

**Rede de Computadores**

(2º Trabalho Laboratorial)

Redes de Computadores **·** L.EIC025

2022/2023 **·** 1º Semestre

Turma: 3LEIC04

Carlos Sousa

up202005954

up202004946

up202005478

Daniela Tomás

Fábio Rocha

# Sumário

No âmbito da unidade curricular de Redes de Computadores, foi-nos proposta a realização de um projeto composto por duas partes. A primeira parte consiste no desenvolvimento de uma aplicação de *download* de um ficheiro utilizando o protocolo FTP (*File Tranfer Protocol*). Na segunda parte, foram efetuadas diversas experiências para a configuração de uma rede.

O objetivo do projeto foi concluído com sucesso, pois conseguimos criar uma aplicação capaz de transferir ficheiros e também configurar a rede corretamente.

# Introdução

Com este relatório, pretendemos explicar de uma forma detalhada e organizada o funcionamento da aplicação de *download* de um ficheiro ao ser fornecido um URL da forma ftp://[<user>:<password>@]<host>/<url-path> e também o funcionamento da configuração da rede de computadores capaz de se ligar à internet e testar a aplicação de *download*. Assim, estruturámos o relatório da seguinte forma:

* **Aplicação *download***

Descrição da arquitetura e do caso de sucesso.

* **Configuração e análise da rede**

Descrição da arquitetura da rede, objetivos, principais comandos de configuração e análise dos principais *logs* capturados para as seis experiências realizadas.

* Experiência 1 – Configurar uma rede IP
* Experiência 2 – Implementação de duas *bridges* no *switch*
* Experiência 3 – Configuração de um *router* em Linux
* Experiência 4 – Configurar um *router* comercial e implementar NAT
* Experiência 5 – DNS
* Experiência 6 – Conexões TCP
* **Conclusões**  
  Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

# Aplicação *download*

# 3.1- Arquitetura

A primeira parte do trabalho consiste em desenvolver uma aplicação de *download* de ficheiros de acordo com o protocolo FTP. Para executarmos a aplicação, inicialmente é necessário compilar o programa e passar-lhe como argumento o seguinte link, que está de acordo com a sintaxe URL descrita no RFC173:

* ftp://[<user>:<password>@]<host>/<url-path>

A aplicação inicialmente faz *parse* do URL, ou seja, cada componente fica armazenada num array. Caso o *user* e a *password* não sejam especificados, estes são assumidos como “anonymous” e “password”. Depois de interpretados os dados do URL necessários para a execução da transferência é necessário comunicar com o servidor através de um *socket* TCP. De seguida, o programa irá tentar aceder ao servidor com as credenciais indicadas e, depois do login, o programa solicita que o servidor responda de modo passivo (PASV), pois é o servidor que decide para que porta o ficheiro será enviado. Abre-se outra conexão num novo *socket* para a transferência do ficheiro. Após ser transferido o ficheiro, os dois *sockets* são fechados.

# 3.2- Caso de sucesso

A aplicação foi testada com dois ficheiros distintos, pic1.jpg e timestamp.txt, com e sem autenticação no servidor (figura 3.2-A e 3.2-B).

Text

Description automatically generated

***Figura 3.2-A:*** *Transferência do ficheiro pic1.jpg (340603 bytes) com autenticação*

Text

Description automatically generated

***Figura 3.2-B:*** *Transferência do ficheiro timestamp.txt (11 bytes) sem autenticação*

# Configuração e análise de redes

# Diagram Description automatically generated4.1- Experiência 1 - Configurar uma rede IP

***Figura 4.1:*** *Experiência 1*

O objetivo desta experiência é configurar duas máquinas, tux33 e tux34, sendo necessário a atribuição de um endereço IP para cada uma delas.

**4.1.1- Análise de Logs**

O protocolo ARP (*Address Resolution Protocol*) é um procedimento para mapear endereços de IP dinâmicos para um endereço de máquina físico numa rede local. Assim, associa o endereço IP com o endereço MAC, traduzindo endereços 32-bit para 48-bit e vice-versa, trabalhando entre a *data link layer* e a *network layer* do OSI model. Depois de apagar a tabela ARP dos computadores, o protocolo ARP foi executado.

