AC 2008-1339: TEACHING REAL OPERATING SYSTEMS WITH THE LTTNG KERNEL TRACER

Mathieu Desnoyers, Ecole Polytechnique de Montreal

Mathieu Desnoyers is the maintainer of the Linux Trace Toolkit (LTT) project since November 2005, taking over the development with the new LTTNG. He is the author of Linux Trace Toolkit Next Generation (LTTNG) and the main developer of Linux Trace Toolkit Viewer (LTTV) since the project started in 2003. He did an internship at the IBM Research T.J. Watson Research Center in 2006 where he applied tracing in commercial scale-out systems. In 2007, he did an internship at Google, where he integrated ideas from Google ktrace into LTTng to merge themin a single project. He also worked on LTTng in collaboration with Autodesk Entertainment, WindRiver, C2 Microsystems, Sony and many open source collaborators. He is completing a Ph.D. in Computer Engineering at Ecole Polytechnique de Montréal.

He is currently lecturer for the real-time system course at Ecole Polytechnique de Montreal and very active in the Linux kernel community, in the process of getting core pieces of tracing infrastructure merged into the Linux kernel.

Michel Dagenais, Ecole Polytechnique de Montreal

Michel Dagenais is professor and Chairman of the Computer and Software Engineering Department at Ecole Polytechnique de Montreal. He authored or co-authored a large number of scientific publications, free documents and free software packages in the fields of software performance analysis, structured documents on the Web, and object oriented distributed systems. In 1995-1996, during a leave of absence, he was the director of software development at Positron Industries and chief architect for the Power911, object oriented, distributed, fault-tolerant, call management system with integrated telephony and databases. In 2001-2002 he spent a sabbatical leave at Ericsson Research Canada, in the group responsible for Carrier Grade Linux, working on the Linux Trace Toolkit, an open source tracing tool.

Teaching the GNU/Linux Operating System with the LTTNG Kernel Tracer

1 Abstract

Computer systems involve increasingly complex hardware components and software abstractions, e.g. an Xen Hypervisor managing virtual machines running on a multi-core CPU. In the industry, computer engineers are asked to understand these technologies. To tackle this task, companies developing large scale and embedded systems rely on tracing tools to get precise performance monitoring and behavior analysis information.

This paper first discusses the industry need for computer engineers with a good understanding of computer systems. Subsequently, it illustrates how the Linux Trace Toolkit Next Generation (LTTNG) kernel tracer, widely used in the industry, can be used to explain operating system concepts through realistic examples. Finally, the methodology used to introduce these examples to undergraduate computer engineering students is outlined.

2 Introduction

As today's computer systems sophistication level increases, it becomes harder for an aspiring computer engineer to understand the interactions between hardware and high level applications, because Virtual Machine Managers (Hypervisors) and operating systems abstract them. With the appearance of central processing units with multiple cores in commodity hardware, and the widespread use of virtualization, typical industrial systems are increasingly complex. Concurrently, the amount of time spent in curriculum on low level computer organization and system programming may be compressed to make room for software architecture and higher level language courses. Yet, a good understanding of the system level issues helps the computer engineer to develop and debug efficient, possibly real-time, multi-threaded applications.

The Linux Trace Toolkit Next Generation (LTTNG) tracer gathers a sequence of events from the Linux kernel, core of the operating system, which manages the hardware and provides services to user-space processes (standard programs). It provide a precise post-mortem playback of the system execution and helps understanding complex interactions between these programs, becoming especially useful when processes are made of many execution threads or when a lot of inter-process communication is done. It can also extract execution traces from the Xen Hypervisor to help studying interactions between multiple virtual machines running on a computer.

It has been used in an advanced Operating Systems undergraduate course to show the precise sequence of events occurring in common cases such as a packet being received, a disk read/write or a keyboard key being pressed. LTTNG was primarily developed for the computer industry and is currently used on large systems at Google, IBM Research, Autodesk and in the embedded systems fields at Monta Vista, Sony and Wind River. It provides an hands-on, in-depth understanding of the interactions between applications and between the different execution layers, including the user-space, operating system and Hypervisors.

