编译原理研讨课实验PR001说明

熟悉Clang的安装和使用

Notice

`以下实验步骤适用于在教室环境（Linux）进行，目前机房尚不能远程访问，请自行下载相关的源代码并进行实验：`

1. 下载地址，LLVM Release page, 版本为3.3;
2. `llvm-3.3.src.tar.gz`对应`LLVM Source Code`;
3. `cfe-3.3.src.tar.gz`对应`Clang Source Code`;
4. 其它为非必须项。

编译安装LLVM和Clang：

第一步：登录到本组服务器的帐号，参见实验环境说明

第二步：将源代码拷贝到本账户的目录下，并解压源代码到相应的目录

利用*scp*远程拷贝：`scp -r clang0@124.16.71.6:/home/clang0/src ./`

解压缩：

LLVM: `mkdir -p llvm-3.3 && tar xfz ./src/llvm-3.3.src.tar.gz -C llvm-3.3 --strip-components=1`

Clang: `mkdir -p llvm-3.3/tools/clang && tar xfz ./src/cfe-3.3.src.tar.gz -C llvm-3.3/tools/clang --strip-components=1`

Clang-extra: `mkdir -p llvm-3.3/tools/clang/tools/extra && tar xfz ./src/clang-tools-extra-3.3.src.tar.gz -C llvm-3.3/tools/clang/tools/extra --strip-components=1`

Compiler-RT: `mkdir -p llvm-3.3/projects/compiler-rt &&tar xfz ./src/compiler-rt-3.3.src.tar.gz -C llvm-3.3/projects/compiler-rt --strip-components=1`

第三步：编译和安装LLVM和Clang

建立构建目录(Out of Source)：`mkdir -p build && cd build`

通过CMake生成GNU标准的Makefile，并使用make进行编译和安装:

1. 设定编译过程使用的*gcc*和*g++*: `export CC=/usr/local/bin/gcc && export CXX=/usr/local/bin/g++`
2. 生成Makefile：`cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug -DCMAKE_INSTALL_PREFIX=~/llvm ../llvm-3.3`，其中*CMAKE_INSTALL_PREFIX*代表了编译完成以后的安装目录，*../llvm-3.3*代表了源代码目录
3. 使用*make*进行编译：`make -j6`，其中*j6*代表使用6个线程并行编译。该过程约需要10分钟。如果仅仅是更改了*Clang*的源代码，则无需执行第二步。
4. 安装到`CMAKE_INSTALL_PREFIX`当中：`make install`
5. 检查`~/llvm`下是否已经有对应的文件安装进去：`~/llvm/bin/clang --version`，将会有`clang version 3.3 (tags/RELEASE_33/final)`对应字样输出

6. 由于编译时间较长，我们提前准备好了一份编译后的工程，源代码在`~/src-sample`，LLVM和Clang的源代码在`~/llvm-3.3-sample`，安装好的LLVM在`~/llvm-sample`，后续实验（仅本次课程）可以用该工程进行。

###生成和查看C程序对应的AST

第一步，准备一个简单的C程序`test.c`：

```
int f(int x) {
  int result = (x / 42);
  return result;
}
```

第二步，使用`clang`将AST给dump出来：`~/llvm/bin/clang -Xclang -ast-dump -fsyntax-only test.c`

样例输出：

```
➜  example ../build/bin/clang -Xclang -ast-dump -fsyntax-only test.c
TranslationUnitDecl 0x60ddea0 <<invalid sloc>>
|-TypedefDecl 0x60de380 <<invalid sloc>> __int128_t '__int128'
|-TypedefDecl 0x60de3e0 <<invalid sloc>> __uint128_t 'unsigned __int128'
|-TypedefDecl 0x60de730 <<invalid sloc>> __builtin_va_list '__va_list_tag [1]'
`-FunctionDecl 0x60de850 <test.c:1:1, line:4:1> f 'int (int)'
  |-ParmVarDecl 0x60de790 <line:1:7, col:11> x 'int'
  `-CompoundStmt 0x60dea88 <col:14, line:4:1>
    |-DeclStmt 0x60dea10 <line:2:3, col:24>
    | `-VarDecl 0x60de910 <col:3, col:23> result 'int'
    |   `-ParenExpr 0x60de9f0 <col:16, col:23> 'int'
    |     `-BinaryOperator 0x60de9c8 <col:17, col:21> 'int' '/'
    |       |-ImplicitCastExpr 0x60de9b0 <col:17> 'int' <LValueToRValue>
    |       | `-DeclRefExpr 0x60de968 <col:17> 'int' lvalue ParmVar 0x60de790 'x' 'int'
    |       `-IntegerLiteral 0x60de990 <col:21> 'int' 42
    `-ReturnStmt 0x60dea68 <line:3:3, col:10>
      `-ImplicitCastExpr 0x60dea50 <col:10> 'int' <LValueToRValue>
        `-DeclRefExpr 0x60dea28 <col:10> 'int' lvalue Var 0x60de910 'result' 'int'
```

使用GDB调试Clang

由于我们是使用GNU的*gcc*和*g++*编译生成的*Clang*，这意味着需要使用gdb来对*Clang*进行跟踪调试。当然，如果你使用的是LLVM编译生成*Clang*，对应的调试工具将是*lldb*。

调试*Clang*的典型流程：

1. 打开gdb：`gdb`
2. 打开要调试的*clang*可执行文件，通过*file*命令：`file ~/llvm/bin/clang`，将会有`Reading symbols from /home/clang1/llvm/bin/clang...done`字样的输出。
3. 设定调试子进程，因为*Clang*会派生一个新进程来执行编译流程，命令：`set follow-fork child`
4. 在处理Pragma的入口函数处打断点：`b clang::PragmaNamespace::HandlePragma`
5. 执行编译：`r example.c`，其中*example.c*是传递给*Clang*的参数
6. 进行正常调试

常见命令介绍：

`r args`，运行已经加载的应用，*args*是传递给应用的参数。

`l`，列出代码

`b`，在指定位置打断点

`c`，继续执行程序