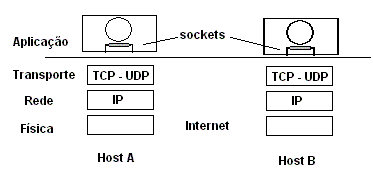


###### 5º Ciência da Computação (CC)

**“DESENVOLVIMENTO DE UMA FERRAMENTA PARA COMUNICAÇÃO EM REDE”**

****

**RA - C082HG8 - ANDERSON MISSON  
RA - C05FEH0 - DANILO DE ARAUJO VASCONCELLOS  
RA - C234125 - MARCIO BARBADO JUNIOR  
RA - C17EAH0 - MARCOS ROBERTO DE MENEZES  
RA - 9791353 - ROGER COUDONARAKIS**

**ÍNDICE**

**1. OBJETIVO\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_04**

**2. INTRODUÇÃO\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_05**

**3. CONCEITOS GERAIS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_06**

**4. TÉCNICAS ULTILIZADAS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_07  
 4.1 SOCKET\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_08  
 4.2 - BLIND\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_08  
 4.3 - LISTEN\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_09  
 4.4 - ACCEPT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_09  
 4.5 - CONNECT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10  
 4.6 - GETHOSTBYNAME E GETHOSTBYADDR\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10  
 4.7 - PROTOCOLOS E ENDEREÇOS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_11**

**5. PROJETOS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_13**

**6. LINHA DE CÓDIGO\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_14**

**7. BIBLIOGRAFIA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_22**

**8. FICHA DA ATIVIDADE PRATICA SUPERVISIONADA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_23**

**1. OBJETIVO**

A comunicação vem se tornando cada vez mais parte do nosso dia a dia e fazendo o nosso mundo girar, uma das ferramentas mais uteis que o homem

desenvolveu ao decorrer dos séculos, a vantagem de poder se comunicar com

facilidade, em qualquer lugar e momento, sem se importar com a distância,

possibilitou a globalização, tornar nosso mundo uma rede completamente

conectada faz economizarmos tempo, dinheiro, facilitando o conhecimento

mantendo um crescimento acelerado da tecnologia, mensagens que

demoravam anos para serem entregues hoje não passam de segundos, não

importa o lugar o planeta em que estamos, podemos transmitir imagens, sons,

textos e vídeos, não precisamos esperar dias, meses ou até mesmo anos em

busca de uma informação, podemos simplesmente pesquisar o que precisamos nos computadores, tablets, celulares e etc... e teremos a informação em um instante.

Pela ciência de saber como poder utilizar das linguagens de programação atuais e dos algoritmos para desenvolver uma plataforma de comunicação, implementar e estudar a técnica usada para possibilitar a comunicação entre as máquinas, explorara a funcionalidade das camadas OSI, TCP/IP e os seus protocolos de rede, usando conceitos “thread” um fluxo de controle sequencial dentro de um programa, mantendo a performance entre “Jobs” e sockets conceito utilizado para manter uma comunicação entre cliente/servidor trocando informações como endereço IP e a porta do servidor, e programação orientada a objeto vamos construir uma plataforma de comunicação on-line e instantânea possibilitando assim a comunicação entre vários usuários em uma mesma sala formando assim um grupo ou até conversas privada entre usuários.

O objetivo geral desta atividade prática supervisionada é colocar em prática o que aprendemos a decorrer do 5º semestre no curso ciência da computação, no qual engloba as respectivas matérias: redes de computadores, arquitetura de computadores, sistemas operacionais. Todo conteúdo vertente dessas

matérias serão praticados e exercitados no trabalho expondo assim o

conhecimento adquirido pelos estudantes.

**2. INTRODUÇÃO.**

Socket é uma interface de programação de aplicativo (API) para sockets de Internet e soquetes de domínio Unix , utilizados para comunicação entre processos (IPC).

É geralmente implementada como uma biblioteca de módulos que podem ser ligadas. Originou-se com o 4.2BSD Unix lançado em 1983.

