



Escuela de Ingeniería en Computación

Principio de Sistemas Operativos

Programa sincronizador de archivos en C

Estudiantes:

José Mario Naranjo Leiva

Gabriel Ramírez Ramírez

Carnet:

2013034348

2010

Profesor:

Armando Arce Orozco

Primer Semestre, 2017

Tabla de contenido

Tabla de contenido	2
Tabla de ilustraciones	3
1. Introducción	4
2. Descripción del problema	4
2.1. Programa Sincronizador	4
3. Definición de estructuras de datos	5
4. Descripción detallada	7
4.1. Manejo de sockets	7
4.1.1. Configuración en el cliente	8
4.1.2. Configuración en el servidor	9
4.2. Manejo de solicitudes	9
4.2.1. Estructuras de solicitud y sincronización	9
4.2.2. Secuencia de solicitudes	10
4.3. Rutina de comparación	13
4.3.1. Identificación de archivos	13
4.3.2. Comparación	14
4.3.3. Almacenamiento de datos del directorio	15
4.4. Mecanismo de bloqueo	15
5. Descripción del protocolos y formatos	16
5.1. Formatos	16
6. Análisis de resultados de pruebas	18
6.1. Definición de pruebas	18
6.2. Resultados de las pruebas	20
7. Conclusiones sobre rendimiento y funcionamiento	24
8. Anexos	25
8.1. Estructura del proyecto	25
8.2. URL al repositorio	25

Tabla de ilustraciones

Ilustración 1: Estructura para almacenar los datos del archivo	5
Ilustración 2: Estructura para un arreglo dinámico de archivos.....	5
Ilustración 3 Vista general de la estructura.....	6
Ilustración 4 - Diagrama comunicación de sockets.....	7
Ilustración 5 - Diagrama de secuencia para las solicitudes (directorio vacío en el server)	10
Ilustración 6 - Definición de la función para solicitudes de recibimiento de archivos.....	11
Ilustración 7 - Definición de la función para solicitudes de envío de archivos.....	11
Ilustración 8 - Diagrama de secuencia para las solicitudes (hay cambios en el cliente y el server ya tiene otros archivos).....	12
Ilustración 9 - Funciones generar solicitudes de cambio en archivos.....	12
Ilustración 10 - Función para manejar las solicitudes de cambios.....	13
Ilustración 11 - Definición de la función para el escaneo del directorio	13
Ilustración 12 - Definición de función de comparación.....	14
Ilustración 13 - Definición de función de comparación (modificados).....	14
Ilustración 14 - Vista general del protocolo de conexión TCP.....	16
Ilustración 15 - Formato del paquete de sincronización del archivo	16
Ilustración 16 - Formato del paquete de mensajes de sincronización.....	17

1. Introducción

En el presente documento, se mencionen y explican las decisiones de diseño, mecanismos de programación y etapas de desarrollo de la aplicación sincronizadora de archivos, implementada en el lenguaje de programación C para ambientes UNIX.

En este, encontraran la descripción detallada del problema y la definición exitosa de la solución implementada. El desarrollo de este proyecto corresponde para el curso de Principios de Sistemas Operativos y en este se resumen conceptos importantes sobre manejo de archivos y distribución de archivos.

Como parte importante del proyecto, se incluye un listado de pruebas y sus resultados que fueron aplicados al sistema durante la última etapa de desarrollo. Se presenta una tabla con los resultados exitosos de cada prueba.

2. 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).

2.1. Programa Sincronizador

El programa sincronizador se debe ejecutar en dos máquinas al mismo tiempo. En la primera 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 primera 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.
- Tome en cuenta que los relojes de las máquinas involucradas pueden estar de sincronizados. Para determinar cuáles archivos son más recientes se debe realizar un cálculo de diferencias entre las horas de ambos relojes.
- 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.

3. Definición de estructuras de datos

Para la implementación adecuada de este programa sincronizador, se implementaron las siguientes estructuras de datos.

```
typedef struct
{
    char name[1000];
    char path[2000];
    int size;
    time_t modification_time;
} file_data;
```

Ilustración 1: Estructura para almacenar los datos del archivo

```
typedef struct
{
    file_data *array;
    size_t used;
    size_t size;
} Array;
```

Ilustración 2: Estructura para un arreglo dinámico de archivos

- **Lista**

Para manejar los archivos contenidos en el directorio, se implementó una lista dinámica enlazada, para almacenar la estructura *file_data* (ver Ilustración 1) que contiene los datos relevantes de un archivo: nombre, carpeta, tamaño y fecha de modificación.

El nodo de la lista enlazada está definido en la estructura *Array* (ver Ilustración 2). Un diagrama general de esta estructura está en Ilustración 3.

