## Protocolo TFTP

El Protocolo de Transferencia de Archivos Trivial (TFTP, por sus siglas en inglés) es un protocolo simple para la transferencia de archivos a través de una red. Se basa en el protocolo UDP (User Datagram Protocol) para transmitir datos, lo que lo hace ligero y adecuado para redes con requisitos bajos en términos de control y fiabilidad.

TFTP permite la transferencia de archivos entre un cliente y un servidor de manera eficiente y sencilla. Su diseño minimalista facilita su implementación en dispositivos con recursos limitados, como routers o switches. A diferencia de otros protocolos de transferencia de archivos como FTP o HTTP, TFTP no proporciona muchas de las características avanzadas de estos protocolos, como autenticación o de cifrado.

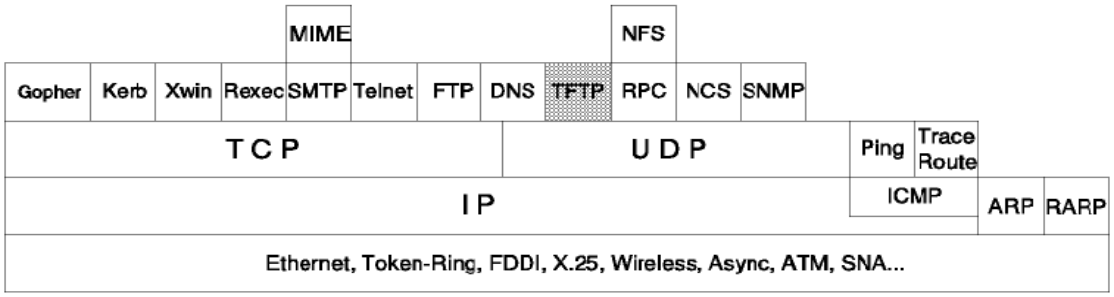
**Características del TFTP**

- **Simplicidad:** Diseño simple y fácil de implementar.
- **Uso de UDP:** No ofrece confirmación de entrega, lo que lo hace menos confiable en comparación con TCP.
- **Transferencia en Bloques:** Los archivos se transfieren en bloques de tamaño fijo.
- **Sin Seguridad:** No incluye características de autenticación o cifrado.

El protocolo TFTP está definido en el RFC 1350, que describe el proceso de transferencia de archivos en TFTP. En TFTP, los archivos se transfieren en bloques, y el protocolo utiliza un esquema de confirmación de paquetes de datos a través de ACK (Acknowledgment).

**Proceso de transferencia:**

- **Inicio de Transferencia:** El cliente TFTP inicia la transferencia enviando una solicitud de lectura (RRQ) o escritura (WRQ) al servidor TFTP. La solicitud incluye el nombre del archivo y el modo de transferencia (por ejemplo, "netascii" o "octet").
- **Envío de Datos (Servidor a Cliente):** En el caso de una solicitud de lectura, el servidor TFTP envía el archivo en bloques de datos. Cada bloque de datos está precedido por un número de bloque único.
- **Confirmación de Datos (Cliente a Servidor):** El cliente responde a cada bloque de datos recibido con un paquete de confirmación (ACK) que indica el número de bloque que se está confirmando.
- **Terminación de la Transferencia:** La transferencia finaliza cuando se envía un bloque de datos con tamaño menor al máximo permitido (generalmente 512 bytes), lo que indica que se ha llegado al final del archivo.

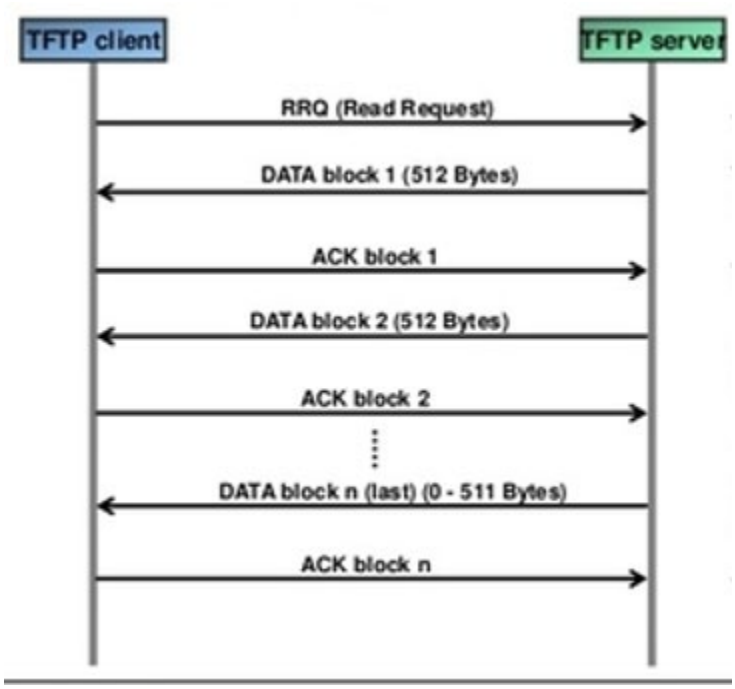


**Desarrollo:**

La finalidad de esta práctica es 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.

