Methods of kernel Tracing

Prof. Laxmi Bewoor, Parool, Priya Kumari, Shristi A. Mishra, Sheetal Chawla
Department of Computer Engineering
Vishwakarma Institute of Information Technology,
Savitribai Phule Pune University,Pune

Abstract—Kernel tracing is essential for extracting appropriate recurring runtime execution patterns and several inter process communication patterns for a system. There have been tools developed to understand process execution at kernel levels, but extracting the relevant information by manually analyzing the traces is a time consuming process. This has motivated us to review all the current methods of kernel tracing and find the most suitable of all utilities available. This would involve understanding of the kernel events . We would use Ltt-ng as a tool to generate the traces in Linux Kernel. The main objective is to improve performance of the applications and reduce the workload. Also we can observe various types of kernel events and their effect on the system.By performing this analysis, the user will be able to view the kernel processes as well as their execution patterns.This survey basically compares Ltt-ng with other utilities available for tracing the linux kernel.

Index Terms—Frequent pattern mining,kernel tracing,Scheduling,Data Mining,operating systems,Multicore systems

1 INTRODUCTION

THE kernel is a part of operating system that is responsible for controlling execution of processes like creation, termination, suspension, communication.It also allocates main memory for the execution of the process.Linux kernel has an extensive tracing infrastructure that is quite useful for debugging.Tracing of kernel is important to uncover useful information like performance issues,to identify bottlenecks in execution of processes and it can further be extended for optimizing the kernel.There are various tracing tools developed to inspect the Linux kernel , because manual inspection of the logs or traces is time consuming.For the Linux kernel, various tools and commands are available for generating the traces like top that provides a snapshot of currently running processes that can be sorted using parameters like CPU usage or memory usage. Another utility is 'strace' that helps to record the interrupts, system calls and signals that a process receives.There is also an inbuilt Linux tracer like 'ftrace' which is "function tracer" but provides services beyond it. It is used for event tracing and other functions like analyzing latencies.Yet another utility available is 'ktrace' which generates traces in an output file. The output file contains I/O , signals and system calls. Here, in this survey, we compare these tools with the Linux Trace Toolkit(Ltt-ng) , which is an open source tracing tool that not only traces kernel processes, but also traces the user processes.A session needs to be created for which the traces and logs will be generated. These traces can further be analysed to understand, in deep, the execution patterns of processes.This analysis can be done by mining the kernel trace data.While performing mining inter-dependencies between the processes and time varying characteristics of processes should be taken into consideration.

2 METHODS OF KERNEL TRACING

2.1 Tracing using ktrace

Ktrace is a utility included with certain versions of BSD Unix and Mac OS X that traces kernel interaction with a program and dumps it to disk for the purposes of debugging and analysis. Traced kernel operations include system calls, processing, and I/O. The aim of ktrace is to provide the user with the state of the system on the occurrence of specific events. Ktrace comprises of two components, user level module and system level module. The kernel level module is the actual tracing module whereas the user level module acts as the front end of the tracing framework.

2.1.1 User Module

The user module provides a user interface. It interacts with the kernel module using device driver software. It passes the following information to the kernel module

1. Event Mask : It is a mask for the list of eventsthat the user wants to be traced.
2. Variable Mask : It is a mask for the list ofvariables that the user wants to be traced. 3.Pid : It is the pid of an already running process which the user wants to be traced.

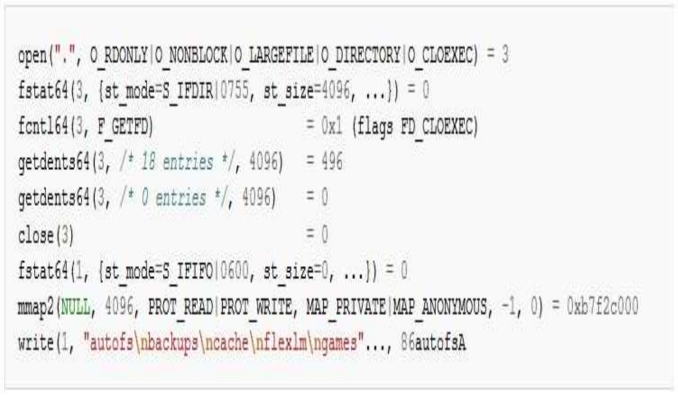
2.1.2 Kernel Module

The kernel module obtains the the tracing information from the user module with the help of a device driver software.At the occurrence of an event to be traced, the kernel module has to store the values of all the required variables. So the basic tracing framework is such as, the user module passes the tracing options to the kernel module.Then the data logged by the kernel module is continuously transferred to each user module which happens continuously till the time the tracing goes on.

