Speeding up kernel builds via automated header refactoring

Tanzir Hasan - Nick Desaulniers

Google Kernel Exchange ‘23H2

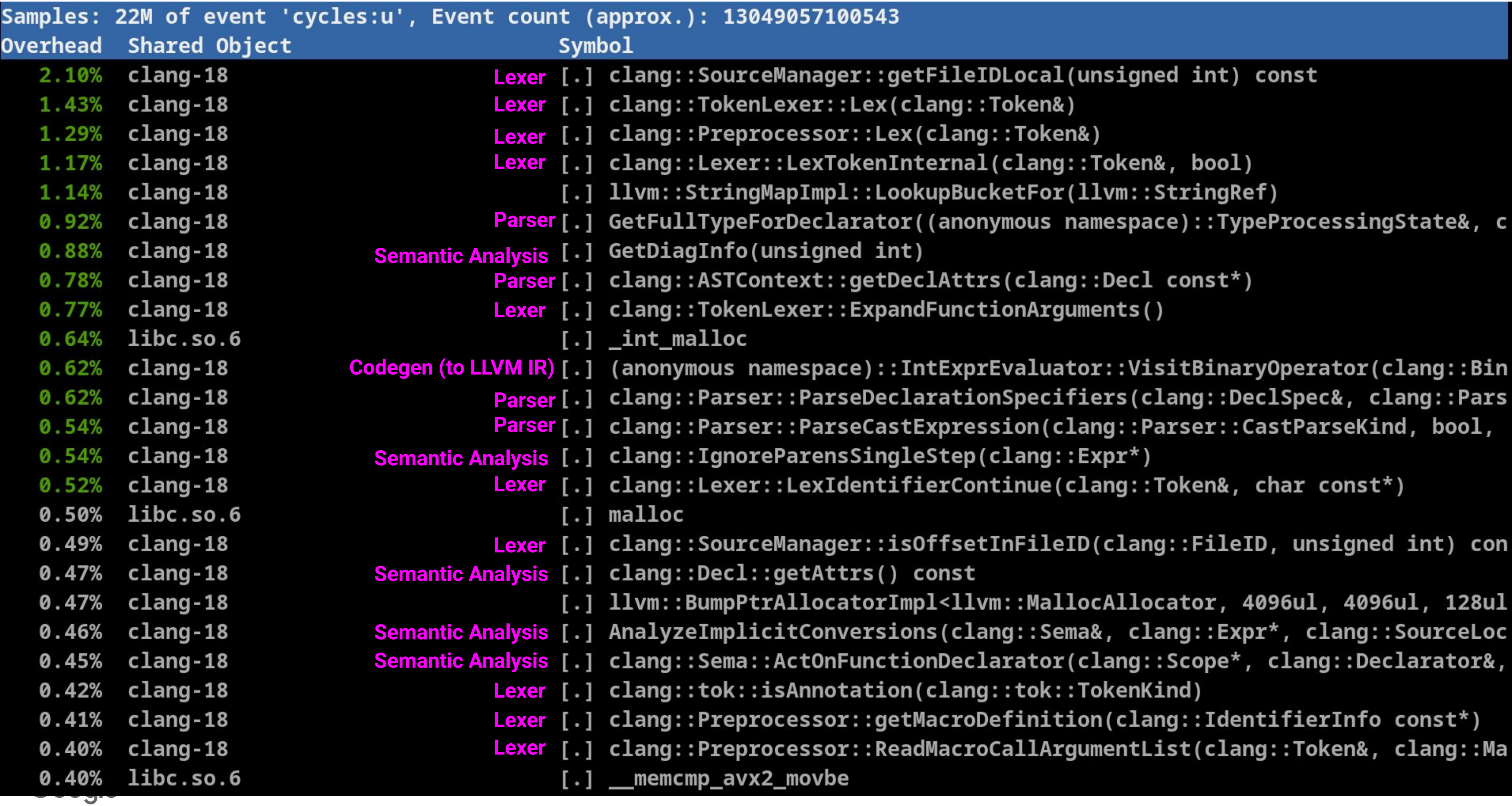
Problem

Build times are held back by lexing and parsing needless tokens.

