Instituto Politécnico Nacional

IPN

Escuela Superior de Cómputo

ESCOM

Academia de Redes Aplicaciones para comunicaciones en red

6CV2

Práctica 10

“Protocolo TFTP”

Alumna:

Navarrete Becerril Sharon Anette

Fecha de entrega: “ 28 - Noviembre - 2024”

Profesor: Ojeda Santillan Rodrigo

OBJETIVO

Demostrar el funcionamiento del protocolo TFTP, por lo que se va a realizar un cliente y un servidor que permita la transferencia de archivos, se sugiere realizarlo en una distribución de Linux.

INTRODUCCIÓN

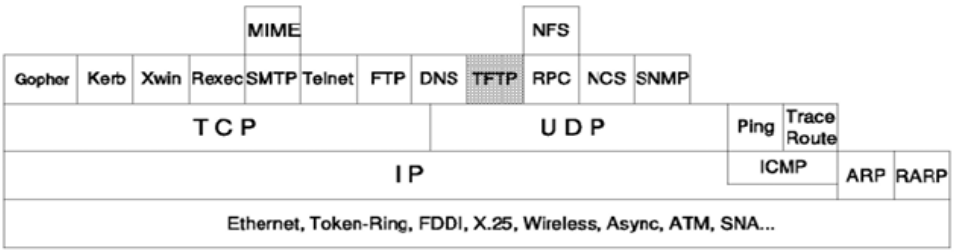
El Protocolo de Transferencia de Archivos Trivial (TFTP) es un protocolo sencillo diseñado para la transferencia de archivos a través de una red. Utiliza el protocolo UDP (User Datagram Protocol) para enviar datos, lo que lo hace liviano y adecuado para redes con bajos requerimientos en términos de control y fiabilidad.

TFTP facilita la transferencia de archivos entre un cliente y un servidor de forma eficiente. Su simplicidad lo convierte en una buena opción para dispositivos con recursos limitados, como routers o switches. A diferencia de protocolos más avanzados como FTP o HTTP, TFTP no ofrece características como autenticación o cifrado. Características de TFTP:

- **Simplicidad**: Diseño básico y fácil de implementar.
- **Uso de UDP**: No garantiza la entrega de los datos, haciéndolo menos confiable que TCP.
- **Transferencia en bloques**: Los archivos se transfieren en bloques de tamaño fijo.
- **Sin seguridad**: No incluye autenticación ni cifrado.

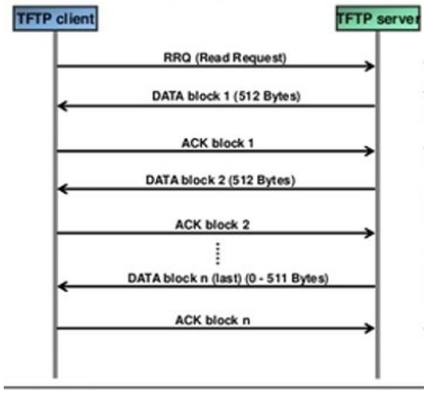
Definido en el RFC 1350, TFTP transfiere archivos en bloques, con la confirmación de recepción mediante paquetes ACK (Acknowledgment). Proceso de transferencia:

- **Inicio**: El cliente TFTP inicia la transferencia enviando una solicitud de lectura (RRQ) o escritura (WRQ) al servidor, especificando el nombre del archivo y el modo de transferencia (como "netascii" o "octet").
- **Envío de datos**: En una solicitud de lectura, el servidor envía el archivo en bloques, precedidos por un número de bloque único.
- **Confirmación**: El cliente confirma la recepción de cada bloque enviando un ACK con el número de bloque correspondiente.
- **Finalización**: La transferencia concluye cuando se envía un bloque de datos más pequeño que el tamaño máximo (512 bytes), indicando el fin del archivo.



DESARROLLO

Para replicar la comunicación mostrada en la imagen siguiente, donde un cliente realiza una petición a un servidor, se sigue el siguiente procedimiento.



El código crea un socket UDP mediante socket(AF_INET, SOCK_DGRAM, 0) y configura tanto el socket local como el remoto. Esto incluye la vinculación con bind y la configuración de la dirección del servidor remoto utilizando inet_addr. Dependiendo de la opción seleccionada por el usuario, el programa puede solicitar la lectura o escritura de un archivo.

