What Change History Tells us About Thread Synchronization

Presentor: Yulin Che, 20292673

Content

Conference : FSE 2015

Title : What Change History Tells Us about Thread Synchronization

Concepts :

- Concurrency Analysis
- Repository Mining
- Over-Synchronization
- Concurrency Bugs (Lack-of-Synchronization)

Key words: Locks, Empirical Study, Repository Mining, Concurrency Bugs, Performance Bugs, Multi-Threaded Software

Research Type: Empirical Study

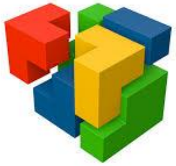
Input: Repository information(Version Control Tool), Bug Report, Revision Log

Output: Relationship of Concurrency Bugs, Performance Bugs and Code Revisions(Statistical & Empirical)



Regular Expression Analysis

Manual Checking

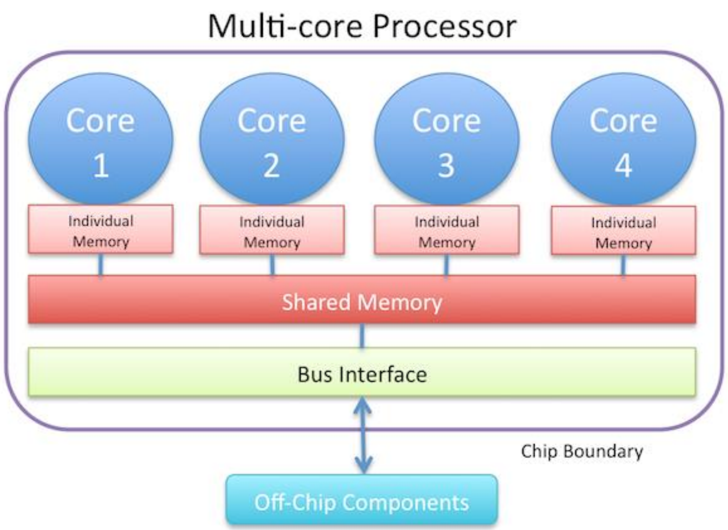



Content

Background

Concurrency

Physically



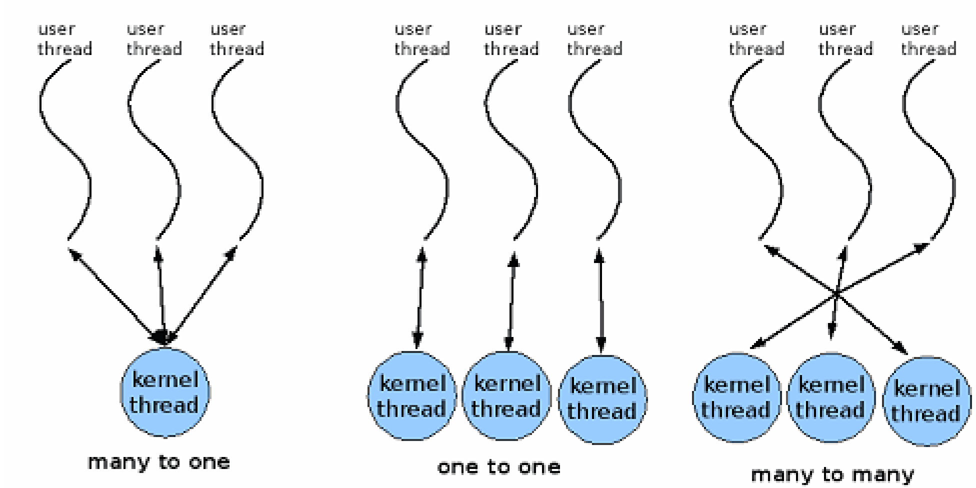
Shared Variable Access

Critical Section

1) Race Condition

Logically

2) Other Synchronization



Iteration Dependency

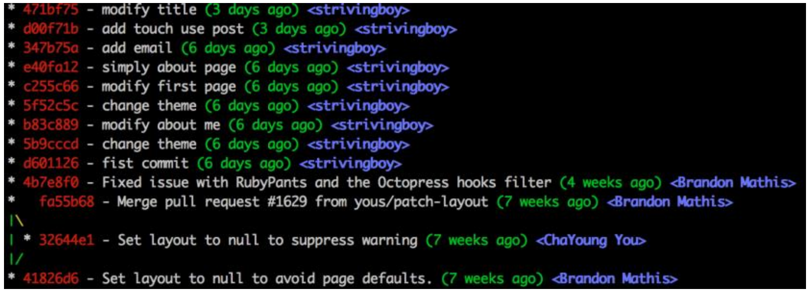
Barrier

Motivation: documents are not complete…

- Bug report is not able to cover all things
- Bug information is scattered throughout the multiple versions of code repositories
- Bug information is hidden in the long repositories' commit log, issues, some pull-requests

