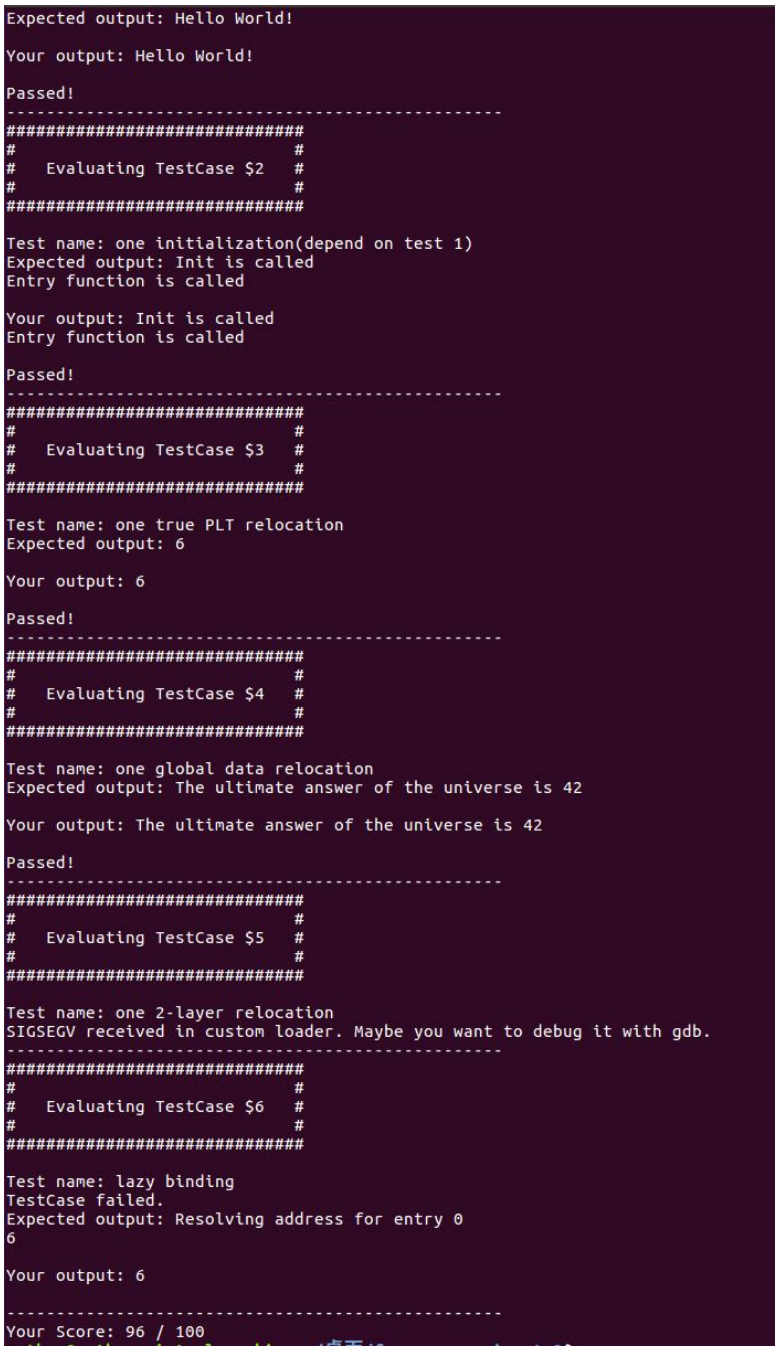
操作系统实验 2

计试 001 冯志远 2203312581

实验得分：



```
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 true PLT relocation
Expected output: 6

Your output: 6

Passed!
-----------------------------------------------
##############################
#                            #
#   Evaluating TestCase $4   #
#                            #
##############################

Test name: one global data relocation
Expected output: The ultimate answer of the universe is 42

Your output: The ultimate answer of the universe is 42

Passed!
-----------------------------------------------
##############################
#                            #
#   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
TestCase failed.
Expected output: Resolving address for entry 0
6

Your output: 6

-----------------------------------------------
Your Score: 96 / 100
```