En una solicitud de lectura, el cliente envía la petición al servidor TFTP y, si tiene éxito, recibe el archivo en bloques que se almacenan localmente. La recepción se realiza en un bucle que continúa hasta que se completa la transferencia o se alcanza un límite de 10 MB. Para la escritura, el archivo se divide en bloques de 512 bytes que se envían al servidor TFTP. Después de cada bloque enviado, el cliente espera un ACK (Acknowledgment) del servidor antes de continuar con el siguiente bloque. El código también maneja errores, como la falta de recepción de ACKs.

El código incluye funciones para crear peticiones de lectura y escritura, enviar ACKs, y gestionar la estructura de los datos enviados y recibidos. Primero, se establecerá la lógica para desarrollar el cliente, añadiendo el siguiente fragmento de código al archivo cliente.c:

Código del cliente:

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#pragma comment(lib, "ws2_32.lib") // Vincula automáticamente la biblioteca WinSock

short numPaq;
unsigned char numPaqHex[2];
int udp_socket, tamEnviado, tamTotal = 0, tamRecivido, tamData;
unsigned char data[516], estrPeticionLectura[516], estrACK[4], mensajeRecivido[516],
nomArch[30];
struct sockaddr_in remota, cliente;
```

```
FILE *fw, *fr;

// Declaración de funciones
void EnviarACK();
int PeticionLectura();
int PeticionEscritura();
int EstructuraACK(unsigned char *paq);
int EstructuraPeticionLectura(unsigned char *paq, unsigned char *nomArch);
int EstructuraPeticionEscritura(unsigned char *paq, unsigned char *nomArch);

int main() {
   WSADATA wsaData;
   if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
      fprintf(stderr, "Error al iniciar WinSock: %d\n", WSAGetLastError());
      exit(EXIT_FAILURE);
   }

   udp_socket = socket(AF_INET, SOCK_DGRAM, 0);
   if (udp_socket == INVALID_SOCKET) {
      fprintf(stderr, "Error al abrir el socket: %d\n", WSAGetLastError());
      WSACleanup();
      exit(EXIT_FAILURE);
   }

   remota.sin_family = AF_INET;
   remota.sin_port = htons(69);
   remota.sin_addr.s_addr = inet_addr("127.0.0.1");  // IP local del servidor

   int opcion = 0;
   while (opcion != 3) {
      printf("\n | 1-Solicitud Lectura | 2-Solicitud Escritura | 3-Salir |\n");
      printf("Seleccione una opcion: ");
      scanf("%i", &opcion);
      getchar();  // Consumir el salto de línea

      switch (opcion) {
         case 1:
            printf("Archivo a leer: ");
            fgets(nomArch, sizeof(nomArch), stdin);
            strtok(nomArch, "\n");
            if (PeticionLectura() == 3) {
               printf("Archivo recibido correctamente.\n");
            } else {
               printf("Error al recibir el archivo.\n");
            }
            break;

         case 2:
            printf("Archivo a escribir: ");
            fgets(nomArch, sizeof(nomArch), stdin);
            strtok(nomArch, "\n");
            if (PeticionEscritura() == 4) {
               printf("Archivo enviado correctamente.\n");
            } else {
```

```
                printf("Error al enviar el archivo.\n");
            }
            break;

        case 3:
            closesocket(udp_socket);
            printf("Cliente cerrado.\n");
            break;

        default:
            printf("Opción no válida.\n");
      }
   }

   WSACleanup();
   return 0;
}

// Función para enviar un ACK
void EnviarACK() {
   int tamACK = EstructuraACK(estrACK);
   sendto(udp_socket, (char *)estrACK, tamACK, 0, (struct sockaddr *)&cliente, sizeof(cliente));
}

// Función para manejar la petición de lectura
int PeticionLectura() {
   int tamData = EstructuraPeticionLectura(estrPeticionLectura, nomArch);
   sendto(udp_socket, (char *)estrPeticionLectura, tamData, 0, (struct sockaddr *)&remota,
sizeof(remota));

   tamRecivido = recvfrom(udp_socket, (char *)mensajeRecivido, sizeof(mensajeRecivido), 0,
NULL, NULL);
   if (mensajeRecivido[1] == 0x03) {
      fw = fopen(nomArch, "wb");
      fwrite(mensajeRecivido + 4, 1, tamRecivido - 4, fw);
      fclose(fw);
      return 3;
   }
   return -1;
}

