Rede Informática

Introdução Engenharia Informática

Mário Antunes

November 10, 2025

Exercícios

Passo 0: Configuração

Antes de começar, vamos configurar o seu sistema com todas as ferramentas necessárias para estes exercícios.

1. Ferramentas de Sistema e Python

Primeiro, atualize as suas listas de pacotes e instale os utilitários principais: `curl` e `wget` para testar serviços web, e o gestor de pacotes do Python (`pip`) e o módulo de ambientes virtuais (`venv`).

```
# 1. Atualize as suas listas de pacotes
sudo apt update; sudo apt full-upgrade -y; \
sudo apt autoremove -y; sudo apt autoclean

# 2. Instale ferramentas gerais, flatpak, e essenciais do Python
sudo apt install -y udisks2 curl wget \
flatpak python3-pip python3-venv

# 3. Adicione o repositório Flathub
flatpak --user remote-add --if-not-exists \
flathub https://flathub.org/repo/flathub.flatpakrepo
```

2. IDE Thonny (para o Exercício 5)

Thonny é um IDE simples para MicroPython. Vamos instalá-lo usando Flatpak para obter a versão mais recente.

```
# 1. Instalar o Thonny
flatpak --user install flathub org.thonny.Thonny

# 2. Adicionar utilizador ao grupo dialout
sudo usermod -a -G dialout $USER
```

O grupo `dialout` fornece acesso total e direto às portas série. Membros deste grupo podem ligar-se a dispositivos série (através de conexões série ou USB).

Pode então executar o Thonny a partir do menu de aplicações ou com `flatpak run org.thonny.Thonny`.

Para utilizadores de Linux nativo, não são necessários mais passos. No entanto, para WSL e SOs virtualizados, são necessários alguns passos adicionais. Verifique as Secções 5.1 e 5.2 respetivamente.

3. 🐍 Boas Práticas de Python

Para cada exercício de Python, por favor, siga estes passos:

1. Crie um novo diretório para o projeto (ex: `mkdir ex01 && cd ex01`).
2. Crie um ambiente virtual isolado:

```
python3 -m venv venv
```

3. Ative o ambiente:

```
source venv/bin/activate
```

4. Crie um ficheiro `requirements.txt` (conforme especificado em cada exercício) e instale a partir dele:

```
pip install -r requirements.txt
```

5. **Use o módulo** `logging` em vez de `print()` para todas as suas mensagens de estado.

```
import logging
logging.basicConfig(level=logging.INFO, format='%(message)s')
logger = logging.GoogletLogger(__name__)

logger.info("Esta é uma mensagem de informação.")
```

4. Arquitetura de Rede

Tipicamente, usará a rede Eduroam para aceder à internet durante as aulas. Para a maioria das atividades, isto é suficiente; no entanto, esta rede (gerida pela universidade) bloqueia a comunicação entre **equipamentos** dos estudantes.

Como tal, fornecemos uma rede sem fios separada chamada `TheOffice` que pode ser usada para ligar aplicações de utilizadores **entre si**. Isto é **opcional** (mas recomendado) para os **exercícios** 1-4, mas **obrigatório** para o exercício 5.

📶 Detalhes da Rede Wi-Fi:

SSID (Nome da Rede)	Palavra-passe
`TheOffice`	`8006002030`
`TheOffice5G`	`8006002030`

{ width=50% }

Exercício 1: Transferência de Ficheiros por UDP

Objetivo: Explorar o script `file_transfer.py` fornecido. Perceber como ele usa `asyncio` para criar um servidor persistente que pode lidar com múltiplos uploads de ficheiros de clientes.

Detalhes:

- **Servidor:** O servidor é persistente. Usa um `dict` para gerir transferências de ficheiros de diferentes clientes, usando como chave o seu IP e porta (`addr`).
- **Cliente:** O cliente envia primeiro os metadados do ficheiro (nome, tamanho), depois envia os pedaços (chunks) de dados, mostrando uma barra de progresso com `tqdm`.
- **Protocolo:** O script usa um protocolo simples baseado em nova linha (newline):
  - `START:<total_chunks>:<total_size>:<filename>`
  - `DATA:<chunk_num>:<data_chunk>`
  - `END`
- O servidor responde com `ACK_ALL` ou `ACK_FAIL`.