A API representa um socket uma representação abstrata ( identificador ) para o ponto de extremidade local de uma conexão como um descritor de arquivo ( identificador de arquivo ), por causa da filosofia Unix que "tudo é um arquivo" e as analogias entre ligações e arquivos: você pode ler, escrever, abrir e fechar ambas. Na prática, as diferenças fazer a analogia tensas, e uma vez envia e recebe através de uma

ligação.

A API evoluiu com pouca modificação de uma de facto padrão em um componente do POSIX especificação. Portanto, o termo sockets POSIX é essencialmente sinônimo de soquetes Berkeley . Eles também são conhecidos como BSD sockets , reconhecendo a primeira implementação na Berkeley Software Distribution .

**3.CONCEITOS GERAIS**

Na década de 1980 nos Estados Unidos a ARPA (Advanced Research Projects Agency of the Department of Defense) deu à Berkeley, Universidade da California a responsabilidade de construir um sistema operacional que pudesse ser utilizado no suporte à ARPAnet, que é o antecessor da internet atual.

Neste sentido, foi desenvolvida, com uma fonte genuina uma interface e adicionada ao sistema operacional, Unix BSD (Berkeley Software Distribution). Tal interface tinha justamente a função de suporte a comunicação em rede. Esta interface ficou então conhecida como Berkeley Sockets Interface, e é a base para a maioria das interfaces entre protocolos de internet TCP/IP existente.

Existe uma nomenclatura específica para se referir a cada “lado” da comunicação. Temos o servidor, que fica esperando por conexões de entrada e que fornece certos tipos de serviços à outra parte. Já o Cliente vem a ser quem solicita a conexão ao servidor para fazer alguma requisição, algum pedido. É importante dizer que não é o computador que distingue quem é servidor e quem é cliente, mas sim a forma como certo programa usa os sockets. Às vezes também se faz confusão no fato de se pensar que um servidor precisa ser um mainframe. Desktops como os que usamos em casa funcionam tanto como cliente quanto como servidor, e é o que ocorre freqüentemente.

Cada socket tem um endereço único na internet. Este endereço é formado por um número IP e por um número de porta. Devido às grandes dimensões da internet, não há como uma pessoa, ou mesmo uma máquina, saber o endereço de todas as outras. Para resolver este problema foi criado protocolo DNS (Domain Name Service). Este protocolo tem a função de traduzir os nomes ou endereços de alto nível das máquinas para o seu respectivo número IP. Assim, ao se passar o endereço de um socket de um servidor, não se passa diretamente seu número IP, mas sim um nome mais fácil de recordar e então o DNS traduz para o endereço real, ou endereço IP.

Os sockets podem ser usados para comunicação via qualquer um dos protocolos UDP ou TCP. Assim, é possível termos tanto comunicação orientada a conexão (via TCP), quanta não orientada a conexão (via UDP). O socket abstrai esse conceito, permitindo assim a utilização de qualquer um dos meios.

**4. TÉCNICAS ULTILIZADAS**

Esta lista é um resumo das funções ou métodos fornecidos pela biblioteca Socket API:

\* socket() cria uma nova socket(Tomada) de um certo tipo de socket, identificado por um número inteiro, e atribui os recursos do sistema para isso.

\* bind()  é normalmente utilizado no lado do servidor, e associa um soquete com uma estrutura endereço de soquete, ou seja, um número de porta local especificado e o endereço IP.

\* listen() é usado no lado do servidor, e faz com que um soquete TCP limite para entrar estado de escuta.   
  
\* connect() é usado no lado do cliente, e atribui um número de porta local livre para um soquete. No caso de um soquete TCP, que provoca uma tentativa de estabelecer uma nova conexão TCP.

\* accept() é utilizado no lado do servidor. Ele aceita uma tentativa de entrada recebida para criar uma nova conexão TCP do cliente remoto, e cria um novo socket associado com o par de endereços de socket desta conexão.