// Función para manejar la petición de escritura
int PeticionEscritura() {
   fr = fopen(nomArch, "rb");
   if (!fr) {
      printf("No se pudo abrir el archivo.\n");
      return -1;
   }

   int tamData = EstructuraPeticionEscritura(estrPeticionLectura, nomArch);
   sendto(udp_socket, (char *)estrPeticionLectura, tamData, 0, (struct sockaddr *)&remota,
sizeof(remota));

   while (!feof(fr)) {
```

```
        tamData = fread(data, 1, 512, fr);
        sendto(udp_socket, (char *)data, tamData, 0, (struct sockaddr *)&remota, sizeof(remota));
    }

    fclose(fr);
    return 4;
}

// Función para estructurar un ACK
int EstructuraACK(unsigned char *paq) {
    unsigned char codOP[2] = {0x00, 0x04};
    memcpy(paq, codOP, 2);
    return 4;
}

// Función para estructurar una petición de lectura
int EstructuraPeticionLectura(unsigned char *paq, unsigned char *nomArch) {
    unsigned char opLec[2] = {0x00, 0x01};
    unsigned char modo[] = "octet";
    memcpy(paq, opLec, 2);
    memcpy(paq + 2, nomArch, strlen(nomArch));
    memcpy(paq + strlen(nomArch) + 3, modo, strlen(modo) + 1);
    return strlen(nomArch) + 3 + strlen(modo) + 1;
}

// Función para estructurar una petición de escritura
int EstructuraPeticionEscritura(unsigned char *paq, unsigned char *nomArch) {
    unsigned char opEscritura[2] = {0x00, 0x02};
    unsigned char modo[] = "octet";
    memcpy(paq, opEscritura, 2);
    memcpy(paq + 2, nomArch, strlen(nomArch));
    memcpy(paq + strlen(nomArch) + 3, modo, strlen(modo) + 1);
    return strlen(nomArch) + 3 + strlen(modo) + 1;
}
```

El servidor implementa la transferencia de archivos utilizando el protocolo TFTP, manejando sockets y operaciones básicas de entrada y salida. Crea un socket UDP y lo enlaza al puerto 69, estándar de TFTP. En su bucle principal, espera recibir solicitudes de lectura o escritura de archivos.

Para una solicitud de lectura, el servidor abre el archivo en modo binario y lo envía en bloques de 512 bytes al cliente, esperando un ACK después de cada bloque. Si no recibe el ACK, reenvía el bloque. En una solicitud de escritura, el servidor crea un archivo nuevo, envía un ACK inicial, y luego recibe bloques de datos del cliente, escribiéndolos en el archivo y enviando un ACK por cada bloque recibido hasta que la transferencia termine.

Para la lógica del servidor se tiene que hacer un contexto similar al del cliente, no obstante se deben de realizar algunas modificaciones, se debe de agregar el siguiente fragmento de código al archivo servidor.c:

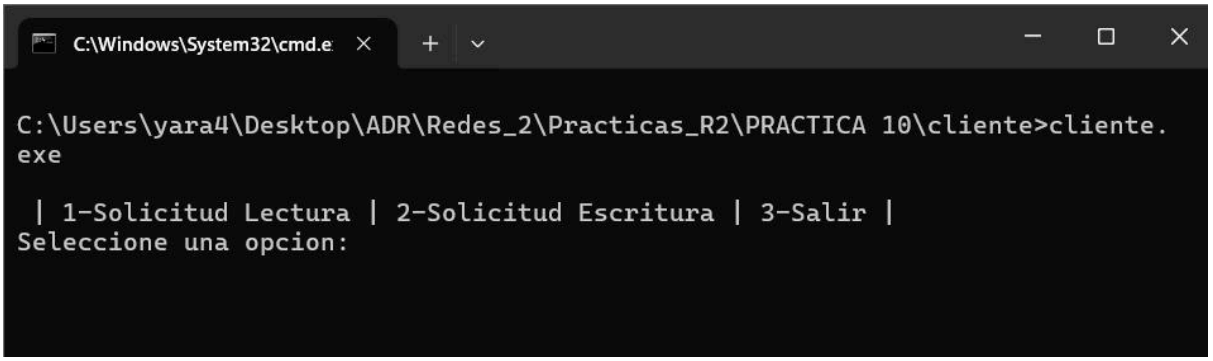
Código del servidor:

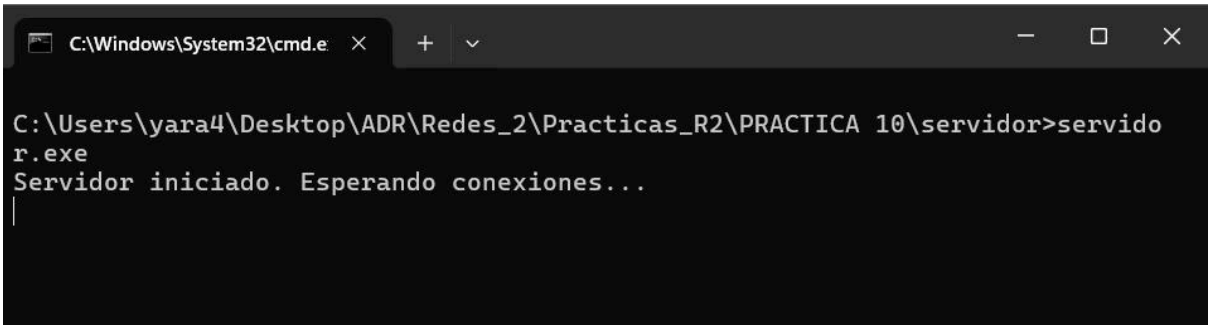
```
 #include <winsock2.h>
#include <ws2tcpip.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#pragma comment(lib, "ws2_32.lib") // Vincula automáticamente la biblioteca
WinSock

