

CS731 Software Testing Project

Mutation Testing Analysis

ReadME

Concurrent Library Management System

Team

Uttam Hamsaraj (IMT2022524)

Pranav Laddhad (IMT2022074)

Overview

This report explains how we designed, ran and analyzed mutation testing for a multi-threaded Library Management System (LMS) written in C. The main goal was to check how good and reliable our CUnit test suite is. We did this by introducing small, intentional errors (mutants) into the code and seeing whether the tests could find and eliminate them. This helped us understand how well the tests work at both the unit level (unit tests) and when different parts of the system interact (integration tests).

1 Project Overview and Environment

1.1 System Under Test (SUT)

The **Library Management System (LMS)** serves as the System Under Test (SUT).

Source Code ([GitHub link](#))

1.2 Overview of the LMS

The project implements a Library Management System in C using socket programming and various system calls. It follows a **client–server architecture**, where the server processes requests from multiple clients simultaneously.

Concurrency is handled through **pthreads**, allowing multiple client operations such as borrowing or returning books, to execute smoothly in parallel. Library data (e.g., `books.txt`, `members.txt`) is stored in flat files and protected using advisory file locking (`flock`) to ensure safe access during concurrent operations.

This architecture allows both administrators and regular users to interact with the system efficiently, with administrators managing library resources and users performing borrowing/return actions in real time.

Source codes used:

- **server.c** – Implements server logic, request handling, concurrency control, and file operations.
- **client.c** – Provides the user-facing client interface for both administrators and regular users.

1.3 Testing Environment and Setup

- **Unit/Integration Framework:** CUnit was used to develop and execute all test cases.
- **Execution Tool:** The final system was prepared for automation using the **Mull** mutation testing framework.
 - (This component did not function as expected, as discussed in the following section.)
- **Implementation Details:** All core server functions were isolated via **wrapper functions** to bypass socket I/O, allowing for direct unit/integration testing of the file manipulation logic.

Codes used: The current implementation, including all wrappers and tests, is available at : ([GitHub link](#))

Key Project Files Used for Testing:

- **server.c: Server Logic.** Contains the original application code and all custom wrapper functions used to isolate critical functions for testing.

- `test_server.c`: **Test Oracle.** Contains the CUnit setup and the logic for the CUnit test cases designed to kill the defined mutants.
- **Makefile: Build System.** Takes care about the entire process: compilation with `clang`, linking with CUnit libraries and generating the necessary file formats for analysis.

2 Methodology: Mutation Testing & Verification

The test suite's quality was evaluated by verifying its ability to kill specific types of mutants required by the project:

- **Unit level mutants** (testing internal algorithms/assignments)
- **Integration level mutants** (testing system I/O, concurrency)

2.1 Test Oracle Stability

The test suite was verified for stability by running the original, non-mutated code. All designed test cases returned PASS establishing a clean baseline.

2.2 Final Mutant Categorization and Kill Analysis

Different high-quality killer tests were designed covering the different mutation operators.

Table 1: Unit-Level mutants (Isolated Logic)

Test ID	Level	Operator	Kill Rationale (Proof of Efficacy)
UT1	Unit	AOR	Mutated <code>+</code> to <code>-</code> (e.g., <code>6 → 4</code>). Test failed because ID calculation was corrupted.
UT2	Unit	ROR	Mutated <code>==</code> to <code>!=</code> in <code>strcmp</code> check. Test failed because correct credentials resulted in authentication failure.
UT3	Unit	ROR	Mutated <code>is_rented == 1</code> to <code>!= 1</code> . Test failed by allowing an unrented book to be “returned,” breaking data integrity.
UT4	Unit	SDL	Deletion of status preservation line. Test failed because the Rented Status was lost after modification.

