# RCOM - Segundo Trabalho Prático
# Rede de Computadores
# Turma 1 Grupo 5

Afonso Carvalho Pereira de Sá - up201604605
Nelson Alexandre Saraiva Gregório - up200900303
Nuno Duarte Ferreira Neves Mourinha Gonçalves - up201706864

# Sumário

Este trabalho surge no contexto da disciplina de redes de computadores, de tal modo durante o trabalho configuramos uma rede de computadores e criamos uma aplicação que tira partido desta rede fazendo downloads a partir de ligações a servidores ftp.

Ao longo do desenvolvimento da rede de computadores foram-nos postas várias questões relativas ao funcionamento das redes e do modo como os computadores interagem com as mesmas.

# Introdução

Neste trabalho os objetivos consistem primeiramente em desenvolver uma aplicação de download capaz de fazer download de um ficheiro através de um cliente e um url. E desenvolver uma rede de computadores através de 3 computadores, um switch e um router.

# Parte 1 – Aplicação download

## Casos de utilização

- download ftp://<user>:<password>@<host>/<url-path>
  Utilizador fornece user e password.

- download ftp://<user>@<host>/<url-path>
  Utilizador fornece user e o programa pede introdução da password.

- download ftp://<host>/<url-path>
  Comunicação com o servidor em modo anônimo.
  A aplicação é dividida em dois grupos:

## Arquitetura

1. **Verificação e validação do URL FTP (Passado como argumento para a aplicação)**

O URL é desconstruído recorrendo à função "**sscanf**" e três strings de formatação para cada tipo de URL permitido. "ftp://%255[^:]:%255[^@]@%255[^/]%255s" - exemplo de string de formatação.

Se o URL tiver uma formatação válida, o programa faz a validação dos caracteres usados (mediante duas strings). Se todos os campos forem válidos, são então copiados para uma estrutura "**ftp_info_t**".

São usadas as funções "**getaddrinfo**" e "**inet_ntoa**" para converter o hostname para IP (e "**getnameinfo**" para imprimir um hostname no caso de o utilizador fornecer um IP)

(Funções usadas por recomendação da página do manual para gethostbyname)

Foram também criadas funções para imprimir o conteúdo da estrutura e para libertar a memória usada.

2. **Utilização de sockets para tentar descarregar o ficheiro pedido no URL**

Para a interação com sockets e comunicação com o servidor FTP foram criadas várias funções para as três fases do processo.

Para criar e estabelecer a ligação TCP:

**int open_socket(int* sock_fd, FILE** sock_file);**
**int connect_socket(int sock_fd, char* addr, uint16_t port);**

Para o processo de troca de mensagens de controlo e dados com o servidor:

**int read_single_msg(int sock_fd, char* code_str, char** msg, size_t tsec, size_t tusec);**
**int read_msg_block(int sock_fd, char* code_str);**
**int write_msg(int sock_fd, char* cmd, ...);**

As funções de leitura estão protegidas com um timeout em caso de erro.

Resumidamente, o programa lê a mensagem inicial do servidor, autentica o utilizador com **USER** e **PASS**.

Pede alteração para modo binário/imagem com o comando **TYPE**. É pedido também o tamanho e mudança de diretório com **SIZE** e **CWD**.

O comando **PASV** pede operação em modo "passivo" e é recebido o IP/Porta para a ligação TCP de dados. É criada esta nova ligação, e depois com **RETR** é pedida a transferência do ficheiro através dessa ligação.

O ficheiro é depois transferência e guardado com:

**int read_file_w_size(int retr_fd, FILE* dl, size_t file_size);**

Resta agora, fechar os descritores de ficheiros e limpar a memória.

## Notas

Alguns funcionalidades extra foram implementadas:

- Barra de progresso para a transferência.
- Medição do tempo da transferência.
- Calculado o tamanho do ficheiro recebido.

Aspectos que podem ser melhorados:

- A fase de encerramento das ligações e libertação de memória ser organizada melhor.
- Separação da camada de sockets e da interação com o servidor FTP em diferentes ficheiros.
- Melhor feedback para o utilizador em caso de erro.

## Output do programa (em anexo)

# Parte 2 – Configuração da rede

## Experiência 1 – Configurar uma rede IP

**Pergunta:** What are the ARP packets and what are they used for?

ARP (address resolution protocol) é usado para encontrar o endereço MAC associado a um determinado endereço IP, cada computador tem uma tabela com os endereços conhecidos, se necessitar de comunicar com um computador que não conheça este envia em broadcast um pacote ARP e o computador de destino responde com outro pacote ARP a identificar-se. Estes pacotes são também enviados periodicamente, não apenas em caso de falha na tabela. Pertencem à data link layer.
What are the MAC and IP addresses of ARP packets and why?

**Pergunta:** What are the MAC and IP addresses of ARP packets and why?

No caso comunicação do tux1 para o tux4, no ARP request o IP e MAC de origem pertencem ao tux1 (172.16.10.1 / 00:1e:0b:a1:f5:00), o IP de destino pertence ao tux4 (172.16.10.254) e o MAC de destino vai preenchido com zeros. Em modo de broadcast é enviado para o MAC especial (ff:ff:ff:ff:ff:ff). Caso já exista um registo na tabela, o MAC destino contém o MAC desse registo (00:c0:df:02:55:95) no cabeçalho do pacote. No ARP reply temos a situação oposta, com o tux4 como origem e o tux1 como destino.

**Pergunta:** What packets does the ping command generate?

O comando ping gera pacotes ICMP. É enviado um pacote request, e espera a recepção de um pacote reply.

**Pergunta:** What are the MAC and IP addresses of the ping packets?

No request tem:

IP e MAC de origem do tux1 (172.16.10.1 / 00:1e:0b:a1:f5:00)
IP e MAC de destino do tux4 (172.16.10.254 / 00:c0:df:02:55:95)

No reply tem o contrário:

IP e MAC de destino do tux1 (172.16.10.1 / 00:1e:0b:a1:f5:00)
IP e MAC de origem do tux4 (172.16.10.254 / 00:c0:df:02:55:95)

**Pergunta:** How to determine if a receiving Ethernet frame is ARP, IP, ICMP?

O cabeçalho ethernet indica no campo "type" se é IPV4 (0x0800) ou ARP (0x0806). Se for IPV4, o cabeçalho IP indica em "Protocol" se é ICMP (1), ou outro.

**Pergunta:** How to determine the length of a receiving frame?

Depende do tipo de dados que se foram enviados, e do tipo de encapsulamento usado.

**Pergunta:** What is the loopback interface and why is it important?

É uma interface que simula o envio de pacotes localmente, como se estes tivessem origem no exterior, e é usada para testar ligações em software sem necessidade de uma interface física. Verificamos que é enviado um pacote “LOOP” de 10 em 10 segundos. Fig.1

## Experiência 2 – Implementar duas VLAN num switch

**Pergunta:** How to configure vlan10?
Enviando os seguintes comandos através da porta de serie para o switch:

> "configure terminal"
> "vlan 10"
> "end"

Estando criada, basta adicionar portas do switch à VLAN para introduzir PCs na LAN virtual:

> "configure terminal"
> "interface fastethernet 0/i" (Seja i o número da porta a adicionar)
> "interface range fastethernet 0/i-j" (Seja [i, j] o intervalo de portas a adicionar)
> "switchport mode access" (Indica que as portas selecionadas pertencem apenas à VLAN indicada em baixo)
> "switchport access vlan 10"
> "end"

**Pergunta:** How many broadcast domains are there? How can you conclude it from the logs?
Há 2 domínios, um para cada VLAN. Pode ser concluído através da verificação dos logs. Entre os 3 computadores na experiência, broadcast em tux1 apenas apanha o tux4, e broadcast em tux2 não encontra ninguém. Fig.2

## Experiência 3 – Configurar um router em LINUX

**Nota:** Para brevidade, tux40 e tux41 representam as interfaces 0 e 1 do tux4 respectivamente. De igual forma, tux10 e tux20 são as interfaces 0 de cada tux respetivo.

**Pergunta:** What routes are there in the tuxes? What are their meaning?

- Tux1:
  Rota com destino 172.16.10.0 e gateway 0.0.0.0 (valor especial) indica que é uma rota local e que não é necessário nenhum salto (hop) extra para comunicar dentro da própria sub-rede.
  Rota com destino 172.16.11.0 e gateway 172.16.10.254 indica que tráfego com destino à sub-rede da vlan11 deve ser encaminhado para 172.16.10.254 (IP da tux40)
- Tux2: (Semelhante ao tux1, justificado pela mesma lógica)
  Rota com destino 172.16.11.0 e gateway 0.0.0.0.
  Rota com destino 172.16.10.0 e gateway 172.16.11.253 (caminho para vlan10 através de 172.16.11.253, IP da tux41)
- Tux4:
  Rota com destino 172.16.10.0 e gateway 0.0.0.0 (eth0 -> vlan10)
  Rota com destino 172.16.11.0 e gateway 0.0.0.0 (eth1 -> vlan11)
  O tux4 está ligado às duas sub-redes, uma em cada vlan, logo consegue receber tráfego de um tux e encaminhá-lo para o outro.

