The University of Queensland
School of Information Technology and Electrical Engineering
Semester One, 2020
CSSE2310 / CSSE7231 - Assignment 4
Due: 20:00pm 12th June, 2020
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 4.1

Integrity

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.

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.**

While you are permitted to use sample code supplied by teaching staff this year in this course. Code supplied for other courses or other offerings of this course is off limits — it may be deemed to be without academic merit and removed from your program before testing.

Purpose

In this assignment you will implement a simple networked simulation of plane flights. A complete assignment will consist of three C99 programs:

- roc2310: Simulates an aircraft
- control2310: Simulates an airport
- mapper2310: Maps airport names to ports

Multiple instances of these programs will be executed to carry out the simulation.

You will use pthreads and communicate via IPv4 TCP networking. Your implementation must use blocking communications. You are not permitted to use pipes or any alternative means of interprocess commmunication. To complete all of the features of this assignment you will need to use pthreads, you are not to call `fork()` nor attempt to use any form of nonblocking I/O or I/O multiplexing.

Your assignment submission must comply with the C style guide (version 2.0.4) available on the course blackboard area. It must also not use banned functions or commands listed in the same place.

Programs

These programs make use of IDs and other text (info). No text or IDs are allowed to contain any of the following characters:

- `'\n'`
- `'\r'`
- `':'`

Note, they may contain spaces.

All outputs from programs should be newline terminated.

mapper2310

This program takes no commandline parameters. When it starts up, it will

1. listen on an ephemeral port,
2. print that port to `stdout`.
3. act on any commands it receives via connection to the port (the other end of the connection will close it when it has no more commands).

Some marks are available for handling connections one at a time. To get all marks related to the mapper, it will need to be able to process requests in parallel.

Command	**Purpose**
?*ID*	Send the port number for the airport called *ID*
!*ID*:*PORT*	Add airport called *ID* with *PORT* as the port number
@	Send back all names and their corresponding ports

All output as a result of commands will be sent back via the connection that the command came from.

- `?ID`
  - If there is an entry corresponding to that ID, send back the port followed by a newline.
  - If there is no mapping, send back a semi-colon followed by a newline.
- `@`
  - Mappings should be printed in lexicographic order (of ID) with newline at the end of each entry. ID and PORT should be separated by a colon.

If any invalid commands are received, silently discard that line and keep reading. If `!` is used for an ID which already has an entry, ignore that command.

Note: the mapper will run until killed.

Sample session

To mapper	**From mapper**
?To	;
!Tom:55451	
!Albert:13021	
@	Albert:13021 Tom:55451
?Albert	13021
!To:4444	
@	Albert:13021 To:4444 Tom:55451

control2310

This program takes the following parameters:

- AirportID
- Airport info
- mapper_port (This is optional).

When the program starts up, it will:

1. Listen on an ephemeral port.
2. Print that port to `stdout`.
3. If a mapper port is given, connect to the mapper and register the ID and port of this airport. It should then disconnect from the mapper.
4. In parallel — wait for connections by planes and act on them.

When a connection is made to the control's port, one of two things should happen:

- If the text sent by the connecting party is "log", then send back a newline seprated list of all the rocs which have visited them (in lexicographic order). Following this, it should send a full-stop followed by a newline and then close the connection.
- For any other text, the control should consider the text as the plane's ID and send back the control's info (newline terminated).

Note: Under normal circumstances `control2310` will run until killed.

Exit conditions for control2310

Condition	Status	Message
Incorrect number of args	1	`Usage: control2310 id info [mapper]`
Invalid characters in ID or info	2	`Invalid char in parameter`
Port specified but not a strictly positive number < 65535	3	`Invalid port`
Could not connect to mapper	4	`Can not connect to map`

roc2310

This program takes the following commandline parameters:

1. planeID
2. mapper_port (This is either a port number or a dash ('-')).
3. zero or more destinations (controls) to connect to in order. These can either be port numbers or control IDs.

If all destinations are port numbers (> 0 and < 65536), the mapper is not required. If any of the destinations are not port numbers, the mapper should be contacted to find the ports for those destinations. Once the roc knows the all the port numbers for its destinations, it should connect to each destination in turn, get the info for that destination and add it (newline separated) to its log. If it is unable to connect to a destination (or the destination did not send back valid information), the roc should just move on to the next destination.

