CS 452 Assignment 0: Polling Loop

Name: Shun Da Suo
ID: 20509411

May 9, 2018

This report contains the **user manual** and **design document** for a terminal program that controls the Märklin 6051 train set.

1 User Manual

1.1 Getting the Executable

Option 1: Build from GitLab source

1. git clone gitlab@git.uwaterloo.ca:sdsuo/trains.git
2. cd trains/a0
3. make && cp build/main.elf /u/cs452/tftp/ARM/sdsuo

Option 2: Use pre-built binary

1. cp /u9/sdsuo/cs452/a0/build/main.elf /u/cs452/tftp/ARM/sdsuo

1.2 Run the Program

To ensure proper testing, please hard reset the train set before running the program. Then issue the following commands in Redboot:

1. load -b 0x00218000 -h 10.15.167.5 "ARM/sdsuo/main.elf"
2. go

Note that the program will initialize the train set upon startup (i.e. perform the following sequence of actions):

1. Power on the train set

2. Set all switches to straight position
3. Initialize all trains to speed 0
4. Enable sensor polling

1.3 Command Line Interface

The command line interface displays:

1. Top left: Current elapsed time (at 100 ms precision)
2. Top right: Mean and max polling loop time
3. Left: Available switches and their current states
4. Right: Most recently activated sensors (up to 20)
5. Bottom: Status, history, and command prompt

Available commands include:

- **go**: Power on the train set
- **stop**: Power off the train set
- **tr train_number[1-80] train_speed[0-14]**: Set speed for specified train
- **rv train_number[1-80]**: Reverse specified train
- **sw switch_number[0-255] switch_direction[S/C]**: Set state for specified switch
- **swa switch_direction[S/C]**: Set state for all available switches
- **ss**: Toggle sensor polling

2 Design Document

At a high level, the program:

1. keeps stateful structures on the main method's stack,
2. uses ring buffers to buffer UART communication,
3. uses a polling loop for event handling.

2.1 Architecture

Main components of the program are:

train.h **TrainController**: Issuing train commands, scheduling delayed execution of commands

terminal.h
- **SmartTerminal**: Issuing escape codes, rendering and updating user interface
- **TerminalController**: Parsing user input commands

time.h **Clock**: Keeping time

track.h **Track**: Storing information about track nodes, available switches and sensors

io.h **BufferedChannel**: Providing read and write buffers for a UART communication channel

ds.h **RingBuffer**: Providing generic byte indexed ring buffer implementation

2.2 Polling Loop

The program is driven by a main polling loop, which does the following:

1. Poll Clock:
   - Check timer's value register. If the current value is larger than the previously recorded value, it means that the timer underflowed and was reset. Thus increment the clock by the tick period (i.e. 10 ms).
   - Check current time against display update period (i.e. 100 ms). Notify SmartTerminal to update display as necessary.
2. Poll BufferedChannel for terminal:
   - If ready to receive, read one byte onto read ring buffer.
   - If ready to transmit and write ring buffer is not empty, move one byte to terminal UART.
3. Poll BufferedChannel for train set:
   - If ready to receive, read one byte onto read ring buffer.

- If ready to transmit and write ring buffer is not empty, move one byte to train set UART.

4. Poll TrainController: If display time has changed (i.e. 100ms has elapsed since the previous check)
    - Check and execute delayed command ring buffers for train set commands scheduled
    - Check read ring buffer for sensor update bytes. Parse raw bytes into sensor ids and update sensor ring buffer. Notify SmartTerminal to update display.
    - If read ring buffer is empty and sensor polling is enabled, send commands to request sensor updates.
5. Poll TerminalController: If read ring buffer is not empty
    - And visible character is read, echo input to cursor position of prompt display, and append to command buffer.
    - And enter is read, call TrainController with completed command and update history display with command.
    - And backspace is read, clear input at cursor position of prompt display and remove last character from command buffer.

2.3 Implementation Details

2.3.1 Time Keeping

The 32-bit timer is used for time keeping. More specifically, we use the lower clock speed at 2kHz, and set the mode to periodic. At this clock speed, we can track time at the granularity of 0.5 ms, which is sufficient for our needs. The initial load value is set to 20 for a timer period of 10 ms (i.e. every underflow/reset corresponds to one period of 10 ms).

Response to Question 1: How do you know that your clock does not miss updates or lose time?)