\* send() e recv(), ou write() e read(), ou sendto() e recvfrom(),são usados ​​para enviar e receber dados de / para um soquete remoto.   
  
\* close() faz com que o sistema para liberar recursos alocados a uma tomada. Em caso de TCP, a conexão é encerrada.

\* gethostbyname() e gethostbyaddr() são usados ​​para resolver nomes de host e endereços. IPv4 somente.   
  
\* select() é usado para pend, à espera de uma ou mais de uma lista fornecida de soquetes para estar pronto para ler, pronto para escrever, ou que tenham erros.

\* poll() é usado para verificar o estado de uma tomada em um conjunto de tomadas. O conjunto pode ser testado para ver se qualquer tomada pode ser escrito, ler ou se ocorreu um erro.

\* getsockopt()é usado para recuperar o valor atual de uma opção de Socket especial para o Socket especificado.   
  
\* setsockopt()  é usado para definir uma opção de socket em particular para o socket especificado.

**4.1 - SOCKET**

A função socket () cria um ponto de extremidade de comunicação e retorna um descritor de arquivo para o socket. Socket () recebe três argumentos:

* *domínio* , que especifica a família do protocolo do socket criado. Por exemplo:
  + AF\_INET para protocolo de rede IPv4 ou
  + AF\_INET6 para IPv6 .
  + AF\_UNIX para socket local (usando um arquivo).
* *tipo* , um dos seguintes:
  + SOCK\_STREAM (serviço orientado ao fluxo confiável ou córrego Sockets )
  + SOCK\_DGRAM (serviço de data gramas ou Dataram )
  + SOCK\_SEQPACKET (confiável serviço de pacotes sequenciados), ou
  + SOCK\_RAW (protocolos matérias no topo da camada de rede).
* *Protocolo* especificando o protocolo de transporte actual a usar. Os mais comuns são IPPROTO\_TCP , IPPROTO\_SCTP , IPPROTO\_UDP , IPPROTO\_DCCP . Estes protocolos são especificados no arquivo *netinet / in.h* . O valor 0 pode ser utilizada para seleccionar um protocolo padrão do domínio e do tipo selecionado.

A função retorna -1 se ocorreu um erro. Caso contrário, ele retorna um inteiro que representa o descritor de recém-atribuído.

**4.2 - BLIND**   
 bind () atribui um soquete para um endereço. Quando um soquete é criado usando socket () , ele só é dada uma família de protocolos, mas que não possuam um endereço. Esta associação com um endereço deve ser realizada com o bind (chamada) do sistema antes da tomada pode aceitar conexões para outros hospedeiros. Bind () recebe três argumentos:

* sockfd , um descritor que representa o soquete para realizar a ligação por diante.
* my\_addr , um ponteiro para uma sockaddr estrutura que representa o endereço para se ligar a.
* addrlen , um socklen\_t campo especifica o tamanho do sockaddr estrutura.

Bind () retorna 0 em caso de sucesso e -1 se ocorrer um erro.

**4.3 - LISTEN**

Depois de um soquete foi associado a um endereço, listen () prepara para conexões de entrada. No entanto, isso só é necessário para os modos de dados (orientados a conexão) orientada para o fluxo, ou seja, para tipos de soquete ( SOCK\_STREAM , SOCK\_SEQPACKET ). listen () requer dois argumentos:

* sockfd , um descritor de socket válido.
* atraso , um número inteiro que representa o número de conexões pendentes que podem ser enfileiradas, em qualquer altura. O sistema operacional normalmente coloca um limite para este valor.

Depois que uma conexão é aceito, ele é retirado da fila. Em caso de sucesso, 0 é retornado. Se ocorrer um erro, -1 é retornado.

