Towards a debuggable kernel design

Ashish Gupta, Chandrika Parimoo
{ashishgu,cparimoo}@andrew.cmu.edu

***Abstract*— This paper describes what it means for a kernel to be debuggable and proposes a kernel design with debuggability in mind. We evaluate the proposed kernel design by comparing the iterations required in cyclic debugging for different classes of bugs comparing a kernel with its variant enhanced with our design rules for debuggability. We discuss the trade offs involved in designing such a kernel, and present our findings along with the design of the debuggability enhanced kernel.**

I. INTRODUCTION

Operating system kernels generally consist of several subsystems handling memory management, process management, process scheduling, network, I/O and peripheral devices. The interactions between and within different parts of the kernel are asynchronous. The size of the kernel and the inherent asynchronous concurrent model of programming makes the implementation of a robust kernel challenging. When bugs arise, isolating and reproducing them is generally harder than userspace programs due to the mentioned properties of a kernel. Kernel development and debugging is arcane[1], therefore creating a high barrier of entry for new kernel developers.

Debugging is one of the most costly phases in software development. Developers spend over 30% of their time debugging and validating software[2]. The cost of debugging rises up as one moves to lower layers of the software stack because of the increasing amounts of guarantees provided to the layers above. Debugging as an early consideration can help make the process shorter alleviating costs.

Kernel debugging has conventionally proceeded through the use of tools which stand at various levels of maturity, but never from the perspective of the kernel's design itself. We define "debuggability" as the kernel's inherent ability to lend itself to debugging. We argue that the kernel knows best about itself and can be designed to provide well structured and relevant information about the deviation in its behavior and make the process of debugging the kernel (itself) more efficient. Debuggability can thus be considered an intrinsic property of the kernel and taken into account during the design of the kernel instead of being an afterthought.

II. RELATED WORK

Existing debugging approaches include tracing and probing with the use of tools such as DTrace[3] which allow live patching of instructions with instrumentation code in order to get more information about instructions being executed. This can be a useful technique for debugging, but only captures the execution trace (where overhead can be as large as 100x for high frequency function calls[4]). Our approach exposes the kernel's high level information to the developer in the form of data structures that the developer understands instead of only relying on the execution stream and function arguments.

Kernel developers use forms of remote debugging (over a serial line)[5]. While this technique has been an effective way for kernel debugging, it requires the possession of an additional machine to run a master kernel on one machine sending commands to the debugee kernel on another machine. We do away with the requirement for additional hardware for debugging by incorporating the ability for a kernel to help debug itself.

Hardware based approaches such as JTAG[6] are very powerful expose a lot of detail about the execution with the help of a special controller. Although one can get a large amount of information , the information provided can be too low-level to be useful for debugging kernel design issues. Our approach, on the other hand, focuses on the kernel's high level structures providing means to infer some low-level information as well.

Hypervisor based record and replay approaches such as Samsara[7] offer an excellent way to tackle hard to reproduce bugs through the use of virtual machines and hypervisors. Although effective, they inherently lack performance as the use of virtual machines and hypervisors slows down performance (upto 6x on a 4-core machine) limiting their usefulness. In our approach, we wish to get rid of the overhead imposed by the use of virtual machines and exploit the hardware the kernel runs on directly to provide a record-and-replay feature as a part of d-mode.

Diagnosys[8] allows the automatic generation of debugging interfaces by statically analyzing the kernel code for safety holes. This allows Diagnosys to generate relevant logs at run time which help in debugging. The approach is restricted to the identified safety holes and Diagnosys is added as a module to the Linux kernel rather than being integrated in the design of the kernel.

The work on exposing bugs before having to debug them provides good examples of the kind of bugs most likely to be found in a kernel. Landslide[9], in its approach of