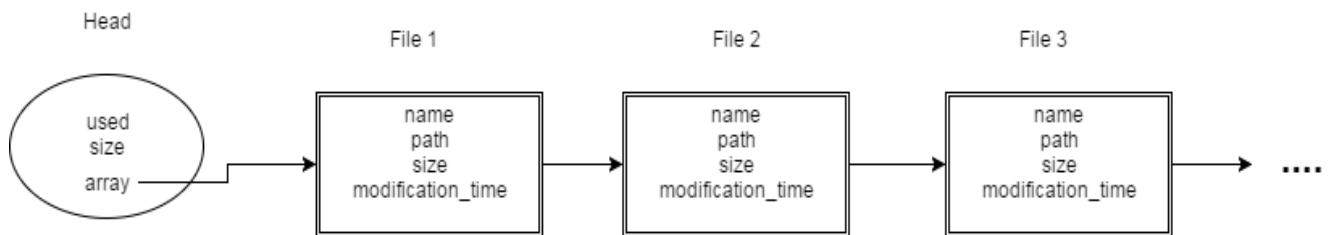


Ilustración 3 Vista general de la estructura

- **Estructuras de sincronización**

Las estructuras de sincronización son definidas para el uso adecuado de la comunicación de las computadoras y la sincronización de los archivos. Estas se ven con más detalle en la sección 5 (subsección 5.1) del presente documento.

4. Descripción detallada

A continuación, se muestra la explicación de los componentes principales del servidor y el cliente del sistema sincronizador.

4.1. Manejo de sockets

Los sockets son parte importante para la implementación del proyecto. Los sockets Berkeley son usados para la comunicación de ambos programas.

Estos sockets son un API para el uso de sockets de internet y de dominio UNIX, usados para la intercomunicación de procesos. Por sus orígenes, estos usan el estándar de componentes POSIX.

Para la implementación del proyecto, se decidió el uso de dos sockets principales: el socket cliente y el socket servidor; entre ambos se realiza la comunicación de ambas máquinas.

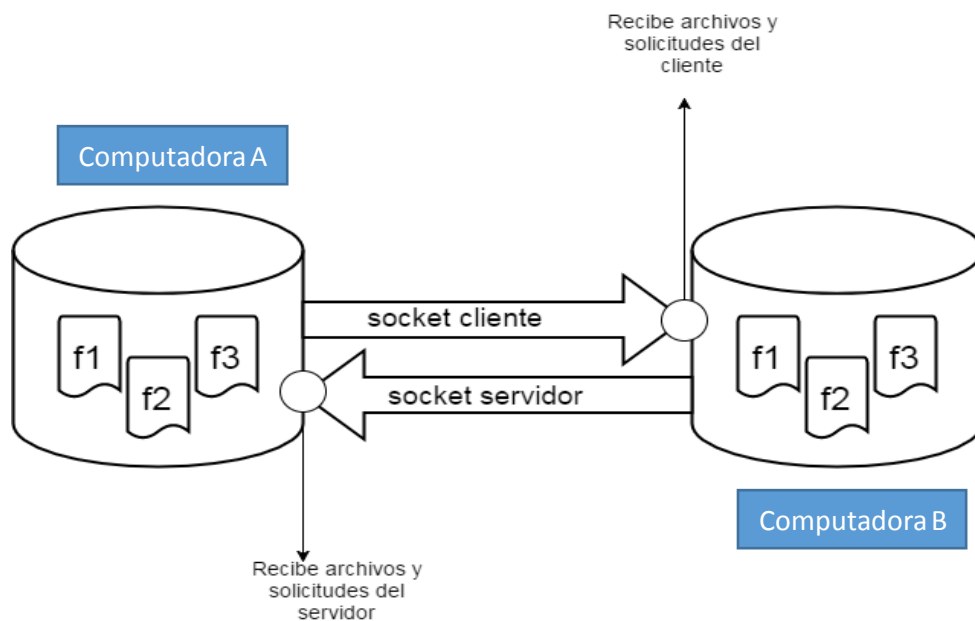


Ilustración 4 - Diagrama comunicación de sockets


Los programas ejecutándose en la computadora A y B tienen instancias de los sockets servidor y cliente para la comunicación entre ambas computadoras. Claramente, la computadora que esté sirviendo como cliente usará el socket cliente y lo mismo para el servidor. La vista general de los programas con los sockets se puede ver en Ilustración 4.

Los sockets son manejados durante la ejecución por medio de variables (variables identificador del socket). En el rol de servidor, el identificador de socket servidor es usado para las operaciones de envío de paquetes al cliente, y el socket del cliente, se usará para la lectura o recibimiento de paquetes del cliente. En el rol cliente, ocurre el modo contrario de los casos.


La configuración de los sockets, tanto en el servidor como el cliente, es llevada a cabo por las siguientes funciones implementadas:¹

4.1.1. Configuración en el cliente

- Inicio de la conexión hacia el servidor. Manejo de errores por conexiones fallidas.

```
/**
 * Realiza la inicialización del lado del cliente.
 * @param : directory : nombre del directorio que se desea sincronizar
 * @return : código de éxito en la comunicación
 */
int init_client(char *hostname, char *directory)
{
```

- Conexión a la dirección del servidor (Conectarse al servidor por la IP)

```
/**
 * Crea un socket e intenta conectarse al servidor por medio del hostname.
 * @Param : hostname : String - Nombre del host por conectarse
 * @Return : el identificador del socket, al cual se realizó la conexión.
 */
int connect_to_server(char *hostname)
{
```

Nota: Estas funciones están definidas en el archivo [client.c](#)

¹ Es válido recalcar que las conexiones de los sockets se realizan sobre el puerto 8889. Este puerto usa el Protocolo de Control de Transmisión.

