OS Driver Test Effort Reduction via Operational Profiling*

Constantin Sârbu†, Stefan Winter†, Nachiappan Nagappan‡ and Neeraj Suri†
Technische Universität Darmstadt†
Hochschulstr. 10, Darmstadt, Germany
{cs,sw,suri}@cs.tu-darmstadt.de
Microsoft Research‡
Redmond, WA, USA
nachin@microsoft.com

1 Introduction and Issues

Operating Systems (OS's) constitute the operational core for computing devices, and consequently, the OS's ability to sustain operations determines the dependability level of the provided system services. In order to facilitate their applicability to a variety of hardware platforms, OS's have evolved into complex, componentized software entities whose key function is to provide applications access to the system's hardware resources. Within the OS, the key components dominating the cause of OS failures are the device drivers (DDs), precisely the OS parts designed to enhance the OS's support for hardware. Unfortunately, despite intensive efforts to elevate DD's robustness levels by employing varied test paradigms, the existing DDs still exhibit very high failure rates. Obviously, testing the complete state space of a DD is neither technically or economically viable. Based on our empirical DD evaluations, we conjecture that the testing deficiencies are the consequence of missing key parts of the DD's operational states in the process of testing, a situation illustrated in Fig. 1.

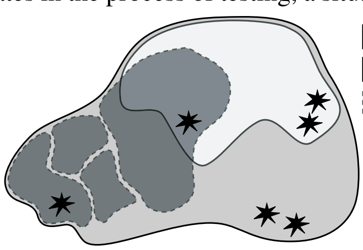

Fig. 1. Test space vs. operational space.

The better the DD's actual operational states can be identified, the better (focused and effective) the testing of those states can be achieved. Hence, the central premise behind our work involves the characterization of a DD's operational profile, and using it for focusing subsequent testing to the functional areas likely to be exercised over the DD deployment, i.e., "*execution hotspots*". Revealing the execution hotspots helps *test space identification* and *test case selection*, two aspects critical for reducing the test effort without losing testing process' adequacy.

*Supported in part by Microsoft, IBM, GK MM and EC Inco-Trust and Inspire.

As a preliminary contribution toward this goal, we have developed two novel methodologies for revealing DD's operational state space from binaries sans source code access [5–8]. Currently, our research focuses on the expansion of our existing operational profiling framework [5] into a full-fledged test tool for DDs.

Related work: Ball and Larus acknowledged the application of path profiling for test coverage assessment [1], "*by profiling a program and reporting unexecuted statements or control flow*". Instrumentation of binaries was used to obtain instruction traces and to identify execution paths. The paths ended at loop and functional boundaries, a limitation resolved later by the "*whole program paths*" approach [2]. However, these approaches are not directly applicable to DDs, as instrumentation induces high overheads, thus limiting the use of such approaches inside the OS kernel space.

Among others, Leon and Podgursky [3] used profiles generated by individual test cases and a clustering technique for evaluating test suite minimization by selecting one test case per cluster for reducing test efforts. Their approach was designed for general purpose software, thus also not being directly applicable for kernel-mode OS components.

2 Identifying Execution Hotspots in DDs

In contrast to contemporary efforts, we have developed two distinct methodologies to capture and analyze the operational profiles of kernel-mode DDs [5]. The first – termed as the *state-based operational profile (SOP)* [6] – is based on the state characterization of the I/O traffic between a selected DD and the rest of the OS kernel. The second – termed as the *execution path profile (EPP)* [7] – observes the functional calls made by the respective DD in the operational phase, thus revealing the code paths followed at runtime. The two approaches are complementary and are directly applicable to DD binaries. Moreover, they do not require source-code access to any of the involved OS kernel components.

Our DD profiles reveal execution hotspots a) as DD reached states (the SOP) and b) as traversed code paths (the EPP). As a DD's subset of states visited at runtime represents only a small fraction of the total state space [6], it highlights the areas to be primarily tested, subsequently enabling for test effort reduction. Complementing the SOP, the EPP discovers execution hotspots as frequently traversed DD code paths. Such code paths are identified as call sequences to kernel functions implemented externally to the selected DD. Based on likeness (computed using string similarity metrics), the code paths are clustered into equivalence classes, thus helping to identify execution hotspots as primary targets for testing.

