UMLinux – A Versatile SWIFI Tool

Volkmar Sieh and Kerstin Buchacker

Institut für Informatik 3
Friedrich Alexander Universität Erlangen-Nürnberg, Germany
{volkmar.sieh,kerstin.buchacker}@informatik.uni-erlangen.de

Abstract. This tool presentation describes UMLinux, a versatile framework for testing the behavior of networked machines running the Linux operating system in the presence of faults. UMLinux can inject a variety of faults into the hardware of simulated machines, such as faults in the computing core or peripheral devices of a machine or faults in the network connecting the machines. The system under test, which may include several machines, as well as the fault- and workload run on this system are configurable.
UMLinux has a number of advantages over traditional SWIFI and simulation tools: speed, immunity of fault-injection and logging processes from the state of the machine into which the faults are injected and binary compatibility with real world data and programs.

1 Introduction

This tool presentation describes UMLinux, a framework capable of evaluating the dependability behavior of networked machines running the Linux operating system in the presence of faults. The Linux operating system is usually employed in networked server environments, for example as web- or mailserver.

The tool uses software implemented fault injection (SWIFI) to inject faults into a simulated system of Linux machines. The simulation environment is made available by virtualization, i.e. by porting the Linux operating system to a new "hardware" — the Linux operating system! Due to the *binary compatibility* of the simulated and the host system, any program that runs on the host system will also run on the simulated machine.

A process paired with each simulated machine injects faults via the `ptrace` interface. This interface allows complete control over the traced process, including access to registers and memory as well as to arguments and return values of input/output operations. Possible faults include hardware faults in computing core and peripheral devices of a single machine as well as faults external to machines, such as faults in external networking hardware.

The tool will be used in the European DBench Project [6] for dependability benchmarking of Linux systems.

The rest of the paper is structured as follows. Section 2 gives a short overview of the main advantages of UMLinux over traditional SWIFI and simulation tools. Section 3 gives an outline of the different parts of the tool. Information about the

F. Grandoni (Ed.): EDCC 2002, LNCS 2485, pp. 159–171, 2002.

configuration of the simulated hardware is found in Sect. 4. Section 5 explains how to inject faults using UMLinux. Section 6 describes an example experiment. The final section outlines a tool demonstration. For information about the implementation of UMLinux please refer to [2].

2 Advantages of UMLinux

Since the issues and problems in implementing a software injection tool at the operating system level as well as the technical details of UMLinux have been treated in [2, 8, 17] this tool presentation will concentrate on the user perspective of UMLinux. UMLinux has advantages over pure simulation and SWIFI tools or virtualization software, because it combines all three — simulation, SWIFI and virtualization. To our knowledge, there is currently no other such tool available worldwide.

[2] gives a short overview of a number of available simulation and SWIFI tools, including VHDL-based simulation ([9, 18]), CrashMe [4], Fuzz [13], FERRARI [10], MAFALDA [14], a fault-injector based on the `ptrace` interface [16], FIAT [1], Xception [3], and Ballista [11].

UMLinux differs from the SWIFI and simulation work named in the previous paragraph in several important aspects. It *combines* SWIFI with a simulation approach and therefore offers all the possibilities of the former. Since we simulate the hardware at a relatively high level, the simulation is unusually fast (slowdown is less then one order of magnitude). Using SWIFI together with a simulated machine has the advantage, that once set up, no user interaction is required.

When using a SWIFI tool to inject faults into the operating system of the machine the tool is actually running on, the faults may also affect the integrity of the tool and cause erroneous results to be logged. In addition, automating testing is difficult, since the machine must usually be rebooted manually when the operating system crashes or hangs and test results may be lost. In UMLinux, on the other hand, the faults are injected into a simulated machine and the fault injection software is not in any way dependent on the integrity of this simulated machine. The integrity of the host machine is in no way affected by the faults injected into the simulated machine. Since the fault injection code is separated from the code for the simulated machine and runs as a separate process, undesired interference and intrusion of the fault injection code on the simulated machine is avoided. Additional capabilities *not* offered by other simulation or SWIFI techniques include fault injection into a system of networked machines.

When using a simulation to represent a real world system, the most important question is, how closely does the simulation mimic the behaviour of the real world system? And inversely, how closely will the real world system follow the behaviour of the simulation in the presence of faults? UMLinux machines can run unmodified real world binaries and can directly use disk images of real world machines. This means that almost the complete software and data part of a UMLinux machine is identical to that of a real world machine. The differences are in the hardware and the closely hardware related software parts (drivers).