CS 452 Project 1

Felix Fung (f2fung)
Dusan Zelembaba (dzelemba)

July 8, 2013

1 How To Run

```
> load -b 0x00218000 -h 10.15.167.4 "ARM/dzelemba/a4_final.elf
> go
```

2 Submitted Files

Files listed here can be found under `/u1/dzelembaba/cs452/frozen_a4/`

2.1 md5sums

3 Project Description

3.1 Key Servers

There are 4 main servers used to track and control the trains: the sensor server, location server, distance server and train controller. There are two main points worth mentioning about how these servers communicate with one another.

First, the servers talk to one another using couriers. Since we want our servers to be Receive blocked most of the time, we use couriers to perform blocking calls to other servers, so that the client server can perform other tasks while waiting for data.

Second, clients that want data from a server must query that server often enough to not miss an update. This shouldn't be a problem as each server has couriers whose sole job it is to wait on updates from other servers.

This approach was chosen for simplicity. Any approach that involves buffering data and keeping of track of which client is requesting what data seemed like it would just add complications to the code and overhead in communication. Additionally, if our couriers aren't getting querying servers fast enough we're probably doing too much anyway and buffering wouldn't solve our problem.

3.1.1 Sensor Server

The sensor server is responsible for providing clients with the newest triggered sensors. Clients ask the sensor server for which sensors were triggered in the latest sensor dump.

The sensor server has a notifier that queries the track sensor data and pings the sensor server when is receives new sensor data. This allows the sensor server to perform other tasks while the notifier is blocked on train input.

Currently, the only clients of the sensor server are the user prompt and the location server.

3.1.2 Distance Server

TODO

3.1.3 Location Server

The location server is responsible for keeping track of the locations of the trains. Clients ask the location server for updates to all train locations.

The location server is given a train to track and the current location of that train by the train controller. After that, it can track the train going around the track assuming no sensors malfunction and our view of the switch states is correct. It does this by getting sensor updates from the sensor server and looking for the next sensor the train should hit. It also reveives a message from the train controller every time a train reverses, so it knows to change direction and look for a different sensor.

The location server also receives updates from the distance server that tells it how far a train has moved in micrometers and the current stopping distance of the train. It can then use this data to determine when a train is on a branch or merge node as well as provide error estimates at sensor nodes.

3.1.4 Train Controller

The train controller is the top level server that controls all messages being sent to the trains. It accepts commands to change speed of the train, give a train a route, or begin tracking a train.

To track a train, the train controller needs to know where the train is located. To do this, is starts moving the train very slowly until it hits a trigger. Once it has this data it notifies the location server of the trains location.

To give a train a route, the train controller gets a shortest path using the path finding algorithm described below. Using the path, it subscribes to location updates from the location server to perform the necessary actions at each step in the path. The location server provides the train's stopping distance and direction that the train controller uses to calculate "lookahead" distances for turning switches, performing a reverse or stopping. If the nodes at which actions need to be performed at are less than "lookahed" distance away, the train controller performs the action. This relies on accurate mesurements, so a

generous error buffer is used to ensure actions don't get performed too early or too late.

3.2 Path Finding

TODO