

# 动态链接器实验报告

2193211079 王嘉禾 计算机试验班 001

## 实验目标（前三个测试点）

1. **建立内存映射**：将共享库机器依赖从硬盘加载进内存
2. **完成符号解析**：对 libc 的伪加载
3. **完成初始化**：对本地符号进行重定位和调用初始化函数

## 测试点一：建立内存映射

### 实验步骤（附一些关键代码片段）

1. 读取 ELF 文件，找到 ELF header（总在 ELF 文件的头部）

```
1. int fd=open(libpath,O_RDONLY);
2. Elf64_Ehdr ehdr;
3. pread(fd,&ehdr,sizeof(Elf64_Ehdr),0);
```

2. 通过 ELF header，找到 program header table 的位置（体现为一个偏移量）和 entry 的数目，读取整个 program header table

```
pread(fd,&phdr,size_phdr*ehdr.e_phnum,ehdr.e_phoff);
```

3. 找到第一个和最后一个类型为 PT\_LOAD 的 segment，计算所有需要加载的 segment 的空间总和，便于预留足够的空间

```
1. size_t i=0;
2. while (i<ehdr.e_phnum && phdr[i].p_type!=PT_LOAD) i++;
3. Elf64_Phdr* first_segment = &phdr[i];
4. Elf64_Phdr* last_segment = &phdr[ehdr.e_phnum-1];
5. while (last_segment>first_segment && last_segment->p_type!=PT_LOAD) last_segment--;
6. size_t span=last_segment->p_vaddr+last_segment->p_memsz-first_segment->p_vaddr;
```

4. 加载第一个 segment（此时不需要指定起始地址），并记下加载到的地址填入 LinkMap 中，并且，求出一个实际的加载地址与段首地址的偏移量，用于后续计算

```
1. uintptr_t start_addr=(uintptr_t)mmap(NULL,ALIGN_UP(span,size_page),prot_from_phdr(first_segment),MAP_FILE|MAP_PRIVATE,fd,ALIGN_DOWN(first_segment->p_offset,size_page));
2. const Elf64_Addr load_bias=start_addr-ALIGN_DOWN(first_segment->p_vaddr,size_page);
3. lib->addr=(uint64_t)start_addr;
```

5. 加载剩余的 PT\_LOAD segment

```
1. for(Elf64_Phdr* ph=first_segment+1;ph<=last_segment;++ph)
2. {
3.     if(ph->p_type==PT_LOAD)
4.     {
5.         Elf64_Addr start=ALIGN_DOWN(ph->p_vaddr+load_bias,size_page);
6.         Elf64_Addr end=ALIGN_UP(ph->p_vaddr+load_bias+ph->p_memsz,size_page);
```

```

7.         mmap((void*)start,end-start,prot_from_phdr(ph),MAP_FILE|MAP
_PRIVATE|MAP_FIXED,fd,ALIGN_DOWN(ph->p_offset,size_page));
8.     }
9. }

```

6. 遍历所有的 segment，找到类型为 DT\_DYNAMIC 的 segment，将其绝对地址存入 lib

```

1. Elf64_Dyn* dynamic;
2. for (i=0;i<ehdr.e_phnum;++i)
3. {
4.     if(phdr[i].p_type==PT_DYNAMIC) dynamic=(Elf64_Dyn*)(load_bias+phdr[
i].p_vaddr);
5. }
6. lib->dyn=dynamic;

```

运行结果（包含测试点通过和分数的截图）

由于第一个测试点会检测到第二个测试点中的 `dlopen()` 和 `dlsym()` 函数而使成绩无效，因此第一个测试点与第二三个测试点的分开测试得到成绩，实际共跑出 90 分

```

Test name: one global data relocation
SIGSEGV received in custom loader. Maybe you want to debug it with gdb.
-----
# Evaluating TestCase $5 #
#
#
Test name: one 2-layer relocation
SIGSEGV received in custom loader. Maybe you want to debug it with gdb.
-----
# Evaluating TestCase $6 #
#
#
Test name: lazy binding
SIGSEGV received in custom loader. Maybe you want to debug it with gdb.
-----
Your Score: 80 / 100
root@caf-virtual-machine: /home/caf/OS/COMP#