int udp_socket, lbind, tamRecivido, tamEnviado, tamTotal;
struct sockaddr_in servidor, cliente;
unsigned char mensajeRecivido[516], data[516];
FILE *fw, *fr;

// Declaración de funciones
void ManejarLectura();
void ManejarEscritura();
int EstructuraDatos(unsigned char *paq, int tam, unsigned short bloque);
int EstructuraACK(unsigned char *paq);
int EstructuraError(unsigned char *paq);

int main() {
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        fprintf(stderr, "Error al iniciar WinSock: %d\n",
WSAGetLastError());
        exit(EXIT_FAILURE);
    }

    udp_socket = socket(AF_INET, SOCK_DGRAM, 0);
    if (udp_socket == INVALID_SOCKET) {
        fprintf(stderr, "Error al abrir el socket: %d\n",
WSAGetLastError());
        WSACleanup();
        exit(EXIT_FAILURE);
    }

    servidor.sin_family = AF_INET;
    servidor.sin_port = htons(69);
    servidor.sin_addr.s_addr = htonl(INADDR_ANY);

    lbind = bind(udp_socket, (struct sockaddr *)&servidor,
sizeof(servidor));
    if (lbind == SOCKET_ERROR) {
        fprintf(stderr, "Error en bind: %d\n", WSAGetLastError());
        closesocket(udp_socket);
        WSACleanup();
        exit(EXIT_FAILURE);
    }

    printf("Servidor iniciado. Esperando conexiones...\n");

    while (1) {
        socklen_t clienteLen = sizeof(cliente);
```

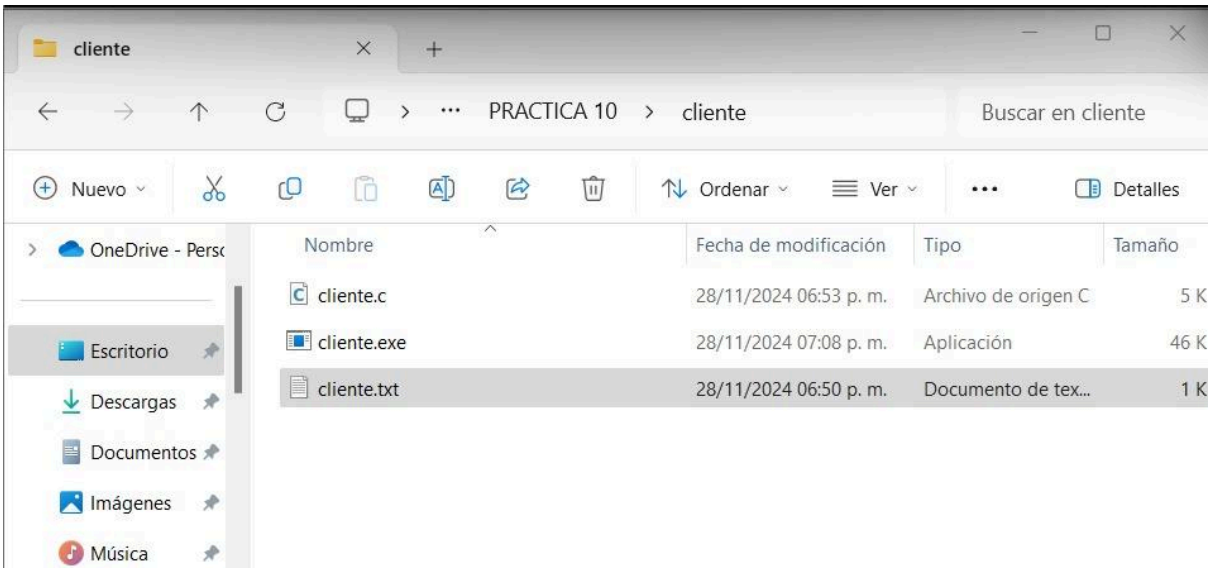
```
        tamRecivido = recvfrom(udp_socket, (char *)mensajeRecivido,
sizeof(mensajeRecivido), 0, (struct sockaddr *)&cliente, &clienteLen);
        if (tamRecivido == SOCKET_ERROR) {
            fprintf(stderr, "Error al recibir datos: %d\n",
WSAGetLastError());
            continue;
        }

