**Tecnológico de Costa Rica**

Campus Tecnológico Central Cartago

**Escuela de Ingeniería en Computación**

Curso: IC 6600 - *Sistemas Operativos*

Grupo n.º 2

**Profesor**

Armando Arce Orozco

**Estudiante**

| | |
|---|---|
| David Salazar Rodriguez | 2019000145 |
| Jonathan Camacho Tencio | 2019083743 |


# Proyecto 3

**Fecha de entrega**

17 de Junio

Semestre I del 2024

## Tabla de Contenidos

## Introducción

El presente documento tiene como objetivo presentar los resultados obtenidos, así como especificar las estructuras y estrategias utilizadas para desarrollar el tercer proyecto, en el cual se codifica un cliente FTP que opere bajo el modelo peer-to-peer, permitiendo la transferencia y recepción de archivos entre programas clientes instalados en máquinas remotas. El proyecto es desarrollado en un ambiente UNIX en el lenguaje de programación C, este enfoque busca simplificar la transferencia de archivos al evitar la necesidad de configurar un servidor FTP, proporcionando una solución más accesible y fácil de usar para los usuarios y la comunicación entre la red de computadoras.

Además, se especifican las funciones que se pueden aplicar para comunicar las máquinas en la red, los directorios y la interacción con los archivos.

## Descripción del problema

Un problema que se presenta con el servicio FTP (file transfer protocol), utilizado tradicionalmente para la transferencia de archivos entre máquinas, es la tarea a veces complicada de instalar un programa servidor en la máquina a la que se desean transferir/recuperar archivos (modelo cliente-servidor), ya que se requieren privilegios de administrador.

Un enfoque más sencillo para implementar un servicio FTP sería simplemente que dos programas clientes se comuniquen entre sí y transfieran archivos entre ellos sin necesidad de instalar un servidor (modelo peer-to-peer).

En este proyecto se pretende desarrollar un cliente FTP que funcione de ese modo, es decir, que permita transferir y recibir archivos entre programas clientes que se encuentren instalados en máquinas remotas.

El programa btfp reconocerá un subconjunto de instrucciones de FTP que permitirán la conexión y transferencia de archivos:

open <dirección-ip>: establece una conexión remota
close: cierra la conexión actual
quit: termina el programa
cd <directorio>: cambia de directorio remoto
get <archivo>: recupera un archivo remoto
lcd <directorio>: cambia de directorio local
ls: lista los archivos del directorio remoto
put <archivo>: envía un archivo a la máquina remota
pwd: muestra el directorio activo remoto

La transferencia de archivos realizará una copia del archivo en cuestión desde la máquina remota hasta la máquina local (o viceversa) almacenando éste con el mismo nombre que se encontraba en la otra máquina.

El programa bftp se debe ejecutar en al menos dos máquinas al mismo tiempo, y quedarán esperando conexiones en el puerto 8889. Posteriormente, permitirá que el usuario ingrese comandos en la terminal y los ejecutará. Note que un usuario solo puede establecer una conexión con una máquina a un tiempo, pero múltiples máquinas remotas pueden establecer conexión con la máquina de dicho usuario al mismo tiempo.

Internamente cada programa se deberá comportar como un servidor y un cliente al mismo tiempo. Por eso se debe utilizar un mecanismo (threads) que le permita al programa estar haciendo varias cosas al mismo tiempo: recibir comandos del usuario, atender todas las conexiones entrantes y ejecutar sus comandos. El programa debe utilizar tanto sockets clientes como un socket servidor para realizar las conexiones adecuadamente.

## Manejo de sockets

El manejo de sockets en C implica una serie de pasos para la creación de sockets, el enlace a direcciones, la escucha de conexiones, la aceptación de conexiones entrantes y el envío/recepción de datos.

**El servidor:** primero, se crea un socket utilizando la función **socket**, especificando el dominio de la dirección (AF_INET para IPv4), el tipo de socket (SOCK_STREAM para TCP), y el protocolo ( 0 para que sea automáticamente). Luego, el socket debe estar asociado a una dirección IP y un puerto mediante bind, configurando la estructura sockaddr_in con la familia de direcciones, la dirección IP (INADDR_ANY para cualquier dirección local), y el puerto, que se convierte a formato de red con htons.

