Instituto Politécnico de Coimbra

Instituto Superior de Engenharia de Coimbra

Departamento de Engenharia Informática e Sistemas

# *Taxi service platform*

**Tymofii Drohin 2022161633**

## 1 Introduction

**This practical work focuses on the simulation of a taxi management platform, which consists of a fleet of autonomous vehicles operating in a Unix environment. The system is structured across three separate applications:**

- **Controller: responsible for managing the fleet and allocating services.**
- **Client: provides the user interface for scheduling trips and interacting with the system.**
- **Vehicle: simulates the execution of transportation tasks and sends telemetry data.**

**Communication between processes primarily uses named pipes for client interactions. Vehicle management by the controller is handled through command-line arguments and stdout redirection (anonymous pipes). Additionally, the system employs signals (`SIGUSR1` and `SIGUSR2`) to start or cancel services and dynamically creates processes to represent each active vehicle.**

**Implemented features include user authentication, request scheduling, vehicle management, administrative interactions, and trip monitoring in simulated time. The project is implemented using only standard C libraries, without relying on any external libraries.**

## 2 Architecture

**The taxi management platform is designed with a modular, process-oriented architecture to simulate a fleet of autonomous vehicles. The system is divided into three main components—Controller, Client, and Vehicle—each implemented as an independent process. This separation ensures clear responsibilities, efficient communication, and dynamic scaling of the vehicle fleet.**

### 2.1 Controller

**The Controller is the central coordinator of the system. Its responsibilities include:**

- **Managing the fleet of vehicles and their states.**
- **Allocating trip requests to available vehicles.**
- **Handling administrative commands and logging system events.**

**The Controller dynamically spawns a new Vehicle process for each active service. It communicates with vehicles via command-line arguments for initialization and stdout redirection (anonymous pipes) for telemetry and status updates. Signals (SIGUSR1 and SIGUSR2) control vehicle behavior, such as starting or cancelling a service.**

## 2.2 Client

**The Client acts as a command interface to the system. Its responsibilities are limited to:**

- **Validating user commands.**
- **Forwarding commands to the Controller through named pipes.**

**The Client does not manage trips or vehicles directly; it serves as a lightweight intermediary, ensuring that only well-formed requests reach the Controller. This design simplifies client logic and centralizes system control in the Controller process.**

## 2.3 Vehicle

**Each Vehicle process simulates the execution of transportation tasks. Responsibilities include:**

- **Updating trip status and telemetry data.**
- **Responding to control signals from the Controller.**
- **Reporting task completion or cancellation.**

**Vehicles operate in simulated time and provide continuous feedback to the Controller through anonymous pipes, allowing real-time monitoring and dynamic fleet management.**

## 2.4 Inter-Process Communication

**The architecture relies on two main communication mechanisms:**

1. **Named Pipes (FIFOs): Used for Client-to-Controller interactions, allowing commands to be transmitted and validated asynchronously.**

2. **Anonymous Pipes and Signals: Used for Controller-to-Vehicle communication, enabling real-time control of vehicle processes and collection of telemetry data.**

**This architecture ensures modularity, centralized control, and dynamic coordination of autonomous vehicles while remaining fully compliant with standard C libraries and Unix process management.**

# 3 Inner workings

### 3.1 Controller

Controller has 3 threads. Main thread, threads for listening to user connection requests, deditated timer thread and the controller worker thread, that is managing all the automatic logic, and a timer thread.

It holds all the relative information about users, requests and vehicles in the arrays, using dedicated structs. Controller is responsible for managing pipes, communicating with different clients at the same time, ensures the integrity of the data.

Controller makes heavy use of the basic thread-safe data structure imlpementations, as well in some cases the conditional variables.

### 3.2 Client

Client application acts as a lightway shell for the user to communicate with controller.

It is responsible for creating the client end of the named pipe, as well as validating commands. It has 2 threads - main thread with the command line logic and a listener thread for incoming messages. Messages are stored in a thread-safe queue, to insure the correct order and no data loss.

## 3.2 Vehicle

Vehicle is a simple application that has two signals handler. It sends status messages to

the

stdout that is getting redirected, as well as messages to client to its named pipe.