Challenge: millions of lines of codes for one snapshot

Opportunity:



```
* 471bf75 - modify title (3 days ago) <strivingboy>
* d00f71b - add touch use post (3 days ago) <strivingboy>
* 347b75a - add email (6 days ago) <strivingboy>
* e40fa12 - simply about page (6 days ago) <strivingboy>
* c255c66 - modify first page (6 days ago) <strivingboy>
* 5f52c5c - change theme (6 days ago) <strivingboy>
* b83c889 - modify about me (6 days ago) <strivingboy>
* 5b9cccd - change theme (6 days ago) <strivingboy>
* d601126 - fist commit (6 days ago) <strivingboy>
* 4b7e8f0 - Fixed issue with RubyPants and the Octopress hooks filter (4 weeks ago) <Brandon Mathis>
*   fa55b68 - Merge pull request #1629 from yous/patch-layout (7 weeks ago) <Brandon Mathis>
|\
| * 32644e1 - Set layout to null to suppress warning (7 weeks ago) <ChaYoung You>
|/
* 41826d6 - Set layout to null to avoid page defaults. (7 weeks ago) <Brandon Mathis>
```

- Commitment logs or revision logs consist of much smaller number of code lines
- History is important for our concurrency bug analysis

Incremental Analysis

Case Studies:

- Mysql
- Mplayer
- Mozilla
- Apache HttpServer

The key problem consists of two parts

- how critical sections are changed to solve performance problems (i.e. over-synchronization issues)

- how software changes lead to synchronization-related correctness problems (i.e. concurrency bugs).

Content

Work: Empirical General Study, Repository Mining

Observations:
- Frequent modifications of critical sections
- Changes of critical sections become stable with aging
- Fixing correctness bugs are more frequent than fixing over-synchronization to improve performance

Case Studies:
- Over-synchronization & Concurrency-bug
- Significance of incremental revision analysis, 50% bugs introduced under old context & 50% bugs introduced with new introduced shared variables

Content

1、 Overview

2、 Background

3、 Contribution

4、 Methodology

5、 Observation

6、 Conclusion

Structural Pattern Category

Add	Adding *CS*es
Add_{All}	Synchronization and body added together
Add_{Syn}	Synchronization introduced after body
Rem	Removing *CS*es
Rem_{All}	Synchronization and body removed together
Rem_{Syn}	Synchronization removed alone
Mod	Modifying existing *CS*es
Mod_{Body}	Critical section body modified
Mod_{Syn}	Critical section synchronization modified
Mod_{SynV}	Synchronization variable modified
Mod_{SynP}	Synchronization primitive modified
Mod_{SynB}	Critical section boundary moved
Mod_{SynS}	Critical section split
Mod_{SynU}	Adding unlock operations

Addition

Removal

Modification

Purpose Category

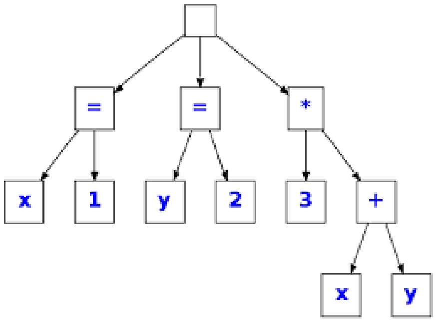
Correctness	Fixing functional bugs
Functionality	Adding or changing code functionality
Maintainability	Code refactoring
Performance	Improving performance
Robustness	Adding sanity checks

Correctness

Performance

Approach:

- Regular expression tools,
    - simply keyword search lock, latch, mutex, special case dealing with: C++ RAII
- Manually Read Bug Reports



None of these:

- No AST
- No control-flow, No pointer-alias

e.g, not deal with : lock(p1) unlock(p2), lock in this, unlock in another invoked function

