**UNIVERSIDAD AUTONOMA METROPOLITANA DOMINGO 30 DE MAYO DEL 2021**

**UNIDAD AZCAPOTZALCO**

**SISTEMAS OPERATIVOS**

**JOSE ENRIQUE GARCIA RAMIREZ SINCRONIZACION DE ARCHIVOS REMOTOS**

**INTRODUCCION**

El manejo de los procesos es importante en los sistemas operativos (SO). Si todas las instrucciones se manejaran en un solo proceso, muchas de la tecnología que hoy en día tenemos no sería posible, debido a que sería extremadamente lento realizar múltiples tareas a comparación a como lo hacemos actualmente. Un SO que cuente con multiprogramación puede ejecutar instrucciones sin desaprovechar tiempo, ya que mantiene al procesador trabajando en alguna instrucción todo el tiempo. Un SO multitarea puede ejecutar instrucciones de diversas tareas, pero los ordenadores trabajan tan rápido que da la apariencia de estar ejecutando todo al mismo tiempo. Gracias a esto hoy en día podemos realizar diversas tareas simultáneamente y tener programas rápidos que puedan aprovechan todo el potencial de una computadora. En este documento se estudiarán dos programas capaces de generar procesoos hijos para ejecutar diversas tareas al mismo tiempo, un Cliente capaz detectar cuando se creé un nuevo archivo en una carpeta especifica y sincronizar este archivo con un programa Servidor.

**Desarrollo**

**Sockets**

Para desarrollar este programa serán necesario utilizar Sockets, que no son más que un descriptor de fichero un tanto especial (en UNIX todo es un fichero), el cual nos permite conectarnos a ordenadores remotos o permitir que estos se conecten al nuestro a través de una red, ósea, se pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada.

Los ficheros generalmente se crean con las llamadas open() o create(), pero para declarar sockets se hace con la llamada socket().

Para montar un socket en modo de servidor (espera a que un socket cliente se conecte a el), se deben de seguir 4 pasos:

1. Crear el socket: socket();

2. Enlazar la información con el socket: bind();

3. Poner al socket en modo de escucha: listen();

4. Recibir un socket cliente: accept();

Para montar un socket en modo de cliente (se conecta a un cliente servidor), solo se necesitan 3 pasos:

1. Crear el socket: socket();

2. Asignar información del socket servidor: sockaddr\_in

3. Conectarse al servidor: connect();

Una vez conectado cliente-servidor estos pueden empezar a intercambiar información mediante las funciones read() y write() de toda la vida. Al finalizar la comunicación se pueden cerrar mediante la llamada close().

**Fork**

En UNIX tenemos una llamada al sistema capaz de crear procesos, esta llamada es fork(). Esta llamada copia Código, pila y datos del proceso padre (el proceso que ejecuto la llamada) y devuelve 0 para identificar este proceso en tiempo de ejecución.

Una de las características de esta llamada es que no copia los punteros creados por el proceso padre, en vez de ello reserva memoria para estos punteros, debido a esto no es posible comunicarse reservando un espacio en la memoria para comunicación entre ambos procesos.

**Pipes**

Los pipies son un descriptor de archivo especiales por los cuales se pueden comunicar varios procesos, estos se crean con la llamada pipe(), se le debe de pasar como parámetro un array de 2 enteros donde regresara los descriptores de archivos, uno para lectura y otro para escritura. Se debe de crear antes de la llamada a fork() y cuando se crea el proceso padre, estos deben de cerrar los extremos del pipe que no ocupan.

Una vez en los procesos, estos pueden escribir o leer información del extremo del pipe correcto con las llamadas read() y write().

**Inotify**

UNIX nos proporciona una alternativa que nos permite monitorizar los cambios que se realizan en un directorio, la cual llamaremos inotify. inotify\_init() retorna un descriptor de archivo que nos permite añadir un watch descriptor que nos notificara mediante la llamada read() cuando se produzca un cambio especificado anteriormente en el directorio configurado.