```

```

Your Score: 0 / 100
root@caf-virtual-machine: /home/caf/OS/COMP# make
gcc -g -shared -fPIC -fcf-protection=none -fno-built-in src/OpenLibrary.c src/MapLibrary.c src/RelocLibrary.c src/FindSymbol.c src/RuntimeResolve.c src/trampoline.S src/InitLibrary.c util/shim.c -o build/loader.so -ldl
root@caf-virtual-machine: /home/caf/OS/COMP# python3 autograder.py
-----
# Evaluating TestCase $0 #
#
#
Test name: zero relocation
Expected output: 6
Your output: 6
Passed!
-----
# Evaluating TestCase $1 #
#
#

```

## 测试点二：对 libc 的伪加载

### 实验步骤（附一些关键代码片段）

1. 通过 `readelf` 找到 `.rela.plt` section 中需要加载的函数为 `puts()`，使用 `dlopen` 和 `dlsym` 找到函数的位置

```
Relocation section '.rela.plt' at offset 0x510 contains 1 entry:
  Offset             Info             Type             Sym. Value          Sym. Name + Addend
0000000004018      0002000000007  R_X86_64_JUMP_SLO 0000000000000000  puts@GLIBC_2.2.5 + 0
root@caf-virtual-machine:/home/caf/OS/COMP#
```

1. `void* handle=dlopen("libc.so.6",RTLD_LAZY);`
2. `void* address=dlsym(handle,"puts");`
2. 从 LinkMap 中取出 dynamic section 的入口，找到其中 tag 为 DT\_JMPREL 的 entry
  1. `Elf64_Dyn* dynamic=lib->dyn;`
  2. `while (dynamic->d_tag!=DT_NULL && dynamic->d_tag!=DT_JMPREL) dynamic++;`
  3. `if (dynamic->d_tag==DT_NULL) return;`
3. 这个 entry 中存储了对应的重定位条目的地址，找到该重定位条目，就可以从 `r_offset` 字段得知这个条目的地址应当填到哪里，在该位置填入刚刚找到的答案
  1. `Elf64_Rela* rela=(Elf64_Rela*)(dynamic->d_un.d_val);`
  2. `uint64_t* addr=(uint64_t*)(lib->addr+rela->r_offset);`
  3. `*addr=(uint64_t)address;`

运行结果截图在第三个测试点之后

## 测试点三：完成初始化

### 实验步骤（附一些关键代码片段）

1. 从 LinkMap 事先存储的 dyn 指针找到 dynamic section 的地址，遍历整个 section 中的所有 entry，按照以下对应关系从 `d_un.d_val` 字段取出需要的信息

d_tag	对应信息
DT_RELA	.rela.dyn section 的起点
DT_RELACOUNT	本地符号的数量，对应需要重定位的次数
DT_INIT	指向一个初始化函数
DT_INIT_ARRAY	指向一个初始化函数指针的数组
DT_INIT_ARRAYSZ	数组的大小，除以函数指针的大小可得数组中函数指针的数目

```
1. Elf64_Dyn* dynamic=lib->dyn;
2. Elf64_Rela* rela;
3. Elf64_Sword tag;
4. int size_section,size_entry;
5. int localnum;
6. void (*initfunc)();
7. void** initlist;
8. int funcnum;
9.
10. while (dynamic->d_tag!=DT_NULL)
11. {
12.     dynamic++;
13.     tag=dynamic->d_tag;
14.     if(tag==DT_RELA)
```

```

15.     {
16.         rela=(Elf64_Rela*)(dynamic->d_un.d_val);
17.     }
18.     else if(tag==DT_RELACOUNT)
19.     {
20.         localnum=(int)(dynamic->d_un.d_val);
21.     }
22.     else if(tag==DT_INIT)
23.     {
24.         initfunc=(void*)(dynamic->d_un.d_val);
25.     }
26.     else if(tag==DT_INIT_ARRAY)
27.     {
28.         initlist=(void*)(dynamic->d_un.d_val);
29.     }
30.     else if(tag==DT_INIT_ARRAYSZ)
31.     {
32.         funcnum=(int)(dynamic->d_un.d_val);
33.     }
34. }

