The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2015
CSSE2310 / CSSE7231 - Assignment 4
Due: 11:10pm 30 October, 2015
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 4.3

Introduction

This assignment will involve creating an abstract simulation of a transportation network (described below). It will requre the use of pthreads, tcp networking and thread-safety. Your assignment submission must comply with the C style guide available on the course Blackboard area.

When your programs execute, they must not create any files on disk not mentioned in this specification. You may create C source and and header files during development as you see fit [1]. This is an individual assignment. You should feel free to discuss aspects of C programming and the assignment specification with fellow students. You should not actively help (or seek help from) other students with the actual coding of your assignment solution. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that all submitted code may be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you. A likely penalty for a first offence would be a mark of 0 for the assignment. Don't risk it! If you're having trouble, seek help from a member of the teaching staff. Don't be tempted to copy another student's code. You should read and understand the statements on student misconduct in the course profile and on the school web-site: `http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`

As with Assignment 1, we will use the subversion (svn) system to deal with assignment submissions. Do not commit any code to your repository unless it is your own work or it was given to you by teaching staff. **If you have questions about this, please ask.**

The simulation

The simulation will consist of a number of "stations".Each station may be connected to a number of other stations via network connections. Network messages representing "trains" will arrive via these network connections, pick up or deposit resouces and move on to the next station.

Implementation

You will produce a single program called `station`.There are no separate client and server programs in this assignment. Each station will listen on a single TCP port for connections. All stations in a simulation will start off isolated and listening on their designated port. Programs connecting to this port will supply an authentication string (one line of text) followed by their station name and a sequence of trains. The station name will be used to forward trains to that station if required. Your program must make non-trivial use of multi-threading and must act on incoming requests from different connections concurrently. You are not permitted to use `select` or any other form of non-blocking I/O.

[1] Note however, that all such files in the /ass4 directory will be style marked

Invocation

The parameters to start a station will be:

- Name — the name of this station
- Authentication file — A non-empty file, the first line of which will be used to decide if incoming connections are permitted.If any new connection does not supply this string as its first bytes, it is to be disconnected.
- logfile — Name of a file to write station statistics to if required.
- [optional] port— port to listen for new connections on.
- [optional] interface to listen on (defaults to listening on all available interfaces (`INADDR_ANY`)). The port argument must be present if this argument is.

For example:

```
./station central auth1.txt t1.log 43000
```

When a station is ready to accept connections, it should output the port number it is listening on followed by a newline to standard out. If the station was started with no port, then use an ephemeral port.

Log file

Whenever a station receives a `SIGHUP` (or after it has processes a train which would shut it down), it should append the following to its logfile (newline separated).

- `=======`
- The name of the station.
- `Processed:` followed by the total number of trains processed by the station since start.That is, the number of trains which did not have errors.
- `Not mine:` followed by the total number of trains discarded due to arriving with at the wrong station.
- `Format err:` followed by the total number of trains disacarded due to format errors.
- `No fwd:` followed by the total number of trains discarded due to having an invalid next station.
- A comma separated list of stations which this station is currently connected to (in lexicographic order). If the station is connected to no other stations, output `NONE` here.
- A list of resources with have been loaded or unloaded at the station (one per line) with each name followed by the net amount loaded and unloaded. This list should be sorted lexicographically by resource name.
- Finally, if the station has been shutdown, output either `doomtrain` or `stopstation` depending on which closed the station.

For example:

```
=======
Roma Street
Processed: 25
Not mine: 0
Format err: 0
No Fwd: 3
Central,Moria,South Brisbane
mythril -20
pineapples 480
pumpkins -700
```

Errors

Condition	Exit status	Message
Incorrect number of arguments	1	Usage: station name authfile logfile [port [host]]
Name is the empty string or the authfile can not be read or the auth file's first line is empty	2	Invalid name/auth
The log file can not be opened for write.	3	Unable to open log
Port is invalid (not a positive integer > 0, < 65535	4	Invalid port
Unable to listen on the socket	5	Listen error
Unable to connect to a station specified by an `add` train.	6	Unable to connect to station
Two stations are found to have the same name	7	Duplicate station names
Unspecified system call failure	99	Unspecified system call failure
Normal shutdown (doomtrain or stopstation)	0	

Train processing

For each line of text arriving on a connection, the station will check if the name at the front of the train matches its own. If not, the train will be discarded. Next, the type of train will be determined:

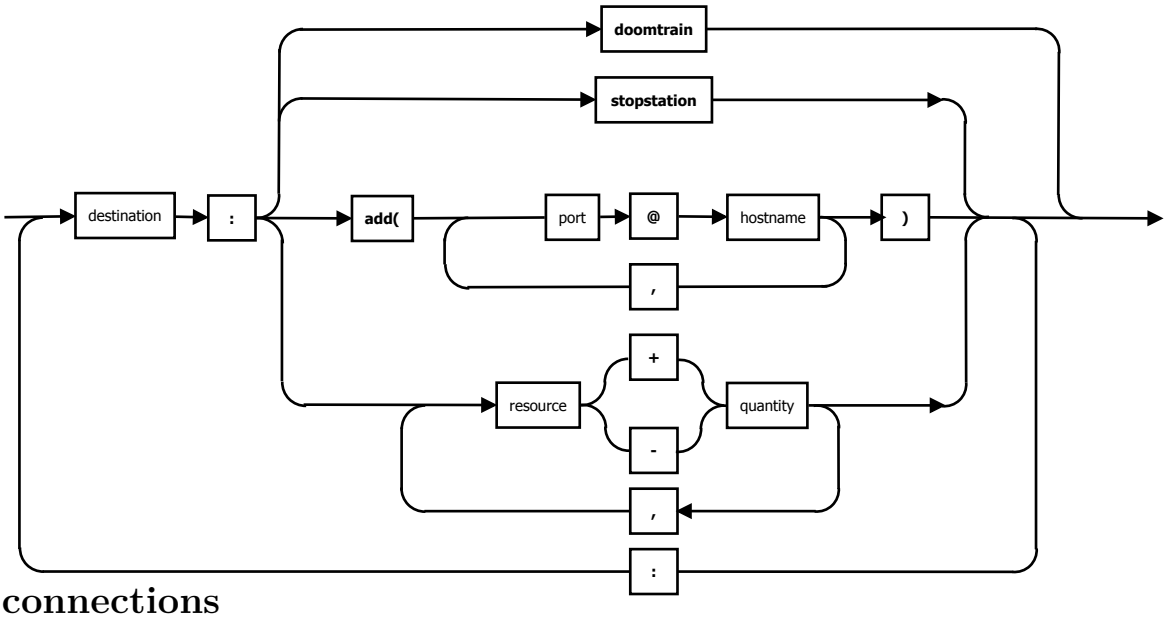
- `doomtrain` — When this train is processed, the station will send a doomtrain message to all other stations it is directly connected to and then shut down. Any further destinations following the doomtrain should be ignored.
- `stopstation` — When this train is processed, the station will forward the train to its next stop and then shut down.
- `add` — gives a list of other stations this station should connect to
- otherwise intrepret it as a list of resouces to load / unload.

Only the current payload will be processed at each station. So.

`currentstation:coal+20:nextstation:add(,,,+`

Would be processed as normal and sent on to nextstation (where it would have a be discarded due to format error).

Message format



New connections

When a station attempts to connect to another station, it must send the authentication string followed by its own name. The station it has connected to will send back its name.

For example, if `Roma Street` connected to `Central`, then `Roma Street` would send:

```
secret
Roma street
```

and `Central` would send back:

```
Central
```

Compilation

Your code must compile (on a clean checkout) with the command:
`make`

Each individual file must compile with at least `-Wall -pedantic -std=gnu99`. You may of course use additional flags (eg -pthread) but you must not use them to try to disable or hide warnings. You must also not use pragmas to achieve the same goal.

If the make command does not produce the required programs, then those programs will not be marked. If none of the required programs are produced, then you will receive 0 marks for functionality. Any code without academic merit will be removed from your program before compilation is attempted [This will be done even if it prevents the code from compiling]. If your code produces warnings (as opposed to errors), then you will lose style marks (see later).

Your solution must not invoke other programs. Your solution must not use non-standard headers/libraries.

Submission

Submission must be made electronically by committing using subversion. In order to mark your assignment, the markers will check out `/trunk/ass4/` from your repository on `source.eait.uq.edu.au`. That is, `https://source.eait.uq.edu.au/svn/csse2310-$USER/trunk/ass4`. Code checked in to any other part of your repository will not be marked.

The due date for this assignment is given on the front page of this specification. Note that no submissions can be made more than 96 hours past the deadline under any circumstances.

Test scripts will be provided to test the code on the trunk. Students are *strongly advised* to make use of this facility after committing.

Note: Any .h or .c files in your trunk will be marked for style *even if they are not linked by the makefile.* If you need help moving/removing files in svn, then ask.

You must submit a `Makefile` *or we will not be able to compile your assignment.* Remember that your assignment will be marked electronically and strict adherance to the specification is critical.

Marks

Marks will be awarded for both functionality and style.