Para usar un Inotify se requieren cuatro pasos:

1. Crear el descriptor de archivo: inotify\_init();

2. Añadir los eventos que capturaremos: inotify\_add\_watch();

3. Esperar a que se produzca el cambio especificado en el directorio: read();

4. Obtener la información del cambio realizado: inotify\_event\*

Con todas estas herramientas se puede entender el diagrama de actividades que se muestra en la **Figura 1.0** en el **apendice A**, el cual muestra el flujo de los puntos importantes del programa.

**Explicación del Programa Cliente**

Todo comienza creando un Inotify el cual nos alertara sobre los directorios creados en la carpeta especificada (la cual se pasa como parámetro al programa en su quinto argumento argv[4]), cuando este detecte un cambio creara un proceso hijo al cual le delegara la tarea de crear un socket cliente que se conectara al socket servidor (la direccion ip y el puerto se le pasan como argumento al programa mediante argv[2] y argv[3] respectivamente). Una vez conectados el cliente leera el archivo creado y lo enviara al servidor, cuando termine cierra la conexión con el socket y sale del proceso hijo. Esta operación la realizan todos los hijos creados por inotify, mientras el proceso padre solo se encarga de detectar los cambios en la carpeta.

**Explicación del Programa Servidor**

Este se encargará de crear un socket servidor y recibir las peticiones de los sockets clientes. Cada que se conecte un socket servidor este creara un hijo, el cual se encarga de atender la peticiónes de los clientes y crearan las carpetas (si no existen) y el archivo, cuando finaliza cierr el socket y sale del proceso hijo; todo esto mientras que el padre cierra el socket con el cliente y espera su siguiente llamada.

**Observaciones**

Cuando el servidor cliente inicia, se le pasa como parametro la ruta de la carpeta que monitoreara. Cuando se ejecuta el programa este hace una busqueda recursiva de subdirectorios y a estos se les añade a la lista de carpetas a ser monitoreadas; tambien cuando detecta que un archivo se ha creado detecta si es directorio o archivo, en caso de ser directorio se le añade a la lista de carpetas a monitorear.

El servidor es capaz de detectar si es una carpeta o un archivo con solo saber la ruta del directorio, gracias a esto puede tratar a un archivo y un directorio de manera distinta, las carpetas se tratan con mkdir() y los directorios con fopen(). Esto nos brinda la posibilidad de crear un archivo en una carpeta que no exista en el directorio cliente y se pasara tanto la carpeta como el archivo creado dentro.

**Procesos**

Cliente -Caso 1: Cuando se ejecuta solo tiene 1 procesos corriendo: uno para inotify.

Cliente -Caso 2: Cuando detecta que un nuevo archivo este crea un proceso hijo por cada archivo creado, suponiendo que solo se añade un archivo este crea solo un proceso mas, en caso de que sean n archivos crea n procesos mas.

Servidor -Caso 1: Cuando se ejecuta solo tiene 1 proceso corriendo: uno para el socket servidor.

Servidor -Caso 2: Cuando se el servidor recibe un cliente y sincroniza un archivo, este crea 1 proceso extra para atender a este cliente, si recive n clientes se crean n procesos.

Suponiendo que se envien 5 archivos se tendrian 1 + 5 = 6 procesos de parte del cliente y 1 + 5 = 6 procesos del servidor teniendo un total de 6 + 6 = 12 procesos de los cuales 10 son procesos hijos creados por los 2 procesos principales del los programas, los procesos hijos eventualmente moriran y al final solo quedaran 2 procesos.