```

2. 从 `.rela.dyn` section 的起点开始，按照本地符号的数量，处理所有的本地符号重定位，具体操作时：每次将整个 library 的起点 `addr` (在测试点一中存储) 加上重定位条目的 `r_addend` 的字段值，得到本地符号的重定位地址，填入重定位条目的 `r_offset` 指向的地址，与测试点二中的操作类似

```

1. for(int i=0;i<localnum;++i)
2. {
3.     uint64_t* addr;
4.     addr=(uint64_t*)(lib->addr+rela->r_offset);
5.     *addr=(uint64_t)(lib->addr+(uint64_t)rela->r_addend);
6.     rela++;
7. }

```

3. 调用所有的初始化函数，首先调用从 `DT_INIT` 对应的 `entry` 读出的函数，然后遍历 `DT_INIT_ARRAY` 对应的 `entry` 读出的指针对应的函数指针数组中所有的函数，这里已经读出了函数指针数组的大小（在 `DT_INIT_ARRAYSZ` 对应的 `entry` 中），是一个字节数，在 64 位计算机中函数指针占 8 个字节，因此将这个值除以 8 就可以得到数组中函数指针的个数

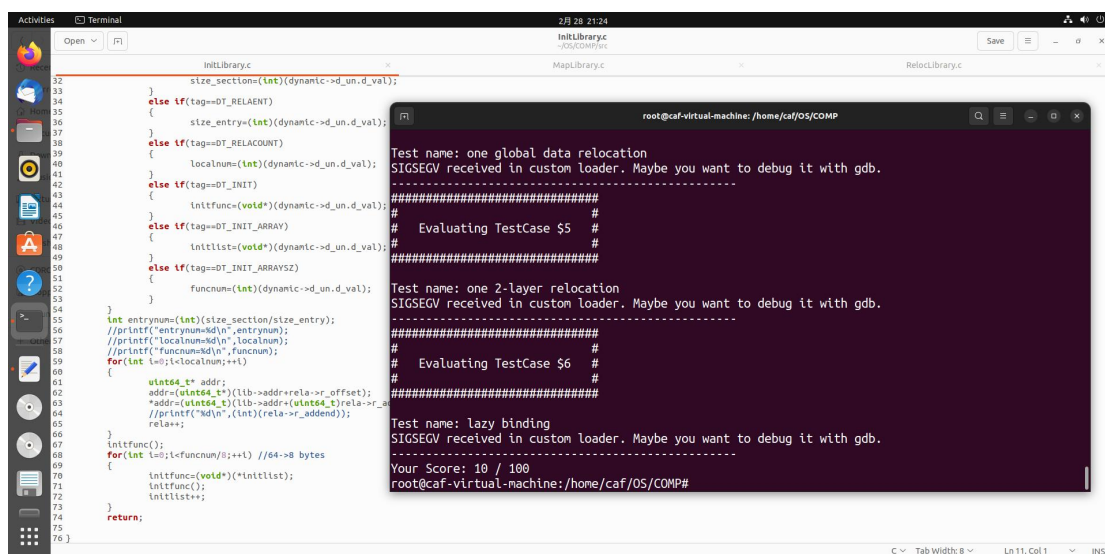
```

1. initfunc();
2. for(int i=0;i<funcnum/8;++i) //64->8 bytes
3. {
4.     initfunc=(void*)(*initlist);
5.     initfunc();
6.     initlist++;
7. }