O endereço MAC é o endereço da máquina física e localiza-se na *data link layer*. Já endereço IP é um endereço exclusivo que identifica um dispositivo na Internet ou numa rede local, localizando-se na *network layer*. Uma máquina pode possuir vários endereços IP, mas apenas um endereço MAC.

Como o ARP é um protocolo de requisição e resposta, a requisição é feita via *broadcast*, solicitando um endereço MAC de uma máquina através do endereço IP. Por sua vez, a resposta é fornecida de forma *Unicast* pela máquina com o endereço lógico requisitado, contendo o endereço físico da mesma.

O comando *ping* gera ICMPs *requests* para o *host* desejado e espera pela resposta ICMP, assim são reportados erros, perda de pacotes e um sumário estatístico de resultados. Nesta experiência o comando *ping* serve para descobrir se existe conectividade entre o tux33, que tem como endereço de IP 172.16.30.1 e endereço MAC 00:21:5a:61:24:92, e o tux34, com endereço IP de 172.16.30.254 e endereço MAC 00:21:5a:5a:7d:74.

Para conseguirmos determinar o tipo de trama recebida, deve-se analisar o cabeçalho da trama. Se o valor for 0x0806, então é do tipo ARP. No caso de ter valor 0x0800, a trama é do tipo IP. Se depois se verificar que o cabeçalho do IP é 1, então é do tipo ICMP. Se for uma trama IP, essa informação encontra-se no seu cabeçalho. Para identificarmos o tamanho da trama recebida, usamos o Wireshark.

Ainda analisando o comando *ping* através do Wireshark, podemos verificar o envio de tramas *loopback* pelo emissor. A *loopback interface* é uma interface virtual que está sempre ativa e acessível desde que pelo menos uma das interfaces IP no *switch* esteja operacional. Assim, uma *loopback interface* é útil para tarefas de *debugging*, pois seu endereço IP pode sempre receber *ping* se qualquer outra interface do *switch* estiver ativa.

**4.1.2- Principais comandos**

O endereço IP de cada computador foi configurado usando o comando *ifconfig* e, com o comando *route* foram estabelecidas as rotas entre eles. Com o comando *ping*, foi possível verificar se a conexão entre os dois computadores foi feita corretamente.

Diagram

Description automatically generated**4.2-** **Experiência 2 – Implementação de duas *bridges* no *switch***

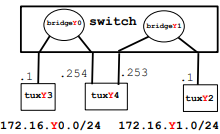
***Figura 4.2:*** *Experiência 2*

O objetivo desta experiência é criar duas *bridges* no *switch*, e perceber a conectividade entre os computadores e como estas influenciam a troca de informação entre as máquinas.

**4.2.1- Análise de Logs**

Para configurarmos as *bridges*,inicialmente criamos a bridge30 e bridge31 através do *gtkterm*, e associamos à primeira o tux33 e o tux34 e à segunda o tux32, tendo assim a arquitetura pretendida. Depois, podemos verificar a conectividade entre os computadores, fazendo um *ping* do tux33 até ao tux34, que foi executado com sucesso, pois estes encontram-se na mesma sub-rede. Contudo, fazendo um *ping* do tux33 até ao tux32, não obtivemos resposta, pois não existe nenhuma rota entre as duas *bridges*, tornando impossível o tux33 chegar à interface de rede do tux32. O mesmo acontece se fizermos um *ping* entre o tux34 e o tux33. Portanto, podemos concluir que existem dois domínios de *broadcast* correspondentes às sub-redes bridge30 e bridge31.

**4.2.2- Principais comandos**

**4.3-** **Experiência 3 – Configuração de um router em Linux**

***Figura 4.3:*** *Experiência 3*

O objetivo desta experiência é configurar o tux34 de modo a este funcionar como um *router*, para possibilitar a comunicação entre o tux33 e o tux32 através das *bridges* configuradas na experiência anterior. Para permitir a comunicação é necessário configurar os endereços IP’s das portas e *ethernet* dos computadores e as rotas que serão usadas.