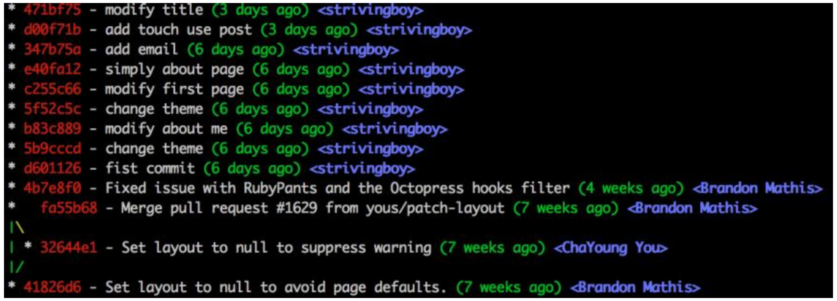
Arguments:

- even thought it is not sound
- false negatives, not much 5%

Content

Observations(Structural Pattern Oriented)

- How many changes happen: frequent modifications of critical sections
- When did changes happen: changes of critical sections become stable with aging
- Why did changes happen: fixing correctness bugs(31%) are more frequent than fixing over-synchronization(8.5%) to improve performance, extracted from the structural pattern and sampled cases manually checking



Arguments:

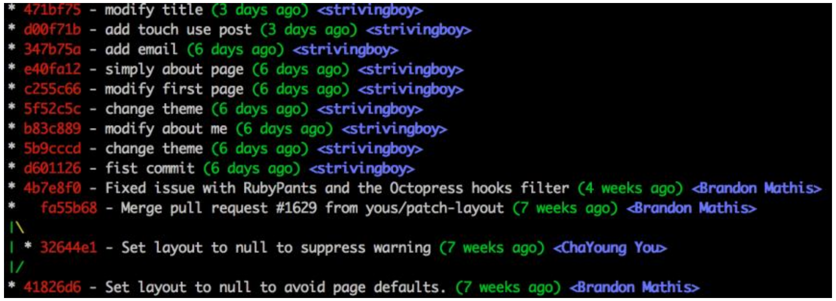
- History is important…

Observations(Structural Pattern Oriented)

- ModSync Split
  - Split Scopes of Critical Section Body
- ModSync Variable
  - Introduce New Variables
- ModSync Boundary(Scope)
  - Shrink Scopes

Arguments:

- Feasible to develop tools to automate part of over-synchronization detection and fix it
- history is useful



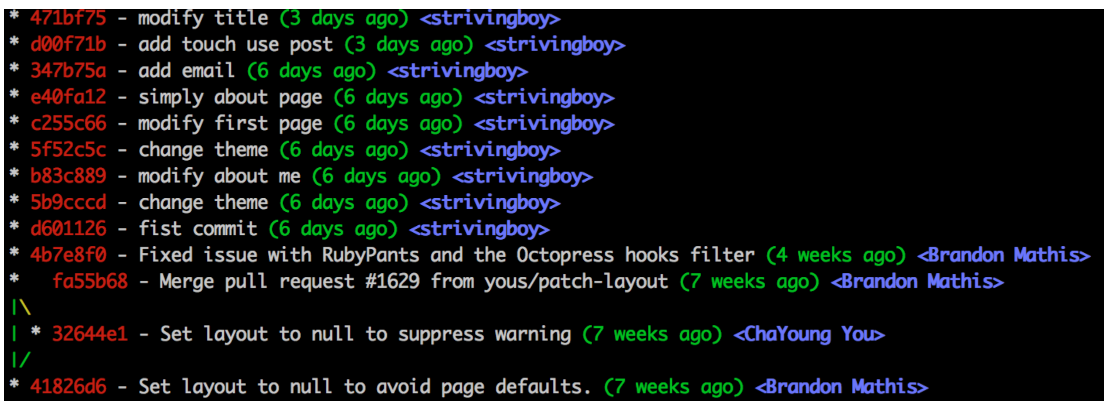
Content:

- Focus:
    - shared variables
    - instructions access them
    - synchronization-context, i.e, locked body and preceding barriers

- Manually check in this part
    - bug reports(28 reports)
    - part of sampled revision logs

Observations

- with old synchronization context information, 50% of studies bugs would require no new synchronization analysis to be detected
- memory-access analysis be simplied, 50% for new bugs, only analyze the changed codes



Arguments:

- History is important…

Content

1、Overview

2、Background

3、Contribution

4、Methodology

5、Observation

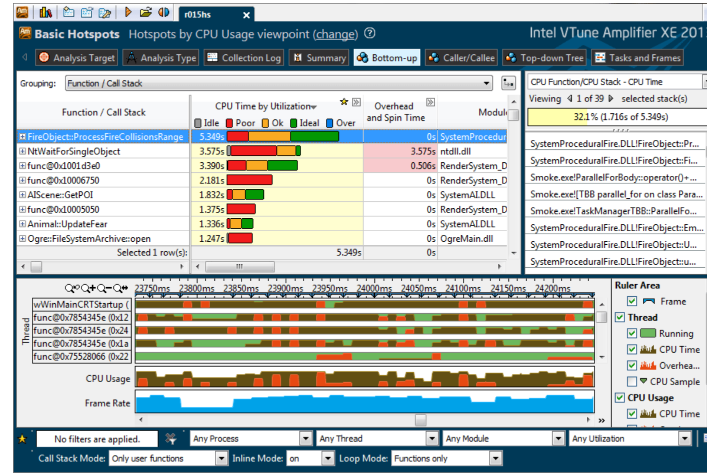
6、Conclusion

Conclusion

Suggestions

- Please keep trace of code-revision history

```
* 471bf75 - modify title (3 days ago) <strivingboy>
* d00f71b - add touch use post (3 days ago) <strivingboy>
* 347b75a - add email (6 days ago) <strivingboy>
* e40fa12 - simply about page (6 days ago) <strivingboy>
* c255c66 - modify first page (6 days ago) <strivingboy>
* 5f52c5c - change theme (6 days ago) <strivingboy>
* b83c889 - modify about me (6 days ago) <strivingboy>
* 5b9cccd - change theme (6 days ago) <strivingboy>
* d601126 - fist commit (6 days ago) <strivingboy>
* 4b7e8f0 - Fixed issue with RubyPants and the Octopress hooks filter (4 weeks ago) <Brandon Mathis>
*   fa55b68 - Merge pull request #1629 from yous/patch-layout (7 weeks ago) <Brandon Mathis>
|\
| * 32644e1 - Set layout to null to suppress warning (7 weeks ago) <ChaYoung You>
|/
* 41826d6 - Set layout to null to avoid page defaults. (7 weeks ago) <Brandon Mathis>
```



- Need tool to analyze over-synchronization feasible

- Require more work on history analysis and propose more sound but efficient algorithms

Q & A

Gu R, Jin G, Song L, et al. What change history tells us about thread synchronization[C]//Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015: 426-438.

Abstract

Multi-threaded programs are pervasive, yet difficult to write. Missing proper synchronization leads to correctness bugs and over synchronization leads to performance problems. To improve the correctness and efficiency of multi-threaded software, we need a better understanding of synchronization challenges faced by real-world developers. This paper studies the code repositories of open-source multi-threaded software projects to obtain a broad and indepth view of how developers handle synchronizations. We first examine how critical sections are changed when software evolves by checking over 250,000 revisions of four representative open-source software projects. The findings help us answer questions like how often synchronization is an afterthought for developers; whether it is difficult for developers to decide critical section boundaries and lock variables; and what are real-world over-synchronization problems. We then conduct case studies to better understand (1) how critical sections are changed to solve performance problems (i.e. over-synchronization issues) and (2) how software changes lead to synchronization-related correctness problems (i.e. concurrency bugs). This in-depth study shows that tool support is needed to help developers tackle over-synchronization problems; it also shows that concurrency bug avoidance, detection, and testing can be improved through better awareness of code revision history

Brief Introduction

Several basic concepts about the parallel programming are briefly introduced to provide a background of parallel program testing. First, the two major concepts parallel and concurrency are introduced, there are different actually. Second, the concurrent program discussed in this report is built for the shared memory setting, e.g, multicore machines with ram as the shared memory for the communication between different threads. Third, the synchronization is introduced, which is the major topic of the paper reported in this report.

Parallel vs Concurrency

The parallel lies in the setting where you have multicore machine, GPU, or other co-processors that gives the ability to run the instructions in parallel, no matter the task-level parallel or data-level parallel. However, concurrency does not mean that the have some many cores to help you conducting computations, there is a typical setting, where you are stuck with network communication or disk input output, then you can release the occupation of computing cores. Here the scheduling policy gives you concurrency.

Shared-Memory

In modern processors, there are multiple cores there, which gives us the ability to bind threads, the logical concepts of computing components to. They may load different instructions, but sometimes, they need to acquire others processing status. This could be done with the shared ram memory. Critical-Section Critical sections are the pieces of instructions issued by multiple threads that wants to acquire the limited number of resources, just as many boys and girls, e.g, 10 wants to compete for 5 seats. To deal with the race condition, in critical section, threads need to use lock or other synchronization techniques to solve the problem.