Se comenzará creando en tu entorno de desarrollo de preferencia dos archivos, uno será para el cliente y uno para el servidor:

- Cliente.c
- Servidor.c




Para replicar la comunicación como la que se muestra en la imagen anterior donde se realiza una petición de parte de un cliente a un servidor.

De manera breve el funcionamiento del código consiste en crear un socket UDP con socket(AF_INET, SOCK_DGRAM, 0) y realizar la configuración inicial del socket local y remoto, incluyendo la vinculación con bind y la configuración de la dirección del servidor remoto usando inet_addr. Dependiendo de la opción seleccionada por el usuario, el programa puede realizar una solicitud de lectura o escritura de un archivo. En el caso de la lectura, el cliente envía una solicitud de lectura al servidor TFTP y, si la solicitud es exitosa, el archivo es recibido en bloques y guardado localmente. La recepción se realiza en un bucle que permite recibir bloques de datos hasta completar la transferencia o alcanzar un límite de 10 MB. En el caso de la escritura, el archivo se lee en bloques de 512 bytes y se envía al servidor TFTP, con el cliente esperando un ACK (Acknowledgment) del servidor después de cada bloque enviado para continuar con el siguiente. Durante la transferencia, se realiza un manejo de errores básico para gestionar problemas como la no recepción de ACKs. El código también incluye funciones para estructurar peticiones de lectura y escritura, enviar ACKs, y manejar la estructura de datos a enviar o recibir.