```

运行结果（包含测试点通过和分数的截图）

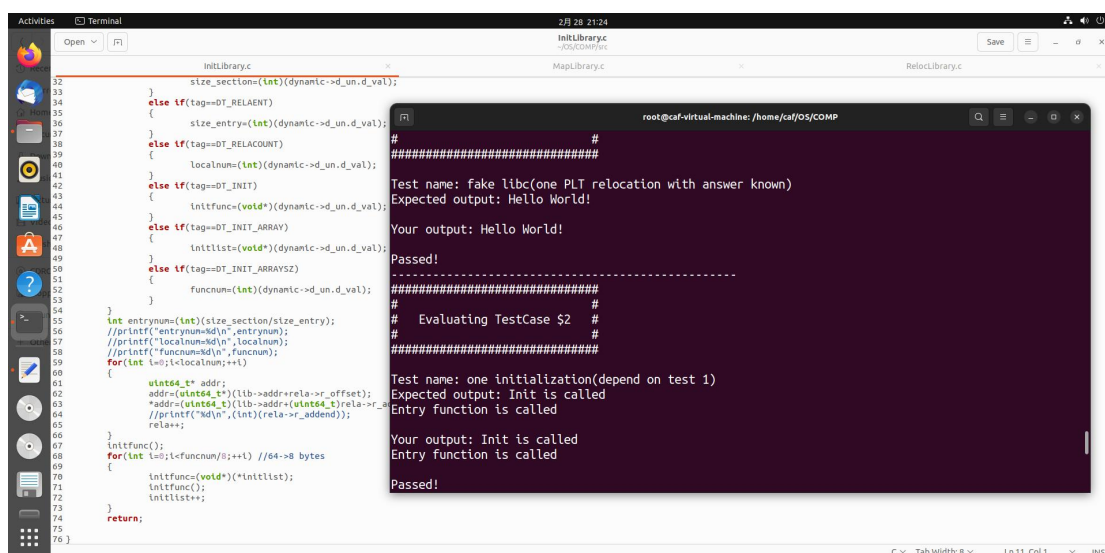
由于第一个测试点会检测到第二个测试点中的 `dlopen()` 和 `dlsym()` 函数而使成绩无效，因此第一个测试点与第二三个测试点的分开测试得到成绩，实际共跑出 90 分



The screenshot shows a C code editor with the file `InitLibrary.c` and a terminal window. The code in `InitLibrary.c` defines a library with various sections and functions. The terminal window shows the output of a test runner, including test names like "one global data relocation", "one 2-layer relocation", and "lazy binding", all of which resulted in a SIGSEGV error. The final score shown is 10 / 100.

```
InitLibrary.c
32 size_section=(int)(dynamic->d_un.d_val);
33
34 else if(tag==DT_RELAENT)
35 {
36     size_entry=(int)(dynamic->d_un.d_val);
37
38     else if(tag==DT_RELACOUNT)
39     {
40         localnum=(int)(dynamic->d_un.d_val);
41
42     else if(tag==DT_INIT)
43     {
44         initfunc=(void*)(dynamic->d_un.d_val);
45
46     else if(tag==DT_INIT_ARRAY)
47     {
48         initlist=(void*)(dynamic->d_un.d_val);
49
50     else if(tag==DT_INIT_ARRAYSZ)
51     {
52         funcnum=(int)(dynamic->d_un.d_val);
53
54 }
55 int entrynum=(int)(size_section/size_entry);
56 //printf("entrynum=%d\n",entrynum);
57 //printf("localnum=%d\n",localnum);
58 //printf("funcnum=%d\n",funcnum);
59 for(int i=0;i<localnum;i++)
60 {
61     uint64_t* addr;
62     addr=(uint64_t*)(lib->addr+rela->r_offset);
63     *addr=(uint64_t)(lib->addr+(uint64_t)rela->r_addend);
64     //printf("%d\n", (int)(rela->r_addend));
65     rela++;
66 }
67 initfunc();
68 for(int i=0;i<funcnum/i;++) //64->8 bytes
69 {
70     initfunc=(void*)(*initlist);
71     initfunc();
72     initlist++;
73 }
74 return;
75
76 }
```

```
Test name: one global data relocation
SIGSEGV received in custom loader. Maybe you want to debug it with gdb.
#####
# Evaluating TestCase $5 #
#####
Test name: one 2-layer relocation
SIGSEGV received in custom loader. Maybe you want to debug it with gdb.
#####
# Evaluating TestCase $6 #
#####
Test name: lazy binding
SIGSEGV received in custom loader. Maybe you want to debug it with gdb.
Your Score: 10 / 100
root@caf-virtual-machine: /home/caf/OS/COMP#
```



