ICS期末出题-第七章

（Linux工具链）简单，基础  1-2 min

1. 以下关于Linux系统上处理可执行文件的工具的说法**不正确**的是
2. 使用 objdump 反汇编 .text 节的机器码
3. 使用 readelf 读取文件的节头部头(section header)和程序头部表(program header)
4. 使用 ls 查询文件是文本文件还是二进制文件
5. 使用 gdb 加载可执行文件、设断点，然后单步调试运行

**C 错误**

解析：

A 正确。objdump -dj .text [file]

B 正确。节头部表：readelf -S [file] 程序头部表：readelf -l [file]

C 错误。ls只是取得文件的元数据，这与文件内容无关，而linux文件元数据中也不包含对binary或者text编码属性的描述。其他诸如grep和file的工具使用heuristics确定文件的类型。

D 正确。gdb 可以支持单步调试。

（静态链接）中等，需要2-4min

1. 对如下两个C程序，用gcc 生成对应的.o模块，再链接在一起得到a.out。则下列说法正确的是：

|  |  |
| --- | --- |
| // main.c  #include <stdio.h>  static int a;  int main() {  int \*func();  printf("%ld\n", func() - &a);  return 0;  } | // util.c  int a = 0;  int \*func() {  return &a;  } |

1. 在 main.o 中，符号 a 位于 COM 伪节
2. 在 util. o 中，符号 a 位于 COM 伪节
3. 无论怎样链接和运行 a.out， a.out 输出的结果都一样，但必不为 0
4. 以上说法都不正确

**D 正确**

A 错。它位于 .bss 节。

B 错。它位于 .bss 节。

C 错。注意 ld 时文件顺序的交换会改变 a.out 中两个符号的相对偏移。

于是 D 正确。

（动态链接）困难，需要理解并分析 需要2-4min

1. 以下程序可以使用 gcc dl.c -ldl 编译并正常运行。如果缺少了 -ldl 标志，链接时会报错 undefined reference to `dlopen'。基于你对于动态链接的理解，请分析出以下说法中**不正确**的一项。

|  |
| --- |
| // dl.c  #include <dlfcn.h>  const char \*path = "/lib/libc.so";  int (\*printf)(const char \*x);  int main() {    // 加载共享库    void \*handle = dlopen(path, RTLD\_NOW);    // 解析符号 "printf" 并返回地址    printf = dlsym(handle, "printf");    // 调用    printf("2022 is coming!\n");    // 关闭共享库    dlclose(handle);  } |

1. 该机器上libdl.so模块中包含符号名为dlopen 的动态链接符号表条目
2. 在a.out 文件中包含printf 的PLT 条目和相应的GOT 条目
3. 在a.out 文件中包含dlopen 的PLT 条目和相应的GOT 条目
4. 如果使用gcc -ldl dl.c编译程序，则会在链接时发生同样错误

**B 错误。**

解析：

省略 -ldl 标志报错，表明 ld 默认不会包含 libdl.so (与之对比，libc.so 默认包含)，并且 dlopen 的定义来自于该共享库。

A 正确。.dynsym 含有一个符号表条目。格式形如

0000000000001390 g DF .text 0000000000000085 GLIBC\_2.2.5 dlopen

“动态链接符号表”的描述是准确的，也不影响理解。

B 错误，printf 只是一个未初始化的全局变量。它不是内置的 printf 函数。

C 正确。默认程序动态绑定 dlopen 到共享库，它需要自己的 PLT 表和 GOT 表。

D 正确。gcc 按照命令行顺序解析。不管是动态库还是静态库只解析当前已经被引用的符号（这一点容易推断，否则没有必要建立专门的库文件格式了。因此没有补充在题目中交代动态链接符号解析的规则。），所以 -ldl 放在第一个位置没有任何效果。最后会在链接阶段产生 dlopen、dlsym 或者 dlclose 未能解析的错误。