        if (mensajeRecivido[1] == 0x01) { // Opcode 1: Solicitud de lectura
            printf("Solicitud de lectura recibida.\n");
            ManejarLectura();
        } else if (mensajeRecivido[1] == 0x02) { // Opcode 2: Solicitud de
escritura
            printf("Solicitud de escritura recibida.\n");
            ManejarEscritura();
        } else {
            printf("Código de operación no reconocido.\n");
        }
    }

    closesocket(udp_socket);
    WSACleanup();
    return 0;
}

// Manejo de solicitudes de lectura
void ManejarLectura() {
    fr = fopen((char *)(mensajeRecivido + 2), "rb");
    if (!fr) {
        printf("Archivo no encontrado: %s\n", mensajeRecivido + 2);
        int tamError = EstructuraError(data);
        sendto(udp_socket, (char *)data, tamError, 0, (struct sockaddr
*)&cliente, sizeof(cliente));
        return;
    }

    int tamParcial;
    unsigned short bloque = 1;

    while (!feof(fr)) {
        tamParcial = fread(data + 4, 1, 512, fr);
        int tamData = EstructuraDatos(data, tamParcial, bloque);
        sendto(udp_socket, (char *)data, tamData, 0, (struct sockaddr
*)&cliente, sizeof(cliente));
        bloque++;
    }

    fclose(fr);
    printf("Archivo enviado correctamente.\n");
}

// Manejo de solicitudes de escritura
void ManejarEscritura() {
    fw = fopen((char *)(mensajeRecivido + 2), "wb");
    if (!fw) {
        printf("Error al crear archivo: %s\n", mensajeRecivido + 2);
        int tamError = EstructuraError(data);
        sendto(udp_socket, (char *)data, tamError, 0, (struct sockaddr
*)&cliente, sizeof(cliente));
        return;
```

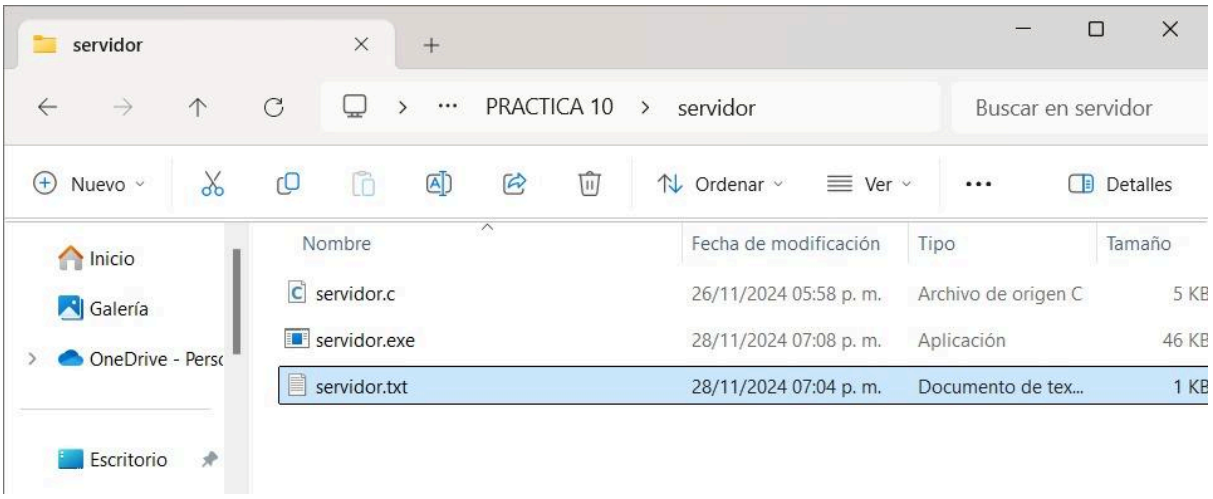
```
    }

