micrOScope: Observability First!

George V. Neville-Neil

The Problem. Getting the best performance out of a system requires access to fine grained, high fidelity, data. Unfortunately, current operating systems are too large and unwieldy to be subjected to easy measurement.

Existing tools, such as DTrace [1], eBPF[2], and KUtrace [5], allow some visibility into the operating system, but they all suffer from the same inherent limitation: they were added to operating systems as an after thought. Given that operating system functionality is often coupled with several layers, it is difficult to get accurate and detailed data.

Our Approach. Rather than trying to retrofit a monitoring framework to an existing operating system, we will build an operating system where *monitoring* is a first class primitive.

Our approach is inspired by ideas from *aspect oriented programming* (AOP) and we treat observability as a cross-cutting concern. The operating system is decomposed into modules with narrow APIs. Crucially, each module exposes *hooks* that allow developers to inject custom monitoring code before and after module functionality is invoked. Similar to the AOP notion of a weaver, we provide powerful, in kernel, run time linker to glue together the monitoring code at run time.

Any time code is injected into the kernel, there are concerns about safety. On the other hand, since the goal is to provide accurate measurements, performance is a key issue. Therefore, the micrOScope linker is unique, in that it offers a flexible binding mechanism. Monitoring code can be called via direct function call, through shared memory, or message passing, with message passing providing the highest level of safety and direct function calls providing the best performance and fidelity. The mechanism is chosen at link time rather than at compile time. This allows users to navigate performance, accuracy, and safety trade-offs.

Example. As a concrete example of how the system can be used, consider the following scenario. A user wants to decide whether or not a NIC driver should use interrupts or polling in the packet reception path. In current designs, the driver can only use crude time and packet counts to determine how busy it is. With micrOScope, the user can insert custom monitoring code directly in to the NIC driver module to collect fine-grained data from hardware counters and send the results to a user-space program for analysis.

Initially, the user might not trust the custom monitoring code they develop. To avoid sharing fate with the rest of the kernel, they can bind the monitoring code to the driver code with message passing, providing a hard boundary. Later, as they gain confidence in its stability, they could change to function call bindings, offering greater fidelity of measurement. While out of scope for this work, one could even imagine sending the telemetry data to other custom code to dynamically adapt the driver code without the involvement of a human user, i.e., changing from polling mode to interrupt mode when the system is mostly idle.

Conclusion. micrOScope is necessary because systems are now so large that they cannot be measured in a principled way. While our primary motivation in designing the system is to allow for better performance tuning, we expect that it will also serve as a platform for OS research. Several prominent systems researchers have argued that today's operating systems are ill-suited to modern hardware [4, 3]—an argument that we agree with. Designing the next generation of operating systems will require a careful understanding of the cost and effectiveness of their abstractions. micrOScope serves as a foundation for this effort.

Attempting to carve up the corpus of millions of lines of existing operating system code requires a framework to support it, and ours is targeted both at these brown field experiments as well as green field systems. The proliferation of multi-processor and multi-core systems over the last twenty years means that there is processing capacity to spare for carrying out measurements. Subsystems to be measured can be isolated away from other parts of the system, reducing extraneous noise. We take advantage of these hardware trends by focusing on discrete components which are flexibly composed.

Our early experiments show that high fidelity measurements can be carried out using a carefully crafted set of atoms running on on independent cores and CPUs, far better than than those carried out on a large monolithic system on the same hardware.

Acknowledgments Avi Silverschatz, Peter Alvaro and Robert Soulé commented on several versions of this submission.

References

[1] Cantrill, B., Shapiro, M. W., Leventhal, A. H., et al. Dynamic instrumentation of production systems. In *USENIX Annual Technical Conference, General Track* (2004), pp. 15–28.

[2] Gregg, B. *BPF Performance Tools*. Addison-Wesley Professional, 2019.

[3] Pike, R. Systems software research is irrelevant.

[4] Roscoe, T. It's time for operating systems to rediscover hardware. USENIX Association.

[5] Shojania, H. Hardware-based performance monitoring with vtune performance analyzer under linux. *Tech. Rep.* (2008).