**4.4 - ACCEPT**

Quando um aplicativo está aguardando as conexões orientadas para o fluxo de outros hosts, ele é notificado de tais eventos (cf. [select ()](https://en.wikipedia.org/wiki/Select_(Unix)" \o "Selecione (Unix)) função) e deve inicializar a conexão usando aaceitar () função. A função accept () cria um novo socket para cada ligação e remove a conexão da fila de escuta. Demora os seguintes argumentos:

* sockfd , o descritor de socket de escuta que tem a conexão em fila.
* cliaddr , um ponteiro para uma estrutura sockaddr para receber informações de endereço do cliente.
* addrlen , um ponteiro para uma socklen\_t localização especifica que o tamanho da estrutura de endereço do cliente passou para aceitar (). Ao aceitar () retorna, este local indica quantos bytes da estrutura foram realmente utilizados.

A função accept () retorna o novo descritor de socket para a conexão aceita, ou -1 se ocorrer um erro. Todos ainda mais a comunicação com o host remoto agora ocorre através deste novo soquete.

sockets de datagramas não exigem processamento por aceitar () uma vez que o receptor pode responder imediatamente ao pedido usando o soquete de escuta.

**4.5 - CONNECT**

O connect () chamada de sistema estabelece uma ligação de comunicação directa a um host remoto específico identificado por seu endereço através de uma tomada, identificado por seu descritor de arquivo.

Ao usar um [conexão-orientado](https://en.wikipedia.org/wiki/Connection-oriented_communication" \o "a comunicação orientada à conexão) protocolo, este estabelece uma conexão . Certos tipos de protocolos são sem conexão , mais notavelmente o [User Datagram Protocol](https://en.wikipedia.org/wiki/User_Datagram_Protocol" \o "User Datagram Protocol) . Quando usado com protocolos sem conexão, conecte define o endereço remoto para enviar e receber dados, permitindo o uso de funções como send () e recv () . Nestes casos, a função de conexão bloqueia a recepção de datagramas de outras fontes.

connect () retorna um inteiro que representa o código de erro: 0 representa sucesso, enquanto -1 representa um erro. Historicamente, nos sistemas derivados do BSD, o estado de um descritor de socket é indefinido se a chamada para se conectar () falha (como é especificado no Single UNIX Specification), portanto, aplicações portáteis deve fechar o descritor de socket imediatamente e obter um novo descritor com socket (), no caso da chamada para connect () falhar.

**4.6 - GETHOSTBYNAME E GETHOSTBYADDR**

O gethostbyname () e gethostbyaddr () funções são usadas para resolver os nomes de host e endereços no [sistema de nome de domínio](https://en.wikipedia.org/wiki/Domain_name_system) ou de outros mecanismos resolvedor do host local (por exemplo, / etc / hosts de pesquisa). Eles retornam um ponteiro para um objeto do tipo *hostent struct* , que descreve um [protocolo Internet](https://en.wikipedia.org/wiki/Internet_Protocol) host. As funções de tomar os seguintes argumentos:

* *Nome* especifica o nome do host. Por exemplo: www.wikipedia.org
* *addr* especifica um ponteiro para uma *in\_addr estrutura* que contém o endereço do hospedeiro.
* *len* especifica o comprimento, em bytes, do *endereço* .
* *Tipo* especifica o tipo de família de endereço (por exemplo, AF\_INET) do endereço do host.

As funções retornar um ponteiro NULL em caso de erro, caso em que o número inteiro externa *h\_errno* pode ser verificado para ver se esta é uma falha temporária ou um hospedeiro inválido ou desconhecido. Caso contrário, uma válida *struct hostent*  é retornado.

Estas funções não são estritamente um componente da BSD soquete API, mas são muitas vezes utilizados em conjunto com as funções de API. Além disso, estas funções são agora considerados interfaces de legado para consultar o sistema de nome de domínio. Novas funções que são completamente protocolo agnóstica (suporte IPv6) foram definidos. Estes são nova função [getaddrinfo () e getnameinfo ()](https://en.wikipedia.org/wiki/Getaddrinfo" \o "getaddrinfo) , e baseiam-se em um novo [*addrinfo*](https://en.wikipedia.org/wiki/Addrinfo) estrutura de dados.