**4.3.1- Análise de Logs**

Para conseguirmos ter ligação entre o tux33 e o tux32, é necessário adicionar uma rota ao tux33 para aceder aos endereços 172.16.31.0/24 a partir do IP 172.16.30.254. Assim, quando o tux33 quer enviar um *ping* para a bridge31 este vai utilizar o router, que neste caso é o tux34(172.16.30.254), como *gateway* e uma rota ao tux32 para aceder aos endereços 172.16.30.0/24 a partir do IP 172.16.31.253. Estas rotas podem ser vistas na *forwarding table* utilizando o comando *route -n*. A *forwarding table* é uma *table* onde cada entrada possui informação do tipo *Destino-Gateway-Interface*, em que o destino é o IP do computador de destino. O *gateway* é o IP do computador para o qual se vai enviar a mensagem e a interface é a placa de rede usada para enviar a mensagem. Assim, com estas rotas definidas, é possível fazer *ping*, a partir do tux33, a todas as interfaces dos outros computadores.

Com isto, ao fazer *ping* do tux33 para o tux32 são observados os seguintes pacotes ARP. Primeiro é observado um pacote que pede que o endereço MAC do IP 172.16.30.254 seja enviado para 172.16.30.1, o endereço MAC observado é o 00:21:5a:5a:7d:74. Isto acontece porque o tux33 está a encontrar a interface do tux34 que encaminha o pacote enviado pelo *ping* para o tux32, o endereço MAC de resposta corresponde à interface eth0 de tux34. Em seguida, na mesma interface é observado o pacote ARP que pede que o endereço MAC de 172.16.30.1 seja enviado para 172.16.30.254. Isto acontece porque o tux34 necessita do endereço MAC da interface do tux33 para encaminhar a resposta do tux32 ao *ping*. O endereço MAC observado é o 00:21:5a:61:24:92. Já na eth1 do tux34, o tux34 pede qual o endereço MAC do tux32 para lhe poder encaminhar o pacote de *ping* e este dá o endereço MAC de eth0 do tux32. Depois, o tux32 pede o endereço MAC de eth1 do tux34 para lhe poder enviar a resposta a *ping* para que esta seja encaminhada a tux33. O tux34 responde com o endereço MAC de eth1 do tux34. Esta troca de mensagens ARP ocorre sempre que uma mensagem é enviada de uma máquina para outra sendo que os endereços MAC não são conhecidos.

Observando os pacotes ICMP, podemos verificar pacotes do tipo *request* e *reply,* pois todas as rotas estão configuradas. Se estas não se encontrassem configuradas seriam ver pacotes ICMP do tipo *Host Unreachable*.

Os endereços de IP de destino associados com os pacotes ICMP correspondem sempre aos IP’s da nossa máquina, enquanto os endereços de IP associados com os pacotes ICMP de origem correspondem aos IP’s pelos quais os pacotes viajam para alcançarem o destino desejado.

O endereço MAC de destino associado com os pacotes ICMP corresponde à interface virtual, enquanto o endereço MAC de origem associado com os pacotes ICMP é o endereço MAC da interface virtual do computador *host*.

**4.3.2- Principais comandos**

**Diagram

Description automatically generated4.4-** **Experiência 4 – Configurar um *router* comercial e implementar NAT**

***Figura 4.4:*** *Experiência 4*

O objetivo desta experiência é estabelecer uma ligação com a rede dos laboratórios e implementar rotas em um *router* comercial, adicionando-lhe funcionalidade NAT para garantir a conexão entre as máquinas e a internet.

**4.4.1- Análise de Logs**

Inicialmente necessitamos de configurar o *router*, para isso iniciamos a sessão no *router* a partir do *GTKTerm*, depois ligamos o cabo S0 de qualquer tux à entrada de configuração do *router*. Com isto temos acesso ao *router* através do *GTKTerm*, e podemos configurar o IP do *router* e também as rotas, utilizando o comando *ip route*.

