可执行文件及链接

什么是可执行文件：

如图中所说：可执行文件就是一个描述了状态机的初始状态和状态机的转移的文件。

是什么决定了某个文件是否是可执行文件？

回答：execve决定的。

回忆一下：可执行文件是是作为参数传给execve的，是因为execve返回了-1，EACCES才导致无法执行。若强行赋予操作权限，那么execve会返回：ENOEXEC错误。

调试信息

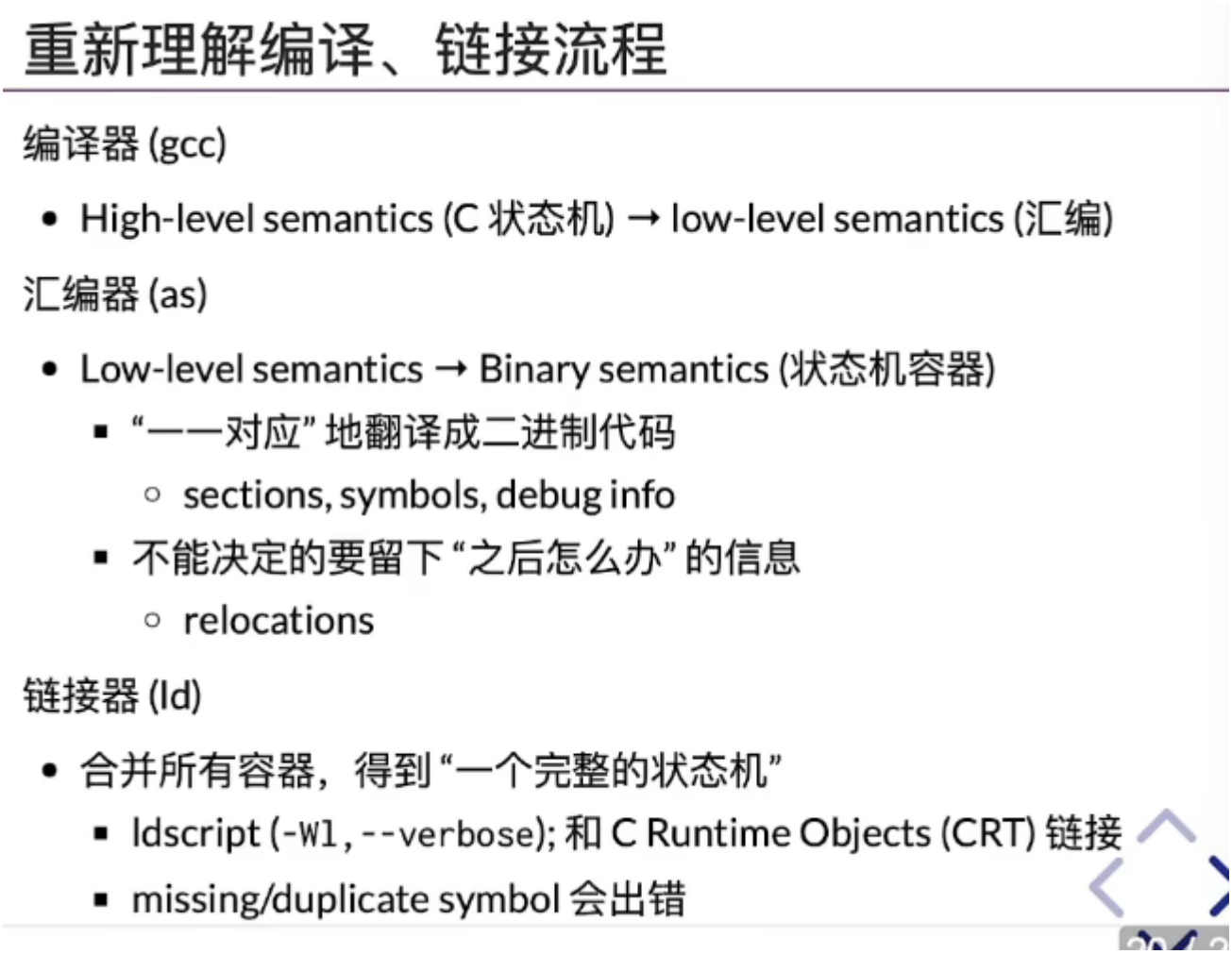
为什么gdb能把错误的backtrace打印出来？

编译器在编译的时候，会把一些调试信息放到可执行文件中，这些调试信息包括：函数名、变量名、行号等等。它定义了一个Turing Complete的语言，这个语言可以描述函数的调用关系，变量的作用域等等。这个语言就是DWARF。debug_info能够把汇编的状态转回为C语言。但是有很多bug，而且在加上编译器的优化后，这种错误会更多。

还可以自己写，只需要把函数栈帧的调用关系用链表链起来就ok了。

从 C 代码到二进制文件

经过gcc编译器把c语言的状态机转换为汇编语言的状态机，然后汇编器再把汇编语言转换为汇编器+一些约束条件，最后是连接器把若干个机器语言的描述转化为可执行文件。



ELF需要满足重定向的需求，然后还需要使得名字空间占用小。

ELF loader

前面提到，可执行文件就是一个描述了状态机的初始状态和迁移的数据结构。不过指针都被偏移量代替了。同时elf文件的头部有一些关于arch的信息，这些信息是用来检查是否是当前arch的。

加载器用来解析数据结构+复制内存+加跳转的。（我们可以用mmap实现loader，以达到未使用execve但是却达到同样的效果的目的）

在操作系统上实现 ELF Loader

可执行文件

- 一个描述了状态机的初始状态 (迁移) 的 数据结构
  - 不同于内存里的数据结构，“指针” 都被 “偏移量” 代替
  - 数据结构各个部分定义：`/usr/include/elf.h`

加载器 (loader)

- 解析数据结构 + 复制到内存 + 跳转
- 创建进程运行时初始状态 (argv, envp, ...)
  - loader-static.c
    - 可以加载任何静态链接的代码 minimal.S, dfs-fork.c
    - 并且能正确处理参数/环境变量 env.c
  - RTFM: System V ABI Figure 3.9 (Initial Process Stack)

```
(gdb) !pmap 19084
19084:   /home/jyy/Projects/os-demos/tmp/loader minimal