**Uso de Programa cliente: ./clientSocket [IP] [PUERTO] [RUTA CARPETA]**

**Uso de Programa servidor: ./serverSocket [IP] [PUERTO] [BACKLOG] [RUTA CARPETA]**

**CONCLUSIONES**

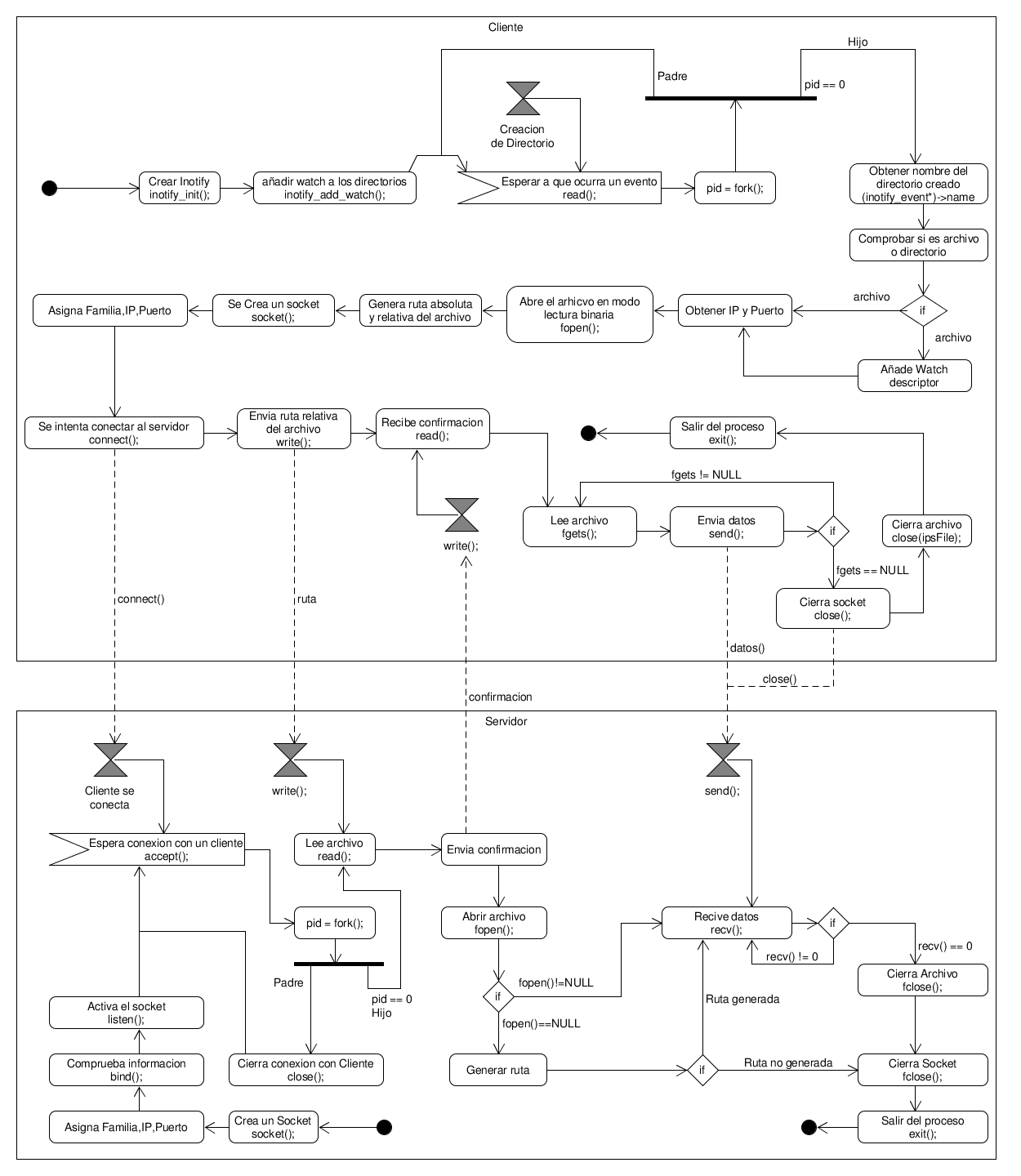
Este programa se puede mejorar de muchas maneras, una de ellas es añadir una bitacora de archivo para poder hacer un mejor administracion de los archivos y poder darles a los clientes una manera rapida y sencilla de saber si estan actualizados o no.

