作业 2

贾城昊
2021K8009929010

1. C 程序如下：

```
#include <stdio.h>
#include <stdlib.h>

int global_variable_data = 10; //  存储在 data 段

int global_variable_bss; //  存储在 bss 段

int main() {
    int stack_variable = 20; //  存储在栈上
    static int static_variable_data = 40; //  初始化的静态局部变量存储在 data 段
    static int static_variable_bss; //未初始化的静态局部变量存储在 bss 段

    int *heap_variable = (int *)malloc(sizeof(int)); //  存储在堆上
    *heap_variable = 30;

    printf("Position of global_variable_data: %p\n", &global_variable_data);
    printf("Position of static_variable_data: %p\n", &static_variable_data);
    printf("Position of global_variable_bss: %p\n", &global_variable_bss);
    printf("Position of static_variable_bss: %p\n", &static_variable_bss);
    printf("Position of stack_variable: %p\n", &stack_variable);
    printf("Position of heap_variable: %p\n", heap_variable);
    free(heap_variable); //  释放堆上的内存

    return 0;
}
```

如上图所示的 C 程序，**.data** 段保存的是那些已经初始化了的全局静态变量和局部静态变量，**.bss** 段用于存储未初始化的全局和静态变量。函数内部，调用函数后，局部变量存储在函数的栈帧上。而 malloc 创建的变量在运行时动态分配了堆上的内存来存储值（即动态变量存在堆上）。

所以，上述变量中：

global_variable_data 和 static_variable_data 存在.data 段；

global_variable_bss 和 static_variable_bss 存在.bss 段

stack_variable 存在栈上

heap_variable 存在堆上

上述程序运行结果如下：

```
Position of global_variable_data: 0x601050
Position of static_variable_data: 0x601054
Position of global_variable_bss: 0x601060
Position of static_variable_bss: 0x60105c
Position of stack_variable: 0x7ffc3ccfcddc
Position of heap_variable: 0x1c6e010
```

2. 使用 readelf 命令进行查看（下面的 hw2 为文件名）

1) 使用 readelf -x .data hw2

该命令可以看到.data 段的内容，结果如下：

```
sai@Computer:~/workspace/OS$ readelf -x .data hw2

“.data”节的十六进制输出：
  0x00601040 00000000 00000000 00000000 00000000 ................
  0x00601050 0a000000 28000000                    ....(...
```

可以看出在 0x00601050 地址上，存的值是 0xa0，（因为是小端序，所以是 0a000000），global_variable_dat 的值；在 0x00601054 地址上，存的值是 0x28，对应 static_variable_data 的值

2) 使用 readelf -S hw2

该命令可以显示段头信息。这会列出每个段的详细信息，包括名称、地址、大小等，部分结果如下：

```
[25] .data             PROGBITS         0000000000601040  00001040
     0000000000000018  0000000000000000  WA       0     0     8
[26] .bss              NOBITS           0000000000601058  00001058
     0000000000000010  0000000000000000  WA       0     0     4
```

第一行从左到右依次为：名称，类型，地址，偏移量

第二行从左到右依次为：大小，全体大小，旗标，链接，信息，对齐

3) 使用 objdump -s hw2

该命令可以显示所有段的内容。这会显示二进制文件中所有段的内容，包括代码、数据、bss 等，部分结果如下：

```
Contents of section .data:
 601040 00000000 00000000 00000000 00000000  ................
 601050 0a000000 28000000                    ....(...
```

与 readelf 得出的结果一样，在 0x00601050 地址上，存的值是 0xa0，即 global_variable_dat 的值；在 0x00601054 地址上，存的值是 0x28，对应 static_variable_data 的值

4) 使用 objdump -h hw2

该命令可以显示文件头信息，可用于查看文件的头部信息，包括文件类型、入口点地址、段数等，结果如下：

```
24 .data         00000018  0000000000601040  0000000000601040  00001040  2**3
                 CONTENTS, ALLOC, LOAD, DATA
25 .bss          00000010  0000000000601058  0000000000601058  00001058  2**2
                 ALLOC
```

5) 使用 objdump -t hw2

该命令可以显示符号表。这会列出文件中定义的所有符号，包括函数和变量的名称、地址等，结果如下：

```
0000000000601054 l     O .data  0000000000000004              static_variable_data.2807
000000000060105c l     O .bss   0000000000000004              static_variable_bss.2808
0000000000601060 g     O .bss   0000000000000004              global_variable_bss
0000000000000000       F *UND*  0000000000000000              __libc_start_main@@GLIBC_2.2.5
0000000000601040 g       .data  0000000000000000              __data_start
0000000000000000  w      *UND*  0000000000000000              __gmon_start__
0000000000601048 g     O .data  0000000000000000              .hidden __dso_handle
0000000000400780 g     O .rodata        0000000000000004              _IO_stdin_used
0000000000601050 g     O .data  0000000000000004              global_variable_data
```

可以看出，global_variable_data 和 static_variable_data 存在.data 段；global_variable_bss 和 static_variable_bss 存在.bss 段，与之前分析的一致

3. 程序使用到了栈

C 程序通常会使用栈来存储函数调用的局部变量和函数调用信息，该程序定义自己的局部变量 stack_variable 变量，因此程序使用到了栈。