Inicialmente se va a establecer la lógica para poder desarrollar el cliente, por lo que se agrega el siguiente fragmento de código en el archivo cliente.c:

```
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <string.h>

int timeEspera = 1000;
struct timeval start, end;
long mtime, seconds, useconds;

short numPaq;
unsigned char numPaqHex[2];

int udp_socket, dbind, tamEnviado, tamTotal=0, tamParcial, lrecv, tamRecivido,
tamData;
unsigned char data[516], estructData[516], estrPeticionLectura[516],
estrPeticionEscritura[516], estrACK[4], mensajeRecivido[516], nomArch[30];
unsigned char numPaqTemp[2]= {0x00,0x01};
struct sockaddr_in local, remota, cliente;
FILE *fw, *fr;

void EnviarACK (); //Envia el ACK al receptor
int EsperarACK (unsigned char *paq, int tam); //Espera el ACK del receptor
int EstructuraDatos(unsigned char *paq, int tam); //Estructura la peticion de
esscritura
int EstructuraACK (unsigned char *paq); //Estructura el ACK a envair
int PeticionLectura (); //Envia la peticion de lectura y espera a que llegue el
ACK del recptor
int EstructuraPeticionLectura (unsigned char *paq, unsigned char *nomArch);
//Estructura la peticion de lectura
int PeticionEscritura (); //Envia la peticion de Escritura y espera a que llegue
el ACK del receptor
int EstructuraPeticionEscritura (unsigned char *paq, unsigned char *nomArch);
//Estructura la peticion de escritura

int main () {
    int opcion;
    udp_socket = socket(AF_INET, SOCK_DGRAM, 0);
    if (udp_socket==-1) { perror("\nError al abrir el socket"); exit(0); }

    else {
        local.sin_family= AF_INET;
        local.sin_port= htons(0);
        local.sin_addr.s_addr= INADDR_ANY;
        dbind= bind(udp_socket, (struct sockaddr *)&local, sizeof(local));
        if(dbind == -1){ perror("\nError en bind"); exit(0); }

        else{
```

```
            remota.sin_family= AF_INET;
            remota.sin_port= htons(69);
            remota.sin_addr.s_addr=inet_addr("tu_ip");
            int opcion = 0, limpiar;
            while (opcion!=3) {
                printf("\n          | 1-Solicitud Lectura | 2-Solicitud Escritura
|     \n");
                scanf("%i",&opcion);
                do{ limpiar = getchar(); }while(limpiar != EOF && limpiar !=
'\n');
                switch (opcion) {
                    case 1:
                        printf("Archivo a leer: ");
                        fgets(nomArch, 30, stdin);
                        strtok(nomArch, "\n");
                        int tamData = PeticionLectura();

                        if (tamData == 3) {
                            tamTotal=0;
                            fw = fopen(nomArch, "wb");
                            fwrite(mensajeRecivido+4, 1, tamRecivido-4, fw);
                            tamTotal += tamRecivido - 4;
                            if ( tamRecivido < 512 ) {
                                EnviarACK();
                                printf("Peso del Archivo: %i bytes\n", tamTotal);
                                fclose(fw);
                                break;
                            }
                            else {
                                EnviarACK();
                                lrecv= sizeof(cliente);
                                while (tamTotal <= 10 * 1024 * 1024) { // Limitar
a 10 MB
                                    tamRecivido = recvfrom(udp_socket,
mensajeRecivido, sizeof(mensajeRecivido), MSG_DONTWAIT, (struct sockaddr
*)&cliente, &lrecv);
                                    if(tamRecivido == -1){}

                                    else{
                                        tamTotal += tamRecivido - 4;
                                        if (tamRecivido >= 516) {
                                            if(mensajeRecivido[2] == estrACK[2]
&& mensajeRecivido[3] == estrACK[3]) {}
                                            else { fwrite(mensajeRecivido+4, 1,
tamRecivido-4, fw); }
                                            EnviarACK();
                                        }
                                        if ( tamRecivido < 516 || tamTotal > 10 *
1024 * 1024 ) { // Parar si se alcanza el límite de 10 MB
                                            fwrite(mensajeRecivido+4, 1,
tamRecivido-4, fw);
                                            printf("Peso del Archivo: %i
bytes\n", tamTotal);
                                            EnviarACK();
                                            break;
                                        }
```

```
                                    }

                                }
                            printf("              ----Recivido----
\n\n");
                            fclose(fw);
                            break;
                            }
                        }

                        if (tamData == 5) { printf("Archivo no existe\n"); }

                        break;

                    case 2:
                        printf("Archivo a escribir: ");
                        fgets(nomArch, 30, stdin);
                        strtok(nomArch, "\n");
                        fr = fopen(nomArch, "rb");
                        if(fr == NULL) { printf("No se pudo abrir\n");}
                        else {
                            int tamData = PeticionEscritura();
                            if (tamData == 4) {
                                tamTotal=0;
                                numPaq=1;
                                int estr;
                                while (!feof(fr) && tamTotal <= 10 * 1024 * 1024)
{ // Limitar a 10 MB
                                    tamParcial = fread(data, 1, 512, fr);
                                    estr = EstructuraDatos(estructData,
tamParcial);
                                    tamTotal += tamParcial;
                                    tamEnviado= sendto(udp_socket, estructData,
estr, 0, (struct sockaddr *)&cliente, sizeof(cliente));
                                    if(tamEnviado == -1){perror("\nError al
enviar");exit(0);}
                                    else {
                                        numPaq += 1;
                                        printf("Enviados: %i bytes\n",
tamEnviado-4);

                                        int resp;
                                        resp = EsperarACK(estructData, estr);
                                        if(resp == 4) {/* printf("4\n");*/ } else
{printf("%i\n",resp); break;}
                                    }
                                }
                                printf("     ---Total: %i bytes---
\n\n",tamTotal);
                            }

                            if (tamData == 5) { printf("Archivo no pudo ser\n");
}

                            fclose(fr);
                        }
                        break;
```

```
                    default:
                        close(udp_socket);
                        break;
                }
            }

        }
    }
}
void EnviarACK () {
    int tamACK;
    tamACK = EstructuraACK (estrACK);
    tamEnviado= sendto(udp_socket, estrACK, tamACK, 0, (struct sockaddr
*)&cliente, sizeof(cliente));
    if (tamEnviado == -1) { printf("Error al enviar ACK"); }
    else { }
}

int EstructuraACK (unsigned char *paq) {
    unsigned char codOP[2] = {0x00,0x04};
    numPaqHex[0] = mensajeRecivido[2];
    numPaqHex[1] = mensajeRecivido[3];
    memcpy(paq+0,codOP,2);
    memcpy(paq+2, numPaqHex, 2);
    return 4;
}

int EstructuraPeticionLectura (unsigned char *paq, unsigned char *nomArch) {
    for (int i = 0; i < 516; i++) { paq[i]=0; }
    unsigned char opLec[2] = {0x00,0x01};
    unsigned char modo[] = "octet";
    memcpy(paq+0,opLec,2);
    memcpy(paq+2, nomArch, strlen(nomArch));
    memcpy(paq+(strlen(nomArch)+3), modo, strlen(modo)+1);
    return (strlen(nomArch) + 3 + strlen(modo) + 1);
}

int PeticionLectura () {
    long mtime=0;
    int tamData, intentos=0;
    tamData = EstructuraPeticionLectura (estrPeticionLectura, nomArch);
    tamEnviado= sendto(udp_socket, estrPeticionLectura, tamData, 0, (struct
sockaddr *)&remota, sizeof(remota));
    if(tamEnviado == -1){perror("\nError en enviar"); exit(0);}
    else{
        lrecv= sizeof(cliente);
        gettimeofday(&start, NULL);
        while (mtime<timeEspera) {
            tamRecivido= recvfrom(udp_socket, mensajeRecivido,
sizeof(mensajeRecivido), MSG_DONTWAIT, (struct sockaddr *)&cliente, &lrecv);
            if (tamRecivido == -1) {}
            else{
                if (mensajeRecivido[1]== 0x05) {return 5;}
                if (mensajeRecivido[1]== 0x03) {return 3;}
            }
```

```
            gettimeofday(&end, NULL);
            seconds = end.tv_sec -start.tv_sec;
            useconds = end.tv_usec -start.tv_usec;
            mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
            if (mtime>=timeEspera) {
                if(intentos >= 5) {return intentos;}
                intentos++;
                mtime=0;
                gettimeofday(&start, NULL);
                tamEnviado= sendto(udp_socket, estrPeticionLectura, tamData, 0,
(struct sockaddr *)&remota, sizeof(remota));
            }
        }
    }
}

int EstructuraPeticionEscritura (unsigned char *paq, unsigned char *nomArch) {
    unsigned char opLec[2] = {0x00,0x02};
    unsigned char modo[] = "octet";
    memcpy(paq+0,opLec,2);
    memcpy(paq+2, nomArch, strlen(nomArch));
    memcpy(paq+(strlen(nomArch)+3), modo, strlen(modo)+1);
    return (strlen(nomArch) + 3 + strlen(modo) + 1);
}

int PeticionEscritura() {
    long mtime=0;
    int tamData, intentos=0;
    tamData = EstructuraPeticionEscritura (estrPeticionEscritura, nomArch);
    tamEnviado= sendto(udp_socket, estrPeticionEscritura, tamData, 0, (struct
sockaddr *)&remota, sizeof(remota));
    if(tamEnviado == -1){perror("\nError en enviar"); exit(0);}
    else{
        lrecv= sizeof(cliente);
        gettimeofday(&start, NULL);
        while (mtime<timeEspera) {
            tamRecivido= recvfrom(udp_socket, mensajeRecivido,
sizeof(mensajeRecivido), MSG_DONTWAIT, (struct sockaddr *)&cliente, &lrecv);
            if (tamRecivido == -1) {}
            else{
                if (mensajeRecivido[1]== 0x04) {return 4;}
                if (mensajeRecivido[1]== 0x03) {return 3;}
            }
            gettimeofday(&end, NULL);
            seconds = end.tv_sec -start.tv_sec;
            useconds = end.tv_usec -start.tv_usec;
            mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
            if (mtime>=timeEspera) {
                if(intentos >= 5) {return intentos;}
                intentos++;
                mtime=0;
                gettimeofday(&start, NULL);
                tamEnviado= sendto(udp_socket, estrPeticionEscritura, tamData, 0,
(struct sockaddr *)&remota, sizeof(remota));
            }
        }
```

```
    }
}

int EstructuraDatos(unsigned char *paq, int tam) {
    unsigned char opLec[2] = {0x00,0x03};
    numPaqHex[0] = numPaq >> 8;
    numPaqHex[1] = numPaq & 0X00FF;
    // unsigned short numOfBlocks = htons(((numPaqA[1] << 8) &0xFF00) |
(numPaqA[0] & 0xFF));
    memcpy(paq+0,opLec,2);
    memcpy(paq+2, numPaqHex, 2);
    memcpy(paq+4, data, tam);
    return (4+tam);
}

int EsperarACK (unsigned char *paq, int tam) {
    long mtime=0;
    int intentos=0;
    lrecv= sizeof(cliente);
    gettimeofday(&start, NULL);
    while (mtime<timeEspera) {
        tamRecivido= recvfrom(udp_socket, mensajeRecivido,
sizeof(mensajeRecivido), MSG_DONTWAIT, (struct sockaddr *)&cliente, &lrecv);
if (tamRecivido == -1) {}
        if(tamRecivido==-1){}
        else{
            if (mensajeRecivido[1]== 0x04 && mensajeRecivido[2]== numPaqHex[0] &&
mensajeRecivido[3]== numPaqHex[1]) {return 4;}
            if (mensajeRecivido[1]== 0x05) {return 5;}
        }
        gettimeofday(&end, NULL);
        seconds = end.tv_sec -start.tv_sec;
        useconds = end.tv_usec -start.tv_usec;
        mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
        if (mtime>=timeEspera) {
            if(intentos >= 10) {return intentos;}
            printf("Volvio a enviar, tiempo: %li",mtime);
            intentos++;
            mtime=0;
            gettimeofday(&start, NULL);
            tamEnviado= sendto(udp_socket, estructData, tam, 0, (struct sockaddr
*)&cliente, sizeof(cliente));
        }
    }
}
```

