KUtrace User Guide

dick sites 2022.07.21

The underlying postprocessing software for KUtrace posted at

```
https://github.com/dicksites/KUtrace
```

is slightly newer than the book text of *Understanding Software Dynamics*. This User Guide gives an updated description of the software as of July 2022, including FreeBSD support and some improvements to the dynamic HTML Search box.

Loadable Module, `kutrace_mod`, for FreeBSD

The loadable module that implements the bulk of KUtrace has two possible environment variables. Use `$ sudo kenv kutrace_xxx=N` to change them before loading the module.

kutrace_mb=20

specifies the number of MB of kernel memory to reserve for the trace buffer. The default is 20MB.

kutrace_nocheck=0

specifies whether to check for the caller of kutrace_control having the proper privilege. The default is 0. Setting this to 1 or any other nonzero value disables the checking, which might be appropriate for a standalone machine used for a class. The required privilege is PRIV_KMEM_READ.

Packet filtering, as described below for Linux, is not yet implemented for FreeBSD.

The command

```
$ sudo kldload kutrace_mod.ko
```

will load and initialize the loadable module, and `kldunload` removes it.

Loadable Module, `kutrace_mod`, for Linux

The loadable module that implements the bulk of KUtrace has three possible command-line parameters:

tracemb=20

specifies the number of MB of kernel memory to reserve for the trace buffer. The default is 2MB.

The module implements an optional network packet-tracing facility. Packets are filtered in a simple way, with those passing the filter adding one KUtrace entry each at the time that the kernel TCP or UDP code *transfers the packet to/from a NIC*. The filtering is done via a hash of the first 24 payload bytes of a packet, i.e. those immediately after the packet headers. These are the data bytes that are visible to user-mode software sending network messages, so if user code records similar entries it becomes possible to see the time delay between user code and NIC on both ends of a transmission.

pktmask=0x0000000f

specifies which bytes of a network packet to hash for a possible tracing match. The low 24 bits correspond to the first 24 payload bytes and the high 8 bits are ignored. A one-bit uses that byte while a zero-bit sets it to zero. The hash is a four-byte-wide simple XOR. In this example, the first four data bytes are used and the other 20 zeroed.

pktmatch=0xd1c517e5

specifies the network packet hash value to match; matched packets get KUtrace entries. In this example, a packet matches if its first four bytes are exactly 0xd1c517e5. This idea is to be able to pick off packets that are the beginnings of possibly multi-packet messages, based on some known bit pattern (signature) at the front of a message. The default parameters here match the RPC library used in the book.

The command

```
$ sudo insmod kutrace_mod.ko tracemb=2 pktmask=0x0000000f pktmatch=0xd1c517e5
```

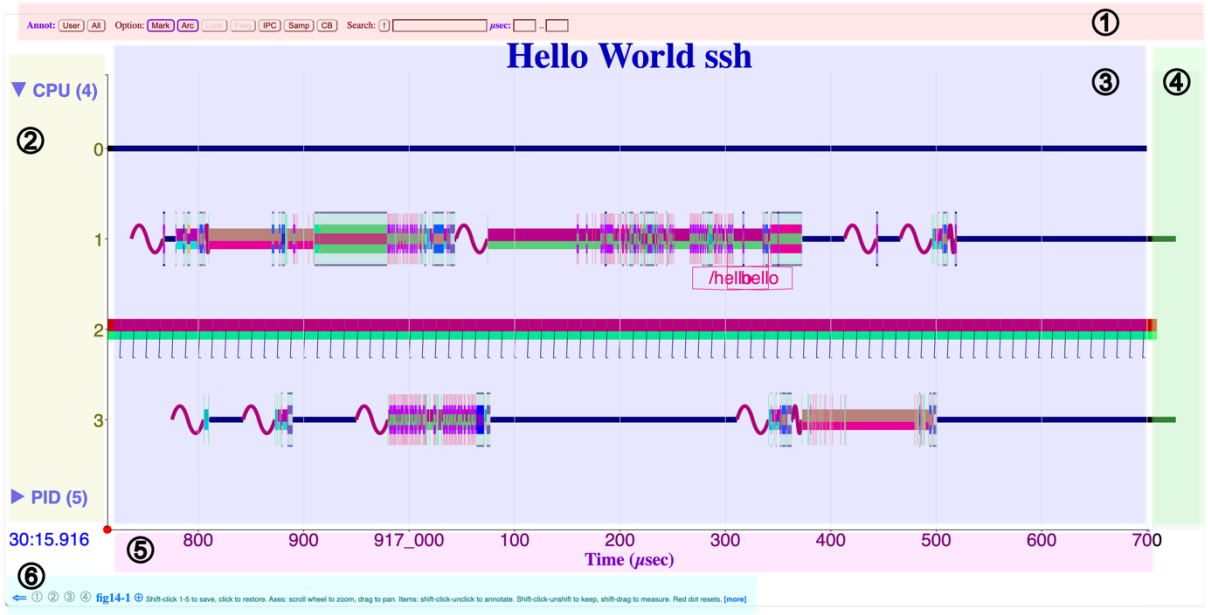
will initialize the loadable module packet filter with these parameters, which are also the defaults.