Instruções:

1. Crie um novo diretório `ex01` e entre nele `cd ex01`.
2. Descarregue o código da solução para este diretório.
3. Ative um `venv` e instale os requisitos:

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

4. Crie um ficheiro para enviar, ex: `echo "Este é um ficheiro de teste UDP." > test.txt`.

5. **Execute o Servidor (Terminal 1):**

   ```
   python file_transfer.py receive --port 9999
   ```

6. **Execute o Cliente (Terminal 2):**

   ```
   python file_transfer.py send test.txt --host 127.0.0.1 --port 9999
   ```

Exercício 2: Jogo do Galo Remoto

Objetivo: Analisar o script `main.py` fornecido para ver como o `asyncio` pode ser integrado com uma biblioteca gráfica (GUI) como o Pygame para criar uma aplicação de rede.

Detalhes:

- **Menus GUI:** O script usa Pygame para desenhar todos os seus próprios menus. Não usa `argparse`.
- **Loop de Jogo Async:** O loop principal `while running:` é `async`. Ele cede o controlo ao event loop do `asyncio` ao chamar `await asyncio.sleep(1/FPS)`.
- **Rede:** O script usa `asyncio.start_server` (para o anfitrião/host) e `asyncio.open_connection` (para o cliente) para criar streams TCP fiáveis.
- **Tratamento de Erros:** As funções `run()` e `close_connection()` usam `try ... finally` e tratam `asyncio.CancelledError` para garantir que a aplicação encerra de forma limpa.

Instruções:

1. Crie um novo diretório `ex02` e entre nele `cd ex02`.
2. Descarregue o código da solução para este diretório.
3. Ative um `venv` e instale os requisitos:

   ```
   python3 -m venv venv
   source venv/bin/activate
   pip install -r requirements.txt
   ```

4. **Execute o Anfitrião (Host - Jogador X):**

   ```
   python main.py
   ```

   - Na GUI, clique em "Host Game" -> insira uma porta (ex: 8888) -> Pressione Enter.

5. **Execute o Cliente (Jogador O):**

   ```
   python main.py
   ```

   - Na GUI, clique em "Join Game" -> insira o IP do anfitrião (`127.0.0.1` se for na mesma máquina) -> Pressione Enter -> insira a porta (8888) -> Pressione Enter.

Exercício 3: Serviço de Cache com FastAPI

Objetivo: Executar e testar o script `main.py` fornecido para entender como construir um endpoint de API de alta performance com cache.

Detalhes:

- **Endpoint:** O script fornece um endpoint `GET /ip/{ip_address}`.
- **Cache:** Usa um ficheiro local `ip_cache.json`.
- **Lógica:** Verifica o `timestamp` de uma entrada em cache contra um `CACHE_DEADLINE_SECONDS`.
- **API Externa:** Se a cache estiver desatualizada (stale) ou em falta, usa a biblioteca `requests` para obter dados ao vivo.

Instruções:

1. Crie um novo diretório `ex03` e entre nele `cd ex03`.
2. Descarregue o código da solução para este diretório.
3. Ative um `venv` e instale os requisitos:

   ```
   python3 -m venv venv
   source venv/bin/activate
   pip install -r requirements.txt
   ```

4. **Execute o Servidor:**

   ```
   uvicorn main:app --reload
   ```

5. **Teste o Serviço (num novo terminal):**

   - **Teste 1 (Falha na Cache - Miss):**

     ```
     # IP Privado (tem de falhar)
     curl [http://127.0.0.1:8000/ip/192.168.132.132](http://127.0.0.1:8000/ip/192.168.