Headers tend to grow over time (30+ years).

Removing/refactoring headers for a fast moving project like the kernel is painful.

Lack of tooling for the problem in general.



```
Samples: 22M of event 'cycles:u', Event count (approx.): 13049057100543
Overhead  Shared Object                     Symbol
   2.10%  clang-18                    Lexer [.] clang::SourceManager::getFileIDLocal(unsigned int) const
   1.43%  clang-18                    Lexer [.] clang::TokenLexer::Lex(clang::Token&)
   1.29%  clang-18                    Lexer [.] clang::Preprocessor::Lex(clang::Token&)
   1.17%  clang-18                    Lexer [.] clang::Lexer::LexTokenInternal(clang::Token&, bool)
   1.14%  clang-18                          [.] llvm::StringMapImpl::LookupBucketFor(llvm::StringRef)
   0.92%  clang-18                   Parser [.] GetFullTypeForDeclarator((anonymous namespace)::TypeProcessingState&, c
   0.88%  clang-18        Semantic Analysis [.] GetDiagInfo(unsigned int)
   0.78%  clang-18                   Parser [.] clang::ASTContext::getDeclAttrs(clang::Decl const*)
   0.77%  clang-18                    Lexer [.] clang::TokenLexer::ExpandFunctionArguments()
   0.64%  libc.so.6                         [.] _int_malloc
   0.62%  clang-18     Codegen (to LLVM IR) [.] (anonymous namespace)::IntExprEvaluator::VisitBinaryOperator(clang::Bin
   0.62%  clang-18                   Parser [.] clang::Parser::ParseDeclarationSpecifiers(clang::DeclSpec&, clang::Pars
   0.54%  clang-18                   Parser [.] clang::Parser::ParseCastExpression(clang::Parser::CastParseKind, bool,
   0.54%  clang-18        Semantic Analysis [.] clang::IgnoreParensSingleStep(clang::Expr*)
   0.52%  clang-18                    Lexer [.] clang::Lexer::LexIdentifierContinue(clang::Token&, char const*)
   0.50%  libc.so.6                         [.] malloc
   0.49%  clang-18                    Lexer [.] clang::SourceManager::isOffsetInFileID(clang::FileID, unsigned int) con
   0.47%  clang-18        Semantic Analysis [.] clang::Decl::getAttrs() const
   0.47%  clang-18                          [.] llvm::BumpPtrAllocatorImpl<llvm::MallocAllocator, 4096ul, 4096ul, 128ul
   0.46%  clang-18        Semantic Analysis [.] AnalyzeImplicitConversions(clang::Sema&, clang::Expr*, clang::SourceLoc
   0.45%  clang-18        Semantic Analysis [.] clang::Sema::ActOnFunctionDeclarator(clang::Scope*, clang::Declarator&,
   0.42%  clang-18                    Lexer [.] clang::tok::isAnnotation(clang::tok::TokenKind)
   0.41%  clang-18                    Lexer [.] clang::Preprocessor::getMacroDefinition(clang::IdentifierInfo const*)
   0.40%  clang-18                    Lexer [.] clang::Preprocessor::ReadMacroCallArgumentList(clang::Token&, clang::Ma
   0.40%  libc.so.6                         [.] __memcmp_avx2_movbe
```
]

Problem Cont.

- Many files become hundreds of times longer when preprocessed meaning millions of extra lines.
- Increased load of bad imports puts significant burden on the lexer and parser in particular, in addition to later parts of the compilation pipeline.
- Unnecessary imports lead to bigger compiler IR.
- Compiler frontend does not sufficiently address preprocessing bloat.