4.1.2. Configuración en el servidor

- Prepara la conexión del servidor con el cliente. Control de errores de conexión.

```
/**
 * Realiza la inicialización del lado del servidor.
 * @param : directory : nombre del directorio que se desea sincronizar
 * @return : código de éxito en la comunicación
 */
int init_server(char *directory)
{
}
```

- Configuración del socket del servidor.

```
/**
 * Configura el socket del servidor. Crea y asocia el socket a un puerto.
 * @return: listenfd (descriptor del socket) donde se está escuchando al cliente
 */
int setup()
{
}
```

Nota: Estas funciones están definidas en el archivo server.c

4.2. Manejo de solicitudes

A continuación, se presentan las decisiones de diseño e implementación de las solicitudes.

Las comunicaciones entre el cliente y el servidor son llevadas a cabo de punto a punto. Cada mensaje es recibido y luego respondido, según el orden donde se realice las lecturas y escrituras sobre los sockets, en ambas partes.

4.2.1. Estructuras de solicitud y sincronización

El protocolo TCP usado permite el envío en secuencia de paquetes del origen hacia el destino. La estructura de estos paquetes fue definida por el equipo de desarrollo para mantener un formato adecuado para manejar las solicitudes y mensajes de sincronización.

El manejo de las solicitudes es llevado por medio de las siguientes estructuras:

```

struct sync_file_message
{
    // Datos del archivo
    char filename[1000];
    time_t mtime;
    int size;
    // Contenido del archivo
    unsigned char fileBuff[1024];
};

```

```

struct sync_message
{
    // Datos de archivo
    char message[1000];
    char name[1000];
    time_t mtime ;
    int size ;

    // Indicadores del evento
    int empty_directory ;
    int deleted_file ;
    int modified_file ;
    int added_file ;
};

```

Nota: Estas son analizadas con más detalle en la sección 5 del presente documento.

4.2.2. Secuencia de solicitudes

Para el sistema de sincronización pueden suceder dos casos especiales de solicitudes: **(1)** El servidor está vacío y solicita la transferencia completa de los ficheros del cliente. **(2)** El servidor ya ha sido sincronizado y hay solicitudes de sincronización del cliente hacia el servidor.

En cada caso, hay dos casos en común: el envío de **un apretón de manos** (para determinar una conexión exitosa entre ambas partes) y una **solicitud de finalización** (notificar el fin exitoso de la sincronización o algún fallo).

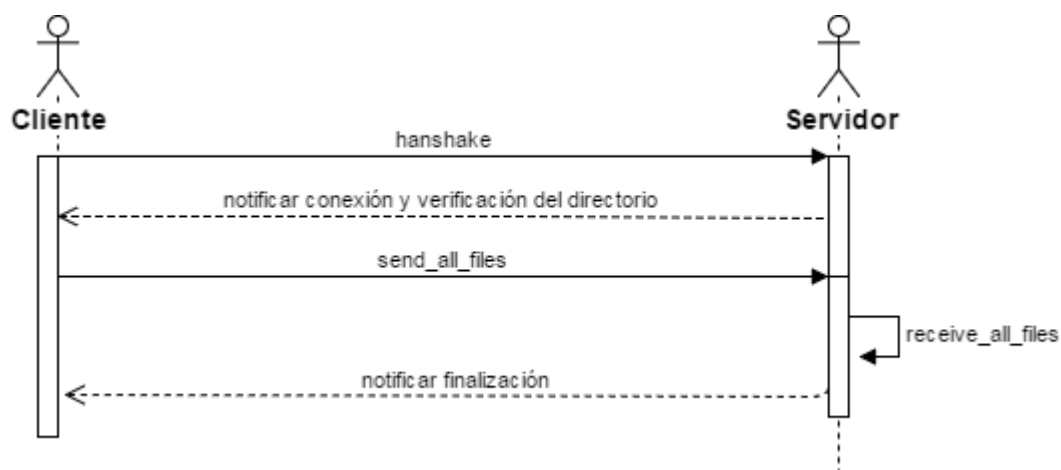


Ilustración 5 - Diagrama de secuencia para las solicitudes (directorio vacío en el server)

En Ilustración 5 se puede apreciar la secuencia de solicitudes y respuestas entre el cliente y el servidor, para el primer caso. En este diagrama resaltan las funciones `send_all_files` y `receive_all_files`. Estas funciones manejan las solicitudes de envío por parte del cliente y la aceptación del envío por parte del servidor.

La definición de las funciones mencionadas, en el código fuente, son las siguientes:²

```
/**
 * Esta función recibe un conjunto de archivos.
 * Se debe ejecutar en caso del directorio actual vacío y se debe recibir multiples archivos
 * @param : client_socket : socket con la conexión al cliente.
 */
void receive_all_files(int client_socket)
{
}
```

Ilustración 6 - Definición de la función para solicitudes de recibimiento de archivos

```
/**
 * Esta función envía un conjunto de archivos al servidor.
 * Se debe ejecutar en caso del directorio actual vacío y se debe transmitir multiples archivos
 * @param : socket : socket con la conexión al servidor.
 */
void send_all_files(int socket, char *directory)
{
}
```

Ilustración 7 - Definición de la función para solicitudes de envío de archivos.

