tracepoints.txt
Using the Linux Kernel Tracepoints

Mathieu Desnoyers

This document introduces Linux Kernel Tracepoints and their use. It provides examples of how to insert tracepoints in the kernel and connect probe functions to them and provides some examples of probe functions.

* Purpose of tracepoints

A tracepoint placed in code provides a hook to call a function (probe) that you can provide at runtime. A tracepoint can be "on" (a probe is connected to it) or "off" (no probe is attached). When a tracepoint is "off" it has no effect, except for adding a tiny time penalty (checking a condition for a branch) and space penalty (adding a few bytes for the function call at the end of the instrumented function and adds a data structure in a separate section). When a tracepoint is "on", the function you provide is called each time the tracepoint is executed, in the execution context of the caller. When the function provided ends its execution, it returns to the caller (continuing from the tracepoint site).

You can put tracepoints at important locations in the code. They are lightweight hooks that can pass an arbitrary number of parameters, which prototypes are described in a tracepoint declaration placed in a header file.

They can be used for tracing and performance accounting.

* Usage

Two elements are required for tracepoints :

- A tracepoint definition, placed in a header file.
- The tracepoint statement, in C code.

In order to use tracepoints, you should include linux/tracepoint.h.

In include/trace/subsys.h :

```
#include <linux/tracepoint.h>

DECLARE_TRACE(subsys_eventname,
        TP_PROTO(int firstarg, struct task_struct *p),
        TP_ARGS(firstarg, p));
```

In subsys/file.c (where the tracing statement must be added) :

```
#include <trace/subsys.h>

DEFINE_TRACE(subsys_eventname);
```

```
void somefct(void)
{
        ...
        trace_subsys_eventname(arg, task);
        ...
}
```

Where :
- subsys_eventname is an identifier unique to your event
    - subsys is the name of your subsystem.
    - eventname is the name of the event to trace.

- TP_PROTO(int firstarg, struct task_struct *p) is the prototype of the function called by this tracepoint.

- TP_ARGS(firstarg, p) are the parameters names, same as found in the prototype.

Connecting a function (probe) to a tracepoint is done by providing a probe (function to call) for the specific tracepoint through register_trace_subsys_eventname(). Removing a probe is done through unregister_trace_subsys_eventname(); it will remove the probe.

tracepoint_synchronize_unregister() must be called before the end of the module exit function to make sure there is no caller left using the probe. This, and the fact that preemption is disabled around the probe call, make sure that probe removal and module unload are safe. See the "Probe example" section below for a sample probe module.

The tracepoint mechanism supports inserting multiple instances of the same tracepoint, but a single definition must be made of a given tracepoint name over all the kernel to make sure no type conflict will occur. Name mangling of the tracepoints is done using the prototypes to make sure typing is correct. Verification of probe type correctness is done at the registration site by the compiler. Tracepoints can be put in inline functions, inlined static functions, and unrolled loops as well as regular functions.

The naming scheme "subsys_event" is suggested here as a convention intended to limit collisions. Tracepoint names are global to the kernel: they are considered as being the same whether they are in the core kernel image or in modules.

If the tracepoint has to be used in kernel modules, an EXPORT_TRACEPOINT_SYMBOL_GPL() or EXPORT_TRACEPOINT_SYMBOL() can be used to export the defined tracepoints.

* Probe / tracepoint example

See the example provided in samples/tracepoints

Compile them with your kernel. They are built during 'make' (not 'make modules') when CONFIG_SAMPLE_TRACEPOINTS=m.

Run, as root :
modprobe tracepoint-sample (insmod order is not important)

```
modprobe tracepoint-probe-sample
cat /proc/tracepoint-sample (returns an expected error)
rmmod tracepoint-sample tracepoint-probe-sample
dmesg
```