**Pergunta:** What information does an entry of the forwarding table contain?

Uma forwarding table contém a seguinte informação:

Destination e Genmask indentificam a sub-rede (ie. 172.16.10.0 e 255.255.255.0 = 172.16.10.0/24)

Flags indicam informação da rota (ie. U para Up, G para indicar que rota encaminha tráfego, através do gateway, para fora da sub-rede)

Metric indica o custo da rota (com base em vários fatores, número de hops e distância por exemplo).

Iface indica a interface associado à rota.

Ref indica o número de vezes que a rota foi usada para ligações.

Use indica o número de lookups feitos na rota, sendo que estes últimos dois não apresentam relevância para a experiência.

**Pergunta:** What ARP messages, and associated MAC addresses, are observed and why?

**Do ponto de vista do tux1 a fazer ping para as outras interfaces:**

São enviados requests periodicamente com origem no endereço MAC da tux10 (00:1e:0b:a1:f5:00) e destino de 00:00:00:00:00:00 (na frame o destino é da tux40 - 00:c0:df:02:55:95, por já existir na tabela, devido às experiências anteriores). A cada request é recebido um reply com origem no endereço MAC da tux40 (00:c0:df:02:55:95) e destino para tux10 (00:1e:0b:a1:f5:00).

A comunicação com a tux41 ou com a tux20 é feita através da tux40 logo só existem estas mensagens ARP.

**Na situação de ping entre tux1 e tux2: (Desta vez, com as tabelas ARP limpas):**

Como no primeiro ping, todas as tabelas estão vazias, a tux10 faz um pedido de broadcast (ff:ff:ff:ff:ff:ff) para encontrar todos os endereços MAC da sua subrede. A tux41 faz o mesmo.

A tux10 recebe reply da tux40, e a tux41 recebe da tux20. Na tux40 temos troca de mensagens ARP entre o tux40 e o tux10 apenas. Na tux41 temos troca de mensagens ARP entre o tux41 e o tux20 apenas.

O request enviado da tux10 vai pela tux40 para chegar à subrede da tux20, e o reply vai da tux20 pela tux41 daí só existirem estas mensagens ARP.

Estas mensagens são trocadas periodicamente de forma a verificar se continuam válidas (e atualizar caso contrário)

**Pergunta:** What ICMP packets are observed and why?

**Do ponto de vista do tux1 a fazer ping para as outras interfaces:**

São trocados pacotes ICMP entre a tux10 e tux40/tux41/tux20 uma vez que foi feito ping separadamente para cada interface.

**Na situação de ping entre tux1 e tux2: (Desta vez, com as tabelas ARP limpas):**

Ambas interfaces (tux40 e tux41) recebem requests da tux10 para a tux20 e replies da tux20 para a tux10.

**Pergunta:** What are the IP and MAC addresses associated to ICMP packets and why?

**Do ponto de vista do tux1 a fazer ping para as outras interfaces:**

Aqui é visível o efeito das rotas. Os únicos endereços MAC visíveis são da tux10 e da tux40, enquanto que os IPs são de todos. Examinando os pings por ordem:

tux40:

Request -> Origem: 172.16.40.1 (tux10 00:1e:0b:a1:f5:00) Destino: 172.16.40.254 (tux40 00:c0:df:02:55:95)

Reply -> Origem: 172.16.40.254 (tux40 00:c0:df:02:55:95) Destino: 172.16.40.1 (tux10 00:1e:0b:a1:f5:00)

tux41:

Request -> Origem: 172.16.40.1 (tux10 00:1e:0b:a1:f5:00) Destino: 172.16.40.253 (tux40 00:c0:df:02:55:95)

Reply -> Origem: 172.16.40.253 (tux40 00:c0:df:02:55:95) Destino: 172.16.40.1 (tux10 00:1e:0b:a1:f5:00)

tux20:

Request -> Origem: 172.16.40.1 (tux10 00:1e:0b:a1:f5:00) Destino: 172.16.41.1 (tux40 00:c0:df:02:55:95)

Reply -> Origem: 172.16.41.1 (tux40 00:c0:df:02:55:95) Destino: 172.16.40.1 (tux10 00:1e:0b:a1:f5:00)

**Na situação de ping entre tux1 e tux2: (Desta vez, com as tabelas ARP limpas):**

Na tux40:

Request -> Origem: 172.16.40.1 (tux1 00:1e:0b:a1:f5:00) Destino: 172.16.41.1 (tux40 00:c0:df:02:55:95)

Reply -> Origem: 172.16.41.1 (tux40 00:c0:df:02:55:95) Destino: 172.16.40.1 (tux1 00:1e:0b:a1:f5:00)

Na tux41:

Request -> Origem: 172.16.40.1 (tux41 00:21:5a:c3:78:76) Destino: 172.16.41.1 (tux2 00:c0:df:25:43:bc)

Reply -> Origem: 172.16.41.1 (tux40 00:c0:df:25:43:bc) Destino: 172.16.40.253 (tux1 00:21:5a:c3:78:76)

Os IPs são iguais nas 2 interfaces (são sempre 172.16.10.1 <-> 172.16.11.1). Os endereços MAC, no entanto, mostram a passagem entre sub-redes através das interfaces do tux4.

## Experiência 4 – Configurar um router comercial e implementar NAT

**Pergunta:** How to configure a static route in a commercial router?

Enviando os seguintes comandos através da porta de serie para o switch:

"conf t"
"ip route <destino> <mascara> <gateway>"
(ie. ip route 172.16.10.0 255.255.255.0 172.16.11.253)
"end"

**Pergunta:** What are the paths followed by the packets in the experiments carried out and why?

Perante a existência de uma rota (tux2 ping para tux1) através do tux4, o caminho é tux20 -> tux41 -> tux40 -> tux1. Ao remover o rota, o pedido é encaminhado para o router, e depois segue para o tux41, tux20 -> rc -> tux41 -> tux40 -> tux1.
No caso do ping para o router do lab, sem nat, o ping não obtém respostas. Com nat implementado, já é recebida a reply.

**Pergunta:** How to configure NAT in a commercial router?

Enviando os seguintes comandos através da porta de serie para o switch: (assumindo configuração previa de IPs e rotas)

"conf t"
"interface gigabitethernet 0/0" (escolher a interface "interna", ligação vlan11)
"ip nat inside"
"exit"
"interface gigabitethernet 0/1" (escolher a interface "externa", ligação ao lab router)
"ip nat outside"
"exit"
"ip nat pool ovrld 172.16.1.y9 172.16.1.y9 prefix 24"
(indica a "pool" de IPs disponiveis, neste caso só um, para o mapeamento de IPs ao sair da nossa vlan)
"ip nat inside source list 1 pool ovrld overload"
(indica os IPs que podem "usufruir" do mapeamento, definidos com o comando "access-list 1 permit")
"end"

Nota: A definição do enunciado "access-list 1 permit 172.16.1x.0 0.0.0.7" apenas permite aos IPs de 1 a 7 comunicar com o exterior. Daí o tux4 não ter acesso, visto que as suas interfaces usam 253 e 254. Para a experiencia 7 foi necessario alterar a access list para permitir acesso ao tux4 ("access-list 1 permit 172.16.1x.0 0.0.0.255").

**Pergunta:** What does NAT do?

A NAT resolve o problema de escassez de endereços IPv4. Cada dispositivo ligado à internet precisa de um IP, não podendo existir duplicados. De tal modo, aNAT permite esconder uma rede inteira e dar-lhe acesso ao exterior através de um único IP.

No caso do tux1, qualquer pacote enviado para fora da sala, quando visto por uma interface fora, vai mostrar como origem o IP do router do laboratório.

Durante a experiência 7 ao implementar NAT no tux4, o tux1 fica escondido por detrás do tux4, e qualquer pedido visto na vlan11, (ping do tux1 para o tux2 por exemplo), tem como IP de origem o tux4.

## Experiência 5 – DNS

**Pergunta:** How to configure the DNS service at an host?

Editando o ficheiro resolv.conf na pasta /etc, de forma a ter o conteúdo:

```
search netlab.fe.up.pt
nameserver 172.16.2.1
```

A opção search é usado para completar um hostname quando este tem menos que "ndots" pontos (ie. ao pesquisar "test" é tentado "test.netlab.fe.up.pt"). O nameserver indica o endereço IP de um name server que contém o mapeamento entre hostnames os respectivos endereços IP (sigarra.up.pt -> 193.137.35.140).

**Pergunta:** What packets are exchanged by DNS and what information is transported?

É enviado um pacote por UDP com uma query do tipo A (pedido de IPv4) para o hostname (google.com), (foi também enviada uma do tipo AAAA, pedido de IPv6).

Uma resposta é recebida com o IP, caso um dos DNS servers no ficheiro resolv.conf conheça o hostname.

## Experiência 6 – Ligações TCP

**Pergunta:** How many TCP connections are opened by your ftp application?

A aplicação abre duas ligações. É verificável pelos pacotes TCP com o flag SYN. Fig.12

**Pergunta:** In what connection is transported the FTP control information?

É usada a primeira ligação estabelecida onde foram trocados os comandos/respostas com o servidor FTP, assim como os "ACKs" durante a transferência do ficheiro.

**Pergunta:** What are the phases of a TCP connection?

Tem três fases: estabelecer a ligação (marcado com a flag SYN), trocar de informação (ie. mensagens de controlo e dados) e terminar a ligação (marcado com a flag FIN). Fig.13

**Pergunta:** How does the ARQ TCP mechanism work? What are the relevant TCP fields? What relevant information can be observed in the logs?

