Process Tracing with the Linux Trace Toolkit

B. B. Ramya, V. Pavithra, and B. Thangaraju

Debugging an application or kernel program can be done with the GNU Debugger (gdb), Linux Kernel Source Level Debugger (kgdb), or Linux Kernel Debugger (kdb), but if we want to trace a particular process, we must use the strace utility, which will trace system calls and signals. Strace will trace only one process and present the result in text form. To trace many processes in a given period of time, Linux Trace Toolkit (LTT) is a better choice.

LTT is distributed as free software under GPL. Applying an LTT-patch to the corresponding kernel will create a module to trace 48 events. The trace toolkit provides a daemon, which will capture the events and write it to disk. The provided trace visualizer is used to analyze the tracing data in three different forms (viz., event graph, process analysis, and raw event descriptions).

LTT is useful for systems administrators for analyzing the performance of the system. It is useful for programmers for getting details of the interaction between kernel and user-level applications and for embedded/real-time programmers for getting information about real- and non-real-time tasks' behavior. The theoretical aspect of LTT is skipped in this article because it is well documented in the Linux Trace Toolkit Reference Manual.

Enabling LTT with 2.4 kernel is very straightforward in the sense that an available trace toolkit contains all the necessary patches with documentation. It can be found at:

```
http://www.opersys.com/ltt/downloads.html
```

However, enabling LTT with the 2.6 kernel requires some guidance to make the task successful and to save time. In this article, we'll guide you through the necessary packages, patches, and implementation details for 2.6 kernel. We will also demonstrate the usage of the LTT with a simple example.

Linux Trace Toolkit with 2.6 Kernel

The procedure for building the Linux kernel has been changed in 2.6. To enable LTT and relayfs, you must first apply the corresponding patches with the source code. These patches will modify the source code in the respective places. The `make xconfig` command will show the menu with a new look in 2.6. As shown in Figures 1 and 2, the tracing support and relayfs are enabled.

Relayfs is a file system, which is used to move data from kernel to user space in an efficient manner. After configuring the kernel, usually in 2.4, we would execute `make dep`, `make bzImage`, and `make modules` command. Now, all these will work together if you execute `make`.

To install kernel modules, the existing installation command will not work in 2.6 kernel. For this, you must download modutilities and install it. Then, execute the `make modules_install` command, which will install all the kernel modules, and the `make install` command, which will update the boot loader. Now you can boot the system with the LTT-enabled 2.6 kernel.

After installing the trace toolkit, we need to mount the relayfs then execute a daemon for a specific interval of time. This tracer daemon will capture all the events for the given time period and store them in the specified file name. The captured data can be analyzed by a trace visualizer.

Necessary Packages

The following packages should downloaded from their respective Web sites:

Kernel —
`ftp://ftp.kernel.org/pub/linux/kernel/v2.6/ \ linux-2.6.3.tar.bz2`

Trace Toolkit — `http://www.opersys.com/ftp/pub/LTT/ \ TraceToolkit-0.9.6pre2.tgz`

Mod-Utils — `http://www.kernel.org/pub/linux/kernel/ \ people/rusty/modules/module-init-tools-3.0-pre9.tar.gz`

Patches for relay file systems — `http://www.opersys.com/ftp/ \ pub/relayfs/patch-relayfs-2.6.0-test11-031203.bz2`

Linux trace toolkit patch for 2.6 kernel —
`http://www.opersys.com/ftp/pub/relayfs/LTT/ \ patch-ltt-linux-2.6.0-test11-vanilla.bz2`

The above two patches are for kernel building with LTT and relayfs enabled. To use relayfs on LTT, download the patch from:

```
http://www.opersys.com/ftp/pub/relayfs/LTT/ \
  patch-ltt-on-relayfs-0.9.6pre2-031203.bz2
```

Implementation

Download the 2.6.3 kernel source from the abovementioned Web site along with the patches and untar the kernel in /usr/src directory:

```
bzip2 -d  /usr/src/linux-2.6.3.tar.bz2
tar xvf linux 2.6.3.tar
```

This will create a linux-2.6.3 directory under /usr/src. Then, copy patch-relayfs-2.6.0-test11-031203.bz2 and patch-ltt-linux-2.6.0-test11-vanilla.bz2 into the linux-2.6.3 directory.

