CS 452 TC1 and TC2

Authors

Sebastian Gaweda (20143503) and Shashwat Pratap (20638036)

Make and Run

checkout - git clone https://git.uwaterloo.ca/sgaweda/cs452-rtos-aka-trains.git
make - cd cs452-rtos-aka-trains/tc1/ && make
run - in gtk terminal run - load ARM/{user-name}/tc1.elf; go

Kernel Modifications

Context Switch Bug

There was an unfortunate bug in our K4 code which resulted in interrupts clobbering a0. Once discovered it was a simple fix which enabled communication on UART1. The following two points cover the remaining issues with UART1 communication.

Switch Atomicity

After Professor Karsten mentioned in class that switching should be atomic, we updated our code to ensure that when a switch statement is sent, we block all communication until it has completed. This was one part of UART1 communication that when fixed resolved some intermittent issues we were observing.

CTS

It turns out that our approach to handling CTS was not robust enough and a small bit of time was spent refactoring our TX1Interrupt logic to use the generic UART1Interrupt and to check the Modem Status register.

Since the last update for this kernel, three of the groups, including mine, have become convinced that we may have been encountering the CTS quirk. For at least two of these groups, properly adjusting our code to check for changes using the Modem Status register has fixed issues with bytes disappearing. Nathan, Lennox, and I hope that we will have some time to isolate this issue and provide a minimum kernel and program which only writes to UART1 so that we can minimize the amount of time between interrupt and checking the Modem Status register and maximize the likelihood that we encounter the CTS quirk.

More Servers

While we initially felt that our K4 design was strong enough, there were several reasons we decided to spend a significant time refactoring K4 to support more servers:

- To facilitate a TC2 architecture we were comfortable with, we needed servers which were responsible for state and calculations related to sensors, switches, and trains.
- Separation of command parsing, and internal command routing so that our navigation server can control trains using our message infrastructure instead of commands that need to be parsed.
- In order to ensure that all train communication occurs at a higher priority level, our sensor client needed to be a sensor server so that messages to the gui server could be sent at a lower priority.

This refactor was a significant engineering effort that took many days to complete, but the result is much simpler logic at the local level for most important tasks. For example, when switching and train speed commands were sent to the Marklin server from the same server, the logic for queuing commands was very complicated, but now that they're separate, it's much easier to manage. The only drawback we've encountered is that by having so many servers, we have to be careful about how we use couriers and there is some minor but noticeable overhead in creating couriers for every communication channel. We have also been very careful not to introduce any loops in the communication channel so that our system cannot enter a state where it is looping infinitely.

TC1 Progress

Routing and Switching

We implemented Dijkstra's for route finding. We chose to use Dijksta's to make it easier to compute paths in TC2. Given a starting position and an end position, we can compute the shortest path to the destination. At our TC1 demo, this did not support being able to reverse, but in the TC2 section of this document there is progress made here. We cannot yet control trains using our generated paths, but we have significant infrastructure in place to make this happen.

Stopping

We have calibrated velocity for train 58 and should be able to calibrate every other train's velocity fairly quickly. Our hope is to support dynamically updating these estimates, but admittedly this is a low priority tasks since most other teams have indicated that their estimates haven't really fluctuated much. Since we cannot control trains yet, we have not been able to validate that we are able to correctly stop at any given landmark and offset.

TC2 Progress

Route Finding

To support the requirements of TC2, our route finding was modified to support reversing at junctions. We also added support for removing landmarks from the track graph so that we can avoid train collisions. This feature enables the possibility of being able to remove faulty components from paths if one has been detected as faulty.

Sensor Attribution

Our TC2 correctly attributes sensors to the trains that trigger them and completely ignores all sensor reads which are not expected by any train. Within our system it should be relatively simple to extend this to support alternate paths so we can detect when a switch has failed to switch. Thanks to the modifications of our graph, it should be possible to dynamically remove the faulty sensor from future path calculations and generate a new path for the train that was incorrectly routed.

Collision Avoidance

Thanks to our Dijkstra's algorithm, we are able to avoid train collision by generating mutually exclusive paths. We hope that we can make this more robust by adding a reservation system that frees up sections of track as they are correctly detected by attributed sensors. There is also a hope to add a time dimension to the reservation system, but this is one of our lowest priority additions because we don't believe it will be trivial to ensure correctness and avoid collisions.

Robustness

Our sensor attribution makes our system robust to spurious sensor reads, but currently our system does not support detection and recovery from unexpected occurrences. Our goal is to create a system which prioritizes collision avoidance and recovery after detecting a collision may occur. We would also like to add some support for detecting component failure, and recovering from that as well.