Para el servidor se implementa el funcionamiento de transferencia de archivos utilizando el protocolo TFTP. Utiliza varias bibliotecas estándar de C para manejar la creación y manipulación de sockets, estructuras de datos de red, y operaciones básicas de entrada y salida.

El servidor crea un socket UDP utilizando socket(AF_INET, SOCK_DGRAM, 0) y lo enlaza a una dirección IP y puerto local con bind. Se configura para escuchar en el puerto 69, que es el puerto estándar de TFTP. En el bucle principal, el servidor espera recibir mensajes de clientes. Estos mensajes pueden ser solicitudes de lectura o escritura de archivos, que están indicadas por el segundo byte del mensaje (mensajeRecivido[1]).

Cuando se recibe una solicitud de lectura (mensajeRecivido[1] == 0x01), el servidor intenta abrir el archivo solicitado en modo de lectura binaria (fopen(mensajeRecivido+2, "rb")). Si el archivo no puede ser abierto, el servidor envía un mensaje de error al cliente. Si el archivo es accesible, el servidor lo lee en bloques de hasta 512 bytes, los empaqueta en un mensaje TFTP utilizando la función EstructuraDatos, y los envía al cliente. Después de enviar cada bloque, el servidor espera recibir un ACK (Acknowledgment) del cliente confirmando la recepción del bloque. Si no se recibe un ACK dentro de un tiempo específico, el servidor reenvía el bloque.

