Optimizing the Compiler's Memory Usage? Let Us Implement a Basic Profiler First!

Gunnar Kudrjavets
University of Groningen
9712 CP Groningen, Netherlands
g.kudrjavets@rug.nl

Aditya Kumar
ACM Distinguished Speaker
Santa Monica, CA 90405, USA
hiraditya@msn.com

Abstract

The number of files and source lines of code in popular industrial code bases is significant. As of 2017, the Microsoft Windows code base contained 3.5 million files [1]. The Linux kernel contained 27.8 million lines of code in 2020 [2]. Compiling code *fast* is essential to developer productivity for thousands of engineers. Compiler performance requirements, such as CPU and I/O usage, are high. One of the application's standard performance criteria is *memory usage* and *memory allocator churn* [3]. Lower memory usage implies a higher capacity to run more compiler instances in parallel. Deceptively easy solutions to reduce memory usage, such as custom memory allocators (e.g., jemalloc [4]), are available [5]. However, in our industry experience, nothing replaces context-dependent targeted optimizations. To optimize memory usage, we need to be able to conduct reliable and valid measurements.

This talk describes the *challenges associated with designing and implementing a performant and scalable mechanism to intercept calls to a memory allocator*. We can use that intercept mechanism as an essential profiling tool. A critical requirement for this type of profiler is low-performance overhead, *enabling us to run the profiling functionality in a production environment*. Attributing and quantifying memory usage in production is a complex problem [6]. The inspiration for this presentation is our experience at Meta (Facebook), where we worked on the performance engineering of various applications.

We discuss the problems related to (a) different methods of intercepting allocator calls, such as `malloc` and `free`, (b) enabling and disabling the allocator intercept mechanism, (c) keeping track of the count and size of allocations that multiple threads request, (d) the concept of "safe" APIs that are available during the execution of the intercept mechanism, and (e) avoiding reentrancy.

We finish our talk by discussing various problems and solutions related to extending the profiling mechanism. If the in-memory data structures are insufficient to keep track of performance-related data, it must be stored somewhere. Interacting with a storage mechanism, such as a hard disk, will add complexity in the case of multiple readers and writers.

As a concrete example for our discussion, we use publicly accessible information about Mac OS X [7] and reference the source code from Apple [8].

Index Terms

Compiler performance, malloc, memory allocator, optimizations, profiling

References

[1] B. Harry. (2017, May) The largest Git repo on the planet. [Online]. Available: https://devblogs.microsoft.com/bharry/the-largest-git-repo-on-the-planet/

[2] S. Bhartiya. (2020, Jan.) Linux in 2020: 27.8 million lines of code in the kernel, 1.3 million in systemd. [Online]. Available: https://www.linux.com/news/linux-in-2020-27-8-million-lines-of-code-in-the-kernel-1-3-million-in-systemd/

[3] G. Kudrjavets, J. Thomas, A. Kumar, N. Nagappan, and A. Rastogi, "Quantifying Daily Evolution of Mobile Software Based on Memory Allocator Churn," in *Proceedings of the 9th IEEE/ACM International Conference on Mobile Software Engineering and Systems*, ser. MOBILESoft '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 28–32. [Online]. Available: https://doi.org/10.1145/3524613.3527803

[4] J. Evans, "A Scalable Concurrent malloc(3) Implementation for FreeBSD," in *Proceedings of the BSDCan Conference*, University of Ottawa, Ottawa, Canada, 2006. [Online]. Available: https://www.bsdcan.org/2006/papers/jemalloc.pdf

[5] G. Kudrjavets, J. Thomas, A. Kumar, N. Nagappan, and A. Rastogi, "There Ain't No Such Thing as a Free Custom Memory Allocator," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2022, pp. 578–581. [Online]. Available: https://doi.org/10.1109/ICSME55016.2022.00079

[6] G. Kudrjavets, A. Rastogi, J. Thomas, and N. Nagappan, "Who Ate My Memory? Towards Attribution in Memory Management," in *Proceedings of 45th International Conference on Software Engineering (ICSE 2023)*. arXiv, Dec. 2022, 45th International Conference on Software Engineering (ICSE 2023) : ICSE 2023, ICSE 2023 ; Conference date: 14-05-2023 Through 20-05-2023. [Online]. Available: https://arxiv.org/abs/2212.11866

[7] A. Singh, *Mac OS X Internals: A Systems Approach*, 4th ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2010, OCLC: 959332240.

[8] Apple, Inc. (2023) The source code for malloc.c. [Online]. Available: https://opensource.apple.com/source/libmalloc/libmalloc-317.140.5/src/malloc.c.auto.html