额外说明，libc.so 和 libdl.so 在实际系统上可能会带上版本号，路径名一般也更复杂。这里为了出题，做了合适的简化。

1. **(本大题共三问，共10分)** 有以下三个 c 文件 hd.h f1.c f2.c。使用

gcc -c f1.c f2.c; gcc f1.o f2.o

编译后得到可执行文件a.out。回答以下问题。Part A 中涉及的符号所对应的变量已在代码中加粗。**本大题无需理解代码的含义。**

|  |  |
| --- | --- |
| f1.c | #include "hd.h"  #include <stdio.h>  const int **total** = 1 << 30;  static int count = 0;  static Point **pnt**;  int **iter**;  int main() {  for (iter = 0; iter < total; ++iter) {  rand\_point(&pnt);  count += if\_inside(&pnt);  }  printf("Integral on [0,1] is %lf.\n",  1.0 \* count / total);  } |
| hd.h | typedef struct {  double x;  double y;  } **Point**;  void rand\_point(Point \*);  int if\_inside(Point \*); |
| f2.c | #include "hd.h"  #include <stdlib.h>  #include <time.h>  void rand\_point(Point \*ptr) {  static int **seed** = 0;  if (!seed) {  srand((unsigned)time(NULL));  seed = 1;  }  ptr->x = 1.0 \* rand() / RAND\_MAX;  ptr->y = 1.0 \* rand() / RAND\_MAX;  }  int if\_inside(Point \*p) {  return 1 / (1 + p->x) >= p->y;  } |

Part A. **(每个符号1分，共5分)** 请说明以下符号是否在a.out的符号表中。如果是，请进一步指出符号定义所在的节，可能的选择有.text、.data、.bss、.rodata、COM、UNDEF、ABS。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 符号名 | iter | pnt | Point | total | seed |
| 在符号表中？**（填是/否）** |  |  |  |  |  |
| 定义所在节 |  |  |  |  |  |

Part B.**(每空1分，共3分)** 使用objdump -dx f1.o f2.o 看到如下几条代码。**这里你可以将重定位类型R\_X86\_64\_PLT32和R\_X86\_64\_PC32同等看待。**

|  |
| --- |
| # objdump 重定位条目格式：  #           OFFSET: TYPE              VALUE  # e.g.          18: R\_X86\_64\_PLT32    rand\_point-0x4  # 所有数值均以十六进制表示  **# f1.o**  0000000000000000 <main>:  ... # 省略无关代码  17: e8 00 00 00 00. callq  1c <main+0x1c>            18: R\_X86\_64\_PLT32      rand\_point-0x4  1c: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi            1f: R\_X86\_64\_PC32       .bss+0xc  23: e8 00 00 00 00 callq  28 <main+0x28>            24: R\_X86\_64\_PLT32      if\_inside-0x4  28: 89 c2 mov %eax,%edx  ... # 省略无关代码  **# f2.o**  0000000000000000 <rand\_point>:  ... # 省略无关代码  0000000000000074 <if\_inside>:  ... # 省略无关代码 |

据此你可以确定 <main+0x1f> 处的重定位条目是针对符号\_\_\_\_\_\_\_\_(**填写符号名，不要填写.bss这个节名**)的重定位，同时该符号定义的位置在 f1.o 中相对于 .bss 节的偏移量是 0x\_\_\_\_\_\_\_\_\_。

现已知 a.out 文件中 <main+0x17> 行变成

|  |
| --- |
| 11a1: e8 69 00 00 00          callq  <rand\_point> |

那么 a.out 中 <main+0x23> 行将变成

|  |
| --- |
| 11ad: e8 \_\_\_\_\_\_\_\_\_\_\_          callq  <if\_inside> |

Part C. **(每空1分，共2分)** 使用execve加载a.out并执行时，其中第一个被执行的语句默认是\_\_\_\_\_\_\_(单选) 函数的开头。已知 gcc -e 可以修改该默认行为到一个程序指定的函数，据此你推断该函数执行在\_\_\_\_\_\_\_态下(填 用户/内核)。

1. \_init     B.main     C.\_\_libc\_start\_main    D.\_start

解析：[Part A 和 Part C 是基础题，Part B 难度适中。**答案均唯一**]