    while (1) {
        socklen_t clienteLen = sizeof(cliente);
        tamRecivido = recvfrom(udp_socket, (char *)mensajeRecivido,
sizeof(mensajeRecivido), 0, (struct sockaddr *)&cliente, &clienteLen);
        if (tamRecivido < 4) break;

        fwrite(mensajeRecivido + 4, 1, tamRecivido - 4, fw);
        int tamACK = EstructuraACK(data);
        sendto(udp_socket, (char *)data, tamACK, 0, (struct sockaddr
*)&cliente, sizeof(cliente));
    }

    fclose(fw);
    printf("Archivo recibido correctamente.\n");
}

// Estructura de datos para enviar
int EstructuraDatos(unsigned char *paq, int tam, unsigned short bloque) {
    unsigned char opLec[2] = {0x00, 0x03};
    unsigned char bloqueHex[2] = {bloque >> 8, bloque & 0xFF};
    memcpy(paq, opLec, 2);
    memcpy(paq + 2, bloqueHex, 2);
    return tam + 4;
}

// Estructura de un ACK
int EstructuraACK(unsigned char *paq) {
    unsigned char codOP[2] = {0x00, 0x04};
    unsigned char bloqueHex[2] = {mensajeRecivido[2], mensajeRecivido[3]};
    memcpy(paq, codOP, 2);
    memcpy(paq + 2, bloqueHex, 2);
    return 4;
}