ARQ (automatic repeat request) usa o protocolo de sliding window. Os campos relevantes são o ack number, seq number e window size. Estes podem ser observados nos logs através da Fig. 4.

**Pergunta:** How does the TCP congestion control mechanism work? What are the relevant fields. How did the throughput of the data connection evolve along the time? Is it according the TCP congestion control mechanism?

O mecanismo de controlo de congestionamento de TCP baseia-se nos seguintes critérios, equidade de fluxo e eficiência. São enviados pacotes com aumento exponencial, até atingir o limite, reduz-se o número de pacotes  e de seguida começa-se a aumentar lentamente até este atingir novamente o limite e depois repetem-se os passos anteriores.

Os campos relevantes são Bytes in flight, window size e número de acks. Fig. 3. Fig. 4.

Subiu até atingir o limite da ligação. No segundo caso, subiu até ao limite da ligação, e depois desceu devido ao início do segundo download no tux2.

Sim. "dente de serra", dup acks. Maior congestionamento, menor throughput.

**Pergunta:** Is the throughput of a TCP data connections disturbed by the appearance of a second TCP connection? How?

Sim. O throughput é cortado para aproximadamente metade. Verificamos também que ao concluir a primeira transferência, a taxa da segunda aumenta. Fig. 6. Fig. 7. Fig. 8. Fig. 9.

## Experiência 7 – Implementar NAT em Linux

A implementação começa com a definição das iptables no tux4.

```
iptables –t nat –A POSTROUTING -o eth1 –j MASQUERADE
iptables –A FORWARD –i eth1 –m state --state NEW,INVALID –j DROP
```

“–A POSTROUTING” Adicionar regra à fase POSTROUTING (imediatamente antes de ser enviado o pacote)
“-o eth1” indica qual a interface de output
“–j MASQUERADE” aplica regra de IP Masquerade
“–A FORWARD” Adicionar regra à fase FORWARD (controla por onde podem ser encaminhados pacotes na rede)
“–i eth1” Filtrar pacotes vindos da interface eth1
“–m state --state NEW,INVALID” Filtrar pacotes nos estados NEW e INVALID
“–j DROP” descartar pacotes que chegam ao fim desta chain

É também importante permitir que o tux4 tenha acesso ao exterior (Alterar a access-list do router para permitir toda a gama de endereços até 254)

A interface eth0 do tux1 passa a pertencer a uma rede “interna” nova com o IP 10.10.0.1. Quando quer comunicar com algum dispositivo exterior à sua rede os pacotes vão para o tux4.

Assim que um pacote lá chega, devido ao sistema de NAT, é guardada a informação do mapeamento (IP e porta) e é transformado o header do pacote para passar o ter o IP do tux4.
Ao analisar tráfego da eth1 do tux4 em diante, o pacote já aparece como tendo origem no tux4 (172.16.11.253).
A resposta também tem como destino o tux4. Quando chega, é usada a informação guardada anteriormente, para a encaminhar para o IP interno que fez o pedido originalmente (ie. tux1).

NAT foi implementado no tux4 com sucesso.
Em anexo estão imagens com a configuração do tux1 e ping para google.com no tux4. Fig.10
Assim como a configuração do tux4 Fig.11

# Conclusão

Com este trabalho ficamos a saber como os computadores, switches e routers são configurados, de forma a criar uma rede composta por várias sub-redes.

Aprendemos como funciona e como implementar o mecanismo de NAT que ajuda a contornar o problema de escassez de IPs.

Vimos também o funcionamento do protocolo de transferência de ficheiros (FTP)

A aplicação implementada é robusta e disponibiliza informação ao utilizador de um modo legível.

As várias experiências guiaram-nos na criação de uma rede de computadores e ajudaram a melhor compreensão das várias camadas de comunicação e dos protocolos que são implementados (TCP/IP, FTP) para comunicação.

**Nota: Logs da experiência 6 não foram incluídos devido ao seu tamanho.**

# Anexos

## Fig. 1 - Loopback interface

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 2 | 1.581537219 | Cisco_7c:8f:87 | Cisco_7c:8f:87 | LOOP | 60 | Reply |
| 8 | 11.589039081 | Cisco_7c:8f:87 | Cisco_7c:8f:87 | LOOP | 60 | Reply |
| 20 | 21.600663045 | Cisco_7c:8f:87 | Cisco_7c:8f:87 | LOOP | 60 | Reply |
| 48 | 31.604295980 | Cisco_7c:8f:87 | Cisco_7c:8f:87 | LOOP | 60 | Reply |
| 72 | 41.603635421 | Cisco_7c:8f:87 | Cisco_7c:8f:87 | LOOP | 60 | Reply |
| 83 | 51.611206007 | Cisco_7c:8f:87 | Cisco_7c:8f:87 | LOOP | 60 | Reply |

## Fig. 2 - Ping broadcast

| Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|
| 172.16.40.1 | 172.16.40.255 | ICMP | 98 | Echo (ping) request id=0x31bc, seq=1/256, ttl=64 (no response found!) |
| 172.16.40.254 | 172.16.40.1 | ICMP | 98 | Echo (ping) reply id=0x31bc, seq=1/256, ttl=64 |

## Fig. 3 - Download único

```
------------- DOWNLOADING -------------
>>>>>>
RETR octave-5.0.91-w64.zip
<<<<<<
150 Opening BINARY mode data connection for octave-5.0.91-w64.zip (517761822 byt
es).
Read Timeout
Download: 100.0% [====================]
Expected size: 517761822
Received size: 517761822
Total time: 45.713948
```