When all of the destintations have been dealt with, the roc should print out its log to `stdout` and exit.

Exit conditions for roc2310

Condition	Status	Message
Normal operation	0	
Incorrect number of args	1	`Usage: roc2310 id mapper {airports}`
Mapper is not dash but is not a valid port number either.	2	`Invalid mapper port`
A destination is not a valid port number (so the mapper is required), but no valid mapper port was given	3	`Mapper required`
Error connecting to the mapper port	4	`Failed to connect to mapper`
Mapper has no value for one of the queried destinations	5	`No map entry for destination`
Could not connect to a destination port	6*	`Failed to connect to at least one destination`

Notes: Exit status 6 is special in this case. If roc fails to connect to one of its destinations, it should not exit immediately. Instead, it should:

1. process the rest of its destinations as normal
2. print out its log to standard out
3. print the error message and exit with the error status.

Pay attention to the order of the error conditions here. All of the destinations must be converted to port numbers before trying to connect to the first destination.

roc2310 / control2310 communication

When roc2310 connects to a control, roc will send its ID to control and the control will send back its info.

For example:

```
$ ./control2310 BNE "Quarantined"
45643
```

```
$ ./roc2310 F100 - 45643
```

In this case, the roc would send `F100` and the control would send `Quarantined`.

Compilation

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

Each individual `.c` file must compile with at least `-Wall -pedantic -std=gnu99`. You may of course use additional flags but you must not use them to try to disable or hide warnings. You must also not use pragmas to achieve the same goal. Your code must be compiled with the `gcc` compiler.

If the make command does not produce the required program, 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).

Any libraries/headers/functions your solution uses must be standard-C or POSIX compliant.

Submission

No late submissions will be marked for this assignment under any circumstances. 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`. Code checked in to any other part of your repository will not be marked.

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/ass4` directory 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. Consult the style guide for other restrictions.

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)

Style marks will be calculated as follows:
Let A be the number of style violations detected by simpatico plus the number of build warnings. Let H be the number of style violations detected by human markers. Let F be the functionality mark for your assignment.

- If $A > 10$, then your style mark will be zero and M will not be calculated.
- Otherwise, let $M_A = 3 \times 0.8^A$ and $M_H = M_A - 0.5 \times H$ your style mark S will be $M_A + \max\{0, M_H\}$.

Your total mark for the assignment will be $F + \min\{F, S\}$.

Functionality (44 marks)

Provided that your code compiles (see above), you will earn functionality marks based on the number of features your program correctly implements, 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.

- mapper
  - `!` and `?` (2 marks)
  - `@` (4 marks)
  - Invalid commands (2 marks)
  - Simultaneous connections (2 marks)
- control2310
  - Argument checking (4 marks)

  - Register with mapper (2 marks)
  - Connection by roc and `SIGHUP` (4 marks)
- roc2310
  - Argument checking (2 marks)
- roc + control
  - roc visits one destination (3 marks)
  - roc visits multiple destinations (4 marks)
  - invalid destinations (3 marks)
- roc + control + mapper
  - mapper connection errors (2 marks)
  - destinations needing mapping (7 marks)
  - destinations with mapping errors (3 marks)

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.

4.0 To 4.1

- Corrected error condition for status 4 of control2310.
- Corrected spacing in error messages.
- Note: IDs can contain any chars apart from those explicitly banned. eg: X@YZ is a valid ID.
- Added sub-headings above the exit conditions table
- Note: The mapper has no exit command and will run until killed.
- The way control2310 is ordered to output it's log of aircraft which have visited it. Previously this was triggered by a signal. Now this is triggered by sending a particular string. Note: previously it output to standard out, now it should send it back to the connection that sent "log".
- Added '-' (ie the minus sign) to show what dash is.
- Added "valid" to roc2310 description of what to do when destination does not send back (valid) information. If you are expecting input and the other party doesn't send anything, just wait, do not attempt to timeout.
- Cleaned up spelling and minor grammar issue.
- The maximum length for any individual ID or info text is 79 chars (excluding terminators like '\n'). What that means about the maximum length of input for different things is left as an exercise to the reader.