Linux Tracing Embedded Systems

Marcel Ochsendorf, Muhammed Parlak

27.06.2022

Inhaltsverzeichnis

Einleitung

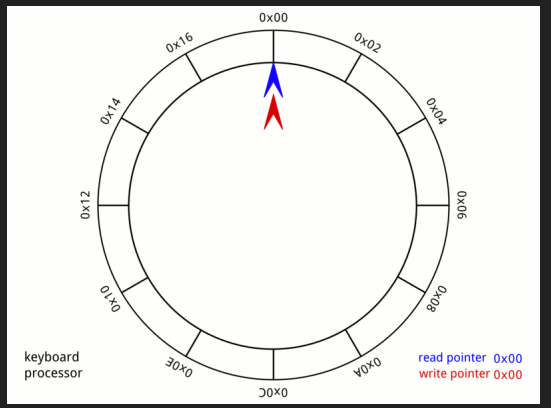
- Tracing?
- Tracing Systeme
- Linux Tracing
- Warum sollte man Tracen?

Grundlagen

- Ringbuffer
- Debug-Filesystem
- Tracing
  - Tracer
  - Events
- Abfangen von Events
- Probes
- Ressourcen

Grundlagen | Ringbuffer

- Basis für das Debug-FS
- Asynchrones Lesen und Schreiben
- Puffer mit Lesezeiger und Schreibzeiger
- Für jeden Core



Grundlagen | Debug-Filesystem

- Eingeführt mit Kernel 2.6.10-rc3
- Notwendig für die Tracing-Funktionalität
- Zugriff auf Diagnose und Debug-Informationen

```
1 # ENABLE DEBUG FS
2 $ sudo mount -t debugfs debugfs /sys/kernel/debug
3 # PRINT FOLDERS OF DEBUG FS
4 $ ls -lah /sys/kernel/debug | awk '{print $9}'
5 hid
6 usb
7 Tracing
8 [...]
```

Grundlagen | Debug-Filesystem

- Gezieltere Aggregierung von Events (z.B. Filterung)

```
1 # GET TRACERS
2 $ cat /sys/kernel/debug/tracing/available_tracers
3 hwlat blk mmiotrace function_graph wakeup_dl wakeup_rt wakeup
    function nop
4 # USE SPECIFIC TRACER
5 $ echo function_graph > /sys/kernel/debug/tracing/
    current_tracer
6 # DISABLE TRACER USAGE
7 $ echo nop > /sys/kernel/debug/tracing/current_tracer
```

Grundlagen | Debug-Filesystem

- Gezieltere Aggregierung von Events (z.B. Filterung)

```
1  # CALL STACK USING FUNCTION_GRAPH TRACER
2  $ echo function_graph > /sys/kernel/debug/tracing/
     current_tracer
3  $ cat /sys/kernel/debug/tracing/trace
4  # tracer: function_graph
5  # CPU-  DURATION            FUNCTION CALLS
6  # |     |   |               |   |   |
7   0)                 |       getname() {
8   0)    4.442 us     |          kmem_cache_alloc() {
9   0)    1.382 us     |             __might_sleep();
10  0)    2.478 us     |          }
11  [...]
```

Grundlagen | Tracing Events

- Auslösung sehr unterschiedlich
  - Beispiel Lesen und Schreiben von I2C Bus
  - Scheduler-Switching
  - Disk-Events (z.B. Dateioperationen)
  - IO-Events