This paper presents an overview of the actual industry requirements regarding computer scientists, briefly reviews the capabilities of LTTNG and then discusses how this tool was used in a course to illustrate, in a direct and intuitive way, the behavior of real, complex, computer systems.

3 Computer industry requirements

Considering today's level of complexity found in computer systems, it comes without wonder that a lot of freshly graduated computer engineers start their career dazing at a task they have not been prepared for.

Dewar and Schonberg pointed out in their paper[4] that teaching only Java to students could diminish their understanding of programming by removing the knowledge of pointers and memory management in general. Albeit the without ever learning the core mechanisms they use in their programming. Adair Dingle also points out[5] that without proper understanding of the way the garbage collector works in Java, students will create inefficient code and will be much more prone to leave stale objects, which leads to clusters of reachable but unused objects and therefore to memory leaks. Although these two articles mainly decry the extensive use of Java as a single programming language in education, they also bring our attention to a phenomenon more tied to this paper. Given that students have a limited amount of time to learn what is required for them to graduate and that the complexity of computer systems increases, they have to specialize in narrower fields of computer engineering.

Following the rules of encapsulation, core system components are presented to students as a black box that they only have to use, without having to really understand it. While this can help to focus more on learning various higher level concepts, it leaves students without any starting point when they face a problem that comes from ill usage or simply bad understanding of the APIs they use. Since they don't understand them, it becomes easy to put the blame on the underlying library rather than on their own misinterpretation.

In addition to the software, hardware complexity is another element responsible for steady increase in computer system complexity. Per Stenstrom, in a Google talk he gave[6] in May 2007, explains that the current move to multi-core architectures will probably be a trend that will increase in a near future. It will however be difficult to benefit from this increasingly common hardware because most of the applications we currently use have been written to be executed sequentially. Turning them into parallel applications will require a redesign that will depend on the understanding of these applications and libraries. However, rewriting the libraries and the operating system today's programmers depend on is not the kind of task they are being prepared to face.

This complexity increase in both software and hardware requires tools to help computer engineers to understand the behavior of the system they use and develop.

4 Analysis tools

Many specialized tools exist to extract information from a running system. We can take for example debuggers (gdb), profilers (system-wide with oprofile or per application with gprof) and memory simulators (Valgrind). An element that does not help in the overall state of the computer technologies is that these tools focus either on a narrow view of what is happening in one process of the system (gdb, Valgrind) and are very intrusive, making their use prohibitive for a system-side analysis or are, like system-wide profilers, only collecting so high-level information (memory usage, CPU usage) that they give little insight into how components behave.

A tool like oprofile is, however, a first step in the good direction. It provides low-level profiling information of both the kernel and user-space. Even though it provides an efficient way to extract

the functions taking the most CPU time and allows use of the CPU performance counters (L1 cache misses, TLB flushes...), it does not allow to freeze the system and analyse the program flow like a debugger would do. Speaking of which, using a kernel debugger like kgdb could help to extract some system-wide information, but stopping the whole system is not always possible to do on SMP machines, especially when what is looked into is the cause of bad system response time.

Therefore, there seems to be a lack of appropriate tools to solve this kind of system-wide performance problem on SMP systems involving multiple processes and multiple execution layers . This is what the Linux Trace Toolkit Next Generation (LTTNG) has been designed to do.

By extracting information from an instrumented Linux kernel, it allows to have a good overview of the system's behavior (scheduling, IPC, memory management, interrupts, faults, and much more). The information is extracted to memory buffers, and optionally written to disk while the system runs, with a minimal impact on the system's performance and behavior. It primarily focuses on kernel tracing, but also allows tracing of user-space applications, which is often required in order to fully understand a trace (some problems, like a video frame dropped, can be easily identified from user-space). After the fact, the Linux Trace Toolkit Viewer (LTTV) analyses both user-space and kernel information as if it was coming from a single source and therefore allows to find the interesting information quickly.

