CS 452 Kernel 3

Brian Forbes & Daniel McIntosh
20617899 & 20632796

July 14, 2018

1 Building, Running, and Usage

The repository for the code can be found at `https://git.uwaterloo.ca/baforbes/cs452-kernel/tree/tc2_brian_revroute` (which includes reversing routes but not reservations) and `https://git.uwaterloo.ca/baforbes/cs452-kernel/tree/tc2_reservations` (which includes reservations, but not reversing routes). To build the executable, run the following command from the root directory: `make` To build this document, run the following command from the root directory: `make docs` The executable will be built to `<repo_root>/bin/kernel.elf`

Once run, the program will initialize the kernel, and then create the first user task. This task will then create the tasks that run our train program, such as the name and clock servers, UART handlers, and the terminal, command, and track state servers.

2 Program Description

Note that each referenced file corresponds to a header file of the same name in the `include` folder (for example, `src/kernel.c` to `include/kernel.h`)

2.1 Track State Server

The track state server is in `track_state.c`. For TC1, the track state server handled all aspects of train control. This has now been split into the Track State and Train State servers. The track state server now handles only the state of the track - so the state of switches and sensors. When a new sensor is triggered, the server sends a notification to the Train State server. This server also handles route finding.

2.1.1 Route Finding

Routes are now expressed as a sequence of switches and associated actions. Actions can be either switching to curved, switching to straight, or reversing.

In the case of a reverse action, the switch is inferred to be a merge, instead of a switch.

When a route is requested from the track state server, it is given the start and end nodes, a bitfield representing unusable (due to reservations) nodes, the minimum distance of the route, and the distance penalty for reversing. Then, a variant of BFS is performed where possible paths are stored in a minheap. In this way, the first path found will necessarily be the shortest path. The impementation of the BFS is in `track.c`, as `find_path_between_nodes`.

2.2 Train State Server

The train state server is in `train_state.c`. The train state server now handles most of train control, including sensor attribution, reservations, and the execution of routes generated by the track state server.

2.2.1 Sensor Attribution

To attribute sensors to trains, when a sensor is triggered, the train state server attributes it to the train which has the shortest path to that sensor. In the future, we plan on adding a limit to how long a path we consider acceptable to prevent erroneous sensors from being falsely attributed to trains.

2.2.2 Route Handling

The train state server stores Active Routes for each train. These contain the route found by the Track State server, and various pieces of data about the train's current place in that route. Each time a sensor is triggered and attributed to a train, any steps of that route which are on track newly reserved by the train are executed. This way, switches are not changed until the train has reserved the track they are on.

As well, once the distance to the end of the route from the next predicted sensor is greater than the stopping distance, a delay is calcualted, and a task is dispatched to stop the train once that delay passes.

2.2.3 Reversing

To reverse a train, similar actions are taken to stopping. After the stop has completed, the train state server is notified. It sends the command to reverse the train, and then dispatches a task to re-accelerate the train after a very short delay.

2.2.4 Reservations

Once a navigate command has been given, a train begins to reserve track in front of it. It reserves track nodes 2 sensors + stopping distance + next switch ahead of itself. This guarantees that we have enough space to stop if we are unable to reserve more track, and a sensor doesn't fire as expected. This is also

the time at which we change switch states. Track reservations are passed to our `track_state` server, which takes these into account when generating a route for trains. At the moment, we don't free up track nodes, but plan to when we hit a sensor at least the length of the trains ahead of the node.

2.3 New Data Structures

No additional non-trivial data structures were added for tc2.

2.4 Changes to the Kernel

The first user task now initializes the track into a section of shared memory. There is no concerns about sharing this memory, as it is treated as read-only by all servers, and is therefore safe to be shared. To determine which track is currently in use, the mac address of the device is queried. For track A, this ends in `0xC5`, whereas for track B, this ends in `0xCC`. The other preceding characters are the same for both tracks. This method of determining current track was shared with the class by `c7zou`.

3 Known Limitations

1. We do not release reservations yet.
2. Short moves were not added to TC2, so reversing works badly with short distances.
3. Spurious sensors are not detected.
4. The train server does not detect when trains divert from the route they were intended to go on (for example, when a switch is in the incorrect position).