Sebastian Just

Brian Mitchell

SE-577-003 - SP 21-22

17 April 2022

Paper 1 - Linux as a Case Study

Introduction

This essay analyzes the paper *Linux as a Case Study: Its Extracted Software Architecture* by Ivan T. Bowman, Richard C. Holt, and Neil V. Brewster, published in ICSE '99, Los Angles, CA.

It is tried to estimate how helpful the extract architecture is for an interested person. Additionally, how the presented software architecture can be used on the Linux kernel of today: A code base that increased almost 35 times in LOC since 1999.

Analysis

Conceptual Linux kernel architecture

The paper starts by building a conceptual architecture based on research and using existing documentation on the Linux kernel and similar kernels.

This conceptual architecture lacks (intentionally) a lot of details and is still helpful for any reader interested in the Linux architecture today: Even though this architecture was developed over 20 years ago, it contains only high-level building blocks that are very unlikely to change. This fact is amplified by the fact that the level of detail of the conceptual architecture is on the level of the architectural style itself: The Linux kernel was and is a monolithic kernel (with loadable modules): It isn't a microkernel and therefore the language used (verbs) in the conceptual architecture remains the same.

This knowledge, as presented in 1999, still remains helpful to this day: The Linux kernel still uses the facade design pattern and the strong implementation-hiding principles ensured easy understandable and maintainable code - evidently after multiple decades of active development of the code base.

Concrete Linux kernel architecture today

The presented process to gather the concrete architecture of the code base in itself would not work today as one crucial step is manual: The hierarchical decomposition by not only looking at the directory structure of the source code but also reading source code files itself.

So while the lined out process pipeline itself could probably be restored from a technical point of view (though a lot of the links of the references don't work anymore, including the link to the PBS tools), the manual part of the process will be hard to replicate: The current stable Linux kernel 5.17.3 has a staggering 4320 directories alone (excluding documentation). Any process that requires a manual input factor seems unrealistic to achieve on such a scale.

Proposed improvements

While the described process worked in 1999 and still holds in 2022 to a certain degree, I'm surprised that a simple fact of the Linux kernel is not taken into account - and probably would be even more helpful in 2022: The fact that it is written in the strongly typed language C. This means that during compile time all information are available (contrary to a interpreter language) and allow for a process depicted below:

Side note

With the strong usage of LLVM as the backend for a bunch of modern compilers, the output of various stages of compiler can be intercepted and further processed without the need to write a compiler from scratch for non-machine-code-output operations. Avoiding the risk of using software that becomes unmaintained or unavailable in the future to follow the process.

Processing architecture

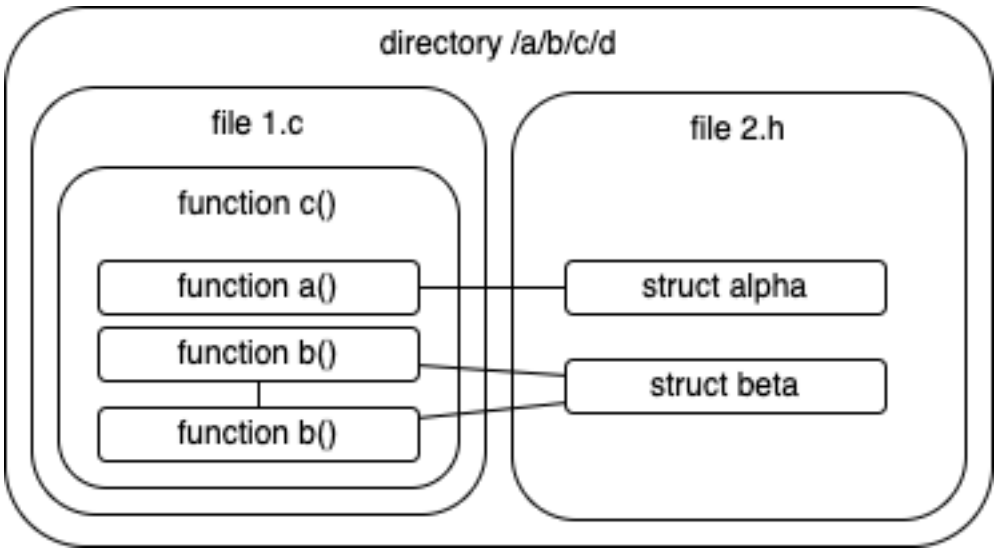
1. (Compiler step) Parse all source files of the Kernel
2. (Compiler step) Build up the abstract syntax tree (AST) of the whole kernel including debug information with file/path location
3. Export the ASL tree as a network: As C is not an object oriented language use functions and structs as nodes:

(a) Connect functions that use each other - if a function is only used by another function, embed it into its 'calling function' node.

(b) Create edges between functions and the structs they use.

(c) Use file names and directories as grouping box for the created nodes

Below is a schematic view of the desired elements of the network:



4. Use a network algorithms - like a hierarchical greedy clustering to achieve superclustering - to automatically identify edges crossing multiple edge boundaries.

   (a) Manually analyze where there are cross-boundary edges between nodes that span directories - those might exist as described in the paper for performance optimizations reasons and such.

5. Use the directory - due to the lack of a namespace concept in C - names of the encompassing structures to derive the bigger building blocks of the architecture.

This approach is overall emphasizing stronger on the 'actual architecture' than the 'desired architecture' - but after more than 2 decades of active work on the Linux kernel, the usage of other resources as comparison material seems to be futile: Too much research and practical experience has been added to the kernel over time that speaks strongly for the current existing architecture itself being proper and maintainable.