Leviathan: Taming the #ifdef Beast in Linux et al.*

Wanja Hofer, Christoph Elsner, Frank Blendinger, Wolfgang Schröder-Preikschat, Daniel Lohmann
Friedrich–Alexander University Erlangen–Nuremberg
{hofer,elsner,wosch,lohmann}@cs.fau.de

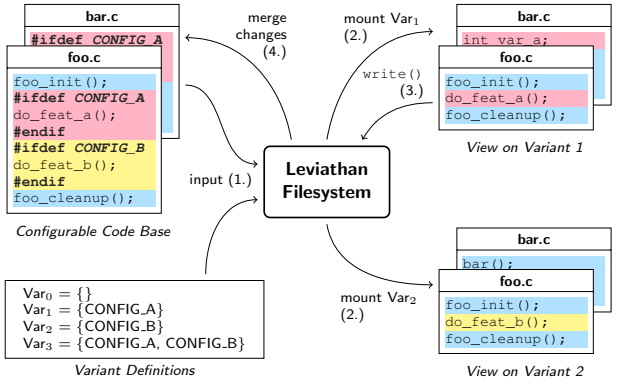
* This work was partly supported by the German Research Council (DFG) under grant no. SCHR 603/7-1.

1. MOTIVATION

A lot of Linux's success in very different application domains stems from its configurability; it currently offers more than 10,000 features, a number that has doubled in the last five years. The *implementation* of configurability in system software is mostly realized via #ifdef directives, leading to Linux having more than 80,000 conditionally included code blocks. In his keynote at ECRTS 2010, Linux kernel developer Thomas Gleixner mentioned this fact to be one of the most pressing maintenance challenges in Linux: "#ifdef's sprinkled all over the place are neither an incentive for kernel developers to delve into the code nor are they suitable for long-term maintenance."

We therefore propose to provide the developer with *variant views* on the filesystem level, effectively providing virtual files with preprocessed content according to a given configuration. Using our LEVIATHAN filesystem, arbitrary editors and tools that work on files can benefit from operating on variants instead of having to operate on potentially cluttered configurable source code. This enables our system to be toolchain-independent, which is a crucial property both in open-source development like the Linux kernel, where the freedom of choosing your editor is existential, and in industry settings, where diverse and domain-specific tools are being used.

2. LEVIATHAN IN ACTION



Using the example of Linux, the LEVIATHAN filesystem roughly works as follows (see the steps in the figure). First, LEVIATHAN is given one or more variant definitions as output by `make config` and the directory of the configurable Linux code base (1.). This makes the variants accessible at the specified mount points (2.), enabling the user to simultaneously work on different views of the same source file. Thus, the developer can focus on a small amount of features or even a single feature at a time. If needed, maintenance operations can be performed using the editor of choice, eventually issuing a `write()` system call (3.). This triggers LEVIATHAN's write-back engine to merge the changes made on the virtual file back to the original file (4.).

LEVIATHAN eases several typical Linux development tasks:

- Reasoning about (and comprehending) Linux variants and their implementation differences by comparing files from mounted configurations.
- Debugging a *specific* Linux variant that has been reported to exhibit a bug and fixing the bug directly in the variant—LEVIATHAN will write back the changes to the code base.
- Refactoring only code that belongs to a specific maintainer by having scripts run on the appropriately configured view.
- Enabling the use of #ifdef-*unaware* analysis tools that do not work on configurable code, such as WCET tools or deadlock detection tools.
- Analyzing feature changes by mounting the *same* variant of *different* kernel versions simultaneously.

3. WRITE-BACK SUPPORT

A core challenge in LEVIATHAN is the write support in the mounted views. Since the filesystem is only invoked upon `write()` system calls, it only has discrete content snapshots to operate on instead of an actual edit protocol. This renders changes performed on the view ambiguous (consider inserting a line between two former configuration blocks), which we aim at controlling using two configurable write-back modes.

A *heuristics engine* uses a merge algorithm as known from revision control systems to assign changes to lines and lines to former configuration blocks. After merging, the configuration blocks are written back to the code base, together with their original #ifdefs.

To make edit intentions unambiguous, LEVIATHAN can output language-dependent comment lines in the views as *markers* between former #ifdef blocks. This way, lines can directly be assigned to blocks and written back to the code base. This option re-introduces clutter to a small extent and only for blocks visible in the view.