Redes de Computadores 2015/2016

Lab 2 - Computer Networks

26 de Dezembro de 2015, Porto

Professor: *Ana Aguiar*

Turma: *MIEIC02*

Autores:

- *João Cepa* (ei12067@fe.up.pt)
- *Luís Carvalho* (up201303030@fe.up.pt)
- *Vitor Esteves* (up201303104@fe.up.pt)

Introdução

Neste segundo trabalho de Redes de Computadores, foi criada uma aplicação que permite o download de ficheiros existentes num servidor de FTP e ainda, uma configuraçao de rede com o objetivo de ligar vários computadores para diversas utilizações.

Com este relatório pretendemos dar a conhecer ao leitor quais as utilizações, bem como os aspectos fundamentais da aplicação de download e ainda, será mostrado o código implementado para resolver os vários desafios a que fomos propostos. Mostraremos ainda, a configuração da rede de computadores para que a mesma seja entendida da melhor forma.

Inicialmente, apresentaremos as instruções necessárias para correr a aplicação, de modo a que o leitor possa utilizar a aplicação de forma simples e eficaz.

Em seguida, passaremos à demonstração propriamente dita, na qual serão incluidas apresentações grãficas do programa em funcionamento, bem como da implementação por nós efetuada.

Aplicação de Download

A aplicação de download foi escrita utilizando a linguagem C, e tem como principal objetivo o download de um ficheiro alojado num servidor FTP, atráves do uso de sockets.

É possivel o utilizador definir o seu login/password, ou então, se assim o desejar, logar anonimamente no servidor FTP (opção por defeito, caso nao sejam introduzidos o conjunto login/password).

No caso de haver algum problema com o login, é retornada uma mensagem de erro, senão, no caso do login ter sido bem sucedido, o client entra no modo Passive FTP, de forma a obter um novo IP e uma porta da qual irá receber um conjunto de bytes que constituirá o ficheiro desejado.

Ao criar uma ligação, o client tenta aceder ao path que inicialmente introduziu, e em caso de sucesso, começa a fazer download do ficheiro.

Quando o download é finalizado, são fechadas as portas dos sockets e o cliente encerra. O ficheiro que foi recebido, encontra-se sempre guardado no diretorio onde client se encontra.

O objetivo do client é criar duas ligações: a primeira acede ao servidor FTP, isto fazendo login e enviando ‘pasv’ de modo a entrar em passive mode. Após resposta do servidor com um array de seis números, o client calcula a porta [256*X1 + X2 , em que a X1 e X2 são números obtidos do array] que irá ser usada, em simultanêo com o IP address necessário para criar uma nova ligação.

Por fim, os dados recebidos são guardados numa struct, com o objetivo de facilitar o acesso â informação adquirida.

Configuração e análise de redes

Experiência 1 – Configurar uma rede IP

Address Resolution Protocol ou **ARP** é um protocolo usado para encontrar um endereço da camada de ligação de dados (Ethernet, por exemplo) a partir do endereço da camada de rede (como um endereço IP).

O emissor difunde em *broadcast* um pacote ARP contendo o endereço IP de outro host e espera uma resposta com um endereço MAC respectivo. Cada máquina mantém uma tabela de resolução em cache para reduzir latência e carga na rede. O ARP permite que o endereço IP seja independente do endereço Ethernet, mas apenas funciona se todos os *hosts* o suportarem. Por outras palavras, é um protocolo usado para obter endereços MAC a partir de endereços de IP.

O protocolo ARP é dividido em duas partes: a primeira determina endereços físicos ao fazer o envio dos pacotes, e a segunda responde os pedidos de outros hosts.

Antes de enviar, o host consulta a cache ARP procurando o endereço físico. Se encontrar o endereço, anexa-o na frame e envia acrescentando os dados. Se o host não encontrar o endereço, é realizado um broadcast de pedido ARP.

A segunda parte do código do ARP manuseia os pacotes recebidos da rede. Quando chega um pacote, o programa extrai e examina o endereço físico e IP para verificar se já existe a entrada no cache e atualiza novamente sobrescrevendo os endereços. Depois, o recetor começa a processar o resto do pacote.