Para las solicitudes de escritura (mensajeRecivido[1] == 0x02), el servidor crea un archivo nuevo en modo de escritura binaria (fopen(mensajeRecivido+2, "wb")). Si el archivo no puede ser creado, se envía un mensaje de error al cliente. Si el archivo es creado exitosamente, el servidor envía un ACK inicial al cliente indicando que está listo para recibir datos. Luego, en un bucle, recibe bloques de datos del cliente, los escribe en el archivo y envía un ACK por cada bloque recibido. El proceso se repite hasta que se recibe un bloque más pequeño de lo normal, indicando que la transferencia ha concluido.

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:

```
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <string.h>

int timeEspera = 5000;
struct timeval start, end;
long mtime, seconds, useconds;

short numPaq;
unsigned char numPaqHex[2];

int udp_socket, lbind, tamRecivido, lrecv, tamEnviado, tamParcial, tamTotal;
struct sockaddr_in servidor, cliente;
unsigned char data[516], estrACK[4], mensajeRecivido[516], estructData[516];
FILE *fw, *fr;

void EnviarACK (int numPaq);  //Envia el ACK al receptor
int EsperarACK(unsigned char *paq, int tam); //Espera el ACK del receptor
```

```
int EstructuraACKInicial (unsigned char *paq); //Estructura el ACK para el inicio
de la comunicacion
int EstructuraDatos(unsigned char *paq, int tam); //Estructura los datos a enviar
int EstructuraError(unsigned char *paq);  //Estructura la respuesta de error
int EstructuraACK (unsigned char *paq); //Estructura el ACK

int main () {

    udp_socket = socket(AF_INET, SOCK_DGRAM, 0);
    if (udp_socket == -1) {
        perror("\nError al abrir el socket");
        exit(0);
    }
    else {
        perror("Exito al abrir el socket");
        servidor.sin_family= AF_INET;
        servidor.sin_port= htons(69);
        servidor.sin_addr.s_addr= INADDR_ANY;
        lbind= bind(udp_socket, (struct sockaddr *)&servidor, sizeof(servidor));
        if (lbind == -1) {
             perror("\nError en bind");
              exit(0);
        }
        else {
            perror("Exito en bind");
            printf("\n            !** Servidor Encendido **!            \n");
            lrecv= sizeof(cliente);
            while(1){
                tamRecivido= recvfrom(udp_socket, mensajeRecivido,
sizeof(mensajeRecivido), 0, (struct sockaddr *)&cliente, &lrecv);
                if(tamRecivido==-1){}
                else {
                    if (mensajeRecivido[1]==0x01) {
                        printf("Peticion de lectura\n");
                        printf("nombre arch: %s\n",mensajeRecivido+2);
                        int tamError, estr;
                        fr = fopen(mensajeRecivido+2, "rb");
                        if(fr == NULL) {
                            printf("Nose pudo leer\n");
                            tamError = EstructuraError(estructData);
                            tamEnviado= sendto(udp_socket, estructData, tamError,
0, (struct sockaddr *)&cliente, sizeof(cliente));
                        }
                        else {
                            tamTotal=0;
                            numPaq=1;
                            while (!feof(fr)) {
                                tamParcial = fread(data, 1, 512, fr);
                                estr = EstructuraDatos(estructData, tamParcial);
                                tamTotal += tamParcial;
                                tamEnviado= sendto(udp_socket, estructData, estr,
0, (struct sockaddr *)&cliente, sizeof(cliente));
                                if(tamEnviado == -1){perror("\nError al
enviar");exit(0);}
                                else {
                                    numPaq += 1;
```

```
                                    printf("Enviados: %i bytes\n", tamEnviado-4);
                                    int resp;
                                    resp = EsperarACK(estructData, estr);
                                    if(resp == 4) {/* printf("4\n");*/ } else
{printf("%i\n",resp); break;}
                                }
                            }
                            printf("     ---Total: %i bytes---    \n\n",tamTotal);
                            fclose(fr);
                        }

                    }

                    if (mensajeRecivido[1]==0x02) {
                        printf("Peticion de Escritura\n");
                        printf("nombre arch: %s\n",mensajeRecivido+2);
                        int tamError, estr;
                        fw = fopen(mensajeRecivido+2, "wb");
                        if(fw == NULL) {
                            printf("Nose pudo escribir\n");
                            tamError = EstructuraError(estructData);
                            tamEnviado= sendto(udp_socket, estructData, tamError,
0, (struct sockaddr *)&cliente, sizeof(cliente));
                        }
                        else {
                            tamTotal=0;
                            numPaq=0;
                            EnviarACK (0);
                            while (1) {
                                    tamRecivido = recvfrom(udp_socket,
mensajeRecivido, sizeof(mensajeRecivido), MSG_DONTWAIT, (struct sockaddr
*)&cliente, &lrecv);
                                    if(tamRecivido == -1){}

                                    else{
                                        EnviarACK(1);
                                        tamTotal += tamRecivido - 4;
                                        if (tamRecivido >= 516) {
fwrite(mensajeRecivido+4, 1, tamRecivido-4, fw); }
                                        if ( tamRecivido < 516 ) {
                                            fwrite(mensajeRecivido+4, 1,
tamRecivido-4, fw);
                                            // printf("Peso del Archivo: %i
bytes\n", tamTotal);
                                            printf("     ---Total: %i bytes---
\n\n",tamTotal);
                                            break;
                                        }
                                    }

                                }
                            fclose(fw);
                        }

                    }
                }
```

