Simulation-Based Debugging of Soft Real-Time Applications

Lars Albertsson
Computer and Network Architectures Laboratory
Swedish Institute of Computer Science
lalle@sics.se, www.sics.se/~lalle

Abstract

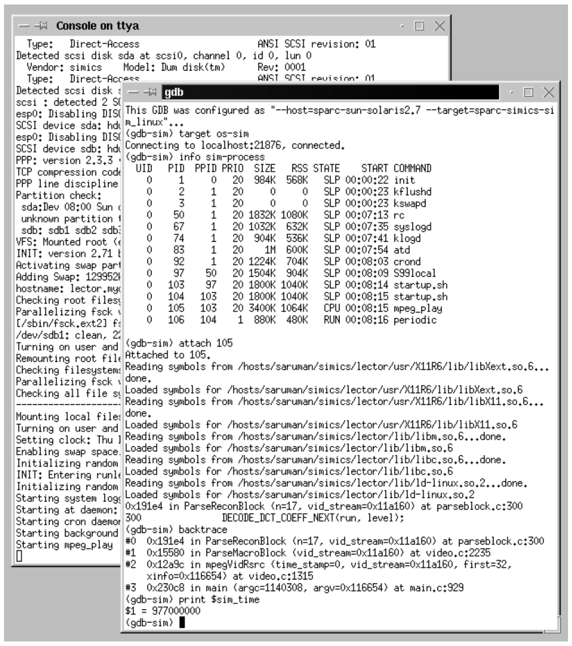
We present a temporal debugger, capable of examining time flow of soft real-time applications in Unix systems. The debugger is based on a simulator modelling an entire workstation in sufficient detail to run unmodified operating systems and applications. It provides a deterministic and non-intrusive debugging environment, allowing reproducible presentation of program time flow.

The primary contribution of this paper is virtual machine translation, a technique necessary to debug applications in a simulated Unix system. We show how a virtual machine translator maps low-level data, provided by the simulator, to data useful to a symbolic debugger. The translator operates by parsing data structures in the target operating system and has been implemented for the GNU debugger and simulated Linux systems.

1. Introduction

The debugger is one of the most important tools for computer programming. Traditional debuggers, however, are inadequate for real-time applications, as they interfere with program execution, thereby modifying time flow. In embedded real-time system design, this problem has been addressed with simulator-based debuggers. Until recently, it has not been practical to use simulators to study commodity desktop and server computer systems, as the simulators have been too slow. Recent advances in simulation technology have resulted in *complete system simulators*, modelling an entire workstation with sufficient performance and detail to run unmodified operating systems and large workloads [2, 3].