2.2 Use of strace

Strace is a tool which is used to trace system calls and signals.It is a diagnostic and debugging utility for Linux. It is used to observe and study the interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state. The operation of strace is made possible by the kernel feature known as 'ptrace'. The most common use of strace is to start a program using strace, which prints a list of system calls made by the program. This is useful if the program is continually crashing many times, or does not behave as expected; for example using strace may reveal that the program is attempting to access a file which does not exist or cannot be read.

As strace is specifically used for only detailing the system calls, it cannot be used to detect as many problems as a code debugger such as GNU Debugger (gdb). It is, however, easier to use than a code debugger, and is an extremely useful tool for system administrators. The following is an example of typical output of the strace command:



```
open(".", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
fcntl64(3, F_GETFD)                    = 0x1 (flags FD_CLOEXEC)
getdents64(3, /* 18 entries */, 4096)  = 496
getdents64(3, /* 0 entries */, 4096)   = 0
close(3)                               = 0
fstat64(1, {st_mode=S_IFIFO|0600, st_size=0, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7f2c000
write(1, "autofs\nbackups\ncache\nflexlm\ngames"..., 86autofsA
```

Fig. 1: Output of strace

2.3 Debugging the kernel using Ftrace

Ftrace is a tracing utility built directly into the Linux kernel. One of the advantages that Ftrace brings to Linux is the ability to monitor what is happening inside the kernel. As such, this makes finding problem areas or simply tracking down bugs more manageable. Ftrace has the ability to show the events that lead up to a crash, which gives a better chance of finding exactly what caused it and can help the developer in creating the correct solution.

2.3.1 Setting up Ftrace

The API to interface with Ftrace is located in the Debugfs file system.Typically, that is mounted at /sys/kernel/debug. For more ease we can create a /debug directory and mount it there. We can choose our own location for Debugfs.When Ftrace is configured, it will create its own directory called tracing within the Debugfs file system. The action on the terminal is shown below:

[˜]# cd /sys/kernel/debug/tracing [tracing]#

2.3.2 Function tracing

Ftrace uses the -pg option of gcc to have every function in the kernel call a special function mcount.To find out which tracers are available,we can simply cat the available tracers file in the tracing directory.

[tracing]# cat available_tracers

To enable the function trace we have to put the following command

[tracing]#echo function *current tracer*
[*tracing*]#*catcurrent tracerfunction*

Difference between LTTng and ftrace is that in ftrace there is no availability of dynamic tracepoints and that live monitoring can be done using LTTng which is not possible with ftrace.

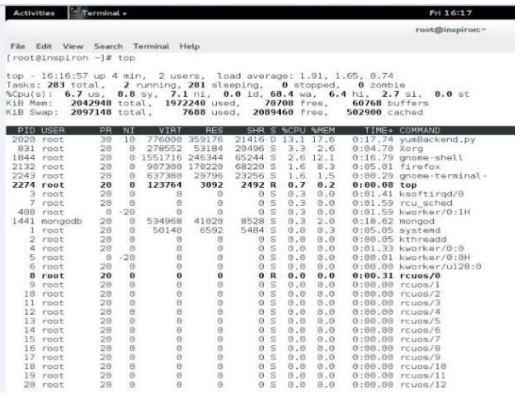
2.4 Top utility

Top provides a closer look at processor activity in real time. It displays a list of the most CPUintensive tasks on the system and can provide an interactive interface for manipulating processes. It can sort the tasks by CPU usage, memory usage and runtime.

2.4.1 Options with top

-d Specifies the delay between screen updates.

-p Monitor only processes with given process id. This flag can be given up to twenty times. This option is neither available interactively nor can it be put into the configuration file.