Existem dois cenários possíveis para o que o computador possa ter acesso a internet, sendo o primeiro em que existe uma rota configurada entre as duas máquinas e assim a informação segue essa rota, e no segundo caso em que essa rota não existe e os pacotes são enviados pela rota *default* e a máquina para qual a rota *default* está definida irá enviar os pacotes para o destino. Como nesta experiência a rota do tux32 para o tux33 foi apagada, os pacotes foram redirecionados para o *router* e de seguida foram passados para o tux34.

O NAT é um mecanismo implementado em *routers* que substitui os endereços IP locais nos pacotes por um endereço IP público de forma a se conseguir estabelecer uma ligação para fora da rede. Sendo assim, o *router* que implementa o NAT torna-se responsável por encaminhar todos os pacotes para o endereço correto, dentro ou fora da rede local. Por exemplo, nesta experiência supondo que o tux33 quer enviar um pacote para um endereço em uma rede pública, o pacote primeiramente é enviado para o *router* que irá modificar o endereço *source* do pacote, para o seu endereço exterior assim assegurando a privacidade e a segurança do seu remetente. O pacote é enviado, e obtêm como resposta um pacote. O *router*, ao receber esse pacote, reenvia-o para tux33, alterando o destinatário do pacote para o seu endereço, possibilitando a comunicação entre a rede privada local e a rede pública.

**4.4.2- Principais comandos**

Configurar *ip address* do router e as suas rotas:

*/ip address add address=172.16.1.Y9/24 interface=ether1*

*/ip address add address=172.16.Y1.254/24 interface=ether2*

*/ip route add dst-address=172.16.Y0.0/24 gateway=172.16.Y1.253*

*/ip route add address=0.0.0.0/0 gateway=172.16.2.254*

No tuxy2 definir RC como *default* *router*:

*route add default gw 172.16.y1.254*

No tuxy3 definir tuxy4 como *default* *router*:

*route add default gw 172.16.y0.254*

No tuxy4 definir RC como *default* *router*:

*route add default gw 172.16.y1.254*

**Diagram, schematic

Description automatically generated4.5-** **Experiência 5 – DNS**

***Figura 4.5:*** *Experiência 5*

O objetivo desta experiência é configurar o DNS responsável por traduzir endereços URL em endereços IP.

**4.5.1- Análise de Logs**

O serviço DNS é configurado no ficheiro *resolv.conf,* que se localiza no diretório */etc/* do determinado tux, bastando adicionar no ficheiro:

* search netlab.fe.up.pt
* nameserver 172.16.1.1

O *host* envia para o server um pacote com o *hostname*, esperando que seja retornado o seu endereço IP. O servidor responde com um pacote que contém o endereço IP do *hostname* em causa.

**4.5.2- Principais comandos**

Inicialmente é necessário em cada tux, editar o ficheiro *resolv.conf*, de modo a configurarmos o DNS.

Para isso executamos o seguinte comando no terminal:

* echo $’search netlab.fe.up.pt\ nnameserver 172.16.1.1’ > /etc/resolv.conf

Diagram

Description automatically generated**4.6-** **Experiência 6 – Conexões TCP**

***Figura 4.6:*** *Experiência 6*

O objetivo desta experiência é observar o comportamento e funcionamento do protocolo TCP, usando a aplicação que desenvolvemos.

**4.6.1- Análise de Logs**

A aplicação que elaboramos foi compilada e executada. Durante a execução desta são abertas duas conexões TCP, uma quando se entra em contacto com o servidor, para o controlo da informação tratando do envio e receção de comandos, e outra para fazer a transferência do ficheiro. Numa conexão TCP encontra-se dividida em três fases: inicialmente estabelece-se conexão, de seguida ocorre a troca de dados e finalmente a conexão é encerrada.

O ARQ TCP é um método de controlo de erros para transmissão de dados que usa confirmações (mensagens enviadas pelo recetor indicando que recebeu corretamente o pacote) e *timeouts* (períodos de tempo especificados antes que uma confirmação seja recebida) para obter uma transmissão de dados confiável através de um canal de comunicação não confiável. Se o remetente não receber uma confirmação antes do tempo limite, ele retransmite o pacote até receber uma confirmação ou exceder um número predefinido de retransmissões.

