P51a - KUtrace

Andrew W. Moore
With many slides adapted or stolen from Dick Sites Comp790 material
Disk / time / throughput measuring tool (left shift and drag, it will snap)
HTML: shift-click-drag for measurement triangle
Why are our lives so hard?

- Computers do not behave the way we want them to.
- Multiplexing is both wonderful and a pain
- Quality of service cross-talk
- We don’t have exclusive access to all the things we need
  - If we did – life would be VERY expensive.
- We don’t share well and sometimes....
Many Sources of Interference

- Most interference comes from software
- But a bit from the hardware underpinnings
- In a shared college stairwell, most interference comes from jerky neighbors
- But thin walls and bad kitchen venting maybe the hardware underpinnings
Kernel-user tracing
Recall: Measurement methods

- Trade intrusiveness vs. overhead vs. information vs. complexity

- Trace information in a program is useful.
  - “printf” as debugging tool?

- The kernel (operating system) is a program too....

- KUtrace specifically traces transitions across the KERNEL/USER barrier
What is KUtrace?

A complete trace of all transitions between kernel mode and user mode on all CPU cores of one server machine.

- Minimal events to assign 100% of the CPU time for all cores
- Not so many events that tracing quickly is impractical
- Can be postprocessed into complete timelines

**Tracing is the only tool that can capture unexpected interactions or interference between programs**
Kernel-User implementation
syscall, interrupt, trap, context switch

User code

Linux Kernel

trace mod

control on/off/ etc.

trace buffer in kernel RAM

post-proc.
kutrace_control program

$ ./kutrace_control
> go
> stop

ku_xxxx.trace written
Postprocessing
Postprocessing quick summary

export LC_ALL=C
cat ku_xxxx.trace | ./rawtoevent | sort -n | ./eventtospan "title" | sort > ku_xxxx.json
cat ku_xxxx.json | ./makeself show_cpu >ku_xxxx.html
google-chrome ku_xxxx.html
rawtoevent
rawtoevent

Convert raw binary trace file to readable text lines, one per event
- extend timestamps, convert to 10ns increments
  linear interpolation between trace start/end time pairs:
  <gettimeofday, time counter>
- propagate cpu, pid, rpcid
  so every event has each value
- put in names
  so every event has the syscall name, IRQ name, process name, etc.
### rawtoevent output sample (text)

<table>
<thead>
<tr>
<th>T</th>
<th>DUR</th>
<th>EVENT CPU PID RPC (10ns time)</th>
<th>ARG0 RETVAL IPC NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5940928129 0 1038 3</td>
<td>10870 0 0 0 13</td>
<td>page_fault (40e)</td>
<td></td>
</tr>
<tr>
<td>5940928563 0 1550 3</td>
<td>10870 0 0 0 10</td>
<td>/page_fault (60e)</td>
<td></td>
</tr>
<tr>
<td>5940928949 0 2050 3</td>
<td>10870 0 7520 0 13</td>
<td>open (802)</td>
<td></td>
</tr>
<tr>
<td>5940929482 0 2562 3</td>
<td>10870 0 0 3 13</td>
<td>/open (a02)</td>
<td></td>
</tr>
<tr>
<td>5940929517 0 2048 3</td>
<td>10870 0 3 0 0</td>
<td>read (800)</td>
<td></td>
</tr>
<tr>
<td>5940929761 0 2560 3</td>
<td>10870 0 0 832 2</td>
<td>/read (a00)</td>
<td></td>
</tr>
<tr>
<td>5940929967 59 2053 3</td>
<td>10870 0 3 0 3</td>
<td>fstat (805)</td>
<td></td>
</tr>
<tr>
<td>5940930094 0 2057 3</td>
<td>10870 0 0 0 0</td>
<td>mmap (809)</td>
<td></td>
</tr>
</tbody>
</table>

- `not merged, dur > 255`
- `not merged, ret > 127`
- `merged call/ret`
Ten event fields

**timestamps**  Start time for a span in seconds and fractions. For easy correlation with logs and other files, the seconds part be offset from an exact multiple of one minute in the gettimeofday() time domain, rather than starting at zero.

**duration**  Duration of a span in seconds and fractions.

**cpu**  CPU number as a small integer.

**pid**  Process ID (actually thread ID; the kernel calls this "pid"), low 16 bits.

**rpcid**  RPC ID for datacenter work; may be used for anything

**event**  Event number that starts the time span. Values 0-512 are names, markers, and other specials. 512-4096 kernel events system call, interrupt, fault. 64K+pid are user-mode.
Ten event fields

**arg0**    The low 16 bits of the first argument to a syscall, else 0.

**retval**  For a call/return span, the low 16 bits of the return value. Byte counts, etc. can be considered unsigned. Return codes can be considered signed.