## Fig. 4 - Download único - Window size + Bytes + Acks

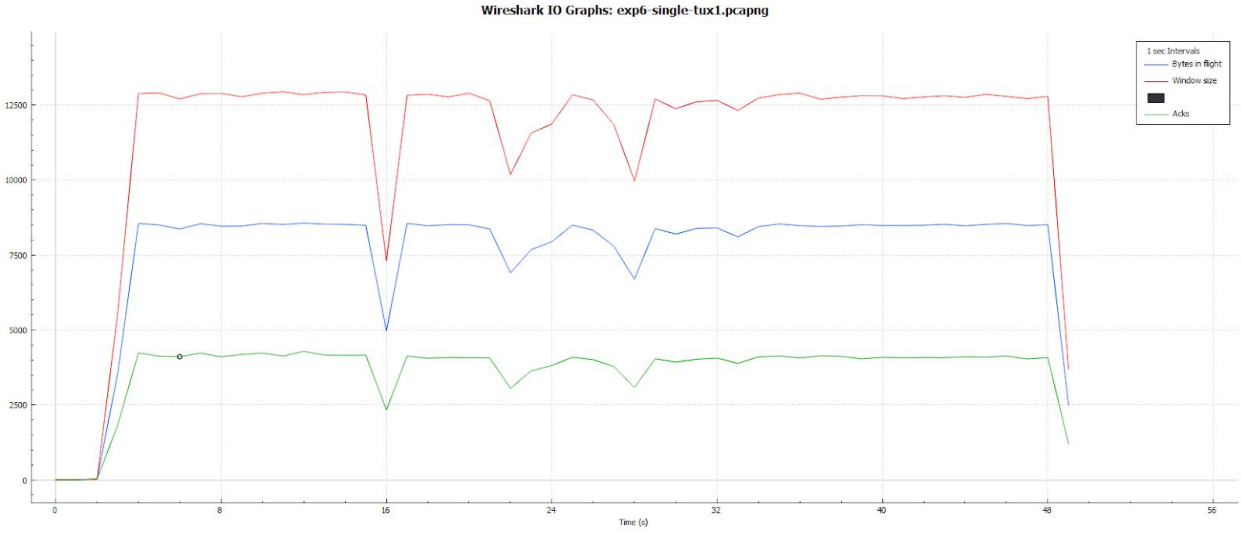


Fig. 5 - Download único - Window size + Errors

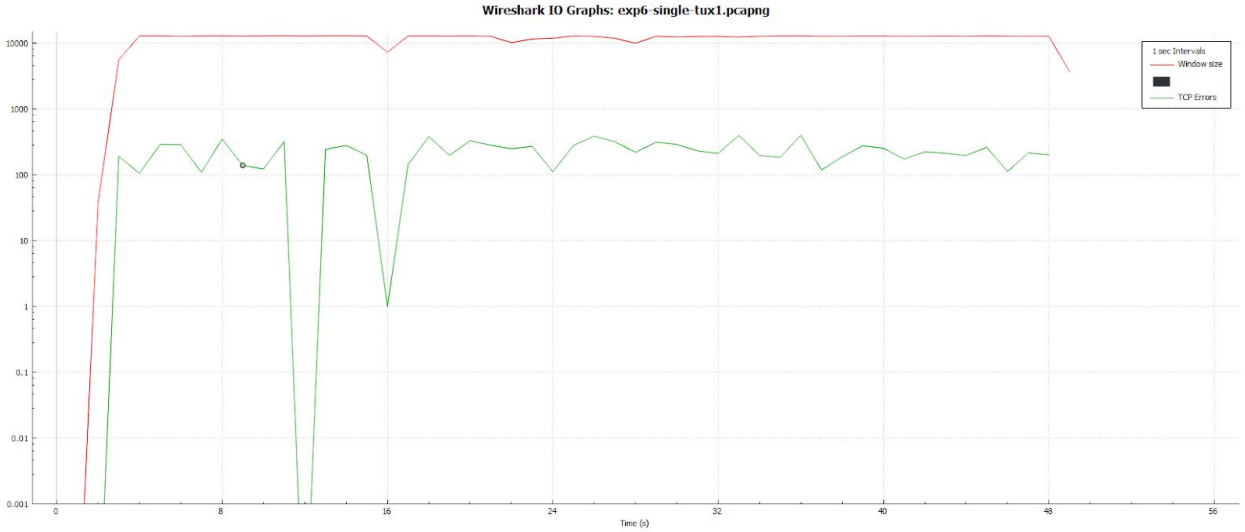



Fig. 6 - Download simultâneo - Tux 2


Fig. 7 - Download simultâneo - Tux2 - Throughput

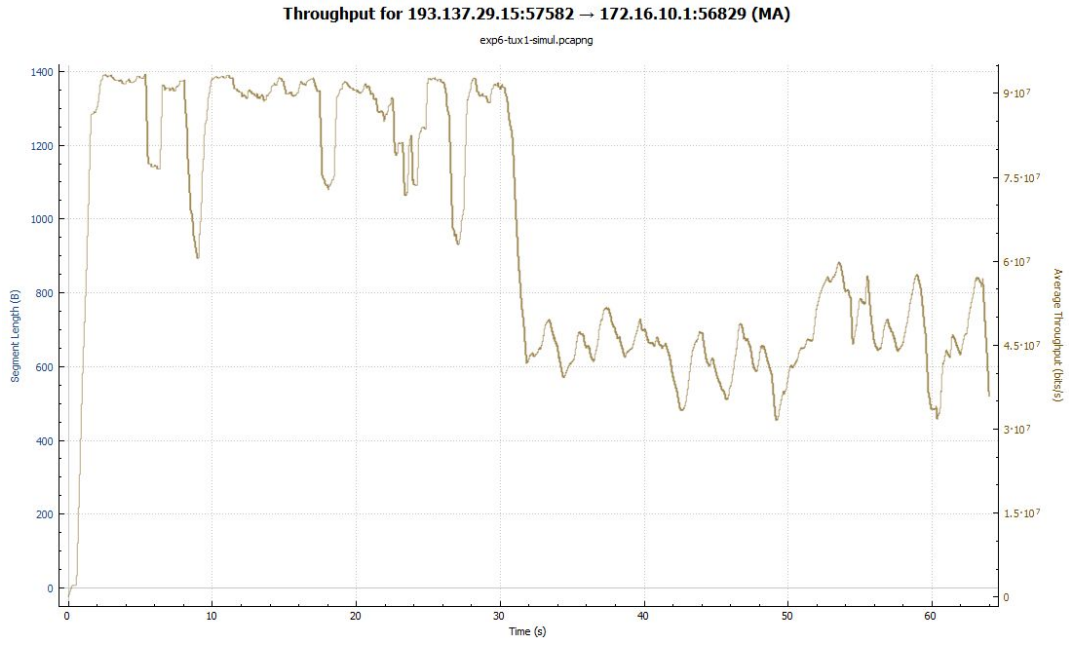



Fig. 8 -

Fig. 8 - Download simultâneo - Tux1 - Throughput

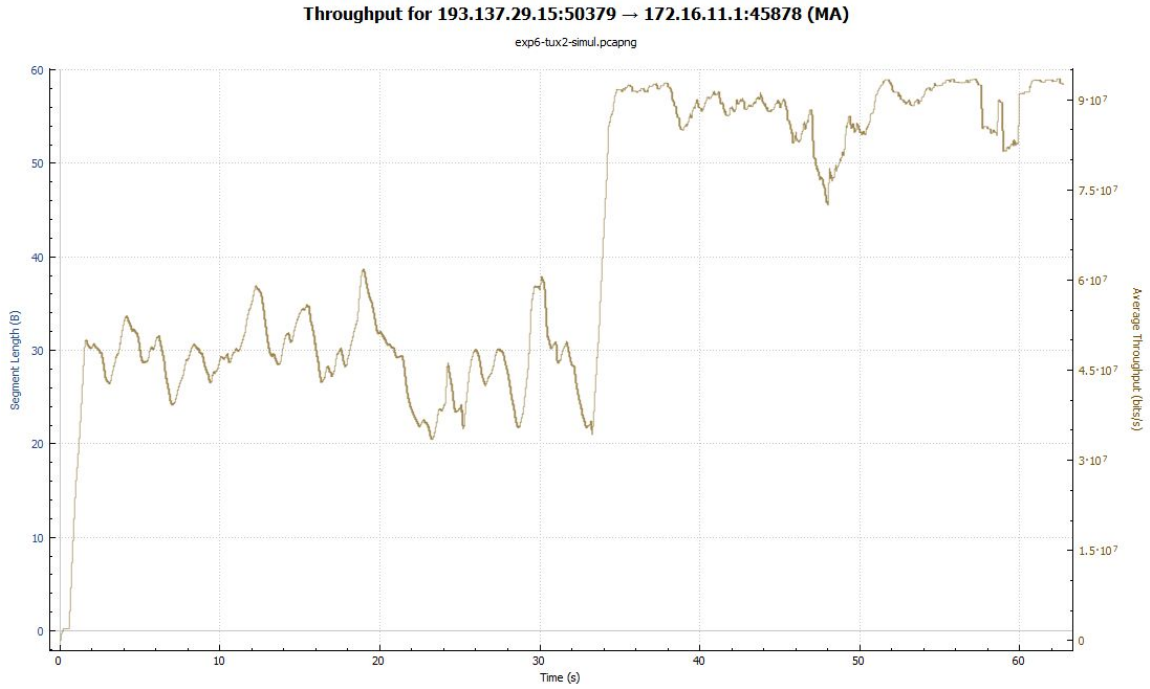



Fig. 9 - Download simultâneo - Tux 1

```
------------- DOWNLOADING -------------
>>>>>>
RETR octave-5.0.91-w64.zip
<<<<<<
150 Opening BINARY mode data connection for octave-5.0.91-w64.zip (517761822 byte
s).
Read Timeout
Download: 100.0% [====================]
Expected size: 517761822
Received size: 517761822
Total time: 62.120286
```

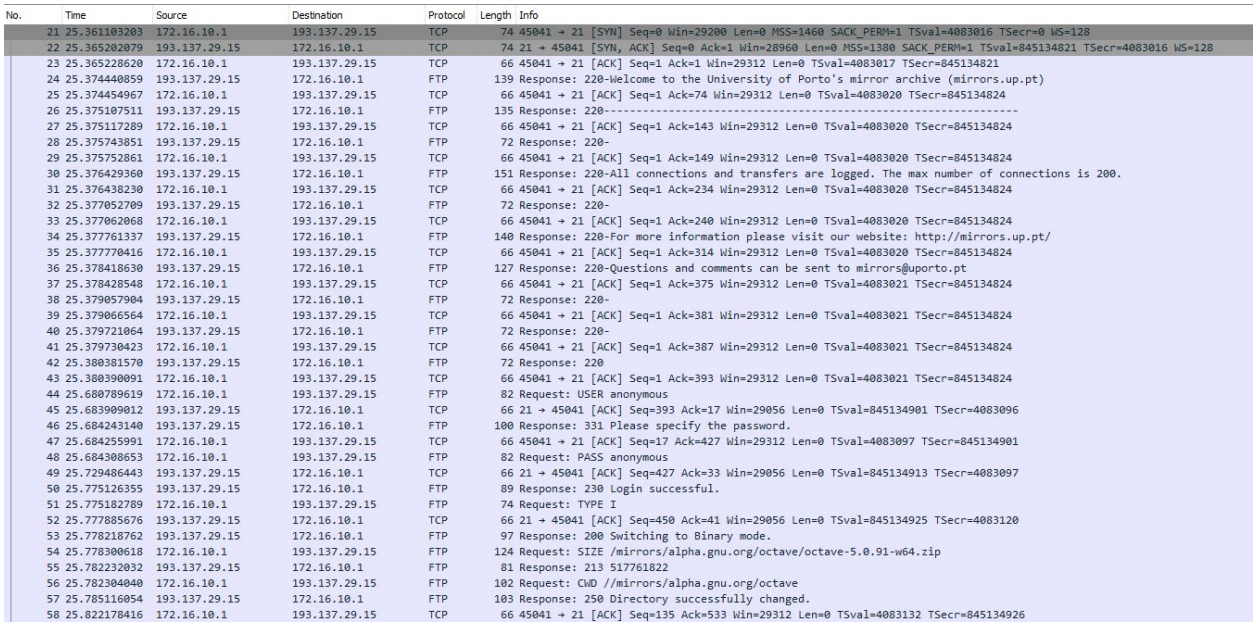
Fig. 10 - Exp7 - Configuração Tux1

```
gnu11:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.0.1  netmask 255.255.255.0  broadcast 10.10.0.255
        inet6 fe80::2c0:dfff:fe06:693e  prefixlen 64  scopeid 0x20<link>
        ether 00:c0:df:06:69:3e  txqueuelen 1000  (Ethernet)
        RX packets 5649  bytes 542482 (529.7 KiB)
        RX errors 0  dropped 237  overruns 0  frame 0
        TX packets 7413  bytes 540273 (527.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 3501  bytes 362036 (353.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3501  bytes 362036 (353.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

gnu11:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.10.0.254     0.0.0.0         UG    0      0        0 eth0
10.10.0.0       0.0.0.0         255.255.255.0   U     0      0        0 eth0
gnu11:~# ping google.com
PING google.com (172.217.16.238) 56(84) bytes of data.
64 bytes from mad08s04-in-f14.1e100.net (172.217.16.238): icmp_seq=1 ttl=49 time=15.5 ms
64 bytes from mad08s04-in-f14.1e100.net (172.217.16.238): icmp_seq=2 ttl=49 time=14.8 ms
64 bytes from mad08s04-in-f14.1e100.net (172.217.16.238): icmp_seq=3 ttl=49 time=14.8 ms
64 bytes from mad08s04-in-f14.1e100.net (172.217.16.238): icmp_seq=4 ttl=49 time=14.9 ms
^C
--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 14.816/15.027/15.524/0.301 ms
```

