Lab03 实验报告

姓名： 何伟　　**学号：** 171240537

2018 年 11 月 24 日

摘要

完成了三个任务, 找到程序入口, 加载了程序, 并实现了简单的 bt 功能.

1　寻找正确的程序入口

讲义上提示内容很充分. 按照讲义提示, 遍历整个符号表, 找到类型为 FUNC 的符号, 在 elf.h 中为 STT_FUNC. 锁定为 FUNC 类型的符号之后在字符串表中根据偏移量找到对应的字符, 判断是否为"main", 如果是则返回函数地址. 实现如下:

```
152
153 uintptr_t get_entry(void) {
154     for(int i=0;i<nr_symtab_entry;i++){
155       if((symtab[i].st_info&0xf)!=STT_FUNC)
156           continue;
157       char *name = strtab+symtab[i].st_name;
158       //if(name[0]=='m'&&name[1]=='a'&&name[2]=='i'&&name[3]=='n'&&name[4]=='\0'){
159           //printf("hello\n");
160           //assert(0);
161           //printf("%x",symtab[i].st_value);
162           //printf("%d\n",strcmp(name,"main"));
163           //return symtab[i].st_value;
164       //}
165       if(strcmp(name,"main")==0){
166           return symtab[i].st_value;
167       }
168     }
169   return 0;
170 }
171
```

图 1: 寻找入口

2　加载程序

第一步中跳到了 main 函数的地址, 但是并没有把程序加载到相应的内存中. 关心的是 segment 的内容. 声明了一个存储程序头表的结构体, 从 elf 文件中读入内容. 之后遍历这个表项, 找到属性为 PL_LOAD 的项之后按照对应的地址和偏移量加载到正确的虚拟内存位置, 并进行清零. 一开始没有将 elf 中所有的内容都读入数组, 出了点问题, 无法加载相应的程序, 已解决. 实现如下:

```
101   //load program header table
102   uint32_t ph_size = elf->e_phentsize * elf->e_phnum;
103   nr_prog_size = ph_size/sizeof(ph[0]);
104   ph = malloc(ph_size);
105   fseek(fp, elf->e_phoff,SEEK_SET);
106   ret = fread(ph, ph_size, 1, fp);
107   assert(ret == 1);
108
```

图 2: 获取程序头表

```
173 void loader(char *filename) {
174   int ret;
175   int size;
176   int i=0;
177
178   FILE *fp = fopen(filename, "rb");
179   assert(fp != NULL);
180
181   fseek(fp,0,SEEK_END);
182   size = ftell(fp);
183   //printf("%d\n",size);
184   fclose(fp);
185
186   char *buf;
187   fp = fopen(filename,"rb");
188   buf = (char *)malloc(sizeof(char)*size);
189   while(!feof(fp)){
190       ret=fscanf(fp,"%c",&buf[i]);
191       assert(ret==1||ret==-1);
192       i++;
193   }
194   //Elf32_Ehdr *elf = (void *)buf;
195
196   for(int i=0; i<nr_prog_size;i++){
197       if(ph[i]. p_type == PT_LOAD){
198           //assert(0);
199           //cnt++;
```

图 3: 加载程序

```
196   for(int i=0; i<nr_prog_size;i++){
197       if(ph[i]. p_type == PT_LOAD){
198           //assert(0);
199           //cnt++;
200           //printf("%d\n",cnt);
201           memcpy((void *)ph[i].p_vaddr,(const void *)(ph[i].p_offset+buf),ph[i].p_filesz);
202           memset((void *)ph[i].p_vaddr+ph[i].p_filesz,0,ph[i].p_memsz-ph[i].p_filesz);
203       }
204   }
205 }
```

图 4: 加载程序

3 打印栈帧链

做之前先用 gdb 试了一下, 看打出来大概是什么样子. 在实现时, 首先写了一个 get-name 用于得到相应的函数名, 之后遍历”链表”并打印栈帧中保存的信息. 实现时有个

纠结的地方, 就是遍历什么什么时候截止, 如果直接根据 ebp 判断, 会在 main.c 中的函数截止, 由于不在符号表中, 打印???, 如果根据 ebp 指向地址中的存的信息判断, 就在加载的程序中截止, 最后一个会打印 main(). 最后还是选择了第一种, 顺便验证一下找不到函数的情况. 实现如下:

```
207 char *getname(uint32_t offset){
208     for(int i=0;i<nr_symtab_entry;i++){
209         if(symtab[i].st_value<=offset&&offset<=symtab[i].st_value+symtab[i].st_size){
210             if((symtab[i].st_info&0xf)!=STT_FUNC)
211                 continue;
212             char *ret = strtab+symtab[i].st_name;
213             return ret;
214         }
215     }
216     return NULL;
217 }
```