Unzip them using:

```
bzip2  -d  patch-relayfs-2.6.0-test11-031203.bz2
bzip2  -d  patch-ltt-linux-2.6.0-test11-vanilla.bz2
```

then apply the above patches to the Linux kernel:

```
patch -p1 < patch-relayfs-2.6.0-test11-031203
patch -p1 < patch- ltt-linux-2.6.0-test11-vanilla
```

These patches will modify the kernel files. Next, we need to configure and rebuild the kernel.

Enable the tracing option and relay file system in the configuration menu as in Figures 1 and 2. Then a `make` will build the kernel and create the modules.

Figure 1 *Enabling tracing support in kernel configuration*

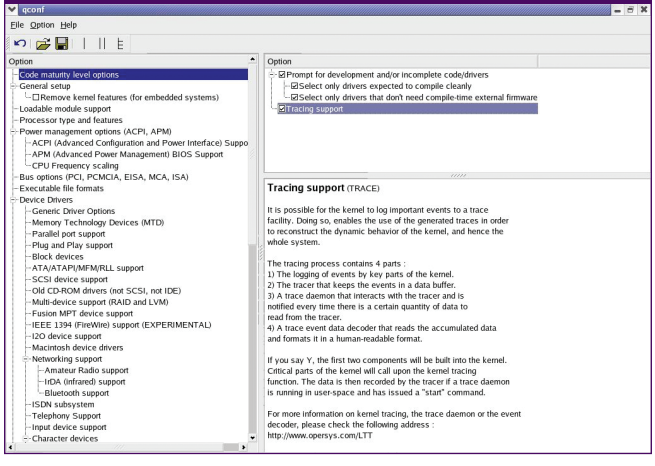
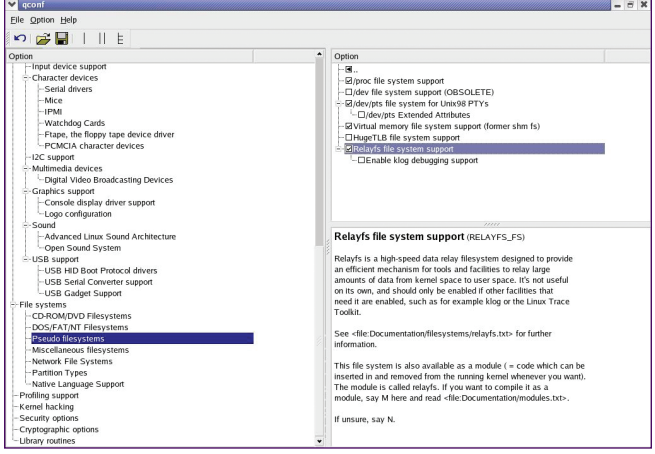


Figure 2 *Enabling relayfs in kernel configuration*



The modules of the 2.6.3 kernel will not be loaded because they come with version 2.4 of module-init tools. So, we must get the latest version of the mod-utils and configure it for the kernel using:

```
./configure   --prefix=/
make moveold
make
make install
```

to translate the old /etc/modules.conf into /etc/modprobe.conf with the ./generate-modprobe.conf script that comes with module-init-tools:

```
./generate-modprobe.conf  /etc/modprobe.conf
```

Run `make modules_install` to install the kernel modules. Next, `make install` will update the boot loader and reboot the system with the new kernel.

Next, traverse to the /usr/src/linux-2.6.3 directory and untar the TraceToolkit-0.9.6pre2.tgz:

```
tar xzvf TraceToolkit-0.9.6pre2.tgz
```

This will create a TraceToolkit-0.9.6pre2 directory and change into that directory. Apply the patch:

```
bzip2  -d patch-ltt-on-relayfs-0.9.6pre2-031203.bz2
patch -p1 < patch-ltt-on-relayfs-0.9.6pre2-031203
```

Next, configure the tracetool using:

```
./configure
make
make install
```

Mount the relay file system:

```
mkdir /mnt/relay
mount -t relayfs relayfs /mnt/relay
```

Note that you can also make an entry in the /etc/fstab file for relayfs so that you need not mount the relayfs every time you restart the system:

```
relayfs   /mnt/relay   relayfs  defauls   1 1
```

Now, the tracetool is up and ready to trace the system.

Working with LTT

We are interested in capturing the system events along with the following program's execution trace. The program calls the fork system call, which will create a new process:

```
int main (void)
{
fork ( );
printf ("Hello Fork%d\n", getpid( ));
return 0;
}
```

To get a trace of the system during the execution of this program, we start the trace daemon for 5 sec as shown below:

```
tracedaemon  -ts5 ./out1.trace ./out.proc
```

`tracedaemon` is the command to run the daemon for a given time period, where t is for time, s for time unit in seconds, and 5 for the given time period. Out1.trace and out.proc files are used to store the trace data for analysis.