Synchronization

The instructions executed on each core may have some dependencies on the data stored in shared memory, e.g, thread 0 wants to get the intermediate result of thread 1 which is stored on the shared ram memory, then thread 0 should be idle, and scheduled out and notified after the data is ready. This could simply be done with the pthread library or win32 thread library with interfaces of mutex and spin lock, condition variable, semaphore, etc. If you are interested in these technologies, please google them.

Introduction Part

They summarize three common things about real-world big projects. First, there is information that goes beyond bug reports. Second, there is information derives from code revisions. Third, there is information that hides within the whole revision history. And intuitively, it is rather hard to collect the above three information. Thus, the authors study how lock-protected critical sections are changed when software evolves, for which they design a hierarchical taxonomy for all critical section changes, based on their structural patterns and purposes. Besides, they conduct case studies to better understand over-synchronization issues (i.e., unnecessary synchronization degrades execution performance) and concurrency bug issues(i.e., lack of or incorrect synchronization hurts execution correctness), the two parts over-synchronization and concurrency bug issues are well elaborated in the paper.

Related Work Part

There are work on concurrency bugs finding and new synchronization primitives evaluation based study, the authors argue that the paper complements them by checking software code repositories, which reveals real-world code development information unavailable in bug databases. Different from previous research, where specifically, two findings are made: (1) the number of racy variables remains high over time; (2) variables may go in and out of being racy over the course of a project, they argue that the difference between their work and others lies in that their study collects different types of software change information and answers different types of questions, including over-synchronization issues and concurrency-bug origin issues, from previous work. Besides, they argue that their study benefits the developing of profiling tools and over-synchronization solving tools.

Concept

Over-synchronization happens when unnecessary synchronization is added to the software. It would overly constrain software interleaving and lead to performance degradation. Over-synchronization is a real problem, and is cared by developers. Developers change synchronization primitives to enable lock-contention profiling in MySQL and Mozilla, and sometimes relieve over synchronization at the cost of code readability or functionality.

Discussion

Their study demonstrates that discovering and fixing over-synchronization take a lot of manual effort and are error prone. (1) All three types of changes/fixes discussed can potentially introduce concurrency bugs and demand non-trivial synchronization correctness reasoning. (2) Many new lock variables are introduced during these fixes. The ad-hoc way of introducing these variables can easily lead to correctness and/or maintenance problems. (3) The code movement during these fixes.

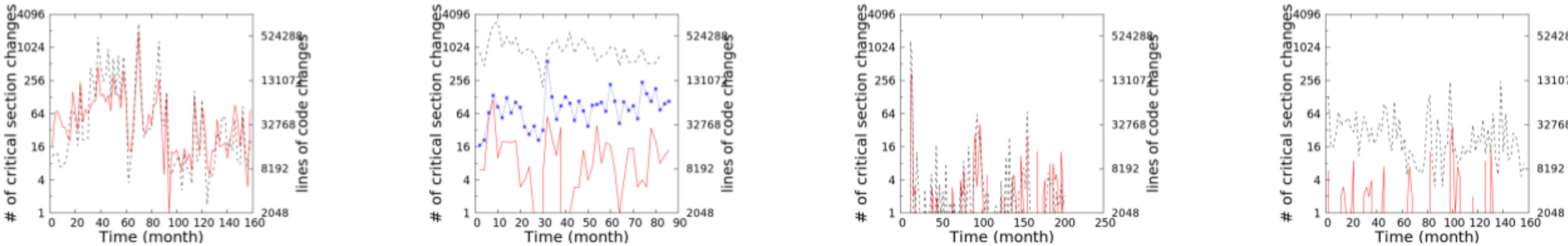
Concept

Facing large real-world multi-threaded software, it is critical to improve the performance and accuracy of existing concurrency-bug analysis techniques.