O recetor processa dois tipos de entrada de pacotes ARP:

- *Pedido ARP de um outro host* : o recetor envia o endereço físico ao emissor e armazena o endereço do emissor no cache. Se o endereço IP do pacote recebido não for igual ao do recetor, o pacote ARP é ignorado.

- *Resposta de um pedido ARP*: Após verificar a entrada no cache ARP, o recetor verifica primeiro a resposta com o pedido ARP enviado anteriormente. Enquanto o recetor espera pela resposta, as aplicações podem gerar outros pacotes que geralmente esperam na fila. Após verificar o endereço IP, o recetor atualiza os pacotes com o mesmo. O ARP retira os pacotes da fila depois de fornecer os endereços.

Se durante o broadcast o destinatário não puder aceitar um pedido, o host emissor deve armazenar o pacote enviado para retransmiti-lo. Pode acontecer, também, de o hardware de um host ter sido substituído. Se algum host tentar enviar dados para ele, utilizará um endereço não existente na rede, por isso é importante atualizar e remover os endereços na cache em períodos regulares.

Noutro sentido, Ping é um utilitário que usa o protocolo ICMP para testar a conectividade entre

equipamentos. É um comando disponível praticamente em todos os sistemas operacionais. Consiste no envio de pacotes para o equipamento de destino e na "escuta" das respostas.

Como tal, quando é executado o comando ping são gerados pacotes que são enviados e é esperada a sua resposta para ser contabilizado o tempo de resposta.

Nos pacotes gerados pelo comando ping os endereços MAC do emissor e do alvo estão localizados no header do Ethernet Frame e os endereços IP do emissor e alvo no header do pacote.

Para determinar se o frame de Ethernet é do tipo ARP,IP,ICMP, é analisado o Ethernet header e no caso do ICMP também o IP header.

No caso do IP o campo type da header do Ethernet frame terá o valor 0x0800, enquanto que, se o seu valor for 0x806, o pacote é do tipo ARP. Os pacotes IP com o campo protocol com valor 1 são pacotes ICMP.

Acrescente-se que, se o campo type do Ethernet Frame tiver valor igual ou inferior a 1500, então esse valor indica o tamanho do payload, senão indica o tipo de pacote. Quando utilizado como tipo de pacote, o comprimento da trama é determinado peça localização da interpacket gap e pela sequência frame check válida.

Um loopback é um canal de comunicação com apenas um ponto final. Qualquer mensagem transmitida por um canal é imediatamente recebida pelo mesmo canal.

Este canal pode ser bastante importante pois,um dispositivo de comunicação em série pode usar um loopback para testar o seu funcionamento.

Experiência 2 – Implementar duas redes LAN virtuais num switch

Nesta experiência, foi solicitada a criação de duas VLANs: vlany0; vlany1. A vlany0 alberga o tux1 e tux4, enquanto que a vlany1 alberga o tux2. A interface eth0 do tux1 (172.16.y0.1) e tux4 (172.16.y0.254) foram ligadas ao switch e depois cada uma das portas foi adicionada à vlany0. Por sua vez, o tux2 (172.16.y0.2) foi configurado igualmente, mas a porta do switch adicionada à vlany1.

Terminando esta segunda experiência, ficaram a existir dois domínios de broadcast, uma vez que existem duas VLANs sem comunicação entre ambas.

Através de uma análise aos logs obtidos, observámos que ao fazer broadcast no tux1, só o tux4 responde. Por outro lado, ao fazer broadcast no tux2, nenhum computador responde, uma vez que não chega nenhum pedido aos outros computadores.

Os comandos concretos para configuração da vlan0 figuram em anexo.