To get process details in a graphical format, execute the following in the shell prompt:

```
tracevisualizer -g  out1.trace out.proc outfile
```

The `tracevisualizer` command will launch the trace toolkit, and the `-g` option is for graphical format. The next two fields are the input files, which we specified to store the data collected by the daemon process earlier. The last argument, `outfile`, is where the trace and analysis are written in text format.

The event graph of the trace is shown in Figure 3. It gives information about the processes that were executing during the trace, along with their process ids. The right side of the figure shows the entire trace of the process. It shows the details of what system calls, signals, traps, hard and soft IRQs were handled for the specific process that is highlighted in the left box, along with its interactions with the kernel and any other processes that are executing.

The highlighted bar shows the trace of myfork that is executable of fork_demo.c. When the CPU executes a system call like fork, the CPU will change mode from user to kernel. The system call will be executed in kernel mode, and the fork system call will spawn a new child (unnamed child with pid 274 in Figure 3).

Processes of interest or system information can be analyzed by the process analysis method as shown in Figure 4. This provides information about the number of system calls the process has called during its execution and the total time the kernel has taken to execute each system call. It also lists the process characteristics such as the number of system calls, traps the process has made, the time spent by the process waiting for I/O, and the quantity of data read and written to files.

The "Raw Trace" view is there to list all events that were logged by the data acquisition module, which is shown in Figure 5. The highlighted bar shows a Scheduler change for process id 271 (i.e., myfork).

Conclusion

The Linux Trace Ttoolkit is a constructive tool to help all kinds of Linux users see and understand system events. In this article, we described how to enable LTT with 2.6.3 kernel, trace simple process events, and analyze trace data in different forms.

Acknowledgement

The authors are very grateful to Mr. Karim Yaghmour, creator of the Linux Trace Toolkit, embedded and real-time Linux expert.

References

Linux Trace Toolkit Reference Manual available at —
`http://www.opersys.com/ltt/dox/ltt-online-help/index.html`
Trace Toolkit -0.9.5a.tgz available at —
`http://www.opersys.com/ltt/downloads.html`

B. B. Ramya and V. Pavithra work as Project Trainees, and B.Thangaraju is a Manager in the Embedded and Product Engineering Solutions (E&FPE), Wipro Technologies in Bangalore, India. B. Thangaraju can be reached at `bt_raju@vsnl.net`.

Figure 4 *Per-process analysis*

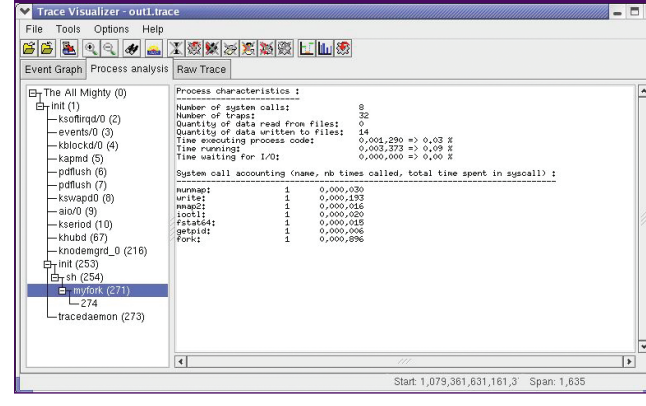


Figure 3 *Event graph*

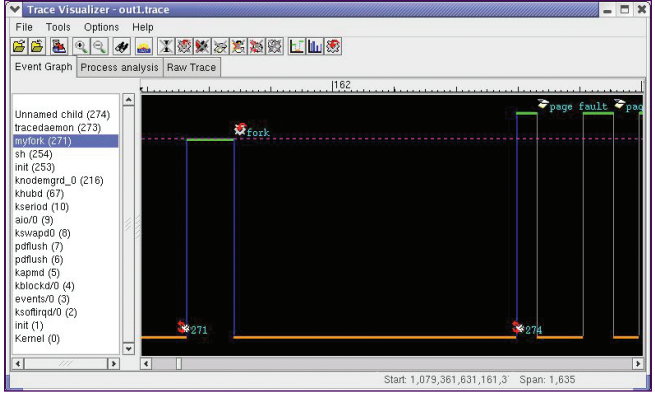


Figure 5 *Raw trace*