El siguiente paso es poner el socket en modo de escucha para aceptar conexiones entrantes usando **listen**. Cuando un cliente intenta conectarse, el servidor acepta la conexión con **accept**, que devuelve un nuevo socket para la comunicación con el cliente.

**El cliente:** por otro lado, en el lado del cliente, se utiliza connect para establecer una conexión con el servidor, configurando la estructura sockaddr_in con la dirección IP del servidor y el puerto, y convirtiéndola a formato de red con inet_pton.

Para la comunicación, se usan las funciones **send** y **recv** para enviar y recibir datos respectivamente, send envía datos a través del socket, y recv recibe datos. Finalmente, se cierra el socket con **close** para liberar los recursos y poder conectarse a una nueva red.. Este proceso de manejo de sockets se integra en un programa donde el servidor puede escuchar conexiones, aceptar clientes, y manejar comandos como listar directorios (ls), cambiar de directorio (cd), y enviar/recibir archivos (get/send). El cliente puede conectarse a un servidor, enviar comandos, y recibir respuestas, incluyendo la transferencia de archivos utilizando las funciones mencionadas.

## Manejo de solicitudes

Cada computadora puede actuar como cliente o servidor, a continuación aparece que pasos realizan para comunicarse y manejar solicitudes:

Servidor:

- Crea un socket y lo enlaza a una dirección IP y un puerto.
- Se pone en modo de escucha para aceptar conexiones entrantes.
- Acepta una conexión y crea un nuevo hilo para manejarla.
- En el hilo, recibe y procesa comandos del cliente.

Cliente:

- Crea un socket y lo conecta al servidor.
- Envía comandos al servidor (como listar directorios o solicitar un archivo).
- Recibe respuestas del servidor y maneja la comunicación.

## Rutina de transferencia

En la rutina de transferencia de archivos en el proyecto, ya sea que actúe como el servidor o como el cliente, tienen roles para lograr una comunicación eficiente y correcta. Cuando un cliente desea obtener un archivo, envía un comando **get <archivo>** al servidor a través de un socket. Este comando es recibido por el hilo correspondiente del servidor, que luego busca el archivo solicitado en el sistema de archivos. Si el archivo existe y puede ser abierto, el servidor lee el contenido del archivo en bloques utilizando un buffer de tamaño predefinido (en el proyecto es la macro BUFFER_SIZE con el tamaño de 1024) y envía estos bloques al cliente a través del socket. El cliente, por su parte, recibe estos bloques de datos y los escribe en un archivo local. Esta transferencia continúa en un bucle hasta que todos los bloques del archivo sean enviados y recibidos. Para asegurar que el proceso se complete correctamente, se manejan posibles errores y se cierra el archivo tanto en el servidor como en el cliente una vez finalizada la transferencia.

De la misma manera, se puede cargar un archivo al servidor, con el comando **put <archivo>,** de igual manera se pasa la información a través del socket, solo que esta vez, va desde la computadora que está actuando de cliente al servidor.

## Manejo de hilos

En este programa, se utiliza el manejo de hilos para permitir que el servidor pueda manejar múltiples conexiones de clientes de manera concurrente. Cuando el servidor acepta una nueva conexión de cliente, se crea un nuevo hilo mediante pthread_create, y este hilo se encarga de manejar toda la comunicación con ese cliente específico, ejecutando la función handle_client. Esta función en el hilo recibe y procesa los comandos enviados por el cliente. Cada conexión al cliente se gestiona en un hilo separado, lo que permite atender múltiples clientes al mismo tiempo, sin problema. En cuanto al cliente, cuando se conecta , utiliza un socket para enviar comandos y recibir respuestas, pero no necesita crear un hilo adicional ya que la comunicación es manejada secuencialmente.