² La función `receive_all_files` está implementada en el archivo `server.c`. Por su parte, la función `send_all_files` está implementada en el archivo `client.c`.

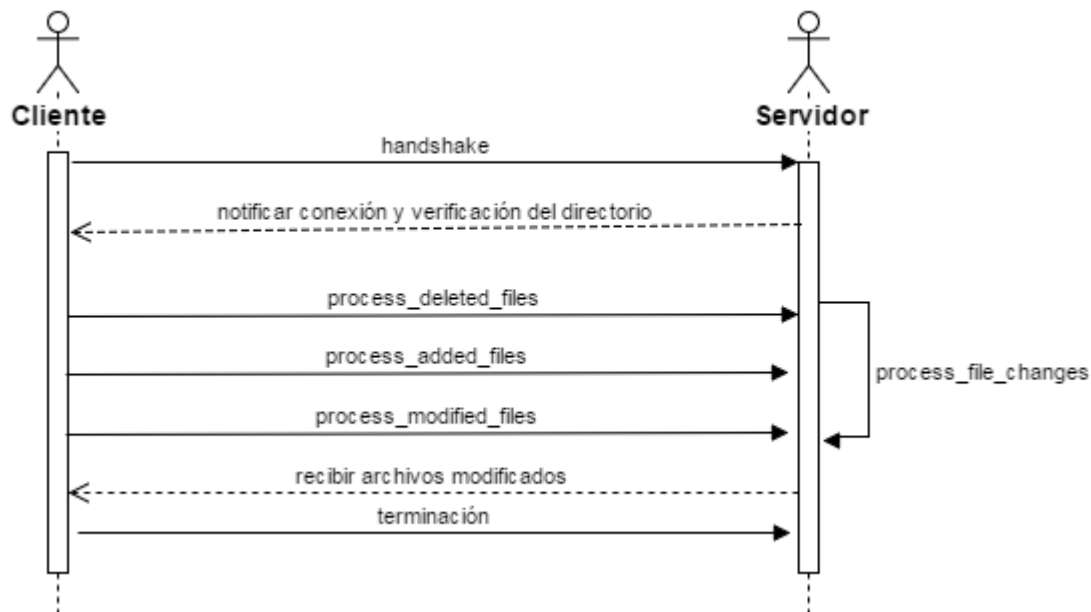


Ilustración 8 - Diagrama de secuencia para las solicitudes (hay cambios en el cliente y el server ya tiene otros archivos)

En el caso 2, el cliente no puede enviar ni el servidor recibir todos los archivos de forma inmediata. Se necesita el manejo de solicitudes para los posibles cambios: eliminación, agregado y modificación de archivos.

Para estas solicitudes de operaciones se hacen pasos de mensajes indicando la operación que se desea realizar. Estos indicadores están en la estructura **syn_message** mostrada anteriormente.

Estos indicadores permiten tomar decisiones sobre qué acción tomar y ejecutar las siguientes operaciones (mostradas en Ilustración 8).

```

// Procesar los archivos eliminados
void process_deleted_files(int socket, Array *deleted_files)
{
}
// Procesar los archivos agregados
void process_added_files(int socket, Array *added_files)
{
}
//Procesar los archivos modificados
void process_modified_files(int socket, Array *modified_files, char *directory)
{
}
  
```

Ilustración 9 - Funciones generar solicitudes de cambio en archivos

En el cliente existe una función *process_file_changes* encargada de procesar las distintas solicitudes del cliente por eliminar, agregar y/o procesar archivos modificados. Esta está implementada en el archivo *server.c*

```

/**
 * Esta función procesa las solicitudes de cambio obtenidas desde el cliente.
 * Se debe ejecutar en caso que haya que sincronizar un directorio NO vacío del servidor
 * @param : client_socket : socket con la conexión al cliente.
 * @param : directory : directorio que se desea procesar y sincronizar
 */
void process_file_changes(int client_socket, char *directory)
{
}

```

Ilustración 10 - Función para manejar las solicitudes de cambios

4.3. Rutina de comparación

A continuación, se explica el algoritmo de comparación implementado para determinar las diferencias en los archivos anteriores y actuales del directorio por sincronizar.

4.3.1. Identificación de archivos

Para la lectura de los archivos del directorio se usa la funcionalidad “*scandir*”, definida en el encabezado `#include <dirent.h>`. Esta permite escanear el directorio y cada archivo es almacenado en una estructura dinámica, para el procesamiento posterior de los archivos.

```

/**
 * Esta función escanea la lista de archivos obtenidas en scandir y las almacena
 * en la estructura <<file_data>>
 * @Param: files : puntero a un arreglo de archivos.
 * @Param: namelist : puntero al resultado de la función scandir.
 * @Param: n : cantidad de archivos por leer.
 * @Return: None
 */
void scanFilesFromDirectory(Array *files, struct dirent **namelist, int n)
{
}

```

Ilustración 11 - Definición de la función para el escaneo del directorio

Luego, se usa la función (ver Ilustración 11) para obtener los datos de los archivos del directorio y almacenarlos en una lista dinámica. Posteriormente, esta lista de archivos actuales es almacenada en un fichero en el directorio `<<.meta>>`, con el objetivo de realizar la comparación con los archivos locales actuales y los archivos de la corrida anterior.

