动态链接器实现

环境配置

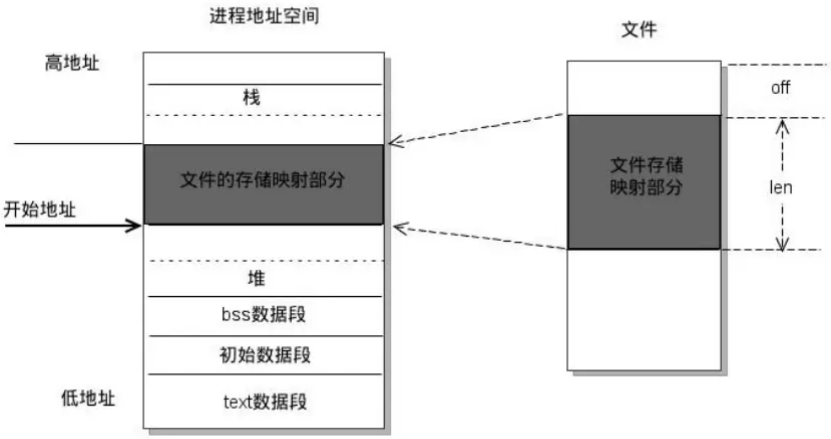
Ubuntu 版本：20.3

python 版本：3.9.12

gcc 版本： 9.4.0

test0: load a library

该测试点的目的，是使用`mmap`把ELF文件中的`LOAD segment`映射到进程的虚拟地址空间，如下图所示



该实验主要分为4个步骤

- 首先打开由`Maplibrary`函数传入参数`libpath`指定的ELF文件，根据返回的文件标识符，来读取`ELF header`和`program header table`，具体代码如下图所示

```
// set space for lib
LinkMap *lib = malloc(sizeof(LinkMap));
int fd = open(libpath, O_RDONLY);

// read elfheader
Elf64_Ehdr *elfh = malloc(sizeof(Elf64_Ehdr));
pread(fd, elfh, sizeof(Elf64_Ehdr), 0);

// read all segment header
Elf64_Phdr *segh = malloc(sizeof(Elf64_Phdr) * elfh->e_phnum);
pread(fd, segh, sizeof(Elf64_Phdr) * elfh->e_phnum, elfh->e_phoff);
```

- 根据读入的`program header table`，可以访问每一个`segment header`，从而可以得知每个`segment`的类型`p_type`，和在ELF文件中的偏移量`p_offset`。找到第一个`LOAD segment`，使用`mmap`将其映射到虚拟内存，并且把其开始地址存放到`LinkMap->addr`。

这里应该注意，由于后续载入的`LOAD`应该保持其`p_vaddr`中所示正确的偏移量，所以第一次映射应该使`mmap`找到一片足够大连续空间，长度应该至少能包含所有的`segment`，在代码中为`len`变量

同时注意由于`mmap`是对齐映射，要求`offset`也应页对齐，因此传入的偏移为`ALIGN_DOWN(pt->p_offset, getpagesize())`，具体代码如下所示

```
Elf64_Phdr *pt = &segh[i]; // get i th segment

if (pt->p_type == PT_LOAD) // point to LOAD
{
    int prot = 0;
    prot |= (pt->p_flags & PF_R) ? PROT_READ : 0;
    prot |= (pt->p_flags & PF_W) ? PROT_WRITE : 0;
    prot |= (pt->p_flags & PF_X) ? PROT_EXEC : 0;
    // NULL means "allow OS to pick up address for you"

    if (first)
    {
        //len保证了第一次映射分配足够大的连续的空间
        size_t len = segh[elfh->e_phnum - 1].p_vaddr + segh[i].p_memsz
- segh[i].p_vaddr;
        void *start_addr = mmap(NULL, ALIGN_UP(len, getpagesize()),
prot,
                                MAP_FILE | MAP_PRIVATE, fd,
ALIGN_DOWN(pt->p_offset, getpagesize()));

        lib->addr = start_addr;
        first = 0;
    }
```

- 将剩下的`LOAD segment`，按照其`p_vaddr`中的偏移量，映射到虚拟地址空间。其开始地址应该为所有`LOAD segment`的起始地址，也就是第一个`LOAD segment`段映射到的`start_addr`，加上在虚拟地址空间的偏移量`p_vaddr`，同时应该注意取整`ALIGN_DOWN(pt->p_vaddr + lib->addr, getpagesize())`；分配的内存应该为其实际映射到虚拟地址空间中所占内存的整数页倍，即`end-start`，具体代码如下所示

```
else
    {
        Elf64_Addr start = ALIGN_DOWN(pt->p_vaddr + lib->addr,
getpagesize());
        Elf64_Addr end = ALIGN_UP(pt->p_vaddr + lib->addr + pt-
>p_memsz, getpagesize());
        mmap((void *)start, end - start, prot, MAP_FILE | MAP_PRIVATE |
MAP_FIXED, fd, ALIGN_DOWN(pt->p_offset, getpagesize()));
    }
```

- 将`dynamic segment`的虚拟内存的地址存放在`LinkMap->dyn`，具体代码如下所示

```
else if (pt->p_type == PT_DYNAMIC)
    lib->dyn = lib->addr + pt->p_vaddr;
```

