**FTRACE**

By

Regards

Ashish Tanwer

Masters, Computer Science, Stony Brook University

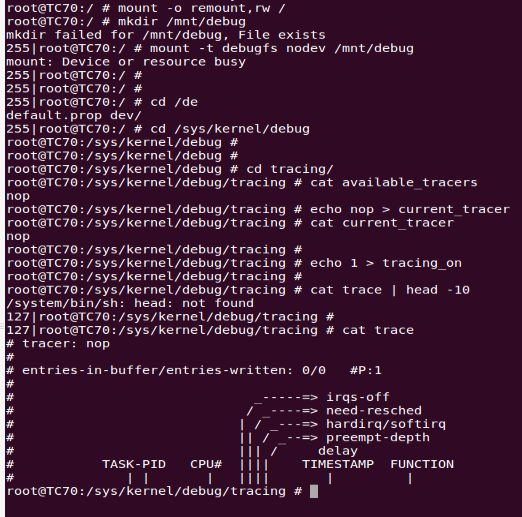
1063 morse avenue, apt 3-200,Sunnyvale,California-94089

[ashishtanwer@gmail.com](mailto:ashishtanwer@gmail.com), +1-[(631) 551-3060](tel:%28631%29%20551-3060)

<http://ashishtanwer.info>, <http://in.linkedin.com/in/ashishtanwer>

<http://www.youtube.com/user/ashishtanwer> <https://github.com/ashishtanwer>

Ftrace kernel tool that provides the low-overhead logging of Linux kernel events.



**USING** **FTRACE**

Ftrace has its control ﬁles in the debugfs system.This is usually mounted in /sys/kernel/debug. If it is not already mounted, then you can mount it yourself with:

# mount -t debugfs nodev /sys/kernel/debug

Many Developpers prefer to have this directory at the root of the system

# mkdir /mnt/debug

# mount -t debugfs nodev /mnt/debugfs

That creates a /debug/tracing subdirectory which is used to control ftrace and for getting output from the tool.

**Note**: all the command of this article will be pass in the /debug/tracing/ directory

**Choosing a tracer**

To find out which tracers are available, simply cat the available\_tracers file in the tracing directory:

# cat available\_tracers

function sched\_switch nop

To enable, for example, the function tracer, just echo "function" into the current\_tracer file.

# echo function > current\_tracer

# cat current\_tracer

function

**Note**: Ftrace can handle only one tracer at the same time

**Using a tracer**

At this point Ftrace is ready to trace the function, to start it just echo 1 in the tracing\_on file

# echo 1 > tracing\_on

To stop the trace, just echo 0 in the same file:

# echo 0 > tracing\_on

The trace is contained in the trace file, here is an example of the output from a function trace. The header helps to decode the various fields in the trace.

# cat trace | head -10

# tracer: function

#

# TASK-PID CPU# TIMESTAMP FUNCTION

# | | | | |

avahi-daemon-630 [000] 9507.400848: vfs\_read <-sys\_read

avahi-daemon-630 [000] 9507.400848: rw\_verify\_area <-vfs\_read

avahi-daemon-630 [000] 9507.400848: do\_sync\_read <-vfs\_read

avahi-daemon-630 [000] 9507.400848: sock\_aio\_read <-do\_sync\_read

avahi-daemon-630 [000] 9507.400848: alloc\_sock\_iocb <-sock\_aio\_read

avahi-daemon-630 [000] 9507.400848: unix\_stream\_recvmsg <-sock\_aio\_read

**Tracing a specific process**

Perhaps you only need to trace a specific process, or set of processes. The file set\_ftrace\_pid lets you specify specific processes that you want to trace. To just trace the current thread you can do the following:

[tracing]# echo $$ > set\_ftrace\_pid

The above will set the function tracer to only trace the bash shell that executed the echo command. If you want to trace a specific process, you can create a shell script wrapper program.

[tracing]# cat ~/bin/ftrace-me

#!/bin/sh

DEBUGFS=`grep debugfs /proc/mounts | awk '{ [print](http://omappedia.org/wiki/Installing_and_Using_Ftrace) $2; }'`

echo $$ > $DEBUGFS/tracing/set\_ftrace\_pid

echo function > $DEBUGFS/tracing/current\_tracer

exec $\*

[tracing]# ~/bin/ftrace-me ls -ltr

Note, you must [clear](http://omappedia.org/wiki/Installing_and_Using_Ftrace) the set\_ftrace\_pid file if you want to go back to generic function tracing after performing the above.

[tracing]# echo -1 > set\_ftrace\_pid

**The Tracers in detail**

All the following tracers are plugins of Ftrace which bring different presentation of trace, and trace different things (Latency, scheduler events)

**nop tracer**

This is the tracer by default which trace no function. But it can still be usefull to trace events (see [#Trace event](http://omappedia.org/wiki/Installing_and_Using_Ftrace#Trace_event))

**Function tracer**

This tracer is activated by enabling CONFIG\_FUNCTION\_TRACER in the kernel configuration. To work, it needs the kernel variable ftrace\_enabled to be turned on, otherwise this tracer is a nop.

# sysctl kernel.ftrace\_enabled=1

# echo function > current\_tracer

# cat current\_tracer

function

[tracing]# cat trace | head -10

# tracer: function

#

# TASK-PID CPU# TIMESTAMP FUNCTION

# | | | | |

bash-16939 [000] 6075.461561: mutex\_unlock <-tracing\_set\_tracer

<idle>-0 [001] 6075.461561: \_spin\_unlock\_irqrestore <-hrtimer\_get\_next\_event

<idle>-0 [001] 6075.461562: rcu\_needs\_cpu <-tick\_nohz\_stop\_sched\_tick

bash-16939 [000] 6075.461563: inotify\_inode\_queue\_event <-vfs\_write

<idle>-0 [001] 6075.461563: mwait\_idle <-cpu\_idle

bash-16939 [000] 6075.461563: \_\_fsnotify\_parent <-vfs\_write

The header explains the format of the output pretty well. The first two items are the traced task name and PID. The CPU that the trace was executed on is within the brackets. The timestamp is the time since boot in <secs>.<usecs>format, followed by the function name with its parent following the "<-" symbol.