Network programing

Introdução Engenharia Informática

Mário Antunes

November 10, 2025

Exercises

Step 0: Setup

Before you begin, let's set up your system with all the necessary tools for these exercises.

1. System Tools and Python

First, update your package lists and install the core utilities: `curl` and `wget` for testing web services, and Python's package manager (`pip`) and virtual environment module (`venv`).

```
# Update your package lists
sudo apt update; sudo apt full-upgrade -y; sudo apt autoremove -y; sudo apt autoclean

# Install general tools and Python essentials
sudo apt install -y curl wget python3-pip python3-venv
```

2. Thonny IDE (for Exercise 5)

Thonny is a simple IDE for MicroPython. We'll install it using Flatpak to get the latest version.

```
# 1. Install Flatpak
sudo apt install -y flatpak

# 2. Add the Flathub repository
flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo

# 3. Install Thonny
flatpak --user install flathub org.thonny.Thonny
```

You can then run Thonny from your application menu or with `flatpak run org.thonny.Thonny`.

3. 🐍 Python Best Practices

For each Python exercise, please follow these steps:

1. Create a new directory for the project (e.g., `mkdir ex01 && cd ex01`).
2. Create an isolated virtual environment:

   ```
   python3 -m venv venv
   ```
3. Activate the environment:

   ```
   source venv/bin/activate
   ```
4. Create a `requirements.txt` file (as specified in each exercise) and install from it:

   ```
   pip install -r requirements.txt
   ```
5. **Use the `logging` module** instead of `print()` for all your status messages.

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

logger.info("This is an info message.")
```

Exercise 1: UDP File Transfer

Goal: Explore the provided `file_transfer.py` script. Understand how it uses `asyncio` to create a persistent server that can handle multiple file uploads from clients.

Details:

- **Server:** The server is persistent. It uses a `dict` to manage file transfers from different clients, keyed by their IP and port (`addr`).
- **Client:** The client sends the file metadata (filename, size) first, then sends the data chunks, showing a progress bar using `tqdm`.
- **Protocol:** The script uses a simple newline-based protocol:
  - `START:<total_chunks>:<total_size>:<filename>`
  - `DATA:<chunk_num>:<data_chunk>`
  - `END`
  - The server responds with `ACK_ALL` or `ACK_FAIL`.

Instructions:

1. Create a new directory `ex01`.
2. Download the solution code into this directory.
3. Activate a `venv` and install the requirements:

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

4. Create a file to send, e.g., `echo "This is a UDP test file." > test.txt`.
5. **Run the Server (Terminal 1):**

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

6. **Run the Client (Terminal 2):**

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

Exercise 2: Remote Tic-Tac-Toe

Goal: Analyze the provided `main.py` script to see how `asyncio` can be integrated with a GUI library like Pygame to create a networked application.

Details:

- **GUI Menus:** The script uses Pygame to draw all its own menus. It does not use `argparse`.
- **Async Game Loop:** The main `while running:` loop is `async`. It yields control to the `asyncio` event loop by calling `await asyncio.sleep(1/FPS)`.
- **Networking:** The script uses `asyncio.start_server` (for the host) and `asyncio.open_connection` (for the client) to create reliable TCP streams.
- **Error Handling:** The `run()` and `close_connection()` functions use `try ... finally` and handle `asyncio.CancelledError` to ensure the application shuts down cleanly.

Instructions:

1. Create a new directory `ex02`.
2. Download the solution code into this directory. (Don't forget the `assets` folder if you have one).

3. Activate a `venv` and install the requirements:

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

4. **Run the Host (Player X):**

```
python main.py
```

   - In the GUI, click "Host Game" -> enter a port (e.g., 8888) -> Press Enter.

5. **Run the Client (Player O):**

