bin” existe dentro del directorio en el cual la instancia del programa se está ejecutando, sí es así, el programa verificará cuáles archivos aún existen dentro del directorio, borrando su información de “registro.bin” dejando en él únicamente la información de los archivos borrados. Este registro modificado será enviado al otro extremo de la comunicación en donde la misma función de la otra instancia en ejecución lo leerá.

Al ir leyendo uno a uno los nombres del registro recibido, los archivos presentes en el directorio que coincidan con los del registro recibido serán borrados, de esta manera se asegura que los archivos que fueron borrados en un directorio local también son borrados del directorio remoto.

## Programa principal.

```
// PROGRAMA PRINCIPAL.  
  
int main(int argc, char *argv[]){ ...  
}
```

En esta sección es donde reside la función “main”, la cual contiene todo lo referente a los códigos del servidor y del cliente, las construcciones de los sockets y las configuraciones necesarias para las comunicaciones, así como las llamadas y manejo de las diferentes funciones de sincronización, los manejos de errores y las terminaciones necesarias de las diferentes instancias del programa.

## Manejo de sockets.

La arquitectura de comunicación subyacente en este proyecto se apoya en la versatilidad y eficiencia de la biblioteca de sockets. En este contexto, el servidor se configura para esperar conexiones en el puerto 8889, actuando como el punto central para la sincronización de archivos. Por otro lado, el cliente se conecta a la dirección IP suministrada como parámetro, estableciendo así una conexión directa con el servidor. Este enfoque permite una comunicación directa y efectiva entre las dos máquinas, facilitando un intercambio rápido y seguro de información sobre archivos.

El proceso de sincronización en sí se lleva a cabo mediante la transmisión de datos específicos sobre cada archivo entre el servidor y el cliente. Esta transmisión de información abarca elementos cruciales

como el nombre del archivo, su tamaño, la marca temporal de la última modificación y el indicador de presencia en el directorio. La utilización de la biblioteca de sockets y este enfoque de transmisión selectiva garantiza una sincronización precisa, permitiendo que ambas máquinas mantengan un entendimiento claro y actualizado del estado de los archivos involucrados. Este enfoque modular y eficaz constituye la columna vertebral de la funcionalidad de sincronización del proyecto, ofreciendo una solución robusta para la transferencia bidireccional de archivos entre máquinas.

## **Manejo de solicitudes.**

El manejo de solicitudes en el proyecto se lleva a cabo principalmente en las funciones “sincronizar\_crear\_actualizar” y “sincronizar\_borrar”. Estas funciones se encargan de coordinar la comunicación entre el cliente y el servidor para lograr la sincronización bidireccional de archivos.

### **Funcionamiento:**

#### **1. Servidor:**

- Envía al cliente la lista de archivos locales, incluyendo información detallada.
- Recibe del cliente la lista de archivos remotos.
- Determina las operaciones necesarias para mantener la sincronización, como la creación, actualización o eliminación de archivos.
- Ejecuta las operaciones necesarias.

## 2. Cliente:

- Recibe del servidor la lista de archivos remotos.
- Envía al servidor la lista de archivos locales.
- Analiza las diferencias entre las listas para determinar las operaciones requeridas.
- Realiza las operaciones necesarias.

## Rutina de comparación.

Las funciones responsables de comparar y sincronizar archivos entre el directorio local y el directorio remoto son principalmente las funciones “sincronizar\_crear\_actualizar” y “sincronizar\_borrar”. Estas funciones realizan la comparación de archivos entre el servidor y el cliente mediante diferentes mecanismos dependiendo de la situación, decidiendo si un archivo debe enviarse, recibirse o si debe ser eliminado en función de la información de modificación y existencia de archivos en ambos extremos.

Algunos puntos que se deben tener en cuenta en este tipo de rutinas son los siguientes:

- **sincronizar\_crear\_actualizar:** Esta función se encarga de comparar la información de archivos entre el servidor y el cliente y decidir si se deben enviar o recibir archivos en función de la fecha de modificación.
- **sincronizar\_borrar:** Esta función sincroniza los directorios tomando en cuenta archivos borrados entre ejecuciones. Compara los registros de archivos locales y remotos y decide qué archivos

deben ser eliminados en función de la existencia en el directorio local o remoto.

