# 作业 2

毕定钧 2021K8009906014

本次作业包含:

2.1

一个 $C$ 程序可以编译成目标文件或可执行文件。目标文件和可执行文件通常包含 $text$、$data$、$bss$、$rodata$ 段，程序执行时也会用到堆（$heap$）和栈（$stack$）。

（1）请写一个 $C$ 程序，使其包含 $data$ 段和 $bss$ 段，并在运行时包含堆的使用。请说明所写程序中哪些变量在 $data$ 段、$bss$ 段和堆上。

（2）请了解 $readelf$、$objdump$ 命令的使用，用这些命令查看（1）中所写程序的 $data$ 和 $bss$ 段，截图展示。

（3）请说明（1）中所写程序是否用到了栈。

提交内容：所写 C 程序、问题解答、截图等。

## Linux 可执行文件的段管理

当程序被装载后，数据和指令分别被映射到两个虚拟内存区域。数据段对进程来讲是可读写的，而代码段对进程来说是只读的，所以这两个虚拟内存区域的权限可以被分别设置为可读写和只读，防止程序的指令被有意和无意地改写。现代 $CPU$ 的缓存一般被设计成数据缓存和指令缓存分离，程序的指令和数据被分开存放对 $CPU$ 的缓存命中率提高有好处。当系统中运行着多个该程序的副本时，例如多个线程同时都运行同一个程序，它们的代码段指令都是一样的，所以内存中只需要保存一份该程序的代码段，然后将每个副本进程的数据段区域分来，这样可以节省大量空间。

### TEXT

存放程序执行代码。该内存区域通常属于只读，大小在程序运行前即编译时就已经确定。在 $TEXT$ 段中也有可能包含一些只读的常数变量，某些在指令中直接使用立即数赋值的变量也会直接存放在 $TEXT$ 段。

### COMMON

存放弱引用的符号，即未初始化的全局变量，在链接时再放入 $BSS$ 段。编译器也可以直接将未初始化的全局变量放到 $BSS$ 段而省略掉 $COMMON$ 段。

### BSS

*Block Started by Symbol*：用于存放程序中未初始化的全局变量、初始化为 0 的全局变量以及已定义但未赋初值的静态变量，它不占用程序文件的大小，而是占用程序执行时的内存空间，属于静态内存分配，是可读写的。出于编译优化的考虑，初始化为 0 的全局变量会存放在 $BSS$ 段内，由于未初始化的全局变量也会默认初始化为 0，没有必要在目标文件里进行实际存储，只需要描述地址和大小，运行时候分配物理地址空间即可，可以减小目标文件体积。$gcc$ 编译选项 $-fno-common$ 会强

制把为未初始化的全局变量放入 $BSS$ 段，使其变为强符号。由于强符号只能存在一处定义，如果有多处强符号定义，则在链接阶段会报错。此编译选项可防止强符号覆盖弱符号引起的未知错误。

### DATA

用来存放程序中已初始化且初始化不为 0 的全局变量及已赋初值的静态变量。$DATA$ 段属于静态内存分配，是可读写的。

### RODATA

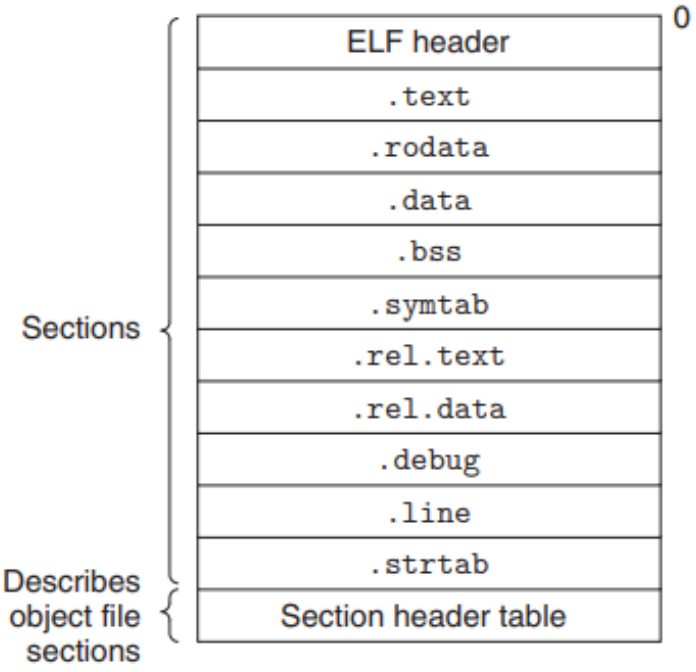
*Read Only Data*：用于存放常量数据，包括使用 *const* 修饰的全局常量、$\#define$ 定义的常量、字符串常量等，属性映射为只读。值得一提的是，使用 *const* 修饰的局部变量没有被放入 $RODATA$ 中，仅起到只读的作用。编译器会去除重复的字符串常量，仅仅保留一份存储在 $RODATA$ 中。