En la bitacora se guardarian el tiempo en el que llego cada archivo asi que se podrian identificar los casos donde 2 archios se llaman igual pero llegaron en tiempo distinto, y con esto poder actuar al respecto.

Otra mejora seria poder añadir un mejor protocolo de comunicación entre cliente y servidor para darle mas funciones, ya que ahora mismo solo es un reslpadador de archivos en la nube. Por ejemplo si el cliente le mandara el mensaje sincronizar al servidor, este le regrese los archivos que aun no tiene el cliente, o si mandara el archivo estatus muestre la informacion de los archivos del servidor.

**APENDICE**

**A. Diagrama del cliente y servidor**



**Figura 1.0** Diagrama de actividad de los programas de sincronización de archivos remotos.

Este diagrama se encuentra adjunto con el documento.

también puede verlo (o descargarlo) junto con el Código del programa el siguiente enlace: <https://github.com/VekzLight/Universidad/tree/main/SistemasOperativos>

**B. clienteSocket.c**

//Simbolos Estandar y salir

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

//Errores y Strings

#include <stdio.h>

#include <string.h>

#include <cerrno>

//Sockets

#include <netinet/in.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netdb.h>

//Monitorizacion

#include <sys/inotify.h>

#include <dirent.h>

#define BUFF\_SIZE 1024

void enviarArchivo(int fdConnection, FILE \*fdFile, char \*name);

void addWatchToTree(int fdDirectory, char \*\_route, int wdCreate);

void buscarArchivo(char \*routeRoot, char \*name, char \*\_route);

void clientSocket(char \*\_ip, int \_port, char \*\_routeRoot, char \*name, int fdDirectory, int wdCreate);

void directoryMonitor(char \*\_ip, int \_port, char \_route[]);

// Proceso principal

int main(int argc, char \*\* argv) {

directoryMonitor(argv[1], atoi(argv[2]), argv[3]);

exit(0);

}

// Inotify

void directoryMonitor(char \*\_ip, int \_port, char \_route[]){

int fdDirectory, wdCreate; // File Descriptor de la carpeta a monitorizar y watch desciptor del evento

char buffInotify[BUFF\_SIZE]; // Buffer de la informacion de InotifyEvent

ssize\_t lengthRd; // Longitud del evento leido

pid\_t pid;

// Crea una instancia de Inotify

fdDirectory = inotify\_init();

if (fdDirectory == -1) {

fprintf(stderr, "[INotify-error]: Creacion de Inotify Fallida. %d: %s \n", errno, strerror(errno));

exit(-1);

}

printf("[INotify]: Se ha Creado Inotify\n");

addWatchToTree(fdDirectory, \_route, wdCreate);

// Ciclo que monitorea constante mente el directorio

while(1){

// Espera a que ocurra un evento y guarda la informacion y su longitud

lengthRd = read(fdDirectory, buffInotify, BUFF\_SIZE);

if(lengthRd <= 0){

fprintf(stderr, "[INotify-error]: La lectura de inotify es <= 0.\n");

exit(-1);

}

printf("[INotify]: Se ha detectado un cambio.\n");

pid = fork();

if(pid == 0){

//close(fdDirectory);

printf("[INotify]: El archivo creado es: %s \n", ((struct inotify\_event\*)buffInotify)->name );

clientSocket(\_ip, \_port, \_route, ((struct inotify\_event\*)buffInotify)->name, fdDirectory, wdCreate);

printf("Saliendo del proceso.\n");

exit(0);

}

}

close(fdDirectory);