**ipc**     Instructions per cycle, *quantized* via truncation into 4 bits.
            Values 0-7 are multiples of 0.125 IPC, i.e. less than one instruction per cycle.
            Values 8-11 are 1.0, 1.25, 1.5, and 1.75 IPC.
            Values 12-15 are 2.0, 2.5, 3.0, and 3.5+ IPC.

**name**    Name of the kernel routine or user PID (originally from the kernel 16-byte command field per pid). These names come from naming entries in the raw trace itself.
eventtospan
eventtospans

Convert event text file to timespans, formatted as proper JSON

- Start a timespan at each call transition, end it at each return transition
- Run a small stack of pending executions: Before call, push current execution name, after return pop back to previous. This is how we know what is running after each return.
- If a process migrates to a different CPU core, migrate its execution stack also.
- Add synthetic sine waves to indicate coming out of deep sleep
- Add some causality arcs from make-runnable to that process actually running
### eventtospan output sample (JSON)

<table>
<thead>
<tr>
<th>#</th>
<th>T</th>
<th>DUR</th>
<th>CPU</th>
<th>PID</th>
<th>RPC EVENT</th>
<th>ARG0</th>
<th>RET</th>
<th>IPC</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ 59.40928129, 0.00000434, 3, 10870, 0, 1038, 0, 0, 10, &quot;page_fault&quot;],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ 59.40928563, 0.00000386, 3, 10870, 0, 76406, 0, 0, 13, &quot;bash.10870&quot;],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ 59.40928949, 0.00000533, 3, 10870, 0, 2050, 7520, 3, 13, &quot;open&quot;],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ 59.40929482, 0.00000035, 3, 10870, 0, 76406, 0, 3, 0, &quot;bash.10870&quot;],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ 59.40929517, 0.00000244, 3, 10870, 0, 2048, 3, 832, 2, &quot;read&quot;],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ 59.40929761, 0.000000244, 3, 10870, 0, 76406, 0, 832, 3, &quot;bash.10870&quot;],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ 59.40929967, 0.00000059, 3, 10870, 0, 2053, 3, 0, 0, &quot;fstat&quot;],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ 59.40930026, 0.00000068, 3, 10870, 0, 76406, 0, 832, 0, &quot;bash.10870&quot;],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ 59.40930094, 0.00000300, 3, 10870, 0, 2057, 0, 12288, 9, &quot;mmap&quot;],</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transitions vs. timespans: 12 raw trace entries

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>delta,ret,arg0</th>
</tr>
</thead>
<tbody>
<tr>
<td>22c51</td>
<td>5fd 0000 0000</td>
<td>reschedule_ipi interrupt</td>
</tr>
<tr>
<td>22ca3</td>
<td>9ff 0000 0000</td>
<td>enter scheduler</td>
</tr>
<tr>
<td>22d19</td>
<td>200 0000 434d</td>
<td>context switch(17229)</td>
</tr>
<tr>
<td>22d77</td>
<td>bff 0000 0000</td>
<td>exit scheduler</td>
</tr>
<tr>
<td>22ea6</td>
<td>40e 0000 0000</td>
<td>fault page fault</td>
</tr>
<tr>
<td>2303c</td>
<td>60e 0000 0000</td>
<td>faultreturn page fault</td>
</tr>
<tr>
<td>230a1</td>
<td>20a 0003 ef1a</td>
<td>mark_a(&quot;read&quot;)</td>
</tr>
<tr>
<td>2334d</td>
<td>40e 0000 0000</td>
<td>fault page fault</td>
</tr>
<tr>
<td>2348a</td>
<td>60e 0000 0000</td>
<td>faultreturn page fault</td>
</tr>
<tr>
<td>23a48</td>
<td>88f 7b00 434d</td>
<td>syscall+ret sched_getparam</td>
</tr>
<tr>
<td>23593</td>
<td>891 1c00 434d</td>
<td>syscall+ret sched_getscheduler</td>
</tr>
<tr>
<td>2378e</td>
<td>40e f500 0000</td>
<td>fault+ret page fault</td>
</tr>
</tbody>
</table>
Transitions vs. timespans: 12 raw trace entries
JSON format
JSON; note leading spaces to sort properly

```json
{
  "Comment" : "V2 with IPC field",
  "axisLabelX" : "Time (sec)",
  "axisLabelY" : "CPU Number",
  "flags" : 128,
  "shortMulX" : 1,
  "shortUnitsX" : "s",
  "thousandsX" : 1000,
  "title" : "Hello World",
  "version" : 2,
  "events" : [
    [23.66655382, 0.00000218, 2, 1995, 0, 2055, 59216, 1, 0, "poll"],
    ...
    [999.0, 0.0, 0, 0, 0, 0, 0, 0, 0, ""]
  ]
}
```
Postprocessing

Raw trace file → raw to event → event to span → json file

"Title" → span to trim → span to span → make self → html file

show_cpu.html → d3.v4.min.js

browser
KUtrace display of timelines
Play with the html interface

- (on each pi)
- ~/KUtrace/hello_world_demo.json

and it’s

- ~/KUtrace/hello_world_demo.html

Make a copy to your own machine, display, and play.

