A mishmash of random stuff
Just another WordPress.com weblog
June 29, 2011 / pjcoup

Building a standalone preprocessor library with clang tools

People are hearing more and more about clang (http://clang.llvm.org) as an alternative compiler for C type languages. One of the positives mentioned for clang (and llvm) is the library/modular code base which can be used to create other compiler/langauge tools. Recently, folks from Intel introduced the ispc compiler (http://ispc.github.com), which is a small little compiler which targets writing C-extensions in a language that can target Intel's SSE instructions more naturally than C/C++. Matt Pharr has created a whole list (https://github.com/ispc/ispc/issues?sort=created&direction=desc&state=open) of open items which people can take a look at and contribute to. One mention that caught my eye was integrating a preprocessor rather than shelling out to `/usr/bin/cpp`. As `clang` and `llvm` are already prerequisites for `ispc`, it seemed that there should be a way to easily crank out a preprocessor from the `clang` libraries. Right?

To me, it seems mixed. Sure enough, you can create a `clang` based preprocessor with a minimal amount of code, but the amount of `clang` infrastructure that this brought in surprised me. The rest of this post will outline how I built the stand-alone preprocessor, and what surprised me. I will happily be corrected if I made things more difficult than needed. And from the outset, I freely admit I am a clang/llvm user, not a clang/llvm internals developer, so my understanding of the internals organization is not up to the clang/llvm internals developers.

First, Fedora has a `clang-devel` package available, which contains all of the `clang` headers and libraries you will need for development. If your distribution doesn't have a `clang` development package available, then follow the instructions from `clang`'s getting started (http://clang.llvm.org/get_started.html) page. The `clang` headers and source are organized in a very straight-forward way. With minimal looking, you'll see `include/clang/Frontend/PreprocessorOptions.h`, and `include/clang/Frontend/PreprocessorOutputOptions.h`. Seems like this would be the natural place where the Preprocessor would belong. Grepping, you'll find that there are a couple of promising sounding functions, `InitializePreprocessor`, and `DoPrintPreprocessedInput` which take `Preprocessor` reference arguments in `include/clang/Frontend/Utils.h`. So where is this Preprocessor? In `clang/Lex`. The same lib that contains all of `clang`'s C/Objective-C/C++ lexing code. So, it seems to get the preprocessor, we will have to take a reasonable chunk of the rest of the clang insfrastructure. It seems the easiest (although, perhaps not the way with the fewest dependencies) way to construct a preprocessor instance is to construct a full `clang::CompilerInstance`, and the fill in the necessary parts of infrastructure for file handling, diagnositcs, and, surprisingling, the target (i.e. i486-redhat-linux). Putting it all together, we get the following:

```
void
preprocess( const std::string& infile, const std::string& outf )
{
 clang::CompilerInstance inst;
 llvm::raw_fd_ostream* myos = inst.createOutputFile( outf );

 inst.createFileManager();
 inst.createDiagnostics(0, NULL);
 clang::TargetOptions& options = inst.getTargetOpts();
 options.Triple = "i486-ubuntu-linux";

 clang::TargetInfo* target =  clang::TargetInfo::CreateTargetInfo(inst.getDiagnostics(), options);
 inst.setTarget(target);
 inst.createSourceManager(inst.getFileManager());
 inst.InitializeSourceManager(infile);
 clang::PreprocessorOptions& opts = inst.getPreprocessorOpts();
 //Push back all of the options...the defs can be "FOO=val"
 //opts.addMacroDef( … );
 inst.createPreprocessor();
 clang::DoPrintPreprocessedInput( inst.getPreprocessor(),
                                  myos,
                                  inst.getPreprocessorOutputOpts() );

 myos->close();
}
```

All in all, this isn't too much code to get a preprocessor. The second part of the story is linking in the clang library dependencies. As pointed out in other forums, there is no "clang-config" in the same vein as "llvm-config", telling, among other things, which order to link the libraries on linux. After a little playing around, I finally looked at the clang makefile to see what order `clang` links its development libraries. Note that `clangFrontent` is near the start of the list, and `clangLex` is near the end of the list. There is probably a way to get by with less, but again, this seemed to be the most straightforward way.

```
CLANG_LIBS = -lclangFrontendTool -lclangFrontend -lclangDriver \
             -lclangSerialization -lclangCodeGen -lclangParse -lclangSema \
             -lclangStaticAnalyzerFrontend -lclangStaticAnalyzerCheckers \
             -lclangStaticAnalyzerCore \
             -lclangAnalysis -lclangIndex -lclangRewrite \
             -lclangAST -lclangLex -lclangBasic
```

And there you go, a preprocessor library built from the clang libraries.

Filed under Uncategorized