This implies that as long as the polling loop runs within 10 ms, we will not lose any periodic resets. Empirically, we observe max polling loop time to be well below this threshold.

2.3.2 Ring Buffer

Ring buffers are the only non-primitive data structures used in the program. They are used extensively for decoupling producers and consumers of messages. Here, ring buffers are implemented with stack allocated static arrays, with pointers to both head (i.e. index of first byte) and tail (i.e. index of last byte + 1). They operate first in first out (FIFO). They are currently configured to silently overwrite the oldest bytes when the max size is reached.

2.3.3 Delayed Commands

To execute commands for reversing trains and flipping switches correctly, we use two ring buffers to store scheduled delayed commands. More specifically, we use a 5-byte protocol (i.e. a [4-byte time ms, 1-byte command] pair). When polling the TrainController, we check if the delayed command buffers are empty, and peak the timestamp at head (i.e. oldest delayed command). While the timestamp is less than the current time, we shrink the delayed command buffers and send the 1-byte command to the train controller.

2.3.4 Sensor Polling

To poll for track sensor updates. We periodically issue the one byte command to read all 5 sensor modules. To account for delays in train set response and transmission, we set this period to 100 ms. To further make sure we can keep up with the sensor updates, we do not issue new command to request sensor update until we have read off all previous bytes from the read buffer.

The 10-byte sensor update response is parsed together. We calculate and push the sensor ids onto the sensor update ring buffer. It is okay to directly reuse the byte indexed ring buffer, since the range for the sensor ids is 0 to 255. We do, however, set an additional size limit of 20 for the sensor update ring buffer, since we only care about the latest 20 sensor hits.

Response to Question 2: How long does the train hardware take to reply to a sensor query?

To verify the delays in train set response and transmission, we use a minimal polling loop that only polls for BufferedChannel updates and Clock updates. From the time the sensor update command is issued, it takes approximately

18 ms to receive the first byte of response. At the rate of 2.4Kb/s (i.e. 0.24KB/s), it takes approximately 4.2 ms to transmit each byte over the communication channel. Thus, it takes approximately **9.6 ms** ($= 18ms - 4.2ms * 2$) for the train set to query all 5 sensor modules and reply with the update. It takes approximately **60 ms** to receive all ten bytes, which is roughly what we expect ($18ms + 4.2ms * 9 \simeq 56ms$).

3 Source Code

Top level
f984e0cd6d3fbd4aa078f0d0dbf5aea9 ./Makefile
9bcd562566ba01c869c8c2759ff90e64 ./orex.ld

Headers
a7895ea1ba118b0b4c7f43c80b517ad7 ./include/bwio.h
9af226f127c1fd759530cd45236c37b8 ./include/ts7200.h
e3bd55ddffe9b1da2d05cd1000579b88 ./include/io.h
f78e35718b07f5b09e1d593c5a6f4a27 ./include/terminal.h
45c696fe33aa8211ffb49de54290c028 ./include/time.h
1cd15a9deef660d88bc2418ce67b0f52 ./include/ds.h
1352f3743944badbb8c2399e6fb2ccd4 ./include/track_data.h
e64108099d229b1cc61e739073ba5bf8 ./include/train.h
0955a30671db0a0150c7a03369254f30 ./include/util.h
8ffa85dd374f6f1226bee787fa6bf71d ./include/track_node.h
9c766bc8610c0b07c034520fd2e252b9 ./include/track.h

Implementations
2d93650ecdab0ab0f355d5cd0fce5067 ./src/bwio.c
b74c67b2873d2b8002ac9eeb473fb80a ./src/ds.c
d11cd29f4d84f43b35e0233b06a013cb ./src/io.c
e1ae9c610bcdbbd6fb2c476d3ffeeea3 ./src/io.s
532f6eecd9a0a1ff7b41dce7bc0dd673 ./src/main.c
0583f58d141f1f64917956f04a0b45a6 ./src/terminal.c
0522d6cea2a6ca669044252b3480c6cd ./src/time.c
5d8c49a0fd5ba8ba9902f95db6389a72 ./src/train.c
beeff5f6addf3f45eae797fc905d4816 ./src/track_data.c
d0dca3ac2c548f64339f3c726d96b4c7 ./src/util.c
b2a36b4b7803fc5705e584650c21b60a ./src/track.c