Textbook Chapter 19 pp257-266 provide a manual for the GUI
The following slides provide insight into the design decisions.
Display issues

Problem:
   Possibly 1-10 million execution spans to display

Solutions:
   - spantospan reduces granularity, so fewer spans
   - spantotrim keeps only a subset time interval, so fewer spans
   - HTML initially shows entire trace at modest resolution, allows user to pan and zoom to reveal detail at various different places.
   - HTML *elides* spans that are narrower than 1 pixel, combining several until they total at least one pixel. The total is then displayed with a single-color line, not three or four different colors -- just idle or kernel or user.
Display issues

Problem:
Showing different processes or syscalls in different colors is too hard to distinguish -- only about 12-20 distinct colors on screen

Solutions:
• Use pairs of colors, PID mod 15 to pick one, PID mod 17 to pick the other, giving $15 \times 17 = 255$ combinations total. Display as two stacked vertical lines.
• For kernel-mode, add third lightened background color to distinguish syscall, interrupt, and fault code.
Display issues

Problem:
Display CPU timelines plus PID timelines plus RPC timelines, but now the vertical dimension is too crowded

Solutions:
- Group the spans: CPU, PID, RPC
- Make groups collapsible, and by default collapse PID and RPC.
- Allow Y-axis to pan and zoom
- Suppress lines whose non-idle content is off-screen
- Allow shift-click on Y-axis labels to highlight one or more only
- Allow shift-click on group labels to collapse un-highlighted rows
- Show vertically narrow spans as single-color lines, to speed up display
Display issues

Problem:
  Color-blind users

Solutions:
  • Try limited gray-scale combinations of three lines per span -- **fails**
  • Try color-Brewer color-blind safe colors [http://colorbrewer2.org](http://colorbrewer2.org) -- **fails** with not enough distinct colors (about 7-9).
  • Equal-luminance and thus low-contrast syscall/IRQ/fault background colors are particularly problematic. So add high-contrast one-pixel edges -- black, gray, white
  • Add CB button that simply toggles RGB channels to GBR
Display issues

Problem:
Too many digits in X-axis time labels

Solutions:
- Give overall base (start) time in HH:MM:SS and then just seconds from there on the X-axis.
- When zoomed in, move millisecond digits to base time and just give milliseconds from there on the X-axis.
- When zoomed in further, move millisecond and microsecond digits to base time and just give microseconds from there on the X-axis.
- Change axis **units** to match
More display issues

easy to get lost -- how to get back? -- red button

hard to get overview -- user button to label all processes, just first instance, short names

hard to click on one span at a time to see detail -- all button to label all spans on screen

want to know details about one span -- long label has start time, full name, syscall(arg)=ret, process id, duration, IPC.

Labels overlap -- stack them vertically in ~3 tracks, use semi-opaque white background

labelling all takes forever to display when 1M spans -- at most one label start per pixel location

same with search results
More display issues

easy to lose place after pan/zoom -- retain nearest label and redisplay it after pan/zoom
want grid, but not intrusive -- almost-white vertical grid lines
straight-line wakeup arrows blend into span lines, not visible if overlapping -- curve them
want to search for unusually long or short spans -- usec min/max range
can't find a particular rarely-executed process -- use search by name or pid number
hard to distinguish IPC triangle angles -- color-code groups of four, notch alternate shapes
More display issues

- want to see total time for particular spans -- search result gives count and total time
- want labels to show which row they label -- put just underneath that row
- want to line up labelled items across different rows -- extend solid line below label row, dotted line above
- want to determine time between events -- click-drag snaps to event row and start time, giving elapsed time
More display issues

clutter from too many identical PID labels -- only label user-mode once at first time from left edge

clutter from too many marks -- suppress if too close together, button to suppress them all

clutter from IPC triangles -- suppress if too close together or X width is too small, button to suppress them all, off by default

too much clutter with wakeup arcs -- suppress short ones if X width is too small, button to suppress them all
More display issues

in a presentation, want to quickly show specific subsets of a trace -- 1..5 buttons to save/restore pan, zoom, and buttons

fonts too small for presentations -- click title to change font sizes

explain UI without taking much space -- single Usage() line at bottom, with [more} to expand to a screenful of detail