Style (6 marks)

If g is the number of style guide violations and w is the number of compilation warnings, A is the number of style warnings as detected by the style checker, and F is your functionality mark: your style mark will be:

$$\begin{cases} = \min\{F,\ 6 \times 0.9^{g+w}\} : \text{if } A < 12 \\ = 0 : \text{if } A \geq 12 \end{cases}$$

That is, your style mark will be set to zero if you have more than eleven (capped) style violations. Also, Your style mark can never be more than your functionality mark — this prevents the submission of well styled programs which don't meet at least a minimum level of required functionality.

The number of compilation warnings will be the total number of distinct warning lines reported during the compilation process described above. The number of style guide violations refers to the number of violations of the current C Programming Style Guide. A maximum of 5 violations will be penalised for each broad guideline area. The broad guideline areas are Naming, Comments, Braces, Whitespace, Indentation, Line Length and Overall. For naming violations, the penalty will be one violation per offending name (not per use of the name) up to the maximum of five. You should pay particular attention to commenting so that others can understand your code. The marker's decision with respect to commenting violations is final — it is the marker who has to understand your code. To satisfy layout related guidelines, you may wish to consider the `indent(1)` and `expand(1)` tools.

Functionality (44 marks)

Provided that your code compiles (see above), you will earn functionality marks based on the number of features your programs correctly implement, as outlined below. Partial marks may be awarded for partially meeting the functionality requirements. Not all features are of equal difficulty. If your program does not allow a feature to be tested then you will receive 0 marks for that feature, even if you claim to have implemented it. For example, if your program can never open a file, we can not determine if your program would have loaded input from it. The markers will make no alterations to your code (other than to remove code without academic merit). Your programs should not crash or lock up/loop indefinitely. Your programs should not run for unreasonably long times.

- Argument checking on a single station (5 marks)
- Simulations with a single station and no next station fields (5 marks)
- Simulations with two stations (5 marks)
- Simluations with a line of stations (6 marks)
- Simulations with an arbitrary arrangement of stations (10 marks)
- As above but with late joining stations (7 marks)
- As above but with early shutdown stattions (6 marks)

Note: *The only output your program produces that can be marked is the log file for each station. If your program does not produce the logfile correctly, you will not be able to get marks.*

Late Penalties

Late penalties will apply as outlined in the course profile.

Specification Updates

It is possible that this specification contains errors or inconsistencies or missing information. It is possible that clarifications will be issued via the course website. Any such clarifications posted 5 days (120 hours) or more before the due date will form part of the assignment specification. If you find any inconsistencies or omissions, please notify the teaching staff.

Test Data

Test data and scripts for this assignment will be made available. The idea is to help clarify some areas of the specification and to provide a basic sanity check of code which you have committed. *They are not guaranteed to check all possible problems nor are they guaranteed to resemble the tests which will be used to mark your assignments.* Testing that your assignment complies with this specification is still *your* responsibility.

Notes and Addenda:

- Changes 4.2 → 4.3
    1. Fixed the text to have resource counts be a single net value.
- Changes 4.1 → 4.2
    1. Added labels to station log output.
    2. Clarify meaning of extra text in doomtrains.
    3. Clarify the case for no-hostname.
    4. Removed spaces between station names in example.
- Changes 4.0 → 4.1

1. Added full URL for submission repo.
2. Clarified optional parameters and changed usage message to match.

- Changes 0.9.9 → 4.0

1. Stations can not forward trains to themselves. For example, If the station is `x`, then `x:coal+20:x:grain+20` would have an invalid destination.
2. Corrected ordering in example to match rest of spec.
3. Introduced an error for being unable to open log file. The log file should be opened (append) when the station starts and kept open for the lifetime of the station. **Note:** To keep the order of error checks sensible, some of the other errors have had their exit statuses changed.
4. "Discarding" a train means don't do anything more with it (other than freeing up relevant memory) and move on to waiting for a new train.
5. Changed column widths in the errors table to make error messages clearer.
6. Fixed one of the notes, to avoid implying that `2130port` hands out ephemeral ports (which it doesn't).

- Notes:

1. Your programs must not invoke any other programs.
2. Be sure to test on moss.
3. You should not assume that system calls always succeed.
4. Tab stops are assumed to be 8 characters wide.
5. You may not use any `#pragma` in this assignment.
6. When you need network ports please use the ones assigned to you(run `2310port` - you may use that port and the two next ports) or use emphemeral ports .
7. None of the following functions (or equivalents) are permitted in this assignment:
   (a) poll()
   (b) select()
   (c) fork()
   (d) system()
   (e) popen()
8. You are not permitted to use non-blocking IO.
9. Shutdown, and SIGHUP handling should be handled immediately. Yes, this means that some things in process may be cut off.