```
python main.py
```

   - In the GUI, click "Join Game" -> enter the host's IP (`127.0.0.1` if on the same machine) -> Press Enter -> enter the port (8888) -> Press Enter.

Exercise 3: FastAPI Caching Service

Goal: Run and test the provided `main.py` script to understand how to build a high-performance, caching API endpoint.

Details:

- **Endpoint:** The script provides a `GET /ip/{ip_address}` endpoint.
- **Cache:** It uses a local `ip_cache.json` file.
- **Logic:** It checks the `timestamp` of a cached entry against a `CACHE_DEADLINE_SECONDS`.
- **External API:** If the cache is stale or missing, it uses the `requests` library to fetch live data.

Instructions:

1. Create a new directory `ex03`.
2. Download the solution code into this directory.
3. Activate a `venv` and install the requirements:

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

4. **Run the Server:**

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

5. **Test the Service (in a new terminal):**

   - **Test 1 (Cache Miss):**

     ```
     curl http://127.0.0.1:8000/ip/8.8.8.8
     ```

     (Check the server logs; it should say "Querying external API".)

   - **Test 2 (Cache Hit):**

     ```
     curl http://127.0.0.1:8000/ip/8.8.8.8
     ```

     (Check the server logs; it should say "Returning cached data".)

Exercise 4: Pub/Sub Chat

Goal: Use Docker to run an MQTT broker and connect to it with a pure JavaScript client to create a "serverless" chat application.

Details:

- **No Python Server:** You will not write *any* server code. The Mosquitto broker *is* the server.
- **Broker:** The `docker-compose.yml` file starts Mosquitto and loads `mosquitto.conf`.

- **Configuration:** The `.conf` file enables anonymous access and opens port `9001` for **MQTT-over-WebSockets**.
- **Client:** The `chat_client.html` file uses the **MQTT.js** library (loaded from a CDN) to connect to `ws://localhost:9001`. It implements a Pub/Sub chat.

Instructions:

1. Create a new directory `ex04`.
2. Download the code solution file into the same directory.
3. **Start the Broker:**

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

4. **Test the Client:**
   - Open `index.html` in your web browser.
   - Open `index.html` in a *second* browser tab or window.
   - Enter different usernames and connect. Messages sent in one window should appear in the other.

Exercise 5: RPi Pico MQTT Sensor

Goal: Deploy the provided MicroPython code to a Raspberry Pi Pico W to publish its internal temperature to your MQTT broker.

Details:

- **Hardware:** This exercise requires a **Raspberry Pi Pico W**.
- **Sensor:** The code uses the Pico's built-in internal temperature sensor, so **no external hardware is needed**.
- **Secrets:** Best practice is to store WiFi credentials in a separate `config.py` file, which is not checked into version control.
- **MQTT Library:** MicroPython requires a special lightweight MQTT library, `umqtt.simple`.

Instructions:

1. **Flash MicroPython:**
   - Hold the "BOOTSEL" button on your Pico while plugging it in.
   - It will mount as a USB drive.
   - Download the latest "Pico W" UF2 file from the MicroPython website and drag it onto the USB drive. The Pico will reboot.
2. **Use Thonny:**
   - Open Thonny.
   - Connect to your Pico (click the bottom-right interpreter menu and select "MicroPython (Raspberry Pi Pico)").
   - In Thonny, go to **Tools -> Manage Packages**.
   - Search for `micropython-umqtt.simple` and install it.
3. **Create the Files:**
   - In Thonny, create a new file named `config.py`. Copy the code from the solution and **fill in your WiFi credentials**. Save it to the **MicroPython device (the Pico)**.
   - Create a new file named `main.py`. Copy the `pico_mqtt.py` solution code into it. **Change `MQTT_BROKER` to your computer's IP address** (not `localhost`).
   - Save this file as `main.py` on the **MicroPython device**.
4. **Run:**
   - Press the "Run" button in Thonny.
   - Watch the Thonny Shell. It should connect to your WiFi, then to your MQTT broker, and start sending temperature data.

🌟 Bonus Exercise: The Classic Echo Server

Goal: Write a simple Echo Server in Python using the built-in `socket` module. This is the "Hello, World!" of network programming.

Task: This is the only exercise where you **must write the code yourself.**

Create a single Python script `echo_server.py`. The script should be able to run in one of two modes using `argparse`:

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

Requirements:

- **TCP Mode:** The server must listen on the given port, accept a client connection, and `recv` data from the client. It must then `sendall` the *exact same data* back. It must handle clients disconnecting gracefully.
- **UDP Mode:** The server must bind to the given port, `recvfrom` a datagram, and `sendto` the *exact same data* back to the address it came from.
- You must write this code from scratch. **Do not use** `asyncio` **for this exercise.**
- Test your TCP server with `netcat`: `nc 127.0.0.1 <port>`.
- Test your UDP server with `netcat`: `nc -u 127.0.0.1 <port>`.

Helpful Documentation:

- **Python** `socket` **Module:** https://docs.python.org/3/library/socket.html
- **Python Socket Programming HOWTO Guide:** https://docs.python.org/3/howto/sockets.html