Project Debugging and Refinement Report

Do Hung Anh - 23BI14015

November 23, 2025

1 Summary of Work Completed

This report details the process of debugging and refining a client-server application written in C. The initial goal was to diagnose and fix compilation warnings and runtime errors. The project consisted of a client and server program that were intended to exchange messages over a network socket. Through a systematic process of analysis and code modification, several layers of bugs were identified and resolved, leading to a fully functional application.

The key accomplishments include:

- Resolving all compilation warnings related to implicit function declarations.
- Correcting multiple runtime logic errors that caused incorrect behavior and deadlocks.
- Hardening the server against common socket errors like "Address already in use".
- Successfully enabling the intended two-way communication between the client and server.

2 Challenges Encountered

The debugging process uncovered several distinct challenges, ranging from simple declaration mismatches to complex logical deadlocks.

2.1 Implicit Function Declarations

The compiler issued warnings for `mon_read` and `mon_write` despite the relevant header files being included. The root cause was a mismatch between the function names in the header files (`read.h`, `write.h`) and the names used in the application code.

2.2 Incorrect Error Logic in `mon_write`

The server would output a confusing "write syscall failed: Success" message. This was traced back to an inverted logic check in the `mon_write.c` function, where the error-reporting code was placed in the `else` block, which is executed upon a successful syscall.

2.3 Socket Binding Failure (`EADDRINUSE`)

The server frequently failed to start, reporting that the "Address already in use". This is a common issue in socket programming where a recently closed socket remains in a `TIME_WAIT` state, preventing immediate reuse of the port.

2.4 `const` Type Mismatch

A later compilation error, "conflicting types for 'mon_read' ", arose from a type mismatch in the function's signature between its declaration in the header and its definition in the C file. The buffer parameter was incorrectly declared as `const`.

2.5 Communication Deadlock

This was the most critical issue. After earlier bugs were fixed, the client and server would connect successfully but then hang, never exchanging data. The cause was a flawed implementation of `mon_read`, which looped until its entire read buffer was full. Since both client and server sent messages far smaller than the buffer size, they would wait indefinitely for data that would never arrive.

3 Solutions and Fixes Implemented

Each challenge was addressed with a specific, targeted solution.

3.1 Header and Logic Corrections

To fix the declaration warnings, the function prototypes in `read.h` and `write.h` were renamed to `mon_read` and `mon_write`. To fix the incorrect error message, the logic in `mon_write.c` was corrected to properly check for a negative return value from the syscall and report errors only when they actually occurred.

```
ssize_t n = syscall(__NR_write, fd, ptr + total, count - total);
if (n < 0) {
    if (errno == EINTR) { // Interrupted by a signal
        continue; // Retry the syscall
    }
    perror("write syscall failed");
    return -1; // An actual error occurred
}
total += n;
```

Listing 1: Corrected error handling in `mon_write.c`

3.2 Enabling Socket Reuse

The "Address already in use" error was resolved by setting the `SO_REUSEADDR` socket option on the server's listening socket. This allows the kernel to reuse the port immediately, which is essential for rapid development and server restarts.

```
int yes = 1;
if (setsockopt(host, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1) {
    perror("setsockopt");
    mon_close(host);
    return 1;
}
```

Listing 2: Adding `SO_REUSEADDR` to `server.c`

3.3 Resolving the Deadlock

The deadlock was fixed by fundamentally changing the behavior of `mon_read`. The loop that waited for a full buffer was removed. The new implementation performs a single `read` syscall (while still handling `EINTR`) and returns immediately with whatever data was read. This change makes the function behave like the standard library `read()`, preventing the application from getting stuck.

```
ssize_t mon_read(int fd, void *buffer, size_t count){
    ssize_t n;
    // Loop to handle EINTR (interrupted system call)
    do {
        n = syscall(__NR_read, fd, buffer, count);
    } while (n < 0 && errno == EINTR);

    if (n < 0) {
        perror("read syscall failed");
    }
    return n;
}
```

Listing 3: New deadlock-free implementation of `mon_read.c`