0000000000400000      4K r---- minimal
0000000000401000      4K r-x-- minimal
0000555555554000      4K r---- loader
0000555555555000      4K r-x-- loader
0000555555556000      4K r---- loader
0000555555557000      4K r---- loader
0000555555558000      4K rw--- loader
0000555555559000   1024K rw---   [ anon ]
00007ffff7dc1000    136K r---- libc-2.31.so
00007ffff7de3000   1504K r-x-- libc-2.31.so
00007ffff7f5b000    312K r---- libc-2.31.so
00007ffff7fa9000     16K r---- libc-2.31.so
00007ffff7fad000      8K rw--- libc-2.31.so
00007ffff7faf000     24K rw---   [ anon ]
00007ffff7fcb000     12K r----   [ anon ]
00007ffff7fce000      4K r-x--   [ anon ]
00007ffff7fcf000      4K r---- ld-2.31.so
00007ffff7fd0000    140K r-x-- ld-2.31.so
00007ffff7ff3000     32K r---- ld-2.31.so
00007ffff7ffb000      4K r---- minimal
00007ffff7ffc000      4K r---- ld-2.31.so
00007ffff7ffd000      4K rw--- ld-2.31.so
00007ffff7ffe000      4K rw---   [ anon ]
00007ffffffde000    132K rw---   [ stack ]
ffffffffff600000      4K --x--   [ anon ]
 total             3396K
(END)
```

先把elf文件里的所有数据结构都放到内存，然后再构建运行时的初始状态（按照手册的规定来重置寄存器，嵌入汇编即可）。按照System V ABI Figure 3.9（Initial Process Stack）规定。