Using LLVM to analyze program

陈奕熹

March 20, 2020

1 Abstract

There are so many ways to analyze program flow, and such analysis is crucial for diving into a new project. Previously I had used some relative tool for analyze a C program (for tracing a QEMU-related extention)[1]. And indeed, having a overlook with function call graph or even thread passing timing is a huge stand point for a beginner.

In this lab, we will use LLVM pass to customize a function call graph tool.

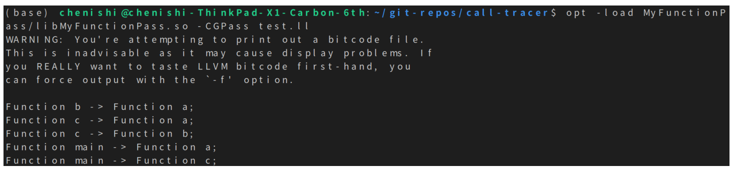
2 LLVM

Just to be clear, LLVM do have its own call-graph tool, and not-surprisingly easy to use[2]. So if not needed (for example, as a assignment), we don't need to rebuild the wheel.

However, if we somehow have to implement the tool on ourselves. We can tell that LLVM developers are clearly satisfied with every documents generated with Doxygen, just like the (in)famous QEMU.

And finally, thank god, I find a relatively new document[3] on how to use LLVM pass to dump call graph (also I do find that "using LLVM Pass to generate call path" is assignment in Purdue University too[4]).

[1]https://hackmd.io/ddmGI8uSTZGn7s4lviH9_Q
[2]https://www.ics.usi.ch/images/stories/ICS/slides/llvm-graphs.pdf
[3]https://programmer.help/blogs/llvm-call-graph-and-control-flow-graph.html
[4]https://www.cs.purdue.edu/homes/xyzhang/spring17/cs510-llvm-S17.pdf



```
(base) chenishi@chenishi-ThinkPad-X1-Carbon-6th:~/git-repos/call-tracer$ opt -load MyFunctionP
ass/libMyFunctionPass.so -CGPass test.ll
WARNING: You're attempting to print out a bitcode file.
This is inadvisable as it may cause display problems. If
you REALLY want to taste LLVM bitcode first-hand, you
can force output with the `-f' option.

Function b -> Function a;
Function c -> Function a;
Function c -> Function b;
Function main -> Function a;
Function main -> Function c;
```

Figure 1: Text-based call graph

3 Conclusion

LLVM pass is not the only way to get function call graph, just as written in abstract, there are still some framework (like gcc). LLVM pass strike a tricky balance between availability and abstractness. However, the mattermost question lingers, which is the coverage. Can it handle huge-scale project like QEMU, or sometime even with assembly language, weird communication (thread/process/RPC ......etc). After all, there is no need for a simple project to use such tool.