Sistema de base de datos distribuida con redundancia: “FARO”

Integrantes:

* Ruben Cuba Lajo
* Alonso Valdivia Quispe
* Fabrizio Flores Pari
* Oscar Ramos Ramirez

1. Problema:

Queremos almacenar datos con atributos y almacenar enlaces entre los datos, pero no queremos que todos los datos estén en un mismo computador (datos distribuidos).

Además de que si una de las computadoras que almacenan los datos se desconectara por cualquier motivo, no se pierdan ni datos ni conexiones (redundancia).

1. Propuesta:

Una base de datos distribuida con redundancia con el modelo Master-Slaves basada en grafos.

Este sistema posee las siguientes funcionalidades:

* Crear nodo.
* Linkear nodos.
* Borrar nodos.
* Deslinkear nodos.
* Actualizar el identificador del nodo.
* Explorar nodos a partir de un nodo inicial.
* Explorar nodos a partir de un nodo inicial mostrando sus atributos.

1. Diseño protocolo:

Primero se muestra una explicación de la notación que se usa para mostrar la estructura del protocolo:

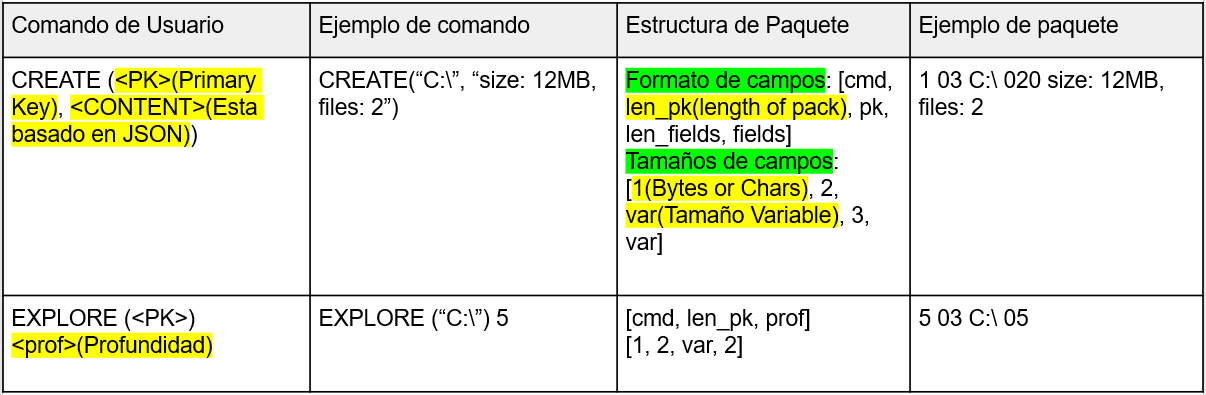
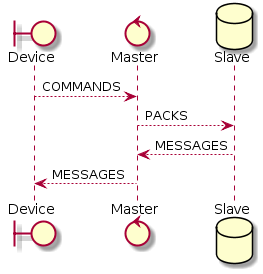
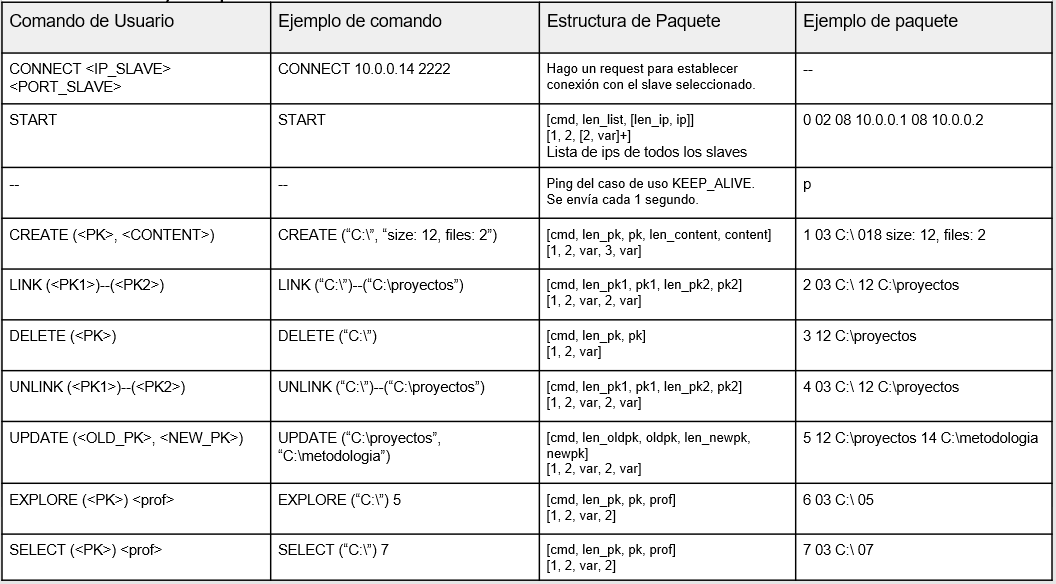


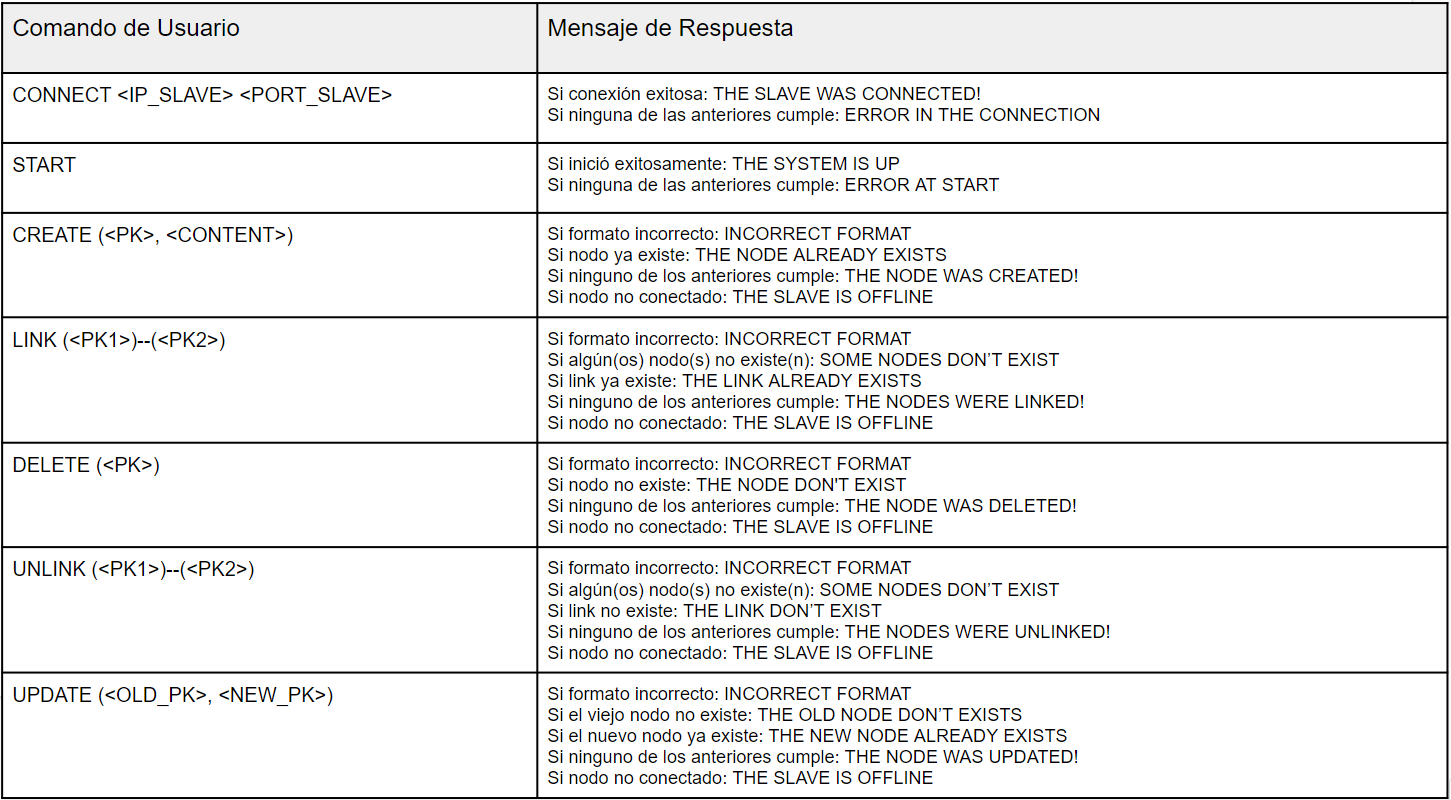
Gráfico de cómo se realizan las comunicaciones en el sistema:



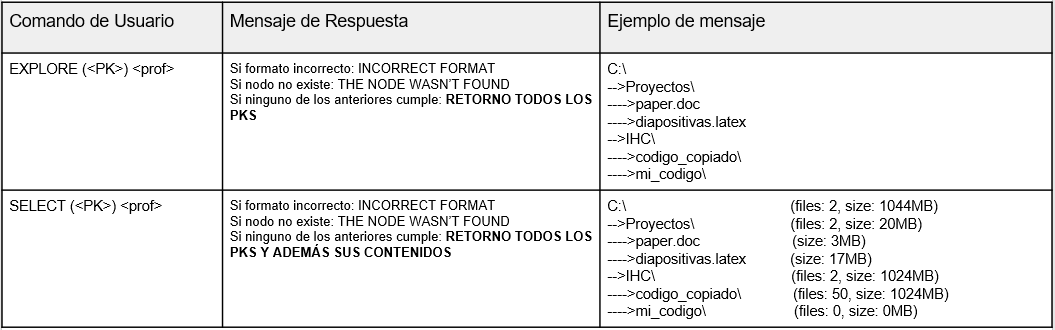
* Comunicaciones de Entrada:
  + Comandos y Paquetes:



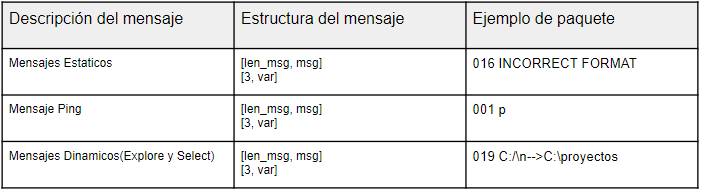
* Comunicaciones de salida:
  + Mensajes Estáticos:

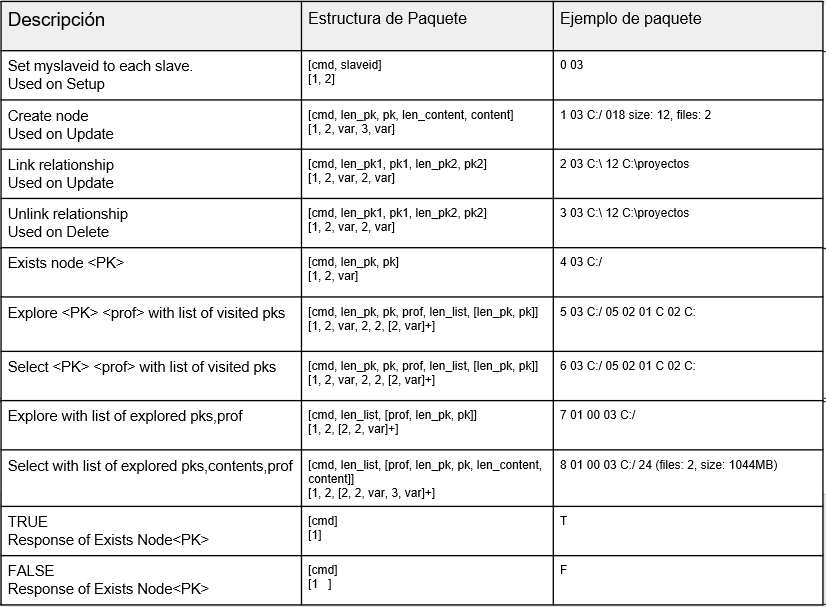


* + Mensajes Dinámicos:



* + Estructura de Mensajes:



* + Protocolo Interno de Slaves:

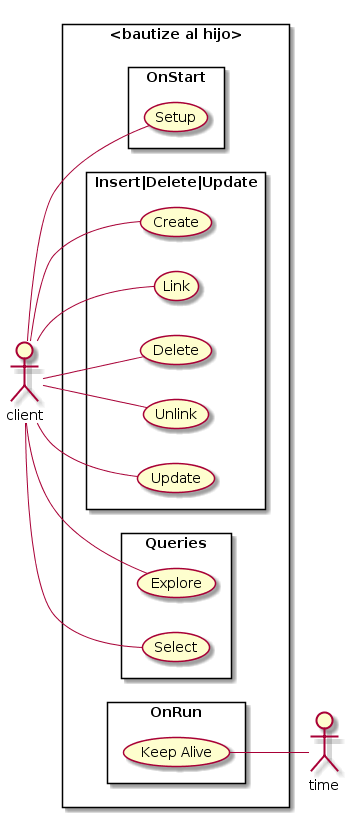
Response

Reques

Actions

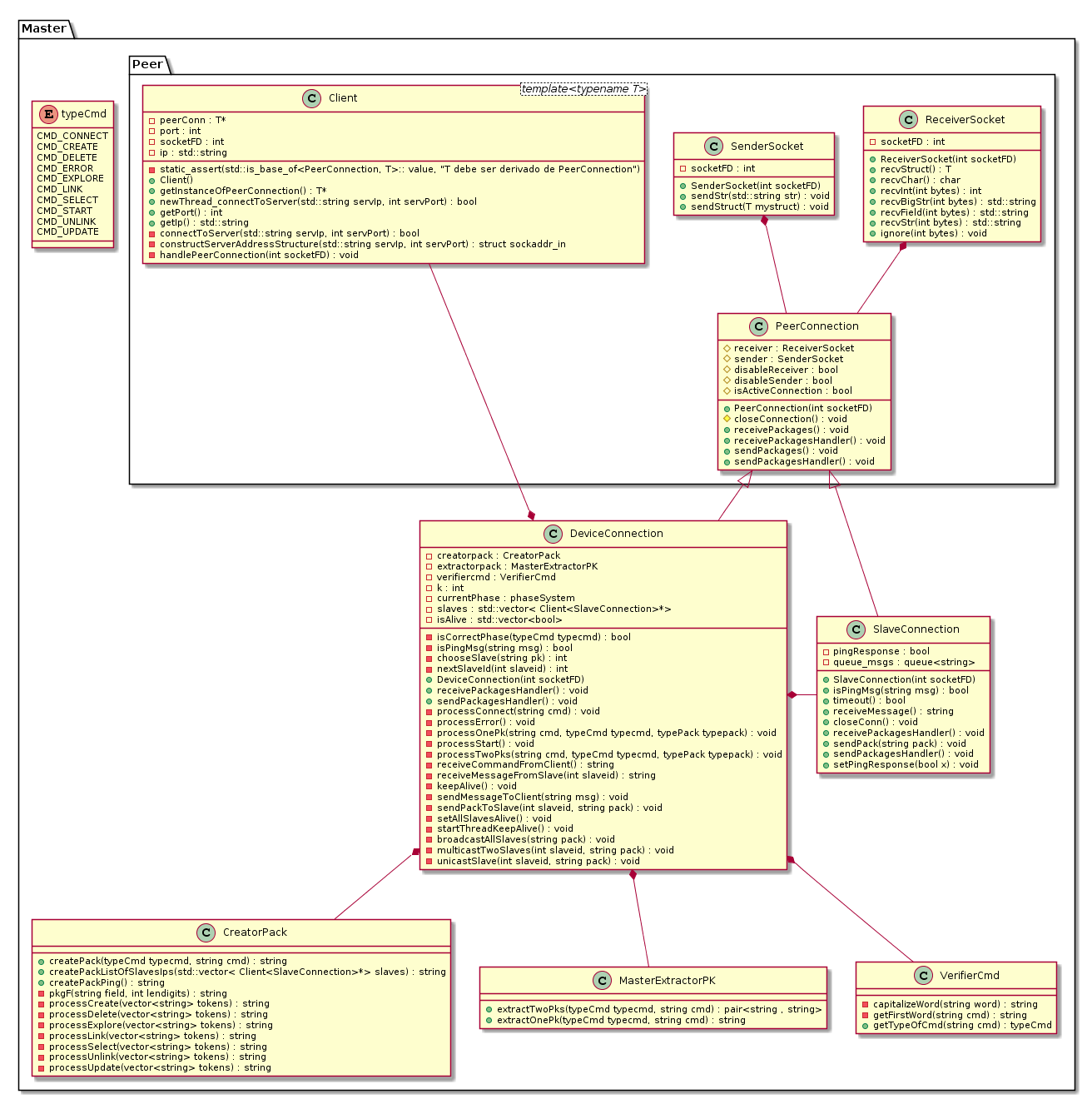
1. Diagramas de clases de uso, secuencia y clases:

* Diagrama de casos de uso:



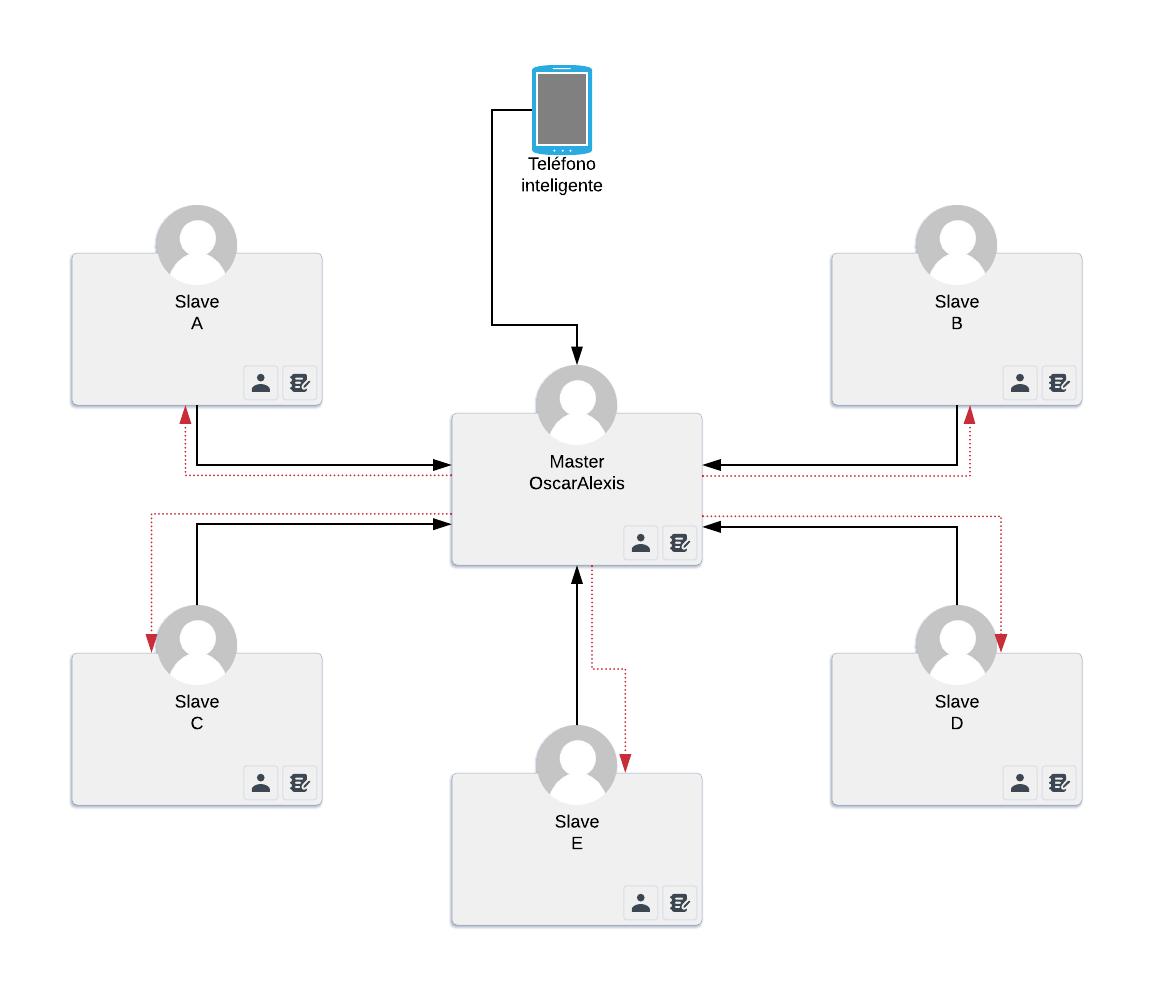
**FARO**

* Diagrama de Clases:



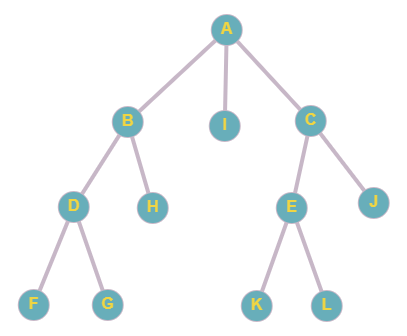
1. Evaluaciones y Resultados:

Se probo el sistema con 1 cliente, 1 master, 5 slaves:

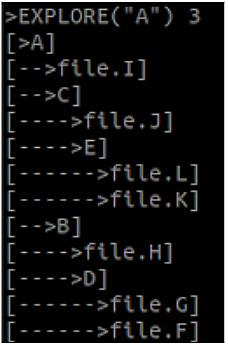


Con los siguientes casos de prueba:

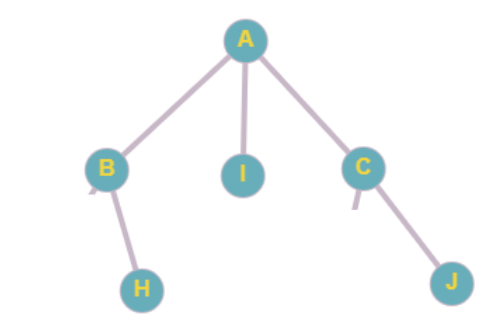
1. Pseudo-Tree:



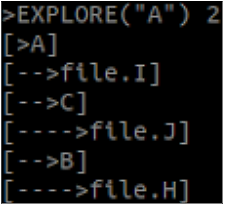
* Comandos que se utilizaron en la prueba:
  + CREATE("A", "size: 42kb, files : 3")
  + CREATE("B", "size: 35kb, files : 2")
  + CREATE("C", "size: 70kb, files : 2")
  + CREATE("D", "size: 79kb, files : 2")
  + CREATE("E", "size: 63kb, files : 2")
  + CREATE("file.F", "size: 42kb")
  + CREATE("file.G", "size: 35kb")
  + CREATE("file.H", "size: 70kb")
  + CREATE("file.I", "size: 79kb")
  + CREATE("file.J", "size: 63kb")
  + CREATE("file.K", "size: 63kb")
  + CREATE("file.L", "size: 63kb")
  + LINK ("A")--("B")
  + LINK ("A")--("C")
  + LINK ("B")--("D")
  + LINK ("C")--("E")
  + LINK ("D")--("file.F")
  + LINK ("D")--("file.G")
  + LINK ("B")--("file.H")
  + LINK ("A")--("file.I")
  + LINK ("E")--("file.K")
  + LINK ("E")--("file.L")
  + LINK ("C")--("file.J")
  + EXPLORE ("A") 3
  + DELETE ("D")
  + UNLINK ("C")--("E")
  + EXPLORE ("A") 2
  + UPDATE ("A", "Z")
  + SELECT ("Z") 2
  1. Resultados de caso de prueba A:
     1. Resultado del comando “ EXPLORE ("A") 3 ”:



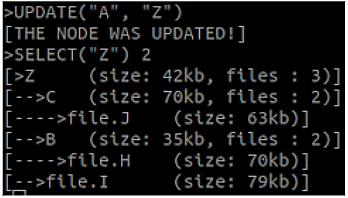
* + 1. Estado del grafo después de los comandos:
       1. DELETE ("D")
       2. UNLINK ("C")--("E")



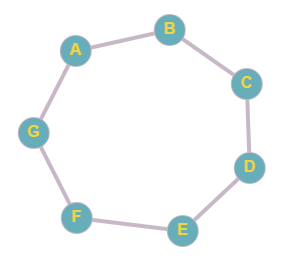
* + 1. Resultado del comando EXPLORE ("A") 2 :



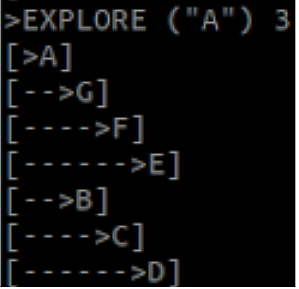
* + 1. Resultado de los comandos:
       1. UPDATE ("A", "Z")
       2. SELECT ("Z") 2



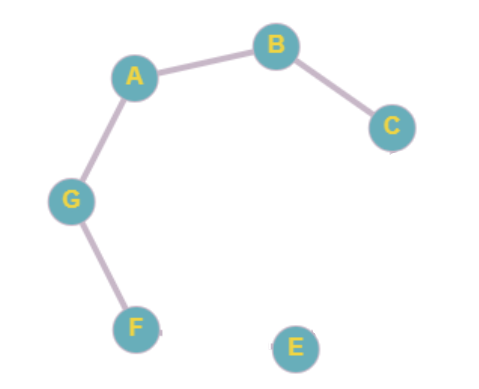
1. Grafo Circular:



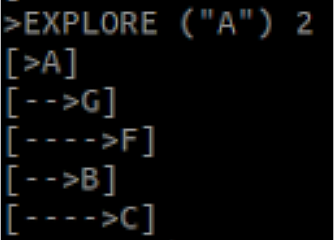
* Comandos que se utilizaron en la prueba:
  + CREATE("A", "size: 42, files : 68")
  + CREATE("B", "size: 35, files : 1")
  + CREATE("C", "size: 70, files : 25")
  + CREATE("D", "size: 79, files : 59")
  + CREATE("E", "size: 63, files : 65")
  + CREATE("F", "size: 6, files : 46")
  + CREATE("G", "size: 82, files : 28")
  + LINK ("A")--("B")
  + LINK ("B")--("C")
  + LINK ("C")--("D")
  + LINK ("D")--("E")
  + LINK ("E")--("F")
  + LINK ("F")--("G")
  + LINK ("G")--("A")
  + EXPLORE ("A") 3
  + DELETE ("D")
  + UNLINK ("E")--("F")
  + EXPLORE ("A") 2
  + UPDATE ("A", "Z")
  + SELECT ("Z") 2
  1. Resultados de caso de prueba B:
     1. Resultado del comando “ EXPLORE ("A") 3 “:



