Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Linux Kernel Profiling

Pablo Correa Gómez

Universität Hamburg
Scientific Computing // Wissenschaftliches Rechnen

Efficient Programming Seminar

Wednesday, January 27, 2021

Outline

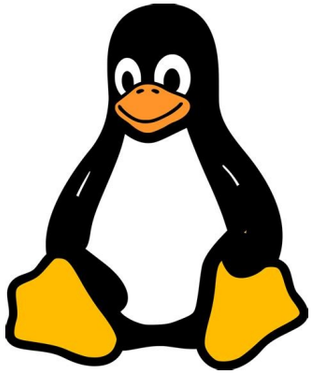
1. Motivation
2. Kernel profiling: What is the matter?
3. Kernel core profiling
4. Conclusion

Outline

1. Motivation
2. Kernel profiling: What is the matter?
3. Kernel core profiling
4. Conclusion

Motivation

- Profiling: Measuring performance/resource usage of a program
- Big-picture (most common routines) and small-picture (optimize)
- Related to debugging, e.g: Program spends a lot of time in a piece of code
- The kernel is always special (and a critical component)



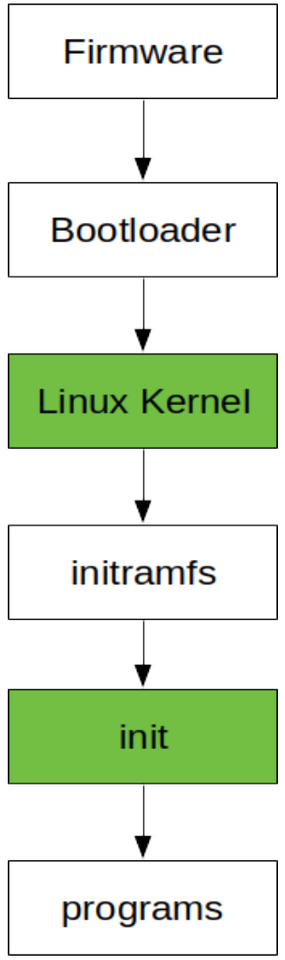
[https://tse2.mm.bing.net]

(Linux) Kernel in a Nutshell

- Kernel: The program that controls hardware
- Also memory management, processes or I/O
- The one program always running
- Everything else needs the kernel
- Linux: hybrid architecture: monolithic core + module
  - Core responsible for functional basics. Always running
  - Modules extend functionality. Can be loaded and unloaded

Kernel core profiling: What is the matter? Timing

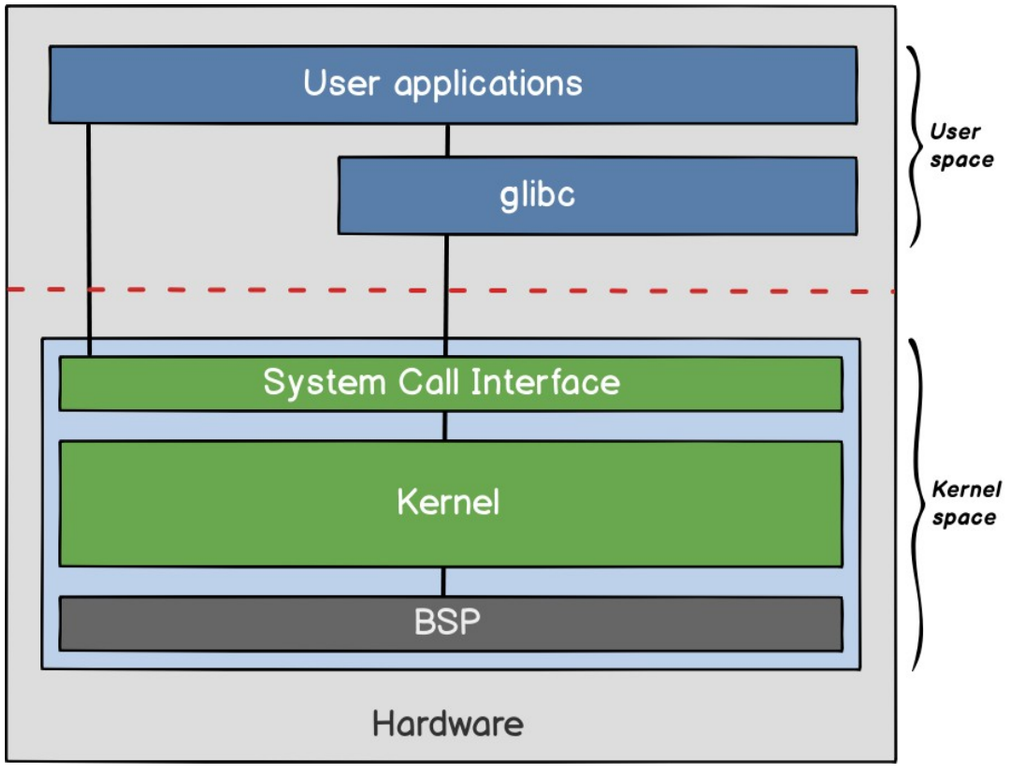
- Init is the “initial” program, that starts everything else (graphical session, network connectivity, etc.)
- Linux starts running before init
- Linux processes keeps running after init!
- Last thing that stops



[debian-handbook]

Kernel core profiling: What is the matter? Usage

- Kernel manages hardware resources → Used by **ALL** processes
- System call: Function call to request something to the kernel
- You might have used them without knowing, e.g: “open”
- Non-sense to run/profile kernel just by itself!