### HEAP

程序运行中动态分配的内存段，大小不固定，可动态扩张或缩减。堆由用户申请和释放，申请时只分配虚存，真正存储数据时才分配对应的实存，释放时也并不是马上释放实存，而可能被反复利用。如果频繁的分配较小的堆内存，容易加剧内存碎片化。

### STACK

栈又称堆栈，存放程序临时创建的局部变量以及函数参数变量，由系统自动分配和释放。在函数被调用时，其参数也会被压入发起调用的进程栈中，并且待到调用结束后，函数的返回值也会被存放回栈中。



# 1 C 程序

## 1.1 源码

```c
#include <stdio.h>
#include <stdlib.h>
#define OK      0                       //------------DATA_TYPE------------
#define ERROR   1                       //  defined_constant
int         init_glo_var = 1;           //  initialized_globle_variable
int         uninit_glo_var;             //  uninitialized_globle_variable
static int  init_glo_static_var = 2;    //  initialized_globle_static_variable
static int  uninit_glo_static_var;      //  uninitialized_globle_static_variable
const int   glo_const = 3;              //  global_constant

int func(int data)
{
    if (data == 21) {
        printf("%d\n", data);
        return OK;
    }
    else
        return ERROR;
}// func

int main()
{
    int         init_loc_var = 4;       //  initialized_local_variable
    int         uninit_loc_var;         //  uninitialized_local_variable
    static int  init_loc_static_var = 5;//  initialized_local_static_variable
    static int  uninit_loc_static_var;  //  uninitialized_local_static_variable
    const int   loc_const = 6;          //  local_constant
    int *       heap_var;               //  heap_variable
    heap_var    = (int *)malloc(sizeof(int));

    *heap_var    = init_glo_var + init_glo_static_var + glo_const + init_loc_var
                + init_loc_static_var + loc_const;
    printf("The function has been executed with return value %d.\n", func(*heap_var));
    return 0;
}// main
```

| | |
|---|---|
| $DATA$ | $init\_glo\_var,\ init\_glo\_static\_var, init\_loc\_static\_var$ |
| $BSS$ | $uninit\_glo\_var, uninit\_glo\_static\_var, uninit\_loc\_static\_var$ |
| $HEAP$ | $heap\_var$ |