Motivation

Ingo Molnar has been reworking the headers of the linux kernel to build faster.

- v3: https://lore.kernel.org/lkml/YjBr10JXLGHfEFfi@gmail.com/
- v2: https://lore.kernel.org/lkml/Ydm7ReZWQPrbIugn@gmail.com/
- v1: https://lore.kernel.org/lkml/YdIfz+LMewetSaEB@gmail.com/
  - *“The fast-headers tree offers a* ***+50-80% improvement in absolute kernel build performance*** *on supported architectures, depending on the config. This is a major step forward in terms of Linux kernel build efficiency & performance.”*
  - *How “we could automate” this? (Unanswered)*
- tree: https://git.kernel.org/pub/scm/linux/kernel/git/mingo/tip.git/log/?h=sched/headers

Unclear what the status of this series is. We want to stop this from being an issue ever again. Can automation help?

Additional benefits

Improves :

- build times
- bisection times
- reduction times

Include-What-You-Use

- Include What You Use (IWYU) is a tool for including only necessary header files.
- This helps make indirect includes direct, as well as remove dead includes.
- This is a tool that is primarily used for C++ but can also be used for C.
- Since the Linux Kernel is a large and organized code base it is possible to use IWYU.
- The problem lies in the fact that not all headers are compatible with every configuration and IWYU has defaults that don’t work out of the box for the Linux kernel.

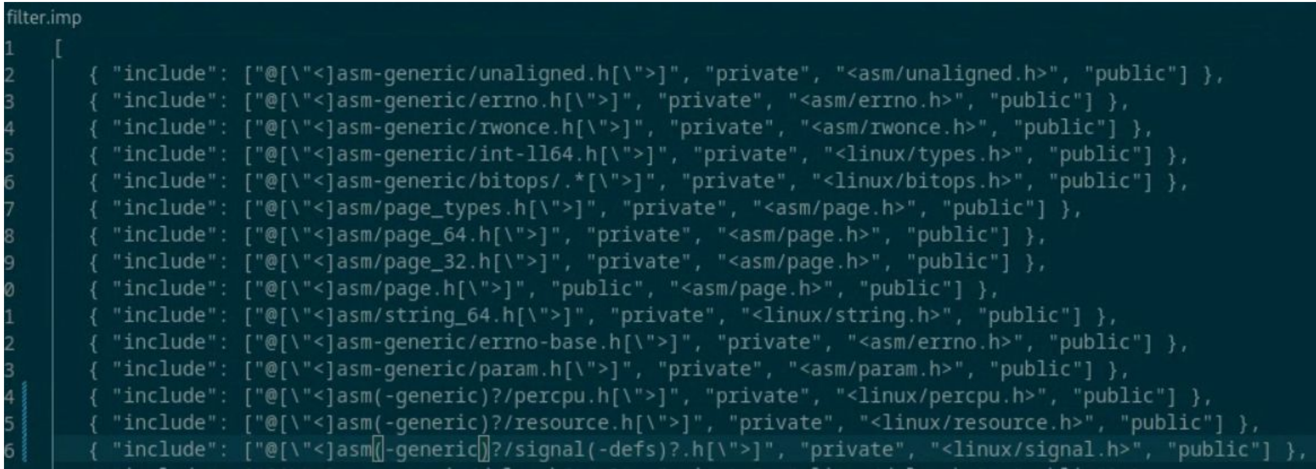
Problems with IWYU continued

- Typedefs like int64_t are commonly defined in stdint.h
  - Linux doesn’t define `int64_t` in `stdint.h.`
  - Linux doesn’t have `stdint.h.`
  - Linux defines `int64_t` in `include/linux/types.h.`
  - IWYU has built in “accelerator tables” which map commonly referenced symbols to headers
  - IWYU uses these tables to recommend including `stdint.`h, which doesn’t exist!
  - Fixed by telling IWYU not to use the standard built in tables, or even ones curated to the kernel.
- When built with a -I, IWYU *sometimes* uses “header” as opposed to <header>.