Because every single inbound and outbound packet has the filtering calculation done when tracing is on, the calculation is kept very simple and fast, about 10 nsec per packet on a 3.9 GHz Intel i3 chip. For messages that are typically many packets, matching on just the first one is usually sufficient to understand networking delays. However, a mask of zero and a match of zero will trace *all* packets, and a mask of zero and match of non-zero will trace *no* packets.

Changing these parameters requires removing (`rmmod`) and then re-inserting the loadable module.

The Dynamic HTML Display

As described in Chapter 19, the dynamic HTML display has six regions, shown here in a copy of Figure 19.1. Regions 1 and 6 contain display and action controls. The main timeline display is Region 3, with a possible legend in Region 4. By default, the timeline region shows one row per CPU in the CPU group. The PID group and optional RPC group are collapsed by default, as shown in the Y-axis, Region 2. Region 5 is the X-axis, showing the time scale.



Region 1, Controls

The Controls region has a number of buttons and text boxes to control what is displayed in the main timeline region 3. By default, the timeline region shows execution timespans (termed *spans* hereafter) for the CPU group and also shows non-execution (waiting) for the Process and RPC groups. Several optional overlays can accompany the main timelines, specified by the `Annot:` and `Option:` controls. The `Search:` controls specify text annotations for spans with matching names. When a KUtrace HTML file is first loaded, the far right of the Controls area shows the build version and date of the Linux kernel software that created the trace, the processor model, and the size in pixels of the browser window. The size disappears after 5 seconds -- its purpose is to allow you to create consistent displays for screen shots. The kernel version and processor model disappear as soon as you pan or zoom.

In addition to the obvious buttons, all the bright blue text items in the Controls and other regions are also control buttons.

The `Annot:` group controls adding short text annotations for some spans. By default, the text for these has accompanying vertical lines, solid below a span and dotted above. This allows you to quickly see which row matched and see the time alignment of spans on different rows. An *alternate style* of annotation just has the name and a short line below a span. Clicking the blue **`Annot:`** button toggles to lower-case `annot:` and toggles the annotation style. Span names longer than 9 characters are elided, using just the first 7 characters, tilde, and last character. System calls have the low 16 bits of the first argument in parentheses followed by an equal sign and 16 bits of the return value.

The **`User`** button annotates the first occurrence of each user-mode process ID that is on-screen in Region 3. This can give you a quick view of what programs and threads are in the displayed portion of a trace.

The **`All`** button annotates every span on the screen, again using short text. It is most useful when zoomed in.

The `Option:` group of controls toggle various timeline overlays.

The **`Mark`** button cycles though four states: showing all manually-inserted mark_a/b/c/d trace entries, showing just the text mark_a/b/c ones, showing just the numeric mark_d ones, and showing none. The button is grayed out if there are no marks in the trace.

The **`Arc`** button cycle through three states: showing wakeup arcs between threads, showing them bolder, and not showing them. To avoid clutter, short wakeups are only visible when you zoom in to just a few milliseconds across the display.

The **`Lock`** button cycles through three states: showing software locks being acquired or held, showing them with the lock-held lines bolder, and not showing them. Locks are shown above associated rows in the PID and RPC groups; locks are not associated with CPUs, so are not shown in that group. To avoid clutter, locks are only visible when there is enough room vertically and they occupy at least a few horizontal pixels across the display. The button is grayed out if there are no software lock events in the trace.