- **Funciones de comparación y manipulación de archivos:**  
Además, hay otras funciones que comparan, envían y reciben archivos específicos (“enviar\_archivo”, “recibir\_archivo”, etc.) que también se encuentran involucradas en todos estos procesos anteriores.

## **Descripción de protocolos y formatos.**

El proyecto de sincronización de archivos implementa un protocolo simple de comunicación entre el servidor y el cliente para lograr una sincronización efectiva de los directorios locales. Este protocolo define cómo se intercambian la información sobre archivos y cómo se gestionan las actualizaciones y eliminaciones entre las dos instancias.

### **Protocolo de Comunicación.**

El protocolo sigue un enfoque de cliente-servidor, donde el servidor maneja la lógica central de sincronización y el cliente responde a las solicitudes del servidor. La comunicación entre el cliente y el servidor se establece a través de sockets TCP/IP.

### **Flujo de Comunicación.**

#### **1. Establecimiento de Conexión:**

- El servidor espera y acepta conexiones entrantes en un puerto específico.
- El cliente se conecta al servidor utilizando la dirección IP del servidor y el puerto asignado.



## **2. Intercambio de Información de Archivos:**

- El servidor y el cliente intercambian información sobre los archivos presentes en sus respectivos directorios locales.
- Esta información incluye el nombre del archivo, tamaño, y la marca de tiempo de la última modificación.

## **3. Sincronización de Archivos:**

- Ambas partes comparan la información recibida y determinan la acción a tomar para cada archivo (enviar, recibir o mantener).
- La sincronización se realiza enviando o recibiendo archivos según sea necesario.

## **4. Manejo de Archivos Eliminados:**

- Se implementa un mecanismo para manejar archivos eliminados entre ejecuciones.
- Un archivo de registro binario ("registro.bin") se utiliza para realizar un seguimiento de los archivos presentes en el directorio.

## **Formato de Mensajes.**

Los mensajes intercambiados entre el cliente y el servidor siguen un formato estructurado para garantizar una interpretación precisa de la información.

### **1. Información de Archivo:**

- Se utiliza una estructura llamada "struct Archivo" para representar la información de un archivo.
- Esta estructura incluye:
  - **nombre\_archivo:** Nombre del archivo.
  - **tamano:** Tamaño del archivo en bytes.

- **ultima\_modificacion:** Marca de tiempo de la última modificación.
- **presente\_en\_directorio:** Marca para indicar si el archivo está presente en el directorio (1) o ha sido eliminado (0).

## **2. Solicitud de Sincronización:**

- Los mensajes de solicitud y respuesta contienen información sobre el número total de archivos y la información detallada de cada archivo.

### **Ejemplo de Uso:**

- **Inicio del Servidor:**

`./a.out <ruta_del_directorio_local>`

- **Inicio del Cliente:**

`./a.out <ruta_del_directorio_local> <IP de la computadora remota>`

## **Análisis de resultados de pruebas.**

### **Ambiente inicial de trabajo.**

Para las pruebas de ejecución de funcionamiento del programa se decidió trabajar con archivos de diferentes tipos para una fácil manipulación y de diferentes pesos para corroborar el correcto funcionamiento del programa, por lo que se construyó el siguiente ambiente de trabajo:

1. El proyecto entero está contenido dentro un directorio de nombre “Proyecto”
2. Dentro del directorio “Proyecto” se encuentran 3 componentes que se utilizaran para las pruebas del proyecto:

- **Directorio\_Local**

Este directorio es dónde se ejecutará la instancia del programa que actuará como servidor.

Para realizar las pruebas, de inicio este directorio cuenta con los siguientes archivos en su interior:

Archivo	Peso
1- Metro 2033.pdf	5,64 MB
2- Metro 2034.pdf	3,63 MB
3- Metro 2035.pdf	7,21 MB
3.5- Metro 2035 - Fin del Camino.pdf	1,13 MB
4- Metro 2033 - Hacia la Luz.pdf	1,19 MB
5- Metro 2033 - San Petersburgo.pdf	2,04 MB
a.out	21,6 KB
Metro 2033 - Britania (Traducido).pdf	2,98 MB
Z_1.txt	41 B

El archivo “Z\_1.txt” está destinado para probar las funciones de actualización de cambios y manejo de conflictos. Para este momento su contenido es el siguiente:

```
≡ Z_1.txt  ×
Directorio_Local > ≡ Z_1.txt
1  Este es el primer mensaje del archivo Z_1
```

- **Directorio\_Remoto**

Este directorio es dónde se ejecutará la instancia del programa que actuará como cliente.