**4.7 - PROTOCOLOS E ENDEREÇOS**

API é uma interface de rede para Unix geral e permite a utilização de vários protocolos de rede e arquitecturas de endereçamento.

As listas a seguir uma amostra de protocolo famílias (precedido pelo identificador simbólico padrão) definidas em um moderno Linux ou BSD execução:

PF\_LOCAL, PF\_UNIX, PF\_FILE - Local para hospedar (tubos e arquivo de domínio)

PF\_INET - Internet Protocol versão 4

PF\_AX25 - Amateur Radio AX.25

PF\_IPX - Da Novell Internetwork Packet Exchange

PF\_APPLETALK - appletalk

PF\_NETROM - Radioamador netrom (relacionado com AX.25)

PF\_BRIDGE - Multiprotocol ponte

PF\_ATMPVC - De transferência assíncrona Modo circuitos virtuais permanentes

PF\_ATMSVC - Asynchronous Transfer Mode circuitos virtuais comutados

PF\_INET6 - Protocolo de Internet versão 6

PF\_DECnet - Reservados para o projeto DECnet

PF\_NETBEUI - Reservado para 802.2LLC projeto

PF\_SECURITY - callback de segurança pseudo AF

PF\_KEY - PF\_KEY API de gerenciamento de chaves

PF\_NETLINK, PF\_ROUTE - roteamento API

PF\_PACKET - soquetes de captura de pacotes

PF\_ECONET - Acorn Econet

PF\_SNA - Linux IBM Systems Network Architecture (SNA) Projeto

PF\_IRDA - IrDA soquetes

PF\_PPPOX - PPP sobre X soquetes

PF\_WANPIPE - soquetes API Sangoma Wanpipe

PF\_BLUETOOTH - Bluetooth soquetes

A tomada para comunicações usando qualquer família é criado com o socket () função, especificando a família do protocolo desejado ( PF\_ -identifier) ​​como um argumento.

O conceito de design original do interface socket distinção entre tipos de protocolo (famílias) e os tipos de endereços específicos que cada um pode usar. Foi previsto que a família de protocolos pode ter vários tipos de endereço. Tipos de endereços foram definidos por constantes simbólicas adicionais, usando o prefixo AF vez de PF . Os AF -identifiers destinam-se a todas as estruturas de dados que tratam especificamente o tipo de endereço e não a família do protocolo. No entanto, este conceito de separação de protocolo e tipo de endereço não encontrou apoio à implementação e os AF -constants foram definidos pelo identificador de protocolo correspondente, deixando a distinção entre AF e PF constantes como um argumento técnico de nenhuma consequência prática. Na verdade, muita confusão existe no uso adequado de ambas as formas. A especificação POSIX.1-2008

**5. PROJETOS**

**6. LINHA DE CODIGO (JAVA)**

import java.io.BufferedReader;  
import java.io.FileReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.InetAddress;

import java.net.NetworkInterface;

import java.net.URL;

import java.util.ArrayList;

import java.util.Enumeration;

import java.util.Scanner;

public class SimpleNetworkCommunicator{

public SimpleNetworkCommunicator(){

ArrayList<String> str\_AddressesArrayList;

ArrayList<String> str\_PrivateAddressesArrayList;

BufferedReader inBufferedReader;

String str\_LoopbackAddress;

String str\_public\_IP;

URL testURL;

str\_AddressesArrayList = new ArrayList<String>();

/\* Private address. \*/

Enumeration<NetworkInterface> enumerationNetworkInterface = NetworkInterface.getNetworkInterfaces();

str\_PrivateAddressesArrayList = new ArrayList<String>();

/\* Loopback address. \*/

str\_LoopbackAddress = InetAddress.getLocalHost().getHostAddress();

/\* Public address. \*/

testURL= new URL("http://checkip.amazonaws.com");

inBufferedReader = new BufferedReader(new InputStreamReader(testURL.openStream()));

str\_public\_IP = inBufferedReader.readLine();

while(enumerationNetworkInterface.hasMoreElements()){

NetworkInterface networkInterface = enumerationNetworkInterface.nextElement();

