Naps User Manual

KernelShark plugin

Contents

Intro

This document serves as a simple to grasp manual for the "naps" KernelShark plugin.

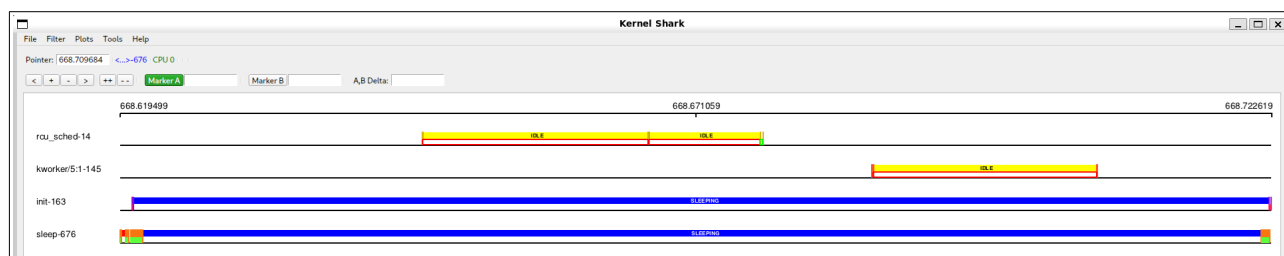For a more technical look, read the doxygen-generated HTML API documentation and source code.

Figure 1 - working Naps

"How do I build and install this plugin?"

There are two ways to build the plugin, each outlined on their own page. But first, let's talk about prerequisites and compatibilities.

Prerequisites

- **CMake** of version at least 3.1.2
- **KernelShark** an its **dependencies**
  - version *2.4.0-couplebreak* and higher for custom KernelShark
  - version *2.3.2* for unmodified KernelShark

Compatibility

Plugin is compatible with KernelShark's **custom** version *2.4.0-couplebreak* and higher. *Unmodified* (version 2.3.2) KernelShark usage is achievable through a build argument, but **discouraged**, as the custom versions improve compatibility with other plugins.

With couplebreak on (this necessitates modified KernelShark), the plugin is fully compatible with every default KernelShark plugin. With couplebreak off, *sched_events* plugin is incompatible. In general, if any plugin changes event's PID data during data stream loading, this plugin will be incompatible with it.

If using unmodified KernelShark, couplebreak cannot be turned on and nap rectangles will not have the option to have their outlines be the same color as the owner task of a sched/sched_switch event.

No other dependencies are necessary, except the standard libraries of C and C++.

Build and install only this plugin

1. Set your working directory in terminal as the build directory (best created in the project's root directory (see *README.md*), if not already present).
2. Run `cmake ..` command (if the main `CMakeLists.txt` file isn't in the parent folder, provide cmake with its valid location).
   - If using an unmodified KernelShark copy, add `-D_UNMODIFIED_KSHARK=1` to the command.
     - If this was on, binaries, symlinks and their corresponding target names will gain `_unmod` suffix in their output names
   - If **Doxygen documentation** is desired, include `-D_DOXYGEN_DOC=1` in the command.
   - If **trace-cmd header files** aren't in `/usr/include`, specify so via `-D_TRACECMD_INCLUDE_DIR=[PATH]`, where `[PATH]` is replaced by the path to the header files.
   - If **trace-cmd shared libraries** aren't in `/usr/lib64`, specify so via `-D_TRACECMD_LIBS_DIR=[PATH]`, where `[PATH]` is replaced by the path to the shared libraries.
   - By default, the **build type** will be `RelWithDebInfo` - to change this, e.g. to `Release`, use the option `-DCMAKE_BUILD_TYPE=Release`.
   - If **Qt6 files** aren't in `/usr/include/qt6`, use the option `-D_QT6_INCLUDE_DIR=[PATH]`, where `[PATH]` is replaced by the path to the Qt6 files.
     - Build instructions still expect that the specified directory has same inner structure as the default case (i.e. it contains `QtCore`, `QtWidgets`, etc.).
   - If **KernelShark source files** aren't in the relative path `../KS_fork/src` from this directory, use option `-D_KS_INCLUDE_DIR=[PATH]`, where `[PATH]` is replaced by the path to KernelShark source files.
   - If **KernelShark's shared libraries** (`.so` files) aren't in `/usr/local/lib64`, use the option `-D_KS_SHARED_LIBS_DIR=[PATH]`, where `[PATH]` is replaced by the path to KernelShark shared libraries.
3. Run `make` while still in the `build` directory.
   - If only a part of building is necessary, select a target of your choice.
   - Just running `make` builds: **the plugin** (target `naps/naps_unmod`), **symlink** to the plugin SO (target `naps_unmod_symlink`) and, if specified, the **Doxygen documentation** (target `docs`).
4. (**Installation**) Plug in the plugin into KernelShark - either via KernelShark's GUI or when starting it via the CLI with the `-p` option and location of the symlink or the SO itself.