The screenshot shows a C code editor with the file `InitLibrary.c` and a terminal window. The code in `InitLibrary.c` defines a library with various sections and functions. The terminal window shows the output of a test runner, including test names like "fake libc(one PLT relocation with answer known)", "one initialization(depend on test 1)", and "one 2-layer relocation", all of which passed. The final score shown is 10 / 100.

```
InitLibrary.c
32 size_section=(int)(dynamic->d_un.d_val);
33
34 else if(tag==DT_RELAENT)
35 {
36     size_entry=(int)(dynamic->d_un.d_val);
37
38     else if(tag==DT_RELACOUNT)
39     {
40         localnum=(int)(dynamic->d_un.d_val);
41
42     else if(tag==DT_INIT)
43     {
44         initfunc=(void*)(dynamic->d_un.d_val);
45
46     else if(tag==DT_INIT_ARRAY)
47     {
48         initlist=(void*)(dynamic->d_un.d_val);
49
50     else if(tag==DT_INIT_ARRAYSZ)
51     {
52         funcnum=(int)(dynamic->d_un.d_val);
53
54 }
55 int entrynum=(int)(size_section/size_entry);
56 //printf("entrynum=%d\n",entrynum);
57 //printf("localnum=%d\n",localnum);
58 //printf("funcnum=%d\n",funcnum);
59 for(int i=0;i<localnum;i++)
60 {
61     uint64_t* addr;
62     addr=(uint64_t*)(lib->addr+rela->r_offset);
63     *addr=(uint64_t)(lib->addr+(uint64_t)rela->r_addend);
64     //printf("%d\n", (int)(rela->r_addend));
65     rela++;
66 }
67 initfunc();
68 for(int i=0;i<funcnum/i;++) //64->8 bytes
69 {
70     initfunc=(void*)(*initlist);
71     initfunc();
72     initlist++;
73 }
74 return;
75
76 }
```

```
#####
# Evaluating TestCase $1 #
#####
Test name: fake libc(one PLT relocation with answer known)
Expected output: Hello World!
Your output: Hello World!
Passed!
#####
# Evaluating TestCase $2 #
#####
Test name: one initialization(depend on test 1)
Expected output: Init is called
Entry function is called
Your output: Init is called
Entry function is called
Passed!
#####
# Evaluating TestCase $3 #
#####
Test name: one 2-layer relocation
SIGSEGV received in custom loader. Maybe you want to debug it with gdb.
Your Score: 10 / 100
root@caf-virtual-machine: /home/caf/OS/COMP#
```

## 实验感想

本次实验中遇到的困难主要来源于

- (1) 理论细节掌握的不足。
- (2) 对于实验中相关数据结构的了解程度不够，实验涉及的几种存储结构花了一定时间才完全理解。
- (3) 对于频繁的指针和类型转换交叉操作的不熟练，在测试点一中，从理清 ELF 文件的存储结构到实现辗转读取所需信息花费了很多时间查阅资料。

此外，在测试点一中，曾遇到很多的 SIGSEGV 的问题，调试了很长时间，最终的 debug 结果表明，这些问题 90%来自于读取空间的溢出和指针类型的不匹配。在测试点二三中，主要障碍来自于对数据存储结构的理解以及不知道怎么下手分解和读取相应的数据结构类型（在测试点一种也遇到类似问题），最终通过反复阅读实验指导书，自行查阅相关资料已经不断尝试和调试代码，都得到解决。通过本次实验我加强了对动态链接过程的理解以及实操的工程能力，获益匪浅。