Relativamente ao mecanismo de controlo de congestão do TCP, este tem como base os ACKs recebidos na transmissão dos pacotes. É utilizada uma nova variável por conexão, *CongestionWindow*, de modo a regular o tamanho da janela deslizante de transmissão de pacotes tendo em conta a congestão da conexão. O valor desta é incrementando se a congestão da rede diminuir e decrementado se a congestão da rede aumentar. Quando se deteta que um pacote é perdido, o valor da *CongestionWindow* passa para metade.

Quando iniciamos uma segunda conexão TCP a taxa de transmissão de pacotes da conexão TCP diminui. Isto deve-se ao facto de quando aumentamos o número de conexões TCP a largura da banda disponível para cada uma das conexões diminui. Com isto, o *download* do ficheiro do servidor TCP leva mais tempo.

**4.6.2- Principais comandos**

Nesta experiência apenas compilamos e executamos a nossa aplicação download.

# Conclusões

# Anexo I – Aplicação *download*

# download.c

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <netdb.h>
4. #include <netinet/in.h>
5. #include<arpa/inet.h>
6. #include <sys/socket.h>
7. #include <arpa/inet.h>
8. #include <unistd.h>
9. #include <string.h>
10. #define MAX\_SIZE 512
11. #define FTP\_PORT 21
12. int main(int argc, char \*\*argv) {
14. struct hostent\* h;
15. char buf[MAX\_SIZE], user[50], password[50], host[100], urlPath[MAX\_SIZE];
16. if (argc != 2) {
17. fprintf(stderr, "Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
18. exit(-1);
19. }
20. sscanf(argv[1],"ftp://%511s",buf);
21. size\_t i = strcspn(buf,":"), size = strlen(buf);
22. if(i < size) {
23. size\_t j = strcspn(buf,"@");
24. if(j == size) {
25. fprintf(stderr,"Usage: %s ftp://[<user>:<password>@]<host>/<url-path>\n", argv[0]);
26. exit(-1);
27. }
28. strncpy(user,buf,i);
29. user[i] = '\0';
30. strncpy(password,buf+i+1,j-i-1);
31. password[j-i-1] = '\0';
32. i = strcspn(buf+j+1,"/");
33. strncpy(host,buf+j+1,i);
34. host[i] = '\0';
35. strcpy(urlPath,buf+j+i+1);
36. }
37. else {
38. i = strcspn(buf,"/");
39. strncpy(host,buf,i);
40. host[i] = '\0';
41. strcpy(urlPath,buf+i+1);
42. strcpy(user,"anonymous");
43. strcpy(password,"password");
44. }
45. printf("\nuser: %s\npassword: %s\nhost: %s\nurl-path: %s\n\n",user,password,host,urlPath);
46. if ((h = gethostbyname(host)) == NULL) {
47. herror("gethostbyname()");
48. printf("%s\n",host);
49. exit(-1);
50. }
51. printf("Host name  : %s\n", h->h\_name);
52. printf("IP Address : %s\n", inet\_ntoa(\*((struct in\_addr\*) h->h\_addr\_list[0])));
53. //server address handling
54. struct sockaddr\_in server\_addr;
55. bzero((char\*) &server\_addr,sizeof(server\_addr));
56. server\_addr.sin\_family = AF\_INET;
57. server\_addr.sin\_addr.s\_addr = inet\_addr(inet\_ntoa(\*((struct in\_addr\*) h->h\_addr\_list[0]))); //32 bit Internet address network byte ordered
58. server\_addr.sin\_port = htons(FTP\_PORT); //server TCP port must be network byte ordered
59. //open a TCP socket
60. int sockfd;
61. if ((sockfd = socket(AF\_INET,SOCK\_STREAM,0)) < 0) {
62. perror("socketfd\n");
63. exit(-1);
64. }
65. //connect to the server
66. if (connect(sockfd,(struct sockaddr\*) &server\_addr,sizeof(server\_addr)) < 0) {
67. printf("%s\n",h->h\_addr\_list[0]);
68. perror("connect\n");
69. exit(-1);
70. }
71. //send a string to the server
72. usleep(100000);
