CS 452 – Assignment #0

full pathname to executable: /u/cs452/tftp/ARM/username/a0.elf
location of files: /u8/username/cs452/asst0/

MD5 hashes:
MD5(include/io.h)= 950e73473c3e995ce272256fe680d78d
MD5(include/trains.h)= 697c211da85b217609743ab3d9d29c65
MD5(lib/libio.a)= 2ffca4d57ae1f4ae5bf8fcc9f9f744f1
MD5(lib/libtrains.a)= 1e4b461df6ab65643644164bdf2de7bd
MD5(src/io.c)= b90ade523414992c263852419ec4a1ed
MD5(src/trains.c)= 33105e4ab4b0e8c60015e6db2ed4186d
MD5(test/a0.c)= ece68e31a3f94e29825f04516e0243fe
MD5(src/Makefile)= 9906232005414069a5342d13cc141724
MD5(test/Makefile)= 6a4a08c9948f0039150f9073a8bac82e

How to run

1. Load and run the program in from Redboot terminal:

```
>load -b 0x00218000 -h 129.97.167.18 "ARM/username/a0.elf"
>go
```

Commands:
tr train-number train-speed: changes train speed with number train-number to speed train-speed
rv train-number: reverses the train with number train-number
sw switch-number switch-direction: changes switch with number switch-number to
q: quits the program

Note: commands issued with incorrect arguments result in undefined behaviour or nothing happening.

To build from scratch:
-run make in /src directory
-run make in /test directory
-use elf in /test directory

Program structure

Most of the program occurs within the main loop of the program in a0.c.

Within the loop, the following actions are performed:
1. input and output to the terminal and the train controller using functionality provided by io.c
2. the time is updated every tenth of a second
3. sensor data is requested and processed
4. train commands are processed using functionality provided by trains.c

Compiled Libraries:
lib/libio.a: compiled library for io functions

lib/libtrains.a: compiled library for train functions

Source Files:
src/io.c - contains all functions used for managing input and output from train controller and terminal
src/trains.c - contains functions used to execute train commands (tr, rv, sw)
src/makefile – builds the io and trains source files and puts results in lib folder

test/a0.c - main program where polling loop is run, as well as functions used to update the terminal screen and argument parsing
test/makefile - builds final .elf file
test/a0.elf – resultant .elf file from build project.

Associated header files:
include/io.h
include/trains.h

Important Data Structures
2 circular buffers for output:
-one to store characters to be written to the screen, and one to store bytes to be sent to the train

1 buffer which stores one command entered by the user (command buffer)
-implemented using a fixed size array and an index to keep track of where to put the next character in the command
-When the ENTER key has been pressed, the command and its arguments are parsed from this buffer, and appropriate action is taken.

1 circular buffer to store trains in need of reacceleration after being stopped for reversal

1 fixed size array to store current train speeds
-this is used when reversing trains, so that it will return to its previous speed after stopping and reversing

1 fixed size array to store sensor data for processing
-also includes one index to keep track of how many bytes have been received from the train controller with regards to the sensor data.

1 fixed size array to store previous sensor data

All circular buffers are implemented in the following way:
-uses a fixed size array and two indexes:
    -the first index (bpos) points to the character about to be sent to the train/terminal
    -the other index (bstore) points to an empty slot where output to the terminal/train can be stored before being sent.
-every time a character is sent, the bpos index is incremented.
-every time a character is stored in the buffer, the bstore index is incremented.
-used this implementation for simplicity

Implementation

Initialization:
-set all switches to 'C' (curved/branch)
-set current train speeds in trains array to 0.
-set up screen for time, switches, sensor data, and command prompt

Keeping track of time:
-32-bit timer with 508kHz clock selected was used for greater accuracy in measurement (the 2kHz would not have been accurate enough to measure tenths of seconds)
-a function called updateTime is called once per polling loop to check whether to update the time.
-time is only actually updated on the screen every tenth of a second, or every 50800 clock ticks with the 508kHz clock.

How do you know that your clock does not miss updates or lose time?
-taking measurements, the worst time for a full loop execution is around 600-800 clock ticks (approx. 1.57 milliseconds), which is well under 1/10 of a second. Since we check whether to update the time every iteration of the polling loop, we will not miss an update.

I/O related Implementations
-getc and putc are non-busy wait functions in io.c used for receiving input and sending output respectively
-incoming input from the terminal is written to the terminal output buffer so that it will be echoed onto the screen as well as a command buffer, where, upon pressing the ENTER key, will be processed.
-output is written to the terminal by writing the next byte to be sent in the terminal output buffer into the appropriate data register
-input from the train is stored into the sensor data buffer (since that's the only information that will be received), and is processed once the buffer is full
-output to the train is handled much in the same way as output to the terminal, except we only send commands every 9 ms since the train can't process commands very quickly

Train Related Implementations
sw and tr commands
-after the command has been parsed, the command is processed and the appropriate bytes are written to the train output buffer, which will eventually get written to the UART data register attached to the train controller and sent.
-for sw, updates the switch position on screen.

rv command
-when the command is parsed, the bytes needed to stop the train (train number and speed 0) are written to the train output buffer and a timestamp is taken.
-using the timer, we wait until 3 seconds after the timestamp is taken for the train in question to ensure that the train has fully stopped
-the appropriate bytes are sent to reverse the train, and using the train speed array, sends the bytes required to reaccelerate the train to its former speed.

Sensor data:
-uses a flag called requested to determine whether to send a request for sensor data. This way we don't have floods of sensor data being sent from the train.
-also uses a counter to determine how many bytes have been received from the train controller.
-if we haven't requested for sensor data already, we put the byte 133 (requests sensor data from all five sensor boxes) into the train output buffer, set the requested flag and set the counter to 0.
-as the sensor data is received, we store it in the buffer sensors and increment the counter.
-when all bytes are received, process the sensor data and write info the the screen.
-to avoid redundant writing of sensor data (ex. C3 appears multiple times on screen after going over that sensor), the previous state of the sensor is checked, and if it is the same, nothing is written to the screen.

How long does the train hardware take to reply to a sensor query?
-measured by taking the time when the byte 133 is sent to the train controller (putc(133, COM1)) and when the first byte received from the train controller
-ranges from 8800-8900 clock ticks using the 508 kHz clock ~17 ms.

<u>Bugs and Quirks</u>
1. Clock overflow has not been handled. I have attempted to handle this, but my program always crashes whenever I tried to. I don't anticipate this should have any visible effects on program execution unless the program is run for over 2 hours.

2. After quitting the program and immediately running the program again, a buffer overrun occurs when polling for train input. This results in the sensor data being displayed out of sync - i.e. sensor B16 being tripped may appear on the screen as B8 or C8. I attempted to fix this problem using a flag to signal buffer overruns, but wasn't able to fix it.

3. If you reverse a train before issuing a tr command to that train, it will reverse, but won't reaccelerate due to the fact that the array containing current speeds of trains is initialized to zero.

4. Copy and pasting commands into the terminal will probably not work too well; as the rate at which the polling loop checks for input is around 150 ticks between checks.

5. Turning the train controller off and then on during program execution will result in some things not working, so please refrain from doing so.