     # DNS Google
     curl [http://127.0.0.1:8000/ip/8.8.8.8](http://127.0.0.1:8000/ip/8.8.8.8)

     # IP Público da MEO
     curl [http://127.0.0.1:8000/ip/144.64.3.83](http://127.0.0.1:8000/ip/144.64.3.83)

     # UA
     curl [http://127.0.0.1:8000/ip/193.137.169.135](http://127.0.0.1:8000/ip/193.137.

     # IP Estático de São Tomé
     curl [http://127.0.0.1:8000/ip/197.159.166.30](http://127.0.0.1:8000/ip/197.159.1
     ```

     (Verifique os logs do servidor; deve dizer "Querying external API".)

   - **Teste 2 (Sucesso na Cache - Hit):**

     ```
     # IP Privado (tem de falhar)
     curl [http://127.0.0.1:8000/ip/192.168.132.132](http://127.0.0.1:8000/ip/192.168.

     # DNS Google
     curl [http://127.0.0.1:8000/ip/8.8.8.8](http://127.0.0.1:8000/ip/8.8.8.8)

     # IP Público da MEO
     curl [http://127.0.0.1:8000/ip/144.64.3.83](http://127.0.0.1:8000/ip/144.64.3.83)

     # UA
     curl [http://127.0.0.1:8000/ip/193.137.169.135](http://127.0.0.1:8000/ip/193.137.

     # IP Estático de São Tomé
     curl [http://127.0.0.1:8000/ip/197.159.166.30](http://127.0.0.1:8000/ip/197.159.1
     ```

     (Verifique os logs do servidor; deve dizer "Returning cached data".)

Exercício 4: Chat Pub/Sub

Objetivo: Usar Docker para executar um broker MQTT e ligar-se a ele com um cliente puramente JavaScript para criar uma aplicação de chat "serverless".

Detalhes:

- **Sem Servidor Python:** Você não vai escrever *nenhum* código de servidor. O broker Mosquitto *é* o servidor.
- **Broker:** O ficheiro `docker-compose.yml` inicia o Mosquitto e carrega o `mosquitto.conf`.
- **Configuração:** O ficheiro `.conf` ativa o acesso anónimo e abre a porta `9001` para **MQTT-sobre-WebSockets**.
- **Cliente:** O ficheiro `chat_client.html` usa a biblioteca **MQTT.js** (carregada de um CDN) para se ligar a `ws://localhost:9001`. Implementa um chat Pub/Sub.

Instruções:

1. Crie um novo diretório `ex04` e entre nele `cd ex04`.
2. Descarregue o código para o mesmo diretório.
3. **Inicie o Broker:**

   ```
   docker-compose up -d
   ```

4. **Teste o Cliente:**
    - Abra `http://localhost:8080/` no seu navegador web.
    - Abra `http://localhost:8080/` num *segundo* separador ou janela do navegador.
    - Insira nomes de utilizador diferentes e ligue-se. As mensagens enviadas numa janela devem aparecer na outra.
    - Pode usar a rede `TheOffice` para conversar com outros estudantes.

Exercício 5: Sensor MQTT com RPi Pico

Como afirmado no início, utilizadores de Linux nativo podem saltar estes passos (saltar para a Secção 5.3. Para WSL e SOs virtualizados, siga os passos abaixo. **Importante:** precisa de desligar a Firewall para este exercício.

5.1 Passagem (Passthrough) de USB no WSL

Os passos nesta secção são baseados no guia original da Microsoft.

Num terminal **PowerShell**, execute os seguintes comandos:

```
# 1. Atualize a versão do WSL
wsl --update

# 2. Desligue a VM leve do WSL
wsl --shutdown

# 3. Atualize as opções de rede do WSL
$wslConfig = @'
[wsl2]
networkingMode=mirrored
'@

Add-Content -Path $env:UserProfile.wslconfig -Value $wslConfig

# 4. Instale a aplicação USBIPD
winget install --interactive --exact dorssel.usbipd-win
```

Após estes passos, pode reiniciar a VM leve do WSL. Basta abrir o terminal correspondente.