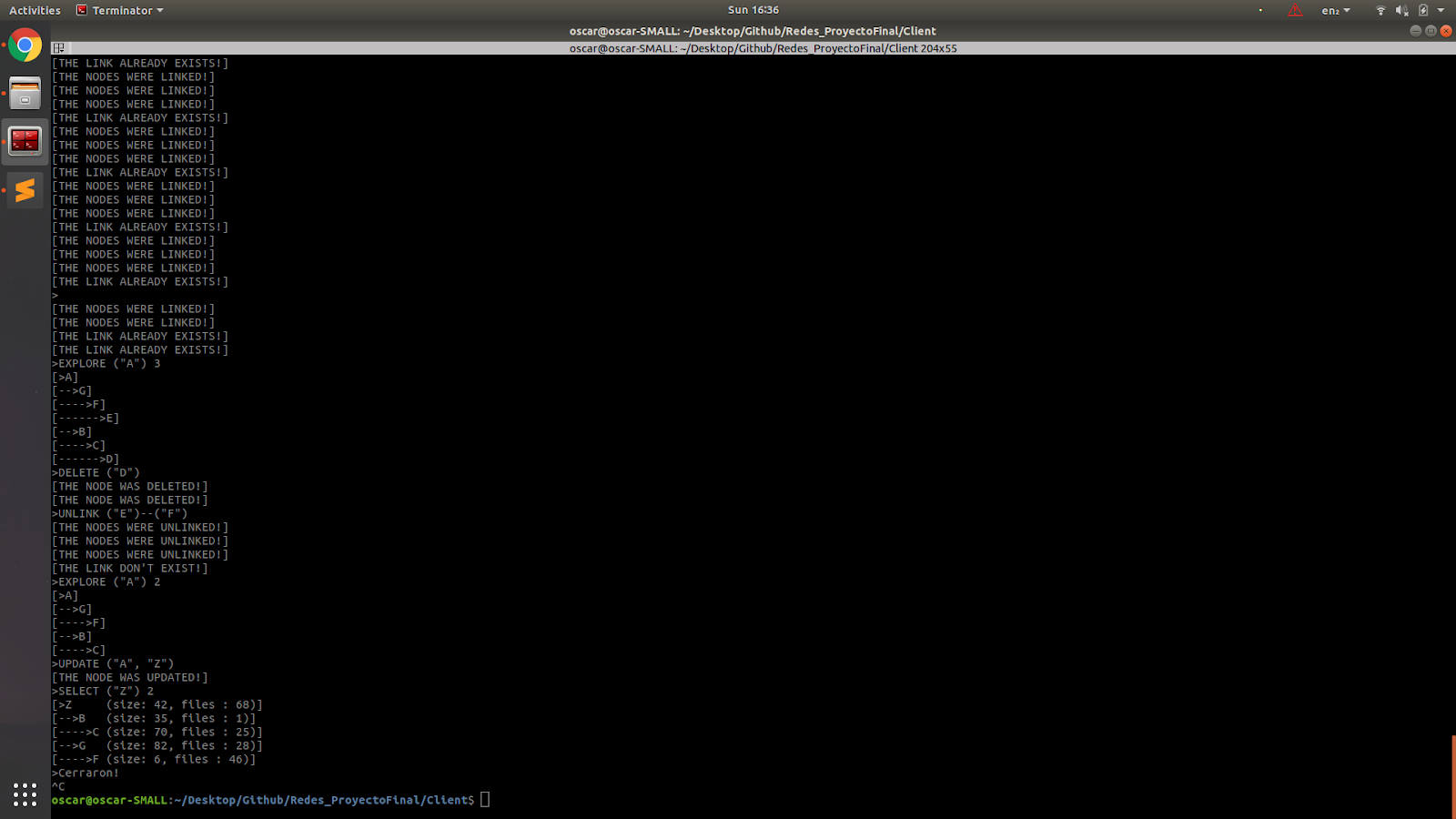
* + 1. Estado del grafo después de los comandos:
       1. DELETE ("D")
       2. UNLINK ("E")--("F")



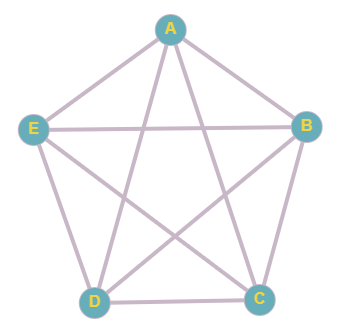
* + 1. Resultado del comando EXPLORE (“A”) 2:



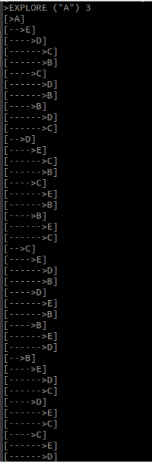
* + 1. Resultado de los comandos:
       1. UPDATE ("A", "Z")
       2. SELECT ("Z") 2



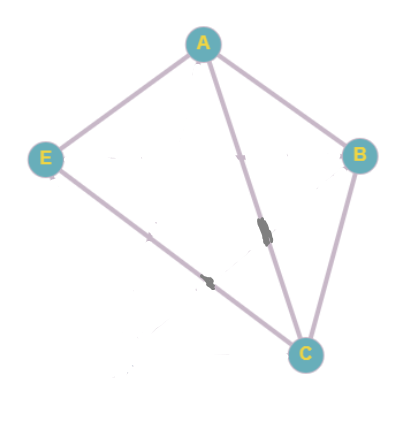
1. Grafo Completo (K5):



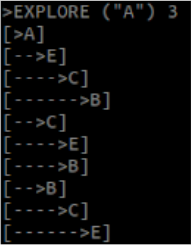
* Comandos que se utilizaron en la prueba:
  + CREATE("A", "size: 42, files : 68")
  + CREATE("B", "size: 35, files : 1")
  + CREATE("C", "size: 70, files : 25")
  + CREATE("D", "size: 79, files : 59")
  + CREATE("E", "size: 63, files : 65")
  + LINK ("A")--("B")
  + LINK ("A")--("C")
  + LINK ("A")--("D")
  + LINK ("A")--("E")
  + LINK ("B")--("C")
  + LINK ("B")--("D")
  + LINK ("B")--("E")
  + LINK ("C")--("D")
  + LINK ("C")--("E")
  + LINK ("D")--("E")
  + EXPLORE ("A") 3
  + DELETE ("D")
  + UNLINK ("E")--("B")
  + EXPLORE ("A") 3
  + UPDATE ("A", "Z")
  + SELECT ("Z") 3
  1. Resultados de caso de prueba C:
     1. Resultado del comando “ EXPLORE ("A") 3 “:



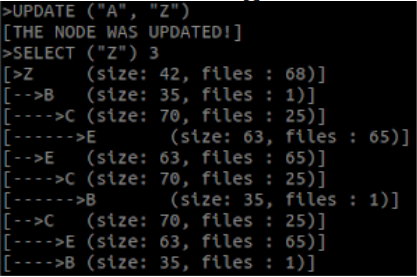
* + 1. Estado del grafo después de los comandos:
       1. DELETE ("D")
       2. UNLINK ("E")--("B")



* + 1. Resultado del comando EXPLORE ("A") 3:



* + 1. Resultado de los comandos:
       1. UPDATE ("A", "Z")
       2. SELECT ("Z") 3



1. Código:

* Se empezara por mostrar el contenido de dos carpetas,aux/ y peer/ que se replican en las carpetas de Cliente/, Master/ y Slave/:
  + Aux/
    - Safe.h

//////////////////////////////AUX/////////////////////////////

// Agrega una capa de seguridad a cada funcion

// Separa el flujo principal del algoritmo, del flujo de excepciones

// Es "Safe" solo porque puede lanzar excepciones, no mas ni menos

// Lo unico que hace es que, si hay error, lanzar excepcion.

// No me pregunten mas de que sirve!!

#ifndef SAFE\_H

#define SAFE\_H

#include <sys/types.h>

#include <sys/socket.h>

namespace Safe

{

int socket(int domain, int type, int protocol);

int inet\_pton(int af, const char\* src, void\* dst);

void connect(int sockFD, struct sockaddr\* serv\_addr, int addrlen);

void bind(int sockFD, struct sockaddr\* my\_addr, int addrlen);

void listen(int sockFD, int backlog);

int accept(int sockFD, struct sockaddr\* addr, socklen\_t\* addrlen);

}

namespace Safe

{

int socket(int domain, int type, int protocol)

{

int sockFD = ::socket(domain, type, protocol);

if (sockFD < 0)

throw std::string("cannot create socket");

return sockFD;

}

int inet\_pton(int af, const char\* src, void\* dst)

{

int res = ::inet\_pton(af, src, dst);

if (0 > res)

throw std::string("error: first parameter is not a valid address family");

else if (0 == res)

throw std::string("char std::string (second parameter does not contain valid ipaddress");

return res;

}

void connect(int sockFD, struct sockaddr\* serv\_addr, int addrlen)

{

int res = ::connect(sockFD, serv\_addr, addrlen);

if (res < 0)

throw std::string("connect failed");

}

void bind(int sockFD, struct sockaddr\* my\_addr, int addrlen)

{

int res = ::bind(sockFD, my\_addr, addrlen);

if (res < 0)

throw std::string("error bind failed");

}

void listen(int sockFD, int backlog)

{

int res = ::listen(sockFD, backlog);

if (res < 0)

throw std::string("error listen failed");

}

int accept(int sockFD, struct sockaddr\* addr, socklen\_t\* addrlen)

{

int connectFD = ::accept(sockFD, addr, addrlen);

if (connectFD < 0)

throw std::string("error accept failed");

return connectFD;

}

}

#endif

* + - SimulatorUserInput.h

#include <iostream>

#include <algorithm>

#include <vector>

#include <string>

#include "stringhelper.h"

// Modifica std::cin!!

class SimulatorUserInput

{

public:

void simulate(std::string in)

{

std::vector<std::string> lines = split(in, '\n');

reverse(lines.begin(), lines.end());

for(std::string line: lines)

simulateLine(line);

}

void simulateLine(std::string line)

{

line += '\n';

reverse(line.begin(), line.end());

for(char c: line)

std::cin.putback(c);

}

};

* + - stringhelper.h

#ifndef STRINGHELPER\_H

#define STRINGHELPER\_H

#include <string.h>

#include <unistd.h>

#include <termios.h>

#include <vector>

#include <string>

#include <initializer\_list>

// Declaraciones

std::string join(std::initializer\_list<std::string> vec, char sep);

std::vector<std::string> split(std::string str, char sep);

std::string intWithZeros(int x, int lendigits); // rellena zeros a la izquierda

std::string packageField(std::string field, int lendigits);

std::string arrcharToString(char\* arr, int sz); // Permite '\0' en str

char getch(); // Recibe entrada teclado sin "Enter"

std::vector<std::string> tokenizeCommand(std::string s); // Util en parsear comandos de usuario

// Definiciones

std::string join(std::initializer\_list<std::string> vec, char sep=' ')

{

std::string ans;

for(std::string s: vec)

ans += s + sep;

ans.pop\_back();

return ans;

}

std::vector<std::string> split(std::string str, char sep=' ')

{

std::vector<std::string> ans;

std::string buf;

for(char c: str)

if(c == sep)

{

ans.push\_back(buf);

buf.clear();

}

else

buf += c;

ans.push\_back(buf);

return ans;

}

// 011

std::string intWithZeros(int x, int lendigits) // rellena zeros a la izquierda

{

int cnt = std::to\_string(x).length();

std::string zeros(lendigits-cnt, '0');

return zeros + std::to\_string(x);

}

// 011 mensaje1234

std::string packageField(std::string field, int lendigits)

{

std::string fieldcnt = intWithZeros(field.size(), lendigits);

return fieldcnt + " " + field;

}

std::string arrcharToString(char\* arr, int sz) // Permite '\0' en str

{

std::string str(sz, '\0');

memcpy((char\*)str.c\_str(), arr, sz);

return str;

}

char getch()

{

char buf = 0;

struct termios old = {0};

if (tcgetattr(0, &old) < 0)

perror("tcsetattr()");

old.c\_lflag &= ~ICANON;

old.c\_lflag &= ~ECHO;

old.c\_cc[VMIN] = 1;

old.c\_cc[VTIME] = 0;

if (tcsetattr(0, TCSANOW, &old) < 0)

perror("tcsetattr ICANON");

if (read(0, &buf, 1) < 0)

perror ("read()");

old.c\_lflag |= ICANON;

old.c\_lflag |= ECHO;

if (tcsetattr(0, TCSADRAIN, &old) < 0)

perror ("tcsetattr ~ICANON");

return (buf);

}

// Token: cadena de texto sin espacio(como std::cin) | cadena entre comillas("")

// No soporta: "MSG "tarea: ejemplos usando comillas "" " "

// Command puede ser: "GET test.txt" | "MSG destinatario1 "Hey, como estas!!""

// Packages puede ser: "1 08 test.txt" | "MSG "

std::vector<std::string> tokenizeCommand(std::string s)

{

std::vector<std::string> tokens;

std::string tok;

bool usingCom = false; // Usa comillas

for(size\_t i=0; i<s.size(); ++i)

{

if(!usingCom) // Caso std::cin

{

if(s[i] != ' ')

{

if(s[i] == '"') usingCom = true;

else tok.push\_back(s[i]);

}

else

{

if(!tok.empty())

{

tokens.push\_back(tok);

tok.clear();

}

}

}

else // Caso ""

{

if(s[i] == '"')

{

usingCom = false;

tokens.push\_back(tok);

tok.clear();

}

else

{

tok.push\_back(s[i]);

}

}

}

tokens.push\_back(tok);

return tokens;

}

#endif

* + peer/
    - Client.h

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <string.h>

#include <string>

#include <thread>

#include "../aux/Safe.h"

#include "PeerConnection.h"

template<typename T>

class Client

{

static\_assert(std::is\_base\_of<PeerConnection, T>::value, "T debe ser derivado de PeerConnection");

private:

T\* peerConn;

public:

Client() = default;

void newThread\_connectToServer(std::string servIp, int servPort)

{

thread th(&Client::connectToServer, this, servIp, servPort);

th.detach();

}

void connectToServer(std::string servIp, int servPort)

{

try{

// Crear socket

int socketFD = Safe::socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

struct sockaddr\_in servAddr = constructServerAddressStructure(servIp, servPort);

// Establecer conexion a server

Safe::connect(socketFD, (struct sockaddr \*)&servAddr, sizeof(struct sockaddr\_in));

printf("Cliente: conectado a servidor %s | puerto %d\n", servIp.c\_str(), servPort);

// Comunicarse con server

peerConn = new T(socketFD);

std::thread t1(&T::sendPackages, peerConn);

std::thread t2(&T::receivePackages, peerConn);

t1.join();

t2.join();

// Cerrar conexion con server

shutdown(socketFD, SHUT\_RDWR);

close(socketFD);

printf("Cliente: Conexion con servidor finalizada\n");

}

catch (std::string msg\_error)

{

fprintf(stderr, "Error en Client.h\n");

perror(msg\_error.c\_str());

exit(EXIT\_FAILURE);

}

}

struct sockaddr\_in constructServerAddressStructure(std::string servIp, int servPort)

{

struct sockaddr\_in servAddr;

memset(&servAddr, 0, sizeof(struct sockaddr\_in));

servAddr.sin\_family = AF\_INET;

servAddr.sin\_port = htons(servPort);

Safe::inet\_pton(AF\_INET, servIp.c\_str(), &servAddr.sin\_addr);

return servAddr;

}

T\* getInstanceOfPeerConnection()

{

return peerConn;

}

};

* + - PeerConnection.h

#ifndef PEERCONNECTION\_H

#define PEERCONNECTION\_H

#include <iostream>

#include <string>

#include "ReceiverSocket.h"

#include "SenderSocket.h"

#include <iostream>

using namespace std;

#define DBG(x) cout << #x << " = " << x << endl;

class PeerConnection

{

protected:

SenderSocket sender;

ReceiverSocket receiver;

bool isActiveConnection = true;

bool disableSender = false;

bool disableReceiver = false;

public:

PeerConnection(int socketFD)

: sender(socketFD), receiver(socketFD)

{

;

}

void sendPackages()

{

try{

do{

sendPackagesHandler();

}while(isActiveConnection && disableSender==false);

}

catch(std::string msg\_error)

{

perror(msg\_error.c\_str());

exit(EXIT\_FAILURE);

}

}

void receivePackages()

{

try

{

do{

receivePackagesHandler();

}while(isActiveConnection && disableReceiver==false);

}

catch(std::string msg\_error)

{

perror(msg\_error.c\_str());

exit(EXIT\_FAILURE);

}

}

// Sobreescribir los virtuals en clases derivadas de PeerConnection

virtual void sendPackagesHandler()

{

printf("Este es un ejemplo de uso de PeerConnection\n");

printf("Que mensaje quiere enviar?\n>");

std::string pack; getline(std::cin, pack);

sender.sendStr(pack);

}

virtual void receivePackagesHandler()

{

std::string pack = receiver.recvStr(1000);

printf("Mensaje recibido:\n%s\n", pack.c\_str());

if(pack.size() == 0)

closeConnection();

}

void closeConnection()

{

this->isActiveConnection = false;

}

};

#endif

* + - ReceiverSocket.h

#include <sys/socket.h>

#include <string>

#include <cmath>

class ReceiverSocket

{

private:

int socketFD;

public:

ReceiverSocket(int socketFD)

{

this->socketFD = socketFD;

}

// TODO: si recibe 0 bytes, se cerro conexion

// Llega a lo mucho x bytes

std::string recvStr(int bytes)

{

std::string s(bytes, '\0');

int cnt = recv(this->socketFD, (char\*)s.c\_str(), bytes, 0);

s = s.substr(0, cnt);

return s;

}

// Se bloquea cuando llega un paquete chiquito

std::string recvBigStr(int bytes)

{

std::string ans, s;

size\_t maxpacksize = 0;

do{

s = recvStr(bytes);

maxpacksize = std::max(maxpacksize, s.size());

ans += s;

}while(s.size() == maxpacksize);

return ans;

}

// Para recibir paquetes tipo: "1 5 Oscar 7 Ramirez"

std::string recvField(int bytes)

{

std::string size = recvStr(bytes);

ignore(1);

std::string field = recvStr(stoi(size));

return field;

}

// La estructura DEBE ser serializable(puro int, char[]. no std::string, std::vector)

template <typename T>

T recvStruct()

{

T mystruct;

recv(this->socketFD, (T\*)&mystruct, sizeof(T), 0); // TODO: Reconocer cuando recv == 0, conexion cerrada

return mystruct;

}

void ignore(int bytes)

{

char trash[bytes];

recv(this->socketFD, trash, bytes, 0);

}

};

* + - SenderSocket.h

#include <sys/socket.h>

#include <string>

class SenderSocket

{

private:

int socketFD;

public:

SenderSocket(int socketFD)

{

this->socketFD = socketFD;

}

void sendStr(std::string str)

{

send(this->socketFD, str.c\_str(), str.size(), 0);

}

template <typename T>

void sendStruct(T mystruct)

{

send(this->socketFD, (T\*)&mystruct, sizeof(T), 0); // TODO: Reconocer cuando recv == 0, conexion cerrada

}

};

* + - Server.h

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <string.h>

#include <thread>

#include "../aux/Safe.h"

#include "PeerConnection.h"

template<typename T>

class Server

{

static\_assert(std::is\_base\_of<PeerConnection, T>::value, "T debe ser derivado de PeerConnection");

private:

int maxNumClients = 10;

public:

// IP y puerto perteneciente a server a conectar

Server() = default;

void newThread\_turnOnServer(int servPort)

{

std::thread th(&Server::turnOnServer, this, servPort);

th.detach();

}

void turnOnServer(int servPort)

{

try{

// Crear socket

int socketFD = Safe::socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

struct sockaddr\_in servAddr = constructServerAddressStructure(servPort);

//Asignar puerto y Habilitar comunicaciones

Safe::bind(socketFD, (struct sockaddr\*)&servAddr, sizeof(struct sockaddr\_in));

Safe::listen(socketFD, this->maxNumClients);

printf("Servidor: abierto!\n");

// Aceptar conexiones de clientes

do{

sockaddr\_in clntAddr;

socklen\_t clntAddrLen = sizeof(clntAddr);

int connectFD = Safe::accept(socketFD, (struct sockaddr\*)&clntAddr, &clntAddrLen);

printf("Servidor: nuevo cliente ");

char clntIp[INET\_ADDRSTRLEN];

if(inet\_ntop(AF\_INET, &clntAddr.sin\_addr.s\_addr, clntIp, INET\_ADDRSTRLEN))

printf("con ip = %s | puerto = %d\n", clntIp, ntohs(clntAddr.sin\_port));

else

printf("no disponible ip y puerto\n");

T\* clientConn = new T(connectFD);

std::thread th1(&T::sendPackages, clientConn);

std::thread th2(&T::receivePackages, clientConn);

th1.detach();

th2.detach();

}while(true);

// Cerrar servidor

shutdown(socketFD, SHUT\_RDWR);

close(socketFD);

printf("Servidor: cerrado!\n");

}

catch (std::string msg\_error)

{

fprintf(stderr, "Error en Server.h\n");

perror(msg\_error.c\_str());

exit(EXIT\_FAILURE);

}

}

struct sockaddr\_in constructServerAddressStructure(int servPort)

{

struct sockaddr\_in servAddr;

memset(&servAddr, 0, sizeof(struct sockaddr\_in));

servAddr.sin\_family = AF\_INET;

servAddr.sin\_port = htons(servPort);

servAddr.sin\_addr.s\_addr = INADDR\_ANY;

return servAddr;

}

void setMaxNumClients(int x)

{

this->maxNumClients = x;

}

};

* Ahora se mostrará el contenido de la carpera Client/
  + client.cpp

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <iostream>

#include <thread>

#include <termios.h>

#include <chrono>

#include <fstream>

#include "peer/Client.h"

#include "peer/Server.h"

#include "aux/SimulatorUserInput.h"

#include "aux/stringhelper.h"

using namespace std;

///////////////////////////////////CONEXIÓN CON SERVER/////////////////////////////////

class ClientConnection: public PeerConnection

{

public:

bool onCreate;

public:

ClientConnection(int socketFD): PeerConnection(socketFD)

{

this->onCreate = true;

}

virtual void sendPackagesHandler() override

{

cout.put('>');

string cmd; getline(cin, cmd);

sender.sendStr(cmd + '\n');

}

virtual void receivePackagesHandler() override

{

string message = receiveCommand();

if(message.empty())

{

printf("Cerraron!\n");

closeConnection();

}

else

{

cout << "\b[" << message << "]\n>";

cout.flush();

}

}

string receiveCommand()

{

string cmd;

for(char c = receiver.recvStr(1)[0]; c!='\0' && c!='\n'; c = receiver.recvStr(1)[0])

cmd += c;

return cmd;

}

};

//////////////////////////////MAIN////////////////////////////////

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

{

SimulatorUserInput sui;

string ip;

int port;

if(argc < 3)

{

printf("Usage: ./client.exe <IP\_SERVER> <PORT\_SERVER>\n");

return 0;

}

if(argc >= 3)

{

ip = argv[1];

port = atoi(argv[2]);

}

if(argc >= 4)

{

printf("TESTING WITH USER INPUT:\n%s\n", argv[3]);

sui.simulate(argv[3]);

}

Client<ClientConnection> sender;

sender.newThread\_connectToServer(ip, port);

sleep(100000); // Con signal se cancela el sleep

printf("Programa terminado\n");

return 0;

}

* Ahora se mostrará el contenido de la carpeta Master/
  + DeviceConnection.h

#ifndef DEVICECONNECTION\_H

#define DEVICECONNECTION\_H

#include <iostream>

#include <vector>

#include <mutex>

#include "peer/Server.h"

#include "peer/Client.h"

#include "SlaveConnection.h"

#include "auxPackCmd.h"

#include "globals.h"

using namespace std;

class DeviceConnection: public PeerConnection

{

private:

enum phaseSystem{PHASE\_SETUP, PHASE\_RUN};

enum typePack{PACK\_UNICAST, PACK\_MULTICAST};

public:

phaseSystem currentPhase;

VerifierCmd verifiercmd;

CreatorPack creatorpack;

MasterExtractorPK extractorpack;

public:

int k; // k = slaves.size();

std::vector< Client<SlaveConnection>\* > slaves;

std::vector<bool> isAlive;

public:

DeviceConnection(int socketFD): PeerConnection(socketFD)

{

currentPhase = PHASE\_SETUP;

}

virtual void sendPackagesHandler() override

{

this->disableSender = true;

}

virtual void receivePackagesHandler() override

{

string cmd = receiveCommandFromClient();

if(cmd.empty()){

printf("Conexion cerrada\n");

closeConnection();

return;

}

typeCmd typecmd = verifiercmd.getTypeOfCmd(cmd);

bool correctPhase = isCorrectPhase(typecmd);

if(correctPhase)

{

switch(typecmd)

{

case CMD\_CONNECT:processConnect(cmd); break;

case CMD\_START: processStart(); break;

case CMD\_CREATE: processOnePk (cmd, typecmd, PACK\_MULTICAST); break;

case CMD\_LINK: processTwoPks(cmd, typecmd, PACK\_MULTICAST); break;

case CMD\_DELETE: processOnePk (cmd, typecmd, PACK\_MULTICAST); break;

case CMD\_UNLINK: processTwoPks(cmd, typecmd, PACK\_MULTICAST); break;

case CMD\_UPDATE: processOnePk (cmd, typecmd, PACK\_UNICAST); break;

case CMD\_EXPLORE:processOnePk (cmd, typecmd, PACK\_UNICAST); break;

case CMD\_SELECT: processOnePk (cmd, typecmd, PACK\_UNICAST); break;

case CMD\_ERROR: processError(); break;

}

}

else{

sendMessageToClient("NOO, COMMAND AT INCORRECT PHASE");

}

}

private:

// Los casos de uso implementados

void processConnect(string cmd);

void processStart();

void processOnePk(string cmd, typeCmd typecmd, typePack typepack);

void processTwoPks(string cmd, typeCmd typecmd, typePack typepack);

void processError();

void keepAlive();

// comunicaciones con Device y Slaves

void sendPackToSlave(int slaveid, string pack);

void sendMessageToClient(string msg);

string receiveMessageFromSlave(int slaveid);

string receiveCommandFromClient();

// comunicaciones avanzadas

void broadcastAllSlaves(string pack);

void multicastTwoSlaves(int slaveid, string pack);

void unicastSlave(int slaveid, string pack);

// funciones auxiliares

int chooseSlave(string pk);

bool isCorrectPhase(typeCmd typecmd);

void setAllSlavesAlive();

void startThreadKeepAlive();

int nextSlaveId(int slaveid);

bool isPingMsg(string msg);

};

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

void DeviceConnection::processConnect(string cmd)

{

vector<string> tokens = simpleTokenizer(cmd);

string ip = tokens[1];

int port = stoi(tokens[2]);

Client<SlaveConnection>\* slave = new Client<SlaveConnection>();

bool isConnected = slave->newThread\_connectToServer(ip, port);

if(isConnected){

slaves.push\_back(slave);

sendMessageToClient("THE SLAVE WAS CONNECTED!");

}

else {

sendMessageToClient("ERROR CONNECTING SLAVE");

}

}

void DeviceConnection::processStart()

{

k = slaves.size();

if(k > 0) {

currentPhase = PHASE\_RUN;

setAllSlavesAlive();

startThreadKeepAlive();

string pack = creatorpack.createPackListOfSlavesIps(slaves);

broadcastAllSlaves(pack);

sendMessageToClient("THE SYSTEM IS UP");

}

else sendMessageToClient("ERROR AT START");

}

void DeviceConnection::processOnePk(string cmd, typeCmd typecmd, typePack typepack)

{

string pack = creatorpack.createPack(typecmd, cmd);

string pk = extractorpack.extractOnePk(typecmd, cmd);

int slaveid = chooseSlave(pk);

if(typepack == PACK\_MULTICAST)

multicastTwoSlaves(slaveid, pack);

else

unicastSlave(slaveid, pack);

}

void DeviceConnection::processTwoPks(string cmd, typeCmd typecmd, typePack typepack)

{

string pk1, pk2;

string pack = creatorpack.createPack(typecmd, cmd);

tie(pk1, pk2) = extractorpack.extractTwoPks(typecmd, cmd);

int slaveid1 = chooseSlave(pk1);

int slaveid2 = chooseSlave(pk2);

if(typepack == PACK\_MULTICAST){

multicastTwoSlaves(slaveid1, pack);

multicastTwoSlaves(slaveid2, pack);

}

else {

unicastSlave(slaveid1, pack);

unicastSlave(slaveid2, pack);

}

}

void DeviceConnection::processError()

{

sendMessageToClient("INCORRECT FORMAT");

}

void DeviceConnection::keepAlive()

{

while(true)

{

cout << "Broadcast pings" << endl;

string pingpack = creatorpack.createPackPing();

broadcastAllSlaves(pingpack);

for(Client<SlaveConnection>\* slave:slaves){

slave->getInstanceOfPeerConnection()->setPingResponse(false);

}

sleep(1); // Espera un segundo

for(int i=0; i<k; ++i){

SlaveConnection\* myslave = slaves[i]->getInstanceOfPeerConnection();

if(isAlive[i] && myslave->timeout())

{

cout << "slave " << i << " ahora no esta vivo" << endl;

isAlive[i] = false;

myslave->closeConn();

}

}

}

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

mutex mutex\_sender;

void DeviceConnection::sendPackToSlave(int slaveid, string pack)

{

mutex\_sender.lock();

cout << "sendPackToSlave: envio paquete["<<pack<<"]" << endl;

slaves[slaveid]->getInstanceOfPeerConnection()->sendPack(pack);

mutex\_sender.unlock();

}

void DeviceConnection::sendMessageToClient(string msg)

{

sender.sendStr(msg + '\n');

}

string DeviceConnection::receiveMessageFromSlave(int slaveid)

{

return slaves[slaveid]->getInstanceOfPeerConnection()->receiveMessage();

}

string DeviceConnection::receiveCommandFromClient()

{

string cmd;

for(char c = receiver.recvChar(); c!='\0' && c!='\n'; c = receiver.recvChar())

cmd += c;

return cmd;

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

void DeviceConnection::broadcastAllSlaves(string pack)

{

for(int i=0; i<k; ++i)

if(isAlive[i])

sendPackToSlave(i, pack);

}

void DeviceConnection::multicastTwoSlaves(int slaveid, string pack)

{

unicastSlave(slaveid, pack);

unicastSlave(nextSlaveId(slaveid), pack);

}

void DeviceConnection::unicastSlave(int slaveid, string pack)

{

if(isAlive[slaveid]){

sendPackToSlave(slaveid, pack);

string msg = receiveMessageFromSlave(slaveid);

sendMessageToClient(msg);

}

else {

sendMessageToClient("THE SLAVE IS OFFLINE");

}

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Puede presentar problemas!!, alerta, solo esta pensado para letras de la [a-z]?

