Storing Clang Data
For IDEs and static analysis

Marc-André Laperle, Ericsson
AGENDA

1. Introductions
2. What is being done now
3. What do we want to store
4. What kind of format
5. Discussions
Introductions

› Marc-André Laperle
  • Software Developer at Ericsson since 2013
  • Eclipse committer for CDT (C/C++) and several other projects
  • New-ish LLVM/Clang contributor (early 2017)
  • Enthusiastic about C/C++, IDEs and tooling in general (Not a compiler expert! Yet?)

Your turn!
What is being done now

- Some ongoing things in Clang and Clang “extra”
  - Xcode 9’s “Index-while-building”
  - Clangd’s indexing
  - Clang Static Analyzer’s “Cross-TU” feature
  - Regular improvements to Clang/Index
Xcode 9 “Index-while-building”

Source: https://docs.google.com/document/d/1cH2sTpgSnJZCKZtJl1aY-ryy4uGPcrl-6RrUpdATO2Q/
Xcode 9 “Index-while-building”

Source: https://docs.google.com/document/d/1cH2sTpgSnJZCkZtJl1aY-rzy4uGPcrl-6RrUpdATO2Q/
Xcode 9 “Index-while-building”

Source: https://docs.google.com/document/d/1cH2sTpgSnJZCkZtJl1aY-rzy4uGPcrI-6RrUpdATO2Q/
Clangd’s ClangdIndexDataStorage

- Malloc-like interface to writing in a file
- Stores raw bytes, ints, string and pointers (to other locations in the file)
- Inspired from CDT’s database
Clangd’s ClangdIndexDataStorage

## File layout

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>File version</td>
</tr>
<tr>
<td>4</td>
<td>Linked list to free blocks of 8 bytes</td>
</tr>
<tr>
<td>8</td>
<td>Linked list to free blocks of 16 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>Linked list to free blocks of 4096B</td>
</tr>
<tr>
<td>2052</td>
<td>“User” data (Blocks)</td>
</tr>
</tbody>
</table>

## Data block

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Block size</td>
</tr>
<tr>
<td>2..size</td>
<td>Any “user” data</td>
</tr>
</tbody>
</table>

## Free block

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Free block size</td>
</tr>
<tr>
<td>2</td>
<td>Pointer to next free block of same size</td>
</tr>
<tr>
<td>6..size</td>
<td>Unused (until it becomes a data block)</td>
</tr>
</tbody>
</table>
### Clangd's ClangdIndexDataStorage

<table>
<thead>
<tr>
<th>Data block</th>
<th>Free block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free</td>
<td>Data</td>
</tr>
<tr>
<td>Data</td>
<td>Free</td>
</tr>
<tr>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Data</td>
<td>Free</td>
</tr>
<tr>
<td>Data</td>
<td>Free</td>
</tr>
</tbody>
</table>

4K pieces

Cache!
Clangd’s BTree

- Tree with nodes containing multiple children
- Balanced
- Logarithmic insertion, search, deletion
Clangd’s BTree

- Keys are pointers to data in ClangdIndexDataStorage
Clangd’s Index Model

ClangdIndex

- BTree of Files
- BTree of Symbols

ClangdIndexFile

- Path
- First Occurrence (linked list)

ClangdIndexOccurrence

- NextForSymbol (linked list)
- NextInFile (linked list)
- Location
- Roles
- Symbol

ClangdIndexSymbol

- USR
- FirstOccurrence (linked list)
Clang Static Analyzer

- When an analysis is executed, a “function to file location” mapping is generated in a file “externalFnMap.txt”
- Other stored information would be useful for performance:
  - A call graph would help incremental Cross-TU analysis
  - An “include graph” would help to know which build commands need to be re-analyzed when a file changes
What do we want to store

• Symbols
• Occurrences (calls, references, definitions, declarations, relations)
• File dependencies
• Static analysis checker-specific information (per symbol, per function function definition, etc). I.e. make the model extensible
• Ability to not store some information (field occurrences, etc)
What kind of format

- Xcode’s libIndexStore (LLVM Bitstream)
- Not yet fully upstreamed to Clang (Swift Github)
- Format used for “Index-while-building”
- Needs liblmdb as new dependency for mappings (or need to replace with something else)
What kind of format

- Clangd’s IndexStorage
- Not yet upstreamed to Clang (Github)
- Stand-alone (no new dependency)
- Not production ready yet
- Not created with “Index-while-building” in mind, but libIndexStore could be used as input
What kind of format

- “Third-party” Databases
  - Some are just too big dependencies to include in Clang (MySQL, PostgreSQL, etc)
  - Smaller ones interesting: SQLite, LMDB. LMDB is quite simple and fast, could be used in combination of other formats (libIndexStore, ClangdIndexStorage)
  - How likely would a third-party database be accepted in LLVM repos?
What kind of format

- How could other tools reuse this?
  - Link libclangIndex?
  - Link libclangDaemon
  - Launch Clangd, communicate with JSON-RPC
  - All of the above?
Things considered for Clangd

- “Index-While-Building” feature in Xcode 9 should be reused. The index store could be used (instead of ClangdIndexStorage). Or used as input to ClangdIndexStorage.

- Move some indexing logic to Clang/Index instead of Clangd, for others to reuse

- Make indexing extensible enough so that other tools can add information to the index (Clang Static Analyzer)

- Use liblmdb, for symbol mapping, similar to Xcode

- Support for multiple “indexes” for merging different projects, libraries, etc

- Use linking information (compile_commands.json) in order to solve the multiple definitions problem
References

- Clangd: [https://clang.llvm.org额外 documentation](https://clang.llvm.org/extra/clangd.html)
- Clang Static Analyzer: [https://clang-analyzer.llvm.org/](https://clang-analyzer.llvm.org/)
- Clang mailing list: [https://lists.llvm.org/mailman/listinfo/cfe-dev](https://lists.llvm.org/mailman/listinfo/cfe-dev)
- Xcode’s Index-while-building:
  [https://docs.google.com/document/d/1cH2sTpgSnJZCkZtJl1aYrzy4uGPcrI-6RrUpdATO2Q/](https://docs.google.com/document/d/1cH2sTpgSnJZCkZtJl1aYrzy4uGPcrI-6RrUpdATO2Q/)
BoF Notes/Minutes