Experiência 3 – Configurar um router em Linux

Nesta experiência importa saber o que é uma forwarding table e que informações podemos obter desta. Assim, uma forwarding table é usada para encontrar a interface adequada para que a input interface encaminhe o pacote de maneira correta. Uma forwarding table contém os seguintes campos:

No tux1 existem duas rotas:

Destination	**Gateway**	**Genmask**	**Flags**	**Metric**	**Ref**	**Use**	**Iface**
172.16.50.0	*0.0.0.0*	*255.255.255.0*	*U*	*0*	*0*	*0*	*eth0*
172.16.51.0	*172.16.50.254*	*255.255.255.0*	*UG*	*0*	*0*	*0*	*eth0*

A primeira permite a aceitação do tráfego vindos de **172.16.50.0/24**, enquanto que a segunda redireciona os IPs do tipo **172.16.51.0/24** para o gateway router, isto é, porta **eth0** do **tux4** (**172.16.50.254**).

No **tux2** existem também duas rotas e a tabela é semelhante à do **tux1**. Contudo, aceita IPs provenientes da rede **172.16.51.0/24** e redireciona IPs do tipo **172.16.50.0/24** para o gateway router **172.16.51.253**, isto é, porta **eth1** do **tux4**.

Por último, o tux4, que funciona como gateway router, tem a seguinte forwarding table:

Destination	***Gateway***	***Genmask***	***Flags***	***Metric***	***Ref***	***Use***	***Iface***
172.16.50.0	*0.0.0.0*	*255.255.255.0*	*U*	*0*	*0*	*0*	*eth0*
172.16.51.0	*0.0.0.0*	*255.255.255.0*	*U*	*0*	*0*	*0*	*eth1*

Este tux redireciona da interface **eth0** para a **eth1**, aceitando assim o tráfego proveniente das duas redes e não enviando para nenhum **gateway**.

Neste ensaio foram observados os endereços MAC referentes à interface eth0 do tux4, enquanto que o segundo à interface eth1 do mesmo tux.

Importa acrescentar que, nesta terceira experiência, os pacotes **ICMP** observados são os **echo** request (gerados pelo computador que faz ping) e os **echo** reply (gerados pelo computador "pingado", como resposta), que são usados durante o ping para ver se um dado computador é alcançável e quanto tempo demoram os pacotes a viajar entre os dois.

Experiência 4 – Configurar um router comercial e implementar NAT

Nesta experiência, foi necessário proceder à atribuição de rotas estáticas no router para a comunicação entre VLANs, sendo, para isso, utilizado o comando ip route <endereço da VLAN> <máscara de sub-rede> <endereço da rota>.

A dado momento, foi pedido que retirássemos a rota de tux2 para a VLAN 0 através de tux4 e que fizéssemos ping de tux1 a partir de tux2. A nossa primeira reação foi pensar que os pacotes chegariam a tux1, mas não poderiam ser devolvidos ao primeiro computador. Esta assunção não foi confirmada, visto que tux1, embora não soubesse como alcançar tux2 diretamente, enviou os pacotes de resposta ao seu gateway predefinido – o router. Este, por sua vez, já tinha conhecimento do caminho a percorrer até chegar a tux2, isto é, através de tux4.

A funcionalidade de NAT foi também adicionada ao router. Esta permite fazer com que uma rede de computadores local possua um número de endereços IP usados unicamente para identificar máquinas localmente, enquanto que apenas um aparelho, neste caso o router, "dá a cara" para o exterior (a internet), reduzindo, assim, o número de IPs públicos que necessitam de ser alocados.

O router gera um número com 16 bits com base no endereço IP local e porta através do qual um pacote é enviado para a internet, que fica armazenado numa tabela de hash interna, e que mais tarde é utilizado para fazer a resposta chegar à máquina original.

Experiência 5 – DNS

Para configurar o serviço DNS num computador, acedemos ao seu ficheiro /etc/resolv.conf e adicionámos duas linhas: uma com o parâmetro search, destinado a permitir a introdução de hostnames curtos que tentarão ser resolvidos dentro do domínio especificado, e outra com o parâmetro nameserver: o endereço IP do servidor que funcionará como DNS. As linhas em concreto estão especificadas em anexo.