Fig. 11 - Exp7 - Configuração Tux4

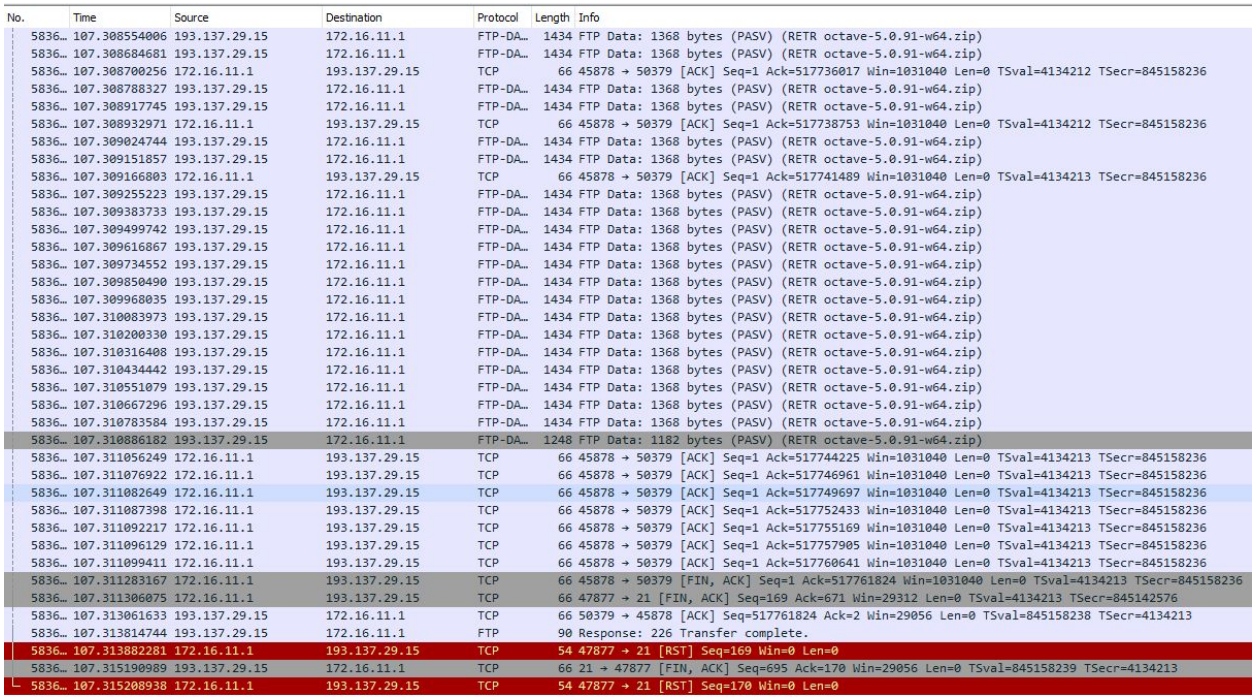
```
gnu14:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.0.254  netmask 255.255.255.0  broadcast 10.10.0.255
        inet6 fe80::221:5aff:fe5a:7b3f  prefixlen 64  scopeid 0x20<link>
        ether 00:21:5a:5a:7b:3f  txqueuelen 1000  (Ethernet)
        RX packets 6612  bytes 566330 (553.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1147  bytes 154221 (150.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 17

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.11.253  netmask 255.255.255.0  broadcast 172.16.11.255
        inet6 fe80::240:f4ff:fe6f:b6a5  prefixlen 64  scopeid 0x20<link>
        ether 00:40:f4:6f:b6:a5  txqueuelen 1000  (Ethernet)
        RX packets 43087  bytes 53267505 (50.7 MiB)
        RX errors 0  dropped 476  overruns 0  frame 0
        TX packets 33787  bytes 2630280 (2.5 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 2046  bytes 194942 (190.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2046  bytes 194942 (190.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

gnu14:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.11.254   0.0.0.0         UG    0      0        0 eth1
10.10.0.0       0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.16.11.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
```

Fig. 12 - SYN



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 21 | 25.361103203 | 172.16.10.1 | 193.137.29.15 | TCP | 74 | 45041 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4083016 TSecr=0 WS=128 |
| 22 | 25.365202079 | 193.137.29.15 | 172.16.10.1 | TCP | 74 | 21 → 45041 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1380 SACK_PERM=1 TSval=845134821 TSecr=4083016 WS=128 |
| 23 | 25.365228620 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=4083017 TSecr=845134821 |
| 24 | 25.374440859 | 193.137.29.15 | 172.16.10.1 | FTP | 139 | Response: 220-Welcome to the University of Porto's mirror archive (mirrors.up.pt) |
| 25 | 25.374454967 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=1 Ack=74 Win=29312 Len=0 TSval=4083020 TSecr=845134824 |
| 26 | 25.375107511 | 193.137.29.15 | 172.16.10.1 | FTP | 135 | Response: 220---------- |
| 27 | 25.375117289 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=1 Ack=143 Win=29312 Len=0 TSval=4083020 TSecr=845134824 |
| 28 | 25.375743851 | 193.137.29.15 | 172.16.10.1 | FTP | 72 | Response: 220- |
| 29 | 25.375752861 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=1 Ack=149 Win=29312 Len=0 TSval=4083020 TSecr=845134824 |
| 30 | 25.376429360 | 193.137.29.15 | 172.16.10.1 | FTP | 151 | Response: 220-All connections and transfers are logged. The max number of connections is 200. |
| 31 | 25.376438230 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=1 Ack=234 Win=29312 Len=0 TSval=4083020 TSecr=845134824 |
| 32 | 25.377052709 | 193.137.29.15 | 172.16.10.1 | FTP | 72 | Response: 220- |
| 33 | 25.377062068 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=1 Ack=240 Win=29312 Len=0 TSval=4083020 TSecr=845134824 |
| 34 | 25.377761337 | 193.137.29.15 | 172.16.10.1 | FTP | 140 | Response: 220-For more information please visit our website: http://mirrors.up.pt/ |
| 35 | 25.377770416 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=1 Ack=314 Win=29312 Len=0 TSval=4083020 TSecr=845134824 |
| 36 | 25.378418630 | 193.137.29.15 | 172.16.10.1 | FTP | 127 | Response: 220-Questions and comments can be sent to mirrors@uporto.pt |
| 37 | 25.378428548 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=1 Ack=375 Win=29312 Len=0 TSval=4083021 TSecr=845134824 |
| 38 | 25.379057904 | 193.137.29.15 | 172.16.10.1 | FTP | 72 | Response: 220- |
| 39 | 25.379066564 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=1 Ack=381 Win=29312 Len=0 TSval=4083021 TSecr=845134824 |
| 40 | 25.379721064 | 193.137.29.15 | 172.16.10.1 | FTP | 72 | Response: 220- |
| 41 | 25.379730423 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=1 Ack=387 Win=29312 Len=0 TSval=4083021 TSecr=845134824 |
| 42 | 25.380381570 | 193.137.29.15 | 172.16.10.1 | FTP | 72 | Response: 220 |
| 43 | 25.380390091 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=1 Ack=393 Win=29312 Len=0 TSval=4083021 TSecr=845134824 |
| 44 | 25.680789619 | 172.16.10.1 | 193.137.29.15 | FTP | 82 | Request: USER anonymous |
| 45 | 25.683909012 | 193.137.29.15 | 172.16.10.1 | TCP | 66 | 21 → 45041 [ACK] Seq=393 Ack=17 Win=29056 Len=0 TSval=845134901 TSecr=4083096 |
| 46 | 25.684243140 | 193.137.29.15 | 172.16.10.1 | FTP | 100 | Response: 331 Please specify the password. |
| 47 | 25.684255991 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=17 Ack=427 Win=29312 Len=0 TSval=4083097 TSecr=845134901 |
| 48 | 25.684308653 | 172.16.10.1 | 193.137.29.15 | FTP | 82 | Request: PASS anonymous |
| 49 | 25.729486443 | 193.137.29.15 | 172.16.10.1 | TCP | 66 | 21 → 45041 [ACK] Seq=427 Ack=33 Win=29056 Len=0 TSval=845134913 TSecr=4083097 |
| 50 | 25.775126355 | 193.137.29.15 | 172.16.10.1 | FTP | 89 | Response: 230 Login successful. |
| 51 | 25.775182789 | 172.16.10.1 | 193.137.29.15 | FTP | 74 | Request: TYPE I |
| 52 | 25.777885676 | 193.137.29.15 | 172.16.10.1 | TCP | 66 | 21 → 45041 [ACK] Seq=450 Ack=41 Win=29056 Len=0 TSval=845134925 TSecr=4083120 |
| 53 | 25.778218762 | 193.137.29.15 | 172.16.10.1 | FTP | 97 | Response: 200 Switching to Binary mode. |
| 54 | 25.778300618 | 172.16.10.1 | 193.137.29.15 | FTP | 124 | Request: SIZE /mirrors/alpha.gnu.org/octave/octave-5.0.91-w64.zip |
| 55 | 25.782232032 | 193.137.29.15 | 172.16.10.1 | FTP | 81 | Response: 213 517761822 |
| 56 | 25.782304040 | 172.16.10.1 | 193.137.29.15 | FTP | 102 | Request: CWD //mirrors/alpha.gnu.org/octave |
| 57 | 25.785116054 | 193.137.29.15 | 172.16.10.1 | FTP | 103 | Response: 250 Directory successfully changed. |
| 58 | 25.822178416 | 172.16.10.1 | 193.137.29.15 | TCP | 66 | 45041 → 21 [ACK] Seq=135 Ack=533 Win=29312 Len=0 TSval=4083132 TSecr=845134926 |