Para anexar um dispositivo USB à VM leve do WSL, use as seguintes instruções num terminal PowerShell com privilégios de administração. **Lembre-se** de ter o terminal WSL já a correr.

```
# 1. Liste os dispositivos USB
usbipd list

# 2. Encontre um com um nome semelhante a "USB Serial Device (COM4)"
# E faça o bind usando o seu BUSID (exemplo 2-7)
usbipd bind --force --busid <BUSID>

# 3. Anexe-o à VM leve do WSL
usbipd attach --wsl --busid <BUSID>
```

O dispositivo deve agora estar disponível na VM leve do WSL. Após completar o exercício, por favor, execute o seguinte comando para desanexar o dispositivo.

```
usbipd detach --busid <BUSID>
```

5.2 Passagem (Passthrough) de USB no VirtualBox

Para convidados (guests) Debian (ou outro Linux) a correr no VirtualBox, precisa de configurar o VirtualBox na sua **máquina anfitriã (host)** para "passar" o dispositivo USB diretamente para a **VM convidada (guest)**. Estes passos são realizados na **máquina anfitriã (host)** (o computador que corre o VirtualBox).

1. Instalar o VirtualBox Extension Pack (No Anfitrião) Isto é **obrigatório** para suporte USB 2.0 e 3.0, que a maioria dos dispositivos série modernos usa.

1. Vá à página de downloads do VirtualBox.
2. Encontre o **VirtualBox Extension Pack** e descarregue-o.
3. **Garanta que a versão do Extension Pack corresponde à sua versão instalada do VirtualBox.**
4. Faça duplo clique no ficheiro descarregado (`.vbox-extpack`) e siga as instruções no gestor do VirtualBox para o instalar.

2. Adicionar Utilizador Anfitrião (Host) ao Grupo `vboxusers` (Em Anfitriões Linux/macOS) Em **máquinas anfitriãs (host) Linux ou macOS**, a sua conta de utilizador deve estar no grupo `vboxusers` para dar permissão ao VirtualBox para aceder ao hardware USB.

```
# Este comando é para anfitriões Linux
sudo usermod -a -G vboxusers $USER

# No macOS, o instalador do Extension Pack deve tratar disto.
```

> **Importante:** Após executar este comando, **tem de fazer logout completo e login novamente** na sua máquina anfitriã para que a alteração do grupo tenha efeito. (Este passo não é necessário se a sua máquina anfitriã for Windows).

3. Configurar Definições USB da VM (No Anfitrião)

1. **Desligue** a sua VM Debian completamente (não faça apenas "Save State").
2. Abra o gestor do VirtualBox, selecione a sua VM Debian, e clique em **Settings (Definições)**.
3. Vá ao separador **USB**.
4. Selecione o **Controlador USB 3.0 (xHCI)**.
5. **Ligue o seu dispositivo MicroPython** (ex: Raspberry Pi Pico, ESP32) ao seu computador anfitrião.
6. Clique no ícone **"Adicionar novo filtro USB"** (o pequeno conector USB com um + verde).
7. Selecione o seu dispositivo da lista. Pode chamar-se "USB Serial Device", "CP210x", "CH340", "Raspberry Pi Pico", ou similar.
   - Isto cria um filtro que passará automaticamente este dispositivo *específico* para a sua VM quando for ligado.
8. Clique **OK** para guardar as definições.

4. Anexar e Verificar (Na VM Convidada)

1. **Inicie** a sua VM Debian.
2. Se o filtro foi configurado corretamente, o dispositivo deve ser automaticamente capturado pela VM convidada.
3. Abra um terminal *dentro da VM Debian*.
4. Primeiro, re-confirme que o seu utilizador está no grupo `dialout` (do **Passo 2** principal deste guia).
5. A seguir, verifique se o dispositivo está presente:

   ```
   ls /dev/tty*
   ```

Deverá ver um novo dispositivo, tipicamente chamado `/dev/ttyACM0` (para Picos) ou `/dev/ttyUSB0` (para placas baseadas em ESP). O Thonny será agora capaz de encontrar e ligar-se a esta porta.