Another, experimental, feature of LTTNG is Hypervisor tracing. There is a proof of concept of LTTNG ported to the Xen Hypervisor to extract its information in separate buffers. Once again, this information is analyzed along with the kernel traces and permits analysis of many hard to characterize aspects of a system, namely the response time of interrupt handlers in a virtualized environment.

The problem one faces when trying to analyze a trace is the huge amount of data collected. This is why most tools are trying to show a summary of the information to users; it is easier to export than megabytes or gigabytes of traces and the users can understand it more easily. By providing the adequate infrastructure to export and analyze gigabytes of information, LTTNG and LTTV can show summary of the information, but also permits to zoom in the details of the execution. This is the ability to link high-level information to low-level execution of the system that makes this tool so helpful to understand interactions between the execution environments.

Another important aspect of LTTNG and LTTV is their extensibility. The LTTNG tracer was built to make it easy for a user to add new events in the kernel source code or in its userspace application, trace the system, and browse in the trace which includes both the standard kernel events and his own extra information. On the trace analysis side, LTTV have some generic views of the information, but, more importantly, is built as a modular infrastructure that permits plugins to hook into the information source to perform specialized analysis on the trace events.

The LTTNG tracer and LTTV analysis tools therefore allow to understand the system's behavior across execution layers. They give the feeling of how things happen in the implementation, beyond simple diagrams in books. The following parts will show how these tools can be leveraged to help teaching computer systems.

5 Explaining abstract concepts with LTTNG

By tracing the detailed sequence of events occurring at the operating system level, LTTNG gathers enough information to drill into the operating system implementation details. However, since the human brain has a hard time parsing gigabytes of trace data, the LTTV trace analysis tool offers different specialized views of the information going from high to low-level. The views available represent the behavior of the operating system as seen from various abstract concepts representation. Understanding operating systems concepts can therefore be done using the right visualization plugin. Interestingly, all the views are synchronized with each other to help linking the information provided by the various analysis plugins. This section will show how abstract concepts (process, paging (copy-on-write), thread, CPU and interruption) can be studied with the help of LTTV.

5.1 Process

A view called “Control flow view” (Figure 1), available in LTTV, shows the single-threaded processes executing on the system. These programs, each made of a single execution thread, have their own virtual memory space and interact with each other and with the computer resources by using the operating system’s “system calls”. We can follow their execution through time, as fork, exit and scheduling events occurs. We can see here, in the case of an incoming SSH connection, that upon reception of a network packet, serviced by an interrupt nesting on top of the “swapper” process, then given to a “softirq” (delayed work executed after an interrupt), the secure remote connexion daemon SSHD process (pid 2893) is woken up. Then, a new process is created (SSHD pid 11599) to manage the session.