Para realizar las pruebas, de inicio este directorio cuenta con los siguientes archivos en su interior:

Archivo	Peso
a.out	21,6 KB
Metro 2033 01 - Wo Die Welt Endet.pdf	303 MB
Metro 2033 02 - Maske Der Dunkelheit.pdf	211 MB
Metro 2033 03 - Augen Der Finsternis.pdf	142 MB
Metro 2033 04 - Tanz Mit Dem Tod.pdf	89,7 MB
Metro Last Light - The Gospel According to Artyom.pdf	6,63 MB
Metro Universe - Britannia Prologue.pdf	28,9 MB
Z_2.txt	41 B

El archivo “Z\_2.txt” está destinado para probar las funciones de actualización de cambios y manejo de conflictos. Para este momento su contenido es el siguiente:

```
≡ Z_2.txt  X
Directorio_Remoto > ≡ Z_2.txt
1  Este es el primer mensaje del archivo Z_2
```

- Proyecto\_3.c

Este es el archivo donde se edita el código del programa.

3. Debido a todo lo anteriormente explicado, la configuración del ambiente de trabajo queda de la siguiente manera:

```
▼ PROYECTO
  ▼ Directorio_Local
    1- Metro 2033.pdf
    2- Metro 2034.pdf
    3- Metro 2035.pdf
    3.5- Metro 2035 - Fin del Camino.pdf
    4- Metro 2033 - Hacia la Luz.pdf
    5- Metro 2033 - San Petersburgo.pdf
    ≡ a.out
    Metro 2033 - Britania (Traducido).pdf
    ≡ Z_1.txt
  ▼ Directorio_Remoto
    ≡ a.out
    1- Metro 2033 01 - Wo Die Welt Endet.pdf
    2- Metro 2033 02 - Maske Der Dunkelheit.pdf
    3- Metro 2033 03 - Augen Der Finsternis.pdf
    4- Metro 2033 04 - Tanz Mit Dem Tod.pdf
    Metro Last Light - The Gospel According to Artyom.pdf
    Metro Universe - Britannia Prologue.pdf
    ≡ Z_2.txt
  Proyecto_3.c
```

## Ejecución de la primera corrida.

Una vez construido el ambiente de trabajo inicial, se procede a ejecutar las instancias del programa presentes en cada directorio.

Aquí se presentan las ejecuciones en consola:

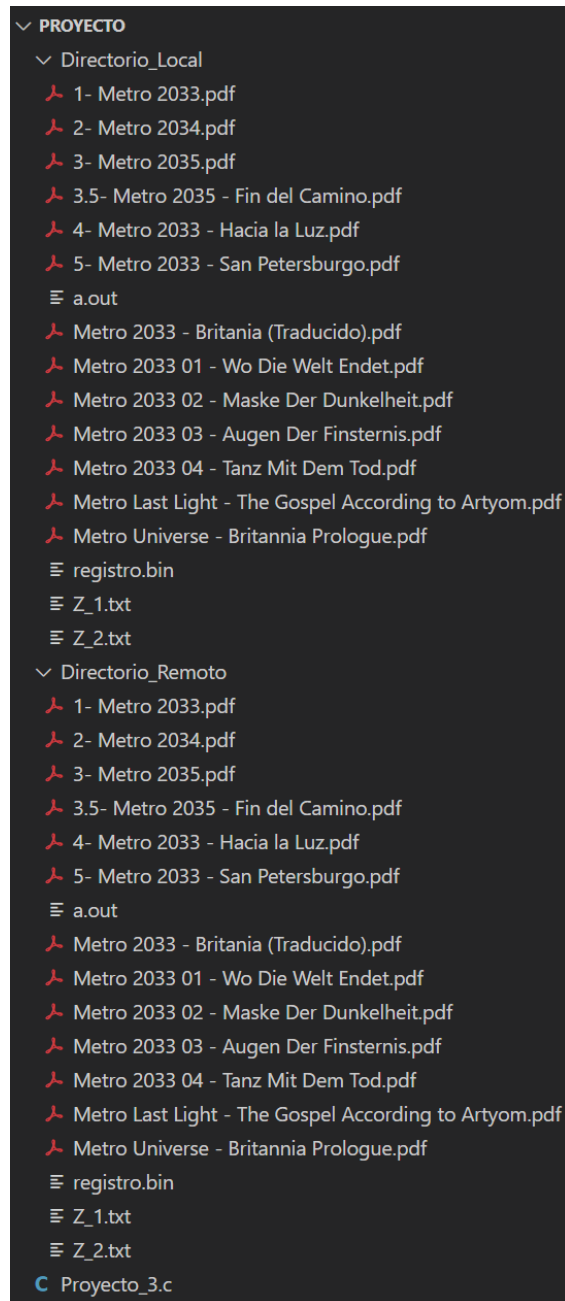
- **Instancia del servidor:**

```
(diego_dd@DiegoMass)-[/mnt/c/Users/diemq/Desktop/Proyecto/Directorio_Local]
$ ./a.out /mnt/c/Users/diemq/Desktop/Proyecto/Directorio_Local
Iniciando servidor...
Configurando dirección del servidor...
Esperando conexiones en el puerto 8889...
Conexión aceptada de 127.0.0.1:42910
Sincronizando directorios...
Sincronización exitosa...
La instancia del servidor ha finalizado.
```

