Dependability Assessment of Operating Systems in Multi-core Architectures

Gabriela Jacques-Silva*, Zbigniew Kalbarczyk, Ravishankar K. Iyer
Center for Reliable and High-Performance Computing
University of Illinois at Urbana-Champaign, IL 61801
{gjsilva,kalbar,rkiyer}@crhc.uiuc.edu

1 Introduction

The dependability of a service is directly influenced by the reliability provided by the lower system layers, such as the operating system and hardware. Due to device and voltage scaling, and the increasing complexity of digital systems, transient errors are forecast to be a problem for all future digital systems [4]. When we consider large-scale machines, with many processors and cores, this phenomenon is furthered exacerbated. Therefore, it is mandatory to evaluate the behavior of systems under such conditions.

In this context, understanding the way errors can manifest and are handled by the platform operating system layer gives an indication of the reliability of the system and also how they affect the user-level applications.

The robustness and fault sensitivity of operating systems have been extensively studied [2, 3, 1]. There are several challenges in developing a fault injection framework for multi-core systems: (i) it is likely that we have more than one error at a time, possibly occurring in different processors/cores; (ii) there is no direct control over where the fault injector is going to be executed or where the workload is being placed by the operating system scheduler; (iii) a fault that is injected in one processor can get manifested in another processor. These issue make it difficult to accurately measure dependability metrics, such as crash latency. The use of standard techniques, such as using performance counters, is not trivial, given that we would have to configure the performance registers in all processors at the same time. This scenario poses a new set of questions that have not been addressed by the current approaches: is the error behavior different with errors occurring in multiple processors/cores? How do latent errors influence the system? Do errors propagate between different processors or cores?

In addressing these questions, this work describes the development of an experimental environment that enables fault injection based evaluation of operating systems, particularly Linux, for multi-core and multiprocessor architectures.

2 NFTAPE Operating System Injection

In this study we leverage the NFTAPE fault injection testing environment [1]. The current NFTAPE implementation for Operating System (OS) assessment consists of two machines: (i) the target machine, which is a machine hosting the operating system under study. The *process manager* application, which performs the actual fault injection, runs on this machine; and (ii) a control host machine (different from the target), which controls the experiment by issuing commands to the target machine, via the *process manager*, and collecting data about the injections.

The process of conducting an OS injection campaign, there are two phases: (i) setting up the injection, and (ii) performing the injection itself, i.e., executing instructions that corrupt a specific value in memory or registers. Figure 1 depicts how the injection works. The control host issues a request to inject a fault to the *process manager*. The process manager invokes a *injector* program, which is a user level program. This process receives the injection parameters, and passes them to the NFTAPE *kernel module*. The *kernel module* writes the parameters into kernel data structures, added specially for fault injection. It injects faults by using breakpoint registers. When the module receives the address of the target data/instruction for the injection, it writes this value into the processor debug register. When the target data/instruction at this address is accessed/executed, the kernel executes the breakpoint handler. The breakpoint handler is instrumented with fault injection instructions, which are used to corrupt the value according to a given fault mask.

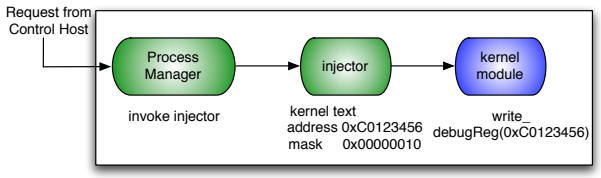

Figure 1. Target machine procedure

*Sponsored by CAPES/Brazil

2.1 Moving to Multiprocessor Systems

The current injection tools typically makes use of debug registers to obtain fine-grained data about injection experiments (e.g., if a fault is activated). In the case of multi-core architecture, each CPU/core has its own register set. Therefore, we have to find a way to set the debug registers in a specific CPU and enable the fault injection triggering. We have to select and force multiple CPUs to trigger a fault. Since we want to emulate the occurrence of multiple faults, we have to set different breakpoint addresses in different CPUs. This means that we need to force the execution of the injection process in different CPUs. Therefore, when we invoke the kernel module it will set the registers in the CPU that is executing the *injection* process.