The **Freq** button cycles through three states: showing a CPU clock-frequency overlay, showing it bolder, or not showing it. The overlay shows very light green for clock frequencies that are in the top 12% of the frequency range recorded in the trace, dark red for the bottom 12%, and successively darker yellows for the middle 75%. Because a slow CPU clock can sometimes completely explain slow performance or is otherwise correlated with execution anomalies, this display is on by default. On some machines (x86), the individual frequency of each CPU core is sampled at timer interrupts, while on others (Raspberry Pi), the single frequency for all cores is recorded just when it changes. To avoid clutter, frequency spans are only visible when they occupy a few pixels across the display. The button is grayed out if there are no frequency events in the trace or if they all specify the same frequency.

The **IPC** button cycles through four states: showing instructions-per-cycle speedometer triangles for all kernel- and user-mode execution spans, showing them just for kernel-mode, just for user-mode, and showing none. IPC triangles are shown on top of each execution span in all three CPU, PID and RPC groups. To avoid clutter, IPC triangles are only visible when they occupy at least a few horizontal pixels across the display. The redundant IPC 0 is not shown for idle spans. When IPC triangles are selected, the IPC legend is shown in Region 4.

The **Samp** button toggles on and off the showing of PC samples. Samples are shown as slightly-rising lines above associated rows in all three CPU, PID and RPC groups. To avoid clutter, samples are only visible when there is enough room vertically and they occupy at least a few horizontal pixels across the display. Also to avoid clutter, PC samples of the idle loop are suppressed. PC samples are taken at every timer interrupt. Kernel-mode samples are shown as thick dashed lines and user-mode as thinner dashed lines. The little "v" mark at the right end of a sample line shows the time it was taken. The slight slope of the overlay line is suggestive of the decreasing probability that the sampled code was being executed earlier. As explained in the book, the `samptoname_k` and `samptoname_u` postprocessing programs can be used to turn hex PC addresses into routine names in kernel- and user-code respectively.

The **CB** button toggles a change in timeline span colors, as a small aid to color-blind users. It simply rotates the red/green/blue values of each color. This can sometimes be helpful is distinguishing the spans.

The `Search:` group of controls select on-screen spans (those starting at or after the tic mark at the left end of the x-axis and before the tic mark at the right end) to annotate. The **text box** accepts a string or JavaScript regular expression to match, and the **[!]** button negates the match, much like "-v" for grep. The other two text boxes specify **minimum** and **maximum** durations for matching spans. By default, these are microseconds, but the blue **µsec** button can be clicked to rotate among nsec, µsec, and msec values.

Each search applies short-text annotations to each matching span and then displays on the far upper right the **match count**, the total duration of those matches, and the minimum and maximum matching span durations. To avoid clutter, only one annotation is shown per horizontal pixel across the display. All of the matches are counted, though.

The search facility is quite powerful. Searching for the name of a process shows all the on-screen places where it is running. Searching for the name of a system call shows all of those. Searching for "ipi" shows all interprocessor interrupts: sending one and also the resulting reschedule_ipi interrupt. Searching for "futex" shows system calls that block on software locks and the points at which they resume. Searching for "wait_cpu" shows all the waiting-for-CPU spans in PID and RPC groups. Searching for "mwait" (x86) or "wfi" (Raspberry Pi) finds all the events that trigger low-power idle. Searching for "=1" can find system calls that return one. Searching for "PC=00556f" finds PC samples with high bits in that address range. Note the capitalization.

Searching for "rx|tx" (no spaces) shows all the network receive or transmit events on-screen, if packet tracing was specified. Searching for "write\(3" with the default `Annot:` style (not `annot:`) finds all the writes to stderr. Note the use of backslash escapes and the dependency on the annotation style to get parentheses at all.

Point events in the trace (as opposed to spans that always have non-zero duration) have an artificial duration of 10 nsec. You can find these by searching for empty search text and a maximum duration of 10 nsec.

A few synthetic-event names are spelled with a leading hyphen: "-c-exit-", "-freq-", "-idle-", "-idlelp-", "-sched-", and "-wakeup-". To avoid clutter, these only match in searches when the leading hyphen is part of the search string.

Note also that search strings are case-sensitive.

To the right of the search group, the operating system version and CPU model name are shown at startup if these items are recorded in a trace.

The July 2022 update adds the ability to search for a time in seconds and the display will change to be centered on that time. To avoid unintended triggering, the time must have at least one fraction digit. The width of the resulting displayed timespan is a function of the number of fraction digits. It can be adjusted by appending one or more plus or minus signs on the right. For example, searching for `50.085+` in the supplied `hello_world_trace.html` will center the screen on the `exit_group` call.

As a convenience, searching for `@@` will calculate a time search string that encompasses the current display, zooming out a little to quantize the width.