Fig. 13 - FIN



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 5836… | 107.308554006 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.308684681 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.308700256 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 45878 → 50379 [ACK] Seq=1 Ack=517736017 Win=1031040 Len=0 TSval=4134212 TSecr=845158236 |
| 5836… | 107.308788327 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.308917745 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.308932971 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 45878 → 50379 [ACK] Seq=1 Ack=517738753 Win=1031040 Len=0 TSval=4134212 TSecr=845158236 |
| 5836… | 107.309024744 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.309151857 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.309166803 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 45878 → 50379 [ACK] Seq=1 Ack=517741489 Win=1031040 Len=0 TSval=4134213 TSecr=845158236 |
| 5836… | 107.309255223 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.309383733 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.309499742 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.309616867 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.309734552 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.309850490 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.309968035 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.310083973 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.310200330 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.310316408 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.310434442 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.310551079 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.310667296 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.310783584 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1434 | FTP Data: 1368 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.310886182 | 193.137.29.15 | 172.16.11.1 | FTP-DA… | 1248 | FTP Data: 1182 bytes (PASV) (RETR octave-5.0.91-w64.zip) |
| 5836… | 107.311056249 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 45878 → 50379 [ACK] Seq=1 Ack=517744225 Win=1031040 Len=0 TSval=4134213 TSecr=845158236 |
| 5836… | 107.311076922 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 45878 → 50379 [ACK] Seq=1 Ack=517746961 Win=1031040 Len=0 TSval=4134213 TSecr=845158236 |
| 5836… | 107.311082649 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 45878 → 50379 [ACK] Seq=1 Ack=517749697 Win=1031040 Len=0 TSval=4134213 TSecr=845158236 |
| 5836… | 107.311087398 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 45878 → 50379 [ACK] Seq=1 Ack=517752433 Win=1031040 Len=0 TSval=4134213 TSecr=845158236 |
| 5836… | 107.311092217 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 45878 → 50379 [ACK] Seq=1 Ack=517755169 Win=1031040 Len=0 TSval=4134213 TSecr=845158236 |
| 5836… | 107.311096129 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 45878 → 50379 [ACK] Seq=1 Ack=517757905 Win=1031040 Len=0 TSval=4134213 TSecr=845158236 |
| 5836… | 107.311099411 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 45878 → 50379 [ACK] Seq=1 Ack=517760641 Win=1031040 Len=0 TSval=4134213 TSecr=845158236 |
| 5836… | 107.311283167 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 45878 → 50379 [FIN, ACK] Seq=1 Ack=517761824 Win=1031040 Len=0 TSval=4134213 TSecr=845158236 |
| 5836… | 107.311306075 | 172.16.11.1 | 193.137.29.15 | TCP | 66 | 47877 → 21 [FIN, ACK] Seq=169 Ack=671 Win=29312 Len=0 TSval=4134213 TSecr=845142576 |
| 5836… | 107.313061633 | 193.137.29.15 | 172.16.11.1 | TCP | 66 | 50379 → 45878 [ACK] Seq=517761824 Ack=2 Win=29056 Len=0 TSval=845158238 TSecr=4134213 |
| 5836… | 107.313814744 | 193.137.29.15 | 172.16.11.1 | FTP | 90 | Response: 226 Transfer complete. |
| 5836… | 107.313882281 | 172.16.11.1 | 193.137.29.15 | TCP | 54 | 47877 → 21 [RST] Seq=169 Win=0 Len=0 |
| 5836… | 107.315190989 | 193.137.29.15 | 172.16.11.1 | TCP | 66 | 21 → 47877 [FIN, ACK] Seq=695 Ack=170 Win=29056 Len=0 TSval=845158239 TSecr=4134213 |
| 5836… | 107.315208938 | 172.16.11.1 | 193.137.29.15 | TCP | 54 | 47877 → 21 [RST] Seq=170 Win=0 Len=0 |

## Output do programa

```
~$ download ftp://ftp.up.pt/mirrors/alpha.gnu.org/octave/octave-3.4.2-rc1.tar.gz
--------------------------------------------------------------------------------
|TYPE     : ANOMYMOUS
|USR      : anonymous
|PWD      : anonymous
|URL      : /mirrors/alpha.gnu.org/octave/octave-3.4.2-rc1.tar.gz
|PATH     : /mirrors/alpha.gnu.org/octave
|FILENAME : octave-3.4.2-rc1.tar.gz
|LOCATION : ftp.up.pt
|HOST     : mirrors.up.pt
|ADDR     : 193.137.29.15
--------------------------------------------------------------------------------
------------ CONNECTION ------------
<<<<<<
220-Welcome to the University of Porto's mirror archive (mirrors.up.pt)
220----------------------------------------------------------------
220-
220-All connections and transfers are logged. The max number of connections is 200.
220-
220-For more information please visit our website: http://mirrors.up.pt/
220-Questions and comments can be sent to mirrors@uporto.pt
220-
220-
220
------------ AUTHENTIFICATION ------------
>>>>>>
USER anonymous
<<<<<<
331 Please specify the password.
>>>>>>
PASS anonymous
<<<<<<
230 Login successful.
------------ SET TYPE ------------
>>>>>>
TYPE I
<<<<<<
200 Switching to Binary mode.
------------ GET SIZE ------------
>>>>>>
SIZE /mirrors/alpha.gnu.org/octave/octave-3.4.2-rc1.tar.gz
<<<<<<
213 17634817
------------ CHANGE DIRECTORY ------------
>>>>>>
CWD //mirrors/alpha.gnu.org/octave
<<<<<<
250 Directory successfully changed.
------------ PASSIVE MODE ------------
>>>>>>
PASV
```

```
<<<<<<
227 Entering Passive Mode (193,137,29,15,207,87).
IP: 193.137.29.15
PORT: 53079
---------- DOWNLOADING ----------
>>>>>>
RETR octave-3.4.2-rc1.tar.gz
<<<<<<
150 Opening BINARY mode data connection for octave-3.4.2-rc1.tar.gz (17634817 bytes).
Download: 100.0% [====================]
<<<<<<
226 Transfer complete.
---------- METRICS ----------
Expected size: 17634817
Received size: 17634817
Total time: 1.273581

~$ md5sum octave-3.4.2-rc1.tar.gz
727d413db458320ee433b28b5787ae59  octave-3.4.2-rc1.tar.gz

~$ wget ftp://ftp.up.pt/mirrors/alpha.gnu.org/octave/octave-3.4.2-rc1.tar.gz
~$ …

~$ md5sum octave-3.4.2-rc1.tar.gz
727d413db458320ee433b28b5787ae59  octave-3.4.2-rc1.tar.gz
```

## Código do programa

### main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <time.h>



#include "ftp_info.h"
#include "socket_helper.h"
#include "defs.h"


// https://tools.ietf.org/html/rfc959
```

```c
int main(int argc, char const *argv[])
{
    if(argc != 2)
    {
        printf("Usage:\ndownload
ftp://[<user>:<password>@]<host>/<url-path>\n");
        return BAD_ARGS;
    }

    // Parse url string
    ftp_info_t ftp_info;
    int err = build_ftp_info(argv[1], &ftp_info);
    if(err != OK)
        return err;

    // Print url information
    print_ftp_info(&ftp_info);


    // Prepare tcp socket
    int sock_fd = 0;
    FILE* sock_file;
    err = open_socket(&sock_fd, &sock_file);
    if(err != OK)
        return err;