3 Preliminary Experimental Results

To validate the effectiveness of the DD operational profiling process, we have conducted a series of extensive case studies including over fifty actual Windows XP and Vista DDs [5]. The key resulting observation is that the distinct code paths taken at runtime constitute only a small fraction of the total number of observed paths. For instance, by exercising the Windows XP floppy disk driver with several off-the-shelf performance and reliability benchmarks only 2.37% of all captured code paths were found to be distinct. Moreover, the observed distinct code paths are very similar to each other, thus revealing the execution hotspots as relatively small areas in the DD's execution space. Therefore, the testing space (and implicitly, the associated test effort) can be drastically reduced by "re-focusing" it onto the newly revealed hotspots.

The first research question guiding our effort toward reducing test effort is: *What is the area covered by actual DD test tools?* To answer this question we exercised the aforementioned floppy DD with a powerful robustness test tool from Microsoft[1] (additional to the benchmarks). Fig. 2 is a multidimensional scaling (MDS) plot where points represent the distinct code paths followed for each benchmark.

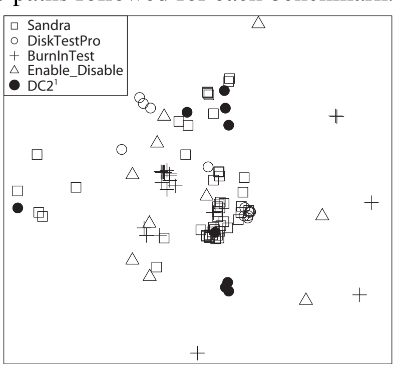

Fig. 2. Floppy DD's execution space (MDS plot)

In MDS plots the relative distance between two data points visually express their dissimilarity. Hence, the agglomerations of data points in the Fig. 2 reveal multiple execution hotspots. Supporting our initial assumption that tested areas do not always match the operational areas, most of the code paths followed by DC2[1] are located far-off the main execution hotspots. We are currently investigating multiple other DD test tools for their followed code paths.

The second pursued research question is: *How to tune test tools to match execution hotspots?* A possible solution to this problem is to use the DD profiles obtained for various use cases and infer the test cases that trigger the following of the same paths inside the DD code. Next, the test cases are fed to an existing test tool.

The selection of test cases should consider the information obtained from a prior DD profiling phase in order to reduce the overall testing overhead. This will also help an early identification of the insufficiently tested DD areas and assess their impact on DD dependability, together with an investigation of the possibility to correlate known OS failures to DD operational profiles.

4 Discussion and Future Work

In order to expand the presented ideas into an operational profile-based DD testing tool, a key prerequisite is validating them against a wider spectrum of DDs, existing test tools and representative use cases.

Goal 1: Currently, we are considering various test tools which are both "*relevant*" (intensively used for testing actual DDs) and "*customizable*" (provide access to the used test cases). Finding the "*typical usage*" for the targeted DD constitutes an important current research direction for defining the test space of the future tool.

Goal 2: Considering the source code (or, alternatively, the binary images) of the profiled DDs, we intend to map the obtained code paths to the control flow graphs. This should serve as a validation for our operational profiling methodology by quantifying its capacity to disclose the followed code paths. Moreover, we believe that this evaluative approach provides for proper comparisons among existing black- and white-box DD test methods from the code coverage perspective.

Goal 3: An ongoing research direction is a quantitative study of driver-relevant workloads that considers using our profiling mechanisms to characterize test workloads from a DD's perspective. This information would guide the choice of adequate workloads for specific test scenarios.

References

[1] T. Ball and J. R. Larus. Efficient path profiling. In *MICRO-29*, pp. 46–57, December 1996.

[2] J. R. Larus. Whole program paths. In *ACM SIGPLAN*, volume 34, pp. 259–269, 1999.

[3] D. Leon and A. Podgurski. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. In *Proc. ISSRE*, pp. 442–453, 2003.

[4] P. Orwick and G. Smith. *Developing Drivers with the Windows Driver Foundation*. Microsoft Press, Redmond, 2007.

[5] C. Sârbu. *Operational Profiling of OS Drivers*. PhD thesis, Technische Universität Darmstadt, May 2009.

[6] C. Sârbu, A. Johansson, F. Fraikin, and N. Suri. Improving Robustness Testing of COTS OS Extensions. In *Proc. ISAS*, LNCS 4328, pp. 120–139, Springer Verlag, 2006.

[7] C. Sârbu, A. Johansson, and N. Suri. Execution Path Profiling for OS Device Drivers: Viability and Methodology. In *Proc. ISAS*, LNCS 5017, pp. 90–107, Springer Verlag, 2008.

[8] C. Sârbu, A. Johansson, N. Suri, and N. Nagappan. Profiling the Operational Behavior of OS Device Drivers. In *ISSRE*, pp. 127–136, 2008.

[1] Microsoft's *Device Path Exerciser* (*DC2*) [4], chapter 21, pp. 671