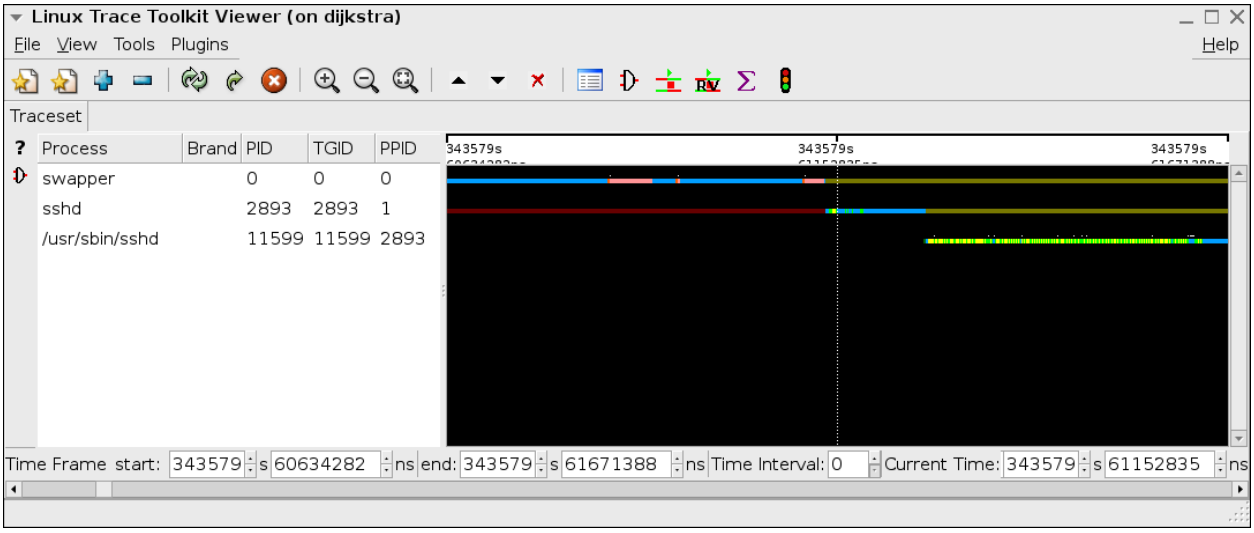

Figure 1: SSH server forking a child process upon new connection (overview).

It can be looked into with much more details than the previous view by zooming in the interesting areas. For instance, if we want to see what happens around the creation of the child SSHD process, we obtain the representation in Figure 2. We can correlate the graphical view with the LTTV “Detailed events list”, which gives the detailed information about each event in the trace. This view is linked with the current time bar of the other views. This detailed list informs us of the

name of the system call used by the SSHD server to create its child : sys_clone. In this system call, executed on behalf of the SSHD process, the kernel allocated memory pages for the new process, issued the fork event, woke up the new task and then the scheduler performed a context switch.

