

programming of the components most important for performance modelling: cache hierarchies, synchronisation in multiprocessor machines, and I/O devices. By focusing on the major performance bottlenecks, the simulator provides a useful time model while executing only 100 times slower than real machines, allowing simulation of large workloads.

A simulated computer is purely artificial and deterministic. Thus, a simulated system will always execute along the same path if the initial state is identical and all input to the simulator is deterministic. Reproducible execution is a very desirable property for application debugging. In order to obtain reproducibility, the user must provide deterministic models of application input, and the debugger must operate without perturbing the simulated system.

3. Virtual machine translation

In general-purpose operating systems, such as Linux, each program runs in a protected environment, with private registers, memory, and operating system resources. This environment is referred to as a *virtual machine*. In order for a debugger to debug a program, it must be able to control execution and probe state of the corresponding virtual machine. Traditional debuggers rely on the operating system to provide a controlling and probing mechanism. As a simulation-based debugger must avoid modifying the simulated system, it cannot use operating system services to probe the target.

In order to support debugging of processes in the simulated system, we introduce an intermediate filter between the debugger and the simulator, a *virtual machine translator* (VMT), that answers a debugger's queries for virtual machine state. Queries for memory content refer to virtual addresses, and must be translated to physical addresses. The VMT performs this translation by parsing data structures in the operating system, first looking up the head of the process list, which is a global variable whose address is found in the kernel symbol table. The appropriate process entry is found by following pointers referring to data structures in the simulated memory. The VMT proceeds by parsing the process's page table until the mapping for the virtual address is found. If the page resides in physical memory, the VMT queries the simulator for the contents and responds to the debugger. In order to read pages that have been written to disk, the VMT needs to walk the kernel data structures further to find which disk block it resides in and query the simulator for disk contents. Other useful information, such as virtual register and file contents, is retrieved in a similar manner.

The VMT supports reproducible debugging of multi-threaded applications by allowing multiple debuggers to attach to different processes in the simulated system (shown in Figure 2). It keeps state for each debugger connected and

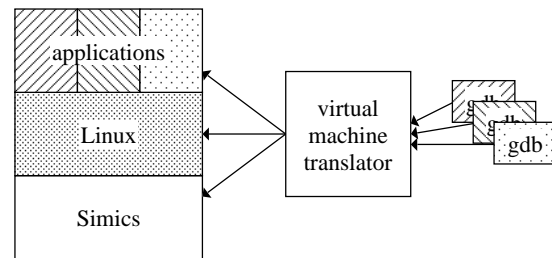


Figure 2. Simulation-based debugging.

maps debugger queries to virtual machines. Simulator execution is controlled coherently by forwarding simulation only when all connected debuggers are ready to execute. The VMT also asserts that all debugger breakpoints refer to physical memory. It maintains a list of breakpoints in use and inserts (removes) simulation breakpoints when code pages are mapped (unmapped) to physical memory.

4. Future Work

As the VMT exposes operating system internals, it can easily be enhanced to collect statistics on performance-related operating system events, such as page faults, context switches, and system calls. Such instrumentation, in combination with Simics's statistics on cache misses and GDB's support for programming breakpoint handlers, enables presentation of performance statistics from multiple abstraction layers in a system. The temporal debugger is therefore a suitable platform for building a more sophisticated tool, that could compare performance statistics from different periods in a real-time application. Such a tool would be very useful for finding causes of missed deadlines, as it could present differences between periods with and without missed deadlines.

References

- [1] The GNU debugger, version 5.0. <http://sources.redhat.com/gdb>.
- [2] S. A. Herrod. *Using Complete Machine Simulation to Understand Computer System Behavior*. PhD thesis, Stanford University, Feb. 1998.
- [3] P. S. Magnusson, F. Dahlgren, H. Grahn, M. Karlsson, F. Larsson, F. Lundholm, A. Moestedt, J. Nilsson, P. Stenström, and B. Werner. SimICS/sun4m: A Virtual Workstation. In *Proceedings of the 1998 USENIX Annual Technical Conference*, 1998.
- [4] Virtutech Simics v0.97/sun4u. <http://www.simics.com>.