The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2019
CSSE2310 / CSSE7231 - Assignment 4
Due: 21:00pm 24th October, 2019
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 4.2

Purpose

In this assignment you will implement a networked simulation (described later). Your assignment will consist of a single peer-to-peer C99 program (`2310depot`). Multiple instances of this program will execute to form the network.

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.

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.

Simulation

Each process (a running instance of `2310depot`) will simulate a single storage depot which will store various resources. Resources can be deposited at the depots, withdrawn from depots (eg to represent sales) or transfered to other depots to meet regional demand.

The depots can be connected to other depots to represent an existing transport link between those locations.

All of the above operations are represented by messages sent to the depots via the network.

Invocation

For commandline parameters, your program will take its name followed by a sequence of pairs of resource names and quantities respresenting the starting stocks of goods at this depot. For example:
`2310depot garden sand 10 seeds 2`
Would start up a depot called "garden" with 10 units of sand and 2 units of seeds.

Neither depot names nor goods names may contain

- spaces
- '\n'
- '\r'
- colons

Names are not permitted to be empty. Starting quantities are not permitted to be negative (stocks could later become negative as a result of transfers).

After checking its arguments, the depot will listen on an ephemeral port (its "command port") and output that port number to `stdout` followed by a newline. You should listen on `127.0.0.1`.

For each connection it has, a depot needs to be able to:

- Receive and act on instructions coming from that connection.
- Send instructions to the depot at the other end of the connection.

Exits

All of the following messages should be ouput to `stderr`

Exit	Condition	Message
1	Incorrect number of arguments	`Usage: 2310depot name {goods qty}`
2	Empty name or name contains banned characters	`Invalid name(s)`
3	Quantity parameter is < 0 or is not a number	`Invalid quantity`

SIGHUP

If a depot receives `SIGHUP`, then print out its current (non-zero) stocks of goods (in lexicographic[1] order). It should then print the names of its neighbours (in lexicographic order). For example:

```
Goods:
Glue 2
Sand 10
Neighbours:
Central
LongPocket
```

[1] ie the order that strcmp gives

Messages

Message	Parameters	Action
`Connect:`*p*	*p* : port to connect to	Connect to another depot
`IM:`*p*`:`*name*	*p* : the command port of the depot sending the message *name* : name of the other depot	Another depot has connected to you and is introducing itself. Both parties should send one of these immediately after a new connection
`Deliver:`*q*`:`*t*	*q* : quantity of good *t* : type of good	Add the quantity of the good to your stocks.
`Withdraw:`*q*`:`*t*	*q* : quantity of good *t* : type of good	Remove the quantity of the good from your stocks.
`Transfer:`*q*`:`*t*`:`*dest*	*dest* : name of depot to send to	As withdraw but send a delivery message to the named depot for that quantity and type of goods.
`Defer:`*k*`:Deliver:`*q*`:`*t*	*k* : key to refer to later	As with `Delivery` but deferred until later
`Defer:`*k*`:Withdraw:`*q*`:`*t*	*k* : key to refer to later	As with `Withdraw` but deferred until later
`Defer:`*k*`:Transfer:`*q*`:`*t*`:`*dest*	*k* : key to refer to later	As with `Transfer` but deferred until later
`Execute:`*k*	*k* : key for deferred tasks to be executed	Carry out all deferred orders with key *k*

Notes:

1. All quantity parameters must be > 0.
2. If a message refers to a resource name you don't know about, treat it as if you had 0 of it.
3. If any badly formatted messages arrive, discard them and continue as normal.
4. Messages involving named depots are only valid if you are directly connected to that named depot.
5. Keys must be unsigned whole numbers.
6. When an operation is `Defer`-ed, the whole operation is held until later. So a defered transfer will not attempt to withdraw any goods until the relevant key is executed.
7. If a `Connect` message arrives, but you already have a connection to that port, silently ignore the message.

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.

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.

- for solo depot (Make sure `SIGHUP` works, since it will be used to mark most of these)
  - Argument checking (3 marks)
  - Deliver Message (3 marks)
  - Withdraw Message (2 marks)
  - Defer and Execute Messages (8 marks)
  - Mixture of Messages (4 marks)
- for multiple depots
  - Connect to other depots (2 marks)
  - Transfer (6 marks)
  - Deferred Transfer (4 marks)
  - Mixture of messages (12 marks)

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 $F + \min\{F, S\}$.

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.

Notes

1. If a depot recieves an invalid message (eg malformed, references a depot they don't know, ...), the depot should silently ignore that message.
2. If a connection closes continue normal operations.
3. Any withdraw or transfer instruction which references a good for which you have no stock, continue to process it.
   - eg: Depot has no record of a good called "treacle".
   - `Withdraw:10:treacle`
   - Depot now has `-10` treacle.
4. Depots should **not** shutdown on recieving `SIGHUP`.
5. You will need threads to ensure that each depot can process multiple messages simultaneously.

Tips and Fixes

4.1 → 4.2

1. Changed explanation of `IM` in table.
2. If a connecting depot sends an invalid message or a message other than `IM`, close the connection.
3. If your depot notices that a connection has closed. Act as if the connection was still there.
   (a) Do not remove that depot's entry from `SIGHUP` output.
   (b) Do not allow another connection to the same port.
   (c) Do not reject `Transfer` commands involving the name of that remote depot.
   (d) Internally you can deal with this however you want; but you should still not die due to `SIGPIPE`

4. If you are connecting with another depot, you should believe whatever its `IM` message tells you its port is (even if you connected to a different port).

5. We won't test duplicate depot names.