Linux User and Kernel space[2]

Kernel core profiling: What is the matter? Practicalities

- Common profilers of two types[3]:
  - Instrumentation → Require special compilation with symbols
  - Sampling → Periodically stop the program + record program counter
- Both types:
  - Generate data after program exits
  - Stop the process to record status

Kernel profiling: What is the matter?

- Kernel core profiling intrinsically different to application profiling:
  - First thing to start, last to exit
  - EVERYTHING uses the kernel
  - Cannot be stopped without halting the system
  - Only program that needs a reboot to be updated
- Kernel modules: Can be dynamically loaded and unloaded → Some properties from application profiling

- Kernel by itself does **NOTHING** useful. To profile, we need programs running!

Kernel core profiling

- Specialist tools
- Some built into the kernel
- Introduce *oprofile*
- Discuss further about *perf*

Kernel core profiling: oprofile

- External to linux
- Built as kernel module
- Samples registers and Program Counters via interrupts
- Uses CPU's Performance Monitoring Unit (PMU) when possible
- Collected via the Linux Kernel Performance Events Subsystem (*perf_events*)

```
CPU: Intel Haswell microarchitecture, speed 3500 MHz (estimated)
Counted CPU_CLK_UNHALTED events (Clock cycles when not halted) with
 a unit mask of 0x00 (No unit mask) count 100000
CPU_CLK_UNHALT...|
  samples|      %|
------------------
    61867 100.000 rhythmbox
        CPU_CLK_UNHALT...|
          samples|      %|
        ------------------
             9133 14.7623 libglib-2.0.so.0.5600.4
             7196 11.6314 libgtk-3.so.0.2200.30
             6453 10.4304 libgobject-2.0.so.0.5600.4
             5734  9.2683 libcairo.so.2.11510.0
             5146  8.3178 kallsyms
             4902  7.9234 libavcodec.so.57.107.100
             4845  7.8313 libharfbuzz.so.0.10702.0
             4268  6.8987 libc-2.27.so
             2846  4.6002 libpango-1.0.so.0.4000.14
             2001  3.2344 libgstreamer-1.0.so.0.1405.0
             1236  1.9978 libpython3.6m.so.1.0
             1038  1.6778 libgdk_pixbuf-2.0.so.0.3611.0
              787  1.2721 libgdk-3.so.0.2200.30
              725  1.1719 libpthread-2.27.so
              597  0.9650 librhythmbox-core.so.10.0.0
```

Kernel core profiling: perf

- Built into the kernel: tools/perf
- Also uses *perf_events*
- Provides information about “events”. Generic and HW specific
- Also useful for debugging/tracing

Kernel core profiling: perf

- Many kind of events
- Provide information from:
  - CPU's Performance Monitoring Unit (PMU): instructions, cycles, caching
  - Kernel software counters: page faults, cpu migrations
  - In-kernel Tracepoints
  - Other metrics
- Can have modifiers: u, k, H, ...
- "*perf list*" to inspect

```
pablo@pablo-SATELLITE-P50-B-10V:~$ sudo perf list hw sw

List of pre-defined events (to be used in -e):

  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  cache-misses                                       [Hardware event]
  cache-references                                   [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  ref-cycles                                         [Hardware event]

  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]
```

Kernel core profiling: Perf basics

- Very versatile tool:
  - System-wide and per-process events
  - Consider only specific cores
  - Collects and display events of execution (perf stat)
  - Store detailed profiling in file + inspect (perf record + perf report)
  - performance counter profile in real time (perf top)
- For HW events: multiplexing and scaling if not enough PMUs

Kernel core profiling: Advanced Perf

- Dynamically define tracepoints (perf probe)
- Provide benchmark suites (perf bench)
- Analyze shared cache (perf c2c)
- Analyze kvm host and guest (perf kvm)
- Other detailed kernel-internal statistics:
  - Memory (perf kmen and perf mem)
  - Scheduler (perf sched)
  - Locking (perf lock)
- And more!