To enforce fault triggering in multiple processors, we take advantage of a set of system calls provided by the Linux kernel. Linux allows a process to set and get a CPU affinity mask. This affinity mask determines which CPUs a process is allowed to run. The affinity is also passed along to any forked child.

Figure 2 shows how we take advantage of such system calls. When the *process manager* receives a injection request from the control host, it instantiates a *core dispatcher*. Since the *core dispatcher* is forked by the *process manager*, they execute in the same CPU. If the injection is targeting processors 0 and 2, we add both processors to the *core dispatcher* queue. It then forks a injection process in CPU0. This process will set the appropriate debug registers in CPU0 by accessing the *kernel module*. After the *injector* is forked, the *core dispatcher* changes its CPU affinity and relinquishes the current processor, forcing it to be executed in another CPU (CPU2 in this case). In the next run of *core dispatcher*, it will be running on CPU2, and it will then dispatch the second *injector* process. This process sets the appropriate debug registers in CPU2. To ensure that a fault is not injected while the debug registers are being set, we add a global kernel variable that is set only after all the configuration has taken place. If a breakpoint exception occurs while this variable is not set, it is ignored by the fault injection instrumentation.

2.2 Fault model

The fault model that is currently being considered for the multi-core injection is single/multiple bit flips. However, here we assume that they can happen more than once during a single experiment run. The current targets for injections in the OS are the following:

Kernel Text - injection into kernel instructions. Since in multiprocessor architectures the kernel code section is shared, we can have two types of injections, depending on the fault location: (i) a fault that occurs in memory and corrupts a instruction, making the fault visible to all the processors in the system, (i) a fault that occurs on the pipeline, bus or in the instruction cache of a particular processor, affecting one processor only.

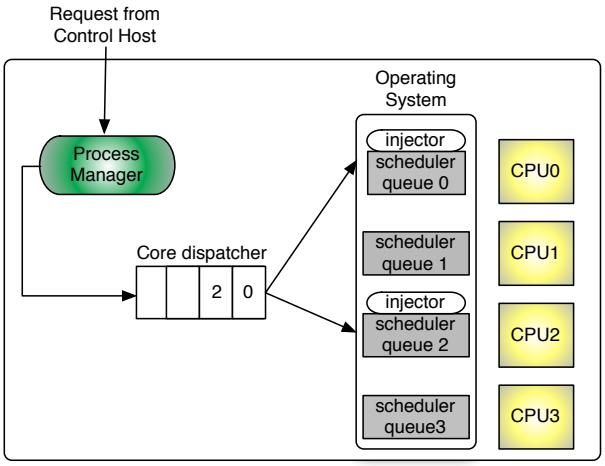

Figure 2. Forcing breakpoint in different CPUs

Kernel Stack - injection to a valid kernel stack range of a process running on the target CPUs.

Global Kernel Data - injection to a global kernel data structure, which is visible to all the processors in the system.

3 Future work

This fast abstract describes a first step towards a framework to evaluate operating system behavior in multi-core environment. In this scenario, we have to consider the possibility that more then one error can occur in a short period of time. We aim at verifying if the failure behavior of operating systems is different when running in environments that are susceptible to higher rates of transient errors, such as multi-core architectures. Other issues to be addressed are the execution and distribution of workload in different processors, and the collection of data for dependability metrics, such as crash latency. Such metric is important for estimating error propagation.

References

[1] W. Gu, Z. Kalbarczyk, and R. K. Iyer. Error sensitivity of the linux kernel executing on powerpc g4 and pentium 4 processors. In *Proc. of DSN 2004*, Washington, DC, USA, 2004.

[2] T. Jarboui, J. Arlat, Y. Crouzet., K. Kanoun, and T. Marteau. Analysis of the effects of real and injected software faults: Linux as a case study. In *Proc. of PRDC2002*, pages 51–58, 16-18 Dec. 2002.

[3] P. Koopman and J. DeVale. The exception handling effectiveness of posix operating systems. *IEEE Trans. Softw. Eng.*, 26(9):837–848, 2000.

[4] N. J. Wang and S. J. Patel. Restore: Symptom-based soft error detection in microprocessors. *IEEE Trans. Dependable Secur. Comput.*, 3(3):188–201, 2006.