}

// Socket cliente (Transmision de archivos)

void clientSocket(char \*\_ip, int \_port, char \*\_routeRoot, char \*name, int fdDirectory, int wdCreate){

int fdSocket; // File Descriptor del socket

struct sockaddr\_in serverAddr; // Direccion del Socket del Servidor

char buffRx[BUFF\_SIZE]; // Buffer de Entrada

// Creacion del Socket

if ((fdSocket = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP)) == -1){

printf("[CLIENTE]: Fallo la Creacion del Cliente.\n");

exit(-1);

}

printf("[CLIENTE]: El Socket se Creo Satisfactoriamente.\n");

// Limpia el socket

memset(&serverAddr, 0, sizeof(serverAddr));

// Asigna la Familia, IPv4, Puerto

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = inet\_addr( \_ip );

serverAddr.sin\_port = htons( \_port );

// Obtiene la ruta absoluta del archivo creado

char \_absoluteRoute[BUFF\_SIZE] = {0};

buscarArchivo(\_routeRoot, name, \_absoluteRoute);

char \_routeTemp1[BUFF\_SIZE];

char \_absoluteTemp1[BUFF\_SIZE];

char \_relativeRoute[BUFF\_SIZE] = {0};

char delim1[2] = "/";

char \*tokenOld;

char \*tokenNew;

strcpy(\_routeTemp1, \_routeRoot);

strcpy(\_absoluteTemp1, \_absoluteRoute);

tokenNew = strtok(\_routeTemp1, delim1);

while(tokenNew != NULL){

tokenOld = tokenNew;

tokenNew = strtok(NULL, delim1);

}

tokenNew = strtok(\_absoluteTemp1, delim1);

bool flag = false;

while(tokenNew != NULL){

if((strcmp(tokenNew, tokenOld)==0)){

flag = true;

} else if(flag){

strcat(\_relativeRoute, "/");

strcat(\_relativeRoute, tokenNew);

}

tokenNew = strtok(NULL, delim1);

}

printf("[CLIENTE]: Ruta absoluta del archivo: %s\n", \_absoluteRoute);

printf("[CLIENTE]: Ruta relativa del archivo: %s\n", \_relativeRoute);

// Abre el archivo creado en modo lectura binaria

FILE \*fileOut = fopen(\_absoluteRoute,"rb");

if(fileOut == NULL){

printf("[CLIENTE-error]: Archivo no encontrado\n");

exit(-1);

}

printf("[CLIENTE]: Archivo creado abierto exitosamente. \n");

// Intenta abrir el archivo en forma de directorio

// para saber si es directorio o archivo

// En caso de que sea directorio le añade un watch descriptor

DIR \*isDir = opendir(\_absoluteRoute);

if (!(isDir = opendir(\_absoluteRoute))){

printf("[CLIENTE]: Es un archivo\n");

} else {

addWatchToTree(fdDirectory, \_absoluteRoute, wdCreate);

strcat(\_relativeRoute, "/");

printf("[CLIENTE]: Es directorio\n");

}

closedir(isDir);

// Intenta conectarse al socket del servidor

if (connect(fdSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) != 0){

fprintf(stderr, "[CLIENTE-error]: No Pudo Conectar el Socket. %d: %s \n", errno, strerror( errno ));

exit(-1);

}

printf("[CLIENTE]: Conexion establecida con %s.\n", (char \*)inet\_ntoa(serverAddr.sin\_addr));

enviarArchivo(fdSocket, fileOut, \_relativeRoute);

printf("[CLIENTE]: Cerrando conexion.\n");

close(fdSocket);

printf("[CLIENTE]: Cerrando Archivo.\n");

fclose (fileOut);

}

// Busca el archivo creado en los subdirectorios y regresa la ruta abtoluta de este

void buscarArchivo(char \*routeRoot, char \*name, char \*\_route){

DIR \*dir;

struct dirent \*entry;

if (!(dir = opendir(routeRoot))){

fprintf(stderr, "[INotify-error]: No se pudo Abrir el directorio. %d: %s \n", errno, strerror(errno));