Discussion

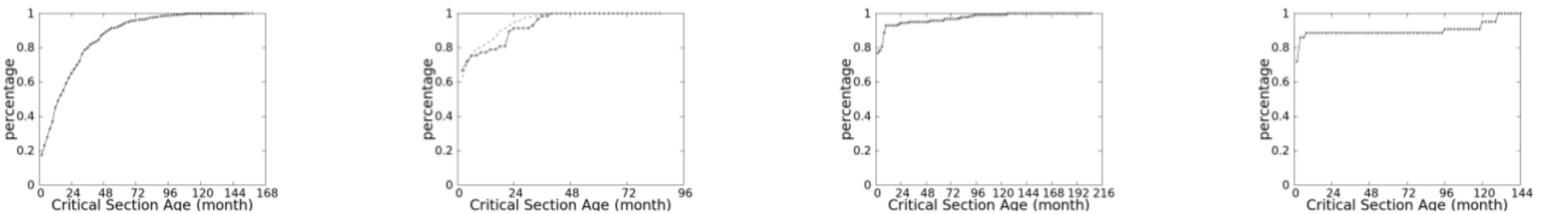
First, synchronization analysis can be significantly simplified for many bugs through history awareness. With old synchronization-context information, about half of the studied bugs would require no new synchronization analysis to be detected, because their buggy code is inside completely old synchronization contexts. Second, memory-access analysis can be significantly simplified for many bugs through history awareness. About half of the studied bugs only involve new variables accessed by new instructions with pointers propagated through new instructions. Therefore, detecting them only requires memory-access analysis for the changed code, instead of the whole Program. Third, about a quarter of the studied bugs can benefit from both almost-no synchronization analysis and revision-local memory-access analysis discussed, and hence would require extremely simple analysis to discover.

Evaluation Example (1/2)



(a) MySQL (b) Mozilla (c) Apache (d) MPlayer

Figure 2: Number of *CS* changes, shown by solid red lines, and lines of changed code, shown by dashed lines, over software ages. (We compute software age by counting how long the software has lived since its first publicly released version; the dotted blue line in Mozilla figure also considers `AutoLock`.)



(a) MySQL (b) Mozilla (c) Apache (d) MPlayer

Figure 3: Percentage of cumulative changes over the age of *CS*es (*CS* additions not counted; the dashed curve in Mozilla also considers `AutoLock`.)

Evaluation Example (2/2)

Table 8: How concurrency bugs are introduced (The subscripts represent b(ug) ids or r(evision) ids. The superscripts, A/O/D/A_m, represent common root-cause patterns [29]: single-variable atomicity violations, order violations, deadlocks, and multi-variable atomicity violations. Td represents thread.)

		New Variable	New Instruction		New Context	
			Td 1	Td 2	Td 1	Td 2
Type 1	$\text{Aget}^{\mathbf{A_m}}$, $\text{Apache}^{\mathbf{D}}_{b42031}$, $\text{Mozilla}^{\mathbf{D}}_{r996770}$ $\text{MySQL}^{\mathbf{A}}_{b791}$, $\text{MySQL}^{\mathbf{A_m}}_{r1810.2246.1}$, $\text{MySQL}^{\mathbf{D}}_{r1110.10.2}$	-	✓	-	-	-
Type 2	$\text{Apache}^{\mathbf{A}}_{b25520}$, $\text{Click}^{\mathbf{O}}$, $\text{MySQL}^{\mathbf{A}}_{b3596}$	-	✓	-	✓	-
Type 3	$\text{HTTrack}^{\mathbf{O}}_{b20247}$ $\text{Mozilla}^{\mathbf{D}}_{b79054}$, $\text{Mozilla}^{\mathbf{A/O}}_{b142651}$, $\text{Mozilla}^{\mathbf{D}}_{b679524}$ $\text{MPlayer}^{\mathbf{D}}_{r30851}$ $\text{MySQL}^{\mathbf{A}}_{r703}$, $\text{SQLite}^{\mathbf{D}}_{b1672}$, $\text{Transmission}^{\mathbf{O}}_{b1818}$, $\text{x264}^{\mathbf{O}}$	✓	✓	✓	-	-
Type 4	$\text{Apache}^{\mathbf{D}}_{r88671}$, $\text{Apache}^{\mathbf{A}}_{r103588}$, $\text{Apache}^{\mathbf{A}}_{r1201146}$, $\text{Cherokee}^{\mathbf{A}}_{b326}$ $\text{Mozilla}^{\mathbf{O}}_{b61369}$, $\text{MySQL}^{\mathbf{A_m/O}}_{b2011}$, $\text{ZSNES}^{\mathbf{O}}_{b10918}$	✓	*	✓	✓	✓

Links

Author-Github-Repo

https://github.com/ruigulala/ConAnalysis

https://github.com/ruigulala/concurrency-exploits

Open-Source-Projects-Mirror

https://github.com/apache/httpd

https://github.com/mozilla/gecko-dev

https://github.com/mpv-player/mpv

https://github.com/MariaDB/server