- Nach Aktivierung
    - Informationen zu Event werden geloggt

Grundlagen | Tracing Events

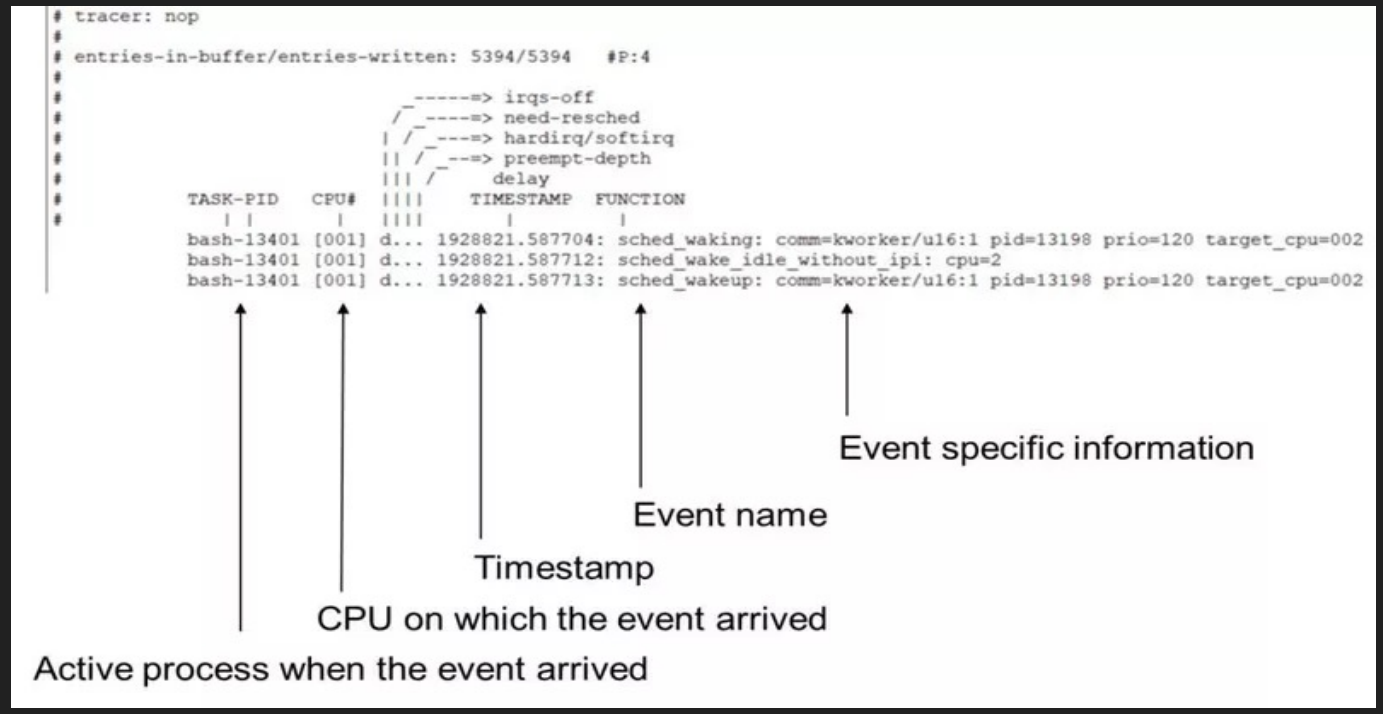
```
# GET AVAILABLE EVENT LIST
$ cd /sys/kernel/debug/tracing/events
$ ls -lah | awk '{print $9}'
alarmtimer
drm
exceptions
ext4 #READPAGE, WRITEPAGE, ERROR, FREE_BLOCKS
filelock
filemap
gpio #GPIO_DIRECTION, GPIO_VALUE
hda
i2c
irq
net
smbus #READ, WRITE, REPLY
sock #STATE_CHANGED, EXCEED_BUFFER_LIMIT, REC_QUEUE_FULL
spi
tcp
timer #TIMER_STOP, TIMER_INIT, TIMER_EXPIRED
[...]
```

```
$ cd /sys/kernel/debug/tracing/events/ext4
$ ls -lah | awk '{print $9}'
# EVENTS FOR EXT4
ext4_write_end
ext4_writepage
ext4_readpage
ext4_error
[...]
# INTERFACE FOR EVENT SETUP
enable
filter
format
```

Grundlagen| Abfangen von Events EVENT ACTIVATION

```
1  $ cd /sys/kernel/debug/tracing/events/ext4
2  # ENABLE ALL EVENTS FROM THIS GROUP
3  $ echo 1 > ./enable
4  # DISBALE ALL EVENTS
5  $ echo 0 > ./enable
6  # ENBABLE SPECIFIC EVENT
7  $ echo 1 > ./ext4_readpage/enable
8  $ echo 1 > ./ext4_writepage/enable
9  # GET EVENT LOG
10 $ cat /sys/kernel/debug/tracing/trace
```

Grundlagen| Abfangen von Events EVENT OUTPUT



Grundlagen Probes Kprobes

- Sammeln von Laufzeit und Performance-Daten des Kernels
- Vorteil:

1. Daten ohne Unterbrechung der Ausführung auf Prozessor-Instruktions-Ebene aggregieren

2. Registrierung dynamisch zur Laufzeit und ohne Änderungen des Programmcodes

Grundlagen | Probes User-Level-Probes

- Weiterentwicklung von kprobes
- Registrierung von Laufzeit Events in Applikation
- Adressoffset wegen eigenem virtuellen Adressraum

Grundlagen | Probes User-Level-Probes FUNCTION TRACING

```c
//test.c
#include <stdio.h>
int main(void)
{
    int i;
    for (i = 0; i < 5; i++)
        printf("Hello uprobe\n");
    return 0;
}
```

```
# BUILD APPLICATION
$ gcc ./test.c -o /tmp/test
# GET OFFSET
$ objdump -F -S -D /tmp/test | less | grep main
0000000000001149 <main> (File Offset: 0x1149):
# REGISTER uprobe_event
$ echo "p:my_uprobe /tmp/test:0x1149" > /sys/kernel/debug/
   tracing/uprobe_events
# ACTIVATE UPROBE EVENTS
$ echo 1 > /sys/kernel/tracing/events/uprobes/enable
# EXECUTE PROGRAM
$ /tmp/test
Hello uprobe
[...]
# PRINT TRACED EVENTS
$ cat /sys/kernel/debug/tracing/trace
# tracer: nop
# TASK-PID   CPU#   TIMESTAMP   FUNCTION
#  |    |      |         |          |
test-24842 [012] 258544.995456: printf: [...]
[...]
```

Grundlagen | Ressourcen

- Zusätzliche Ressourcen werden benötigt
- Unerhebliche Menge an Speicherplatz
- Bandbreite vom genutzten Medium
- Betrieb und Aufzeichnung wird beeinflusst

Was sollte getan werden?

- Aufzeichnung separat
- Nur Events die im Fokus stehen

Tools

- trace-cmd
- bpftrace
- Kernelshark

Tools trace-cmd

- Meistgenutztes Tool zum aufzeichnen von Tracelogs
- Aufzeichnung von Events spezifizierter Anwendungen

Tools | trace-cmd USAGE

```
# CHECK IF TRACING IS ENABLED
$ sudo mount | grep tracefs
none on /sys/kernel/tracing type tracefs (rw,relatime,seclabel
    )
## ONLY SCHEDULER EVENTS
$ echo sched_wakeup >> /sys/kernel/debug/tracing/set_event
## ALL EVENTS USING set_event
$ echo *:* > /sys/kernel/debug/tracing/set_event
# RECORD
$ trace-cmd record ./program_executable
# RECORD SPECIFIC EVENT
$ trace-cmd record -e sched ./program_executable
# USING A ADDITIONAL TRACER
$ trace-cmd -t function ./program_executable
```

Tools | trace-cmd OUTPUT

```
cpus=4
          sleep-19405 [000]   915.755341: sched_stat_runtime:   comm=trace-cmd pid=19405 runtime=168772 [ns] vruntime=124673167878 [ns]
          sleep-19405 [000]   915.755344: sched_switch:         trace-cmd:19405 [120] D ==> swapper/0:0 [120]
          <idle>-0    [003]   915.759205: sched_waking:         comm=rcu_sched pid=10 prio=120 target_cpu=003
          <idle>-0    [003]   915.759210: sched_wakeup:         rcu_sched:10 [120] success=1 CPU:003
          <idle>-0    [003]   915.759212: sched_waking:         comm=kworker/3:1 pid=2850 prio=120 target_cpu=003
          <idle>-0    [003]   915.759214: sched_wakeup:         kworker/3:1:2850 [120] success=1 CPU:003
          <idle>-0    [003]   915.759226: sched_switch:         swapper/3:0 [120] R ==> kworker/3:1:2850 [120]
    kworker/3:1-2850  [003]   915.759231: sched_stat_runtime:   comm=kworker/3:1 pid=2850 runtime=14167 [ns] vruntime=60565680765 [ns]
    kworker/3:1-2850  [003]   915.759233: sched_switch:         kworker/3:1:2850 [120] W ==> rcu_sched:10 [120]
      rcu_sched-10    [003]   915.759238: sched_stat_runtime:   comm=rcu_sched pid=10 runtime=7532 [ns] vruntime=60565679935 [ns]
      rcu_sched-10    [003]   915.759246: sched_switch:         rcu_sched:10 [120] W ==> swapper/3:0 [120]
          <idle>-0    [002]   915.762699: sched_waking:         comm=apt-get pid=2554 prio=120 target_cpu=002
          <idle>-0    [002]   915.762706: sched_wakeup:         apt-get:2554 [120] success=1 CPU:002
          <idle>-0    [002]   915.762709: sched_waking:         comm=kworker/2:3 pid=389 prio=120 target_cpu=002
```

