Project 1 Report

Overall Status: Complete, nothing left unimplemented

Implemented Correctly

Everything assigned

Difficulties Encountered

1: Issues in skeleton code

The provided skeleton code had multiple spacing, output, log output, log name, ... provided in it. I will list the changes made here, none make any impact other than making stdout outputs more readable for us:

- avoided printing duplicating information when read from a file.
- fixed openlog() so logfilename is printed correctly.
- changed spacing of print statements such that data relating to one operation is grouped together. (removing duplicate \n)
- changed the order of functions in zgt_tx to make the code more readable and to outline our work
- made commit print out correct transaction id. Only effected print, not execution

These changes were all in zgt_test.c, so they are not part of the submitted code, but they were helpful to enhancing readability of the printed information.

As a last bit of complaint about the skeleton code. It is not C++, it is a C/C++ amalgamation. This is not uncommon in projects that started as C and became C++, but it makes reading the code significantly more difficult.

2: Interacting with the skeleton was far harder than the logic of the implementation

The logic of each transaction function is relatively simple given the information available in the PowerPoints and lectures. The vast majority of the challenge of this project was figuring out how to use semaphors for locking, where to find stored information (specificly by way of code, not necessarily the data structures), etc. as opposed to when those things need to be done.

If the intent of the project was for that to be the real challenge, we understand, however it was increadibly frustrating.

File Descriptions

No new files were created.

No new functions were created.

Division of Labor

- Landon Moon
    - 2/29/2024 - 1 hour - Learning the requirements

    - 3/07/2024 - 3 hours - Learning the skeleton
    - 3/07/2024 - 3 hours - Fixing and cleaning up provided code
    - 3/13/2024 - 1 hour - Made transaction operations run in order
    - 3/13/2024 - 2 hours - Found where object values are located. read code
    - 3/14/2024 - 2 hours - Completed perform_..._operation()
    - 3/15/2024 - 2 hours - First implementation of gaining read locks
    - 3/15/2024 - 1 hour - refactor. possibly done
    - 3/16/2024 - 3 hours - memory leak investication
- Jacob Holz
    - 2/29/2024 - 1 hour - Learning the requirements
    - 3/07/2024 - 3 hours - Learning the skeleton
    - 3/14/2024 - 2 hours - Learning the new code and planning next steps (didn't know the next time I looked, the code would be done)
    - 3/15/2024 - 3 hour - Debugging and Testing on Omega
    - 3/15/2024 - 2 hours - Report Writing
    - 3/15/2024 - 1 hour - Debugging and Testing on Omega
    - 3/16/2024 - 1 hours - Finishing Report

Logical errors and how you handled them

Most logical errors were avoided before they cropped up through the use of pair programming, and reviewing line by line.

- Logical Error 1: Reading also decreases the value?

  Before we ran tests for the first time, we noticed that if the read operation is decrementing the value of the object (and the example logs show that it does), it need to have an exlusive lock to avoid race conditions. This feels very odd as it introduces deadlocks into cases that should logically be free of them if reads were not exclusive.

- Logical Error 2: Alternative solution to Logical Error 1

  Another option would be to simply ignore the logical error and allow the very rare race conditions. Theoretically, the odds of a race condition are extremely low in the given context. This may be the intended solution as deadlocking is far more likely than a race condition due to S2PL with lots of exclusive locks. This is the state that the program was in at the time of submission. The commit that reverted the solution to the logical error in favor of avoiding illogical deadlocks is here:

- Logical Error 3: Library memory leak instability

  The provided code and library from the skeleton code NEVER frees any dynamically allocated memory. This causes the operating system to mistake memory values if executions of the program are done back-to-back. Time must be given for the operating system to clean up the programs unfreed memory. This was tested extensivly and was determined through multiple means:

    - tid values passed to a given thread could be non-existent in the txt file. Causing an operation of an invalid tid to try to gain an invalid mutex lock. This was confirmed multiple times and causes the thread with the invalid tid to hang the entire program. The int returned from string2int is incorrect.

  - malloc calls fail more often directly after a previous execution. The skeleton code rarely tests for a successful malloc call, instead it segfaults due to trying to access a null pointer. The dynamic memory allocation is located in the library and is not our responsibility to rewrite the entire framework from scratch. This occurs primarily during param creation. Again this was tested by printing out the ptr value from malloc and it is occasionally null which will always cause a segfault if not checked for. Additional checks were put in place to mitigate this.

- Logical error 4: remove_tx() never removed a transaction

  a bug was found in remove_tx() which caused it to never remove a transaction. This was due to both pointers always pointing to the same transaction. When the transaction found nextr it is set to nextr which does nothing. additionally the special case for the first transaction was not originally considered by the skeleton code. It would have required double pointers with the original method.

- Logical error 5: general logical errors with the skeleton code

  There are still a bunch of other smaller logical errors that aren't worthy of an entire section. This section is a general complaint with the state of the skeleton code at the beginning of the project. Many C++ coding standards could have prevented the vast majority of bugs in the provided library. Due to not rewriting every line provided in the skeleton code, there are bound to be errors beyond our control.