O serviço de DNS usa maioritariamente pacotes UDP para comunicar, sendo que pedidos que excedam um certo tamanho (tradicionalmente 512 bytes) são realizados através do protocolo TCP. Este serviço permite a um computador apenas ter conhecimento do nome de domínio do servidor que pretende alcançar, sendo que o endereço IP real é fornecido pelo servidor DNS. Ou seja, este faz a "conversão" entre nomes de domínio e endereços IP, sendo que estes poderão ser múltiplos e/ou inconstantes.

Experiência 6 – Ligações TCP

Durante o download de um ficheiro através da aplicação contruída, são abertas duas ligações. A principal é estabelecida quando a aplicação tenta ligar-se pela primeira vez ao servidor FTP, tratando assim, do login, anónimo ou não, do envio do comando que faz pedido do ficheiro e, por fim, do envio de um último comando de modo a que entre em passive mode. Por outro lado, a segunda ligação usa como IP e porta o que é retornado pelo servidor ao entrar em passive mode, e está exclusivamente encarregado do transporte de dados do ficheiro pretendido. O protocolo TCP é usualmente dividido em três fases, sendo que as mesmas são connection establishment, connection termination, data transfer:

- ***Connection establishment***: Ocorre antes de entrar em passive mode e começar a transferir o ficheiro pretendido pelo utilizador. Para estabelecer uma ligação, o protocolo TCP implementa um three-step handshake. O primeiro ocorre no client que envia ao servidor uma flag SYN (sincronização), que por sua vez, envia a resposta com uma SYN-ACK (reconhecimento). Ao receber uma SYN-ACK, o client irá responder com a flag ACK;

- ***Data transfer***: Para que o ficheiro seja transferido do servidor para o client sem existir qualquer espécie de falha, foram implementadas algumas funcionalidades-chave, como, por exemplo, a retransmissão de pacotes cuja transmissão falhou, a ordenação dos pacotes e o congestion control, que será explicado mais abaixo;

- ***Connection termination***: Concluído o download do ficheiro, ocorre um four-way handshake, sendo que se trata, à semelhança do connection establishment, da troca entre client-server de flags. Ao acabar a transferência, é enviada uma flag FIN, que é respondida com uma flag ACK seguida pela devolução da flag FIN. Para finalizar a ligação, é enviada uma segunda vez a flag ACK.

O uso de repetição de mecanismos de solicitação automática (**ARQ**) é uma técnica bem conhecida para lidar com erros de transmissão, que ocorrem quando os dados são transmitidos em canais com ruído.

Os esquemas ARQ têm um canal de controlo de qualidade para confirmar a receção de pacotes pelo recetor. Pacotes recebidos com erro são retransmitidos na data-link layer, o que faz com que o link apareça error-free nas camadas superiores.

A existência de um esquema **ARQ** na camada **MAC** é particularmente importante para o desempenho de protocolos de controlo de fluxo, tais como o **TCP**. Quando o **protocolo TCP** é usado em ligações com elevadas taxas de erros é frequente notar-se numa falta de eficiência. O **ARQ** aborda este problema, desencadeando retransmissões na camada MAC, que fazem a ligação aparecer livre para a camada **TCP** erro.

O sistema implementa **ARQ** em hardware, que lhe permite fazer várias retransmissões numa duração infinitamente pequena. Através desta facilidade, a transferência torna-se muito mais eficiente, pois, caso o pacote **TCP** não for recuperado num certo período de tempo, o mesmo é retransmitido. De forma a que se consigam alcançar bons desempenhos e evitar um congestion collapse, foram

implementados vários algoritmos de congestion control. O que estes algoritmos fazem é, na sua maioria, reduzir o rácio de envio de packets. Alguns exemplos de algoritmos são: **slow start**, onde o número de pacotes enviados vai aumentando até certo ponto, **congestion avoidance**, **fast retransmit** e **fast recovery**. Quando não aplicados corretamente, o sistema que estamos a desenhar corre sérios riscos de vir a falhar.