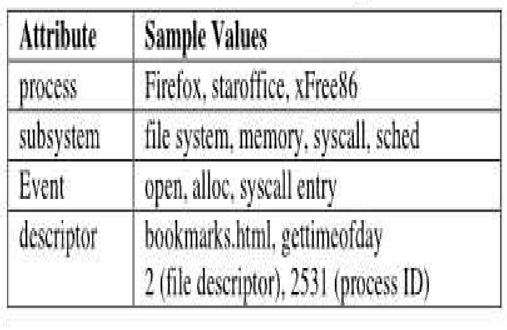


```
[root@inspiron ~]# top

top - 16:16:57 up 4 min,  2 users,  load average: 1.91, 1.65, 0.74
Tasks: 283 total,   2 running, 281 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6.7 us,  8.8 sy,  7.1 ni,  0.0 id, 68.4 wa,  6.4 hi,  2.7 si,  0.0 st
KiB Mem:   2042948 total,  1972240 used,    70708 free,    60768 buffers
KiB Swap:  2097148 total,     7688 used,  2089460 free,   502900 cached

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 2020 root      30  10  776000 359176  21416 D 13.1 17.6   0:17.74 yumBackend.py
  831 root      20   0  278552  53184  20496 S  3.3  2.6   0:04.70 Xorg
 1844 root      20   0 1551716 246344  65244 S  2.6 12.1   0:16.79 gnome-shell
 2132 root      20   0  987300 170220  68220 S  1.6  8.3   0:05.01 firefox
 2243 root      20   0  637380  29796  23256 S  1.6  1.5   0:00.29 gnome-terminal-
 2274 root      20   0  123764   3092   2492 R  0.7  0.2   0:00.08 top
    3 root      20   0       0      0      0 S  0.3  0.0   0:01.41 ksoftirqd/0
    7 root      20   0       0      0      0 S  0.3  0.0   0:01.59 rcu_sched
  408 root       0 -20       0      0      0 S  0.3  0.0   0:01.59 kworker/0:1H
 1441 mongodb   20   0  534968  41020   8528 S  0.3  2.0   0:18.62 mongod
    1 root      20   0   50140   6592   5484 S  0.0  0.3   0:05.05 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.05 kthreadd
    4 root      20   0       0      0      0 S  0.0  0.0   0:01.33 kworker/0:0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.01 kworker/0:0H
    6 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kworker/u128:0
    8 root      20   0       0      0      0 R  0.0  0.0   0:00.31 rcuos/0
    9 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/1
   10 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/2
   11 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/3
   12 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/4
   13 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/5
   14 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/6
   15 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/7
   16 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/8
   17 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/9
   18 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/10
   19 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/11
   20 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuos/12
```

Fig. 2: Output of top

2.5 Linux Trace Toolkit

Linux Trace Toolkit , an open source tracing framework for Linux allows extensive and transparent reporting of process,event,descriptor and operating system activity. Thousands of kernel-event records can be traced every second. Each kernel event has a multi-feature ordered pair containing, operating system sub-system involved with executing the event, process record which caused the event, the event type, the address or descriptor for any resource , and the time at which the event occurred. LTTng allows the user to see thoroughly , information about the processes and events that were running during the trace period, including when context switches occurred, how long the processes were blocked for, and how much time the processes spent executing vs. how much time the processes were blocked,the amount of memory allocated etc..Sample values for the attributes are shown in fig.3



Attribute	Sample Values
process	Firefox, staroffice, xFree86
subsystem	file system, memory, syscall, sched
Event	open, alloc, syscall entry
descriptor	bookmarks.html, gettimeofday 2 (file descriptor), 2531 (process ID)

Fig. 3: Trace Attributes with sample values

A brief description of the sample values:

- **Syscall Entry**:A system call is ready to start execution in the kernel mode.The Linux kernel contains a centralized mechanism to invoke system calls and this central point corresponds to the system call entry event.
- **Syscall Exit**:A system call is ready to return to the user mode.The return from a system call also takes place through a central mechanism and it corresponds to the system call exit event
- **Page Alloc**:A page of memory is assigned by the linux kernel.
- **Page Exit**:A page of memory is freed by the linux kernel.
- **Schedule In**:The scheduler is invoked by the linux kernel.The actual point is the beginning of the context switch function in the linux kernel.
- **Schedule Out**:The scheduler's job is finished.The actual point is the end of the context switch function in the linux kernel.
- **Free Memory**:It denotes the amount of free memory of the entire system.
- **Total Memory**:It denotes the total memory of the system.

