第 2 行告诉我们第一个段(代码段)有读/执行访问权限，开始于内存地址 0x400000 处，总共的内存大小是 0x69c 字节，并且被初始化为可执行目标文件的头 0x69c 个字节，其中包括 ELF 头、程序头部表以及.init、.text 和.rodata 节。

第 3 行和第 4 行告诉我们第二个段(数据段)有读/写访问权限，开始于内存地址 0x600df8 处，总的内存大小为 0x230 字节，并用从目标文件中偏移 0xdf8 处开始的.data节中的 0x228 个字节初始化。该段中剩下的 8 个字节对应于运行时将被初始化为 0 的.bss 数据。

对于任何段 s，链接器必须选择一个起始地址 vaddr，使得

$$\text{vaddr mod align} = \text{off mod align}$$

这里，off 是目标文件中段的第一个节的偏移量，align 是程序头部中指定的对齐(2^{21} = 0x200000)。例如，图 7-14 中的数据段中

$$\text{vaddr mod align} = \text{0x600df8 mod 0x200000} = \text{0xdf8}$$

以及

$$\text{off mod align} = \text{0xdf8 mod 0x200000} = \text{0xdf8}$$

这个对齐要求是一种优化，使得当程序执行时，目标文件中的段能够很有效率地传送到内存中。原因有点儿微妙，在于虚拟内存的组织方式，它被组织成一些很大的、连续的、大小为 2 的幂的字节片。第 9 章中你会学习到虚拟内存的知识。

7.9　加载可执行目标文件

要运行可执行目标文件 prog，我们可以在 Linux shell 的命令行中输入它的名字：

```
linux> ./prog
```

因为 prog 不是一个内置的 shell 命令，所以 shell 会认为 prog 是一个可执行目标文件，通过调用某个驻留在存储器中称为加载器(loader)的操作系统代码来运行它。任何 Linux 程序都可以通过调用 execve 函数来调用加载器，我们将在 8.4.6 节中详细描述这个函数。加载器将可执行目标文件中的代码和数据从磁盘复制到内存中，然后通过跳转到程序的第一条指令或入口点来运行该程序。这个将程序复制到内存并运行的过程叫做*加载*。

每个 Linux 程序都有一个运行时内存映像，类似于图 7-15 中所示。在 Linux x86-64 系统中，代码段总是从地址 0x400000 处开始，后面是数据段。运行时堆在数据段之后，通过调用 malloc 库往上增长。(我们将在 9.9 节中详细描述 malloc 和堆。)堆后面的区域是为共享模块保留的。用户栈总是从最大的合法用户地址($2^{48}-1$)开始，向较小内存地址增长。栈上的区域，从地址 2^{48} 开始，是为内核(kernel)中的代码和数据保留的，所谓内核就是操作系统驻留在内存的部分。

为了简洁，我们把堆、数据和代码段画得彼此相邻，并且把栈顶放在了最大的合法用户地址处。实际上，由于.data 段有对齐要求(见 7.8 节)，所以代码段和数据段之间是有间隙的。同时，在分配栈、共享库和堆段运行时地址的时候，链接器还会使用地址空间布局随机化(ASLR，参见 3.10.4 节)。虽然每次程序运行时这些区域的地址都会改变，它们的相对位置是不变的。

当加载器运行时，它创建类似于图 7-15 所示的内存映像。在程序头部表的引导下，加载器将可执行文件的片(chunk)复制到代码段和数据段。接下来，加载器跳转到程序的