## Descripción de protocolos y formatos

### Análisis de resultados de pruebas

Ejecución de la instrucción **“local ls**” la cual lista los archivos del directorio en donde se encuentra actualmente el programa.

```
> Servidor escuchando en el puerto 8889
local ls
Lorem.txt  cliente.c  hola.py  main  main.c  servidor.c
```

Utilizando **open** seguido de la ip de la máquina a la cual nos queremos conectar se establece una conexión, esta se confirma del lado del cliente del servidor
**Maquina 1:**

```
> open 192.168.10.161
Conexión establecida con 192.168.10.161
```

**Maquina 2:**

```
Conexión aceptada
```

Haciendo uso de la instrucción **“remote ls”** esta nos devuelve un listado de los archivos que se encuentran en el directorio actual de la máquina remota luego de 5 segundos podremos volver a introducir instrucciones:

```
> remote ls
Archivos en el Servidor:.git
cliente.c
hola.py
knight.jpg
Lorem.txt
main
main.c
proy
servidor.c
```

Utilizando la función **local cd <path>** es posible cambiar de directorio local, en el ejemplo se muestra un uso de esto y posteriormente se hace un listado del nuevo directorio.

```
> local cd /home/kbaio/Sistemas/Tarea 1
Directorio cambiado a: /home/kbaio/Sistemas/Tarea 1
> local ls
Tarea1_1-2019000145    Tarea1_2-2019000145.c  Tarea1_3-2019000145.c  tarea1
Tarea1_1-2019000145.c  Tarea1_2019000145.zip  Tarea1_4-2019000145    tarea1.c
Tarea1_2-2019000145    Tarea1_3-2019000145    Tarea1_4-2019000145.c
>
```

De la misma manera haciendo uso de **remote cd <path>** es posible cambiar el path de la máquina remota:

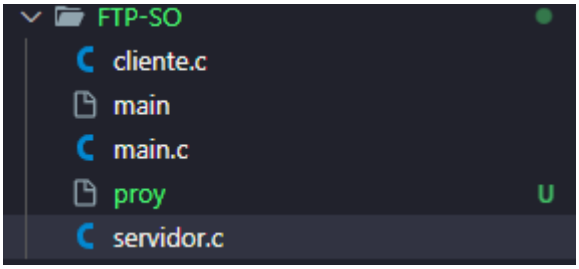
```
> remote cd /home/david
> remote ls
Archivos en el Servidor:.bashrc
.bash_history
.bash_logout
.cache
.dotnet
.gitconfig
.local
.motd_shown
.profile
.sudo_as_admin_successful
.vscode-server
Proyecto
>
```

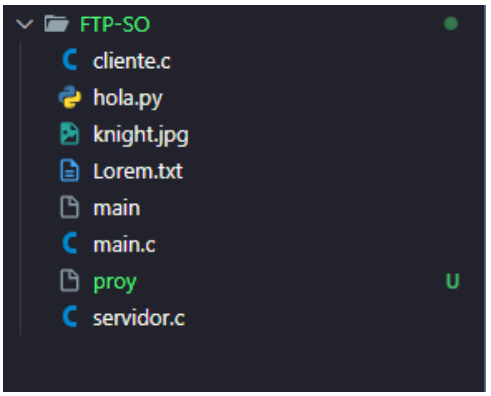
Haciendo uso del comando **send <archivo.ext>** se logran transferir archivos de una maquina a otra.

**Maquina Origen:**

```
Conexión establecida con 192.168.10.161
> send hola.py
Archivo enviado con éxito: hola.py
> send Lorem.txt
Archivo enviado con éxito: Lorem.txt
> send
```

**Máquina Destino:**




## Conclusiones sobre rendimiento y correctitud

Durante la ejecución del proyecto logramos recrear en su mayoría el procedimiento de FTP, haciendo uso de hilos de ejecución, sockets y creando nuestros propios comandos para interactuar, realizar la conexión haciendo uso de un modelo peer to peer (P2P), esto nos permitió analizar y comprender mejor cómo es que funcionan no solo este tipo de conexiones si no que también pudimos experimentar un poco de cómo es el File Transfer Protocol funciona por debajo, a lo largo de este proyecto tuvimos distintos problemas ya fueran con los threads, los sockets o la transferencia de archivos, para llegar a encontrar soluciones a esto tuvimos que investigar, replantear y rehacer multiples veces el codigo. Sin embargo con todos los problemas que se nos presentaron fue una experiencia que nos permitió conocer y entender mejor estas herramientas.