- **Instancia del cliente:**

```
(diego_dd@DiegoMass)-[/mnt/c/Users/diemq/Desktop/Proyecto/Directorio_Remoto]
$ ./a.out /mnt/c/Users/diemq/Desktop/Proyecto/Directorio_Remoto 127.0.0.1
Iniciando cliente...
Configurando dirección del servidor para el cliente...
Conexión exitosa al servidor. IP: 127.0.0.1, Puerto: 8889
Sincronizando directorios...
Sincronización exitosa...
La instancia del cliente ha finalizado.
```

Los resultados de la sincronización ejecutada han sido los siguientes:



Después de esta ejecución se procedió a abrir los directorios y corroborar que los pesos de los archivos sincronizados fueran los correctos con respecto a los originales. También se procedió a abrir cada archivo para verificar que todos se hubieran reconstruido correctamente y no se encontraran corruptos.

Estas verificaciones resultaron todas correctas.

## Movimientos post-corrida.

Una vez corroborado que la sincronización inicial de archivos entre los directorios ha sido un éxito, debemos proceder a probar las características de agregación, modificación y eliminación de archivos además del manejo de conflictos, por lo que se procederá de la siguiente manera:

### 1. En “Directorio\_Local”:

- Se agregó el siguiente archivo:

Archivo	Peso
Metro 2033 - Caminos Paralelos.pdf	713 KB

- Se eliminaron los siguientes archivos:

Archivo	Peso
2- Metro 2034.pdf	3,63 MB
3- Metro 2035.pdf	7,21 MB
3.5- Metro 2035 - Fin del Camino.pdf	1,13 MB
Metro 2033 - Britania (Traducido).pdf	2,98 MB

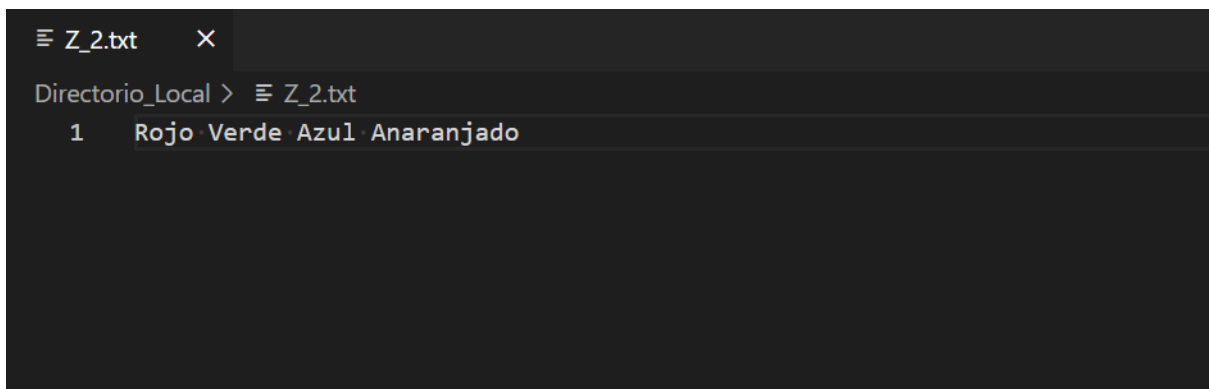
- Se editó el contenido del contenido del archivo “Z\_1” de la siguiente manera:

```
≡ Z_1.txt  X
Directorio_Local > ≡ Z_1.txt
1 Este es el primer mensaje del archivo Z_1
2
3 Este es el segundo mensaje del archivo Z_1
```