IWYU Mappings for avoiding asm/asm-generic

Mappings allow us to specify certain headers as private.

This allows IWYU to propose changes that work across multiple configurations as headers that are exclusive to a few configurations are not included unless they were already in the file. Maybe can be generated from `include/asm-generic/Kbuild`, and `arch/*/include/asm/Kbuild`.

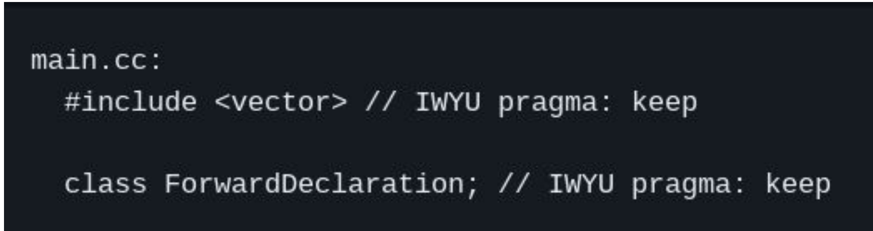


filter.imp

```
[
    { "include": ["@[\"<]asm-generic/unaligned.h[\">]", "private", "<asm/unaligned.h>", "public"] },
    { "include": ["@[\"<]asm-generic/errno.h[\">]", "private", "<asm/errno.h>", "public"] },
    { "include": ["@[\"<]asm-generic/rwonce.h[\">]", "private", "<asm/rwonce.h>", "public"] },
    { "include": ["@[\"<]asm-generic/int-ll64.h[\">]", "private", "<linux/types.h>", "public"] },
    { "include": ["@[\"<]asm-generic/bitops/.*[\">]", "private", "<linux/bitops.h>", "public"] },
    { "include": ["@[\"<]asm/page_types.h[\">]", "private", "<asm/page.h>", "public"] },
    { "include": ["@[\"<]asm/page_64.h[\">]", "private", "<asm/page.h>", "public"] },
    { "include": ["@[\"<]asm/page_32.h[\">]", "private", "<asm/page.h>", "public"] },
    { "include": ["@[\"<]asm/page.h[\">]", "public", "<asm/page.h>", "public"] },
    { "include": ["@[\"<]asm/string_64.h[\">]", "private", "<linux/string.h>", "public"] },
    { "include": ["@[\"<]asm-generic/errno-base.h[\">]", "private", "<asm/errno.h>", "public"] },
    { "include": ["@[\"<]asm-generic/param.h[\">]", "private", "<asm/param.h>", "public"] },
    { "include": ["@[\"<]asm(-generic)?/percpu.h[\">]", "private", "<linux/percpu.h>", "public"] },
    { "include": ["@[\"<]asm(-generic)?/resource.h[\">]", "private", "<linux/resource.h>", "public"] },
    { "include": ["@[\"<]asm(-generic)?/signal(-defs)?.h[\">]", "private", "<linux/signal.h>", "public"] },
```

Problems with Macros

- IWYU doesn’t know when Macros are called/used
- Oftentimes IWYU rips headers with Macros out entirely.
- Duplicate headers are always removed. This makes it impossible to use X-Macros (https://quuxplusone.github.io/blog/2021/02/01/x-macros/)
- Dealing with these will require manual effort.
- This could be assisted by changes in the kernel code to include IWYU Pragmas.
- Token pasting identifiers makes analysis tricky.

