Using clang-tidy and clang-format

Linux Plumbers Conf 2020 - LLVM MC
Miguel Ojeda, Nathan Huckleberry
What is Clang Tooling?

- Clang is one of the LLVM compiler frontends
- But it is also designed as a \textit{platform} for building \textit{source level tools}
  - Syntax checking
  - Automatic fixing of compile errors
  - Code formatting
  - Refactoring
  - Linting
  - Static code analysis
  - ...

What could the kernel use Clang Tooling for?

- In the kernel we already use a lot of custom tools
  - `sparse`
  - `spatch` (Coccinelle)
  - `checkpatch.pl`
  - ...

- Clang Tooling allows us to build more of them
  - Without having to deal with parsing C code
  - With support for all C extensions we use
  - With easy access to the AST (Abstract Syntax Tree)
clang-format — What is it?

- A tool to format C-like code
  - Supports different languages
  - Many configurable rules and heuristics
  - Very fast
  - Good enough

- Allows us to spend less time on formatting and reviewing style

- Kernel style already *pre-configured* for you
  - Overriding the global style for particular subsystems is possible
  - See `.clang-format`
static inline void
cpuhp_lock_acquire ( bool bringup ){
lock_map_acquire ( bringup
? & cpuhp_state_up_map
: & cpuhp_state_down_map );
}
static inline void cpuhp_lock_acquire ( bool bringup ){
    lock_map_acquire ( bringup 
        ? & cpuhp_state_up_map 
        : & cpuhp_state_down_map );
}

static inline void cpuhp_lock_acquire (bool bringup) {
    lock_map_acquire (bringup ? &cpuhp_state_up_map : &cpuhp_state_down_map);
}
clang-format — Common use cases

- Re-formatting blocks
  - After indenting to keep line length within the limit
  - After refactoring and moving code around

- Re-formatting full files

- Sorting `#includes`

- Aligning
  - Variable blocks (on types or on the `=`)
  - Multi-line macro continuations (`\`)

- Reflowing comments

- ...
clang-format — Kernel use cases

- Re-formatting patches
  - Clang provided script: `clang-format-diff.py`
  - Useful to spot coding style mistakes, typos, etc.
  - For both authors and maintainers

- Dealing with “code dumps”
  - e.g. big systems developed out-of-tree and then submitted at once
  - Get them kernel-formatted at a single blow

- Lowering the barrier of entry a tiny bit
  - Kernel development is already different enough
  - One less thing to care about for newcomers
clang-format — How to run it

● In your editor
  ○ Binding it to a key or at save file time
  ○ Vim, Emacs, VS Code, CLion, gedit, Sublime Text…
  ○ See [https://clang.llvm.org/docs/ClangFormat.html](https://clang.llvm.org/docs/ClangFormat.html)

● In the command-line
  ○ `clang-format -i kernel/up.c`
  ○ `git diff`
clang-format — The {plan, proposal}

- **Short-term**
  - Get more devs aware of it
  - Get more devs to use it themselves

- **Medium-term**
  - Get maintainers to format their subsystems with it
  - Converge the slightly different code styles in the kernel

- **Long-term**
  - Kernel code automatically formatted
  - Style nitpicking in reviews not needed anymore!
clang-format — Further information

- **Quick kernel guide**
  - Documentation/process/clang-format.rst

- **Official documentation**
  - https://clang.llvm.org/docs/ClangFormat.html
  - https://clang.llvm.org/docs/ClangFormatStyleOptions.html
clang-tidy

“Linter” that can identify style violations, interface misuse, or bugs that can be deduced via static analysis

Used to write codebase-specific checks

More than just a linter
Clang-tidy checks

Checks consist of an AST Matcher and a check callback

AST Matcher - Look for AST Node Pattern

Check callback - Run on matching AST nodes
Writing a check

Write AST matcher ([https://clang.llvm.org/docs/LibASTMatchersReference.html](https://clang.llvm.org/docs/LibASTMatchersReference.html))

```cpp
auto ErrFn = functionDecl(hasName("ERR_PTR"));
Finder->addMatcher(
    callExpr(callee(ErrFn), hasParent(compoundStmt()))
    .bind("call", this
);

[https://clang.llvm.org/extra/clang-tidy/Contributing.html](https://clang.llvm.org/extra/clang-tidy/Contributing.html)
Writing a check

Write check callback

Matcher gives Clang AST nodes to the check callback

Now we have the full power of Clang’s frontend behind our check

https://clang.llvm.org/extra/clang-tidy/Contributing.html
Running clang-tidy

make CC=clang clang-tidy

https://patchwork.kernel.org/project/linux-kbuild/list/?series=337007
Clang analyzer

Static analysis tool for finding C/C++ bugs

Can be run through clang-tidy

Path sensitivity

Symbolic execution

Data flow analysis
Running Clang analyzer

Text only mode:

make CC=clang clang-analyzer

HTML reports

scan-build --use-cc=clang make
Clang analyzer output
Future Work

More checks for the clang-tidy linuxkernel module