} else {

while ((entry = readdir(dir)) != NULL) {

if(entry->d\_type == DT\_DIR){

char path[BUFF\_SIZE];

if (strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0)

continue;

snprintf(path, sizeof(path), "%s/%s", routeRoot, entry->d\_name);

//printf("\nDirectorio: %s:%s", entry->d\_name, name);

if(strcmp(entry->d\_name, name) == 0){

strcpy(\_route, path);

break;

} else {

buscarArchivo(path, name, \_route);

if(\_route[0] != 0) break;

}

} else {

char path[BUFF\_SIZE];

snprintf(path, sizeof(path), "%s/%s", routeRoot, entry->d\_name);

//printf("\nArchivo: %s:-%s", path, entry->d\_name);

if(strcmp(entry->d\_name, name) == 0){

strcpy(\_route, path);

break;

}

}

}

closedir(dir);

}

}

// Añade una Directorio y sus subdirectorios un watch descriptor

void addWatchToTree(int fdDirectory, char \*\_route, int wdCreate){

DIR \*dir;

struct dirent \*entry;

if (!(dir = opendir(\_route))){

fprintf(stderr, "[INotify-error]: No se pudo Abrir el directorio. %d: %s \n", errno, strerror(errno));

} else if(\_route != ""){

// Crea un Watch Descriptor para la ruta dada

wdCreate = inotify\_add\_watch(fdDirectory, \_route, IN\_CREATE);

if (wdCreate == -1) {

fprintf(stderr, "[INotify-error]: No se pudo añadir el Watch. %d: %s \n", errno, strerror(errno));

exit(-1);

}

printf("[INotify]: Se ha añadodo watch a %s \n", \_route );

while ((entry = readdir(dir)) != NULL) {

char \_routeTemp[BUFF\_SIZE] = {0};

if (entry->d\_type == DT\_DIR) {

char path[BUFF\_SIZE];

if (strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0)

continue;

snprintf(path, sizeof(path), "%s/%s", \_route, entry->d\_name);

addWatchToTree(fdDirectory, path, wdCreate);

}

}

closedir(dir);

}

}

// Se conecta con el servidor y envia el archivo

void enviarArchivo(int fdConnection, FILE \*fdFile, char \*name){

char data[BUFF\_SIZE] = {0};

char msj[BUFF\_SIZE] = {0};

// Le envia la ruta relativa del archivo y recibe confirmacion

printf("[CLIENTE]: Mandando ruta de archivo.\n");

write(fdConnection, name, BUFF\_SIZE);

read(fdConnection, msj, BUFF\_SIZE);

printf("[SERVIDOR]: %s.\n");

// Inicia la tranferencia del archivo

printf("[CLIENTE]: Iniciando Transferencia\n");

while(fgets(data, BUFF\_SIZE, fdFile) != NULL) {

if (send(fdConnection, data, sizeof(data), 0) == -1) {

perror("[CLIENTE]: Error al mandar el archivo.");

exit(-1);

}

bzero(data, BUFF\_SIZE);

}

printf("[CLIENTE]: Transferencia Completada\n");

close(fdConnection);

}

**C. serverSocket.c**

//Estandar

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

//Errores y Strings

#include <errno.h>

#include <string.h>

//Sockets

#include <netinet/in.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <netdb.h>

//Monitorizacion

#include <sys/inotify.h>

//Directorios

#include <sys/stat.h>

#include <dirent.h>

#define BUFF\_SIZE 1024

void crearDirectorio(char \*routeRoot, char \*route);

void crearArchivo(int fdConnection, char\* routeRoot);

void socketServer(char \_ip[], int \_port, int \_backlog, char \_route[]);