实验目的：

按照实验指导，实现一个动态链接器，完成加载、重定位、初始化等操作。

实验结果：

Test 0

这个 test 的目的很简单，就是将一个 shared library 映射到内存中，test0 只要求映射 PL_load 段。但是只需要加载指定的段就可以，每个段的数据类型都是 ELF64_Phdr 的数据结构，里面的内容就是下图所示。

```
// elf.h
typedef struct
{
  Elf64_Word    p_type;                 /* Segment type */
  Elf64_Word    p_flags;                /* Segment flags */
  Elf64_Off       p_offset;             /* Segment file offset */
  Elf64_Addr    p_vaddr;                /* Segment virtual address */
  Elf64_Addr    p_paddr;                /* Segment physical address */
  Elf64_Xword   p_filesz;               /* Segment size in file */
  Elf64_Xword   p_memsz;                /* Segment size in memory */
  Elf64_Xword   p_align;                /* Segment alignment */
} Elf64_Phdr;
```

主要利用 fopen 函数打开文件，fseek 函数移动文件指针，fread 函数读取文件。需要做的，就是正确读取 ELF Header 的信息。里面包含内容格式如下。

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  ...
  Start of program headers:          64 (bytes into file)
  Start of section headers:          14720 (bytes into fil
  ...
```

读取上面的信息后，也就知道了每个段的信息。下面要做的就是利用 mmap 将 p_type 为 PT_load 段从文件映射到内存区。但在分配的时候需要注意，就是两个段直接必须有足够的间隔，因为存在内碎片的情况，不然很有可能第二个段在分配的时候分配到了第一个段的空间。说白了就是不能直接交给操作系统

来分配。通过搜索资料，找到一种解决方法就是可以通过申请一片较大的内存，然后记录首地址，然后清空，这样就获得了一片自由分配的连续大内存空间。

之后还需要注意的就是页对齐的情况，因为内存是按页为单位的，这部分代码在 md 都有说明。

核心代码如下

```
for (int i=0;i<pdr_num;i++)
{
    if (pdr[i]->p_type!=PT_LOAD && pdr[i]->p_type!=PT_DYNAMIC) continue;
    Elf64_Phdr *segm = pdr[i];
    if (pdr[i]->p_type == PT_DYNAMIC)
    {
        lib->dyn = (Elf64_Dyn *)(segm->p_vaddr + base_address);
        continue;
    }

    int prot = 0;
    prot |= (segm->p_flags & PF_R)? PROT_READ : 0;
    prot |= (segm->p_flags & PF_W)? PROT_WRITE : 0;
    prot |= (segm->p_flags & PF_X)? PROT_EXEC : 0;
    void *mmap_addr =(void*) ALIGN_DOWN(segm->p_vaddr+base_address, getpagesize());
    siz = ALIGN_UP(segm->p_vaddr+base_address+segm->p_memsz, getpagesize()) - ALIGN_DOWN(segm->p_vaddr+base_address, getpagesize());
    record_address =(uint64_t) mmap(mmap_addr, siz, prot, MAP_FILE | MAP_PRIVATE, fd, ALIGN_DOWN(segm->p_offset, getpagesize()));

    if (i==0) base_address=record_address;
    offset_address+=siz;

    //print("base_address = %d",base_address);
    // print(" i= %d",i);
}
```

Test 1

就是先找到需要重定位的表，并且求出重定位表的大小，还有符号表和字符串表。重定位表和符号表分别对应这需要重新定位的信息以及需要重新定位符号的信息，对于字符串表，也就是名称表，记录着名字。

实现的过程就是先找到重定位函数符号表，然后找到对应在字符串表中的偏移量，知道偏移量和起始地址，也就知道了具体位置，至于怎么获得函数的位置，可以用 dlopen 和 dlsm 函数。

核心部分代码如下：