// Estructura de un mensaje de error
int EstructuraError(unsigned char *paq) {
    unsigned char codOP[2] = {0x00, 0x05};
    unsigned char codError[2] = {0x00, 0x01};
    char mensaje[] = "Archivo no encontrado";
    memcpy(paq, codOP, 2);
    memcpy(paq + 2, codError, 2);
    memcpy(paq + 4, mensaje, strlen(mensaje) + 1);
    return strlen(mensaje) + 5;
}
```

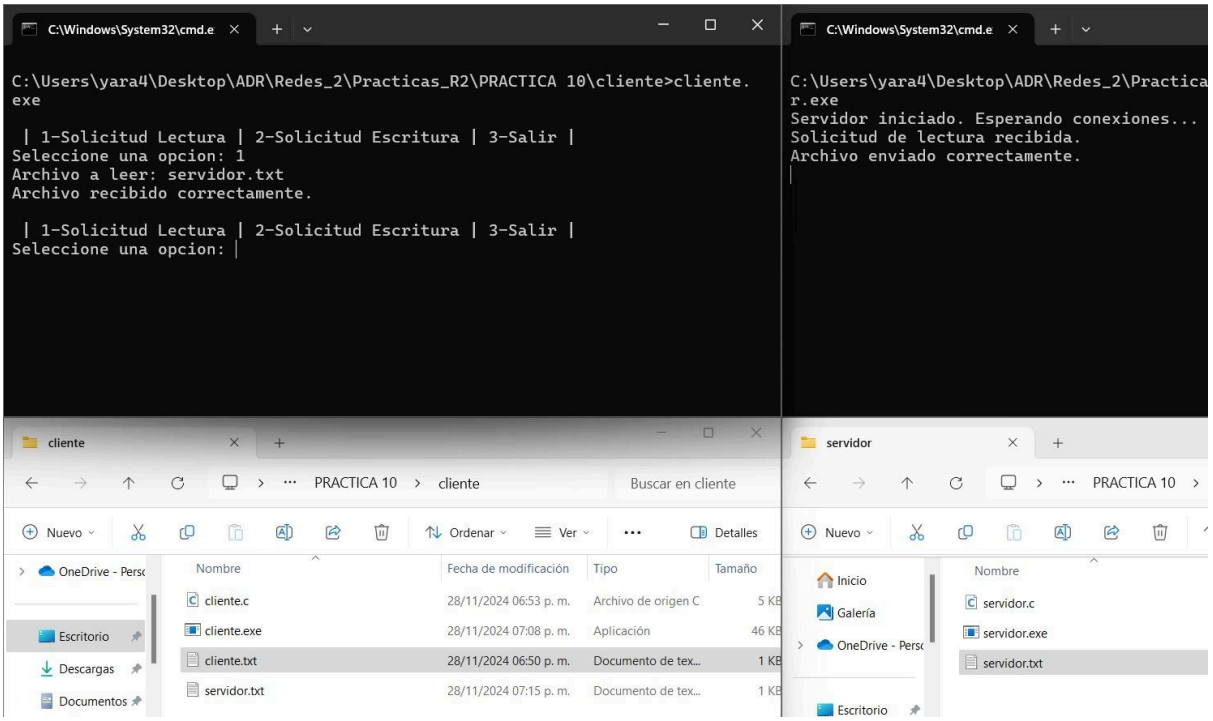
Capturas de funcionamiento:

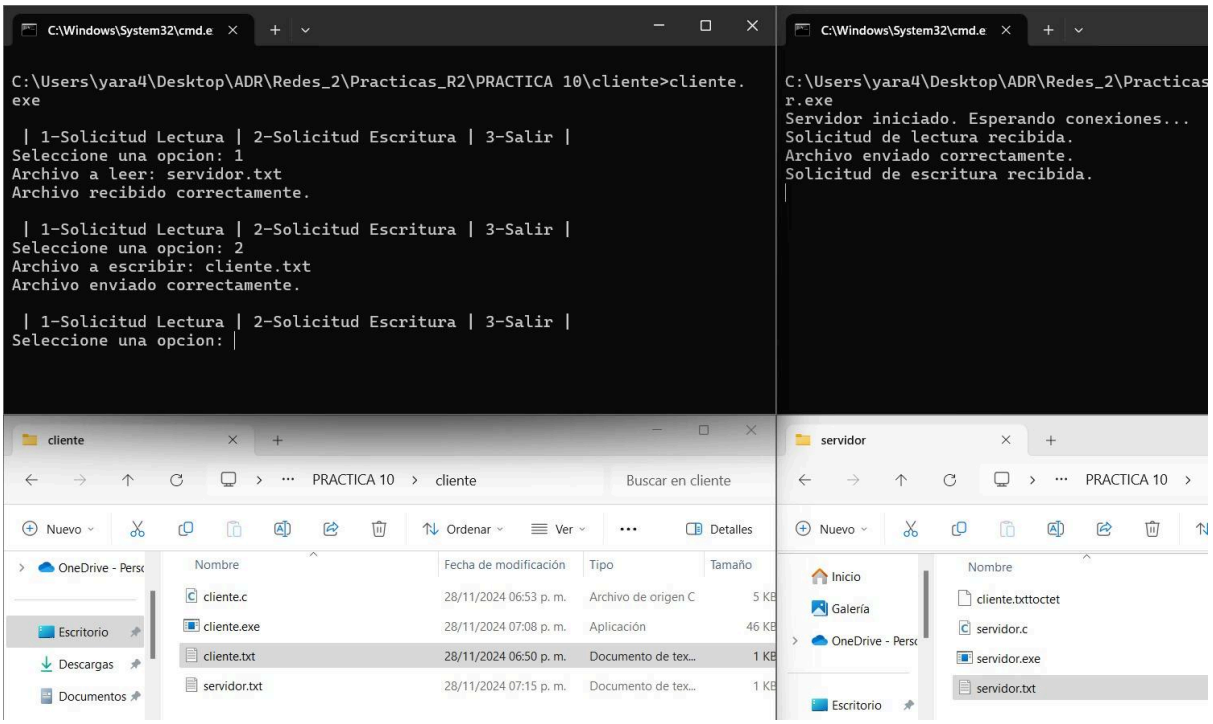

C:\Windows\System32\cmd.e
C:\Users\yara4\Desktop\ADR\Redes_2\Practicas_R2\PRACTICA 10\cliente>cliente.exe
| 1-Solicitud Lectura | 2-Solicitud Escritura | 3-Salir |
Seleccione una opcion:


C:\Windows\System32\cmd.e
C:\Users\yara4\Desktop\ADR\Redes_2\Practicas_R2\PRACTICA 10\servidor>servidor.exe
Servidor iniciado. Esperando conexiones...


cliente
PRACTICA 10
cliente
Buscar en cliente
Nuevo
Ordenar
Ver
Detalles
OneDrive - Perso
Escritorio
Descargas
Documentos
Imágenes
Música
Nombre
Fecha de modificación
Tipo
Tamaño
cliente.c
28/11/2024 06:53 p. m.
Archivo de origen C
5 K
cliente.exe
28/11/2024 07:08 p. m.
Aplicación
46 K
cliente.txt
28/11/2024 06:50 p. m.
Documento de tex...
1 K


servidor
PRACTICA 10
servidor
Buscar en servidor
Nuevo
Ordenar
Ver
Detalles
Inicio
Galería
OneDrive - Perso
Escritorio
Nombre
Fecha de modificación
Tipo
Tamaño
servidor.c
26/11/2024 05:58 p. m.
Archivo de origen C
5 KB
servidor.exe
28/11/2024 07:08 p. m.
Aplicación
46 KB
servidor.txt
28/11/2024 07:04 p. m.
Documento de tex...
1 KB


C:\Windows\System32\cmd.e
C:\Users\yara4\Desktop\ADR\Redes_2\Practicas_R2\PRACTICA 10\cliente>cliente.exe
| 1-Solicitud Lectura | 2-Solicitud Escritura | 3-Salir |
Seleccione una opcion: 1
Archivo a leer: servidor.txt
Archivo recibido correctamente.
| 1-Solicitud Lectura | 2-Solicitud Escritura | 3-Salir |
Seleccione una opcion:
C:\Windows\System32\cmd.e
C:\Users\yara4\Desktop\ADR\Redes_2\Practicar.exe
Servidor iniciado. Esperando conexiones...
Solicitud de lectura recibida.
Archivo enviado correctamente.
cliente
PRACTICA 10 › cliente
Buscar en cliente
Nuevo
Ordenar
Ver
Detalles
OneDrive - Perso
Escritorio
Descargas
Documentos
Nombre
Fecha de modificación
Tipo
Tamaño
cliente.c
28/11/2024 06:53 p. m.
Archivo de origen C
5 KB
cliente.exe
28/11/2024 07:08 p. m.
Aplicación
46 KB
cliente.txt
28/11/2024 06:50 p. m.
Documento de tex...
1 KB
servidor.txt
28/11/2024 07:15 p. m.
Documento de tex...