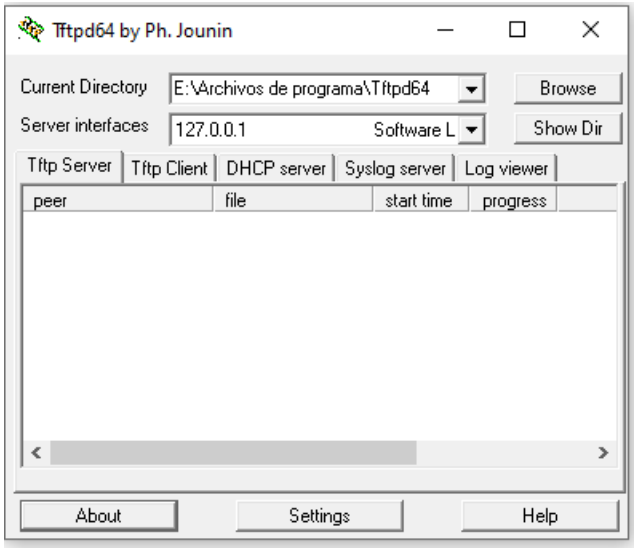
```
            }
            close(udp_socket);
        }
    }
}

int EsperarACK (unsigned char *paq, int tam) {
    long mtime=0;
    int intentos=0;
    lrecv= sizeof(cliente);
    gettimeofday(&start, NULL);
    while (mtime<timeEspera) {
        tamRecivido= recvfrom(udp_socket, mensajeRecivido,
sizeof(mensajeRecivido), MSG_DONTWAIT, (struct sockaddr *)&cliente, &lrecv);
if (tamRecivido == -1) {}
        if(tamRecivido==-1){}
        else{
            if (mensajeRecivido[1]== 0x04 && mensajeRecivido[2]== numPaqHex[0] &&
mensajeRecivido[3]== numPaqHex[1]) {return 4;}
            if (mensajeRecivido[1]== 0x05) {return 5;}
        }
        gettimeofday(&end, NULL);
        seconds = end.tv_sec -start.tv_sec;
        useconds = end.tv_usec -start.tv_usec;
        mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
        if (mtime>=timeEspera) {
            if(intentos >= 10) {return intentos;}
            printf("Volvio a enviar, tiempo: %li",mtime);
            intentos++;
            mtime=0;
            gettimeofday(&start, NULL);
            tamEnviado= sendto(udp_socket, estructData, tam, 0, (struct sockaddr
*)&cliente, sizeof(cliente));
        }
    }
}

void recivirMensaje(){
    fw = fopen("Recibido", "ab");
    if(fw==NULL){perror("Error al abrir archivo"); exit(1);}
    lrecv = sizeof(cliente);
    while (1){
        tamRecivido= recvfrom(udp_socket, data, 512, MSG_DONTWAIT, (struct
sockaddr *)&cliente, &lrecv);
        if (tamRecivido==-1) {}
        if ( (tamRecivido!=-1 && tamRecivido<512) ||  tamRecivido == 0) {
            if (data[0]== 0x00 && data[1]== 0x01) {
                printf("Codigo: %.2x%.2x\n", data[0], data[1]);
                printf("Nombre de archivo: %s\n",data+2);
                printf("Modo: %s\n",data+(strlen(data+2)+3));
                tamEnviado= sendto(udp_socket,"ok", 2, 0, (struct sockaddr
*)&cliente, sizeof(cliente));
            }
            printf("Recibido: %i\n",tamRecivido);
            printf("\nTransmicion Completa\n");
            break;
```

```
        }
        if (tamRecivido>=512) {
            printf("Recibido: %i\n",tamRecivido);
            fwrite(data, 1, tamRecivido, fw);
        }
    }
    fclose(fw);
}

int EstructuraDatos(unsigned char *paq, int tam) {
    unsigned char opLec[2] = {0x00,0x03};
    numPaqHex[0] = numPaq >> 8;
    numPaqHex[1] = numPaq & 0X00FF;
    // unsigned short numOfBlocks = htons(((numPaqA[1] << 8) &0xFF00) |
(numPaqA[0] & 0xFF));
    memcpy(paq+0,opLec,2);
    memcpy(paq+2, numPaqHex, 2);
    memcpy(paq+4, data, tam);
    return (4+tam);
}

int EstructuraError(unsigned char *paq) {
    unsigned char opLec[2] = {0x00,0x05};
    unsigned char codError[2] = {0x00,0x01};
    memcpy(paq+0,opLec,2);
    memcpy(paq+2, codError, 2);
    memcpy(paq+4, "Error", 5);
    return (9);
}

void EnviarACK (int numPaq) {
    int tamACK;
    if(numPaq == 0) {
        tamACK = EstructuraACKInicial(estrACK);
        tamEnviado= sendto(udp_socket, estrACK, tamACK, 0, (struct sockaddr
*)&cliente, sizeof(cliente));
        if (tamEnviado == -1) { printf("Error al enviar ACK"); }
        else { }
    }
    else {
        tamACK = EstructuraACK (estrACK);
        tamEnviado= sendto(udp_socket, estrACK, tamACK, 0, (struct sockaddr
*)&cliente, sizeof(cliente));
        if (tamEnviado == -1) { printf("Error al enviar ACK"); }
        else { }
    }
}

int EstructuraACKInicial (unsigned char *paq) {
    unsigned char codOP[2] = {0x00,0x04};
    unsigned char numPaqHex[2] = {0x00,0x00};
    memcpy(paq+0,codOP,2);
    memcpy(paq+2, numPaqHex, 2);
    return 4;
}
```

```
int EstructuraACK (unsigned char *paq) {
    unsigned char codOP[2] = {0x00,0x04};
    numPaqHex[0] = mensajeRecivido[2];
    numPaqHex[1] = mensajeRecivido[3];
    memcpy(paq+0,codOP,2);
    memcpy(paq+2, numPaqHex, 2);
    return 4;
}
```