Whenever a <CR> is typed in the search box, the search string is put on the front of a list of recent searches. Subsequent search typing will show the most recent strings that contain the typed characters. Click on one of those to search for it.

Searching for something that does not exist, such as `zzzz`, and then clicking the `[!]` button matches everything currently on screen. The resulting match count and duration information can be useful for summarizing the visible activity.

Region 2, Y-axis
The y-axis has up to three groups of rows and a label for each row. Each group header line has an expand/contract triangle, the group name, and the number of rows in that group.

The CPU row labels are just integers, sorted numerically.

The PID group labels are process-name dot PID number, sorted by PID number. The idle process, PID number zero, is not shown. Some process names have plus signs, showing that the name changed during execution, for example `bash+time_getpid.11046` indicates that PID 11046 was first named `bash` but then changed (via `execve` for example) to `time_getpid`. Linux PID numbers less than 1024 are usually kernel threads. In reality, the PID numbers are TID numbers -- Linux thread numbers or FreeBSD thread numbers with the leading 100000 removed. The name comes from the Linux kernel's misnamed "pid" variables.

The RPC row labels have method-name dot RPCID number, sorted by start time. Thus, reading top to bottom, RPCs are shown in order of arrival.

When there are too many rows to show all the names readably, some names are omitted, but the tic mark for each row is always shown, so you can tell if any names are missing.

Clicking the leading triangle in a group name initially toggles between two states: expanded and collapsed. The active click area for this action is the leftmost 1/4 of Region 2. Mouse **click-drag** and **wheel turns** in the rightmost 3/4 allow vertical pan and zoom. Clicking and dragging vertically pans rows, and using the mouse wheel zooms the rows.

Shift-click on a row name highlights the row by making the text bold, and highlights all the spans in that row by graying out all other spans. These span highlights carry across the CPU/PID/RPC groups, but do not cause rows in other groups to be highlighted. You can manually highlight these other-group rows if desired. Shift-clicking multiple rows ORs all the highlighted spans. When any row in a group is highlighted, clicking the leading triangle in a group name will rotate among three states instead of two: expanded, expanded only for highlighted rows, and collapsed.

As a convenient shortcut, **shift-right-click** on a row name toggles all the rows with the same prefix, up to the first non-letter/digit/underscore/hyphen. This makes it easy to highlight multiple related threads or multiple related RPCs.

Only highlighted items are considered on-screen for User, All, and Search annotations. This can be useful for careful filtering of matches by specific CPU, PID, or RPC.

Region 3, Timelines
There is a lot going on with the timelines. The initial display has all the kernel- user- and idle-mode execution spans on all CPUs for the entire trace start-to-finish time. Clicking the **red dot** where the axes meet reverts to this view at any time.

To avoid clutter and to substantially speed up the display of large traces, only one span is shown per horizontal pixel across the display and only one row is shown per vertical pixel. In the horizontal direction, the JavaScript internally accumulates spans that are less than one pixel wide until the deferred time reaches one pixel wide. Then a representative 1-pixel-wide single-color span is drawn. The deferred time separately counts kernel, user, and idle time; the representative span's height and color is based on the largest of these. For many traces, this design can give a quite choppy initial display, possibly showing mostly timer interrupts. As you zoom in, more and more detail will be revealed.

In the vertical direction, a similar mechanism reduces very skinny rows to single colors with overlays dropped.

If you zoom in far enough, the approximate KUtrace overhead for each event is shown as **diagonal white lines** at the leading edge of a span. This is to remind you that there *is* overhead and to let you see if it might be enough to distort observations. The approximate values are determined by running the `time_getpid` program with no tracing, with KUtrace go, and with KUtrace goipc (which unfortunately has higher overhead because of the terribly slow read of the instructions-retired performance counter). For the Intel i3 at 3.9 GHz, the measured overhead is about 20 nsec for go, and about 50 nsec for goipc. For the Raspberry Pi at 1.5 GHz, the overhead is about 100 nsec for go, and about 150 nsec for goipc.

The **User** button can be initially helpful to see all the process names and then pick time areas that you care about.

Within the timeline region, mouse **click-drag** pans and **mouse wheel** zooms in and out. **Shift-click-drag** annotates the start and end points and shows the elapsed time difference at the bottom of the screen. **Shift-click** on any span annotates it. If there is enough vertical room, annotation text rotates through several lines, reducing overwriting of names. As a convenience, **shift-right-click** on a span annotates it and highlights all matching spans by graying out everything else. This also shows the match count and duration information across all the matches.