4.3.2. Comparación

El algoritmo de comparación implementado (ver Ilustración 12) está basado en la diferencia de conjuntos; así, siendo el listado actual y el listado anterior los conjuntos a cuáles se les van comparar.

La comparación se realiza sobre el listado del directorio anterior contra el listado del directorio actual, para determinar los archivos eliminados y agregados. El algoritmo genera un listado con los ficheros agregados y otro listado para ficheros eliminados.

No obstante; existe una extensión del algoritmo propio (ver Ilustración 13) para determinar los archivos modificados, esto porque se necesitan realizar una comparación especial de fechas, que no se necesitan para archivos eliminados o agregados.

```
/**
 * Realiza la diferencia entre el listado de dos directorios.
 * Se lleva acabo la operación A - B sobre dos arreglos de archivos
 * @param: x : Array -> arreglo A
 * @param: lenx : tamaño del arreglo x
 * @param: y : Array -> arreglo B
 * @param: leny : tamaño del arreglo y
 * @param: res : Array -> arreglo para almacenar el resultado
 */
void diff(Array *x, int lenx, Array *y, int leny, Array *res)
{
}
```

Ilustración 12 - Definición de función de comparación

```
/**
 * Realiza la búsqueda de los archivos que ha sido modificados.
 * Compara el listado actual del directorio y el listado anterior
 */
void diffModified(Array *x, int lenx, Array *y, int leny, Array *res)
{
}
```

Ilustración 13 - Definición de función de comparación (modificados)

El algoritmo funciona de la siguiente manera:

- Siendo A = lista de archivos anterior y B = lista de archivos actuales, para un directorio dado.
- La función *diff* mencionada anteriormente, obtiene la diferencia de los conjuntos A y B.
- Cuando se invoca la función *diff* (A, B, C), ($A - B = C$), se obtiene en C los archivos eliminados en el directorio.
- Cuando se invoca la función *diff* (B, A, C), ($B - A = C$), se obtiene en C los archivos agregados al directorio.

4.3.3. Almacenamiento de datos del directorio

Para conservar un historial de archivos pasados, se decidió tener una carpeta `<<.meta>>`. En este se almacena un archivo *files_data.bin* con el listado de archivos actual del directorio; así posteriormente se puede recuperar los archivos y determinar qué cambios han existido.

4.4. Mecanismo de bloqueo

Durante su ejecución, el sistema sincronizador debe mantener completo del directorio y sus archivos; así ningún otro proceso en la máquina puede o no debe acceder a estos archivos durante la sincronización. Esto con el objetivo, de evitar daño en la transferencia de los datos.

Para esto, el diseño del sistema nos asegura la utilización de la librería *fcntl* para mantener el control de un archivo abierto mediante una llamada previa a *open*.

El enfoque fue el siguiente:

- Se crea un cerrojo. Este indica que el proceso actual está leyendo el archivo, por lo que ningún otro archivo debe tener acceso al archivo.
- El cerrojo de lectura tiene más importancia para no corromper datos del archivo que puedan estar siendo transferido.

5. Descripción del protocolos y formatos

A continuación, se explican el formato y protocolo definidos para la transferencia y recibimiento de solicitudes de archivos.