Esta modificación tiene la finalidad de probar la funcionalidad de modificaciones de archivos, por lo que se realizó únicamente en el archivo presente en “Directorio\_Local”.

- Se editó el contenido del contenido del archivo “Z\_2” de la siguiente manera:



```
≡ Z_2.txt  X
Directorio_Local > ≡ Z_2.txt
1  Rojo Verde Azul Anaranjado
```

Esta modificación tiene la finalidad de probar la funcionalidad de manejo de conflictos, por lo que esta modificación será la que tenga la marca de última modificación más vieja.

## 2. En “Directorio\_Remoto”:

- Se agregó el siguiente archivo:

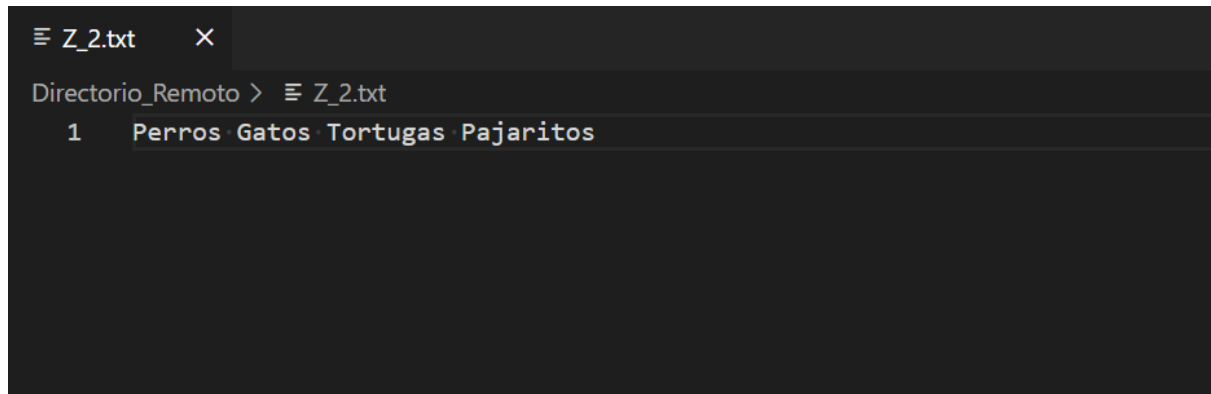
Archivo	Peso
The Outpost - America - A Metro 2033 Universe Graphic Novel - Vol. 1 (Scanned).pdf	8,05 MB

- Se eliminaron los siguientes archivos:

Archivo	Peso
Metro 2033 01 - Wo Die Welt Endet.pdf	303 MB

Metro 2033 02 - Maske Der Dunkelheit.pdf	211 MB
Metro Last Light - The Gospel According to Artyom.pdf	6,63 MB
Metro Universe - Britannia Prologue.pdf	28,9 MB

- Se editó el contenido del contenido del archivo “Z\_2” de la siguiente manera:



```

≡ Z_2.txt  ×
Directorio_Remoto > ≡ Z_2.txt
1  Perros Gatos Tortugas Pajaritos

```

Esta modificación tiene la finalidad de probar la funcionalidad de manejo de conflictos, por lo que esta modificación será la que tenga la marca de última modificación más reciente.