LTTng is a set of software components.These components allow interaction between the Linux kernel and user applications.It also controls the tracing sessions i.e enabling or disabling of events,starting or stopping of tracing, and more. Those components are clusterd into the following packages:

LTTng-tools: libraries and command line interface to control tracing sessions.
LTTng-modules: Linux kernel modules for tracing the kernel.
LTTng-UST: user space tracing library.

The data gathered after tracing the Linux kernel or a user space application which is initially in CTF(Common Trace Format) can be viewed using:

- **babeltrace** is a command line utility which converts trace formats. It supports the format used by LTTng, CTF, as well as a basic text output which may be greped.
- **Trace Compass** is an Eclipse plugin used to visualize and analyze various types of traces, including LTTng's. It also comes as a standalone application.

2.5.1 Advantages of using LTTng

- LTTng can be used for analyzing the system where delays occur.
- To see how processes interact with respect to scheduling, interrupts, synchronization primitives, etc.
- It can be used to log program execution details from a patched Linux kernel and then perform various analyses on them, using console-based and graphical tools.

2.5.2 Why LTTng?

The unique features of LTTng is that it produces correlated kernel and user space traces.These traces have lowest overhead as compared to other solutions. The trace files are produced in the CTF format, an optimized file format for production and analyses of multi-gigabyte data. LTTng is developed by a community of passionate developers and has been actively used since 10 years. It is currently available on all major desktop, server, and embedded Linux distributions.

The main interface for tracing control is a single command line tool named lttng. This lttng can create several tracing sessions, enable or disable events on the fly, filter them efficiently with custom user expressions, start or stop tracing, and do much more. Traces can be recorded on disk or sent over the network, kept totally or partially, and viewed once tracing becomes inactive or in real-time.

3 CONCLUSION

Kernel tracing is an important process to derive suitable recurring, run-time execution patterns and different inter-process communication patterns for a system. There are various methods of kernel tracing using utilities like 'ktrace', 'strace', 'ftrace' discussed in the paper. Moreover, there are tools developed to inspect the Linux kernel, because the manual inspection of the logs/traces is pretty time consuming. The Linux Trace Toolkit(LTT), is an efficient, open source tracing framework for Linux, that allows extensive and transparent reporting of processes, events, descriptors and operating system activities. The gathered data can be viewed using two tools - Babeltrace and Trace Compass. This data can be analysed by the system programmer in order to optimize the scheduler by improving its performance and reducing the workload on the system.

REFERENCES

[1] G. Anderson, T. Marwala, and F. V. Nelwamondo, "Use of data mining in scheduler optimization,"

[2] C. LaRosa, L. Xiong, and K. Mandelberg, "Frequent pattern mining for kernel trace data,"

[3] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: Current status and future directions,"

[4] Xiaonan, "Application of data mining in scheduling of single machine system,"

[5] G. N. Matni and M. R. Dagena, "Operating system level trace analysis for automated problem identification,"

[6] E. Budilovsky, "Kernel based mechanism for high performance i/o,tel aviv university, apr 2013,"

[7] A. C. C. Aggarwal, M. A. Bhuiyan, and M. AlHasan, "Frequent pattern mining algorithms: A survey frequent pattern mining, 2014. isbn: 978-3-319-07820-5,"

[8] S. Nasreen, M. A. Azamb, K. Shehzada, U. Naeemc, and M. A. Ghazanfara, "Frequent pattern mining algorithms for finding associated frequent patterns for data streams: A survey the 5th international conference on emerging ubiquitous systems and pervasive networks (euspn-2014),"

[9] P. Shendge and T. Gupta, "Comparitive study of apriori and fp growth algorithms, indian journal of reasearch. volume : 2, issue : 3, march 2013,"

[10] P. Kulkarni, *Knowledge Innovation Strategy*. Pune: Bloomsbury Publication, 2015.

[11] "Lttng is an open source tracing framework for linux.." https://lttng.org/.

[12] "Lttng." https://en.wikipedia.org/wiki/ LTTng.

[13] "System monitoring tools." http://www.cyberciti.biz/tips/ top-linux-monitoring-tools.html.

[14] "Top." http://man7.org/linux/ man-pages/man1/top.1.html.

[15] "Linux / unix command: top." http://linux.about.com/od/commands/ l/blcmdl1top.htm.

[16] "ftrace - function tracer." https: //www.kernel.org/doc/Documentation/ trace/ftrace.txt.

[17] "Ftrace." http://elinux.org/Ftrace.