Kernel core profiling: perf stat

- Provides statistics of events for the time running
- System-wide (-a) or a specific pid (-p) or command
- Is not sampling based! But real counts between start and end
- Can repeatedly run a command and provide statistics

```
pablo@pablo-SATELLITE-P50-B-10V:~$ sudo perf stat -a --big-num --scale
^C
 Performance counter stats for 'system wide':

    172847,901662      cpu-clock (msec)          #    8,000 CPUs utilized
           66.032      context-switches          #    0,382 K/sec
            1.668      cpu-migrations            #    0,010 K/sec
           10.752      page-faults               #    0,062 K/sec
   11.514.063.911      cycles                    #    0,067 GHz
    8.396.113.296      instructions              #    0,73  insn per cycle
    1.670.082.752      branches                  #    9,662 M/sec
       56.819.346      branch-misses             #    3,40% of all branches

     21,606841156 seconds time elapsed
```

Kernel core profiling: perf record + perf report

- *“perf record”*: run a command (or pid or system-wide) and sample events
- Two types of sampling:
  - “-F, --freq”: Frequency to sample counters. Default
  - “-c, --count”: Interrupt at *count* number of events
- Generate output: *perf.data*
- *“perf report”*: presents results
- Can create call graphs

```
Samples: 944  of event 'faults', Event count (approx.): 12370
  Children      Self  Command          Shared Object                Symbol
+   53,48%    53,48%  rhythmbox        libglib-2.0.so.0.5600.4      [.] g_time_zone_new
+   14,64%     0,00%  pool             [unknown]                    [.] 0xb890890000027887
+   14,64%     0,00%  pool             libjpeg.so.8.1.2             [.] 0x0000000000025bf0
+   14,64%     0,00%  pool             libjpeg.so.8.1.2             [.] 0x0000000000025ce5
+    9,43%     0,00%  rhythmbox        [unknown]                    [.] 0xb890890000027887
+    9,43%     0,00%  rhythmbox        libjpeg.so.8.1.2             [.] 0x0000000000025bf0
+    9,43%     0,00%  rhythmbox        libjpeg.so.8.1.2             [.] 0x0000000000025ce5
+    9,43%     0,00%  rhythmbox        [unknown]                    [.] 0x0000000300000000
+    8,88%     8,88%  rhythmbox        libc-2.27.so                 [.] __memmove_avx_unaligned_erm
```

```
Available samples
944 faults
16K cycles
```

```
Samples: 16K of event 'cycles', Event count (approx.): 8671510912
  Children      Self  Command          Shared Object                Symbol
+   30,34%     0,00%  rhythmbox        [unknown]                    [.] 0000000000000000
+    5,86%     0,00%  rhythmbox        [unknown]                    [.] 0x0000000000000001
-    4,15%     0,06%  rhythmbox        [kernel.kallsyms]            [k] entry_SYSCALL_64_after_hwframe
   - 4,09% entry_SYSCALL_64_after_hwframe
      - 4,02% do_syscall_64
         + 1,09% sys_poll
         + 0,73% sys_mmap
         + 0,62% sys_openat
-    4,06%     0,13%  rhythmbox        [kernel.kallsyms]            [k] do_syscall_64
   - 3,93% do_syscall_64
      - 1,09% sys_poll
           1,05% do_sys_poll
      - 0,73% sys_mmap
         - 0,72% sys_mmap_pgoff
            - 0,70% vm_mmap_pgoff
                 0,65% do_mmap
      - 0,62% sys_openat
           0,61% do_sys_open
+    3,67%     0,00%  rhythmbox        [unknown]                    [.] 0x0000000000000002
```

Kernel core profiling: perf top

- Realtime sampling of events
- Can select specific cores
- Useful for first look into what is happening
- Cannot specify command like previous examples

```
Samples: 16K of event 'branch', Event count (approx.): 20337588, DSO: [kernel]
Overhead  Symbol
   0,73%  [k] ktime_get
   0,63%  [k] schedule_hrtimeout_range_clock
   0,38%  [k] try_to_wake_up
   0,27%  [k] put_prev_task_fair
   0,27%  [k] copy_page_to_iter
   0,22%  [k] up_read
   0,22%  [k] find_get_entry
   0,22%  [k] __radix_tree_lookup
   0,21%  [k] rcu_all_qs
   0,21%  [k] fsnotify
   0,21%  [k] __update_load_avg_se.isra.38
   0,21%  [k] __wake_up_common_lock
   0,18%  [k] account_entity_dequeue
   0,18%  [k] check_preempt_curr
   0,18%  [k] decay_load
   0,18%  [k] sys_futex
   0,18%  [k] update_load_avg
```

Kernel core profiling: Conclusion

- Kernel profiling is special
- But specialist tools exist
- Require extensive knowledge about kernel internals
- perf is hard, but extremely powerful

References

- [1] https://infoslack.com/images/linux-arquitetura.png
- [2] Susan L. Graham Peter B. Kessler Marshall K. McKusick, gprof: a Call Graph Execution Profiler: https://docs.freebsd.org/44doc/psd/18.gprof/paper.pdf
- [3] Adeneo Embedded, Embedded Linux Conference Europe 2013, Linux Kernel Debugging And Profiling Tools
- [4] Elena Zannoni, Oracle America, Tracing on Linux Updates
- [5] oprofile documentation: https://oprofile.sourceforge.io/doc/
- [6] perf Tutorial: https://perf.wiki.kernel.org/index.php/Tutorial
- [7] perf: https://www.man7.org/linux/man-pages/man1/perf.1.html