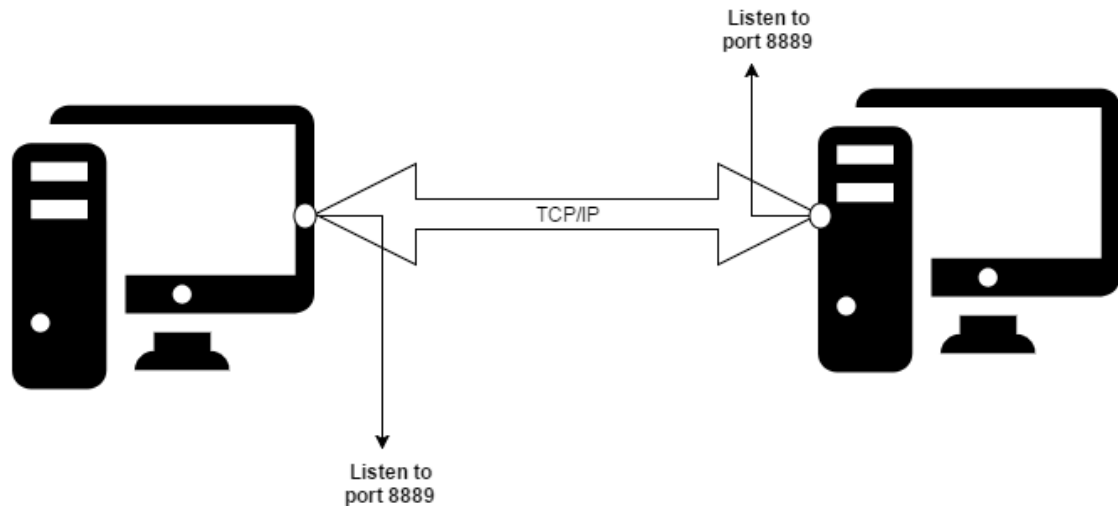


Ilustración 14 - Vista general del protocolo de conexión TCP

Para que el programa sincronizador pueda realizar la comunicación efectiva entre las computadoras se usa el Protocolo de Control de Transmisión. TCP es un protocolo principal en redes de computadoras TCP/IP, este está orientado a la conexión y necesita una comunicación *peer-to-peer* (con apretón de manos) para determinar la comunicación de inicio a fin.

5.1. Formatos

Se definieron, os siguientes formatos para los paquetes enviados entre ambas máquinas.

```
struct sync_file_message
{
    // Datos del archivo
    char filename[1000];
    time_t mtime;
    int size;
    // Contenido del archivo
    unsigned char fileBuff[1024];
};
```

Ilustración 15 - Formato del paquete de sincronización del archivo

- **Sync_file_message:** Este paquete (ver Ilustración 15) es utilizado enviar un archivo al servidor o al cliente desde alguno de los roles. Además, contiene un buffer de tamaño máximo de 1024 bytes, para la transmisión del archivo en pedazos de 1024.

- **Sync_message:** Este paquete (ver Ilustración 16) es utilizado para enviar solicitudes de eliminación, modificación y agregado de un archivo al servidor. Además, trae un indicador para una sincronización directa, en caso que el directorio del servidor esté vacío.

```
struct sync_message
{
    // Datos de archivo
    char message[1000];
    char name[1000];
    time_t mtime ;
    int size ;

    // Indicadores del evento
    int empty_directory ;
    int deleted_file ;
    int modified_file ;
    int added_file ;
};
```

Ilustración 16 - Formato del paquete de mensajes de sincronización

6. Análisis de resultados de pruebas

A continuación, se presenta la definición y resultado de las pruebas aplicadas al sistema de sincronización de archivos.

Las pruebas de conexión se realizaron con dos máquinas virtuales corriendo Ubuntu Versión 16.94.1. La conexión se realizó a través de redes virtuales configuradas manualmente desde Virtual Box.

6.1. Definición de pruebas

A continuación, se presenta la definición de pruebas para el sistema implementado.

Numero de prueba	Definición de la prueba	Operaciones usadas
1	Se agregan los siguientes archivos al cliente: <ul style="list-style-type: none">• Chrysanthemum.jpg• Comandos_CISCO.docx• ConversionesDeBases.py• Desert.jpg• Empaquetado.zip• Hydrangeas.jpg• Jellyfish.jpg• PruebaFuncionesRecurativas.py• USHER_house.txt• alice_in_wonderland.txt• circuitos_aritemeticos.pdf• italian.mp3• latin.mp3• portuguese.mp3	Agregar
2	Se eliminan los siguientes del cliente: <ul style="list-style-type: none">• USHER_house.txt• Italian.mp3• circuitos_aritemeticos.pdf	Eliminar
3	Se modifican los siguientes archivos del cliente: <ul style="list-style-type: none">• ConversionesDeBases.py• PruebaFuncionesRecurativas.py	Archivo modificado solo en el cliente.

4	<p>Se agrega los siguientes archivos antes de la prueba:</p> <ul style="list-style-type: none"> • LINKS.txt • Sym.java • Movimientos.xml <p>Se modifican los siguientes archivos tanto en el cliente como en el servidor.</p> <ul style="list-style-type: none"> • LINKS.txt 	Archivo modificado en ambos, servidor y cliente.
5	Se eliminan todos los archivos del cliente	Eliminar
6	Se trata de sincronizar el cliente con el servidor una vez el directorio vacío.	Directorios vacíos
7	<p>Se sincroniza otro directorio existente agregando los siguientes documentos:</p> <p>(ahora se actualiza del servidor al cliente)</p> <ul style="list-style-type: none"> • alice_in_wonderland.txt • alice_in_wonderlandV2.txt • cancionbonita.mp3 • empaquetado.zip • especificacionProyecto.pdf • file1.txt • file2.txt • file3.txt • hola.txt <p>y modificando los siguientes en el servidor:</p> <ul style="list-style-type: none"> • file1 • file 2 <p>y eliminado los siguientes</p> <ul style="list-style-type: none"> • cancionbonita.mp3 • alice_in_wonderlandV2.txt 	Se agrega y se modifica archivos

6.2. Resultados de las pruebas

A continuación, se presenta pantallazos de los resultados obtenidos en las pruebas aplicadas al sistema.