73. int bytesRead = read(sockfd,buf,MAX\_SIZE);
74. buf[bytesRead] = '\0';
75. printf("%s\n",buf);
76. if(strncmp(buf,"220",3) != 0) {
77. perror("220\n");
78. exit(-1);
79. }
80. char buf2[519];
81. sprintf(buf2,"user %s\r\n",user);
82. size\_t bytes = write(sockfd, buf2, strlen(buf2));
83. bytesRead = read(sockfd,buf,MAX\_SIZE);
84. buf[bytesRead] = '\0';
85. printf("%s\n",buf);
86. if(strncmp(buf,"331 Please specify the password.",32) != 0) {
87. perror("331 Please specify the password.\n");
88. exit(-1);
89. }
91. sprintf(buf2,"pass %s\r\n",password);
92. write(sockfd,buf2,strlen(buf2));
93. bytesRead = read(sockfd,buf,MAX\_SIZE);
94. buf[bytesRead] = '\0';
95. printf("%s",buf);
96. if(strncmp(buf,"230",3) != 0) {
97. perror("230\n");
98. exit(-1);
99. }
100. strcpy(buf2,"pasv\r\n");
101. write(sockfd,buf2,strlen(buf2));
102. bytesRead = read(sockfd,buf,MAX\_SIZE);
103. buf[bytesRead] = '\0';
104. printf("%s",buf);
105. if(strncmp(buf,"227",3) != 0) {
106. perror("227\n");
107. exit(-1);
108. }
109. int h1 = 0, h2 = 0, h3 = 0, h4 = 0, p1 = 0, p2 = 0, pasvport;
110. sscanf(buf,"227 Entering Passive Mode (%i,%i,%i,%i,%i,%i).",&h1,&h2,&h3,&h4,&p1,&p2);
111. pasvport = p1\*256+p2;
112. //server address handling
113. struct sockaddr\_in server\_addrC;
114. sprintf(buf,"%i.%i.%i.%i",h1,h2,h3,h4);
115. bzero((char\*) &server\_addrC,sizeof(server\_addrC));
116. server\_addrC.sin\_family = AF\_INET;
117. server\_addrC.sin\_addr.s\_addr = inet\_addr(buf); //32 bit Internet address network byte ordered
118. server\_addrC.sin\_port = htons(pasvport); //server TCP port must be network byte ordered
119. int sockfdC;
120. if ((sockfdC = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {
121. perror("socketfdC\n");
122. exit(-1);
123. }
124. //connect to the server
125. if (connect(sockfdC,(struct sockaddr\*) &server\_addrC,sizeof(server\_addrC)) < 0) {
126. perror("connect\n");
127. exit(-1);
128. }
129. sprintf(buf2,"retr %s\r\n",urlPath);
130. bytes = write(sockfd,buf2,strlen(buf2));
131. if (bytes > 0) {
132. printf("Written %ld bytes\n", bytes);
133. }
134. else {
135. perror("write\n");
136. exit(-1);
137. }
138. bytesRead = read(sockfd,buf,MAX\_SIZE);
139. buf[bytesRead] ='\0';
140. printf("%s\n", buf);
141. if(strncmp(buf,"150",3) != 0) {
142. perror("150\n");
143. exit(-1);
144. }
145. char\* filename = strrchr(urlPath, '/');
146. printf("Receiving file %s...\n",filename+1);
147. FILE\* fd = fopen(filename+1,"w");
148. if(!fd) {
149. perror("fopen\n");
150. exit(-1);
151. }
152. while((bytesRead = read(sockfdC,buf,MAX\_SIZE)) > 0){
153. fwrite(buf,1,bytesRead,fd);
154. //printf("%s",buf);
155. }
156. fclose(fd);
157. do {
158. memset(buf, 0, MAX\_SIZE);
159. read(sockfd,buf,MAX\_SIZE);
160. } while (buf[0] < '1' || buf[0] > '5' || buf[3] != ' ');
161. printf("%s", buf);
162. buf[3] = '\0';
163. if(strncmp(buf,"226",3) != 0) {
164. perror("226\n");
165. exit(-1);
166. }
167. sprintf(buf2,"QUIT\r\n");
168. write(sockfd,buf2,strlen(buf2));
169. bytesRead = read(sockfd,buf,MAX\_SIZE);
170. buf[bytesRead] = '\0';
171. printf("%s",buf);
172. if(strncmp(buf,"221",3) != 0) {
173. perror("221\n");
174. exit(-1);
175. }
176. if (close(sockfd) < 0) {
177. perror("close sockfd\n");
178. exit(-1);
179. }
180. if (close(sockfdC) < 0) {
181. perror("close sockfdC\n");
182. exit(-1);
183. }
184. return 0;
185. }