Una vez realizados los cambios manuales entre ejecuciones del programa, el ambiente de trabajo queda de la siguiente manera:

✓ PROYECTO

✓ Directorio\_Local

- 🔴 1- Metro 2033.pdf
- 🔴 4- Metro 2033 - Hacia la Luz.pdf
- 🔴 5- Metro 2033 - San Petersburgo.pdf
- ≡ a.out
- 🔴 Metro 2033 - Caminos Paralelos.pdf
- 🔴 Metro 2033 01 - Wo Die Welt Endet.pdf
- 🔴 Metro 2033 02 - Maske Der Dunkelheit.pdf
- 🔴 Metro 2033 03 - Augen Der Finsternis.pdf
- 🔴 Metro 2033 04 - Tanz Mit Dem Tod.pdf
- 🔴 Metro Last Light - The Gospel According to Artyom.pdf
- 🔴 Metro Universe - Britannia Prologue.pdf
- ≡ registro.bin
- ≡ Z\_1.txt
- ≡ Z\_2.txt

✓ Directorio\_Remoto

- 🔴 1- Metro 2033.pdf
- 🔴 2- Metro 2034.pdf
- 🔴 3- Metro 2035.pdf
- 🔴 3.5- Metro 2035 - Fin del Camino.pdf
- 🔴 4- Metro 2033 - Hacia la Luz.pdf
- 🔴 5- Metro 2033 - San Petersburgo.pdf
- ≡ a.out
- 🔴 Metro 2033 - Britania (Traducido).pdf
- 🔴 Metro 2033 03 - Augen Der Finsternis.pdf
- 🔴 Metro 2033 04 - Tanz Mit Dem Tod.pdf
- ≡ registro.bin
- 🔴 The Outpost - America - A Metro 2033 Universe Graphic Novel - Vol. 1 (Scanned).pdf
- ≡ Z\_1.txt
- ≡ Z\_2.txt

🔵 Proyecto\_3.c

## Ejecución de la segunda corrida.

Una vez modificado el ambiente de trabajo, se procede a ejecutar las instancias del programa presentes en cada directorio por segunda vez.

Aquí se presentan las ejecuciones en consola:

- **Instancia del servidor:**

```
(diego_dd@DiegoMass)-[/mnt/c/Users/diemq/Desktop/Proyecto/Directorio_Local]
$ ./a.out /mnt/c/Users/diemq/Desktop/Proyecto/Directorio_Local
Iniciando servidor...
Configurando dirección del servidor...
Esperando conexiones en el puerto 8889...
Conexión aceptada de 127.0.0.1:40366
Sincronizando directorios...
Sincronización exitosa...
La instancia del servidor ha finalizado.
```

- **Instancia del cliente:**

```
(diego_dd@DiegoMass)-[/mnt/c/Users/diemq/Desktop/Proyecto/Directorio_Remoto]
$ ./a.out /mnt/c/Users/diemq/Desktop/Proyecto/Directorio_Remoto 127.0.0.1
Iniciando cliente...
Configurando dirección del servidor para el cliente...
Conexión exitosa al servidor. IP: 127.0.0.1, Puerto: 8889
Sincronizando directorios...
Sincronización exitosa...
La instancia del cliente ha finalizado.
```

Los resultados de la sincronización ejecutada han sido los siguientes:

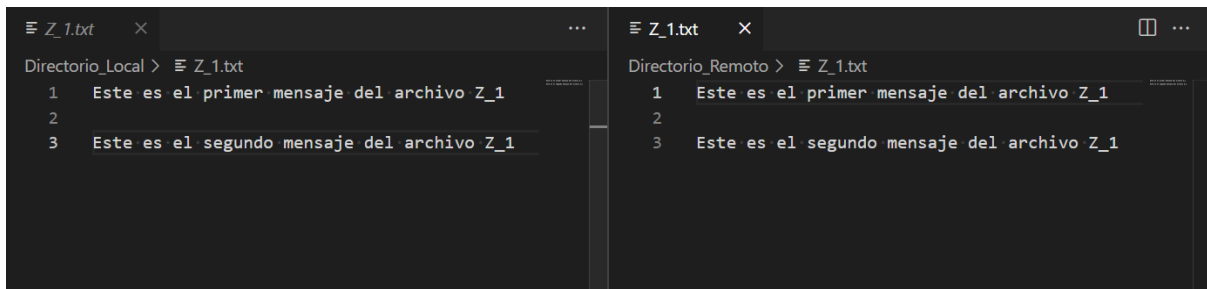


Analizando el ambiente de trabajo resultante de la segunda ejecución podemos notar lo siguiente:

- El archivo agregado en “Directorio\_Local” ha sido copiado exitosamente en “Directorio\_Remoto”.
- El archivo agregado en “Directorio\_Remoto” ha sido copiado exitosamente en “Directorio\_Local”.