// fuente: https://cp-algorithms.com/string/string-hashing.html

int DeviceConnection::chooseSlave(string pk)

{

const int p = 31;

const int m = 1e9 + 9;

long long hash\_value = 0;

long long p\_pow = 1;

for (char c : pk) {

hash\_value = (hash\_value + (c - 'a' + 1) \* p\_pow) % m;

p\_pow = (p\_pow \* p) % m;

}

hash\_value = abs(hash\_value)%k;

return hash\_value;

}

bool DeviceConnection::isCorrectPhase(typeCmd typecmd)

{

if(typecmd==CMD\_CONNECT || typecmd==CMD\_START){

if(currentPhase == PHASE\_SETUP)

return true;

else return false;

}

else { // Cualquier otro comando

if(currentPhase == PHASE\_RUN)

return true;

else return false;

}

}

void DeviceConnection::setAllSlavesAlive()

{

isAlive.assign(k, true);

}

void DeviceConnection::startThreadKeepAlive()

{

thread th(&DeviceConnection::keepAlive, this);

th.detach();

}

int DeviceConnection::nextSlaveId(int slaveid)

{

return (slaveid+1)%k;

}

#endif

* + SlaveConnection.h

#ifndef SLAVECONNECTION\_H

#define SLAVECONNECTION\_H

#include <iostream>

#include <queue>

#include "peer/Client.h"

#include "globals.h"

using namespace std;

class SlaveConnection: public PeerConnection

{

private:

bool pingResponse;

queue<string> queue\_msgs;

public:

SlaveConnection(int socketFD): PeerConnection(socketFD)

{

;

}

virtual void sendPackagesHandler() override

{

this->disableSender = true;

}

// Recuerda, no debe recibir mensajes entre '\n'

virtual void receivePackagesHandler() override

{

string message;

try{

message = receiver.recvField(3);

}catch(...){

cout << "Recibi mensaje con formato invalido, cerrando conexion! " << endl;

closeConnection();

queue\_msgs.push("Error interno en comunicaciones entre Master & Slaves");

}

if(isPingMsg(message))

{

cout << "-Recibi ping" << endl;

pingResponse = true;

}

else

queue\_msgs.push(message);

}

void sendPack(string pack)

{

sender.sendStr(pack);

}

string receiveMessage()

{

string msg;

while(queue\_msgs.empty()){;}

msg = queue\_msgs.front(); queue\_msgs.pop();

return msg;

}

bool timeout()

{

return !pingResponse;

}

void setPingResponse(bool x)

{

pingResponse = x;

}

bool isPingMsg(string msg)

{

return msg[0]=='p';

}

void closeConn()

{

closeConnection();

}

};

#endif

* + auxPackCmd.h

#ifndef AUXPACKCMD\_H

#define AUXPACKCMD\_H

#include <iostream>

#include "globals.h"

#include "aux/stringhelper.h"

#include "SlaveConnection.h"

using namespace std;

class VerifierCmd

{

public:

typeCmd getTypeOfCmd(string cmd)

{

string word = getFirstWord(cmd);

word = capitalizeWord(word); // Para permitir "connect", ...

if(word == "CONNECT") return CMD\_CONNECT;

else if(word == "START") return CMD\_START;

else if(word == "CREATE") return CMD\_CREATE;

else if(word == "LINK") return CMD\_LINK;

else if(word == "DELETE") return CMD\_DELETE;

else if(word == "UNLINK") return CMD\_UNLINK;

else if(word == "UPDATE") return CMD\_UPDATE;

else if(word == "EXPLORE")return CMD\_EXPLORE;

else if(word == "SELECT") return CMD\_SELECT;

else return CMD\_ERROR;

}

private:

string getFirstWord(string cmd)

{

string s;

for(size\_t i=0; i<cmd.size() && isalpha(cmd[i]); ++i)

s += cmd[i];

return s;

}

string capitalizeWord(string word)

{

string capword;

for(char c: word)

capword += toupper(c);

return capword;

}

};

class CreatorPack

{

public:

string createPack(typeCmd typecmd, string cmd)

{

string pack;

vector<string> tokens = commandTokenizer(cmd);

switch(typecmd)

{

case CMD\_CREATE: pack = processCreate(tokens); break;

case CMD\_LINK: pack = processLink(tokens); break;

case CMD\_DELETE: pack = processDelete(tokens); break;

case CMD\_UNLINK: pack = processUnlink(tokens); break;

case CMD\_UPDATE: pack = processUpdate(tokens); break;

case CMD\_EXPLORE:pack = processExplore(tokens);break;

case CMD\_SELECT: pack = processSelect(tokens); break;

default: break;

}

return pack;

}

string createPackListOfSlavesIps(std::vector< Client<SlaveConnection>\* > slaves)

{

string pack = join({"0", intWithZeros(slaves.size(), 2)});

for(Client<SlaveConnection>\* slave: slaves)

pack += " " + join({pkgF(slave->getIp(), 2)});

return pack;

}

string createPackPing()

{

return "p";

}

private:

string processCreate(vector<string> tokens)

{

string pk = tokens[0];

string content = tokens[1];

return join({"1", pkgF(pk, 2), pkgF(content, 3)});

}

string processLink(vector<string> tokens)

{

string pk1 = tokens[0];

string pk2 = tokens[1];

return join({"2", pkgF(pk1, 2), pkgF(pk2, 2)});

}

string processDelete(vector<string> tokens)

{

string pk = tokens[0];

return join({"3", pkgF(pk, 2)});

}

string processUnlink(vector<string> tokens)

{

string pk1 = tokens[0];

string pk2 = tokens[1];

return join({"4", pkgF(pk1, 2), pkgF(pk2, 2)});

}

string processUpdate(vector<string> tokens)

{

string oldpk = tokens[0];

string newpk = tokens[1];

return join({"5", pkgF(oldpk, 2), pkgF(newpk, 2)});

}

string processExplore(vector<string> tokens)

{

string pk = tokens[0];

int prof = stoi(tokens[1]);

return join({"6", pkgF(pk, 2), intWithZeros(prof, 2)});

}

string processSelect(vector<string> tokens)

{

string pk = tokens[0];

int prof = stoi(tokens[1]);

return join({"7", pkgF(pk, 2), intWithZeros(prof, 2)});

}

// Solo para acortar codigo

string pkgF(string field, int lendigits)

{

return packageField(field, lendigits);

}

};

class MasterExtractorPK

{

public:

string extractOnePk(typeCmd typecmd, string cmd)

{

string pk;

vector<string> tokens = commandTokenizer(cmd);

switch(typecmd)

{

case CMD\_CREATE: pk = tokens[0]; break;

case CMD\_DELETE: pk = tokens[0]; break;

case CMD\_UPDATE: pk = tokens[0]; break;

case CMD\_EXPLORE:pk = tokens[0]; break;

case CMD\_SELECT: pk = tokens[0]; break;

default: break;

}

return pk;

}

pair<string, string> extractTwoPks(typeCmd typecmd, string cmd)

{

string pk1, pk2;

vector<string> tokens = commandTokenizer(cmd);

switch(typecmd)

{

case CMD\_LINK: pk1 = tokens[0]; pk2 = tokens[1]; break;

case CMD\_UNLINK: pk1 = tokens[0]; pk2 = tokens[1]; break;

default: break;

}

return make\_pair(pk1, pk2);

}

};

#endif

* + globals.h

#ifndef GLOBALS\_H

#define GLOBALS\_H

#define DBG(x) cout << #x << " = " << x << endl;

#define DBGVEC(v) cout << #v << " = "; for(const auto&vv: v) cout << vv << ", "; cout << endl;

enum typeCmd{CMD\_CONNECT,

CMD\_START,

CMD\_CREATE,

CMD\_LINK,

CMD\_DELETE,

CMD\_UNLINK,

CMD\_UPDATE,

CMD\_EXPLORE,

CMD\_SELECT,

CMD\_ERROR};

#endif

* + master.cpp

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <iostream>

#include <thread>

#include <termios.h>

#include <chrono>

#include <fstream>

#include "peer/Client.h"

#include "peer/Server.h"

#include "aux/SimulatorUserInput.h"

#include "aux/stringhelper.h"

#include "globals.h"

#include "auxPackCmd.h"

#include "DeviceConnection.h"

using namespace std;

//////////////////////////////MAIN////////////////////////////////

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

{

int port;

if(argc < 2)

{

printf("Parameters: ./master.exe <PORT\_SERVER>\n");

return 0;

}

if(argc >= 2)

{

port = atoi(argv[1]);

}

Server<DeviceConnection> receiver;

bool isServerOpened = receiver.newThread\_turnOnServer(port);

if(isServerOpened){

sleep(100000); // Con signal se desactiva

}

else {

printf("El servidor del Master no pudo iniciar correctamente\n");

}

printf("Programa terminado\n");

return 0;

}

* Ahora se mostrará el contenido de la carpeta Slave:
  + Connections/
    - MasterConnection.h

#ifndef DEVICECONNECTION\_H

#define DEVICECONNECTION\_H

#include <iostream>

#include <vector>

#include <map>

#include <future>

#include <any>

#include "SlaveConnections.h"

#include "../peer/Client.h"

#include "../globals.h"

#include "../Pack/VerifierPacks.h"

#include "../Pack/Packager.h"

#include "../Pack/UnpackagerMaster.h"

using namespace std;

class MasterConnection: public PeerConnection

{

public:

VerifierMasterPack verifierpack;

Packager packager;

UnpackagerMaster unpackager;

public:

MasterConnection(int socketFD): PeerConnection(socketFD), unpackager(socketFD)

{

}

virtual void sendPackagesHandler() override

{

this->disableSender = true;

}

virtual void receivePackagesHandler() override

{

char packid = receiver.recvChar();

if(packid == '\0'){

printf("Conexion con usuario cerrada\n");

closeConnection();

return;

}

cout << "packid = " << packid << endl;

typeMasterPack typepack = verifierpack.getTypeOfPack(packid);

switch(typepack)

{

case PCKMASTER\_START: {thread th(&MasterConnection::responseMaster, this, typepack, unpackager.unpackageStart()); th.detach(); break;}

case PCKMASTER\_CREATE: {thread th(&MasterConnection::responseMaster, this, typepack, unpackager.unpackageCreate()); th.detach(); break;}

case PCKMASTER\_LINK: {thread th(&MasterConnection::responseMaster, this, typepack, unpackager.unpackageLink()); th.detach(); break;}

case PCKMASTER\_DELETE: {thread th(&MasterConnection::responseMaster, this, typepack, unpackager.unpackageDelete()); th.detach(); break;}

case PCKMASTER\_UNLINK: {thread th(&MasterConnection::responseMaster, this, typepack, unpackager.unpackageUnlink()); th.detach(); break;}

case PCKMASTER\_UPDATE: {thread th(&MasterConnection::responseMaster, this, typepack, unpackager.unpackageUpdate()); th.detach(); break;}

case PCKMASTER\_EXPLORE:{thread th(&MasterConnection::responseMaster, this, typepack, unpackager.unpackageExplore()); th.detach(); break;}

case PCKMASTER\_SELECT: {thread th(&MasterConnection::responseMaster, this, typepack, unpackager.unpackageSelect()); th.detach(); break;}

case PCKMASTER\_PING: {thread th(&MasterConnection::responseMaster, this, typepack, unpackager.unpackagePing()); th.detach(); break;}

case PCKMASTER\_ERROR: {thread th(&MasterConnection::responseMaster, this, typepack, unpackager.unpackageError()); th.detach(); break;}

}

cout << "receivePackagesHandler::Continuing\_listening" << endl;

}

private:

void responseMaster(typeMasterPack typepack, any args)

{

future<string> futuremessage;

switch(typepack)

{

case PCKMASTER\_START: futuremessage=async(&MasterConnection::processStart,this, any\_cast<vector<string>>(args)); break;

case PCKMASTER\_CREATE: futuremessage=async(&MasterConnection::processCreate, this, any\_cast<tuple<string,string>>(args)); break;

case PCKMASTER\_LINK: futuremessage=async(&MasterConnection::processLink, this, any\_cast<tuple<string,string>>(args)); break;

case PCKMASTER\_DELETE: futuremessage=async(&MasterConnection::processDelete, this, any\_cast<string>(args));break;

case PCKMASTER\_UNLINK: futuremessage=async(&MasterConnection::processUnlink, this, any\_cast<tuple<string, string>>(args)); break;

case PCKMASTER\_UPDATE: futuremessage=async(&MasterConnection::processUpdate, this, any\_cast<tuple<string, string>>(args)); break;

case PCKMASTER\_EXPLORE:futuremessage=async(&MasterConnection::processExplore, this, any\_cast<tuple<string, int>>(args)); break;

case PCKMASTER\_SELECT: futuremessage=async(&MasterConnection::processSelect, this, any\_cast<tuple<string, int>>(args)); break;

case PCKMASTER\_PING: futuremessage=async(&MasterConnection::processPing, this); break;

case PCKMASTER\_ERROR: futuremessage=async(&MasterConnection::processError, this); break;

}

string message = futuremessage.get();

message = intWithZeros(message.size(), 3) + " " + message;

cout << "Send master message = " << message << endl;

sendMessageToMaster(message);

cout << endl;

}

// Los casos de uso implementados

string processStart(vector<string> ip\_slaves);

string processCreate(tuple<string, string> args);

string processLink(tuple<string, string> args);

string processDelete(string pk1);

string processUnlink(tuple<string, string> args);

string processUpdate(tuple<string, string> args);

string processExplore(tuple<string, int> args);

string processSelect(tuple<string, int> args);

string processPing();

string processError();

// comunicaciones con Master y Slaves

void sendMessageToMaster(string msg);

// comunicaciones internas Slaves y Slaves

void startSlaveid(int slaveid);

void createNode(string pk, string content);

void linkRelationship(string pk1, string pk2);

void unlinkRelationship(string pk1, string pk2);

bool existsNodeInSlaves(string pk);

vector<pair<int,string>> exploreInSlaves(string pk, int prof);

vector<pair<int,pair<string,string>>> selectInSlaves(string pk, int prof);

// base de datos local

bool existsNodeHere(string pk);

bool existLinkHere(string pk1, string pk2);

};

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

string MasterConnection::processStart(vector<string> ip\_slaves)

{

cout << "START: iniciado" << endl;

cout << "START: son " << ip\_slaves.size() << " ips" << endl;

slavesconn.k = ip\_slaves.size();

cout << "START: k = " << slavesconn.k << endl;

for(unsigned int slaveid=0; slaveid<ip\_slaves.size(); ++slaveid)

{

string ip = ip\_slaves[slaveid];

cout << "START: Conectando con ip " << ip << " y puerto " << port\_slave << endl;

Client<SlaveClientConnection>\* client = new Client<SlaveClientConnection>();

bool isconnected = client->newThread\_connectToServer(ip, port\_slave+(debugMode?slaveid:0));

if(isconnected){

slavesconn.slavesclients.push\_back(client);

}

else{

cout << "START: conexion fallida con: " << ip << endl;

exit(0); // termino el programa!!