## 1.2 运行结果

### 1.2.1 函数运行结果

```
21
The function has been executed with return value 0.
```

### 1.2.2 *readelf* 结果

```
   28: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
   29: 00000000000010b0     0 FUNC    LOCAL  DEFAULT   16 deregister_tm_clones
   30: 00000000000010e0     0 FUNC    LOCAL  DEFAULT   16 register_tm_clones
   31: 0000000000001120     0 FUNC    LOCAL  DEFAULT   16 __do_global_dtors_aux
   32: 000000000000401c     1 OBJECT  LOCAL  DEFAULT   26 completed.8061
   33: 0000000000003db8     0 OBJECT  LOCAL  DEFAULT   22 __do_global_dtors_aux_fin
   34: 0000000000001160     0 FUNC    LOCAL  DEFAULT   16 frame_dummy
   35: 0000000000003db0     0 OBJECT  LOCAL  DEFAULT   21 __frame_dummy_init_array_
   36: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.c
   37: 0000000000004014     4 OBJECT  LOCAL  DEFAULT   25 init_glo_static_var
   38: 0000000000004020     4 OBJECT  LOCAL  DEFAULT   26 uninit_glo_static_var
   39: 0000000000004018     4 OBJECT  LOCAL  DEFAULT   25 init_loc_static_var.2842
   40: 0000000000004024     4 OBJECT  LOCAL  DEFAULT   26 uninit_loc_static_var.284
   41: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
   42: 00000000000021bc     0 OBJECT  LOCAL  DEFAULT   20 __FRAME_END__
   43: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS
   44: 0000000000003db8     0 NOTYPE  LOCAL  DEFAULT   21 __init_array_end
   45: 0000000000003dc0     0 OBJECT  LOCAL  DEFAULT   23 _DYNAMIC
   46: 0000000000003db0     0 NOTYPE  LOCAL  DEFAULT   21 __init_array_start
   47: 0000000000002048     0 NOTYPE  LOCAL  DEFAULT   19 __GNU_EH_FRAME_HDR
   48: 0000000000003fb0     0 OBJECT  LOCAL  DEFAULT   24 _GLOBAL_OFFSET_TABLE_
   49: 0000000000001000     0 FUNC    LOCAL  DEFAULT   12 _init
   50: 0000000000001290     5 FUNC    GLOBAL DEFAULT   16 __libc_csu_fini
   51: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_deregisterTMCloneTab
   52: 0000000000004000     0 NOTYPE  WEAK   DEFAULT   25 data_start
   53: 000000000000401c     0 NOTYPE  GLOBAL DEFAULT   25 _edata
   54: 0000000000004028     4 OBJECT  GLOBAL DEFAULT   26 uninit_glo_var
   55: 0000000000001298     0 FUNC    GLOBAL HIDDEN    17 _fini
   56: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND printf@@GLIBC_2.2.5
   57: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
   58: 0000000000004000     0 NOTYPE  GLOBAL DEFAULT   25 __data_start
   59: 0000000000002008     4 OBJECT  GLOBAL DEFAULT   18 glo_const
   60: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
   61: 0000000000004008     0 OBJECT  GLOBAL HIDDEN    25 __dso_handle
   62: 0000000000002000     4 OBJECT  GLOBAL DEFAULT   18 _IO_stdin_used
   63: 0000000000001169    57 FUNC    GLOBAL DEFAULT   16 func
   64: 0000000000001220   101 FUNC    GLOBAL DEFAULT   16 __libc_csu_init
   65: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND malloc@@GLIBC_2.2.5
   66: 0000000000004030     0 NOTYPE  GLOBAL DEFAULT   26 _end
   67: 0000000000001080    47 FUNC    GLOBAL DEFAULT   16 _start
   68: 0000000000004010     4 OBJECT  GLOBAL DEFAULT   25 init_glo_var
   69: 000000000000401c     0 NOTYPE  GLOBAL DEFAULT   26 __bss_start
   70: 00000000000011a2   124 FUNC    GLOBAL DEFAULT   16 main
   71: 0000000000004020     0 OBJECT  GLOBAL HIDDEN    25 __TMC_END__
   72: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_registerTMCloneTable
   73: 0000000000000000     0 FUNC    WEAK   DEFAULT  UND __cxa_finalize@@GLIBC_2.2
```

### 1.2.3 *objdump* 结果

```
0000000000004000 l    d  .data  0000000000000000              .data
0000000000004014 l     O .data  0000000000000004              init_glo_static_var
0000000000004018 l     O .data  0000000000000004              init_loc_static_var.2325
0000000000004000  w      .data  0000000000000000              data_start
000000000000401c g       .data  0000000000000000              _edata
0000000000004000 g       .data  0000000000000000              __data_start
0000000000004008 g     O .data  0000000000000000              .hidden __dso_handle
0000000000004010 g     O .data  0000000000000004              init_glo_var
0000000000004020 g     O .data  0000000000000000              .hidden __TMC_END__
```

```
000000000000401c l    d  .bss   0000000000000000              .bss
000000000000401c l     O .bss   0000000000000001              completed.8061
0000000000004020 l     O .bss   0000000000000004              uninit_glo_static_var
0000000000004024 l     O .bss   0000000000000004              uninit_loc_static_var.2326
0000000000004028 g     O .bss   0000000000000004              uninit_glo_var
0000000000004030 g       .bss   0000000000000000              _end
000000000000401c g       .bss   0000000000000000              __bss_start
```

## 1.3 栈的使用

前面提到，在函数被调用时其参数会进入进程栈中，之后其返回值也会被存储在栈中，该 $C$ 程序定义并使用了函数 $func$，并以 $heap\_var$ 为参数，返回值为 $0/1$，因此用到了栈。