/\*\*

\* argv[1] = IP

\* argv[2] = PUERTO

\* argv[3] = BACKLOG

\* argv[4] = RUTA

\*/

int main(int argc, char \*\* argv){

socketServer(argv[1], atoi(argv[2]), atoi(argv[3]), argv[4]);

exit(0);

}

// Crea e inicia el socket servidor

void socketServer(char \_ip[], int \_port, int \_backlog, char \_route[]){

int fdSocket, fdConnection; // File Descriptors

struct sockaddr\_in serverAddr, clientAddr; // Formato de direccion del servidor y cliente

socklen\_t clientAddrLen; // Tamaño de la direccion del cliente

pid\_t pIdSocket; // Id para control del proceso padre e hijo

// Crea y Comprueba si el socket se ha creado exitosamente

if ( (fdSocket = socket(AF\_INET, SOCK\_STREAM, 0)) == -1 ) {

fprintf(stderr, "[SERVIDOR-error]: Creacion de Socket Fallida. %d: %s \n", errno, strerror(errno));

exit(EXIT\_FAILURE);

}

printf("[SERVIDOR]: Socket Creado Satisfactoriamente.\n");

// Inilializa con ceros la estructura

memset(&serverAddr, 0, sizeof(serverAddr));

// Asigna la Familia, IPv4, Puerto

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = inet\_addr(\_ip);

serverAddr.sin\_port = htons(\_port);

// Asigna y Comprueba si se asigno correctamente la IP y el Puerto al Socket

if ( (bind(fdSocket, (struct sockaddr \*)&serverAddr, sizeof(serverAddr))) != 0 ){

fprintf(stderr, "[SERVIDOR-error]: Enlazamiento de Socket Fallida. %d: %s \n", errno, strerror( errno ));

exit(EXIT\_FAILURE);

}

printf("[SERVIDOR]: Socket Enlazado Satiscatoriamente.\n");

// Pone el Socket en modo escuchar (Listen)

if ((listen(fdSocket, \_backlog)) != 0){

fprintf(stderr, "[SERVIDOR-error]: No Pudo Activar el Socket. %d: %s \n", errno, strerror( errno ));

exit(EXIT\_FAILURE);

}

printf("[SERVIDOR]: Escuchando en el Puerto %d \n", ntohs(serverAddr.sin\_port) );

// Ciclo que espera a que se conecte un cliente y crea un hijo para que lo atienda

while(1){

// Espera la coneccion de un cliente

clientAddrLen = sizeof(struct sockaddr\_in);

fdConnection = accept(fdSocket, (struct sockaddr \*)&clientAddr, &clientAddrLen);

printf("[SERVIDOR]: Conexion Detectada %s.\n", (char \*)inet\_ntoa(clientAddr.sin\_addr));

if (fdConnection < 0){

// Error en la conexion

fprintf(stderr, "[SERVIDOR-error]: Conexion no aceptada. %d: %s \n", errno, strerror( errno ));

} else {

pIdSocket = fork();

if(pIdSocket == 0){ // Proceso hijo que servira al cliente

crearArchivo(fdConnection, \_route);

printf("[SERVIDOR-hijo]: Cerrando conexion\n");

close(fdConnection);

printf("[SERVIDOR-hijo]: Saliendo de proceso hijo.\n");

exit(0); // Sale del proceso hijo

} else { close(fdConnection); } // Cierra la coneccion para el padre

}

memset(&fdConnection, 0, sizeof(fdConnection));

}

}

// Crea el archivo en la carpeta del servidor

void crearArchivo(int fdConnection, char\* routeRoot){

int \_len;

FILE \*fdFile;

char \_buffer[BUFF\_SIZE];

char route[BUFF\_SIZE];

printf("[SERVIDOR-hijo]: Obteniendo ruta de archivo.\n");

read(fdConnection, route, BUFF\_SIZE);

printf("[SERVIDOR-hijo]: Ruta obtenida. %s\n", route);

write(fdConnection, "Recibido", BUFF\_SIZE);