}

}

for(unsigned int slaveid=0; slaveid<ip\_slaves.size(); ++slaveid){

startSlaveid(slaveid);

}

cout << "START: terminado" << endl;

return "p";

}

string MasterConnection::processCreate(tuple<string, string> args)

{

cout << "Create: iniciado" << endl;

string pk, content; tie(pk, content) = args;

cout << "Create: pk=" << pk << " content=" << content << endl;

if(!existsNodeHere(pk)){

database.createNode(pk, content);

return "THE NODE WAS CREATED!";

}

else{

return "THE NODE ALREADY EXISTS";

}

}

string MasterConnection::processLink(tuple<string, string> args)

{

cout << "Link: iniciado" << endl;

string pk1, pk2; tie(pk1, pk2) = args;

cout << "Link: pk1="<<pk1<<" pk2="<<pk2<< endl;

if(existsNodeInSlaves(pk1) && existsNodeInSlaves(pk2)){

bool isLinked = false;

if(existsNodeHere(pk1)){

if(!existLinkHere(pk1, pk2)){

database.linkRelationship(pk1, pk2);

isLinked = true;

}

}

if(existsNodeHere(pk2)){

if(!existLinkHere(pk2, pk1)){

database.linkRelationship(pk2, pk1);

isLinked = true;

}

}

if(isLinked) return "THE NODES WERE LINKED!";

else return "THE LINK ALREADY EXISTS!";

}

else{

return "SOME NODES DON’T EXIST";

}

}

string MasterConnection::processDelete(string pk1)

{

cout << "Delete: pk=" << pk1 << endl;

if(existsNodeHere(pk1)){

vector<string> relationships = database.getAllRelationships(pk1);

for(string pk2: relationships){

database.unlinkRelationship(pk1, pk2);

unlinkRelationship(pk2, pk1);

}

database.deleteNode(pk1);

return "THE NODE WAS DELETED!";

}

else

return "THE NODE DON'T EXIST";

}

string MasterConnection::processUnlink(tuple<string, string> args)

{

cout << "Unlink: iniciado" << endl;

string pk1, pk2; tie(pk1, pk2) = args;

cout << "Unlink: pk1="<<pk1<<" pk2="<<pk2<< endl;

if(existsNodeInSlaves(pk1) && existsNodeInSlaves(pk2)){

bool isUnlinked = false;

if(existsNodeHere(pk1)){

if(existLinkHere(pk1, pk2)){

database.unlinkRelationship(pk1, pk2);

isUnlinked = true;

}

}

if(existsNodeHere(pk2)){

if(existLinkHere(pk2, pk1)){

database.unlinkRelationship(pk2, pk1);

isUnlinked = true;

}

}

if(isUnlinked)

return "THE NODES WERE UNLINKED!";

else

return "THE LINK DON'T EXIST!";

}

else{

return "SOME NODES DON’T EXIST";

}

}

string MasterConnection::processUpdate(tuple<string, string> args)

{

cout << "Update: iniciado" << endl;

string old\_pk, new\_pk; tie(old\_pk, new\_pk) = args;

cout << "Update: oldpk="<<old\_pk<<" new\_pk="<<new\_pk<< endl;

if(!existsNodeHere(old\_pk)) return "THE OLD NODE DON'T EXIT";

if(existsNodeInSlaves(new\_pk)) return "THE NEW NODE ALREADY EXIST";

createNode(new\_pk, database.getContent(old\_pk));

vector<string> relationships = database.getAllRelationships(old\_pk);

for(string pk2: relationships){

unlinkRelationship(pk2, old\_pk);

linkRelationship(pk2, new\_pk);

linkRelationship(new\_pk, pk2);

}

database.deleteNode(old\_pk);

return "THE NODE WAS UPDATED!";

}

string MasterConnection::processExplore(tuple<string, int> args)

{

cout << "MasterConnection::Explore: iniciado" << endl;

string pk; int prof; tie(pk, prof) = args;

cout << "MasterConnection::Explore: pk="<<pk<<" prof="<<prof<<endl;

string message;

vector<pair<int,string>> explored = exploreInSlaves(pk, prof);

reverse(explored.begin(), explored.end());

for(pair<int,string> p: explored){

cout << "MasterConnection::DATAExplore: rawprof="<<p.first<<" pk="<<p.second<<endl;

message += string(2\*(prof-p.first), '-') + ">" + p.second + "\n";

}

message.erase(message.end()-1);

return message;

}

string MasterConnection::processSelect(tuple<string, int> args)

{

cout << "MasterConnection::Select: iniciado" << endl;

string pk; int prof; tie(pk, prof) = args;

cout << "MasterConnection::Select: pk="<<pk<<" prof="<<prof<<endl;

string message;

vector<pair<int,pair<string,string>>> explored = selectInSlaves(pk, prof); // retorna pair<prof, pk, content>

reverse(explored.begin(), explored.end());

for(pair<int,pair<string,string>> p: explored){

cout << "MasterConnection::DATASelect: rawprof="<<p.first<<" pk="<<p.second.first<<" content="<<p.second.second<<endl;

message += string(2\*(prof-p.first), '-') + ">" + p.second.first + "\t("+p.second.second+")" + "\n";

}

message.erase(message.end()-1);

return message;

}

string MasterConnection::processPing()

{

return "p";

}

string MasterConnection::processError()

{

return "INCORRECT FORMAT";

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

void MasterConnection::sendMessageToMaster(string msg)

{

sender.sendStr(msg);

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

void MasterConnection::startSlaveid(int slaveid)

{

string pack = packager.packageStart(slaveid);

cout << "startSlaveid: envio paquete ["<<pack<<"]" << endl;

cout << "startSlaveid: al slave " << slaveid << endl;

slavesconn.sendPackToSlave(slaveid, pack);

}

void MasterConnection::createNode(string pk, string content)

{

string pack = packager.packageCreate(pk, content);

int slaveid = slavesconn.chooseSlave(pk);

cout << "createNode: envio paquete ["<<pack<<"]" << endl;

cout << "createNode: al slave " << slaveid << endl;

slavesconn.sendPackToSlave(slaveid, pack);

slaveid = slavesconn.nextSlaveid(slaveid);

cout << "createNode: envio paquete ["<<pack<<"]" << endl;

cout << "createNode: al slave " << slaveid << endl;

slavesconn.sendPackToSlave(slaveid, pack);

}

void MasterConnection::linkRelationship(string pk1, string pk2)

{

string pack = packager.packageLink(pk1, pk2);

int slaveid = slavesconn.chooseSlave(pk1);

cout << "createNode: envio paquete ["<<pack<<"]" << endl;

cout << "createNode: al slave " << slaveid << endl;

slavesconn.sendPackToSlave(slaveid, pack);

slaveid = slavesconn.nextSlaveid(slaveid);

cout << "createNode: envio paquete ["<<pack<<"]" << endl;

cout << "createNode: al slave " << slaveid << endl;

slavesconn.sendPackToSlave(slaveid, pack);

}

void MasterConnection::unlinkRelationship(string pk1, string pk2)

{

string pack = packager.packageUnlink(pk1, pk2);

int slaveid = slavesconn.chooseSlave(pk1);

cout << "createNode: envio paquete ["<<pack<<"]" << endl;

cout << "createNode: al slave " << slaveid << endl;

slavesconn.sendPackToSlave(slaveid, pack);

slaveid = slavesconn.nextSlaveid(slaveid);

cout << "createNode: envio paquete ["<<pack<<"]" << endl;

cout << "createNode: al slave " << slaveid << endl;

slavesconn.sendPackToSlave(slaveid, pack);

}

bool MasterConnection::existsNodeInSlaves(string pk)

{

string pack = packager.packageExist(pk);

int slaveid = slavesconn.chooseSlave(pk);

cout << "ExistNode: envio paquete ["<<pack<<"]" << endl;

cout << "ExistNode: al slave " << slaveid << endl;

slavesconn.sendPackToSlave(slaveid, pack);

bool exist = slavesconn.receivePackTrueOrFalseFromSlave(slaveid);

cout << "ExistNode: exist = " << exist << endl;

return exist;

}

vector<pair<int,string>> MasterConnection::exploreInSlaves(string pk, int prof)

{

string pack = packager.packageExploreRequest(pk, prof, {});

int slaveid = slavesconn.chooseSlave(pk);

cout << "MasterConnection::exploreInSlaves: envio paquete ["<<pack<<"]" << endl;

cout << "MasterConnection::exploreInSlaves: pk="<<pk<<" slaveid="<<slaveid<<endl;

slavesconn.sendPackToSlave(slaveid, pack);

vector<pair<int,string>> explored = slavesconn.receivePackExplore(slaveid);

return explored;

}

vector<pair<int,pair<string,string>>> MasterConnection::selectInSlaves(string pk, int prof)

{

string pack = packager.packageSelectRequest(pk, prof, {});

int slaveid = slavesconn.chooseSlave(pk);

cout << "MasterConnection::selectInSlaves: envio paquete ["<<pack<<"]" << endl;

cout << "MasterConnection::selectInSlaves: pk="<<pk<<" slaveid="<<slaveid<<endl;

slavesconn.sendPackToSlave(slaveid, pack);

vector<pair<int,pair<string,string>>> explored = slavesconn.receivePackSelect(slaveid);

return explored;

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

bool MasterConnection::existsNodeHere(string pk)

{

return database.existNode(pk);

}

bool MasterConnection::existLinkHere(string pk1, string pk2)

{

return database.existLink(pk1, pk2);

}

#endif

* + - SlaveClientConnection.h

#ifndef SLAVECLIENTCONNECTION\_H

#define SLAVECLIENTCONNECTION\_H

#include <iostream>

#include <queue>

#include "../peer/PeerConnection.h"

#include "../Pack/Unpackager.h"

using namespace std;

class SlaveClientConnection: public PeerConnection

{

public:

Unpackager unpackager;

public:

SlaveClientConnection(int socketFD): PeerConnection(socketFD), unpackager(socketFD)

{

;

}

virtual void sendPackagesHandler() override

{

;

}

virtual void receivePackagesHandler() override

{

this->disableReceiver = true;

}

void sendPack(string pack)

{

sender.sendStr(pack);

}

// ejemplo: T

// estructura: [cmd]

// longitudes: [1]

bool receivePackTrueOrFalse()

{

char packid = receiver.recvChar();

cout << "SlaveClientConnection(T|F): packid = " << packid << endl;

if(packid == 'T') return true;

else if(packid == 'F') return false;

else exit(0);

}

// ejemplo: 4 01 00 03 C:/

// estructura: [cmd, len\_list, [prof, len\_pk, pk]]

// longitudes: [1, 2 [2, 2 var]+]

vector<pair<int, string>> receivePackExplore()

{

receiver.ignore(1);

vector<pair<int , string>> response = unpackager.unpackageExploreResponse();

cout << "SlaveClientConnection(Explore): son " << response.size() << endl;

return response;

}

// es como el explore, pero + contenido

// ejemplo: 5 01 00 03 C:/ 24 (files: 2, size: 1044MB)

// estructura: [cmd, len\_list, [prof, len\_pk, pk, len\_content, content]]

// longitudes: [1, 2 [2, 2 var, 3 var]+]

vector<pair<int, pair<string, string>>> receivePackSelect()

{

receiver.ignore(1);

vector<pair<int, pair<string, string>>> response = unpackager.unpackageSelectResponse();

cout << "SlaveClientConnection(Select): son " << response.size() << endl;

return response;

}

};

#endif

* + - SlaveConnections.h

#ifndef SLAVECONNECTIONS\_H

#define SLAVECONNECTIONS\_H

#include <vector>

#include "SlaveClientConnection.h"

class SlaveConnections

{

public: // TODO: QUITAR PUBLIC

int k; // k = slaves.size();

vector< Client<SlaveClientConnection>\* > slavesclients;

vector<bool> isAlive; // TODO

int myslaveid = -1; // refiere al slaveid de esta maquina

public:

void sendPackToSlave(int slaveid, string pack)

{

slavesclients[slaveid]->getInstanceOfPeerConnection()->sendPack(pack);

}

bool receivePackTrueOrFalseFromSlave(int slaveid)

{

return slavesclients[slaveid]->getInstanceOfPeerConnection()->receivePackTrueOrFalse();

}

vector<pair<int,string>> receivePackExplore(int slaveid)

{

return slavesclients[slaveid]->getInstanceOfPeerConnection()->receivePackExplore();

}

vector<pair<int, pair<string, string>>> receivePackSelect(int slaveid)

{

return slavesclients[slaveid]->getInstanceOfPeerConnection()->receivePackSelect();

}

// Puede presentar problemas!!, alerta, solo esta pensado para letras de la [a-z]?

