170D WOBC Module Q Practical 2B

170D WOBC Module Q Practical 2B

Overview

Create two programs, `seller` and `buyer`.

The `seller` application is a server than handles any number of connections (up to operating system limits) from the `buyer` application.

When the `buyer` application is executed, it connects to the `seller` and begins sending transactions for the `seller` to clear.

It is guaranteed that the number of transactions handled by the `seller` will be less than `UINTMAX`.

Requirements

1. The `seller` creates five accounts (with ids 1 through 5) using a structure that contains at least these fields (requirements or extra credit features may dictate additional fields):

   ```
   typdef struct {
       int amt_owed;
       int num_orders;
       int num_payments;
   } my_account;
   ```

2. With up to 8 `buyer` processes running simultaneously, the `seller` must not drop any transactions sent to it.

3. The `buyer` processes will send no less than 50,000 transactions and must have a one second delay every 10,000 transactions — give or take up to 100 transactions — but not more often than that. (No other intentional or programmed delays are acceptable.)

4. The `buyer` program will be executed with a single filename on the command line. That file contains one transaction per line, account id and amount.

5. The `buyer` must send an account id and an amount for each "transaction". Transactions can be sent separately or in batches of up to no more than 300 at a time.

6. Each `buyer` process must send transactions for at least two different accounts.
7. Both `seller` and `buyer` must exit cleanly upon receiving either `SIGINT` or `SIGTERM`, including any end of execution reporting as described next.
8. The `seller` must print the following information for each account upon termination:
    - Use this `printf()` format `%-20s %u %7d %5u %5u\n` and these fields:
        - input filename (use "network" for the `seller`)
        - account id
        - amount owed
        - number of deposits
        - number of withdrawals
9. A zero amount should be treated as an empty transaction and ignored by both `seller` (if received) and `buyer` (if read).
10. Each `buyer` will print a subtotal of the value of all transactions for each account, the number of orders and payments it makes (should be equal), and a grand total across all accounts (use the same format as the `seller` report, above).
11. The totals printed by the `buyer` processes must match the totals printed by the `seller` process.
12. When the `seller` terminates, all `buyer` processes must also terminate (ignoring any pending transactions).

Extra credit features:

1. Add the `-p` option to keep a performance counter on the `seller` that reports the number of transactions per second being received from each `buyer`.
2. Add the `-p` option to keep a performance counter on the `buyer` that reports the number of transactions per second being sent.
3. The `seller` should calculate the standard deviation across each account and across all accounts and report those values when it reports the totals at termination.

DIcE Rubric

Document	Design Plan	Does the design plan provide a clear general overview of the project?	3%
		Is the design plan easy to understand?	2%
	Test Plan	Are test cases detailed enough to repeat easily?	2%
		Are expected results stated clearly?	2%
		Are requirements adequately covered by test cases?	1%
	Project Writeup	Does the writeup document challenges and successes encountered?	2%
		Does the writeup document any lessons learned?	3%
	Writing	Is the project free of grammatical and spelling errors?	4%
		Is non-code formatting consistent?	1%
	Code Formatting	Does `indent -linux` produce no warnings?	4%
		Are appropriate names chosen to enable code readability?	2%
		Are comments added where appropriate and aid understanding of the logic?	2%

		Is any outside code cited appropriately?	2%
	Total		**30%**
	Version Control	Does the project have the correct name and default branch?	1%
		Were commits broken down into appropriate scopes?	3%
		Are commit messages simple and informative?	1%
Implement	Architecture	Are effective and efficient data structures used?	5%
		Was the code designed and constructed in a modular fashion?	5%
		Were generally sound decisions made with regard to architecture?	10%
	Testing	Does the program include robust unit tests?	4%
		Do all automated tests pass under `make check`?	1%
	Total		**30%**
	Safety	Does the program avoid crashing or infinite loops, even on invalid input?	5%
		Does `valgrind` report no errors or warnings?	5%
		Does the program build with no warnings?	5%