A shift-click annotation in the default style gives the long-form version: starting time of a span relative to the base time at the far left of Region 5 (avoiding excess-digit clutter), the name of the span, its duration, and if available its IPC. For system calls, the low 16 bits of the first argument and the low 16 bits of the return value are shown as `foo(1234)=12`. Alternate-style annotations give just the name.

If shift is still down when you unclick, it clears annotations. Unshifting first leaves annotations showing, allowing you to annotate many spans. If you pan or zoom horizontally, the single annotation nearest the mouse x-position is kept and converted to long form. This can help keep you located near something of interest while you pan and zoom.

If the Lock or PC sample overlay is visible, shift-clicking just *above* a timeline will select the overlay span instead of the CPU/wait span.

If packet tracing was specified for the loadable module, then the KUtrace packet entries will show as **little diagonal lines above CPU 0**, downward-slanting for incoming rx packets, and upward-slanting for outgoing tx packets. Their activity can then be correlated with the sending or receiving process activity. A search for "tx" or "rx" will annotate these entries, showing in uppercase hex 16 bits of the unmasked data hash over the first 32 data bytes of a packet. These hash values can help correlate each traced packet across client-server connections between two machines, and across kernel-user interfaces within one machine.

Region 4, IPC Legend
This is just a simple legend of the IPC triangles, visible when the IPC overlay is active. Otherwise, the space allows annotation names near the right edge to hang over a little.

Region 5, X-axis
The x-axis shows time. To avoid clutter, most of the time is shown to the left of the x-axis: day, hours, minutes, seconds. As you zoom in, the x-axis units change from seconds to milliseconds to microseconds to nanoseconds and the base time adds high-order seconds, milliseconds and then microseconds digits. When the low-order digits along the x-axis wrap around, the next second, millisecond, etc. is shown followed by an underscore in the corresponding label.

Region 6, Save/Restore
It is useful, when you find an anomaly in a view, to record the time and control settings such that you can get back to that view. At the lower left there are four **circled-digit** buttons for this purpose. If you shift-click on one of these, the current display state is saved and the button turns from gray to blue. If you later click (no shift) on a blue button, that view will be restored. The **back arrow** will go back exactly one view when clicked.

You can also add new buttons by clicking the **circle-plus** sign. A popup offers you a name for the new button and clicking OK creates the button and stores the current view state there.

Unfortunately, there is no simple way to save this information across HTML file opens, because many security mechanisms prevent updating an HTML file on disk from within a browser. Instead, I have implemented two kludges.

First, whenever the browser closes a KUtrace HTML file, the current view state is saved as a text string in `localStorage` (a limited 2MB of browser file space holding <key, value> pairs). Each KUtrace HTML file has a 32-bit random ID, and that is used as the key. Whenever a KUtrace HTML file is opened, the `localStorage` is checked for the matching key. If found, the view state is loaded but not immediately used. Instead, the back-arrow button is turned blue to indicate that a previous state is available. Clicking it uses the saved state. Not using the state automatically makes the mechanism robust against corrupted or redesigned state. This facility gives a one-view local-machine state saving.

Second, the circle-1 through circle-4 states and any added buttons and callouts can be *exported* in text form to the clipboard and then hand-edited into an underlying JSON file and thence via `makeself` turned into a new HTML file that carries along the saved states. This is further explained in the next section. I hope someone can do better.

At the right side of Region 6 is a blue **`[more]`** button. Click it to expand the sketchy directions on that line to somewhat more text. Click again (it will read `[less]`) to shrink.

Secondary Controls
Shift-click on the **red dot** to toggle on some secondary controls. These are mainly for tweaking the display for the purpose of making presentations (or book diagrams). There are five text boxes for numbers and four buttons.

The Aspect box accepts a viewing aspect ratio of n:m, where each is a single digit. For example, "4:3" makes a view that is the same aspect ratio as old TV sets, and "9:4" closely matches the 16:9 aspect ratio of new TV sets, while "0:0" turns off the restriction. When the aspect ratio is active, the height of the display area is restricted to be a multiple of 100 pixels and the width is determined by the aspect ratio. The upper right of Region 1 will show the display area size in pixels for five seconds after you change the browser window size.

The Ychars and Ypx boxes specify the approximate number of characters reserved for the y-axis labels and their font size in pixels. The Ypx value is also used for the font size of the x-axis labels.