    // Try to connect
    printf("------------- CONNECTION -------------\n");
    err = connect_socket(sock_fd, ftp_info.addr, 21);
    if(err != OK)
        return err;

    char* reply;
    size_t timeout_sec = 30;
```

```
    size_t timeout_usec = 0;
    char* line_endings = "\r\n";

    err = read_msg_block(sock_fd, "220");
    if(err != OK)
        return err;


    printf("\n------------- AUTHENTIFICATION -------------\n");

    write_msg(sock_fd, "USER %s%s", ftp_info.usr, line_endings);

    err = read_single_msg(sock_fd, "331", NULL, timeout_sec,
timeout_usec);
    if(err != OK)
        return err;

    write_msg(sock_fd, "PASS %s%s", ftp_info.pwd, line_endings);

    err = read_single_msg(sock_fd, "230", NULL, timeout_sec,
timeout_usec);
    if(err != OK)
        return err;




    printf("\n------------- SET TYPE -------------\n");

    write_msg(sock_fd, "TYPE I%s", line_endings);

    err = read_single_msg(sock_fd, "200", NULL, timeout_sec,
timeout_usec);
    if(err != OK)
        return err;
```

```
    printf("\n------------- GET SIZE -------------\n");

    write_msg(sock_fd, "SIZE %s%s", ftp_info.url_path, line_endings);

    int got_size = 0;
    size_t file_size = 0;

    err = read_single_msg(sock_fd, "213", &reply, timeout_sec,
timeout_usec);
    if(err == OK) {
        file_size = strtol(&reply[4], NULL, 10);
        got_size = 1;
        free(reply);
    }


    printf("\n------------- CHANGE DIRECTORY -------------\n");

    write_msg(sock_fd, "CWD /%s%s", ftp_info.path, line_endings);

    err = read_msg_block(sock_fd, "250");
    if(err != OK)
        return err;




    printf("\n------------- PASSIVE MODE -------------\n");

    write_msg(sock_fd, "PASV%s", line_endings);

    err = read_single_msg(sock_fd, "227", &reply, timeout_sec,
timeout_usec);
    if(err != OK)
        return err;
```

```
    uint8_t ip1;
    uint8_t ip2;
    uint8_t ip3;
    uint8_t ip4;
    uint8_t port_h;
    uint8_t port_l;
    err = sscanf(reply, "%*[^(](%hhu,%hhu,%hhu,%hhu,%hhu,%hhu).", &ip1,
&ip2, &ip3, &ip4, &port_h, &port_l);

    free(reply);

    if(err != 6)
        return 1;

    uint16_t retr_port = port_h * 256 + port_l;
    char retr_ip[15];
    snprintf(retr_ip, 15, "%u.%u.%u.%u", ip1, ip2, ip3, ip4);

    printf("IP: %s\n", retr_ip);
    printf("PORT: %u\n", retr_port);

    int retr_fd = 0;
    FILE* retr_file;
    err = open_socket(&retr_fd, &retr_file);
    if(err != OK)
        return err;

    err = connect_socket(retr_fd, retr_ip, retr_port);
    if(err != OK)
        return err;

    printf("\n------------- DOWNLOADING -------------\n");

    write_msg(sock_fd, "RETR %s%s", ftp_info.filename, line_endings);
```

```
    err = read_single_msg(sock_fd, "150", &reply, timeout_sec,
timeout_usec);
    if(err != OK)
        return err;

    FILE* dl;
    char trash[1024];

    if ( (dl = fopen(ftp_info.filename, "wb")) == NULL )
    {
        perror("fopen:");
        return 3;
    }

    if(got_size == 0) {

        err = sscanf(reply, "%[^(](%ld", trash, &file_size);
        printf("%ld\n", file_size);
    }

    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);
    read_file_w_size(retr_fd, dl, file_size);
    clock_gettime(CLOCK_MONOTONIC, &end);

    fseek(dl, 0L, SEEK_END);
    long received_size = ftell(dl);

    fclose(dl);

    err = read_single_msg(sock_fd, "226", &reply, timeout_sec,
timeout_usec);
    if(err != OK)
        return err;

    printf("\n------------- METRICS -------------\n");
```

```
    close(retr_fd);
    free(reply);

    close(sock_fd);
    free_ftp_info(&ftp_info);

    long seconds = end.tv_sec - start.tv_sec;
    long ns = end.tv_nsec - start.tv_nsec;

    if (start.tv_nsec > end.tv_nsec) { // clock underflow
        --seconds;
        ns += 1000000000;
    }

    printf("Expected size: %ld\n", file_size);
    printf("Received size: %ld\n", received_size);
    printf("Total time: %f\n", (double)seconds +
(double)ns/(double)1000000000);

    return OK;
}
```

## ftp_info.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <netdb.h>
#include <arpa/inet.h>

#include "ftp_info.h"
#include "defs.h"
```

```
const char* valid_login =
"abcdefghijklmnopqrstuvxywzABCDEFGHIJKLMNOPQRSTUVXYWZ0123456789@.";
const char* valid_url =
"abcdefghijklmnopqrstuvxywzABCDEFGHIJKLMNOPQRSTUVXYWZ0123456789-_./";

typedef struct addrinfo addrinfo_t;
typedef struct sockaddr_in sockaddr_in_t;

#define EXPECTED_URL_COUNT 4

int host_to_ipv4(const char* hostname, ftp_info_t* ftp_info)
{

    addrinfo_t hints;
    addrinfo_t* results;

    memset(&hints, 0, sizeof(addrinfo_t));
    hints.ai_family = AF_INET;    /* Allow IPv4 or IPv6 */
    hints.ai_socktype = 0; /* Datagram socket */
    hints.ai_flags = AI_NUMERICSERV | AI_PASSIVE;    /* For wildcard IP
address */
    hints.ai_protocol = 0;          /* Any protocol */
    hints.ai_canonname = NULL;
    hints.ai_addr = NULL;
    hints.ai_next = NULL;

    // Translate host to ip address
    int err = getaddrinfo(hostname, "21", &hints, &results);
    if (err != OK) {
        //fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(err));
        printf("Failed to resolve host name\n");
        return GETADDRINFO_ERROR;
    }

    // Copy address to ftp_info_t
    sockaddr_in_t* addr = (sockaddr_in_t*)results->ai_addr;
    char* addr_str = inet_ntoa((struct in_addr)addr->sin_addr);
```

```
    size_t len = strlen(addr_str);
    ftp_info->addr = (char*) malloc(len);
    memcpy(ftp_info->addr, addr_str, len);

    // Try to translate address to host (in case user inserted ip address
directly)
    char hbuf[256], sbuf[8];
    err = getnameinfo((struct sockaddr*) addr, 16, hbuf, sizeof(hbuf),
sbuf, sizeof(sbuf), NI_NUMERICSERV);
    if (err == OK) {
        len = strlen(hbuf);
        ftp_info->host = (char*) malloc(len);
        memcpy(ftp_info->host, hbuf, len);
    }
    else
    {
        ftp_info->host = (char*) malloc(4);
        memcpy(ftp_info->host, "N/A", 4);
    }

    // Free memory used by getaddrinfo
    freeaddrinfo(results);


    //struct hostent *h;
    // if ((h=gethostbyname(hostname)) == NULL) {
    //     herror("gethostbyname");
    //     exit(1);
    // }
    // printf("Host name  : %s\n", h->h_name);
    // printf("IP Address : %s\n",inet_ntoa(*((struct in_addr
*)h->h_addr)));

    return OK;
}

int validade_string(char* str, const char* valid, char* field_name)
```

```
{
    if (str[strspn(str, valid)] == '\0')
        return OK;

    printf("INVALID %s -> %s\n", field_name, str);
    return INVALID_CHAR;
}

int build_ftp_info(const char* full_url, ftp_info_t* ftp_info)
{

    char usr[256] = {0};
    char pwd[256] = {0};
    char location[256] = {0};
    char url_path[256] = {0};
    char prompt[300] = {0};

    // Try User and password
    int i = sscanf(full_url, "ftp://%255[^:]:%255[^@]@%255[^/]%255s", usr,
pwd, location, url_path);
    if (i != 4)
    {
        // Try User only
        memset(usr, 0, 256);
        memset(pwd, 0, 256);
        memset(location, 0, 256);
        memset(url_path, 0, 256);
        i = sscanf(full_url, "ftp://%255[^@]@%255[^/]%255s", usr,
location, url_path);
        if(i != 3)
        {
            // Try Anonymous mode
            memset(usr, 0, 256);
            memset(location, 0, 256);
            memset(url_path, 0, 256);
            i = sscanf(full_url, "ftp://%255[^/]%255s", location,
url_path);
```

```
            if (i != 2)
            {
                printf("Invalid URL format. Try:\n");
                printf("download
ftp://<user>:<password>@<host>/<url-path>\n");
                printf("download ftp://<user>@<host>/<url-path>\n");
                printf("download ftp://<host>/<url-path>\n");
                return INVALID_URL_FORMAT;
            }
            else
            {
                strncpy(usr, "anonymous", 9);
                strncpy(pwd, "anonymous", 9);
            }
        }
        else
        {
            sprintf(prompt, "Type password for user %s: ", usr);
            char* pass = getpass(prompt);
            if(pass == NULL)
            {
                printf("Invalid password\n");
                return INVALID_URL_FORMAT;
            }
            strncpy(pwd, pass, strlen(pass));
        }
    }


    // Invalid characters
    if (validade_string(usr, valid_login, "USER") != OK ||
        validade_string(pwd, valid_login, "PASSWORD") != OK ||
        validade_string(location, valid_url, "HOST/ADDR") != OK ||
        validade_string(url_path, valid_url, "PATH") != OK)
    { return INVALID_CHAR; }

    size_t len = strlen(usr);
```

```
    ftp_info->usr = (char*) malloc(len);
    memcpy(ftp_info->usr, usr, len);

    len = strlen(pwd);
    ftp_info->pwd = (char*) malloc(len);
    memcpy(ftp_info->pwd, pwd, len);

    len = strlen(location);
    ftp_info->location = (char*) malloc(len);
    memcpy(ftp_info->location, location, len);

    len = strlen(url_path);
    ftp_info->url_path = (char*) malloc(len);
    memcpy(ftp_info->url_path, url_path, len);

    char* filename = strrchr(url_path, '/');
    if(filename == NULL) {
        printf("Format validation failed?\n");
        return INVALID_URL_FORMAT;
    }

    size_t filename_len = strlen(filename);
    ftp_info->path = (char*) malloc(len - filename_len);
    memcpy(ftp_info->path, url_path, len - filename_len);
//sprintf(ftp_info->path, "./%s", url_path)

    ftp_info->filename = (char*) malloc(filename_len);
    memcpy(ftp_info->filename, &url_path[len - filename_len + 1],
filename_len);

    ftp_info->type = ( strncmp(ftp_info->usr, "anonymous", 10) == 0 ) ?
FTP_ANON :  FTP_USER;

    return host_to_ipv4(ftp_info->location, ftp_info);
}

void print_ftp_info(ftp_info_t* ftp_info)
```

```
{
    if(ftp_info == NULL)
        return;


printf("\n----------------------------------------------------------------
----------------\n");
    printf("|TYPE     : %.68s\n", (ftp_info->type == 0 ? "ANOMYMOUS" :
"LOGIN"));
    printf("|USR      : %.68s\n", ftp_info->usr);
    printf("|PWD      : %.68s\n", ftp_info->pwd);
    printf("|URL      : %.68s\n", ftp_info->url_path);
    printf("|PATH     : %.68s\n", ftp_info->path);
    printf("|FILENAME : %.68s\n", ftp_info->filename);
    printf("|LOCATION : %.68s\n", ftp_info->location);
    printf("|HOST     : %.68s\n", ftp_info->host);
    printf("|ADDR     : %.68s\n", ftp_info->addr);

printf("----------------------------------------------------------------
--------------\n\n");
}

void free_ftp_info(ftp_info_t* ftp_info)
{
    if(ftp_info == NULL)
        return;

    if(ftp_info->usr)
        free(ftp_info->usr);
    if(ftp_info->pwd)
        free(ftp_info->pwd);
    if(ftp_info->location)
        free(ftp_info->location);
    if(ftp_info->host)
        free(ftp_info->host);
    if(ftp_info->addr)
        free(ftp_info->addr);
```

```
    if(ftp_info->url_path)
        free(ftp_info->url_path);
    if(ftp_info->path)
        free(ftp_info->path);
    if(ftp_info->filename)
        free(ftp_info->filename);
}
```