Para pequenos números de ligações **TCP**, o throughput não é afetado de forma considerável; contudo, caso seja em larga escala, a quantidade de flags a serem recebidas e enviadas pode causar congestionamento no servidor, afetando assim o throughput do mesmo. **SYN flood** é um exemplo do que pode acontecer.

Conclusão

No final do projecto, considerámos o desenvolvimento da aplicação um método muito bom de aprendizagem a todos os níveis (programação, interface com hardware, configuração de redes).

Apesar das dificuldades sentidas pelo grupo consideramos que o trabalho foi extremamente útil e que nos permitiu, não só de consolidar conhecimentos em programação em C, mas também o que fora leccionado nas aulas teóricas e produzido nos laboratórios.

Anexo I - Configuração dos computadores

TUX51

```
ifconfig eth0 up
ifconfig eth0 172.16.50.1/24
route add default gw 172.16.50.254
route -n
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

TUX52

```
ifconfig eth0 up
ifconfig eth0 172.16.51.1/24
route add -net 172.16.50.0/24 gw 172.16.51.253
route add default gw 172.16.51.254
route -n
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
echo 1 > /proc/sys/net/ipv4/conf/eth0/accept_redirects
echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects
```

TUX54

```
ifconfig eth0 up
ifconfig eth0 172.16.60.254/24
ifconfig eth1 up
ifconfig eth1 172.16.61.253/24
route add default gw 172.16.61.254
route -n
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

Anexo II - Configuração do Switch

- Configuração efectuada na consola do tux54 (cabo switch ligado)

```
configure terminal
vlan 50
exit

vlan 51
exit

interface fastethernet 0/1
switchport mode access
switchport access vlan 50
exit

interface fastethernet 0/4
switchport mode access
switchport access vlan 50
exit

interface fastethernet 0/2
switchport mode access
switchport access vlan 51
exit

interface fastethernet 0/3
switchport mode access
switchport access vlan 51
exit

interface fastethernet 0/5
switchport mode access
switchport access vlan 51
end
```

Anexo III - Configuração do router

- Configuração efectuada na consolda do tux54 (cabo router ligado)

```
configure terminal

 interface gigabitethernet 0/0
 ip address 172.16.51.254 255.255.255.0
 no shutdown
 exit

 ip route 172.16.50.0 255.255.255.0 172.16.51.253
 end

configure terminal

interface gigabitethernet 0/0
ip address 172.16.51.254 255.255.255.0
no shutdown
ip nat inside
exit

interface gigabitethernet 0/1
ip address 172.16.1.59 255.255.255.0
no shutdown
ip nat outside
exit

ip nat pool ovrld 172.16.1.59 172.16.1.59 prefix 24
ip nat inside source list 1 pool ovrld overload

 access-list 1 permit 172.16.50.0 0.0.0.7
 access-list 1 permit 172.16.51.0 0.0.0.7

 ip route 0.0.0.0 0.0.0.0 172.16.1.254
 ip route 172.16.50.0 255.255.255.0 172.16.51.253

 end
```

Anexo IV - Código-fonte da aplicação desenvolvida

- *File - Include.h*

includes.h

```
// DEFINES
#define MAX_BUFFER_SIZE 256
#define MAX_SERVER_RESPONSE_NUMBER_SIZE 4
#define TRUE 0
#define FALSE 1
#define END_OF_CYCLE 3


#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>

// declarations

void fillUserSettings(char* );
int createConnection(int);
void serverConversation(int );
void makeLogin(int );
void handlingFiles(int );
void get_file_name_from_path();
int parsingURLGivenByServer(int );
int responseFromServer(int, int);
void clearScreen();
```

- *File - download.c*

Struct - UserSettings

```
typedef struct UserSettings {

  // ID information
  char* username;
  char* password;

  // Server Information
  int port;
  char* domain;
  char* address;
  char* path;
  char* newIP;

  // File info.
  int pasv_port;
  unsigned long fileSize;
  char* filename;

} Settings;
```

