CS452 Project

Ronuk Raval (20345750) & Christophe Biocca (20322763)

July 27, 2012

Part I
Usage

1 Executable

The executable is located at `/u/cbiocca/cs452/project/kernel.elf` (checksum: `c725bd067c1a130f41cb`
To remake the executable:

```
cd /u/cbiocca/cs452/project/src/
make prod
```

2 Running the program

1. The program can be started using the standard commands.
2. It will take a second or two to boot up, as it needs to reset the track to a known state.
3. Once running, the clock can be found in the top right, the switch positions in the center area, and the recently triggered sensors below.
   (a) The most recently triggered sensor is on the left, less recent sensors on the right.
4. It also displays the train's known position in the top left.
5. The command area is at the bottom of the screen.
   (a) It accepts the 4 required commands.
   (b) It is picky about syntax. Trailing characters are an error, and so is more than one space as a separator. Commands execute when the enter key is typed.
   (c) To help diagnose typing errors, if the parser notices that invalid input is being entered, it will highlight it in red, starting at the position the command became invalid. It validates ranges for numeric arguments.

(d) All numbers are in decimal, including switch addresses (Making the center switches 153-156 inclusively).

(e) Syntax is case insensitive, all letters are lowercased coming in.

(f) Starting an engineer to manage a specific train is `e #train`.

(g) Once the engineer has acquired a location, routing it to a given spot is `z #train landmark`.

2.1 Troubleshooting

Hopefully this is unneeded, but if necessary, rebooting the train controller, clearing the screen and resetting the arm controller before starting the program again tends to work nicely.

3 Source Code

The source code submitted for this assignment can be found under `/u/cbiocca/cs452/project/src/`. Here are the checksums for the files:

```
83185b7100ee5f0d25ac34c0d3d584d3  ./.gitignore
4ca0d8d7e008482f8463d81131d31523  ./Makefile
436afe7fc14408ee2f8be159b83c4ac1  ./README.md
0d7272fd2615423d956ebf7b375929d8  ./data/BEST_HASH
19380cbbab26baf24037832a08da5c36  ./data/parse_track.py
ba66a1256e252e3ee3f38ad235e61b15  ./data/parseoutput.rb
2f6395e2f4c680f79e35804b969c4f54  ./data/tracka
445fe48f0181038f32c3bd58e4472a3c  ./data/trackb
943217198c8f49c0ba9fea5d40f6baf8  ./doc/CALIBRATION
d761266f3eb599d0dfc6dd88d52619c9  ./doc/WORKFLOW
5fb080eae0d5af66c80b275837d829c6  ./doc/k1.lyx
af526ee1c1869d021c620b322b9f7e25  ./doc/k2.lyx
691be4bc843ac1b54e2e701a98b4fd81  ./doc/k4.lyx
acd7d9afde797dda0c3d097569d301af  ./doc/tc1-hack.lyx
1b5890fbf8d8274f9e3dcdd2ebfb12ea  ./doc/tc1.lyx
3fb369afd0a490365655a5bfa157adf8  ./include/bwio.h
c13942a5833307674dba2f254809595d  ./include/cpsr.h
994cb3d4c9fdf6d59f7d2e8fd5190a55  ./include/debug.h
91ee8f0b6de1f3e23857502e3732c852  ./include/kernel/interrupts.h
f48ea9097d647cf0e299eea0cb838117  ./include/kernel/ipc.h
04095eebff3a2559d09e16cd4e998617  ./include/kernel/task.h
9976c1ab7ad644682dafae51172ef619  ./include/kernel/trampoline.h
aeb47082076d178c1784cc5dcd786a2b  ./include/lib.h
7ccf926195225fcab016186c8c5185fe  ./include/stdbool.h
ab50e9674e5eb7aa23b97d35028320fa  ./include/ts7200.h
1a026162a95f84eb28bfae5e8c50c429  ./include/user/clock.h
b11121fa5217f977a0441d8721f0520d  ./include/user/clock_drawer.h
```

```
f6fb657b597079f5a7821c54e8b24962  ./include/user/controller.h
dbdf4d53d41168a98bd51fd9ea4997f3  ./include/user/courier.h
ebed7e5cb0e74134077adcbe34b95141  ./include/user/engineer.h
13fa7622b7c42f761c5dbef88033b643  ./include/user/freight.h
e9a7384b7de24bf6eca953fb6555e0c0  ./include/user/heap.h
c43f99b170424b893e6769414cca0292  ./include/user/init.h
028a92575621233bc0689523896a6c71  ./include/user/kinematics.h
caf77700473316bef55fe272ecbcfc32  ./include/user/log.h
af578ce26dc1a2dd92828b10bfe1175e  ./include/user/mio.h
1f083cc3eb8d2b6cc7fe0467b7bd871b  ./include/user/nameserver.h
1133e3d3b68d8ddfc68f48b6de2a0112  ./include/user/parser.h
d5b21bea98ff50825398fff55b01601c  ./include/user/pathfinding.h
4c0bdd4751ecb8ed3be25dd71d7c86be  ./include/user/priorities.h
65dc14ad865906588c2f844ef3761116  ./include/user/sensor.h
16fa2976ca7fa795f870412a5e5c2dd4  ./include/user/string.h
403814223eb206dec10dd4796c930b42  ./include/user/syscall.h
6948a47691ec5fe21fe91d8026a00fa3  ./include/user/tio.h
936d372675c83ab6ace0c553594580cf  ./include/user/turnout.h
08ae8d519a47df09d891402ef4480e64  ./include/user/vt100.h
25757ddee6d82a16be926075e04b4972  ./kernel/bwio.c
3880bd8c7594d3b0b5f11e55a7c0f41d  ./kernel/interrupts.c
3ebd6f0e170452eda4094e49b2bc7507  ./kernel/ipc.c
6adcd146cc145f41e75222c464d6ba18  ./kernel/kernel.c
b6927156163cab78bc599a7cc1e76949  ./kernel/lib.c
a5551cb2004e046f38ebd83a618db810  ./kernel/task.c
ee3f874c894c039f95edbbf2e1350e05  ./kernel/task_internal.h
ba2bbda6474d8eb0c742de1778c353d1  ./kernel/trampoline.c
85a06c179e618d41cfa86f5a51346f43  ./linker.ld
ee9847d34ef3f14582e58f2bbdb3c2da  ./user/clock.c
ae9d089d2f5260d6f2ecaea987e19b67  ./user/clock_drawer.c
537ca13a501e93cd4e7ba5deafd800f5  ./user/controller.c
5db228d5d3288c110d866f049d6f6f94  ./user/courier.c
d8ae55d544e7b5293e0f7c5ae79fa235  ./user/engineer.c
e3810e414941275f21c099fe160815d6  ./user/freight.c
7ceb083605959f58f2c36785210a2399  ./user/init.c
e2af7655b1c5b954c3e9c4e964bf27d2  ./user/kinematics.c
a35284dd4646ca6c3d0784651e5a4bf2  ./user/log.c
10d8149abda2c2a5247caadb4118484d  ./user/mio.c
1056255ba9ee68987414d82da357f47e  ./user/nameserver.c
7104c426418ada62001c961f6aab1d59  ./user/parser.c
825e85bc28c7e806a24eb84fab973207  ./user/pathfinding.c
bedac67e6bda1fd538f0ae710a9e1bdf  ./user/sensor.c
33da1c0024215d6c4469d6050e90b91a  ./user/syscall.c
dabfe5cebd5955c1f839def8cf457d4b  ./user/tio.c
68ddaf08a986158a67985af6b6729086  ./user/turnout.c
```