## socket_helper.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <unistd.h>
#include <string.h>

#include <arpa/inet.h>

#include "socket_helper.h"
#include "defs.h"

char request[256];

int open_socket(int* sock_fd, FILE** sock_file)
{
    if ((*sock_fd = socket(AF_INET,SOCK_STREAM,0)) < 0)
    {
        perror("socket()");
        return OPEN_SOCKET_ERROR;
    }

    *sock_file = fdopen(*sock_fd, "r");

    return OK;
}

int connect_socket(int sock_fd, char* addr, uint16_t port)
{
```

```
    struct sockaddr_in server_addr;

    /*server address handling*/
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(addr);  /*32 bit Internet
address network byte ordered*/
    server_addr.sin_port = htons(port);               /*server TCP port
must be network byte ordered */

    if(connect(sock_fd, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0)
    {
        perror("connect()");
        return CONNECT_SOCKET_ERROR;
    }
    return OK;
}

int read_msg(int sock_fd, char* code_str, char** out_msg)
{

    char msg[256] = "";
    uint8_t len = 0;

    uint8_t res = 0;
    char buf[1] = "";

    char code[4];


    while ( 1 )
    {

        res = read(sock_fd,buf,1);
        if (res == -1)
        {
```

```
            perror("read: ");
            return 200;
        }

        msg[len++] = buf[0];

        if( (msg[len - 2] == '\r' && msg[len - 1] == '\n') || len == 256 )
        {

            strncpy(code, msg, 3);
            printf("%s", msg);

            if(strncmp(code, code_str, 3) != OK)
            {
                printf("Return code mismatch\n");
                return 128;
            }
            else
            {
                if(out_msg != NULL)
                {
                    *out_msg = malloc(strlen(msg));
                    strncpy(*out_msg, msg, strlen(msg));
                }
                return OK;
            }
        }
    }
}

int read_file_w_size(int retr_fd, FILE* dl, size_t file_size)
{
    size_t block_size = 1024;
    uint8_t* data = malloc(block_size);
    size_t size = file_size;
    size_t print_timer = 1000;
    size_t timer = 0;
```

```
    int br = 0;
    while ( size > 0 )
    {
        br = read(retr_fd, data, block_size);
        size -= br;

        if (br < 0)
        {
            printf("Download fail\n");
            return 4;
        }

        if (fwrite(data, br, 1, dl) < 0)
        {
            printf("Saving download fail\n");
            return 5;
        }

        if(timer >= print_timer) {
            double j = (1-((double)size/file_size))*100;
            int k = (int)j/5;

            printf("Download: %.1f%% [%.*s>%.*s]\r", j, k,
"====================", (20-k), "                    ");
            fflush(stdout);
            timer = 0;
        }
        timer++;
    }

    free(data);

    fflush(stdout);
    printf("Download: 100.0%% [====================]\n");

    return OK;
}
```

```c
int read_msg_block(int sock_fd, char* code_str)
{

    printf("<<<<<<\n");

    fd_set read_check;
    FD_ZERO(&read_check);
    FD_SET(sock_fd, &read_check);
    struct timeval timeout;
    //char code[4];

    int retries = 3;

    while( retries > 0 ) {
        timeout.tv_sec = 0;
        timeout.tv_usec = 100 * 1000;

        int n = select(sock_fd + 1, &read_check, NULL, NULL, &timeout);
        if (n == -1) {
            retries--;
            continue;
        }
        else if (n == 0) {
            retries--;
            continue;
        }
        if (!FD_ISSET(sock_fd, &read_check)) {
            retries--;
            continue;
        }

        if(read_msg(sock_fd, code_str, NULL) != OK) {
            return 128;
        }
        retries = 3;
    }
```

```
    return OK;
}

int read_single_msg(int sock_fd, char* code_str, char** msg, size_t tsec,
size_t tusec)
{

    fd_set read_check;
    FD_ZERO(&read_check);
    FD_SET(sock_fd, &read_check);
    struct timeval timeout;

    timeout.tv_sec = tsec;
    timeout.tv_usec = tusec;

    int n = select(sock_fd + 1, &read_check, NULL, NULL, &timeout);
    if (n == -1) {
        printf("Read Timeout\n");
        return -1;
    }
    else if (n == 0) {
        printf("Read Timeout\n");
        return -1;
    }
    if (!FD_ISSET(sock_fd, &read_check)) {
        printf("Read Timeout\n");
        return -1;
    }

    printf("<<<<<<\n");
    return read_msg(sock_fd, code_str, msg);
}

int write_msg(int sock_fd, char* cmd, ...)
{
    va_list va;
```

```
    va_start (va, cmd);
    vsnprintf(request, 256, cmd, va);
    va_end(va);

    write(sock_fd, request, strlen(request));
    printf(">>>>>>\n");
    printf("%s", request);

    return OK;
}
```

## socket_helper.h

```
#ifndef __SOCKET_HELPER__
#define __SOCKET_HELPER__

int open_socket(int* sock_fd, FILE** sock_file);

int connect_socket(int sock_fd, char* addr, uint16_t port);

int read_file_w_size(int retr_fd, FILE* dl, size_t file_size);

int read_msg_block(int sock_fd, char* code_str);

int read_single_msg(int sock_fd, char* code_str, char** msg, size_t tsec,
size_t tusec);

int write_msg(int sock_fd, char* cmd, ...);

#endif /*__SOCKET_HELPER__*/
```

## ftp_info.h

```
#ifndef __FTP_INFO__
```

```c
#define __FTP_INFO__

typedef struct sockaddr_in sockaddr_in_t;

typedef enum {
    FTP_ANON,
    FTP_USER
} ftp_type_t;

typedef struct {
    char* usr;
    char* pwd;
    char* location;
    char* host;
    char* addr;
    char* url_path;
    char* path;
    char* filename;
    ftp_type_t type;
} ftp_info_t;


int build_ftp_info(const char* full_url, ftp_info_t* ftp_info);

void print_ftp_info(ftp_info_t* ftp_info);

void free_ftp_info(ftp_info_t* ftp_info);

#endif /*__FTP_INFO__*/
```

## defs.h

```c
#ifndef __DEFS_H__
#define __DEFS_H__

#define OK 0
```

```
#define BAD_ARGS 1
#define GETADDRINFO_ERROR 2
#define INVALID_URL_FORMAT 3
#define INVALID_CHAR 4
#define OPEN_SOCKET_ERROR 5
#define CONNECT_SOCKET_ERROR 6

#endif /*__DEFS_H__*/
```