CS503 Paper Review

David Perry
perry74@purdue.edu

Recovering Device Drivers
Michael M. Swift, Muthukarrupan Annamal Ai, Brian N. Bershad, and Henry M. Levy

1. **Give three technical reasons why this is a good paper.**
   - This paper provides a novel technique that improves the reliability of applications that depend on device drivers. Previous research has provided a way to make operating systems resistant to crashes due to device driver faults. However, this paper presents an approach that allows individual applications that rely on device drivers to be crash resistant as well. The paper accomplishes such a feat by presenting a mechanism called shadow driver. A shadow driver is a generalized solution that lies in the background of a kernel passively listening to communication between requesting applications and corresponding device drivers. If the shadow driver detects a fault it disconnects this communication, records the current state of the device driver, and logs proceeding requests until the driver is able to recover. Once the device recovers the shadow driver resets the driver's state and forwards the logged requests.

   - The implementation of the mechanism provided showed that very little over head was required to become effective. The paper experimented with their implementation on varying systems and drivers and showed that on average a systems efficiency was 99% with the worst coming in at 97%. This low over-head is mainly explained by the low complexity related to shadowing execution. A large number of the executions required during passive mode are no ops. In fact, of the 177 communication taps added only 31 of them required any code implementation. The majority of complexity added to a system from shadowing comes from TLB misses caused by fault isolation which only has noticeable impact on a few device drivers. However, even with such small over head the implementations on three varying drivers were able to be quite effective evading 10 application crashes on a Linux-native operating system, and 7 malfunctions or crashes on a fault isolation based Linux distribution..

   - The amount of effort required for such reported results were surprisingly low. Of the varying shadow driver implementations the largest only took 666 lines of code. This is quite small especially when considering that the device driver itself was implemented in 7,831 lines of code. The changes required to a Linux distribution already capable of fault isolation were also quite low. According to the authors the mechanism only required adding around 3,300 added lines of code to the kernel. These statistics show that not only are shadow drivers effective but they are practical. The amount of required code is realistic, implementable, and should be easy to maintain.

2. **Describe at least one technical weakness or limitation of the paper and propose possible improvement.**

- Shadow drivers are only capable of preventing application crashes when the device driver fault is both transient and fail-stop. This is undesirable as the majority of device driver faults are of type deterministic failure. However, the paper justifies their mechanism by claiming that transient failures are much more difficult to find during the testing phase when compared to deterministic failures. The implementation described is also not necessarily applicable to all device drivers. When shadow drivers log requests sent to the disconnected device driver some requesting applications will attempt to make duplicate communications. However, the shadow driver will not be able to distinguish between authentic and duplicate requests. Thus, the shadow driver must forward all communications. While this may not matter with many applications it is highly undesirable in others. Consider the instance of sending a request to print a document to a printer. During the shadow driver's execution multiple requests may be made corresponding to a single desired print of a document.

The review should be limited to one page