 总而言之,产生段错误就是访问了错误的内存段，一般是你没有权限，或者根本就不存在对应的物理内存,尤其常见的是访问0地址. 一般来说,段错误就是指访问的内存超出了系统所给这个程序的内存空间，通常这个值是由gdtr来保存的，他是一个48位的寄存器，其中的32位是保存由它指向的gdt表，后13位保存相应于gdt的下标，最后3位包括了程序是否在内存中以及程序的在cpu中的运行级别,指向的gdt是由以64位为一个单位的表，在这张表中就保存着程序运行的代码段以及数据段的起始地址以及与此相应的段限和页面交换还有程序运行级别还有内存粒度等等的信息。一旦一个程序发生了越界访问，cpu就会产生相应的异常保护，于是segmentation fault就出现了. 在编程中以下几类做法容易导致段错误,基本是是错误地使用指针引起的 1)访问系统数据区，尤其是往 系统保护的内存地址写数据最常见就是给一个指针以0地址 2)内存越界(数组越界，变量类型不一致等) 访问到不属于你的内存区域 解决方法 我们在用C/C 语言写程序的时侯，内存管理的绝大部分工作都是需要我们来做的。实际上，内存管理是一个比较繁琐的工作，无论你多高明，经验多丰富，难 免会在此处犯些小错误，而通常这些错误又是那么的浅显而易于消除。但是手工“ 除虫 ”（debug），往往是效率低下且让人厌烦的，本文将就"段错误"这个 内存访问越界的错误谈谈如何快速定位这些"段错误"的语句。下面将就以下的一个存在段错误的程序介绍几种调试方法：1

dummy_function (void)2 {3 unsigned char *ptr = 0x00.4 *ptr = 0x00.5 }67 int main (void)8 {9 dummy_function ().1011 return 0.12 } 作为一个熟练的C/C 程序员，以上代码的bug应该是很清楚的，因为它尝试操作地址为0的内存区域，而这个内存区域通常是不可访问的禁区，当然就会出错了。我们尝试编译运行它:xiaosuo@gentux test $ ./a.out段错误 果然不出所料，它出错并退出了。 1.利用gdb逐步查找段错误: 这种方法也是被大众所熟知并广泛采用的方法，首先我们需要一个带有调试信息的可执行程序，所以我们加上“ -g -rdynamic"的参数进行编译，然后用gdb调试运行这个新编译的程序,具体步骤如下:xiaosuo@gentux test $ gcc -g -rdynamic d.cxiaosuo@gentux test $ gdb ./a.outGNU gdb 6.5Copyright (C) 2006 Free Software Foundation, Inc.GDB is free software, covered by the GNU General Public License, and you arewelcome to change it and/or distribute copies of it under certain conditions.Type "show copying" to see the conditions.There is absolutely no warranty for GDB. Type "show warranty" for details.This GDB was configured as "i686-pc-linux-gnu"...Using host libthread_db library "/lib/libthread_db.so.1".(gdb) rStarting program: /home/xiaosuo/test/a.outProgram received signal SIGSEGV, Segmentation fault.0x08048524 in dummy_function () at d.c:44 *ptr = 0x00.(gdb) 好像不用一步步调试我们就找到了出错位置d.c文件的第4行，其实就是如此的简单。 从这里我们还发现进程是由于收到了SIGSEGV信号而结束的。通过进一步的查阅文档(man 7 signal)，我们知道SIGSEGV默认handler的动作是打印 ” 段错误"的出错信息，并产生Core文件，由此我们又产生了

方法二。 2.分析Core文件： Core文件是什么呢？The default action of certain signals is to cause a process to terminate and produce a core dump file, a disk file containing an image of the processs memory at the time of termination. A list of the signals which cause a process to dump core can be found in signal(7). 奇怪了，我的系统上并没有找到core文件。后来，忆起为了渐少系统上的拉圾文件的数量，禁止了core文件的生成，查看了以下果真如此，将系统的core文件的大小限制在512K大小，再试:xiaosuo@gentux test $ ulimit -c0xiaosuo@gentux test $ ulimit -c 1000xiaosuo@gentux test $ ulimit -c1000xiaosuo@gentux test $ ./a.out段错误 (core dumped)xiaosuo@gentux test $ lsa.out core d.c f.c g.c pango.c test_iconv.c test_regex.c core文件终于产生了，用gdb调试一下看看吧:xiaosuo@gentux test $ gdb ./a.out coreGNU gdb 6.5Copyright (C) 2006 Free Software Foundation, Inc.GDB is free software, covered by the GNU General Public License, and you arewelcome to change it and/or distribute copies of it under certain conditions.Type "show copying" to see the conditions.There is absolutely no warranty for GDB. Type "show warranty" for details.This GDB was configured as "i686-pc-linux-gnu"...Using host libthread_db library "/lib/libthread_db.so.1".warning: Cant read pathname for load map: 输入/输出错误.Reading symbols from /lib/libc.so.6...done.Loaded symbols for /lib/libc.so.6Reading symbols from /lib/ld-linux.so.2...done.Loaded symbols for /lib/ld-linux.so.2Core was generated by `./a.out.Program terminated with signal 11, Segmentation fault.#0 0x08048524 in dummy_function () at d.c:44

*ptr = 0x00. 还是一步就定位到了错误所在地，佩服一下Linux/Unix系统的此类设计。接着考虑下去，以前用windows系统下的ie的时侯，有时打开某些网页，会出现“运行时错误”，这个时侯如果恰好你的机器上又装有windows的编译器的话，他会弹出来一个对话框，问你是否进行调试，如果你选择是，编译器将被打开，并进入调试状态，开始调试。