Função main

```c
int main(int argc, char** argv) {

    int sockfd;

    // Allocating the memory
    userSet = malloc(sizeof(Settings));
    userSet->username = malloc(255 * sizeof(char)+1);
    userSet->password = malloc(255 * sizeof(char)+1);
    userSet->path = malloc(255 * sizeof(char)+1);
    userSet->domain = malloc(255 * sizeof(char)+1);
    userSet->address = malloc(255 * sizeof(char)+1);
    userSet->filename = malloc(255 * sizeof(char)+1);
    userSet->newIP = malloc(255 * sizeof(char)+1);

    if(argc != 2){
        printf("\n> ERROR: %s ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
        exit(1);
    }
    else {
        clearScreen();
        printf("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n");
        printf("%%%%%%%%%%%%%  WELCOME TO FTP-CLIENT %%%%%%%%%%%%%\n");
        printf("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%n");
      }

    fillUserSettings(argv[1]);
    sockfd = createConnection(21);
    serverConversation(sockfd);

    printf("> Now exiting from server.\n");
    char command[strlen("quit\n")];
    strcpy(command,"quit\n");
    send(sockfd, command, strlen(command), 0);

    close(sockfd);
    printf("> Everything went ok. Exit successfully.\n");
    return 0;
}
```

Função FillUserSettings

```
/**
 * Inserts data into a struct which name is userSet.
 * @param  Receives a url with this template 'ftp:///[<user>:<password>@]<host>/<url-path>'
 * @return [VOID]
 */
void fillUserSettings(char* url){

    /*
        TODO: VERIFICAR SIZE EM RFC1738
    */
    struct hostent *h;
    int i;
    char username[MAX_BUFFER_SIZE], password[MAX_BUFFER_SIZE], host[MAX_BUFFER_SIZE],path[MAX_BUFFER_SIZE];

    if ((i = sscanf(url, "ftp://%255[^:]:%255[^@]@%255[^/]/%s",username,password,host,path)) != 4 && (i = sscanf(url, "ftp://%255[^/]/%s",host,path)) != 2){
        printf("\n> ERROR: The url you entered is not acceptable.\n");
        exit(2);
    }

    if ((h = gethostbyname(host)) == NULL) {
            printf("> We're sorry but either the server is not responding or this url is invalid.\n");
            exit(3);
    }

    strcpy(userSet->domain,host);

    if(i == 4){
        strcpy(userSet->username,username);
        strcpy(userSet->password,password);
    }
    else {
        strcpy(userSet->username,"anonymous");
        strcpy(userSet->password,"12345");
    }

    userSet->port = 21;
    strcpy(userSet->path, path);
    strcpy(userSet->address,inet_ntoa(*((struct in_addr *)h->h_addr)));
    printf(">  INITIAL IP ADDRESS : %s\n",userSet->address);
    printf("> Username : %s\n",userSet->username);
    printf("> Password : %s\n\n",userSet->password);
}
```

Função CreateConnection

```c
/**
 * Creates a conection
 * @param  Description of method's or function's input parameter
 * @return [int] returns File descriptor.
 */
int createConnection(int port){
    int sockfd;
    struct  sockaddr_in server_addr;


    /*server address handling*/
    bzero((char*)&server_addr,sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    if(port == 21) server_addr.sin_addr.s_addr = inet_addr(userSet->address);
    else server_addr.sin_addr.s_addr = inet_addr(userSet->newIP);  /*32 bit Internet address network byte ordered*/
    server_addr.sin_port = htons(port);                 /*server TCP port must be network byte ordered */

    /*open an TCP socket*/
    if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0) {
            perror("socket()");
            exit(4);
    }
    /*connect to the server*/
    if(connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
            perror("connect()");
            exit(0);
    }

    return sockfd;
}
```

Função ServerConversation

```c
/**
 * See if server is
 * @param  Description of method's or function's input parameter
 * @return [int] returns File descriptor.
 */
void serverConversation(int sockfd){
    // waiting for answer from server and dealing with it
    int response = responseFromServer(sockfd, FALSE);

    // Login stuff
    if (response == 220) makeLogin(sockfd);
    //else if (response == 225) makeLogin(sockfd); // CHECK
    else {
        printf("> Something is wrong with the FTP server. The program will now exit!\n");
        exit(7);
    }
}
```