// Rutas temporales

char \_tempRouteDir[BUFF\_SIZE];

strcpy(\_tempRouteDir,routeRoot);

strcat(\_tempRouteDir,route);

// Distingue en tre archivo y directorio

int contador = 0;

while( route[contador] != 0 ) contador++;

contador--;

if(route[contador] == '/'){

printf("[SERVIDOR-hijo]: Es un directorio.\n");

DIR \*dirFD = opendir(\_tempRouteDir);

if(!(dirFD = opendir(\_tempRouteDir))){

fprintf(stderr, "[SERVIDOR-error]: Iniciando creacion de directorio. %d: %s \n", errno, strerror(errno));

crearDirectorio(routeRoot, route);

} else { printf("El directorio ya existe\n"); }

closedir(dirFD);

} else {

printf("[SERVIDOR-hijo]: Es un archivo.\n");

fdFile = fopen(\_tempRouteDir, "wb");

if(fdFile == NULL){

// No existe el directorio donde se quiere crear el archivo

fprintf(stderr, "[SERVIDOR-error]: No existe el directorio. %d: %s \n", errno, strerror(errno));

// Intenta crear el directorio faltante

int index = 0;

int indexcut = 0;

char routeCut[BUFF\_SIZE] = {0};

char routeRelative[BUFF\_SIZE] = {0};

while(route[index] != 0){

routeCut[indexcut] = route[index];

indexcut++;

index++;

if(route[index] == '/'){

strcat(routeRelative, routeCut);

strcat(routeRelative, "/");

bzero(routeCut, BUFF\_SIZE);

indexcut = 0;

}

}

crearDirectorio(routeRoot, routeRelative);

// Intenta crear nuevamente el archivo

fdFile = fopen(\_tempRouteDir, "wb");

if(fdFile == NULL){

fprintf(stderr, "[SERVIDOR-error]: No se pudo crear el directorio. %d: %s \n", errno, strerror(errno));

fclose(fdFile);

exit(-1);

}

// Inicia la tranferencia del archivo

while((\_len=recv(fdConnection, \_buffer, BUFF\_SIZE,0))!=0) {

fwrite(\_buffer, sizeof(\_buffer), 1, fdFile);

printf("[RECIBIDO]: %s\n", \_buffer);

bzero(\_buffer, BUFF\_SIZE);

}

printf("[SERVIDOR]: Tranferencia finalizada\n");

fclose(fdFile);

exit(-1);

} else {

// Inicia la transferencia del archivo

while((\_len=recv(fdConnection, \_buffer, BUFF\_SIZE,0) != 0)) {

fwrite(\_buffer, sizeof(\_buffer), 1, fdFile);

printf("[RECIBIDO]: %s\n", \_buffer);

bzero(\_buffer, BUFF\_SIZE);

}

printf("[SERVIDOR]: Transferencia finalizada\n");

fclose(fdFile);

exit(-1);

}

}

}

// Crea la ruta especificada

void crearDirectorio(char \*routeRoot, char \*route){

char \_routeTemp[BUFF\_SIZE];

char delim[2] = "/";

char \*token1;

char \*token2;

char \_routeFoward[BUFF\_SIZE] = {0};

strcpy(\_routeFoward, routeRoot);

strcat(\_routeFoward, "/");

strcpy(\_routeTemp, route);

token1 = strtok(\_routeTemp, delim);

while(token1 != NULL){

token2 = token1;

token1 = strtok(NULL, delim);

strcat(\_routeFoward, token2);

printf("[SERVIDOR]: Creando directorio: %s\n", \_routeFoward);

if(mkdir(\_routeFoward, S\_IRWXU) == -1)

fprintf(stderr, "[SERVIDOR-error]: Este directorio ya existe. %d: %s \n", errno, strerror(errno));

else printf("[SERVIDOR]: Directorio creado.");

if(token1 != NULL) strcat(\_routeFoward, "/");

}

}