Use `make clean-bin-dir` to remove the directory containing binaries and symlinks. Using plain `make clean` will only remove most recently (re)built target and its symlink.

Building KernelShark from source and this plugin with it

1. Ensure all source files (`.c`, `.cpp`, `.h`) of Naps are in the `src/plugins` subdirectory of your KernelShark project directory.
2. Ensure the `CMakeLists.txt` file in said subdirectory contains instructions for building the plugin (copy the style of other Qt-using GUI plugins).
3. Build KernelShark (plugins are built automatically).
4. (**Installation**) Start KernelShark. Plugins built this way will be loaded automatically. If that for some reason failed, look for the SO as for any other default-built KernelShark plugin, again in GUI or via the CLI.

"How do I enable/disable naps?"

Enabling the plugin is very simple, since plugin follows standard KernelShark plugin loading behaviour. All one has to do is open KernelShark and navigate to `Tools > Manage Plotting` plugins toolbar menu (figure 2) button. If the plugin was loaded via the command-line interface, it will be shown in the list of plotting plugins as a checkbox plus the name, checkbox already being ticked. If not, it is possible to search for the plugin via provded `Tools > Add plugin` button - it's sufficient to find the symlink, but searching for the actual shared object file is possible too.
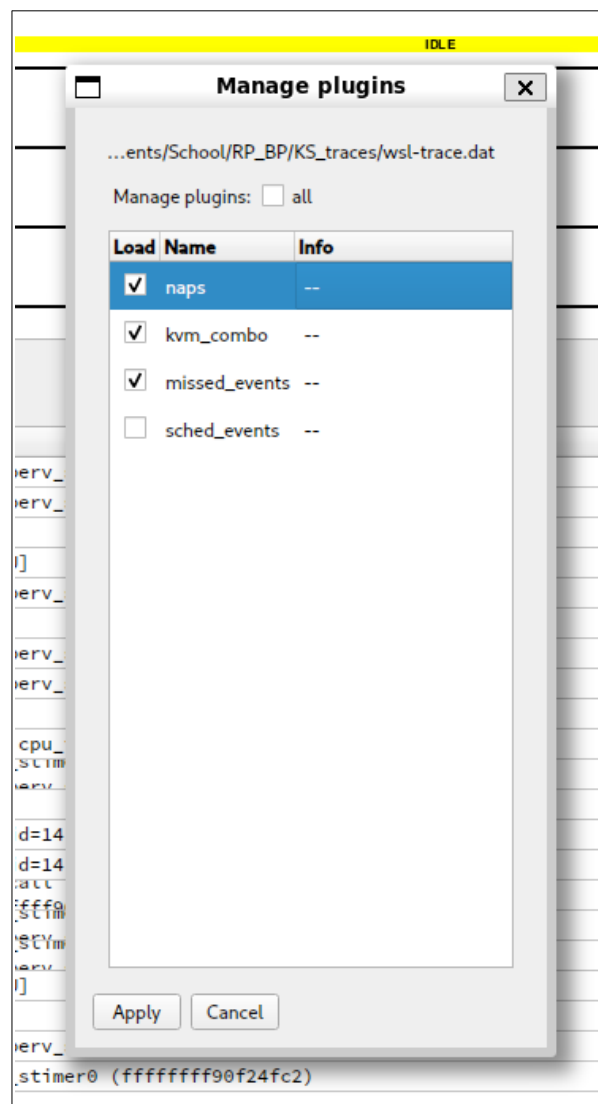

