CS 452/652 Fall 2023 - Assignment 0

(Version 1)

Due Date: Tue Sep 19th, 8:30am

Introduction

In this assignment you will familiarize yourselves with the basic functionality of the Raspberry Pi, its ARM CPU, and the train controller, while learning techniques for handling asynchronous events by polling. It is recommended that you first write, compile, and execute several simple programs for the Raspberry Pi, and only then start working on the assignment.

Description

Write a program that runs on the Raspberry Pi system. Your program interacts with the user via the serial interface to a virtual terminal while controlling the following independent real-time activities:

1. a digital clock showing minutes, seconds, and tenths of seconds, which measures time accurately, in the sense that it does not slow down or lose ticks,
2. a user command interface that can be used to set train speeds and control switch turn-outs, and
3. a real-time display on the terminal showing the state of the track.

The monitor should use cursor addressing (using appropriate ASCII sequences for terminal control) to display on its screen at least the following:

1. the current time,
2. a table of switch positions,
3. a list of the most recently triggered sensors, and
4. a prompt at which the user can type commands.

These four components should be organized so that the display is easy to understand.

The user command interface should support, at a minimum, the following commands.

1. `tr <train number> <train speed>` – Set any train in motion at the desired speed (0 for stop).

2. `rv <train number>` – The train should reverse direction. To reverse direction set the train speed to zero, wait until the train stops, send the reverse direction command, then send a speed command to re-accelerate the train to the same speed in reverse as it was going forward before the command. Do not send the reverse direction command before the train has stopped!

3. `sw <switch number> <switch direction>` – Throw the given switch to straight (S) or curved (C). Make sure you can reliably communicate with the track and train hardware before attempting to throw switches.

IMPORTANT: A solenoid must be switched off (via Command 0x20) between 100 and 500 milliseconds after it was activated! Also, do not under any circumstances activate a switch over and over again. Doing so will burn out the solenoid!

4. `q` – halt the program and return to boot loader (reboot).

Use the Raspberry Pi's built-in system timer (BCM Chapter 10) to implement the clock. Do not use interrupts.

Program Structure

Your program should be written as a polling loop, using output channel 1 to talk to the terminal and channel 2 to talk to the train set. (This is the default configuration and you should always leave it so.) The clock should not lose time.

Possibly Helpful Comments

You can use the headlight on the trains as an easy method for telling if your commands to the trains are successful. Also, the headlights provide an indication of the current travelling direction of a train.

It should not be necessary to use the Märklin 6021 manual control box other than the 'stop' and 'go' buttons (stop+go = reset).

The Pi-equipped workstations in the main lab have multiple serial ports. You can run two gtkterm terminals on the workstations, with one set to the characteristics of the train interface. In that way you can separate a hard problem, 'Is the problem that the UART transmits incorrectly or that my train commands are incorrect?' into two simpler problems.

For this assignment only, do not use compiler optimization!

Submission

The assignment must be done completely individually. Hand in the following, nicely formatted and printed.

1. A description of the structure of your program. We will judge your program primarily on the basis of this description. Describe which algorithms and data structures are used and why you chose them. This description should include a list of unimplemented aspects of the assignments, if any. Also, if you know of bugs in your implementation describe them explicitly.
2. Answers to the following questions:
    i. How do you know that your clock does not miss updates or lose time?
    ii. How long does the train hardware take to reply to a sensor query? (Note: To answer these questions, you need to perform and report some timing of your polling loop.)
3. You should have a code repository on UW's git server git.uwaterloo.ca containing the source code of your assignment, instructions how to make the executable, and the PDFs containing the descriptions of your program and answers to the questions above.

    - The code for sensor timing measurements must be part of your submission, along with any special instructions to build, load, and run.
    - Code and documentation must remain unmodified after submission until the assignments have been marked.
    - Email the commit SHA to the instructor and TAs before the deadline. The repository must be set to 'Private' with TAs and instructor added as 'Reporter'.
4. Make sure (test!) that the following sequence of operations results in a successful build of your program in the `linux.student.cs` environment. Make it clear in `README.md` where to find the documentation and executable!

    ```
    git clone <your repo>
    git checkout <commit hash>
    make
    ```

Three examples of highly rated documentation from previous years:

- example 1
- example 2
- example 3