Table 2: Integration-Level mutants (Dependencies)

Test ID	Level	Operator	Kill Rationale (Proof of Efficacy)
IT1	Integration	VRR	Mutated <code>remove(temp)</code> to <code>remove(books.txt)</code> . Test failed because it exposed catastrophic file deletion upon non-existent request.
IT2	Integration	COR	Mutated <code>&&</code> to <code> </code> . Test failed by allowing the system to attempt renting a non-existent book due to faulty conditional logic.
IT3	Integration	SCR	Deletion of <code>O_CREAT</code> flag in <code>open</code> call. Test failed because the system could not create the primary data file, proving OS resource dependence.
IT4	Integration	Robustness	Asserted system survival when file data was corrupted, validating <code>sscanf</code> integration robustness.

2.3 Verification Evidence: Unit Test 1 (AOR Mutant)

Here, we have explained in brief the overall working of the first unit test case which is an instance of the Arithmetic Operator Replacement (AOR). Here mutant targets the core arithmetic function responsible for generating unique IDs.

A. Source code targeted and mutation applied

The mutation was applied within the `get_next_id` helper function in `server.c`.

```

203 int get_next_id(const char *filename)
204 {
205     FILE *file = fopen(filename, "r");
206     if (!file)
207         return 1;
208
209     int id = 0;
210     char buffer[BUFFER_SIZE];
211
212     while (fgets(buffer, BUFFER_SIZE, file))
213     {
214         int current_id;
215         sscanf(buffer, "%d", &current_id);
216         if (current_id > id)
217         {
218             id = current_id;
219         }
220     }
221
222     fclose(file);
223
224     // Original Code (must be changed to create the mutant)
225     return id + 1;
226
227     // MUTANT CODE: Change '+' to '-'
228     return id - 1;
229 }
```

Figure 1: `get_next_id` helper function in `server.c`

As shown in Figure 1, we specifically changed the addition operation to subtraction.

B. Killer test case

The test case was generated using the get `test_kill_aor_mutant` function in `test_server.c`.

- The CUnit test case ensures that the function correctly returns the next sequential ID (Max ID +1) after reading a seeded file.

```
55 void test_kill_aor_mutant(void) {
56     unlink("books.txt");
57     // 1. Test case 1: Clean file. Expected: 1
58     CU_ASSERT_EQUAL(get_next_id("books.txt"), 1);
59
60     // 2. Create a dummy record with a known high ID (e.g., ID 5)
61     FILE *f = fopen("books.txt", "w");
62     if (f) {
63         // Write the highest ID as 5
64         fprintf(f, "5 TitleA AuthorA 0\n");
65         fprintf(f, "1 TitleB AuthorB 0\n");
66         fclose(f);
67     }
68
69     // 3. Test case 2: File is NOT empty.
70     // Original Code (id + 1) expected: 5 + 1 = 6
71     // Mutant Code (id - 1) expected: 5 - 1 = 4 <-- This should fail the test
72
73     // The test asserts that the result MUST be 6.
74     CU_ASSERT_EQUAL(get_next_id("books.txt"), 6);
75 }
```

Figure 2: `test_kill_aor_mutant` function in `test_server.c`

As can be seen in Figure 2, we explicitly write in `books.txt` to add a new book with ID 5 (this is the most recent book). So, when `get_next_id` works it takes `current_id` as 5 and then `id` as 5.

C. Execution results and kill rationale

- **Pass Case** (original Code): When `return id + 1;` is executed the function returns 6. The assertion passes confirming the baseline.
- **Fail Case** (mutant Code): When `return id - 1;` is executed the function returns 4 ($5 - 1$). The test fails because $4 \neq 6$.

D. Resulting failure output (mutant killed):

Suite: Server File Logic Tests

Test: UT1: AOR Mutant - Get Next ID ...FAILED

1. `test_server.c:XXX - CU_ASSERT_EQUAL(get_next_id("books.txt"), 6)`

E. Rationale:

The mutant was killed because the injected bug (subtraction instead of addition) caused the **algorithmic requirement** (generating the next unique ID) to fail, proving the test suite is sensitive to calculation errors.