Figure 2 - Manage Plotting plugins with loaded Naps

Ticked checkbox means the plugin is enabled, empty checkbox means the plugin is disabled.

"How do I use naps?"

Configuration

Plugin configuration can be done at any time, even before any trace file is loaded.

If the plugin is enabled, additional button will appear in `Tools` menu with the label `Naps Configuration` (figure 3). Clicking on it will show a window dialog (figure 4), which will house configuraton options for the plugin. One of the two configuration options available for this plugin is the maximum amount of entries visible on the graph before the plugin is allowed to work. This configuration option can help if there are either not enough visible plugin shapes for a current zoom level or if there are too many and program memory is is too great.
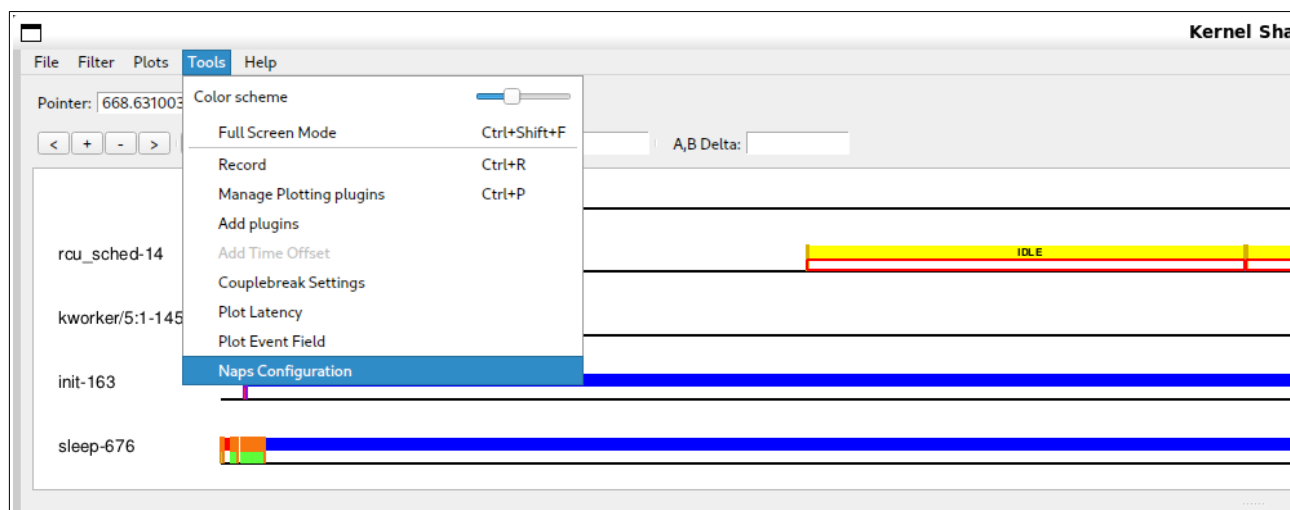

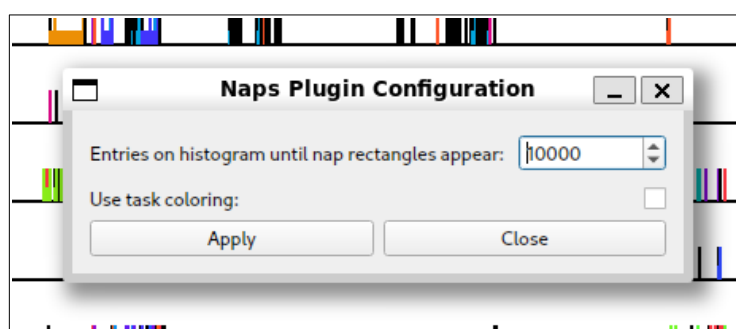Figure 3 - Naps configuration button in the Tools menu