```
int pos = relocation_table_addr->r_info>>32;
for (int i=0; i<pos; i++,sym_table_addr++);
int str_offset = sym_table_addr->st_name;
Elf64_Addr recod =(Elf64_Addr) (sadr + str_offset + relocation_table_addr->r_addend);
Elf64_Addr *renew =(Elf64_Addr *) (relocation_table_addr->r_offset + lib->addr);
*renew = recod;
void *handle = dlopen("libc.so.6", RTLD_LAZY);
void *real_address = dlsym(handle, (const char*)recod);
if (real_address == NULL)
    for (int i=0; i<10; i++)
    {
        real_address = dlsym(pt, (const char*)recod);
        if (real_address != NULL)
            break;
    }
*renew = (Elf64_Addr)real_address;
```

Test 2

这个 test 要实现的是初始化部分，初始化的执行是在 entry 之前的。初始化中，对于 rela 表进行重定位，我们只需要简单的将基地址和 r_addend 相加，最终的和填入即可。对于初始化函数的调用，我们需要用到两个初始化函数，DT_INIT 和 DT_INIT_ARRAYSZ ，两者分别对应的是系统的初始化和函数数组的初始化。当然，初始化肯定是执行顺序最早的。整体过程不复杂，实现代码非常简短。还有部分代码和 test4 相关，在 test4 一并给出。

```
int tag=siz/8;
while(tag--)
{
    void (*func)(void) =(void*) (*pt);
    if (func<(l->addr)) func+=(l->addr);
    func(),pt++;
}
```

Test 3

这个 test 的主要目的就是通过自己来解决文件之间的依赖性。就是在加载一个 library 的时候，把它的依赖项也加载进来。首先利用 dlopen 打开对应依赖库，如果"libc.so.6"库返回的 dlsym 为 NULL，则表示重定位的函数不 在 libc 中，而是对应的依赖库中，这是调用 dlsym 获得正确函数地址即可保证程序正常运行。核心代码如下

```
int pos = relocation_table_addr->r_info>>32;
for (int i=0; i<pos; i++,sym_table_addr++);
int str_offset = sym_table_addr->st_name;
Elf64_Addr recod =(Elf64_Addr) (sadr + str_offset + relocation_table_addr->r_addend);
Elf64_Addr *renew =(Elf64_Addr *) (relocation_table_addr->r_offset + lib->addr);
*renew = recod;
void *handle = dlopen("libc.so.6", RTLD_LAZY);
void *real_address = dlsym(handle, (const char*)recod);
if (real_address == NULL)
    for (int i=0; i<10; i++)
    {
        real_address = dlsym(pt, (const char*)recod);
        if (real_address != NULL)
            break;
    }
*renew = (Elf64_Addr)real_address;
```

Test 4

Test4 要解决的问题就是重定位全局变量，对于全局变量，我们可以通过查看 r_info 的值来确定是否是全局变量。并且通过符号表，找到其中的索引就可以知道全局变量的符号名，然后 symbolLookup 即可得到全局符号的正确地址。至于重定位的方法，和重定位函数没什么大的区别。核心代码如下

```
int tag=st/ht;
while (tag--)
{
    renew = (Elf64_Addr*) (relocation_table_addr[tag].r_offset+lib->addr);
    recod =(Elf64_Addr) (lib->addr+relocation_table_addr[tag].r_addend);
    *renew = recod;
    if (relocation_table_addr[tag].r_info != 8)
    {
        int pos=relocation_table_addr[tag].r_info>>32;
        str_offset=sym_table_addr[pos].st_name;
        char *str=sadr + str_offset;
        real_address=symbolLookup(lib, str);
        *renew = (Elf64_Addr) real_address;
    }
}
```

Test 5　(并未成功实现)

依赖树问题，这部分的难点，就是对于依赖项，还有依赖项的依赖项，一个树形结构的依赖树，所以需要 BFS 之类的算法，变量整个依赖树，然后一个一个加载进去，load 方式和上面提到的相同。

实验总结

实验整体感觉是比较难的，应该也算是目前接触到的最难的实验之一，因为需要阅读大量的资料，而且会用到许多很陌生的函数。最大的收获学会如何去查询资料，并且不断调 bug，改错，最后通过实验的方式，亲自体会了动态链接简要的一个过程，并且用 c 语言进行了一个简单的实现。