**Part A**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 符号名 | iter | pnt | Point | total | seed |
| 是否在符号表中 | **是** | **是** | **否** | **是** | **否** |
| 定义所在节 | **.bss** | **.bss** |  | **.rodata** |  |

Point和total的部分容易出错。Point作为类型定义，在C中其结构信息已经作为偏移量被汇编代码包含，不需要再显式地输出到.o文件中。total已经被初始化成一个非零值，由于const修饰 [read-only]，将放入.rodata中。[改卷时 .data 和 .rodata 均给分]

**评分标准：在符号表中的符号，必须正确写出其定义所在节才能得分。**

**Part B**

**pnt;  10;  d1 00 00 00**

由汇编可知 <main+0x1c> 处是准备if\_inside函数的参数。于是符号是pnt。假设其相对.bss偏移为x，refaddr表示条目的地址，则根据重定位类型都是相对偏移，有

.bss + 0xc - refaddr = .bss + x - %rip

于是

x = 0xc + %rip – refaddr = 0xc + 0x4 = 0x10

因为if\_inside – rand\_point = 0x74不变，故第四问的结果是

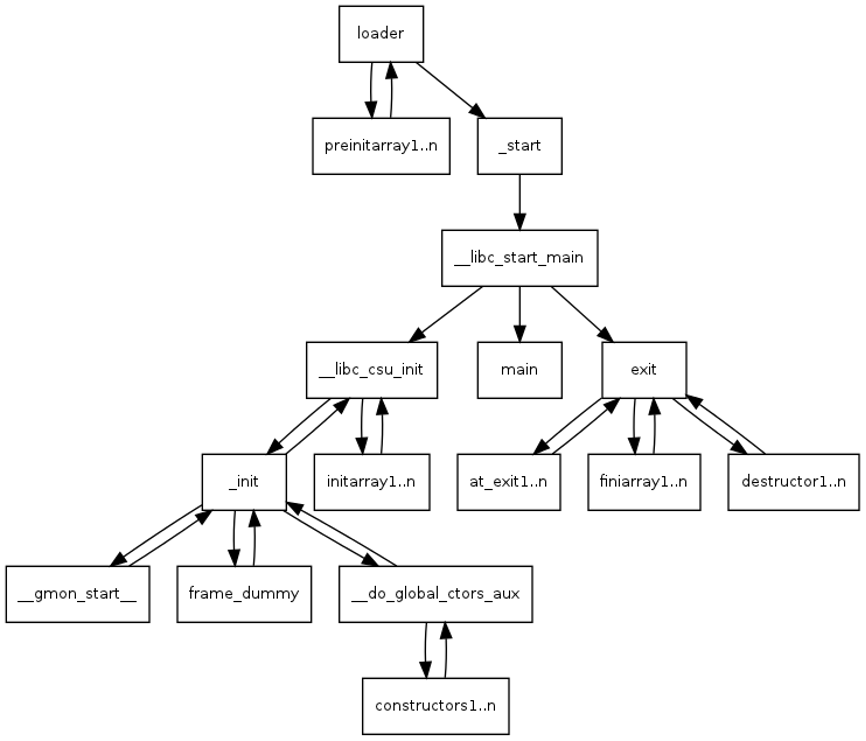
0x69 – (0x11ad – 0x11a1) + 0x74 = 0xd1

**评分标准：第二空允许有若干前导0，其余每空必须完全一致才得分**

**Part C** **D; 用户**  \_start是OS执行这段程序的第一条语句，即入口点，它来自于crt0.o(或者crt1.o，代表 c run-time)模块[因为这段程序没有自定义入口点函数]。第二问的信息已经强烈暗示了该函数只能运行在用户态下[否则直接破坏操作系统对机器的保护和用户间隔离的作用]。实际上，\_start函数只需准备好 argc、argv和envp，然后准备好 \_\_libc\_start\_main 的参数。

具体的启动过程如下图所示[图源：<http://dbp-consulting.com/tutorials/debugging/linuxProgramStartup.html>]

**评分标准：必须完全一致才得分**



本题代码是一个简单的 Monte Carlo 法求积分的算法。