完整代码如下所示

```
void *MapLibrary(const char *libpath)
{
    // set space for lib
    LinkMap *lib = malloc(sizeof(LinkMap));
    int fd = open(libpath, O_RDONLY);

    // read elfheader
    Elf64_Ehdr *elfh = malloc(sizeof(Elf64_Ehdr));
    pread(fd, elfh, sizeof(Elf64_Ehdr), 0);

    // read all segment header
    Elf64_Phdr *segh = malloc(sizeof(Elf64_Phdr) * elfh->e_phnum);
    pread(fd, segh, sizeof(Elf64_Phdr) * elfh->e_phnum, elfh->e_phoff);

    // load PT_LOAD into the buffer
    int first = 1;
    for (int i = 0; i < elfh->e_phnum; i++)
    {
        Elf64_Phdr *pt = &segh[i]; // get i th segment

        if (pt->p_type == PT_LOAD) // point to LOAD
        {
            int prot = 0;
            prot |= (pt->p_flags & PF_R) ? PROT_READ : 0;
            prot |= (pt->p_flags & PF_W) ? PROT_WRITE : 0;
            prot |= (pt->p_flags & PF_X) ? PROT_EXEC : 0;
            // NULL means "allow OS to pick up address for you"

            if (first)
            {
                size_t len = segh[elfh->e_phnum - 1].p_vaddr +
segh[i].p_memsz - segh[i].p_vaddr;
                void *start_addr = mmap(NULL, ALIGN_UP(len, getpagesize()),
prot,
                                        MAP_FILE | MAP_PRIVATE, fd,
ALIGN_DOWN(pt->p_offset, getpagesize()));

                lib->addr = start_addr;
                first = 0;
            }
            else
            {
                Elf64_Addr start = ALIGN_DOWN(pt->p_vaddr + lib->addr,
getpagesize());
                Elf64_Addr end = ALIGN_UP(pt->p_vaddr + lib->addr + pt-
>p_memsz, getpagesize());
```

```
                mmap((void *)start, end - start, prot, MAP_FILE |
MAP_PRIVATE | MAP_FIXED, fd, ALIGN_DOWN(pt->p_offset, getpagesize()));
            }
        }
        else if (pt->p_type == PT_DYNAMIC)
            lib->dyn = lib->addr + pt->p_vaddr;
    }

    fill_info(lib);
    setup_hash(lib);

    return lib;
}
```

test 1: function relocation

ELF文件格式并不知道外部函数和全局变量的真实地址，使用`GOT`和`PLT`来延迟地址计算的真实时间。这里直接询问`libc`得到地址，实现伪重定位。

首先判断需不需要对外部变量进行重定位，通过`lib->dyn`得到`dynamic section`，在内通过`entry DT_JMPREL`可以得到`relocation table`的地址；然后在`relocation table`中，`rela->r_offset + lib->addr`就可以获知要填入重定位地址的位置。

完整代码如下所示：

```
void RelocLibrary(LinkMap *lib, int mode)
{
    /* Your code here */
    // find the address of referred symbol (we already know)
    Elf64_Dyn *dyn = lib->dyn;
    Elf64_Rela *rela;
    int count = 0;
    if (dyn->d_tag == DT_NEEDED) // outer symbol relocation
    {
        void *handle = dlopen("libc.so.6", RTLD_LAZY);
        void *address = dlsym(handle, "printf");

        while (dyn->d_tag != DT_JMPREL && dyn->d_tag != DT_NULL)
            dyn++;
        if (dyn->d_tag == DT_NULL)
            return;
        rela = dyn->d_un.d_val;
        // add the address with addend and fill it
        Elf64_Addr *pos = rela->r_offset + lib->addr;
        *pos = (Elf64_Addr)address;
        return;
    }
}
```

test 2: initialization

初始化分为两部分，一部分是对本地符号重定位，另一部分是进行初始化，调用初始化函数

在对本地符号重定位时，仍然使用`lib->dyn`找到`DT_RELA`和`DT_RELACOUNT`，获得到重定位表的地址和需要重定位的条目数，然后将`lib->addr + rela->r_addend`赋值给要重定位的位置即可

在调用函数初始化的时候，`DT_INIT`单个函数，`DT_INIT_ARRAY`是一个函数指针数组，其大小由`DT_INIT_ARRAYSZ`确定，仍然是通过`lib->dyn`找到他们，然后调用这些函数。

完整代码如下所示：

```
void InitLibrary(LinkMap *lib)
{
    Elf64_Dyn *Dyn = lib->dyn;
    void (*init_func)(void) = NULL;
    void **init_addr; //???
    Elf64_Rela *rela;
    int fcount = 0;
    int lcount = 0;
    while (Dyn->d_tag != DT_NULL)
    {
        if (Dyn->d_tag == DT_RELA)
            rela = (Elf64_Rela *)(Dyn->d_un.d_val);
        if (Dyn->d_tag == DT_RELACOUNT)
            lcount = (int)(Dyn->d_un.d_val);
        if (Dyn->d_tag == DT_INIT)
            init_func = (void *)Dyn->d_un.d_ptr;
        if (Dyn->d_tag == DT_INIT_ARRAY)
            init_addr = (void *)Dyn->d_un.d_ptr;
        if (Dyn->d_tag == DT_INIT_ARRAYSZ)
            fcount = (int)Dyn->d_un.d_val;
        Dyn++;
    }

    for (int i = 0; i < lcount; ++i)
    {
        uint64_t *addr = lib->addr + rela->r_offset;
        *addr = lib->addr + rela->r_addend;
        rela++;
    }
    init_func();
    for (int i = 0; i < fcount / 8; i++)
    {
        void (*func)(void) = (void *)(*init_addr);
        func();
        init_addr++;
    }
}
```

autograder测试结果

```
Test name: lazy binding
SIGSEGV received in custom loader. Maybe you want to debug it with gdb.
-----------------------------------------------
Your Score: 90 / 100
(base) macondo@macondo-VirtualBox:~/Documents/COMP461905-master$
```