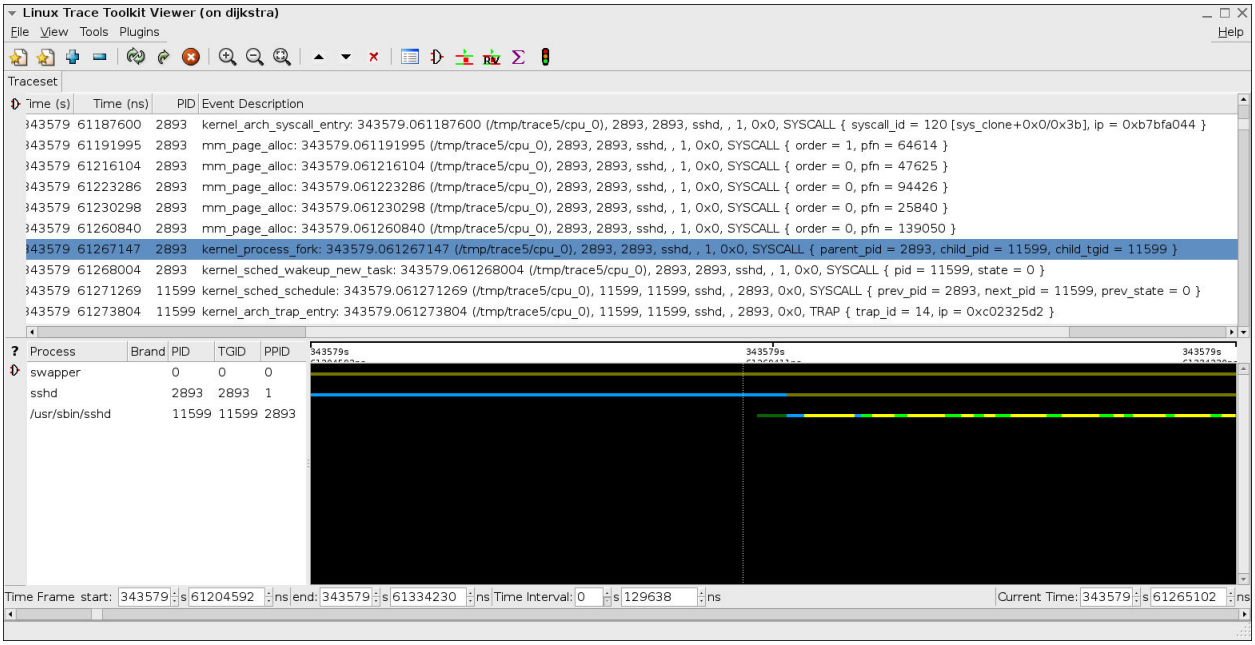

Figure 2: SSH server forking a child process upon new connection (detail).

5.2 Paging (copy-on-write)

Still referring to Figure 2, if we look at the execution following the context switch from process "sshd" to process "/usr/sbin/sshd", we see that they are traps generated by the copy-on-write mechanism used at process fork; each time the new process tries to modify a memory page (including its own stack), a page fault will occur, the parent page will be copied and only then will the child be allowed to write to the new copy.

5.3 Thread

In order to look at threads, we must know that the Linux kernel uses "pid" as an identifier for threads and "tgid" (thread group id) as identifier for processes. Figure 3 shows how OpenOffice manages to create a huge amount of short-lived thread when it runs. They all share the same PGID (11630), but have a different PID (thread identifier).

By zooming into one of these threads execution, can can see that almost invisible dots become filled with execution information (Figure 5.3), where the thread issues multiple system calls to enter in kernel mode from user-space. In the detailed view, we can see that the thread is busy opening the $/tmp$ directory and returns the file descriptor 35. It shows that, in this step of using the GUI to select the file name $/tmp/testfile.odt$, this thread was opening the destination directory.

