Preventing Race Condition in Multi-Agent Programming

Dongjae Lee

2026.02.12

Abstract

Multi-agent programming in coding assistants, such as Cursor and Claude Code, enhances task efficiency and reduces context contamination. However, concurrent access to shared resources often leads to race conditions. To address this, we propose a reader-writer lock mechanism inspired by Rust's memory safety model to ensure data integrity during multi-agent collaboration.

While multi-agent programming solves context limitations and enables complex tasks, it introduces a critical risk of race conditions. Agents operating on stale contexts may generate incorrect patches or overwrite each other's changes. This conflict often results in infinite modification loops as agents repeatedly attempt to fix errors introduced by concurrent edits on intertwined code. Consequently, the efficiency of multi-agent programming is seriously degraded.

To address this problem, we propose a reader-writer lock mechanism to prevent race conditions. Our lock acquires locks based on semantic relevancy in the code, preventing two agents from editing the same related region simultaneously. When an agent submits the region it wants to modify, locks are placed on the related code, and other agents cannot read or edit those sections. Modifying any region without holding the lock is prohibited as well.

If an agent has already read a file, other agents may still read it but cannot obtain a write lock for that file. In other words, code that is currently referenced by another agent cannot receive a write lock.

In summary, unreferenced code can receive a write lock, in which case both reads and writes by other agents are prohibited. Conversely, code referenced by another agent allows reads but prohibits writes. We expect this mechanism effectively prevents race condition while maintaining the efficiency of multi-agent programming.