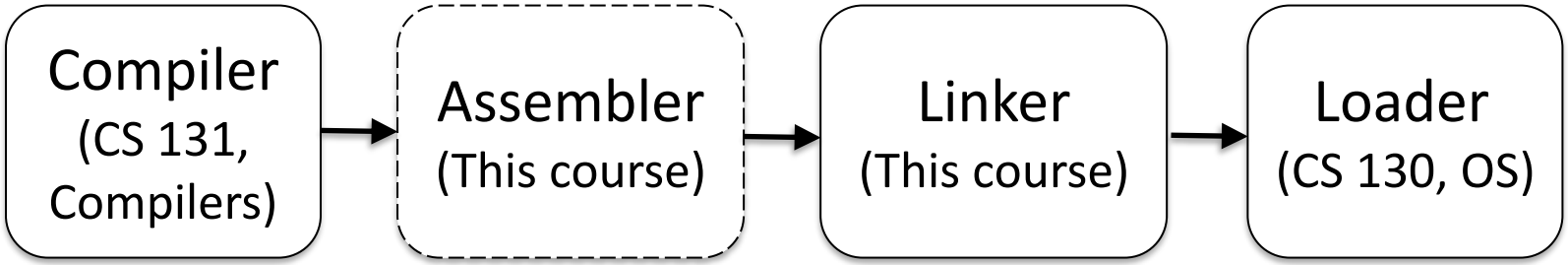
Discussion 4 CALL

Chibin Zhang

What is CALL?

- E: Editor
- C: Compiler
- A: Assembler
- L: Linker
- L: Loader
- Sometimes things get messed up when you're using an IDE…

CALL Pipeline



CALL

- C: Compiler
- A: Assembler
- L: Linker
- L: Loader

Compiler

- Translates high-level programming language into assembly (Strictly speaking)
- Input: .c, .cpp, .rs, etc.
- Output: .s
- Pipeline:
  - Lexer
  - Parser
  - Semantics
  - Optimization
  - Code Generation

Compiler (in practice)

- `clang test.c` -> `a.out`
- Directly producing executable instead of assembly!
- Contradiction?
- In this process, clang is the **compiler driver**, which connects the entire compilation process in an automated way.

Demo

- `clang test.c --verbose`
- "/usr/bin/clang-11" -cc1 -triple x86_64-pc-linux-gnu …
- "/usr/sbin/ld" … -o a.out … /crt1.o …
- -cc1 for invoking the C frontend. (cc1plus, cc1obj)
- Implicitly invoking the system linker `ld` to link against `crt1.o`(more on this later)

CALL

- C: Compiler
- A: Assembler
- L: Linker
- L: Loader

Assembler

- Project 1.1
- Input: .s, .S
  - Capitalized to signify **hand written** assembly)
  - Text format (ASCII/UTF-8)
- Output: .o
  - Binary encoded
  - Actually an ELF file in Linux (But not runnable)
  - Use readelf/objdump to parse each segment

Assembler (continued)

- Expand pseudo instruction (if any)
- Process directives
  - Some are architecture specific (.thumb for ARM32)
  - Some are assembler specific (.macro for GNU as)
  - Usually assembly written for one assembler won't be compatible with other assemblers
- **Primary job:** Encode human readable text into machine code (binary)
- Relocation (.rela), Symbol table (.symtab)

Relocation Demo

- `clang –c –Os –o test.o test.c`
- `objdump –dr test.o` / `readelf –r test.o`

```
test.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <fib>:
   0:   55                      push   %rbp
   1:   53                      push   %rbx
   2:   50                      push   %rax
   3:   bb 01 00 00 00          mov    $0x1,%ebx
   8:   83 ff 01                cmp    $0x1,%edi
   b:   74 16                   je     23 <fib+0x23>
   d:   89 fd                   mov    %edi,%ebp
   f:   ff cd                   dec    %ebp
  11:   31 db                   xor    %ebx,%ebx
  13:   89 ef                   mov    %ebp,%edi
  15:   e8 00 00 00 00          call   1a <fib+0x1a>
                        16: R_X86_64_PLT32      fib-0x4
```

CALL

- C: Compiler
- A: Assembler
- L: Linker
- L: Loader

Linker

- Static linking
- Concatenate .text and .data section of each .o file
- Resolve relocation
- Runtime/dynamic linking
  - .so files
  - GOT (Global Offset Table), PLT (Procedure Linkage Table)
  - Beyond the scope of this discussion

Demo

- Manually linking to system library by hand is too tedious…
- `clang --verbose test.o`

0x1156 + 0xffffffe6 (-26)
= 0x113c

```
000000000000113c <fib>:
    113c:       55                      push   %rbp
    113d:       53                      push   %rbx
    113e:       50                      push   %rax
    113f:       bb 01 00 00 00          mov    $0x1,%ebx
    1144:       83 ff 01                cmp    $0x1,%edi
    1147:       74 16                   je     115f <fib+0x23>
    1149:       89 fd                   mov    %edi,%ebp
    114b:       ff cd                   dec    %ebp
    114d:       31 db                   xor    %ebx,%ebx
    114f:       89 ef                   mov    %ebp,%edi
    1151:       e8 e6 ff ff ff          call   113c <fib>
```

Previously zero

CALL

- C: Compiler
- A: Assembler
- L: Linker
- L: Loader

Loader

- Resource allocation
  - Spawn new thread
  - Page table (address space)
- Program state initialization
  - Pass in `argc`, `argv` onto the stack
  - Map ELF file to into memory
  - Clear interrupt
  - Setup registers (Stack pointer)
  - Jump to program entry (Not necessarily main)
- Discussed in detail in CS130 Operating Systems.

No more demos :)

- ELF loading in the Linux kernel
  - `load_elf_binary` in `fs/binfmt_elf.c`

```
1141                error = elf_map(bprm->file, load_bias + vaddr, elf_ppnt,
1142                                elf_prot, elf_flags, total_size);
```

- Passing arguments onto the stack
  - `transfer_args_to_stack` in `fs/exec.c`

```
for (index = MAX_ARG_PAGES - 1; index >= stop; index--) {
        unsigned int offset = index == stop ? bprm->p & ~PAGE_MASK : 0;
        char *src = kmap(bprm->page[index]) + offset;
        sp -= PAGE_SIZE - offset;
        if (copy_to_user((void *) sp, src, PAGE_SIZE - offset) != 0)
                ret = -EFAULT;
        kunmap(bprm->page[index]);
        if (ret)
                goto out;
}
```