LLD

The new new ELF linker

Davide Italiano   Rafael Ávila de Espíndola

BSDCan 2016

outline

- motivation
- history
- differences from other ELF linker
- cool new stuff for freebsd
- implementation
- plan for integration and future work

motivation

```
% ld --version
GNU ld 2.17.50 [FreeBSD] 2007-07-03
Copyright 2007 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License.  This program has absolutely no warranty.
```

Goals

- Simple and fast
- Close to feature parity with GNU linkers
- Different when appropriate
- Part of llvm. Same license
- Possibility to introduce features across all parts of the toolchain
- Make linking great again

history

- ld.bfd got ELF suffort in 1993. Supports multiple formats
- ld.bfd is about 75K lines of C and script
- gold started in may 2006, added in Aug 2006
- gold is 160k lines of C++. Doesn't use bfd
- gold showed a fast ELF linker is possible

lld history

- old new ELF linker started in Jul 2012
- new new COFF linker stated May 2015
- new new ELF linker started Jul 2015

atom model

- Atom is an indivisible chunk of code or data
- Sections are split into atoms, normally via symbols
- Generic. Needed for MachO
- All work done on a graph of atoms
- On ELF, they have to be merged back
- COMDAT doesn't quite fit this model

just sections

- On ELF, a section is the atom
- It is explicit in the format, with any number of symbols
- COMDAT is implemented by just not reading some sections
- gc-sections is directly implemented
- linker structure matches ELF spec

differences from other ELF linker

- Library files just provide “lazy” symbols
- Resolution doesn’t have exactly the same semantics
- But matches COFF, and is easier and faster (no groups)
- mmap everything. Don’t use a 32 bit host
- Always a cross linker
- Only add feature if needed. Always allows undef in .so for example

cool new stuff

- AArch64 support
- Identical Code Folding (unsafe mode only)
- LTO
- Supports new ABIs (-mtls-dialect=gnu2)
- Linker optimizations (SHF_MERGE, .eh_frame, relocations)
- Fast!
- Maintained

Link Time Optimizations

- Increase the scope of optimizations to (almost) the whole executable
- The compiler emits bitcode instead of object files
- The linker reads the bitcode files back, resolving symbols
- Bitcode files and object files can be mixed transparently
- All the bitcode files are merged in a single file on which optimizations are applied
- IPO (e.g. inlining/constant folding) is more effective as symbols visible only from bitcode are internalized

Identical code folding

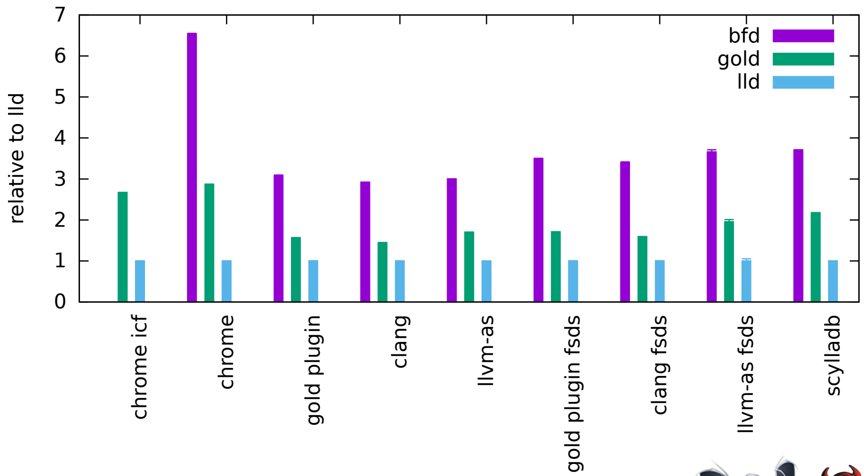
- C++ libraries have functions with identical code
- ICF merges these function in a single copy
- (Might) change the semantic of code
- Safe variant (not yet implemented) guarantee that only functions which address is not used for comparison are folded.

Other linker optimizations

- Contents of SHF_MERGE sections are uniqued
- –gc-sections removes dead SHF_MERGE entries too
- Strings are tail merged at -O2
- .eh_frame is split, CIEs uniqued
- We create a .eh_frame_hdr
- Relocations are relaxed:

```
addq a@GOTTPOFF(%rip), %rax →
leaq a@TPOFF(%rax), %rax
movq foo@GOTPCREL(%rip), %rax →
leaq foo(%rip), %rax
```

Fast

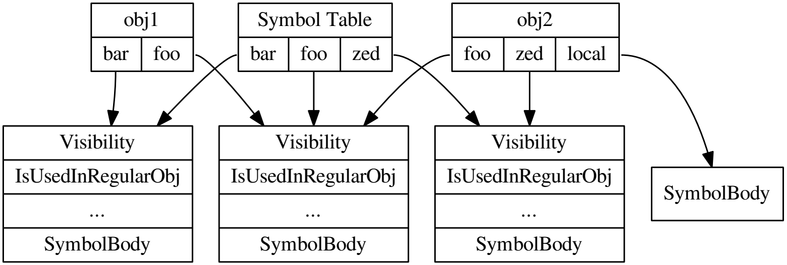


relative to lld
0
1
2
3
4
5
6
7
bfd
gold
lld
chrome icf
chrome
gold plugin
clang
llvm-as
gold plugin fsds
clang fsds
llvm-as fsds
scylladb

Implementation

- Currently "just" 13k lines
- Archives can go in any order (drop tsort?)
- We create InputSections for each section in the .o
- Special treatment done by specialized classes: MergeInputSection, EhInputSection.
- Duplicated comdats are not read, symbols in them become undefined
- For the first part, LTO objects just provide symbols like ELF

Symbol Table



layout

- We now know all sections and symbols we need
- We can gc unused sections (and part of merge sections)
- We can merge identical sections (icf)
- Remaining input sections are concatenated
- Linker creates other sections (symbol table for example)

relocation processing

- Lots of relocations, so has to be fast
- Cannot be done in one pass for default layout
- Target just maps relocation type to an expression enum
- Target independent logic decides if we need a GOT, PLT, dynamic reloc, etc
- Save the computed expression to simplify the relocation application on a second pass

future lld work

- Version script
- Linker script (buildworld + buildkernel should work)
- Section layout optimization
- Split dwarf (.dwo)
- gdb index
- special case for R_X86_64_RELATIVE
- direct binding

integrating in freebsd

- No RW .text support (fixed in FreeBSD)
- Remove dependency on legacy –oformat/-Y (FreeBSD)
- Handle library search differences (FreeBSD)
- Implement -Ttext (lld)
- Implement version scripts (lld)
- Implement linker scripts (lld)
- Testing! (both)