**Pruebas:**

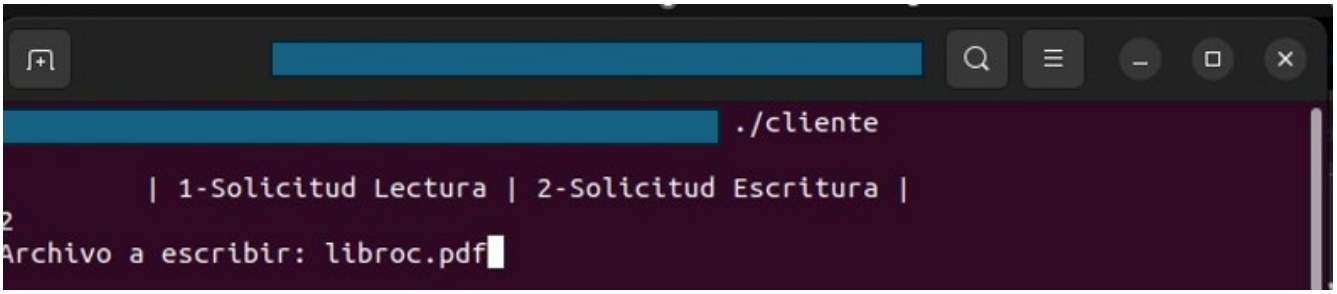
Ahora se realizarán las pruebas ejecutando tu cliente o servidor y haciendo uso de una aplicación de escritorio llamada tftpd64, donde se puede tener el rol de cliente o de servidor:

