Linux认证:Linux内存管理Linux认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022_Linux_E8_AE_A4_E8_AF_c103_644913.htm 前言 内存管理一向是所有操作系统书籍不惜笔墨重点讨论的内容，无论市面上或是网上都充斥着大量涉及内存管理的教材和资料。因此，我们这里所要写的Linux内存管理采取避重就轻的策略，从理论层面就不去班门弄斧，贻笑大方了。我们最想做的和可能做到的是从开发者的角度谈谈对内存管理的理解，最终目的是把我们在内核开发中使用内存的经验和对Linux内存管理的认识与大家共享。 当然，这其中我们也会涉及到一些诸如段页等内存管理的基本理论，但我们的目的不是为了强调理论，而是为了指导理解开发中的实践，所以仅仅点到为止，不做深究。 遵循“ 理论来源于实践 ”的“ 教条 ”，我们先不必一下子就钻入内核里去看系统内存到底是如何管理，那样往往会让你陷入似懂非懂的窘境（我当年就犯了这个错误！）。所以最好的方式是先从外部（用户编程范畴）来观察进程如何使用内存，等到大家对内存的使用有了较直观的认识后，再深入到内核中去学习内存如何被管理等理论知识。最后再通过一个实例编程将所讲内容融会贯通。 进程与内存 进程如何使用内存？ 毫无疑问，所有进程（执行的程序）都必须占用一定数量的内存，它或是用来存放从磁盘载入的程序代码，或是存放取自用户输入的数据等等。不过进程对这些内存的管理方式因内存用途不一而不尽相同，有些内存是事先静态分配和统一回收的，而有些却是按需要动态分配和回收的。 对任何一个普通进程来讲，它都会涉及到5种不同的数据段。稍有编

程知识的朋友都能想到这几个数据段中包含有“ 程序代码段 ”、“ 程序数据段 ”、“ 程序堆栈段 ”等。不错，这几种数据段都在其中，但除了以上几种数据段之外，进程还另外包含两种数据段。下面我们来简单归纳一下进程对应的内存空间中所包含的5种不同的数据区。 代码段：代码段是用来存放可执行文件的操作指令，也就是说是它是可执行程序在内存中的镜像。代码段需要防止在运行时被非法修改，所以只准许读取操作，而不允许写入（修改）操作它是不可写的。 数据段：数据段用来存放可执行文件中已初始化全局变量，换句话说就是存放程序静态分配[1]的变量和全局变量。 BSS段[2]：BSS段包含了程序中未初始化的全局变量，在内存中bss段全部置零。 堆（heap）：堆是用于存放进程运行中被动态分配的内存段，它的大小并不固定，可动态扩张或缩减。当进程调用malloc等函数分配内存时，新分配的内存就被动态添加到堆上（堆被扩张）；当利用free等函数释放内存时，被释放的内存从堆中被剔除（堆被缩减） 栈：栈是用户存放程序临时创建的局部变量，也就是说我们函数括弧“ {} ”中定义的变量（但不包括static声明的变量，static意味着在数据段中存放变量）。除此以外，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。由于栈的先进先出特点，所以栈特别方便用来保存/恢复调用现场。从这个意义上讲，我们可以把堆栈看成一个寄存、交换临时数据的内存区。 进程如何组织这些区域？ 上述几种内存区域中数据段、BSS和堆通常是被连续存储的 内存位置上是连续的，而代码段和栈往往会被独立存放。有趣的是，堆和栈两个区域关系很“ 暧昧 ”，他

们一个向下“ 长 ”（i386体系结构中栈向下、堆向上），一个向上“ 长 ”，相对而生。但你不必担心他们会碰头，因为他们之间间隔很大（到底大到多少，你可以从下面的例子程序计算一下），绝少有机会能碰到一起。 下图简要描述了进程内存区域的分布： 数据段 BSS 代码段 堆 栈 “ 事实胜于雄辩 ”，我们用一个小例子（原形取自《User-Level Memory Management》）来展示上面所讲的各种内存区的差别与位置。 #includegt. #includegt. #includegt. int bss_var. int data_var0=1. int main(int argc,char **argv) { printf("below are addresses of types of processs mem\n"). printf("Text location:\n"). printf("\tAddress of main(Code Segment):%p\n",main). printf("_____\n"). int stack_var0=2. printf("Stack Location:\n"). printf("\tInitial end of stack:%p\n",amp.stack_var1). printf("_____\n"). printf("Data Location:\n"). printf("\tAddress of data_var(Data Segment):%p\n",amp.data_var1). printf("_____\n"). printf("BSS Location:\n"). printf("\tAddress of bss_var:%p\n",&amp.bss_var). printf("_____\n"). char *b = sbrk((ptrdiff_t)0). printf("Heap Location:\n"). printf("\tInitial end of heap:%p\n",b). brk(b 4). b=sbrk((ptrdiff_t)0). printf("\tNew end of heap:%p\n",b). return 0. } 它的结果如下 below are addresses of types of processs mem Text location: Address of main(Code Segment):0x8048388 _____ Stack Location: Initial end of stack:0xbffffab4 new end of stack:0xbffffab0