// fuente: https://cp-algorithms.com/string/string-hashing.html

int chooseSlave(string pk)

{

const int p = 31;

const int m = 1e9 + 9;

long long hash\_value = 0;

long long p\_pow = 1;

for (char c : pk) {

hash\_value = (hash\_value + (c - 'a' + 1) \* p\_pow) % m;

p\_pow = (p\_pow \* p) % m;

}

hash\_value = abs(hash\_value)%k;

return hash\_value;

}

int nextSlaveid(int slaveid)

{

return (slaveid+1)%k;

}

};

#endif

* + - SlaveServerConnection.h

#ifndef SLAVESERVERCONNECTION\_H

#define SLAVESERVERCONNECTION\_H

#include <iostream>

#include <queue>

#include <thread>

#include "SlaveConnections.h"

#include "../globals.h"

#include "../Pack/VerifierPacks.h"

#include "../Pack/Packager.h"

#include "../Pack/Unpackager.h"

using namespace std;

class SlaveServerConnection: public PeerConnection

{

private:

VerifierSlavePack verifierpack;

Packager packager;

Unpackager unpackager;

public:

SlaveServerConnection(int socketFD): PeerConnection(socketFD), unpackager(socketFD)

{

;

}

virtual void sendPackagesHandler() override

{

this->disableSender = true;

}

virtual void receivePackagesHandler() override

{

char packid = receiver.recvChar();

if(packid == '\0'){

printf("Conexion con slaves cerrada\n");

closeConnection();

return;

}

cout << "SlaveServerConnection: packid = " << packid << endl;

typeSlavePack typepack = verifierpack.getTypeOfPack(packid);

switch(typepack)

{

case PCKSLAVE\_START: {thread th(&SlaveServerConnection::processStart, this, unpackager.unpackageStart()); th.detach();} break;

case PCKSLAVE\_CREATE: {thread th(&SlaveServerConnection::processCreate, this, unpackager.unpackageCreate()); th.detach();} break;

case PCKSLAVE\_EXIST: {thread th(&SlaveServerConnection::processExist, this, unpackager.unpackageExist()); th.detach();} break;

case PCKSLAVE\_LINK: {thread th(&SlaveServerConnection::processLink, this, unpackager.unpackageLink()); th.detach();} break;

case PCKSLAVE\_UNLINK: {thread th(&SlaveServerConnection::processUnlink, this, unpackager.unpackageUnlink()); th.detach();} break;

case PCKSLAVE\_QEXPLORE: {thread th(&SlaveServerConnection::processExplore, this, unpackager.unpackageExploreRequest(), false);th.detach();} break;

case PCKSLAVE\_QSELECT: {thread th(&SlaveServerConnection::processSelect, this, unpackager.unpackageSelectRequest(), false); th.detach();} break;

case PCKSLAVE\_ERROR: processError(); break;

case PCKSLAVE\_SEXPLORE: case PCKSLAVE\_SSELECT: case PCKSLAVE\_TRUE: case PCKSLAVE\_FALSE: break; // ignorados por el compilador, y pertenecen al SlaveClientConnection.h

}

}

void processStart(int slaveid);

void processCreate(tuple<string,string> args);

void processExist(string pk);

void processLink(tuple<string,string> args);

void processUnlink(tuple<string,string> args);

vector<pair<int, string>> processExplore(tuple<string, int, vector<string>> args, bool localinvoc);

vector<pair<int, pair<string, string>>> processSelect(tuple<string, int, vector<string>> args, bool localinvoc);

void processError();

};

void SlaveServerConnection::processStart(int myslaveid)

{

cout << "processStart: Inicia" << endl;

cout << "processStart: myslaveid = " << myslaveid << endl;

slavesconn.myslaveid = myslaveid;

cout << "processStart: Termina" << endl;

}

void SlaveServerConnection::processCreate(tuple<string,string> args)

{

cout << "processCreate: Inicia" << endl;

string pk, content;

tie(pk, content) = args;

cout << "processCreate: PK = " << pk << " content=" << content << endl;

database.createNode(pk, content);

}

void SlaveServerConnection::processExist(string pk)

{

cout << "processExist: Inicia" << endl;

cout << "processExist: PK = " << pk << endl;

string pack;

if(database.existNode(pk)) pack = packager.packageTrue();

else pack = packager.packageFalse();

cout << "SlaveServerConnection: envio paquete = ["<<pack<<"]" << endl;

sender.sendStr(pack);

}

void SlaveServerConnection::processLink(tuple<string,string> args)

{

cout << "processLink: Inicia" << endl;

string pk1, pk2;

tie(pk1, pk2) = args;

cout << "processLink: PK1="<<pk1<<" PK2="<<pk2<< endl;

database.linkRelationship(pk1, pk2);

}

void SlaveServerConnection::processUnlink(tuple<string,string> args)

{

cout << "processUnlink: Inicia" << endl;

string pk1, pk2;

tie(pk1, pk2) = args;

cout << "processUnlink: PK1="<<pk1<<" PK2="<<pk2<< endl;

database.unlinkRelationship(pk1, pk2);

}

vector<pair<int, string>> SlaveServerConnection::processExplore(tuple<string, int, vector<string>> args, bool localinvoc)

{

cout << "SlaveServerConnection::processExplore: Inicia" << endl;

string pk1; int prof; vector<string> visited;

tie(pk1, prof, visited) = args;

cout << "SlaveServerConnection::processExplore: pk1="<<pk1<<" prof="<<prof<<" lenlist="<<visited.size()<<endl;

for(string vis: visited)

cout << "SlaveServerConnection::processExplore(vis): vis=" << vis << endl;

vector<string> new\_visited;

new\_visited = concatVectors(new\_visited, visited);

new\_visited.push\_back(pk1);

vector<pair<int, string>> explored;

if(prof > 0){

for(string pk2: database.getAllRelationships(pk1)){

if(find(new\_visited.begin(), new\_visited.end(), pk2) != new\_visited.end()) continue; // No visito los ya visitados

vector<pair<int, string>> response;

int slaveid = slavesconn.chooseSlave(pk2);

cout << "SlaveServerConnection::processExplore(for): pk2="<<pk2<<" slaveid="<<slaveid<<endl;

if(slaveid != slavesconn.myslaveid){

string pack = packager.packageExploreRequest(pk2, prof-1, new\_visited);

cout << "SlaveServerConnection::processExplore(for): Envio pack["<<pack<<"]" << endl;

slavesconn.sendPackToSlave(slaveid, pack);

response = slavesconn.receivePackExplore(slaveid);

} else {

cout << "SlaveServerConnection::processExplore(for): local invocation"<<endl;

response = processExplore(make\_tuple(pk2, prof-1, new\_visited), true);

}

for(pair<int,string> expl: response)

cout << "SlaveServerConnection::processExplore(for)(expl): prof=" << expl.first << " pk=" << expl.second << endl;

explored = concatVectors(explored, response);

}

}

explored.push\_back(make\_pair(prof, pk1));

cout << "SlaveServerConnection::processExplore: Termina" << endl;

if(!localinvoc) {

string pack = packager.packageExploreResponse(explored);

cout << "SlaveServerConnection: envio paquete = ["<<pack<<"]" << endl;

sender.sendStr(pack);

}

else

cout << "SlaveServerConnection::processExplore: retorno local invocation" << endl;

return explored;

}

vector<pair<int, pair<string, string>>> SlaveServerConnection::processSelect(tuple<string, int, vector<string>> args, bool localinvoc)

{

cout << "SlaveServerConnection::processSelect: Inicia" << endl;

string pk1; int prof; vector<string> visited;

tie(pk1, prof, visited) = args;

cout << "SlaveServerConnection::processSelect: pk="<<pk1<<" prof="<<prof<<" lenlist="<<visited.size()<<endl;

for(string vis: visited)

cout << "SlaveServerConnection::processSelect(vis): vis=" << vis << endl;

vector<string> new\_visited;

new\_visited = concatVectors(new\_visited, visited);

new\_visited.push\_back(pk1);

vector<pair<int, pair<string, string>>> explored;

if(prof > 0){

for(string pk2: database.getAllRelationships(pk1)){

if(find(new\_visited.begin(), new\_visited.end(), pk2) != new\_visited.end()) continue; // No visito los ya visitados

vector<pair<int, pair<string, string>>> response;

int slaveid = slavesconn.chooseSlave(pk2);

cout << "SlaveServerConnection::processSelect(for): pk2="<<pk2<<" slaveid="<<slaveid<<endl;

if(slaveid != slavesconn.myslaveid){

string pack = packager.packageSelectRequest(pk2, prof-1, new\_visited);

cout << "SlaveServerConnection::processSelect(for): Envio pack["<<pack<<"]" << endl;

slavesconn.sendPackToSlave(slaveid, pack);

response = slavesconn.receivePackSelect(slaveid);

} else {

cout << "SlaveServerConnection::processSelect(for): local invocation"<<endl;

response = processSelect(make\_tuple(pk2, prof-1, new\_visited), true);

}

for(pair<int, pair<string, string>> expl: response)

cout << "SlaveServerConnection::processSelect(for)(expl): prof=" << expl.first << " pk=" << expl.second.first << "content="<<expl.second.second << endl;

explored = concatVectors(explored, response);

}

}

explored.push\_back(make\_pair(prof, make\_pair(pk1, database.getContent(pk1))));

cout << "SlaveServerConnection::processSelect: Termina" << endl;

if(!localinvoc) {

string pack = packager.packageSelectResponse(explored);

cout << "SlaveServerConnection: envio paquete = ["<<pack<<"]" << endl;

sender.sendStr(pack);

}

else

cout << "SlaveServerConnection::processSelect: retorno local invocation" << endl;

return explored;

}

void SlaveServerConnection::processError()

{

cout << "SlaveServerConnection: error paquete no reconocido" << endl;

exit(0);

}

#endif

* + Pack/
    - Packager.h

#ifndef PACKAGER\_H

#define PACKAGER\_H

#include <string>

#include <vector>

#include "../aux/stringhelper.h"

#include "VerifierPacks.h"

using namespace std;

class Packager

{

private:

VerifierSlavePack verifier;

public:

// ejemplo: 0 01

// estructura: [cmd, slaveid]

// longitudes: [1, 2]

string packageStart(int slaveid)

{

return join({verifier.getPackid(PCKSLAVE\_START), intWithZeros(slaveid, 2)});

}

// ejemplo: 1 03 C:/ 004 hola

// estructura: [cmd, len\_pk, pk, len\_content, content]

// longitures: [1, 2 var, 3 var]

string packageCreate(string pk, string content)

{

return join({verifier.getPackid(PCKSLAVE\_CREATE), pkgF(pk, 2), pkgF(content, 3)});

}

// ejemplo: 2 03 C:/ 08 C:/a.exe

// estructura: [cmd, len\_pk1, pk1, len\_pk2, pk2]

// longitures: [1, 2 var]

string packageLink(string pk1, string pk2)

{

return join({verifier.getPackid(PCKSLAVE\_LINK), pkgF(pk1, 2), pkgF(pk2, 2)});

}

// ejemplo: 3 03 C:/ 08 C:/a.exe

// estructura: [cmd, len\_pk, pk]

// longitures: [1, 2 var]

string packageUnlink(string pk1, string pk2)

{

return join({verifier.getPackid(PCKSLAVE\_UNLINK), pkgF(pk1, 2), pkgF(pk2, 2)});

}

// ejemplo: 4 03 C:/

// estructura: [cmd, len\_pk, pk]

// longitudes: [1, 2 var]

string packageExist(string pk)

{

return join({verifier.getPackid(PCKSLAVE\_EXIST), pkgF(pk, 2)});

}

// ejemplo: 5 03 C:/ 05 02 01 C 02 C:

// estructura: [cmd, len\_pk pk, prof, len\_list, [len\_pk, pk]]

// longitudes: [1, 2 var, 2, 2 [2 var]+]

string packageExploreRequest(string pk, int prof, vector<string> visited)

{

string pack = join({verifier.getPackid(PCKSLAVE\_QEXPLORE), pkgF(pk, 2), intWithZeros(prof, 2), intWithZeros(visited.size(), 2)});

for(string pk: visited)

pack += " " + pkgF(pk, 2);

return pack;

}

// lo mismo que el explore!! casi...

// ejemplo: 6 03 C:/ 05 02 01 C 02 C:

// estructura: [cmd, len\_pk pk, prof, len\_list, [len\_pk, pk]]

// longitudes: [1, 2 var, 2, 2 [2 var]+]

string packageSelectRequest(string pk, int prof, vector<string> visited)

{

string pack = packageExploreRequest(pk, prof, visited);

pack[0] = verifier.getPackid(PCKSLAVE\_QSELECT)[0];

return pack;

}

// ejemplo: 7 01 00 03 C:/

// estructura: [cmd, len\_list, [prof, len\_pk, pk]]

// longitudes: [1, 2 [2, 2 var]+]

string packageExploreResponse(vector<pair<int,string>> explored)

{

string pack = join({verifier.getPackid(PCKSLAVE\_SEXPLORE), intWithZeros(explored.size(), 2)});

for(auto p: explored){

int prof = p.first; string pk = p.second;

pack += " " + join({intWithZeros(prof, 2), pkgF(pk, 2)});

}

return pack;

}

// es como el explore, pero + contenido

// ejemplo: 7 01 00 03 C:/ 24 (files: 2, size: 1044MB)

// estructura: [cmd, len\_list, [prof, len\_pk, pk, len\_content, content]]

// longitudes: [1, 2 [2, 2 var, 3 var]+]

string packageSelectResponse(vector<pair<int,pair<string,string> > > explored)

{

string pack = join({verifier.getPackid(PCKSLAVE\_SSELECT), intWithZeros(explored.size(), 2)});

for(auto p: explored){

int prof = p.first; string pk = p.second.first; string content = p.second.second;

pack += " " + join({intWithZeros(prof, 2), pkgF(pk, 2), pkgF(content, 3)});

}

return pack;

}

string packageTrue()

{

return verifier.getPackid(PCKSLAVE\_TRUE);

}

string packageFalse()

{

return verifier.getPackid(PCKSLAVE\_FALSE);

}

private:

// Solo para acortar codigo

string pkgF(string field, int lendigits)

{

return packageField(field, lendigits);

}

};

#endif

* + - Unpackager.h

#ifndef UNPACKAGER\_H

#define UNPACKAGER\_H

#include <string>

#include <vector>

#include <tuple>

#include "../peer/ReceiverSocket.h"