eea47287ff66b4bc33eb8d2da57c93ca ./user/vt100.c

Part II
Track/Train Control

4 Train Controller

The train controller manages three pieces of data:

1. A circular buffer of trains trying to find their position.
2. An array with engineer task id and last recorded position for each train.
3. A two-dimensional array mapping sensor and sensor number to engineer task id, (keeping track of engineer expectations).

It essentially runs in an event loop, receiving engineer requests (through their respective couriers) to set sensor expectations and switching turn-outs,
sending messages to them based on new sensor readings. It also updates the screen as needed.

5 Train Engineer

The engineer first spawns 2 helper processes:

- A timeout worker, which waits until a given time, then replies to the engineer, ad nauseam.
- A courier, used in messaging the train controller.

Then it drives until it hits a sensor. This sets its starting position. When it receives a sensor update, or a timeout, it recalculates its current position. It also sets expectations as to which sensor will next be hit, and when. As it moves it keeps a model of its current acceleration and speed, periodically corrected by the sensor updates coming in from the train controller.

5.1 Pathing

When sent an instruction to navigate to another position, the engineer first calculate an optimal path using Djikstra's algorithm (technically A* with a heuristic of 0), using a closed list to avoid revisiting the same nodes over and over. After planning a path, the engineer starts moving forward. As it does it continuously sets the switches that are at most its stopping distance along its path. The stopping distance is derived from the current calculated speed. Finally when it notes that the final position is exactly the stopping distance ahead of it, it starts decelerating.

5.2 Engineer Modes

An engineer can be in several modes, which can be set from the command line:

User
When a user explicitly specifies a destination for the train.

Circle 1
A circle composed of the sensor destination sequence `{E14, D6, D1}`.