5.3 Construir o sensor

Neste exercício, vamos explorar o RPI Pico W com um sensor de temperatura e humidade DHT11. Antes de montar o circuito, dedique algum tempo a verificar o pinout tanto da placa como do sensor.

{ width=100% }

{ width=45% }

O diagrama de ligações para o circuito é apresentado na figura abaixo.

{ width=65% }

5.4 Implementação (Deployment) do Código

Objetivo: Implementar o código MicroPython fornecido num Raspberry Pi Pico W para publicar a sua temperatura interna no seu broker MQTT.

Detalhes:

- **Hardware:** Este exercício requer um **Raspberry Pi Pico W**.
- **Sensor:** O código usa o sensor de temperatura interno incorporado no Pico, por isso **não é necessário hardware externo**.
- **Segredos:** A boa prática é armazenar as credenciais de WiFi num ficheiro `config.py` separado, que não é submetido para o controlo de versões.
- **Biblioteca MQTT:** O MicroPython requer uma biblioteca MQTT leve especial, `umqtt.simple`.

Instruções:

1. Crie um novo diretório chamado `ex05` e entre nele:

```
mkdir ex05 && cd ex05
```

2. Descarregue o código para o mesmo diretório.

3. **Inicie o Broker:**

```
docker-compose up -d
```

4. **Use o Thonny**

   - Abra o Thonny.
   - Ligue-se ao seu Pico (clique no menu do interpretador no canto inferior direito e selecione "MicroPython (Raspberry Pi Pico)").

5. **Execute o código**

   - Abra o `main.py` fornecido no editor.
   - Edite o endereço IP do broker MQTT (`mqtt_host`) e o ID do cliente (`mqtt_host`). Estas devem ser as únicas alterações necessárias.
   - Execute o script (clique no botão "Run") para executar o código na placa.

6. **Abra a Página Web**

   - Abra `http://localhost:8080/` no seu navegador web.
   - Preencha o IP do broker (se estiver a correr na mesma máquina, use `localhost`) e o tópico (padrão: `deti/pico/dht11`).
   - Observe o gráfico a mostrar informação em tempo real.

7. **MQTT5 Explorer**

   - O MQTT5 Explorer pode ser usado para depurar (debug) a conexão MQTT.
   - Instale usando o comando:

```
flatpak --user install flathub io.github.Omniaevo.mqtt5-explorer
```

   - Abra a aplicação e preencha a informação pedida.

🌟 Exercício Bónus: O Clássico Servidor de Eco (Echo Server)

Objetivo: Escrever um Servidor de Eco (Echo Server) simples em Python usando o módulo `socket` incorporado. Este é o "Olá, Mundo!" da programação em rede.

Tarefa: Este é o único exercício onde **tem de escrever o código você mesmo.**

Crie um único script Python `echo_server.py`. O script deve ser capaz de correr num de dois modos usando `argparse`:

1. `python echo_server.py tcp --port <num>`
2. `python echo_server.py udp --port <num>`

Requisitos:

- **Modo TCP:** O servidor deve escutar na porta indicada, aceitar uma conexão de cliente, e `recv` (receber) dados do cliente. Deve então `sendall` (enviar tudo) os *exatos mesmos dados* de volta. Deve lidar graciosamente com clientes que se desligam.
- **Modo UDP:** O servidor deve fazer `bind` à porta indicada, `recvfrom` (receber de) um datagrama, e `sendto` (enviar para) os *exatos mesmos dados* de volta para o endereço de onde vieram.
- Tem de escrever este código de raiz. **Não use** `asyncio` **para este exercício.**
- Teste o seu servidor TCP com `netcat`: `nc 127.0.0.1 <porta>`.
- Teste o seu servidor UDP com `netcat`: `nc -u 127.0.0.1 <porta>`.

Documentação Útil:

- **Módulo** `socket` **do Python:** https://docs.python.org/3/library/socket.html
- **Guia HOWTO de Programação de Sockets em Python:** https://docs.python.org/3/howto/sockets.html