CS 452 – Train Control 2

March 16, 2017

Zhengkun Chen

Student ID: 20557054

Xiwen Liu

Student ID: 20468618

Operating Instructions

Full pathname to executable: /u/cs452/tftp/ARM/z283chen/kernel.elf

To load and run the program in the Redboot terminal, run the following command:

> load -b 0x00218000 -h 10.15.167.5 "ARM/z283chen/kernel.elf" ; go

The executable can also be created by going into the directory "kernel" and calling make:

> cd kernel

> make

And then copy the elf to wherever you want to run it.

Data Structures and Algorithms

Calibration

Data structures:

- The track structure from track_data.c, from the course website, was used to store the track's information.
  - Additional fields index and switch direction were added to the track structure to better reflect the state of the track dynamically.
- A 4 dimensional array train_velocity[train number][speed][start sensor][end sensor] stores the number of ticks needed for a train to travel from one sensor to the next, at a set speed.
- A 2-d array default_speed[train number][speed] stores the default velocity of a train, if there is no data in train_velocity.
- A 3-d array train_acc[train number][speed][type] stores a specific train's acceleration at different types of sensors. For example, E14/15 and E5/6 are considered the same "type" of sensor, since they lie on the same track orientation.
- These arrays offer constant time access, are easy to manage, and have very little overhead.

Algorithms:

- As a baseline, the velocity of the train at a number of different sensors was calculated by recording the average time difference of when a sensor and the following sensor was activated. During execution, the times are dynamically adjusted as the trains move, using a weighted average of the new time and the old time.
  - Entering the command "cal [train number] [speed]" will print the new velocities, which can then be imported into our program.
  - All time is measured in ticks, where 1 tick is 0.01 seconds.
  - All distances are measured in millimetres.

Path Finding

Data Structures:

- The track structure from track_data.c is used to represent the graph of the track. This structure is a linked list of nodes, where a node is a sensor, switch, or an end of the track.

Algorithms:

- A path is found with BFS, since the graph of the track is sparse. The path is the shortest path in terms of number of nodes. If switches need to be flipped in the path, the switches are flipped when the train is two sensors before the switch. If the train is not on the running on the path we found, the path finding algorithm will detect that and cancel the path finding to be safe.

Stopping

Data Structures:

- A stop wait queue keeps track of the amount of delay time and the sensor number that the train needs to stop at.

Algorithms:

- A worker task sends the stop command to the train when it is required. This way, the worker task does not block the calling task.
- To determine when the stop command should be sent to the train, the stopping distance of the train was measured. Once a path has been determined, the stopping distance is subtracted from the end point to determine the distance of the train from the preceding sensor when the stop command needs to be issued. Then the distance is converted into a delay time based on the experimental initial velocity at the preceding sensor.

Train Position

Data Structures:

- A train is a structure that contains its current speed, location, previous sensor activated, and its predicted path.
- A structure was chosen since we only run one train and it was the simplest approach. In the next part, the structure will be re-implemented as a server to better support multiple trains.

Algorithms:

- To find the train's predicted path, the latest sensor activated is used to traced the next three sensors in the path, using the track data graph.
- Path correction: If a set of "noisy" sensors actually forms a valid path (i.e. occurs in order), chances are the train is being repositioned to another location than we expected. We correct the path accordingly.
- We also use the expected time to next sensor to determine the exact location of the train while it is in between two sensors.

Short Moves

Data Structures:

- The model was developed by measuring the distance travelled vs the time interval between sending the start and stop command. Then a quadratic regression was done on the collection data to find a model for distance as a function of time.

Algorithms:

- To calculate the amount of time to wait between the start and stop commands, the root of the quadratic regression formula is found.
- A square root function fast_sqrt was implemented, which uses Newton's method to approximate the numerical value of the square root of a given number.

Error Detection

Data Structures:

- The train structure contains the previous sensor information needed to detect errors.

Algorithms:

- Malfunctioned sensors or invalid sensor input are ignored. Using a BFS, all sensors within 2 levels of the latest valid sensor are found. If a new sensor input is not one of the valid sensors for the train, it is ignored.
- If a sensor is valid, missed switches are detected by checking if a branch exists between the sensor and the previous sensor. A path from the previous sensor that has the same time as the predicted time is found, and if this path does not match the actual sensor, then a switch must be in the wrong position. Then the program is updated to match the track data.
- Missed sensors are detected by finding the path from the previous sensor to the current sensor, and checking if the path contains any other sensors.

Commands:

- stat [train number] [sensor name] [distance]
  - "**St**op **at**" stops the train at the specified sensor. The optional parameter "distance" tells the program how far after the sensor to stop in millimeters. For example, "stat 64 B6 20" will stop train 64 2 cm after sensor B6. The distance can be excluded from the command to stop the train exactly at a sensor. For example, "stat 64 B6" will stop the train at sensor B6.
- staf [train number] [sensor name]
  - "**St**op **af**ter" sends the stop command when the train triggers the chosen sensor.
- move [train number] [distance]
  - This moves the train by a short distance, under 40 cm. The distance is in millimeters. For example, "move 64 280" will move train 64 by 28 centimeters.
- mot [train number][time ticks]
  - "**Mo**ve **t**ime" sets the speed to 4, and after the specified number of ticks, where 1 tick is 0.01s, the stop command is sent.

Things to note

- Train 64 was calibrated on track A, on speeds 6, 8, 10 and 12.
- Train 63 was calibrated on track A for speed 8 and 12.

Source Code

Git repository: https://git.uwaterloo.ca/x272liu/cs452

Sha1 of the commit: 0d4ca710b270e9eecc42536766e622dae1f5ca8e

All files in the repository are part of this submission.