Figure 4 - Naps configuration window

The second configuration option is toggled via the labeled checkbox visible in (figure 4). The option is whether to color the nap rectangle's top and bottom outlines the same color as KernelShark uses for the task (figure 5). Otherwise, the same color as the one used for the filling will be used (figure 6).
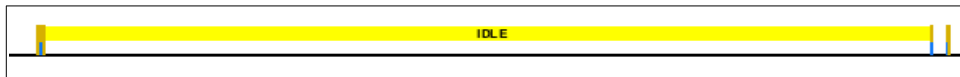
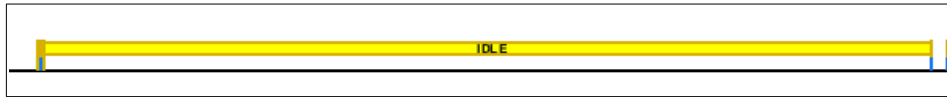Figure 5 - default coloring of a visualised nap

Figure 6 - task-like coloring of the top and bottom outlines of a visualised nap

Clicking on **Apply** button will confirm changes made to the configuration and close the window, showing a pop-up (figure 7) if the the operation was successful. Clicking on the **Close** button or the **X** button in the window header will close the window without applying any changes. Changes made to a window that hasn't applied them to the configuration will be lost and upon reopening, the window will again show what's in the currently applied configuration.

![Fig. 7](../images/NapsConfigSuccess.png)
Figure 7.

In regards to KernelShark's sessions, the configuration is NOT persistent and options included before will have to be adjusted again upon a new session or trace file load.

In the graph

Visualised nap has already been shown in figures 3 and 4, this section talks about these visualisations more.

To view naps in a graph of a trace, let KernelShark show you task plots. The plugin will automatically draw rectangles between switch and waking events, coloring them according to the previous state of the task that will then switch to another task. Top and bottom outlines of the rectangles are colored using the same color, or, if turned on in the configuration, using the task's own color. If the rectangle is wide enough, then the full name of the previous state of the task will be displayed as well (figure 8).
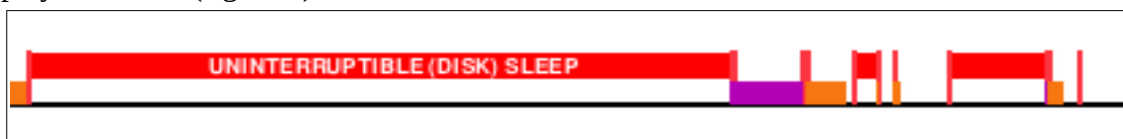

Figure 8 - a nap with multiple disk sleeps of different durations at the same zoom level

The rectangles will be visible as long as the zoom level allows two entries belonging to the same nap to also be visible and as long as there aren't too many entries visible on the graph (this can be adjusted in the configuration).

The rectangles cannot be interacted with in any capacity.

Using naps as a library

See technical documentation, as this is not intended usage of the plugin and such usage explanations will be omitted.

Bugs & glitches

No known bugs, but there were plenty of unexpected surprises during plugin testing - if more were discovered, contact the author via e-mail *djsebofficial@gmail.com*.

Recommendations

A few recommendations of usage by the author for the smoothest user experience.

It is recommended to turn on couplebreak in KernelShark to allow greater compatiblity with other plugins, especially sched_events.

It is recommended to not set the histogram limit in the configuration too high as to not make the plugin use too much memory with many nap rectangles being present.

While KernelShark's sessions work, they are a little buggy. This plugin attempts its best to not get in the way of their inner logic, but a warning should be issued that if the plugin isn't loaded beforehand, there might be unexpected behaviours, e.g. loading a session when the plugin was active won't add the plugin's menu to the `Tools` menu.