Train Control (Part 2)

Niclas Heun
nheun@uwaterloo.ca
21111245

Trevor J. Yao
t27yao@uwaterloo.ca
20830290

Professor Ken Salem
CS 452/652: Real-Time Programming (Fall 2023)
University of Waterloo
22 November 2023

Contents

List of Figures

1 Overview

This assignment finishes our implementation of the complex real-time problem of train control, building upon the work done in Part 1. In particular, this document assumes familiarity (or at the very least, easy access to the documentation of TC1) with the decision and implementation decisions made in TC1. When necessary, these choices may be repeated.

In order to accurately route and stop trains on arbitrary positions on the track, our train control program runs atop the kernel which we have built over the last month. The user interface is built upon the K4 UI, adding a few additional interactive commands to route trains between arbitrary positions on the track. All the commands to interact with the track are as follows:

- `tr <train number> <train speed>`: Set the speed of the specified train. The train number must be a valid supported train (i.e. listed in the interface), and the speed must be an integer in the range $[0, 14] \cup [16, 30]$, where the latter is used to toggle the light status.
- `rv <train number>`: Change the direction of the specified train to reverse. When run, the train will stop, before reversing at the same speed as before. Preserves the status of headlights.
- `sw <switch number> <switch direction>`: Throw the given switch in the given direction (`s`/`S` or `c`/`C`). The switch number must exist on the track.
- `init`: Initialises the track by learning the positions of any of the supported trains on the track. Must be called before the `tc` command.
- `tc <end> <offset> <train number> <speed>`: Route the specified train from the start to end track nodes, stopping offset mm after (if positive) or before (if negative) at the specified speed. The track nodes must match their names in the track data (i.e. case sensitive). The speed is specified as one of `lo`/`med`. Note that the start node must be in the forward direction of the train.
- `st <train number>`: Stops the train mid-route. Useful for force quitting.
- `go`: Send the 'Go' signal to the track.
- `hlt`: Send the 'Stop' signal to the track.
- `q`: Halt the program and return to boot loader. This will send any remaining commands to the track before stopping control.

1.1 Submission SHA

The repository for the kernel is ist-git@git.uwaterloo.ca:t27yao/cs452-kernel.git. The submission SHA is ⟨SHA⟩.

1.2 Program Structure

The program (including the kernel) is structured into three sections. The `kernel/` directory contains all the kernel code, including task descriptors, syscall/interrupt handlers, context switching functions, program execution timers, and allocators for task descriptors and stacks.

The root directory `include/` and `src/` directories contain system library code, which is used by user-level programs to interact with the kernel, and also any shared data structures between the kernel and user programs. These include interfaces for using any kernel-provided functionalities (i.e. syscalls, interrupt events, message passing). The interfaces for interacting with the Raspberry Pis, as well as assorted utility functions (string/memory interactions) are also found in the system library, and are used by both the kernel and user-level programs. The system library directory also holds the implementation of user-level servers, including the Name Server, Clock Server, and two I/O Servers, as well as their respective APIs. Finally, the system library also contains the deque (double-ended queue) and binary heap data structures.

All the user-level code needed to implement train control is contained in the `track-control/` directory.[1] The contents of this directory are all discussed in detail throughout this document.

2 Execution

The Train Control program can be built from the root directory by running `make` or `make all`. It may be necessary to set the `XDIR` variable to point to the directory containing the cross-compiler. For example, in the `linux.student.cs` environment, `XDIR` should be set to `/u/cs452/public/xdev` (i.e. the parent of the `bin` directory). Therefore, the program can be compiled in the `linux.student.cs` environment by changing directory to the program root and executing:

```
make XDIR=/u/cs452/public/xdev
```

[1]It was only realised when writing this document that this directory was indeed misnamed.

3 Routing

4 Sensor Attribution

5 Track Locking

6 Train Control Design

6.1 Position Modelling

6.2 Train Administrator