1 KB
servidor
PRACTICA 10
Inicio
Galería
OneDrive - Perso
Escritorio
Nombre
servidor.c
servidor.exe
servidor.txt


C:\Windows\System32\cmd.e
C:\Users\yara4\Desktop\ADR\Redes_2\Practicas_R2\PRACTICA 10\cliente>cliente.exe
| 1-Solicitud Lectura | 2-Solicitud Escritura | 3-Salir |
Seleccione una opcion: 1
Archivo a leer: servidor.txt
Archivo recibido correctamente.
| 1-Solicitud Lectura | 2-Solicitud Escritura | 3-Salir |
Seleccione una opcion: 2
Archivo a escribir: cliente.txt
Archivo enviado correctamente.
| 1-Solicitud Lectura | 2-Solicitud Escritura | 3-Salir |
Seleccione una opcion:
C:\Windows\System32\cmd.e
C:\Users\yara4\Desktop\ADR\Redes_2\Practicasr.exe
Servidor iniciado. Esperando conexiones...
Solicitud de lectura recibida.
Archivo enviado correctamente.
Solicitud de escritura recibida.
cliente
PRACTICA 10 › cliente
Buscar en cliente
Nuevo
Ordenar
Ver
Detalles
OneDrive - Perso
Escritorio
Descargas
Documentos
Nombre
Fecha de modificación
Tipo
Tamaño
cliente.c
28/11/2024 06:53 p. m.
Archivo de origen C
5 KB
cliente.exe
28/11/2024 07:08 p. m.
Aplicación
46 KB
cliente.txt
28/11/2024 06:50 p. m.
Documento de tex...
1 KB
servidor.txt
28/11/2024 07:15 p. m.
Documento de tex...
1 KB
servidor
PRACTICA 10
Inicio
Galería
OneDrive - Perso
Escritorio
Nombre
cliente.txttoctet
servidor.c
servidor.exe
servidor.txt

CONCLUSIONES

Navarrete Becerril Sharon Anette:

Similitud entre cliente y servidor: La lógica del servidor TFTP comparte muchas similitudes con la del cliente, lo que facilita su desarrollo y mantenimiento. Sin embargo, se requieren ajustes específicos para manejar las solicitudes de lectura y escritura correctamente, destacando la flexibilidad de TFTP para adaptarse a diferentes necesidades de transferencia.

BIBLIOGRAFÍA

Stevens, W. Richard. Programación en red con Unix: La API de Sockets. Addison-Wesley Professional, 2003.

Forouzan, Behrouz A. Comunicaciones de Datos y Redes. McGraw-Hill, 2012. Tanenbaum, Andrew S., y David J. Wetherall. Redes de Computadoras. Pearson, 2010.

Comer, Douglas E. Internetworking con TCP/IP Volumen Uno. Prentice Hall, 2006. Kurose, James F., y Keith W. Ross. Redes de Computadoras: Un Enfoque Descendente. Pearson, 2017.

Beazley, David, y Brian K. Jones. Python Cookbook: Recetas para Dominar Python 3. O'Reilly Media, 2013. Begg, A. Sockets TCP/IP en C: Guía Práctica para Programadores. Morgan Kaufmann, 2000