Enumeration<InetAddress> enumerationInetAddress = networkInterface.getInetAddresses();

while (enumerationInetAddress.hasMoreElements()){

InetAddress inetAddr = enumerationInetAddress.nextElement();

str\_PrivateAddressesArrayList.add(inetAddr.getHostAddress());

}

}

inBufferedReader.close();

str\_AddressesArrayList.add(str\_LoopbackAddress);

str\_AddressesArrayList.add(str\_PrivateAddressesArrayList.get(2));

str\_AddressesArrayList.add(str\_public\_IP);

return str\_AddressesArrayList;

} /\* Method obtainExternalIP() end. \*/

static BufferedReader parseCSV(String str\_File) throws Exception{

String str\_CSVFile;

BufferedReader linesBufferedReader;

str\_CSVFile = str\_File;

// linesBufferedReader = null;

// linesBufferedReader = new BufferedReader(new FileReader(str\_CSVFile));

try{

/\* BufferedReader used with FileReader improve reading performance. \*/

linesBufferedReader = new BufferedReader(new FileReader(str\_CSVFile));

return linesBufferedReader;

}

catch(Exception exceptionException){

System.err.print("\nThere is a problem with file " + str\_CSVFile + ". Maybe it was not found.\n" +

"Working directory is " + System.getProperty("user.dir") + "\n");

exceptionException.printStackTrace();

}

return null;

} /\* Method parseCSV() end. \*/

ArrayList<Integer> groupFilePortsIntegerArrayList;

/\* All addresses on the group file. \*/

ArrayList<String> str\_GroupFileAddressesArrayList;

/\* All of the addresses which are available to local peer. \*/

ArrayList<String> str\_LocalAddressesArrayList;

ArrayList<String[]> str\_GroupFileValuesArrayList;

BufferedReader readCSVBufferedReader;

int int\_action;

int int\_error\_code;

int int\_index;

OperatingSystemSpecific operatingSystemSpecific;

Peer peerPeer;

Scanner scannerScanner;

String str\_topo;

String str\_usage;

String str\_group\_file;

String str\_id\_file;

String str\_Line;

String[] str\_IDFileValuesArray;

StringBuilder messageStringBuilder;

StringBuilder[] discoverOSStringBuilderArray;

groupFilePortsIntegerArrayList = new ArrayList<Integer>();

str\_GroupFileAddressesArrayList = new ArrayList<String>();

str\_LocalAddressesArrayList = new ArrayList<String>();

str\_GroupFileValuesArrayList = new ArrayList<String[]>();

int\_action = -1;

int\_error\_code = 0;

int\_index = 0;

str\_topo = " ------------------------------------------------------\n" +

"| |\n" +

"| ▄██▀██▄ ██ ██ ▄██▀██▄ |\n" +

"| ▀██▄▄ ███▄ ██ ██ |\n" +

"| ▀▀██▄ ██ ▀███ ██ |\n" +

"| ▀██▄██▀ ██ ██ ▀██▄██▀ SIMPLE NETWORK COMMUNICATOR |\n" +

"| |\n" +

"| license: GNU GPL v2 |\n" +

"| |\n" +

" ------------------------------------------------------\n";

str\_usage = "usage example for one-to-one communication:\n" +

"java SimpleNetworkCommunicator [ADDRESS] [PORT]\n";

str\_group\_file = "src/br/com/bdslabs/snc/dev/.group.csv";

str\_id\_file = "src/br/com/bdslabs/snc/dev/.id.csv";

messageStringBuilder = new StringBuilder();

str\_Line = "";

operatingSystemSpecific = new OperatingSystemSpecific();

peerPeer = new Peer();

scannerScanner = new Scanner(System.in);

discoverOSStringBuilderArray = operatingSystemSpecific.identifyOSStringBuilderArray();