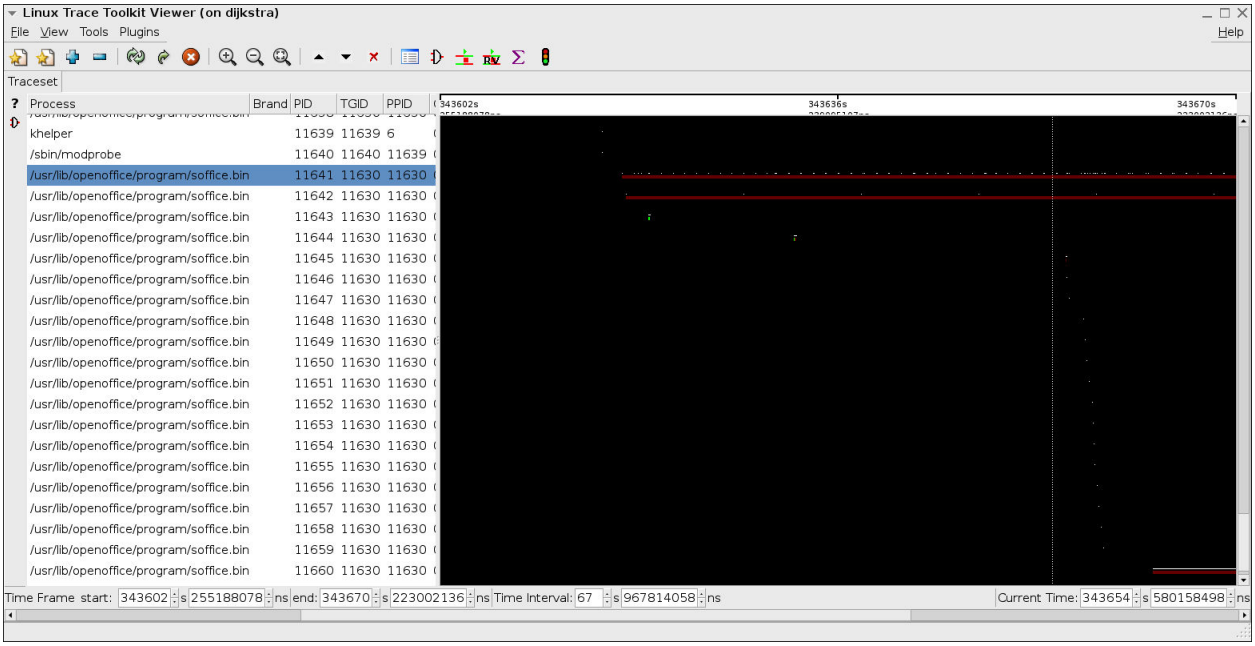

Figure 3: OpenOffice threads saving a file (overview).

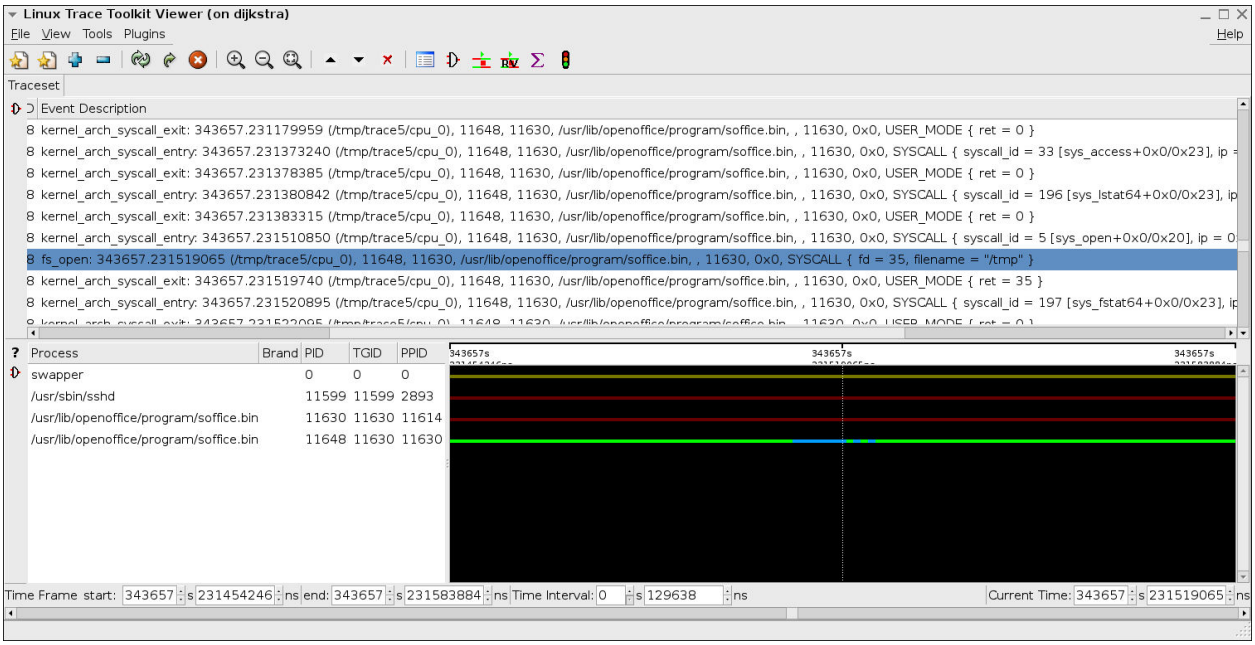

Figure 4: OpenOffice threads saving a file (detail).

5.4 CPU

Looking at processes and threads is not as interesting on an uniprocessor machine as it can be on SMP. Figure 5.5 shows how the scheduler manages to run the X server Xorg and the gdm greeting application on one CPU (graphical applications) while the other is busy running find and exporting the output through SSH. This trace is taken on a dual core AMD64. It can be used as a good example to show how the scheduler manages each CPU independently, except for thread migration. This figure also introduces a new view : the resources view. It shows how the various resources of the system are used (CPU, IRQ, softirqs). On the CPU aspect, we see that CPU idle time is represented by the grey color, while CPU active is presented as a white line.