In this paper, we present a temporal debugger for soft real-time Unix applications. It is based on the GNU debugger [1], modified to debug Linux applications running in a complete system simulator. The debugger is able to non-intrusively debug applications in the simulated system and present program time flow using the simulator's time model. The debugger user interface is shown in Figure 1. The foreground window shows how GDB connects to the simulator and attaches to a process. Simulated time, measured in number of cycles since boot, is available in the debugger variable `sim_time`. The window in the background shows the simulated console, printing output from Linux during boot.



```
  Type:   Direct-Access                      ANSI SCSI revision: 01
Detected scsi disk sda at scsi0, channel 0, id 0, lun 0
  Vendor: simics    Model: Dum disk(tm)      Rev: 0001
  Type:   Direct-Access                      ANSI SCSI revision: 01
Detected scsi disk s
scsi : detected 2 S
esp0: Disabling DIS
SCSI device sda: hd
esp0: Disabling DIS
SCSI device sdb: hd
PPP: version 2.3.3
TCP compression cod
PPP line discipline
Partition check:
 sda:Dev 08:00 Sun
 unknown partition
 sdb: sdb1 sdb2 sdb
VFS: Mounted root (
INIT: version 2.71
Activating swap par
Adding Swap: 129952
hostname: lector.my
Checking root files
Parallelizing fsck
[/sbin/fsck.ext2] f
/dev/sdb1: clean, 2
Turning on user and
Remounting root fil
Checking filesystem
Parallelizing fsck
Checking all file s
--------------------
Mounting local file
Turning on user and
Setting clock: Thu
Enabling swap space
Initializing random
INIT: Entering runl
Initializing random
Starting system log
Starting at daemon:
Starting cron daemo
Starting background
Starting mpeg_play
```

```
This GDB was configured as "--host=sparc-sun-solaris2.7 --target=sparc-simics-si
m_linux"...
(gdb-sim) target os-sim
Connecting to localhost:21876, connected.
(gdb-sim) info sim-process
  UID   PID  PPID PRIO  SIZE   RSS STATE    START COMMAND
    0     1     0   20  984K  568K   SLP 00:00:22 init
    0     2     1   20     0     0   SLP 00:00:23 kflushd
    0     3     1   20     0     0   SLP 00:00:23 kswapd
    0    50     1   20 1832K 1080K   SLP 00:07:13 rc
    0    67     1   20 1032K  632K   SLP 00:07:35 syslogd
    0    74     1   20  904K  536K   SLP 00:07:41 klogd
    0    83     1   20    1M  600K   SLP 00:07:54 atd
    0    92     1   20 1224K  704K   SLP 00:08:03 crond
    0    97    50   20 1504K  904K   SLP 00:08:09 S99local
    0   103    97   20 1800K 1040K   SLP 00:08:14 startup.sh
    0   104   103   20 1800K 1040K   SLP 00:08:15 startup.sh
    0   105   103   20 3400K 1064K   CPU 00:08:15 mpeg_play
    0   106   104    1  880K  480K   RUN 00:08:16 periodic

(gdb-sim) attach 105
Attached to 105.
Reading symbols from /hosts/saruman/simics/lector/usr/X11R6/lib/libXext.so.6...
done.
Loaded symbols for /hosts/saruman/simics/lector/usr/X11R6/lib/libXext.so.6
Reading symbols from /hosts/saruman/simics/lector/usr/X11R6/lib/libX11.so.6...
done.
Loaded symbols for /hosts/saruman/simics/lector/usr/X11R6/lib/libX11.so.6
Reading symbols from /hosts/saruman/simics/lector/lib/libm.so.6...done.
Loaded symbols for /hosts/saruman/simics/lector/lib/libm.so.6
Reading symbols from /hosts/saruman/simics/lector/lib/libc.so.6...done.
Loaded symbols for /hosts/saruman/simics/lector/lib/libc.so.6
Reading symbols from /hosts/saruman/simics/lector/lib/ld-linux.so.2...done.
Loaded symbols for /hosts/saruman/simics/lector/lib/ld-linux.so.2
0x191e4 in ParseReconBlock (n=17, vid_stream=0x11a160) at parseblock.c:300
300             DECODE_DCT_COEFF_NEXT(run, level);
(gdb-sim) backtrace
#0  0x191e4 in ParseReconBlock (n=17, vid_stream=0x11a160) at parseblock.c:300
#1  0x15580 in ParseMacroBlock (vid_stream=0x11a160) at video.c:2235
#2  0x12a9c in mpegVidRsrc (time_stamp=0, vid_stream=0x11a160, first=32,
    xinfo=0x116654) at video.c:1315
#3  0x230c8 in main (argc=1140308, argv=0x116654) at main.c:929
(gdb-sim) print $sim_time
$1 = 977000000
(gdb-sim) 
```

Figure 1. Temporal debugger user interface.

2. Complete system simulation

The complete system simulator used for this work is Virtutech Simics [4]. It models all hardware components in an UltraSPARC workstation, and runs unmodified installations of Solaris or Linux. The simulator approximates time by assuming that an instruction executes in constant time, plus stall time from memory accesses. Simics supports efficient 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 multithreaded applications by allowing multiple debuggers to attach to different processes in the simulated system (shown

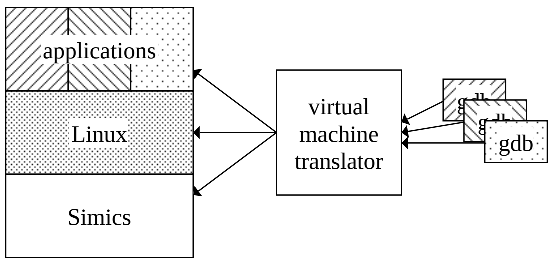


Figure 2. Simulation-based debugging.

in Figure 2). It keeps state for each debugger connected and 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.