入口点，也就是_start 函数的地址。这个函数是在系统目标文件 ctrl.o 中定义的，对所有的 C 程序都是一样的。_start 函数调用系统启动函数__libc_start_main，该函数定义在 libc.so 中。它初始化执行环境，调用用户层的 main 函数，处理 main 函数的返回值，并且在需要的时候把控制返回给内核。

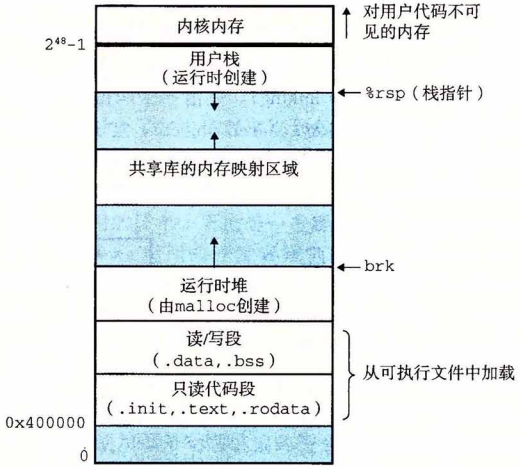

图 7-15 Linux x86-64 运行时内存映像。没有展示出由于段对齐要求和地址空间布局随机化(ASLR)造成的空隙。区域大小不成比例

旁注 加载器实际是如何工作的？

我们对于加载的描述从概念上来说是正确的，但也不是完全准确，这是有意为之。要理解加载实际是如何工作的，你必须理解进程、虚拟内存和内存映射的概念，这些我们还没有加以讨论。在后面第 8 章和第 9 章中遇到这些概念时，我们将重新回到加载的问题上，并逐渐向你揭开它的神秘面纱。

对于不够有耐心的读者，下面是关于加载实际是如何工作的一个概述：Linux 系统中的每个程序都运行在一个进程上下文中，有自己的虚拟地址空间。当 shell 运行一个程序时，父 shell 进程生成一个子进程，它是父进程的一个复制。子进程通过 execve 系统调用启动加载器。加载器删除子进程现有的虚拟内存段，并创建一组新的代码、数据、堆和栈段。新的栈和堆段被初始化为零。通过将虚拟地址空间中的页映射到可执行文件的页大小的片(chunk)，新的代码和数据段被初始化为可执行文件的内容。最后，加载器跳转到_start地址，它最终会调用应用程序的 main 函数。除了一些头部信息，在加载过程中没有任何从磁盘到内存的数据复制。直到 CPU 引用一个被映射的虚拟页时才会进行复制，此时，操作系统利用它的页面调度机制自动将页面从磁盘传送到内存。

7.10 动态链接共享库

我们在 7.6.2 节中研究的静态库解决了许多关于如何让大量相关函数对应用程序可用的问题。然而，静态库仍然有一些明显的缺点。静态库和所有的软件一样，需要定期维护和更新。如果应用程序员想要使用一个库的最新版本，他们必须以某种方式了解到该库的