# Makefile

1. CC = gcc
2. CFLAGS = -Wall
3. .PHONY: all
4. all:
5. $(CC) $(CFLAGS)  download.c -o download
6. .PHONY: clean
7. clean:
8. rm -f download
9. #///////////////////////////////////////////////////////////
10. .PHONY: pic1
11. pic1:
12. ./download ftp://rcom:rcom@netlab1.fe.up.pt/files/pic1.jpg
13. .PHONY: pipe
14. pipe:
15. ./download ftp://rcom:rcom@netlab1.fe.up.pt/pipe.txt

# Anexo II – Configuração de comandos e *logs* capturados

# 7.1- Experiência 1 - Configurar uma rede IP

Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated***Figura 7.1-A:*** *ifconfig no tux33*

Graphical user interface, application, table

Description automatically generated***Figura 7.1-B:*** *ifconfig no tux34*

***Figura 7.1-C:*** *Log experiência 1*

# 7.2- Experiência 2 – Implementação de duas bridges no *switch*

Graphical user interface

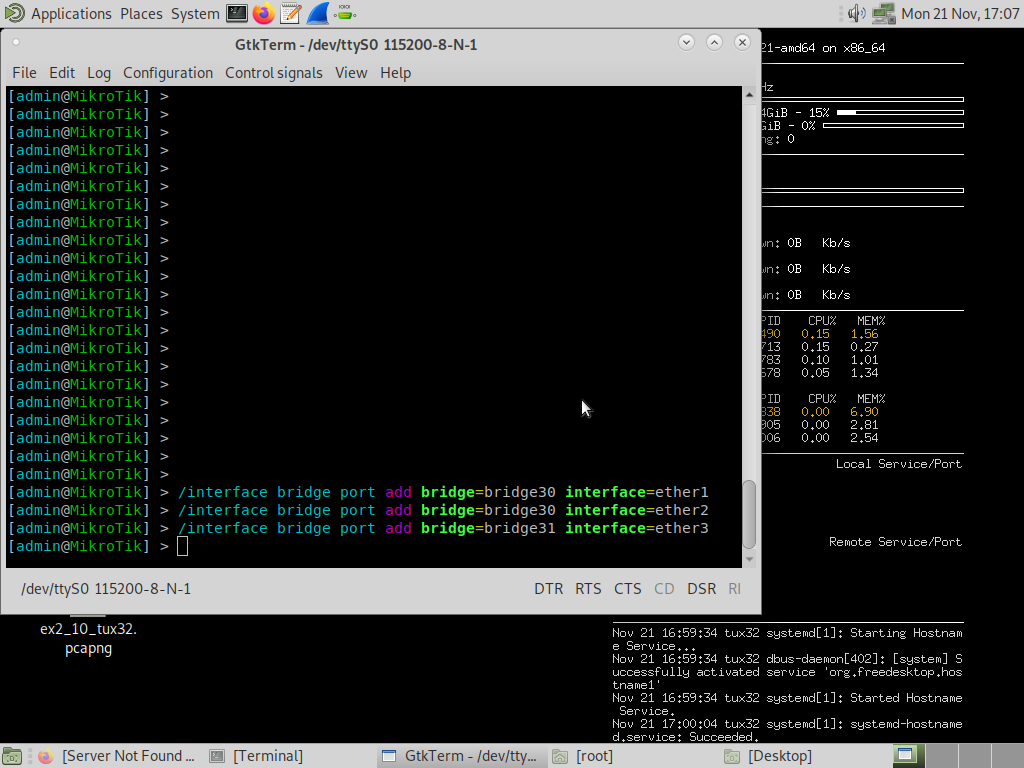
Description automatically generated

Graphical user interface, text

Description automatically generated***Figura 7.2-A:*** *GtkTerm – Configuração do baudrate*