/\* Reads id file (one line). \*/

readCSVBufferedReader = parseCSV(str\_id\_file);

str\_IDFileValuesArray = (str\_Line = readCSVBufferedReader.readLine()) != null ? str\_Line.split("\" \*, \*\"") : null;

/\* It was open on initialization section. \*/

readCSVBufferedReader.close();

if (discoverOSStringBuilderArray[1].equals("FLAG")){

int\_error\_code = 6;

System.err.print("\nException " + int\_error\_code + ": operating system not allowed.\n");

System.exit(int\_error\_code);

}

operatingSystemSpecific.clearConsole(discoverOSStringBuilderArray[0].indexOf("indows") != -1);

str\_LocalAddressesArrayList = obtainAvailableLocalAddresses();

System.out.print(str\_topo +

"\nHello " + System.getProperty("user.name", str\_IDFileValuesArray[1]) + ".\n" +

discoverOSStringBuilderArray[0] +

"loopback IP in use:\t" + str\_LocalAddressesArrayList.get(0) + "\n" +

"internal IP in use:\t" + str\_LocalAddressesArrayList.get(1) + "\n" +

"external IP in use:\t" + str\_LocalAddressesArrayList.get(2) + "\n");

/\* Use address and port from command line. \*/

if (args.length == 2){

/\* TODO: handle input. \*/

peerPeer = new Peer(args[0], Integer.parseInt(args[1]));

}

/\* Use addresses and ports from file. \*/

else if (args.length == 0){

/\* Reads group file (multiple lines). \*/

readCSVBufferedReader = parseCSV(str\_group\_file);

System.out.print("\n" +

"G R O U P\n" +

"\n");

while ((str\_Line = readCSVBufferedReader.readLine()) != null){

/\* Each line is added, in a FIFO scheme. \*/

str\_GroupFileValuesArrayList.add(str\_Line.split("\" \*, \*\""));

str\_GroupFileAddressesArrayList.add(str\_GroupFileValuesArrayList.get(int\_index)[4]);

groupFilePortsIntegerArrayList.add(new Integer(str\_GroupFileValuesArrayList.get(int\_index)[5].replaceAll("\"", "")));

/\* Print the second column (user name). \*/

System.out.print("\* " + str\_GroupFileValuesArrayList.get(int\_index)[1] + "\n");

int\_index++;

}

readCSVBufferedReader.close();

/\* Nice trick on method toArray(), passing an array as argument. \*/

peerPeer = new Peer(str\_GroupFileAddressesArrayList.toArray(new String[0]), groupFilePortsIntegerArrayList.toArray(new Integer[0]));

}

else{

System.err.print("\nException 001: " + str\_usage);

System.exit(1);

}

/\* Main loop. \*/

do{

System.out.print("\nCHOOSE ACTION:\n" +

"\n" +

"0) quit\n" +

"1) send text\n" +

"2) send file\n" +

"3) receive file\n");

int\_action = Integer.parseInt(scannerScanner.next());

switch(int\_action){

case 0: {

break;

}

/\* Play the client role. \*/

case 1: {

peerPeer.playClient(messageStringBuilder);

break;

}

/\* Send file. \*/

case 2: {

peerPeer.playSeeder(new StringBuilder());

break;

}

/\* Receive file. \*/

case 3: {

peerPeer.playLeecher(new StringBuilder());

break;

}

default: {

peerPeer.playServer();

break;

}

}

if (scannerScanner.next().equals(null)){

/\* Play the server role. \*/

peerPeer.playServer();

}

} while (int\_error\_code != 0 && int\_action != 0);

scannerScanner.close();

System.exit(int\_error\_code);

} /\* Method main() end. \*/

} /\* Class SimpleNetworkCommunicator end. \*/

**7.BIBLIOGRAFIA**

**\* TANENBAUNM,** Andrew S. - Redes De Computadores: Quarta Edição.

**\* DEITEL,** Paul**, DEITEL,** Harvey - Java Como Programar: 8 Edição

\* https://en.wikipedia.org/wiki/Berkeley\_sockets

**8. FICHAS APS**