Função MakeLogin

```
/**
 * Try to login in server if needed. If server needs login, and user did not insert data, the function will try a default login.
 * @param  File descriptor.
 * @return [VOID]
 */
void makeLogin(int sockfd){

        // Send username and wait for response.
        char usernameCompleted[MAX_BUFFER_SIZE];
        strcpy(usernameCompleted,"user ");
        strcat(usernameCompleted,userSet->username);
        strcat(usernameCompleted,"\n");

        send(sockfd, usernameCompleted, strlen(usernameCompleted),0);
        int response = responseFromServer(sockfd, FALSE);

        if (response == 331)
        {
            char password_comp[MAX_BUFFER_SIZE];
            strcpy(password_comp,"pass ");
            strcat(password_comp,userSet->password);
            strcat(password_comp,"\n");
            send(sockfd, password_comp, strlen(password_comp),0);
            response = responseFromServer(sockfd, FALSE);

            if (response != 230){
                printf("> Login wasn't succefully ended! Sorry.\n");
                exit(9);
            }
            else {
                handlingFiles(sockfd);
            }
        }
        else {
            printf("> Sorry. We tried to login but that didn't help. Got your username and password straight and try again.\n");
            exit(8);
        }
}
```

Função handlingFiles

```
195 void handlingFiles(int sockfd){
196     send(sockfd,"pasv\n",5,0);
197     int answerGivenByServer = parsingURLGivenByServer(sockfd);
198
199     if ( answerGivenByServer == TRUE) {
200         int newSockFD = createConnection(userSet->pasv_port);
201         char command[MAX_BUFFER_SIZE];
202         strcpy(command,"retr ");
203         strcat(command,userSet->path);
204         strcat(command,"\n");
205
206         send(sockfd,command,strlen(command),0);
207
208         answerGivenByServer = responseFromServer(sockfd,TRUE);
209
210         if (answerGivenByServer == 150)
211         {
212             get_file_name_from_path();
213             FILE* wantedFile;
214             printf("> STARTING NOW THE TRANSFER OF FILE '%s'.\n\n",userSet->filename);
215             wantedFile = fopen(userSet->filename, "wb");
216             char buffer[128];
217             int res = 1;
218
219             /* Reading File */
220             while( (res = read(newSockFD,buffer,1)) > 0)
221                 fwrite(buffer,1,1,wantedFile);
222
223
224             fclose(wantedFile);
225             close(newSockFD);
226
227         } else {
228             printf("We're sorry. There was an error trying to access file. Try again.\n");
229             exit(10);
230         }
231     }
232 }
```