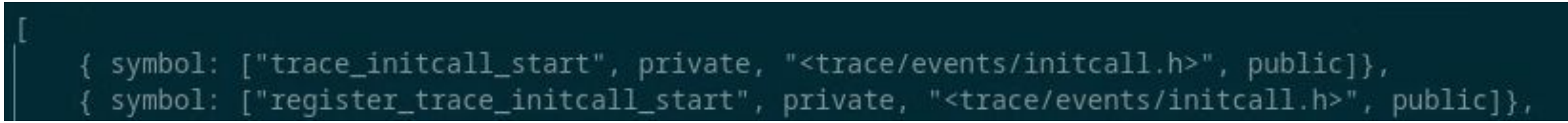


```
main.cc:
  #include <vector> // IWYU pragma: keep

  class ForwardDeclaration; // IWYU pragma: keep
```

Going Forward with IWYU

- Just as IWYU has inclusion tables, it also has symbol tables.
- Calls and functions included in the header alongside macro definitions can be used to ensure that X-Macros function properly.
- Symbol tables form more accurate header inclusions and lower the amount of manual work needed for the automation process.
- They are time consuming to create and must be kept in sync with kernel version.



```
[
    { symbol: ["trace_initcall_start", private, "<trace/events/initcall.h>", public]},
    { symbol: ["register_trace_initcall_start", private, "<trace/events/initcall.h>", public]},
```

Some Statistics

For the x86 defconfig build lib/string.o:

- Pre IWYU Preprocessing Size: 23941 lines of Code
- Post IWYU Preprocessing Size: 5092 lines of Code (78% smaller)
- Pre IWYU build time: .36 seconds
- Post IWYU build time: .12 seconds

When using an automated IWYU script on lib/string.c the actual binary code did not change across 3 distributions and configurations except for one *LINE* number used in a WARN_ON when dead headers were removed.

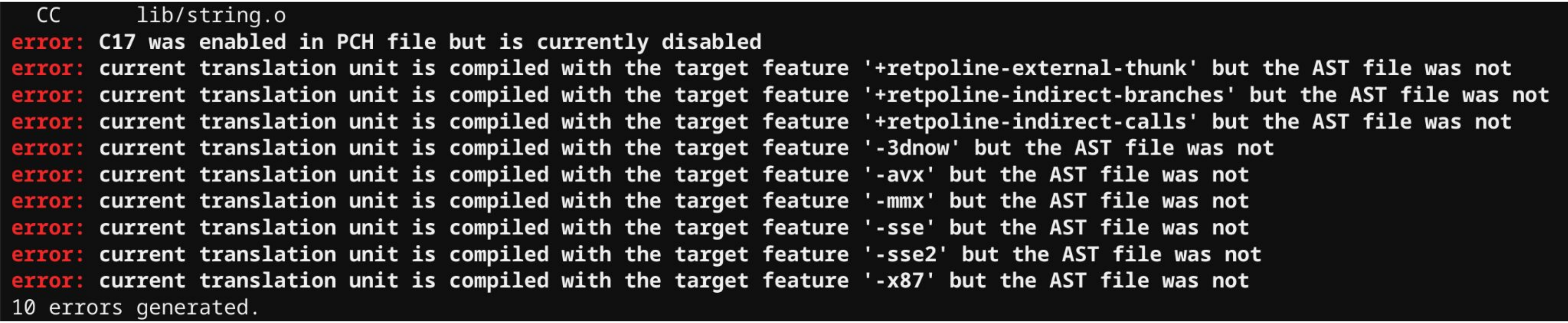
Progress so Far:

- On a machine with 128 cores an x86 defconfig all build takes around 72.3 seconds
- After changes to 220 files it took around 69.0 seconds.
- The script looked at 300 files in total and was able to automatically change only 220.
- In the compile commands.json there are roughly 2700 files in a defconfig all, So there are significantly greater speed gains available.
- Over 1 million lines of code removed.