5.5 Interruption

Interrupt balancing between the two cores can also be seen in the resources view in Figure 5.5. Actually, the network card (eth1) has its interrupts serviced by the second core (CPU 1). However, the local APIC timer (IRQ 239) is sometimes serviced by CPU 0, and sometimes by CPU 1. This view is powerful in showing CPU usage and how fairly the computational load is spread on many CPUs.

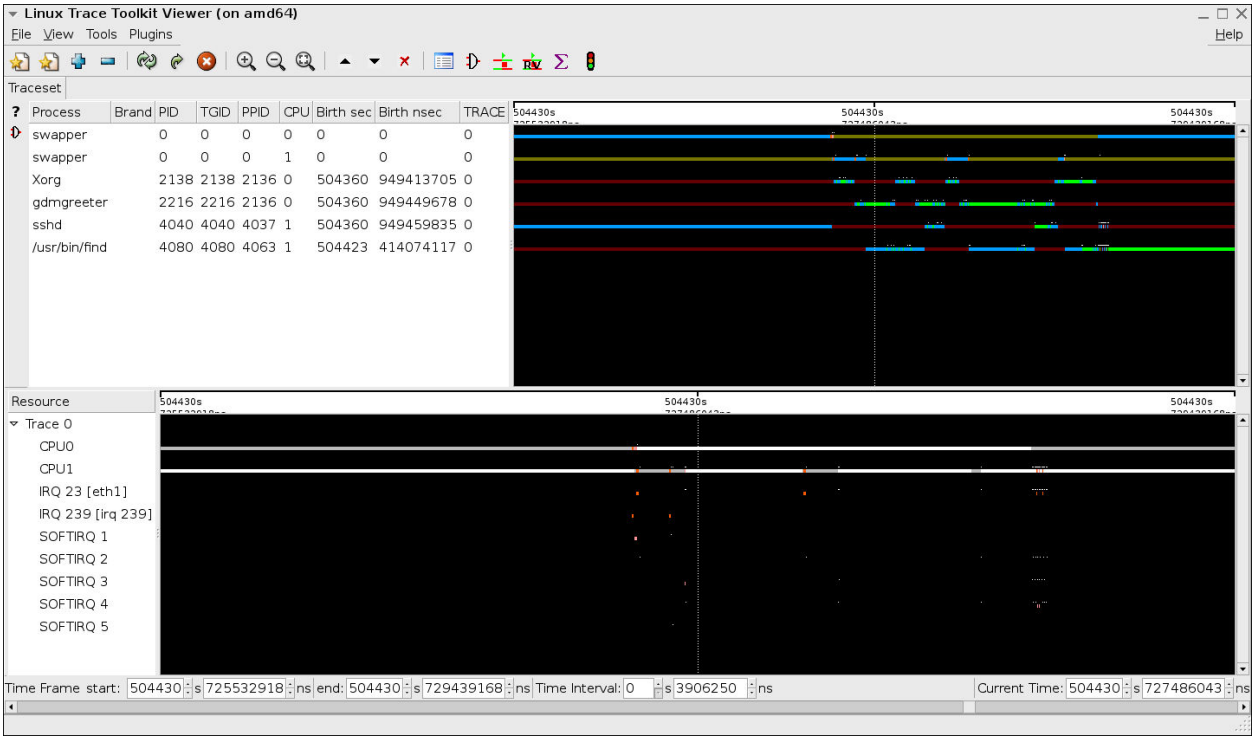

Figure 5: SMP scheduling and interrupt servicing.

6 Experience in teaching

This section discusses how LTTNG has been used in a real-time operating systems undergraduate course to teach operating system concepts through concrete examples. Since the focus of the course was mostly to teach real-time concepts, only a limited amount of time was available to bring operating system concepts, which was in fact a prerequisite of the course.