Número	Resultado
1	<p>Notificaciones del servidor</p> <pre>nara15:~/workspace/production/server \$./main_sync -d prueba/ SERVIDOR ===== directorio: prueba/ ¡Hola!, soy el server esperando por conexiones Conexión aceptada Nombre del archivo prueba/Chrysanthemum.jpg y su tamaño 879394 Nombre del archivo prueba/Comandos_CISCO.docx y su tamaño 90869 Nombre del archivo prueba/ConversionesDeBases.py y su tamaño 3807 Nombre del archivo prueba/Desert.jpg y su tamaño 845941 Nombre del archivo prueba/Hydrangeas.jpg y su tamaño 595284 Nombre del archivo prueba/Jellyfish.jpg y su tamaño 775702 Nombre del archivo prueba/PruebaFuncionesRecursivas.py y su tamaño 225 Nombre del archivo prueba/USHER_house.txt y su tamaño 63245 Nombre del archivo prueba/alice_in_wonderland.txt y su tamaño 167550 Nombre del archivo prueba/circuitos_aritmeticos.pdf y su tamaño 332689 Nombre del archivo prueba/empaquetado.zip y su tamaño 3607 Nombre del archivo prueba/italian.mp3 y su tamaño 354328 Nombre del archivo prueba/latin.mp3 y su tamaño 223506 Nombre del archivo prueba/portuguese.mp3 y su tamaño 459532 Cliente desconectado</pre> <p>Directorio del servidor después de la transferencia</p> <pre>nara15:~/workspace/production/server/prueba \$ ls Chrysanthemum.jpg Hydrangeas.jpg alice_in_wonderland.txt latin.mp3 Comandos_CISCO.docx Jellyfish.jpg circuitos_aritmeticos.pdf portuguese.mp3 ConversionesDeBases.py PruebaFuncionesRecursivas.py empaquetado.zip Desert.jpg USHER_house.txt italian.mp3</pre>
2	<p>Notificaciones del servidor</p> <pre>nara15:~/workspace/production/server \$./main_sync -d prueba/ SERVIDOR ===== directorio: prueba/ ¡Hola!, soy el server esperando por conexiones Conexión aceptada ELIMINANDO el archivo prueba/USHER_house.txt ELIMINANDO el archivo prueba/circuitos_aritmeticos.pdf ELIMINANDO el archivo prueba/italian.mp3 Cliente desconectado</pre>

	<p>Directorio del servidor después de la operación de la eliminación</p> <pre>nara15:~/workspace/production/server/prueba \$ ls Chrysanthemum.jpg Desert.jpg PruebaFuncionesRekursivas.py latin.mp3 Comandos_CISCO.docx Hydrangeas.jpg alice_in_wonderland.txt portuguese.mp3 ConversionesDeBases.py Jellyfish.jpg empaquetado.zip</pre>
3	<p>Notificaciones del servidor</p> <pre>nara15:~/workspace/production/server \$./main_sync -d prueba/ SERVIDOR ===== directorio: prueba/ ¡Hola!, soy el server esperando por conexiones Conexión aceptada MODIFICANDO el archivo ConversionesDeBases.py MODIFICANDO el archivo PruebaFuncionesRekursivas.py Cliente desconectado</pre> <p>Notificaciones del cliente</p> <pre>nara15:~/workspace/production/client \$./main_sync -d prueba/ -h 172.168.12.1 CLIENTE ===== directorio: prueba/ , y el servidor: 172.168.12.1 El cliente se conectó Procesando los archivos del cliente El cliente tiene el más reciente que el servidor El cliente tiene el más reciente que el servidor</pre>
4	<p>Notificación del servidor</p> <pre>nara15:~/workspace/production/server \$./main_sync -d prueba/ SERVIDOR ===== directorio: prueba/ ¡Hola!, soy el server esperando por conexiones Conexión aceptada Nombre del archivo prueba/LINKS.txt y su tamaño 0 Nombre del archivo prueba/Movimientos.xml y su tamaño 20643410 Nombre del archivo prueba/sym.java y su tamaño 132 Cliente desconectado nara15:~/workspace/production/server \$./main_sync -d prueba/ SERVIDOR ===== directorio: prueba/ ¡Hola!, soy el server esperando por conexiones Conexión aceptada MODIFICANDO el archivo LINKS.txt El archivo LINKS.txt está cambiado en ambas partes, y su nombre en el server es: prueba/gQzLINKS.txt El nombre en el cliente es: prueba/OciLINKS.txt y su tamaño es 8 Cliente desconectado</pre> <p>Notificación del cliente</p> <pre>nara15:~/workspace/production/client \$./main_sync -d prueba/ -h 172.168.12.1 CLIENTE ===== directorio: prueba/ , y el servidor: 172.168.12.1 El cliente se conectó Procesando los archivos del cliente El nombre en el servidor es: prueba/gQzLINKS.txt y su tamaño es 6</pre>