Tools | bpftrace

- Kernelversion >4.x
- Eigene Skriptsprache
  - Aggregation
  - Eventfilter
  - Verarbeitung der Ergebnisse

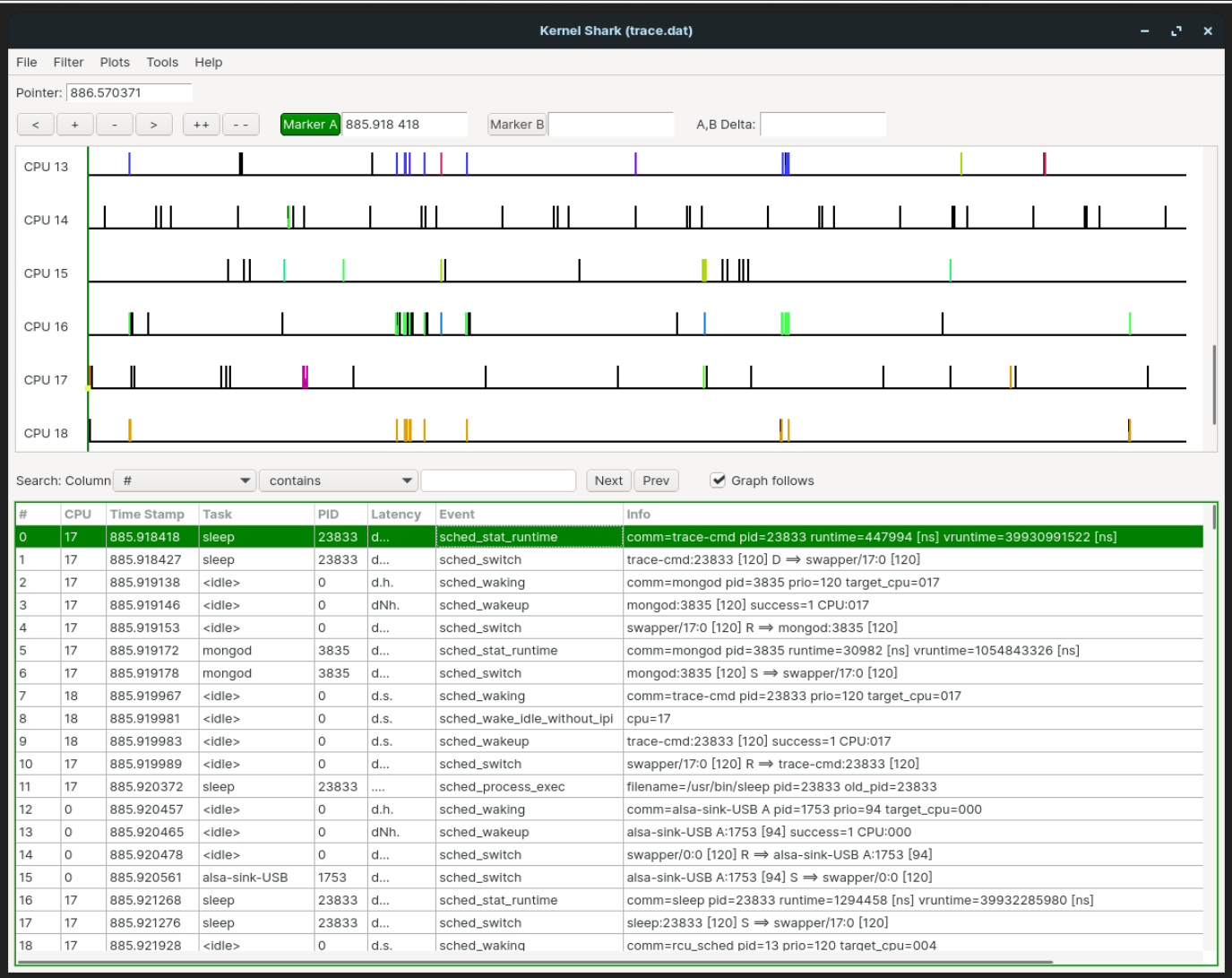
Tools | bpftrace SCRIPTING

```
51  kprobe:tcp_drop
52  {
53    $sk = ((struct sock *) arg0);
54    $inet_family = $sk->__sk_common.skc_family;
55
56    if ($inet_family == AF_INET || $inet_family == AF_INET6) {
57      if ($inet_family == AF_INET) {
58        $daddr = ntop($sk->__sk_common.skc_daddr);
59        $saddr = ntop($sk->__sk_common.skc_rcv_saddr);
60      } else {
61        $daddr = ntop($sk->__sk_common.skc_v6_daddr.in6_u.u6_addr8);
62        $saddr = ntop($sk->__sk_common.skc_v6_rcv_saddr.in6_u.u6_addr8);
63      }
64      $lport = $sk->__sk_common.skc_num;
65      $dport = $sk->__sk_common.skc_dport;
66
67      // Destination port is big endian, it must be flipped
68      $dport = bswap($dport);
69
70      $state = $sk->__sk_common.skc_state;
71      $statestr = @tcp_states[$state];
72
73      time("%H:%M:%S ");
74      printf("%-8d %-16s ", pid, comm);
75      printf("%39s:%-6d %39s:%-6d %-10s\n", $saddr, $lport, $daddr, $dport, $statestr);
76      printf("%s\n", kstack);
77    }
78  }
79
80  END
81  {
82    clear(@tcp_states);
83  }
```