Circle 2
A circle composed of the sensor destination sequence `{C10, E2, A4}`.

Random
A random destination is picked for the train.

Freight
Described by Section 5.2.1.

With the exception of user mode, modes are comprised of a sequence of destinations. The next destination is only generated when the train has reached its current one and so a mode change only affects the train if it doesn't already have a destination. However, user mode overrides any previous destination by design, as a sort of manual override.

5.2.1 Freight Mode

A freight is defined as a package that exists at a certain location which must be picked up and delivered to another location. The freight server is responsible for randomly generating freights (or using user specified freights specified from the command line) and making them accessible in a globally readable array.

Since only a single train may carry a freight to its destination, some synchronization needs to be implemented. We did this by making sure that writes to the available freights are only done by the freight server, which avoids an entire class of race conditions. Furthermore, when a train arrives at the source location of a freight, it must claim the freight by contacting the freight server with the unique freight ID. The server will then respond with an integer 1 or 0, corresponding to the success or failure of claiming the freight respectively. It claiming a freight fails, it is because another train has already claimed it and the train must find another freight to claim.

Once a train has successfully claimed a freight, the freight server generates a new freight in its place. If a user has specified a freight from the command line, this is the freight generated, otherwise a random one is generated.

Trains in freight mode query the available freights, find the one closest to them, and navigate towards the source of the chosen freight. This process repeats until a freight is successfully claimed, at which point the train navigates to the freights destination to successfully deliver the freight. After that, it goes back into finding freights.

6 Reservations

Reservations are implemented as 2 independent halves by the train engineer task: respecting reservations on the track and making and releasing reservations as the engineer needs them. In the reservation system implemented, the granularity supported is limited to track edges only. For the purposes of the reservation system, a track edge and its reverse are treated equivalently.

6.1 Respecting Reservations

All the track edges are available as part of the global track graph, and is thus readable by all engineers. An extra field called `reserved` was added to the track edge data type, which stores the train ID that the edge is allocated to or `-1` otherwise. Thus, the invariant that all engineers must maintain is simple: one must never traverse an edge that they do not own. Since the state of reservations is available globally, checking reservations is cheap and doesn't even require a `Send()`/`Receive()` cycle.

When an engineer approaches a piece of track they do not own, they slow down and check the status later. An engineer must always ensure that they are capable of stopping without violating reservation invariants. If an engineer comes to a complete stop, they set a timeout which has both a static component (8 seconds) plus an additional random component based on the train ID (train 0 adds 0 seconds, train 80 would add 22.5 seconds). If this timeout is hit and the train is still unable to get the reservations it needs to follow the planned path, it replans its route. Route planning takes into account the current state of reservations and plans around it.

The random component based on train ID allows for fairly intelligent emergent behaviour from the trains. Since there is a large variance between the timeouts, in a deadlock situation, it is very likely that a single train would choose to replan before the other train. This first train would then follow its alternate route, giving up reservations which can then be used by the other train.

6.2 Making and Releasing Reservations

All updates to the reservation information in the track graph are made by the train controller, which is a single task. This synchronizes access to the graph and avoids an entire class of concurrency issues.

The engineer keeps track of reservations in three different classes. First, a function computes the set of edges that it thinks it needs. It does so by adding the edge that it is currently on, as well as all the edges its tail might be on if its near the ends of the edge. Finally, it projects forward by the current stopping distance, adding all traversed edges to the set, as well as edges on both sides of traversed branches.

Then a function compares what is in the set that is needed and the information presented in the track graph. If it detects that certain edges that it needs are are assigned to it, it moves said edges to a new set of granted reservations.

Finally, a function computes edges that are in the granted set but are no longer in the needed set. These edges are released back to the controller.

All the while, a courier is used as a communication bridge between the train controller and the engineer.

6.3 Interaction with Track Features

The train makes primary and secondary expectations (sensors that when triggered should be passed along to this specific engineer) with the train controller. Two engineers expecting a sensor is strictly an error and manifests itself as an assertion failure. The idea behind setting primary and secondary expectations is allow for track failures. For malfunctioning switches, the notion of an alternate sensor is also supported, and the switch would be marked as disfunctional should an alternate sensor be triggered enough times. However, at the time of writing, these systems are only partially implemented and no modifications to track topology are actually made.

To synchronize control of various track features, an engineer is only allowed to set an expectation on a sensor if it owns reservations on the edges on both sides of said sensor. Likewise, an engineer may only switch a branch turnout if it owns all 3 edges. This conservative system prevents quite a few accidents since expecting a sensor expressed the notion that a train rightfully intends to pass over it and thus can only do so if it has the reservations to back it up. Similarly, a turnout should not be switched unless the train owns all edges leading to/from it.