Função ParsingURLGivenByServer

```c
/**
 * Return TRUE if the parsing is done correctly or exit the program if not..
 * @param  File descriptor.
 * @return [INT] if correct.
 */
int parsingURLGivenByServer(int sockfd){
    int receivedFromServer = -1;
    char buffer[MAX_BUFFER_SIZE];

    while(receivedFromServer == -1)
        receivedFromServer = recv(sockfd,buffer,MAX_BUFFER_SIZE,0);

    buffer[receivedFromServer] = '\0';
    printf("> Resposta: %s\n", buffer);

    char temp[255];
    char AUX[2]; AUX[0] = '(';AUX[1] = ')';

    int i = 0, j = 0, CONTROLO = FALSE;
    for (; i < strlen(buffer); i++)
    {
        if ( buffer[i] == AUX[0]){
            temp[j++] = buffer[i];
            CONTROLO = TRUE;
        }
        else if (CONTROLO == TRUE){
            int k = i;
            for (;k < strlen(buffer); k++)
            {
                if (buffer[k] == AUX[1]) {
                    temp[j++] = buffer[k];
                    CONTROLO = END_OF_CYCLE;
                    break;
                }
                temp[j++] = buffer[k];
            }
            temp[k] = '\0';
        }
        else if (CONTROLO == END_OF_CYCLE) break;
    }
    // NO NEED TO CHANGE THE CODE BELOW.

    int parsing;
    // ip_1[3], ip_2[3], ip_3[3], ip_4[3]
    char ip_1[MAX_SERVER_RESPONSE_NUMBER_SIZE], ip_2[MAX_SERVER_RESPONSE_NUMBER_SIZE], ip_3[MAX_SERVER_RESPONSE_NUMBER_SIZE], ip_4[MAX_SERVER_RESPONSE_NUMBER_SIZE], port_1[MAX_SERVER_RESPONSE_NUMBER_SIZE], port_2[MAX_SERV
    if ((parsing = sscanf(temp, "(%4[^,],%4[^,],%4[^,],%4[^,],%4[^,],%4[^)])", ip_1, ip_2, ip_3, ip_4, port_1, port_2)) != 6){
        printf("> Server is not responding well. Try again again.\n");
        exit(9);
    }
    else {
        char newIP[MAX_BUFFER_SIZE];
        sprintf (newIP, "%s.%s.%s.%s", ip_1, ip_2, ip_3, ip_4);
        strcpy(userSet->newIP,newIP);
        userSet->pasv_port = 256*atoi(port_1) + atoi(port_2);
        printf(">  NEW IP: '%s:%d'\n",userSet->newIP,userSet->pasv_port);
        return TRUE;
    }
}
```

```
/**
 * Fill the variable userSet->filename
 * @param  [NONE]
 * @return [VOID]
 */
void get_file_name_from_path(){

    char temp[1]; temp[0] = '/';
    // get last '/' index
    int i = strlen(userSet->path);
    for (; i > 0; i--)
    {
        if( userSet->path[i] == temp[0]){
            i += 1;
            break;
        }
    }
    // get filename and extension.
    char filename[strlen(userSet->path) - i];
    int k=0;
    for(; k < strlen(userSet->path); k++){
        filename[k] = userSet->path[i++];
    }

    strcpy(userSet->filename,filename);
}
```

```c
/**
 * Fill the variable userSet->fileSize
 * @param  Buffer called message. This buffer contains a message received from server.
 * @return [VOID]
 */
void getFileSizeFromServerResponse(char* message){
    char temp[255];
    char AUX[2];
    AUX[0] = '(';AUX[1] = ' ';

    int i = 0, j = 0, CONTROLO = FALSE;
    for (; i < strlen(message); i++)
    {
        if ( message[i] == AUX[0]){
            CONTROLO = TRUE;
        }
        else if (CONTROLO == TRUE){
            int k = i;
            for (;k < strlen(message); k++)
            {
                if (message[k] == AUX[1]) {
                    temp[j++] = message[k];
                    CONTROLO = END_OF_CYCLE;
                    break;
                }
                temp[j++] = message[k];
            }
            temp[k] = '\0';
        }
        else if (CONTROLO == END_OF_CYCLE) break;
    }

    userSet->fileSize = atoi(temp);
}
```

```c
/**
 * Return the response of a socket.
 * @param  File descriptor, 'BOOL' with either want to search for size in the message or not.
 * @return [Char*] containing the response from server.
 */
int responseFromServer(int sockfd, int getSize) {
    int receivedFromServer = -1;
    char buffer[MAX_BUFFER_SIZE];

    while(receivedFromServer == -1)
        receivedFromServer = recv(sockfd,buffer,MAX_BUFFER_SIZE,0);

    buffer[receivedFromServer] = '\0';
    printf("> Resposta: %s", buffer);

    if (getSize == TRUE)
      getFileSizeFromServerResponse(buffer);

    return atoi(buffer);
}
```

```
376  /**
377   * Clears the screen.
378   * @param  [NONE]
379   * @return [VOID]
380   */
381  void clearScreen(){
382    int i = 0;
383    for(; i < 50; i++)
384      printf("\n");
385  }
386
```