	<p>Directorio del servidor</p> <pre>nara15:~/workspace/production/server/prueba \$ ls Movimientos.xml OciLINKS.txt gQzLINKS.txt sym.java</pre> <p>Directorio del cliente</p> <pre>nara15:~/workspace/production/client/prueba \$ ls Movimientos.xml OciLINKS.txt gQzLINKS.txt sym.java</pre>
5	<p>Notificaciones del servidor</p> <pre>nara15:~/workspace/production/server \$./main_sync -d prueba/ SERVIDOR ===== directorio: prueba/ ¡Hola!, soy el server esperando por conexiones Conexión aceptada ELIMINANDO el archivo prueba/Movimientos.xml ELIMINANDO el archivo prueba/OciLINKS.txt ELIMINANDO el archivo prueba/gQzLINKS.txt ELIMINANDO el archivo prueba/sym.java Cliente desconectado</pre>
6	<p>Notificación del servidor</p> <pre>nara15:~/workspace/production/server \$./main_sync -d prueba/ SERVIDOR ===== directorio: prueba/ ¡Hola!, soy el server esperando por conexiones Conexión aceptada Cliente desconectado</pre>
7	<p>Notificación del servidor al agregar</p> <pre>nara15:~/workspace/production/server \$./main_sync -d hola/ SERVIDOR ===== directorio: hola/ ¡Hola!, soy el server esperando por conexiones Conexión aceptada Nombre del archivo hola/alice_in_wonderland.txt y su tamaño 173588 Nombre del archivo hola/alice_in_wonderlandV2.txt y su tamaño 173588 Nombre del archivo hola/cancionbonita.mp3 y su tamaño 3062186 Nombre del archivo hola/empaquetado.zip y su tamaño 3607 Nombre del archivo hola/especificacionProyecto.pdf y su tamaño 89029 Nombre del archivo hola/file1.txt y su tamaño 17 Nombre del archivo hola/file2.txt y su tamaño 19 Nombre del archivo hola/file3.txt y su tamaño 17 Nombre del archivo hola/hola.txt y su tamaño 4 Cliente desconectado</pre>

Notificaciones del cliente, como servidor, al realizar las operaciones.

```
nara15:~/workspace/production/client $ ./main_sync -d hola/  
SERVIDOR ===== directorio: hola/  
¡Hola!, soy el server esperando por conexiones  
Conexión aceptada  
ELIMINANDO el archivo hola/alice_in_wonderlandV2.txt  
ELIMINANDO el archivo hola/cancionbonita.mp3  
MODIFICANDO el archivo file1.txt  
MODIFICANDO el archivo file2.txt  
Cliente desconectado
```

Estado del directorio en el cliente luego de las operaciones

```
nara15:~/workspace/production/client/hola $ ls  
alice_in_wonderland.txt  especificacionProyecto.pdf  file2.txt  folder_prueba/  
empaquetado.zip         file1.txt                  file3.txt  hola.txt
```

7. Conclusiones sobre rendimiento y funcionamiento

A continuación, se presenta un cuadro resumen sobre el funcionamiento en el sistema. En este se muestra las características implementadas y faltantes.

Aspecto	Puntos logrados	Puntos débiles / mejoras
Manipulación de sockets	El programa realiza una conexión exitosa entre ambas máquinas desde el puerto 8889 de ambas.	
Manejo de solicitudes	Se implementó una solución satisfactoria para el transporte de mensajes entre las máquinas y la sincronización de los archivos	
Rutina de comparación	Exitosamente se genera un listado de archivos modificado, eliminados y agregados. Así el programa mantiene el control de los cambios pertinentemente.	
Mecanismo de bloqueos	Los archivos del directorio son todos bloqueados durante su proceso de lectura.	No se logró un mecanismo de bloqueo del directorio. El programa no implementa hilos, pero eventualmente su implementación así, estos bloqueos ya implementados sería de gran ayuda.
Transferencia	La transferencia de cliente-servidor se realiza exitosamente entre las máquinas	No se implementó un mecanismo de sincronización bidireccional. El server es una especie de respaldo del cliente, no se puede realizar sincronización servidor-cliente.

Como cualquier ejercicio académico, el programa está sujeto a un mejoramiento posterior. No obstante; la implementación fue exitosa y se concluye que el programa cumple con los mínimos requerimientos técnicos para efectos del proyecto.

8. Anexos

8.1. Estructura del proyecto

A continuación, se adjunta una tabla resumen de los archivos contenidos en el repositorio del sistema.

Archivo	Descripción
client.c	Archivo con las funcionalidades de solicitud y envío de archivos del lado del cliente. Además, cuenta con la configuración del socket cliente.
main_sync.c	Programa principal de entrada. Toma la decisión de qué rol tomar el programa dependiendo de las entradas.
Makefile	Makefile para compilar todo el proyecto. Se genera un ejecutable main_sync
procs.c	Este archivo contiene todas las funciones para el manejo y escaneo de archivos de un directorio. Así, como las funciones para manejar los meta datos generados sobre el directorio.
server.c	Archivo con las funcionalidades de envío y recibimiento de archivos del lado del servidor. Además, cuenta con la configuración del socket server para recibir solicitudes del cliente.
socket.c	Archivo con funcionalidades de escritura y lectura sobre los sockets.
structs.c y structs.h	Definición de las estructuras y tipos de datos del sistema
test.c	Archivo misceláneo y pruebas.

8.2. URL al repositorio

Se adjunta el link al repositorio del proyecto

https://github.com/nara15/dropbox_simulation.git

Además, se adjunta junto con este documento el código fuente del sistema sincronizador.