Given that the concepts had already been explained theoretically and that the students were familiar with them, a guided tour has been prepared to illustrate these concepts to the class. Before the presentation, a few traces has been taken, both on uniprocessor and multiprocessor machines, where interesting execution scenarios were recorded in a trace. The neighborhood surrounding interesting locations in the traces were identified using LTTV filtering capabilities. Their time stamp were noted to help finding back the interesting events in the traces.

The student feedback has been very good during the presentation. A verification of their understanding was possible by asking them what concept or important event was occurring in the graphical view presented to them. By using the LTTV trace viewer rather than static screen shots, it was possible to guide them through the detailed steps of the system execution. At the end of the presentation, they have been very participative when asked for execution scenarios they were interested to look at. Live traces have been collected and shown with LTTV to answer their questions immediately.

If the ideal setup would have been available, each student would have access to a computer running a Linux kernel instrumented with LTTNG and would have to identify by himself execution scenarios given as exercices. Fulfillment of the task could be verified by requiring screen shots of the information identified or by asking for a copy of the trace and the time stamp at which the event occurred to the teacher can find it back in the trace. In order to facilitate these experiments, the custom setup requiring a patched Linux kernel should be made available.

Students were very interested in the presentation. Immediately afterward, a few of them came to express their interest in pursuing work on LTTNG/LTTV for their end of undergraduate program project.

7 Tracer use in the industry

Companies from various application fields use tracers in their work : as Martin Bligh presented at Ottawa Linux Symposium 2007,[1] Google uses tracers to identify the cause of hard to reproduce bugs and performance issues on production servers. He goes in the details on how this tool can be used in production environment to understand such problems. So does Autodesk in the development phase of their video edition applications and IBM Research.

In the embedded systems field, Wind River already use LTTNG in their Linux products and Montavista includes it in its Linux distribution used by, among others, Sony. LTTNG use to pinpoint realtime latency issues as been thoroughly presented at the Embedded Linux Conference 2006[2] and Ottawa Linux Symposium 2006.[3] Since this tool is already widely used in the industry to solve particularly difficult problems, showing students how to use it can become a competitive advantage for them on the job market.

When confronted by a difficult problem, students should feel confident of their understanding, given the right tools to tackle the task, not mystified by the complexity of the environment in which that have to develop. This is, at least, what is expected from good engineering candidates by the industry. This is why giving them a tool that helps them digging as deep as necessary in the core of the system is so important, both to help them to understand the system they rely upon and make them confident of their ability to solve the problems they will encounter.

8 Student projects

Given the extensible nature of the project, students can easily add their own instrumentation in the kernel code to study aspects of the system that interest them particularily and create plugin modules to LTTV that will perform specialized analysis.

Linux, LTTNG and LTTV being distributed under the GPL licence, such characteristic opens the door to external contributions from universities as well as from the industry. Many plugin views already integrated in the LTTV project come from such student projects. It allows them to experiment and extend their understanding of a particular operating system aspect.

References

[1] Martin Bligh, Mathieu Desnoyers, and Rebecca Schultz. Linux kernel debugging on google-sized clusters. In *Proceedings of the Linux Symposium*, July 2007.

[2] Mathieu Desnoyers and Michel Dagenais. Low disturbance embedded system tracing with linux trace toolkit next generation. In *Embedded Linux Conference 2006*, April 2006.

[3] Mathieu Desnoyers and Michel Dagenais. The lttng tracer : A low impact performance and behavior monitor for gnu/linux. In *Proceedings of the Linux Symposium*, July 2006.

[4] Robert B.K. Dewar and Edmond Schonberg. Where are the software engineers of tomorrow? *STSC Crosstalk*, January 2008.

[5] Adair Dingle. Reclaiming garbage and education: Java memory leaks. *J. Comput. Small Coll.*, 20(2):8–16, 2004.

[6] Per Stenstrom. Multi-core architecture. In *Google Speaker Series*, May 2007.