Execute	Builds	Do all required build targets get built correctly?	5%
	Requirements	Were all inputs parsed correctly and yield the correct output?	5%
		Are all other requirements met?	10%
	Performance	Does the program scale appropriately with input and data?	4%
		Does the program execute in a timely manner?	1%
	Total		**40%**

Requirements

Area	Requirement
Document	All documentation must be in PDF format unless otherwise specified.
Document	All documentation must be located in a `doc/` folder at the top level of the project.
Document	The design document must be located in `doc/design.pdf`
Document	The test plan must be located in `doc/testplan.pdf`
Document	The project writeup must be located in `doc/writeup.pdf`
Document	All code must match the Linux kernel style guide, with the exception of blocks *always* having braces.
Implement	Project must be stored in the assigned VCS account, under the project name `buyer-seller`.
Implement	No third-party header files/libraries may be used unless signed off by the Program Manager or Instructor.

Implement	Project must use appropriate data types or structures.
Implement	All automated tests and test code must be located in a `test/` folder at the top level.
Implement	Project must provide appropriate automated unit tests.
Implement	All error messages *must* be sent to stderr, and only output should go to stdout. (When an application is executed with `-h`, the result goes to stdout. When the usage message is displayed because of an error, the output should go to stderr.)
Implement	The `seller` program must record amount owed for five different account ids, numbered 1 through 5.
Implement	The `seller` must accept connections from up to 10 `buyer` processes.
Implement	Each `buyer` process will send 50,000 transactions to the `seller`. There must be a 1 second delay at intervals of 10,000 transactions — give or take 100 transactions — but not more often than that. (No other intentional or programmed delays are acceptable.)
Implement	Each transaction consists of an account id and an amount (negative values are payments, positive values are orders).
Implement	Transactions may be sent separately or batched into groups of 300 or less.
Implement	Each `buyer` must send transactions for at least two different account ids (a single `buyer` cannot process just a single account id).
Implement	Both `seller` and `buyer` must exist cleanly upon receiving either `SIGINT` or `SIGTERM`, including adhering to any reporting requirements.
	The `seller` must report the following information for each account at termination using the `printf()` format `%-20s %u %7d %5u %5u\n`:

Implement	• input filename (use "network" for `seller`) • account id • amount owed • number of deposits • number of withdrawals
Implement	The `buyer` must report the same information as the `seller` at termination, but only for the transactions that it sends.
Implement	The `seller` totals must match the `buyer` totals. When using the provided transaction files, all balances should be zero.
Implement	When the `seller` terminates, all `buyer` processes must terminate (ignoring any pending transactions).
Execute	Project must build and run on the class machine.
Execute	Project must not crash or get stuck in an infinite loop, even on invalid input.
Execute	Project must build both `buyer` and `seller` in the top-level directory in response to `make`.
Execute	Project must build both `buyer` and `seller` with debugging symbols in response to `make debug`.
Execute	Project must build both `buyer` and `seller` with profiling information in response to `make profile`.
Execute	Project must build and run any automated tests in response to `make check` (which may assume the program has already been built).
Execute	Project must clean all project-generated files in response to `make clean`.
Execute	Project must build against C18 (`std=c18`)
Execute	Project must build with no warnings from `-Wall -Wextra -Wpedantic -Waggregate-return -Wwrite-strings -Wvla -Wfloat-equal`

Suggested Extra Credit Features

Area	Feature	+
Document	Write a `man(1)` page to document the program. Documentation must show at least one example command line for each implemented feature.	+2
Implement	Add a `-p` option to the `seller` program that prints the transactions per second from each `buyer` process.	+5
Implement	Add a `-p` option to the `buyer` program that prints the transactions per second sent to the `seller`. Do NOT include any processing time required to read/generate the transaction data.	+3
Implement	The `seller` should calculate standard deviation across each account and across all accounts, and report those values when it reports the totals at termination. Add column five to the `printf` format and display the results using `%5.2f`.	+2

Last updated 2023-12-07 14:00:10 UTC