#include "../aux/stringhelper.h"

using namespace std;

// atencion: usa receiver, puede afectar la recepcion de packetes

class Unpackager

{

public:

ReceiverSocket receiver;

public:

Unpackager(ReceiverSocket receiver): receiver(receiver)

{

;

}

int unpackageStart()

{

receiver.ignore(1); int slaveid = receiver.recvInt(2);

return slaveid;

}

tuple<string,string> unpackageCreate()

{

receiver.ignore(1); string pk = receiver.recvField(2);

receiver.ignore(1); string content = receiver.recvField(3);

return make\_tuple(pk, content);

}

tuple<string,string> unpackageLink()

{

receiver.ignore(1); string pk1 = receiver.recvField(2);

receiver.ignore(1); string pk2 = receiver.recvField(2);

return make\_tuple(pk1, pk2);

}

tuple<string,string> unpackageUnlink()

{

receiver.ignore(1); string pk1 = receiver.recvField(2);

receiver.ignore(1); string pk2 = receiver.recvField(2);

return make\_tuple(pk1, pk2);

}

string unpackageExist()

{

receiver.ignore(1); string pk = receiver.recvField(2);

return pk;

}

// <pk, prof, vector<pk>>

tuple<string, int, vector<string>> unpackageExploreRequest()

{

receiver.ignore(1); string pk = receiver.recvField(2);

receiver.ignore(1); int prof = receiver.recvInt(2);

receiver.ignore(1); int len\_list = receiver.recvInt(2);

vector<string> visited;

for(int i=0; i<len\_list; ++i){

receiver.ignore(1); string pk = receiver.recvField(2);

visited.push\_back(pk);

}

return make\_tuple(pk, prof, visited);

}

// <pk, prof, vector<pk>>

tuple<string, int, vector<string>> unpackageSelectRequest()

{

return unpackageExploreRequest();

}

vector<pair<int,string>> unpackageExploreResponse()

{

receiver.ignore(1); int len\_list = receiver.recvInt(2);

vector<pair<int,string>> explored;

for(int i=0; i<len\_list; ++i){

receiver.ignore(1); int prof = receiver.recvInt(2);

receiver.ignore(1); string pk = receiver.recvField(2);

explored.push\_back(make\_pair(prof, pk));

}

return explored;

}

vector<pair<int,pair<string,string>>> unpackageSelectResponse()

{

receiver.ignore(1); int len\_list = receiver.recvInt(2);

vector<pair<int,pair<string,string>>> explored;

for(int i=0; i<len\_list; ++i){

receiver.ignore(1); int prof = receiver.recvInt(2);

receiver.ignore(1); string pk = receiver.recvField(2);

receiver.ignore(1); string content = receiver.recvField(3);

explored.push\_back(make\_pair(prof, make\_pair(pk, content)));

}

return explored;

}

};

#endif

* + - UnpackagerMaster.h

#ifndef UNPACKAGERMASTER\_H

#define UNPACKAGERMASTER\_H

#include <string>

#include <vector>

#include <tuple>

#include "../peer/ReceiverSocket.h"

#include "../aux/stringhelper.h"

using namespace std;

// atencion: usa receiver, puede afectar la recepcion de packetes

class UnpackagerMaster

{

public:

ReceiverSocket receiver;

public:

UnpackagerMaster(ReceiverSocket receiver): receiver(receiver)

{

;

}

// ejemplo: [0 02 08 10.0.0.1 08 10.0.0.2]

// estructura: [1, 2 [2 var]+]

// vector<ips>

vector<string> unpackageStart()

{

vector<string> ips\_slaves;

receiver.ignore(1); int lenlist = receiver.recvInt(2);

for(int i=0; i<lenlist; ++i){

receiver.ignore(1); string ip = receiver.recvField(2);

ips\_slaves.push\_back(ip);

}

return ips\_slaves;

}

// ejemplo: [1 03 C:\ 018 size: 12, files: 2]

// estructura: [1,2 var,3 var]

// <pk, content>

tuple<string, string> unpackageCreate()

{

receiver.ignore(1); string pk = receiver.recvField(2);

receiver.ignore(1); string content = receiver.recvField(3);

return make\_tuple(pk, content);

}

// ejemplo: [2 03 C:\ 12 C:\proyectos]

// estructura: [1, 2 var, 2 var]

// <pk1, pk2>

tuple<string, string> unpackageLink()

{

receiver.ignore(1); string pk1 = receiver.recvField(2);

receiver.ignore(1); string pk2 = receiver.recvField(2);

return make\_tuple(pk1, pk2);

}

// ejemplo: [3 12 C:/proyectos]

// <pk>

string unpackageDelete()

{

receiver.ignore(1); string pk = receiver.recvField(2);

return pk;

}

// ejemplo: [4 03 C:\ 12 C:\proyectos]

// estructura: [1, 2 var, 2 var]

// <pk1, pk2>

tuple<string, string> unpackageUnlink()

{

receiver.ignore(1); string pk1 = receiver.recvField(2);

receiver.ignore(1); string pk2 = receiver.recvField(2);

return make\_tuple(pk1, pk2);

}

// ejemplo: [5 03 C:\ 12 C:\proyectos]

// estructura: [1, 2 var, 2 var]

// <old\_pk, new\_pk>

tuple<string, string> unpackageUpdate()

{

receiver.ignore(1); string old\_pk = receiver.recvField(2);

receiver.ignore(1); string new\_pk = receiver.recvField(2);

return make\_tuple(old\_pk, new\_pk);

}

// ejemplo: [6 03 C:\ 05]

// estructura: [1, 2 var, 2]

// <pk, prof>

tuple<string, int> unpackageExplore()

{

receiver.ignore(1); string pk = receiver.recvField(2);

receiver.ignore(1); int prof = receiver.recvInt(2);

return make\_tuple(pk, prof);

}

// ejemplo: [7 03 C:\ 05]

// estructura: [1, 2 var, 2]

// <pk, prof>

tuple<string, int> unpackageSelect()

{

receiver.ignore(1); string pk = receiver.recvField(2);

receiver.ignore(1); int prof = receiver.recvInt(2);

return make\_tuple(pk, prof);

}

// ejemplo: [p]

// estructura: [1]

char unpackagePing()

{

return 'p'; // sin valor

}

char unpackageError()

{

return '?'; // sin valor

}

};

#endif

* + - VerifierPacks.h

#ifndef VERIFIERPACKS\_H

#define VERIFIERPACKS\_H

#include "../globals.h"

class VerifierMasterPack

{

public:

typeMasterPack getTypeOfPack(char packid)

{

switch(packid)

{

case '0': return PCKMASTER\_START;

case '1': return PCKMASTER\_CREATE;

case '2': return PCKMASTER\_LINK;

case '3': return PCKMASTER\_DELETE;

case '4': return PCKMASTER\_UNLINK;

case '5': return PCKMASTER\_UPDATE;

case '6': return PCKMASTER\_EXPLORE;

case '7': return PCKMASTER\_SELECT;

case 'p': return PCKMASTER\_PING;

}

return PCKMASTER\_ERROR;

}

};

class VerifierSlavePack

{

private:

static const char id\_start = '0';

static const char id\_create = '1';

static const char id\_link = '2';

static const char id\_unlink = '3';

static const char id\_exist = '4';

static const char id\_qexplore = '5';

static const char id\_qselect = '6';

static const char id\_sexplore = '7';

static const char id\_sselect = '8';

static const char id\_true = 'T';

static const char id\_false = 'F';

static const char id\_error = '?';

public:

typeSlavePack getTypeOfPack(char packid)

{

switch(packid)

{

case id\_start: return PCKSLAVE\_START;

case id\_create: return PCKSLAVE\_CREATE;

case id\_link: return PCKSLAVE\_LINK;

case id\_unlink: return PCKSLAVE\_UNLINK;

case id\_exist: return PCKSLAVE\_EXIST;

case id\_qexplore:return PCKSLAVE\_QEXPLORE;

case id\_qselect: return PCKSLAVE\_QSELECT;

case id\_sexplore:return PCKSLAVE\_SEXPLORE;

case id\_sselect: return PCKSLAVE\_SSELECT;

case id\_true: return PCKSLAVE\_TRUE;

case id\_false: return PCKSLAVE\_FALSE;

case id\_error: return PCKSLAVE\_ERROR;

}

return PCKSLAVE\_ERROR;

}

string getPackid(typeSlavePack typepack)

{

switch(typepack)

{

case PCKSLAVE\_START: return string(1, id\_start);

case PCKSLAVE\_CREATE: return string(1, id\_create);

case PCKSLAVE\_LINK: return string(1, id\_link);

case PCKSLAVE\_UNLINK: return string(1, id\_unlink);

case PCKSLAVE\_EXIST: return string(1, id\_exist);

case PCKSLAVE\_QEXPLORE: return string(1, id\_qexplore);

case PCKSLAVE\_QSELECT: return string(1, id\_qselect);

case PCKSLAVE\_SEXPLORE: return string(1, id\_sexplore);

case PCKSLAVE\_SSELECT: return string(1, id\_sselect);

case PCKSLAVE\_TRUE: return string(1, id\_true);

case PCKSLAVE\_FALSE: return string(1, id\_false);

case PCKSLAVE\_ERROR: return string(1, id\_error);

}

return to\_string(id\_error);

}

};

#endif

* + Database.h

#ifndef DATABASE\_H

#define DATABASE\_H

#include <map>

#include <vector>

#include <algorithm>

#include <iostream>

using namespace std;

class Database

{

public:

map<string, string> nodes; // map<pk, content>

map<string, vector<string> > relationships;// map<pk1, vector<pk2>>

public:

bool existNode(string pk)

{

return nodes.count(pk) > 0;

}

bool existLink(string pk1, string pk2)

{

if(!existNode(pk1))

return false;

vector<string>& vec = relationships[pk1];

auto it = find(vec.begin(), vec.end(), pk2);

cout << "DATABASE: existe? link pk1=["<<pk1<<"] y" << " pk2=["<<pk2<<"] ";

if(it != vec.end())

{

cout << "SI" << endl;

return true;

}

cout << "NO" << endl;

return false;

}

void createNode(string pk, string content)

{

nodes[pk] = {content};

cout << "DATABASE: creado nodo pk=["<<pk<<"] con contenido=["<<content<<"]" << endl;

}

void deleteNode(string pk)

{

if(existNode(pk))

{

nodes.erase(nodes.find(pk));

cout << "DATABASE: eliminado nodo pk=["<<pk<<"]" << endl;

}

}

void linkRelationship(string pk1, string pk2)

{

relationships[pk1].push\_back(pk2);

cout << "DATABASE: linkeado nodos pk1=["<<pk1<<"]" << " pk2=["<<pk2<<"]" << endl;

}

void unlinkRelationship(string pk1, string pk2)

{

vector<string>& vec = relationships[pk1];

auto it = find(vec.begin(), vec.end(), pk2);

if(it != vec.end()){

vec.erase(it);

cout << "DATABASE: deslinkeados nodos pk1=["<<pk1<<"]" << " pk2=["<<pk2<<"]" << endl;

}

}

vector<string> getAllRelationships(string pk)

{

return relationships[pk];

}

string getContent(string pk)

{

return nodes[pk];

}

};

#endif

* + globals.h

#ifndef GLOBALS\_H

#define GLOBALS\_H

#include "Database.h"

#include "Connections/SlaveConnections.h"

enum typeMasterPack{PCKMASTER\_START,

PCKMASTER\_CREATE,

PCKMASTER\_LINK,

PCKMASTER\_DELETE,

PCKMASTER\_UNLINK,

PCKMASTER\_UPDATE,

PCKMASTER\_EXPLORE,

PCKMASTER\_SELECT,

PCKMASTER\_PING,

PCKMASTER\_ERROR};

enum typeSlavePack{PCKSLAVE\_START,

PCKSLAVE\_CREATE,

PCKSLAVE\_LINK,

PCKSLAVE\_UNLINK,

PCKSLAVE\_EXIST,

PCKSLAVE\_QEXPLORE, // Q = reQuest

PCKSLAVE\_QSELECT,

PCKSLAVE\_SEXPLORE, // S = reSponse

PCKSLAVE\_SSELECT,

PCKSLAVE\_TRUE,

PCKSLAVE\_FALSE,

PCKSLAVE\_ERROR};

int port\_master, port\_slave;

Database database;

SlaveConnections slavesconn;

template<typename T>

vector<T> concatVectors(vector<T> v1, vector<T> v2)

{

vector<T> ans(v1.begin(), v1.end());

ans.insert(ans.end(), v2.begin(), v2.end());

return ans;

}

// debug

int slaveid = 0;

bool debugMode = false;

//

#endif

* + slave.cpp

#include <stdio.h>

#include <iostream>

#include "peer/Client.h"

#include "peer/Server.h"

#include "Connections/MasterConnection.h"

#include "Connections/SlaveServerConnection.h"

using namespace std;

void argvHandler(int argc, const char\*\* argv);

//////////////////////////////MAIN////////////////////////////////

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

{

argvHandler(argc, argv);

Server<MasterConnection> masterconn;

Server<SlaveServerConnection> slaveconn;

bool isopen\_master = masterconn.newThread\_turnOnServer(port\_master);

bool isopen\_slave = slaveconn.newThread\_turnOnServer(port\_slave+slaveid); // slaveid es de debug

if(isopen\_master && isopen\_slave){

sleep(100000); // Paro este thread. Con signal se desactiva esto

}

else {

if(!isopen\_master) printf("El servidor para el Master no pudo iniciar\n");

if(!isopen\_slave) printf("El servidor para los Slave no pudo iniciar\n");

}

printf("Programa terminado\n");

return 0;

}

//////////////////////////////FUNCTIONS////////////////////////////////

void argvHandler(int argc, const char\*\* argv)

{

if(argc < 3)

{

printf("Parameters: ./slave.exe <PORT\_SRVMASTER> <PORT\_SRVSLAVES> [<SLAVE\_ID>]\n");

exit(0);

}

if(argc >=3)

{

port\_master = atoi(argv[1]);

port\_slave = atoi(argv[2]);

}

if(argc >= 4)

{

slaveid = atoi(argv[3]);

debugMode = true;

printf("Fase de debug\n");

}

}

TODO EL CÓDIGO SE ENCUENTRA ONLINE EN: https://github.com/oscaramos/Redes\_ProyectoFinal