Precompiled Headers

- Precompiled headers can speed up build times. Are the basis for C++20 modules.
- This can be done with the most frequently occurring headers across all builds.
- Portability paper cuts (designed to mmap AST into memory; AST representations differ between compilers and also compiler versions).
- Some Candidates (forcibly injected into all TUs via `-include`):
  - compiler_types.h
  - kconfig.h
  - compiler-version.h

PCH and ABI



```
  CC      lib/string.o
error: C17 was enabled in PCH file but is currently disabled
error: current translation unit is compiled with the target feature '+retpoline-external-thunk' but the AST file was not
error: current translation unit is compiled with the target feature '+retpoline-indirect-branches' but the AST file was not
error: current translation unit is compiled with the target feature '+retpoline-indirect-calls' but the AST file was not
error: current translation unit is compiled with the target feature '-3dnow' but the AST file was not
error: current translation unit is compiled with the target feature '-avx' but the AST file was not
error: current translation unit is compiled with the target feature '-mmx' but the AST file was not
error: current translation unit is compiled with the target feature '-sse' but the AST file was not
error: current translation unit is compiled with the target feature '-sse2' but the AST file was not
error: current translation unit is compiled with the target feature '-x87' but the AST file was not
10 errors generated.
```

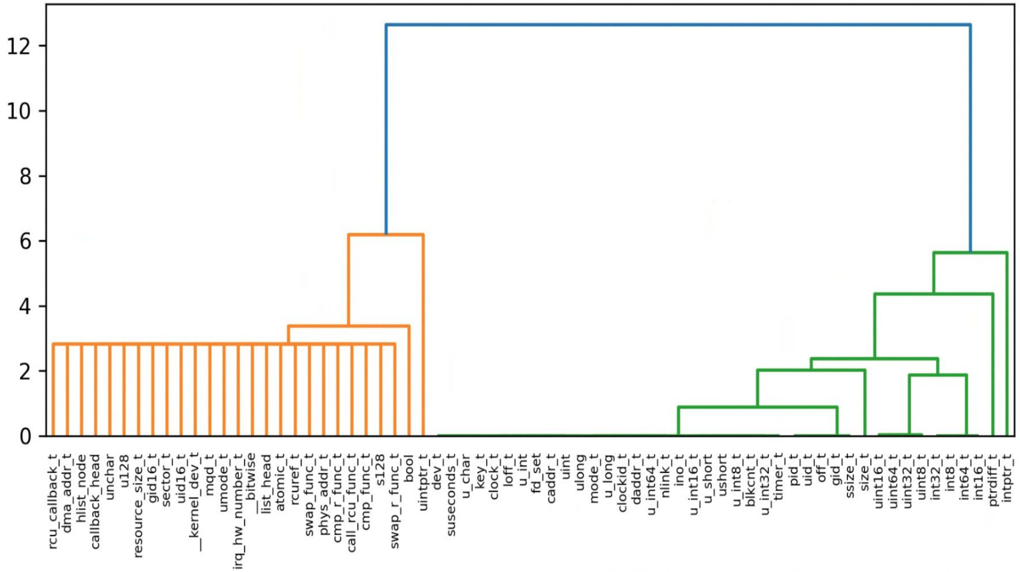
Future research & tooling

- Precompiled headers
- Automating header refactoring
    - Statistical analyses
    - Given an identifier, where are the uses?