-As shown in Figure 3 and Figure 4 we can see that in normal version the test case passes but in mutated version it fails.

```

Suite: Server File Logic Tests
Test: UT1: AOR Mutant - Get Next ID ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
                suites   1      1     n/a      0       0
                  tests   1      1      1      0       0
                  asserts  2      2      2      0     n/a

Elapsed time =  0.000 seconds
● prana@Pranav's-MacBook-Air LibraryManager % make

```

Figure 3: UT1: AOR Mutant - Get Next ID ...passed

```

Suite: Server File Logic Tests
Test: UT1: AOR Mutant - Get Next ID ...FAILED
  1. test_server.c:74  - CU_ASSERT_EQUAL(get_next_id("books.txt"),6)

Run Summary:    Type  Total    Ran Passed Failed Inactive
                suites   1      1     n/a      0       0
                  tests   1      1      0      1       0
                  asserts  2      2      1      1     n/a

Elapsed time =  0.000 seconds
○ prana@Pranav's-MacBook-Air LibraryManager %

```

Figure 4: UT1: AOR Mutant - Get Next ID ...FAILED

2.4 For Other Tests as Well

-As shown in Figure 5 all test cases pass when the original, non-mutated code is executed.

```

--- Running CUnit Tests ---
./test_runner

CUnit - A unit testing framework for C - Version 2.1-3
http://cunit.sourceforge.net/

Suite: Server File Logic Tests
Test: UT1: AOR Mutant - Get Next ID ...passed
Test: UT2: ROR Mutant - Auth Check ...passed
Test: UT3: ROR Mutant - Return Unrented ...passed
Test: UT4: SDL Mutant - Status Preservation ...passed
Test: IT1: VRR Mutant - Delete File Integrity ...passed
Test: IT2: COR Mutant - Rent Logic Integrity ...passed
Test: IT3: SCR Mutant - File Creation Integrity ...passed
Test: IT4: Data Parsing Robustness ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
                suites   1      1     n/a      0       0
                  tests   8      8      8      0       0
                  asserts 15     15     15     0     n/a

Elapsed time =  0.002 seconds
● prana@Pranav's-MacBook-Air LibraryManager % make clean

```

Figure 5: Output when tests are executed with non-mutated code

For each of the other tests, we can present similar information, including:

- The mutated parts of the code
- Wrapper and helper functions
- Test case generation and assertions
- Logic, rationale, and expected outputs

3 Mutation Framework Roadblock

In order to automate the process of injecting thousands of mutants and generating a **mutation score** we attempted to integrate the **Mull** (open-source) mutation testing framework. This tool relies on processing the program's compiled code in the LLVM Intermediate Representation (IR) format.

Framework Dependency Issues The automated execution phase was permanently blocked by severe external library instability:

- **Version Conflict:** The specific pre-compiled Mull runner (`mull-runner-19`) relied on a hardcoded outdated dependency on the **LLVM library** (`llvm@19`).
- **Fatal Failure:** Attempts to run the framework resulted in a **Dynamic Linker error** (`dyld`) and process termination as the system failed to locate the specific necessary libraries. This demonstrated a fatal version conflict.

```
prana@Pranav-MacBook-Air LibraryManager % make mull
Compiling server.c to LLVM IR...
clang -c server.c -o server.bc -emit-llvm -Wall -Wextra -std=c99 -I/opt/homebrew/opt/cunit/include -g -O0
--- Running MULL Mutation Testing ---
# FINAL COMMAND: Uses the specific runner name and correct argument structure
/opt/homebrew/bin/mull-runner-19 server.bc --test-program=./test_runner
[info] Using config /Users/prana/Desktop/SEM7/ST/project/LibraryManager/mull.yml
[error] Cannot create binary object from: /Users/prana/Desktop/SEM7/ST/project/LibraryManager/server.bc: The
[error] Error messages are treated as fatal errors. Exiting now.
make: *** [mull] Error 1
```

Figure 6: Example of issue faced while trying to use MULL

- We decided to not use MULL framework.
- As a result, the mutation score could not be calculated.

However, the test mutants executed successfully for the provided test cases.