Subject: Re: theide with libclang
Posted by mirek on Sat, 24 Sep 2022 06:44:58 GMT
View Forum Message <> Reply to Message

OK, so one week in, I guess I owe the community a bit of explanation about libclang and the way theide uses it...

So, obviously what libclang does is parsing the files and giving info about them. Additionally, it can provide autocomplete lists at specified point in the source files, but NOT HEADERS (and also not inside templates, but that is store for another time).

The important thing to know is that parsing the file with all headers is long: on my not so slow machine, it can take up to several seconds to parse the file. To remedy this, libclang has a concept of "preamble" and "reparsing". Preamble is basically automated precompiled header - if file starts with multiple headers, preamble is creating and when after the first parse the file is reparsed or autocomplete requested, preamble is used to greatly speedup the process. With preamble, autocomplete (after the first parse) is almost immediate. That said, premable creation is sort of fragile. Also, normally it gets created in system temp directory, which would be fine unless sometimes libclang does not delete it. But it can be solved be setting env variable which we do now.

So what we do now is the when you start editing some file, in the background thread the first parse of the file is started with a request to create preamble. In theide, you can know that this is happening by left bar going "sort of orange" for a couple of seconds - it does that each time file is parsed or reparsed. If you then change the file, it gets (after some time of inactivity) reparsed again. Parsing current file gets it ready for autocomplete, but also provides information about file for file annotations, current file navigation and "jumps" (Alt-J / Ctrl clicks).

Originally, there was just one parsed file, which meant each time after switching to another file it had to have that initial a couple of seconds parsing period. Later I have added a cache, there are now several files with parsed info. Downside is that each "parsing unit" consumes a lot of memory, about 300MB so with 12 entries in the cache, you can do the math....

Include files that clang is unable to process: What we do is that we find "master file" - the well defined file that includes the header and we pretend that we are parsing that one. It is not so simple: we still want to use preamble to make things fast and if we just parsed master file and hoped to catch the info include file from it, it would not work with preambles as our include file would actually be in preamble. I have spent a lot of time trying to solve this: What we actually do now is that we create a syntetic file content that actually manually does including so that it all works (it is hard to explain in detail; in assist diagnostic mode you can see the content of the syntetic file with Assist/Current parsed file content).

So that is about all I can say about "current file" without going into gruesome details; if you want to study the code, libclang/CurrentFile.cpp and ide/AssistTrick.cpp are good starting points.

Next troublesome part of code is "Indexer" - that is something that scans the whole sources and creates a program database of, well, everything everywhere. That is needed for Navigator and Alt+J command. We are using blitz(like) process there and we are running that in multiple threads.

Now there is "Scheduler" (Indexer::SchedulerThread()) thread that scans for changed files and basically prepares jobs for "Indexing" (Indexer::IndexerThread) threads. Once again, each of indexer threads can eat up to 300MB of memory, so finetuning default values will take some time. More indexer threads mean faster reindexing but perhaps too much memory...

In terms of slowdown/freezing during reindexing, well, the plan was to give indexer threads really low priority. It should be possible with

```
  clang_CXIndex_setGlobalOptions(clang.index,
CXGlobalOpt_ThreadBackgroundPriorityForIndexing);
```

but maybe this does not work as expected in Linux. In Windows, it seems to work fine.

Now for testing and troubleshooting, please note that there is "Assist diagnostics" option in Assist/Debug. When active, it will start reporting in Console what it is doing and also there are new entries at the end of assist menu, showing a dump of actual index, errors libclang reported during parsing and what was given to clang to parse as current file.

So if e.g. there is no autocomplete for include files, please check "Current file parse errors" and "Current parsed file content".

I know this is a big change (and one I was extremly hesitant to do) and not everything works smoothly at this point, but hopefully all will be finetuned in time. I guess we fixed a lot already in the first week....

Mirek