*A screenshot of a computer

Description automatically generated****Figura 7.2-B:*** *Criar bridges*

*****Figura 7.2-B:*** *Remover portas das bridges*

***Figura 7.2-C:*** *Adicionar portas às bridges*

Graphical user interface, application

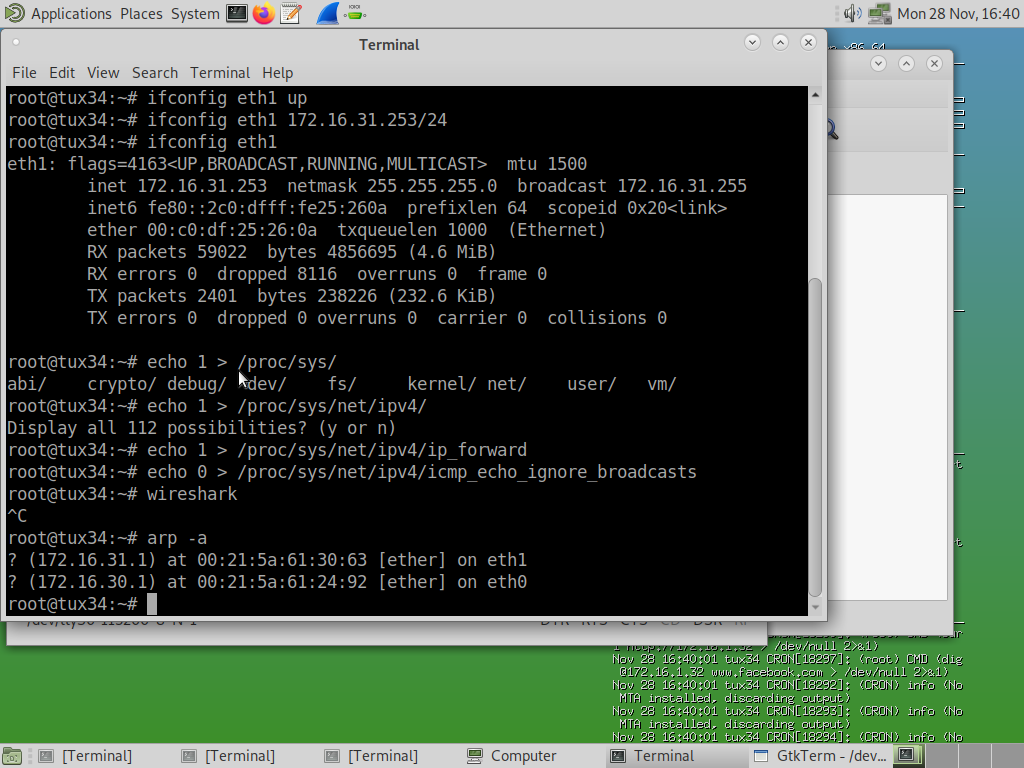
Description automatically generated*Graphical user interface, text, application, email

Description automatically generated****Figura 7.2-D:*** *Ping 172.16.30.255 no* *tux33*

***Figura 7.2-D:*** *Ping broadcast tux32*

# 7.3- Experiência 3 – Configuração de um *router* em Linux

Text

Description automatically generated ***Figura 7.3-A:*** *Enable IP forwarding, disable ICMP echo-ignore-broadcast*

***Figura 7.3-B:*** *Endereço MAC e IP no tux34.eth0 e tux34.eth1*

*Table

Description automatically generated*

***Figura 7.3-C:*** *Ping 172.16.30.254 no tux33*

Table

Description automatically generated

***Figura 7.3-D:*** *Ping 172.16.31.1 no tux33*

# 7.4- Experiência 4 – Configurar um *router* comercial e implementar NAT

# 7.5- Experiência 5 – DNS