- Los archivos borrados en “Directorio\_Local” fueron borrados exitosamente de “Directorio\_Remoto”.
- Los archivos borrados en “Directorio\_Remoto” fueron borrados exitosamente de “Directorio\_Local”.

Ahora procederemos a abrir los archivos “Z\_1” de ambos directorios:

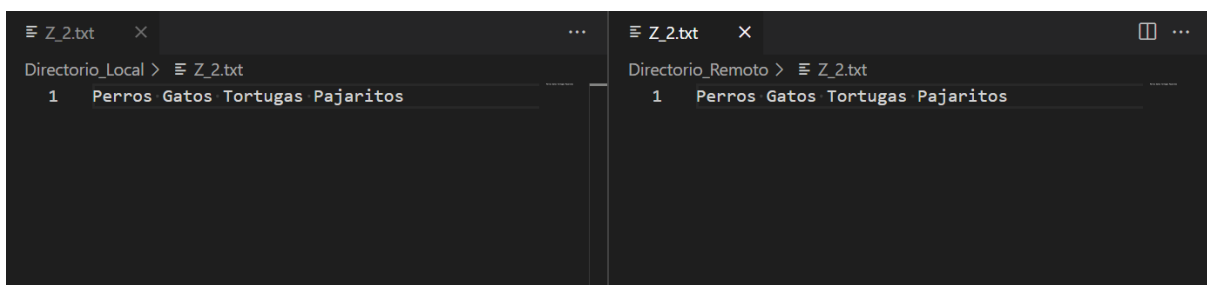


```
Directorio_Local > Z_1.txt
1 Este es el primer mensaje del archivo Z_1
2
3 Este es el segundo mensaje del archivo Z_1

Directorio_Remoto > Z_1.txt
1 Este es el primer mensaje del archivo Z_1
2
3 Este es el segundo mensaje del archivo Z_1
```

Como podemos observar, la modificación al archivo realizada en “Directorio\_Local” se realizó exitosamente en el archivo de “Directorio\_Remoto”.

Ahora procederemos a abrir los archivos “Z\_2” de ambos directorios:



```
Directorio_Local > Z_2.txt
1 Perros Gatos Tortugas Pajaritos

Directorio_Remoto > Z_2.txt
1 Perros Gatos Tortugas Pajaritos
```

Como podemos observar, la modificación que prevaleció fue la realizada en el archivo de “Directorio\_Remoto”, ya que fue la que tenía la marca de última modificación más reciente de las dos.

## **Conclusiones sobre rendimiento y correctitud.**

En conclusión, el proyecto de sincronización de archivos sin conexión ha demostrado ser una solución robusta y eficiente para abordar los desafíos asociados con la transferencia de datos entre máquinas en entornos donde la conectividad a Internet no es constante. La implementación de sockets TCP/IP ha facilitado una comunicación directa y segura entre el servidor y el cliente, permitiendo un intercambio rápido de información sobre archivos.

El enfoque bidireccional del programa, que actúa tanto como servidor como cliente, añade versatilidad y adaptabilidad a diferentes escenarios. Además, la capacidad de retomar la sincronización de forma automática después de episodios de desconexión garantiza la continuidad operativa, incluso en entornos donde la estabilidad de la red no es un hecho garantizado.

La gestión de conflictos, al conservar la versión del archivo con la marca de última modificación más reciente, proporciona coherencia en la sincronización cuando se realizan modificaciones en ambas máquinas. La estructura de datos "Archivo" ofrece una representación completa y detallada de cada archivo, facilitando la evaluación precisa de cambios y actualizaciones.

El protocolo de comunicación implementado, basado en un enfoque cliente-servidor con intercambio estructurado de información, ha demostrado ser efectivo en la coordinación de la sincronización bidireccional. Las pruebas realizadas en diferentes escenarios, incluyendo la adición, eliminación y modificación de archivos, han

validado la capacidad del programa para mantener la coherencia entre directorios locales y remotos.

La herramienta desarrollada se presenta como una opción esencial para la gestión eficiente de archivos en entornos sin conexión a Internet, destacando por su rendimiento sólido, capacidad de adaptación y manejo efectivo de situaciones de conectividad intermitente.