图 5: getname

```
219 void bt(uint32_t ebp, uint32_t eip) {
220     int num = 0;
221     while(ebp!=0)   {
222         //eip = *(uint32_t *)(ebp+1);
223         //ebp = *(uint32_t *)ebp;
224         char *ret = getname(eip);
225         uint32_t *args;
226         args = (uint32_t *)ebp+2;
227         if(ret!=NULL){
228             printf("#%d  0x%x in %s()  ",num++,eip,ret);
229             printf("(%d %d %d %d)\n",args[0],args[1],args[2],args[3]);
230         }
231         else{
232             printf("#%d  0x%x in ???  ",num++,eip);
233             printf("(%d %d %d %d)\n",args[0],args[1],args[2],args[3]);
234         }
235         eip = *((uint32_t *)ebp+1);
236         //eip = *(uint32_t *)eip;
237         ebp = *(uint32_t *)ebp;
238     }
239 }
```

图 6: bt

```
 maruko@me ~/lab3  master ● ./a.out prog/segmentfault
entry = 0x80489af
This is prog/segmentfault.c
catch SIGSEGV
ebp = fff301a8, eip = 80488ea
#0  0x80488ea in die()  (-850844 134515119 -851496 134515058)
#1  0x8048941 in recursive()  (5 5 1 2)
#2  0x8048972 in recursive()  (5 4 1 2)
#3  0x8048972 in recursive()  (5 3 1 2)
#4  0x80489a6 in recursive()  (5 3 1 1)
#5  0x80489a6 in recursive()  (5 3 1 0)
#6  0x804898c in recursive()  (5 3 0 0)
#7  0x8048958 in recursive()  (4 3 0 0)
#8  0x8048958 in recursive()  (3 3 0 0)
#9  0x8048972 in recursive()  (3 2 0 0)
#10  0x8048972 in recursive()  (3 1 0 0)
#11  0x8048958 in recursive()  (2 1 0 0)
#12  0x8048958 in recursive()  (1 1 0 0)
#13  0x8048972 in recursive()  (1 0 0 0)
#14  0x8048958 in recursive()  (0 0 0 0)
#15  0x80489e7 in main()  (-850844 134515119 -851016 1448753312)
#16  0x565a38a0 in ???  (-135483392 -135483392 0 -137302399)
 maruko@me ~/lab3  master ●
```

图 7: 执行示例

4 思考题

4.1 堆和栈在哪里?

二者都在 cpu 的 RAM 中. 栈是内存分配的程序执行分配的暂存带, 粗略的理解为流水线的存储. 在进程开始时, 栈就会被创建, 而且大小固定, 会存储函数调用的信息和一些局部变量. 在函数调用时, 栈保存函数用到的信息. 由于后进先出的性质, 栈使得进程的执行流程变得十分清晰.

堆是内存中的可以动态分配的空间, 会在程序运行时分配, 退出时回收. 用户可以通过函数申请堆中的空间, 并随时回收.

4.2 如何识别不同格式的可执行文件

ELF 文件开头几个字节可以用来确定文件的类型或格式. 在加载时, 会根据魔数判断文件类型是否正确.

4.3 消失的符号

符号表中存三种符号: 1 . 在模块m中定义并被其他模块引用的全局符号; 2 . 由其他模块定理并被m引用的外部符号; 3 . 在模块m中定义并在m中引用的本地符号. 因此, 在程序中定义或引用的函数名和全局变量会出现在符号表中. 宏定义的 N 在预编译被替换, 不会出现在符号表中, 形参和局部变量不是全局变量, 不会出现在符号表中. 符号可以理解为内存和字符串的一种映射.

4.4 寻找"Hello World"

在字符串表中的偏移量为 0x603f68, 对应了 elf 文件节中的.rodata 节..rodata 存储只读数据, 如 printf 语句中的格式串, 开关语句中的跳转表等.

4.5 冗余的符号表

在可执行文件中去掉了符号表, 程序仍正常运行, 但是链接是去掉了 hello.o 的符号表, 连接失败找不到"main " 定义的地方. 链接器在生成一个可执行目标文件时, 必须完成符号解析, 而要进行符号解析, 就需要用到符号表, 去掉符号表, 在解析"main" 时就已经发生了错误. 连接完成之后生成了可执行文件, 加载程序时就不会再用到程序头表, 正如 task 中实现的 loader 一样, 因此程序正常运行.

4.6 冗余的属性

在找到了程序的入口之后需要加载 elf 文件中的部分内容到对应的虚拟地址空间, 程序才能正常执行. FileSiz 属性指明了需要加载的文件的大小,MemSize 是这段加载内容占用的虚拟地址中的内存, 因为要将这个文件加载到内存中的区域, 因此文件的大小通常不会大于内存的大小.

4.7 为什么要清零

没搞懂, 一开始以为是防止有数据段写到这个区域, 预留了一段空白内存预防这种情况, 后来发现如果不清零, 程序无法正常加载. 但是没有想明白为什么会出现这种情况.

5 实验中遇到的问题

可能刚开始比较难吧, 要熟悉一下 elf 文件的结构可以利用的各种结构体. 在熟悉了之后做起来也挺流畅. 通过字符串表与符号表和程序头表的映射关系完成 3 个部分的内容. 可能也是做了 PA3 的原因, 对程序的加载有了初步的认识, 但是花了很久还是没有弄清楚为什么不清零就无法正常的加载程序. 之后自己写了一下文件进行测试, 不过测试的时候没有输出, 是群里大佬们讨论的没有换行符, 字符串留在了缓冲区的问题.