The txt and spn boxes specify the use of vertical space for each row of the timeline display. The txt box specifies the number of lines of text to be used for annotations, and the spn box specifies the number of same-size lines to be used as the space for drawing timelines. So for each timeline row, txt / (txt+spn) of the height is used for annotations and spn / (txt+spn) is used for the kernel/user/idle drawing. For example, 2 and 4 specifies 2/6 of the row height for 2 lines of annotation text and 4/6 for timelines, while 3 and 6 specifies the same proportional split of vertical space for text and timelines but with three lines of annotation text. The number of lines of text and the y-axis zoom determine the font size of the annotations.

The Legend button cycles through three states: showing the normal timeline display, showing the KUtrace HTML Legend from the back of the book in landscape orientation and showing it in portrait orientation.

The Fade button grays out all the main timeline diagram elements, leaving just the overlay items. This can be occasionally useful for emphasizing some overlay data.

The Speech-bubble button allows you to insert a speech-bubble callout in the diagram. A popup asks for the callout's text. Clicking OK displays the callout. You can then drag the spike part to the span you want to label and then drag the bubble part to its desired position. There is an underlying grid for the bubble positions, to make it easier to align several related bubbles. If exactly one span is annotated when you start, the callout will be positioned at that span.

The Export button copies the set of view buttons (described above in the Save/Restore section) and the callout bubbles to the clipboard, as JSON text. The last line of the JSON for a full trace is "]}". You can replace this line with the clipboard text, beginning with "]," and ending with "]}". This adds two more arrays to the JSON, the "extra" array giving callouts and the "savedview" array giving view buttons,

including any you have added. The resulting JSON can then go through `makeself` to create an augmented HTML file.

At the right of these buttons, the **x:** value gives the time offset and width of the x-axis, both in microseconds, and the **y:** value gives the vertical offset and height of the y-axis in units of "tracks" where there are 20 tracks per timeline row. These numbers can be used to align and size screen shots across multiple views or multiple HTML files. Finally, the version date of the show_cpu.html template is shown at the far right (for July 2022 templates and later).

Added Program: the `spantoprof` program

The `spantoprof` program is a filter that reads a JSON file from stdin and writes a new *profile* JSON to stdout. The default profile (also selected with the `-row` command-line flag) combines all the same-name items in each row into one item and then sorts those by accumulated time and by type. Items that contribute useful work are sorted to the left side in decreasing order of total time, and those that indicate delays are sorted to the right side, also in decreasing order of accumulated time. Within each CPU/PID/RPC group, there are the same number of rows in the profile output as in the original input.

These *per-row* profiles thus show the aggregated kernel/user/idle time for each CPU or PID or RPC. They also show the overlays: aggregated lock holding and waiting time, CPU frequency time, and PC samples. The overlays will no longer line up with the execution spans or each other, but they correctly show the overall distribution. The IPC values for each kernel and user interval are proper weighted-by-duration sums of the individual IPC values.

These profiles show where larger amounts of execution and non-execution times go, and can reveal significant differences between similar threads or similar RPCs, showing how slower ones differ from normal ones. Note that *just* the user-mode PC-samples in this information for a single PID matches what a simple profiler, one that covers one process and only user-mode execution, would show.

The `-group` command-line flag accumulates row profiles as above and then within each CPU and PID and RPC group combines multiple rows based on their names and elapsed time.

Within the CPU group, all the time for all CPUs is accumulated and then divided by the number of CPUs, giving the average time breakdown across all CPUs in an entire trace. Note that *just* the PC-sample portion of this information matches what a full-CPU profiler, one that covers all processes and both kernel- and user-mode execution, would show.

Within the PID group, multiple threads with the same initial name -- letters/digits/underscore/hyphen -- are accumulated into power-of-two buckets based on elapsed time from start to finish of each thread. There are ten microsecond buckets 1, 2, 4, ... ten millisecond buckets, and four seconds buckets 1, 2, 4, 8+. In addition, for each group of like-named threads there is an average bucket. These profiles can reveal what is different between faster-than-average and slower-than-average threads doing similar work.

Within the RPC group, multiple RPCs with the same initial method name are accumulated into power-of-two buckets. This can be particularly revealing about what is different about slow RPCs, be it execution time, waiting time, low IPC, and so on.

To avoid clutter groups of just one row are dropped from the group profile unless the `-all` command-line flag is used. Their group profile would just be a duplicate of their row profile.

Keep in mind that you may need to zoom in substantially near time 0.00 to see the detail in short profile rows that initially appear to be empty.