Tools | Kernelshark

- Input von Trace-cmd
- Grafische Visualisierung

Tools | Kernelshark VISUALISATION



#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	17	885.918418	sleep	23833	d...	sched_stat_runtime	comm=trace-cmd pid=23833 runtime=447994 [ns] vruntime=39930991522 [ns]
1	17	885.918427	sleep	23833	d...	sched_switch	trace-cmd:23833 [120] D ⇒ swapper/17:0 [120]
2	17	885.919138	<idle>	0	d.h.	sched_waking	comm=mongod pid=3835 prio=120 target_cpu=017
3	17	885.919146	<idle>	0	dNh.	sched_wakeup	mongod:3835 [120] success=1 CPU:017
4	17	885.919153	<idle>	0	d...	sched_switch	swapper/17:0 [120] R ⇒ mongod:3835 [120]
5	17	885.919172	mongod	3835	d...	sched_stat_runtime	comm=mongod pid=3835 runtime=30982 [ns] vruntime=1054843326 [ns]
6	17	885.919178	mongod	3835	d...	sched_switch	mongod:3835 [120] S ⇒ swapper/17:0 [120]
7	18	885.919967	<idle>	0	d.s.	sched_waking	comm=trace-cmd pid=23833 prio=120 target_cpu=017
8	18	885.919981	<idle>	0	d.s.	sched_wake_idle_without_ipi	cpu=17
9	18	885.919983	<idle>	0	d.s.	sched_wakeup	trace-cmd:23833 [120] success=1 CPU:017
10	17	885.919989	<idle>	0	d...	sched_switch	swapper/17:0 [120] R ⇒ trace-cmd:23833 [120]
11	17	885.920372	sleep	23833	....	sched_process_exec	filename=/usr/bin/sleep pid=23833 old_pid=23833
12	0	885.920457	<idle>	0	d.h.	sched_waking	comm=alsa-sink-USB A pid=1753 prio=94 target_cpu=000
13	0	885.920465	<idle>	0	dNh.	sched_wakeup	alsa-sink-USB A:1753 [94] success=1 CPU:000
14	0	885.920478	<idle>	0	d...	sched_switch	swapper/0:0 [120] R ⇒ alsa-sink-USB A:1753 [94]
15	0	885.920561	alsa-sink-USB	1753	d...	sched_switch	alsa-sink-USB A:1753 [94] S ⇒ swapper/0:0 [120]
16	17	885.921268	sleep	23833	d...	sched_stat_runtime	comm=sleep pid=23833 runtime=1294458 [ns] vruntime=39932285980 [ns]
17	17	885.921276	sleep	23833	d...	sched_switch	sleep:23833 [120] S ⇒ swapper/17:0 [120]
18	17	885.921928	<idle>	0	d.s.	sched_waking	comm=rcu_sched pid=13 prio=120 target_cpu=004

Reverse Engineering DEMO

LIVE DEMO