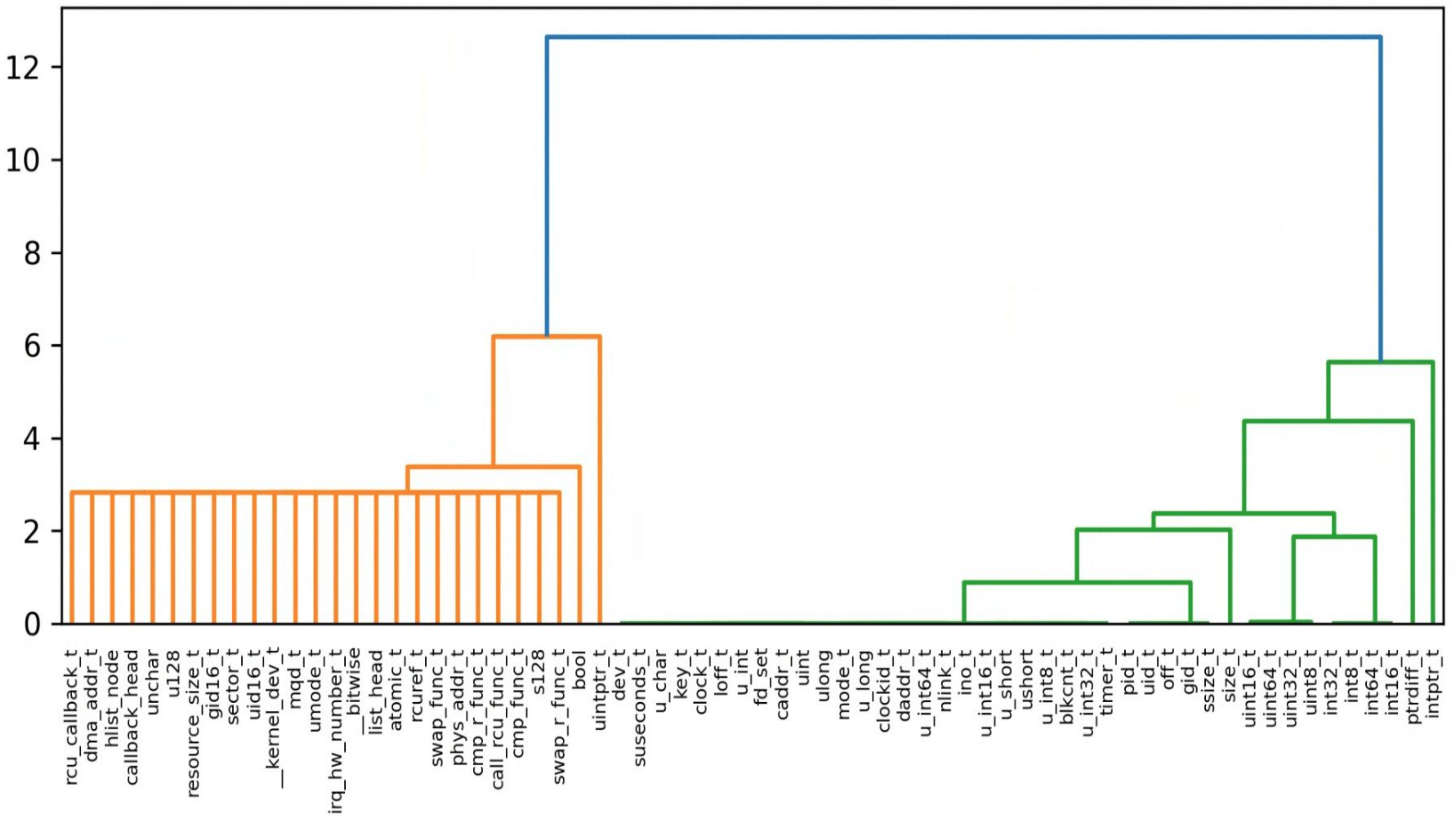
Statistical analysis to inform header refactoring

IWYU is only updating the header include list in .c files. A potentially more effective way could be to actually break up the headers themselves.

We can use hierarchical agglomeration or other graph partitioning techniques to essentially break a fully connected graph of symbols into 2 parts.



Basic Hierarchical Agglomeration of linux/types.h



Future research & tooling

- Precompiled headers
- Automating header refactoring
  - Statistical analyses
  - Given an identifier, where are the uses?
- llvm-extract equivalent for C code
  - Given an identifier and a source file defining it, move it to a new file, update uses
- modpost improvements
  - Not specific to kernel headers
  - commit 4074532758c5 ("modpost: Optimize symbol search from linear to binary search") was a nice recent win
    - *“saves a few seconds of wall time for defconfig builds, but* ***can save several minutes on allyesconfigs****”*
- Detecting circular includes
- <What other tooling should we be looking to build?>