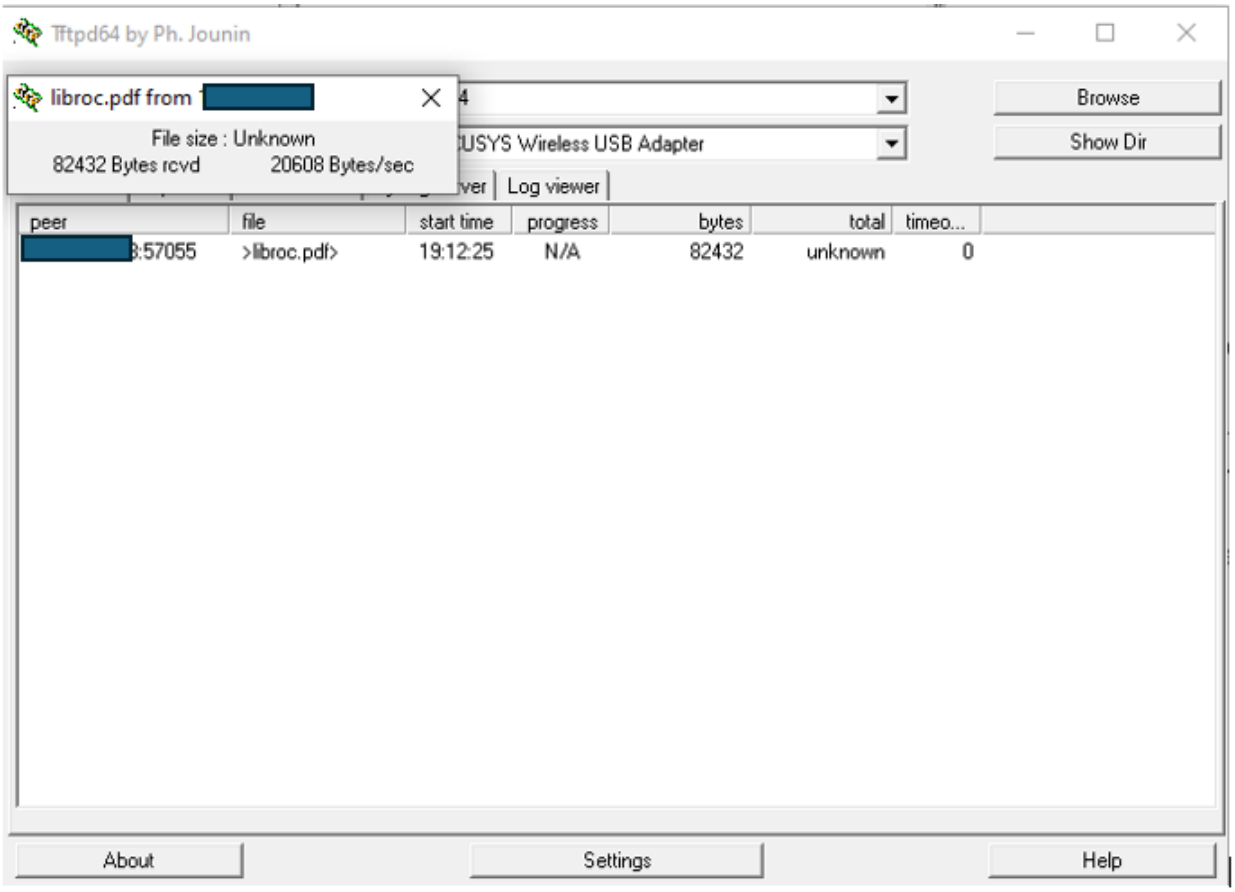


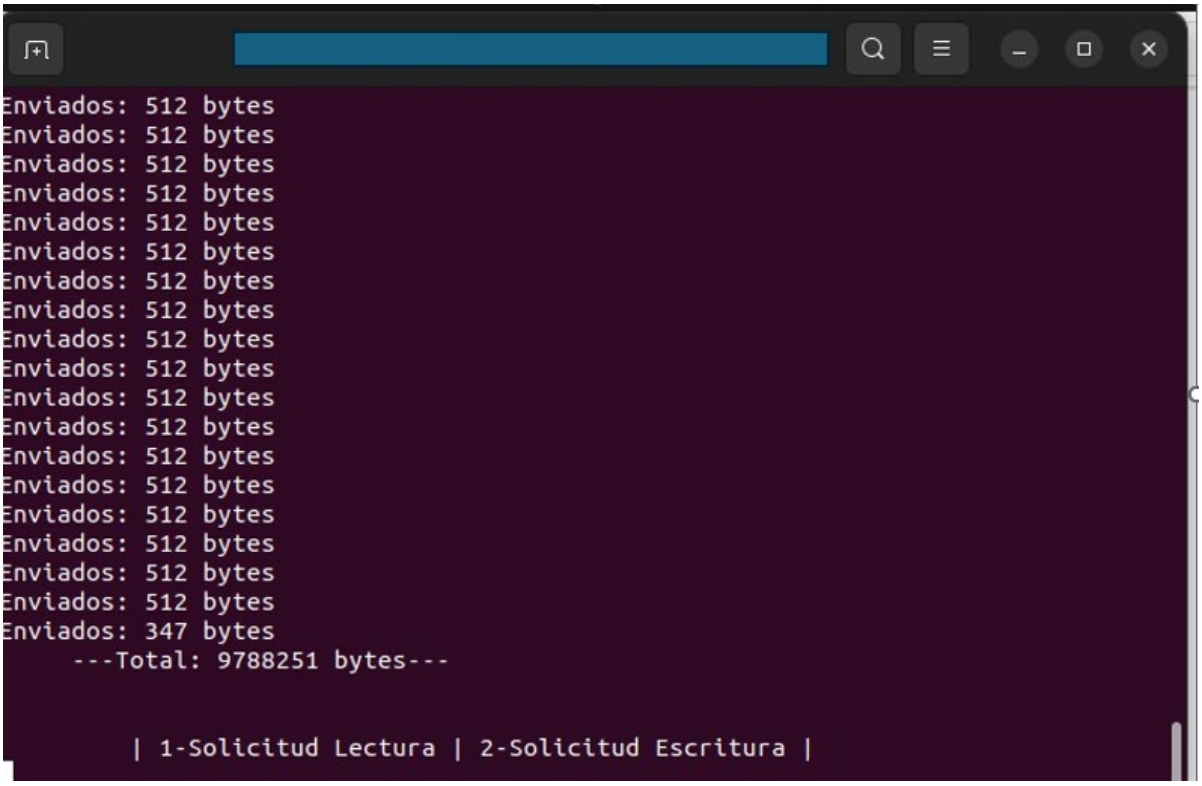

Para realizar la prueba, en tu entorno de desarrollo compila tu cliente y tu servidor como se muestra a continuación:

| | |
|---|---|
| gcc cliente.c -o cliente | gcc servidor.c -o servidor |

Ahora para el caso del cliente se va a enviar un archivo que se encuentre en nuestra PC pero en el servidor que en este caso será el tftpd64 se deben de configurar para que apunte a tu dirección IP, como prueba se va a mandar un archivo de tu elección en este caso mandare un libro en formato PDF:




Tftpd64 by Ph. Jounin
libroc.pdf from [redacted]
File size : Unknown
82432 Bytes rcvd
20608 Bytes/sec
Browse
Show Dir
USYS Wireless USB Adapter
Log viewer
peer
file
start time
progress
bytes
total
timeo...
[redacted]:57055
>libroc.pdf>
19:12:25
N/A
82432
unknown
0
About
Settings
Help


Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 512 bytes
Enviados: 347 bytes
---Total: 9788251 bytes---
| 1-Solicitud Lectura | 2-Solicitud Escritura |


libroc.pdf - Adobe Acrobat Pro DC
Archivo Edición Ver Ventana Ayuda
Inicio Herramientas
libroc.pdf
Con base en el ANSI C
SEGUNDA EDICION
EL
LENGUAJE DE
PROGRAMACION