C 语言指针错误的分析及调试①

许永达

(湛江师范学院 信息与教育技术中心, 湛江 524048)

摘 要: C 语言指针的有些错误在程序编译阶段难以发现, 且现行教材主要从概念、理论上对指针错误进行讲述, 存在不足. 分析了带有此类错误的示例程序, 并在 VISUAL C++ 6.0 进行调试, 展示此类指针错误的错误现象, 分析其产生的原因, 提出正确使用指针的方法, 以达到预防此类指针错误发生的目的.

关键词: C 语言; 指针; Visual C++ 6.0; 调试; 编程习惯

Analyzing and Debugging the Errors of C Language Pointer

XU Yong-Da

(Information and Educational Technology Center, Zhanjiang Normal University, Zhanjiang 524048, China)

Abstract: Some pointer errors in C Programming are not easily found at the compiling phase. The current teaching materials cannot provide sufficient description on those errors, but mainly focusing on concept or theory. This article aims at preventing those errors by analyzing those errors in sample programs, debugging those errors in VISUAL C++ 6.0, showing the phenomena of those errors, analyzing their causes, and putting forward the correct way to use pointers.

Key words: C language; pointer; Visual C++ 6.0; debug; programming practice

1 引言

C 语言功能丰富、表达能力强、使用灵活方便、应用面广、目标程序效率高、可移植性好, 既具有高级语言的优点, 又具有低级语言的许多特点, 既适于编写系统软件, 又能方便地用来编写应用软件[1].

C 语言和其它语言比较起来, 指针是其特有的, 指针使 C 更加简洁、高效, 也是 C 语言非常流行的原因之一. 利用指针可以方便地表示各种数据结构, 并能像汇编语言一样处理内存地址, 从而可以编写出精练而高效的程序. 指针虽然强大, 与之相伴的风险却也不小, 如果使用不当, 就会引起程序错误甚至崩溃, 并且有时候极难发现原因[2,3].

鉴于 C 语言的重要性和广泛使用性, 需要学习编程的学生, 必须要学好 C 语言. 其中, 指针是很重要, 也是很容易出错的部分, 而有些指针错误很隐蔽, 在程序编译阶段难以发现. 目前许多教材对于指针错误都是从概念, 理论上进行讲述, 存在不足, 本文将列举各种在编译阶段难以发现的指针错误, 并在 VISUAL C++ 6.0 中进行调试, 以展示其错误现象, 分析其产生的原因, 并提出正确使用指针的方法. VISUAL C++ 6.0 界面简洁, 占用资源少, 操作方便, 在实际工作, 学习中被广泛使用, 全国计算机等级考试二级 C 语言程序设计自 2008 年也开始以 VC++6.0 作为考试环境, 所以本文选用它作为编程和调试环境[4-7].

2 指针错误的分析及调试

指针赋值错误, 类型转换错误等的语法错误在 VC 6.0 的编译阶段就可以发现, 然而, 指针变量实质存储的是一个内存地址, 当指针非法访问了该内存, 就产生了指针错误, 此类错误在程序运行时才会出现, 且其中的一些错误在调试时不能直接定位到程序的出错行, 这就增加了调试程序的难度. 下文将列举此类指针错误的三种常见类型: 1) 指针在使用前未初始化; 2) 指针指向的内存已经释放; 3) 指针指向的地址越界. 并在 VC 6.0 中

① 收稿时间:2012-08-28;收到修改稿时间:2012-10-15

进行调试，分析原因，及提出解决的方法. 以下示例程序是 VC 6.0 调试版本的程序，请读者在 VC 6.0 中进行以下设置：在 Build 工具栏中选择“Win32 Debug”.

2.1 指针在使用前未初始化

野指针不是空指针，是指向垃圾内存的指针. 野指针的成因主要有两种：其一是指针变量没有被初始化，其二是指针被 free 或者 delete 之后，没有置为 NULL，让人误以为是合法的指针[8]. C 语言的指针变量在创建时不会自动初始化，它们的值是随机的，若访问它们指向的内存可能导致程序崩溃，这种未初始化的指针就是其中一种野指针.

示例程序如下：

```
void main()
{
int *p;
if(*p>0)
{
    printf("%d ",*p);
}
}
```

在 VC 6.0 中 Compile 会产生如下警告：

warning C4700: local variable 'p' used without having been initialized

继续 Build，然后运行程序，VC 6.0 会报错如下：

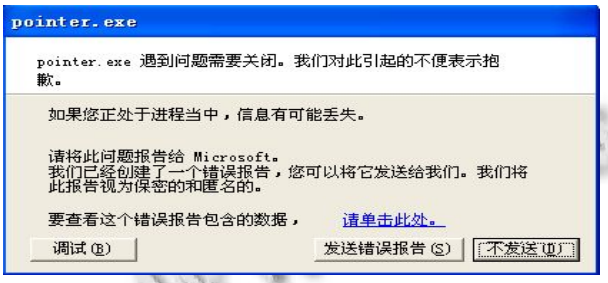

图 1　报错窗口 1

点击调试，进入调试环境，弹出错误窗口如下：

图 2　报错窗口 2

点击确定后，VC 6.0 能定位到出错行

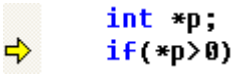

图 3　定位到出错行

因此，在程序编译时产生警告：“warning C4700: local variable 'p' used without having been initialized”，就应当检查程序，检查变量“p”是否已经初始化了. 指针变量在定义时就应当初始化，或者设为 NULL 指针，使用前先判断此指针是否为 NULL 指针，例如 if(p!=NULL)，若指针 p 不为空才能使用，即可避免此类野指针错误的出现[9].

2.2 指针指向的内存已经释放

使用 malloc 成功分配动态内存的指针，在不使用时，通过 free 函数把内存释放，但实际上，指针变量存储的仍是之前成功分配的内存的地址值，若没有及时把指针设为 NULL，这个指针就成为野指针，后面程序有可能把已经 free 的指针当作合法指针来使用，就可能导致程序出错[10].

示例程序如下：

```
void main()
{
int *p=(int *) malloc(sizeof(int));
free(p);
if(p!=NULL)
{
    *p=13;
    printf("%d ",*p);
}
}
```

此程序运行结果为：“13”. VC 6.0 没有报错，因为 C 语言给了程序员很大的自由，其编译器不作此类检查. 然而这个程序是错误的，p 指向的内存空间已经释放，但由于 p 存储了 malloc 分配的内存的地址，所以其值不为空，if(p!=NULL)返回为真，*p=13 可对其进行赋值. 由于程序后面的操作没有覆盖这个内存空间，所以程序正常运行，但要是在一个大型的程序中，此内存空间有可能被后面的程序覆盖，程序可能因此而崩溃，并且很难找出错误的源头.

因为此类野指针错误在其指向的内存空间被后面程序覆盖前不会报错，VC 6.0 无法帮你查找此类错误，所以需要养成良好的编程习惯来避免此类错误，在 free 指

针后，应当把指针设为 NULL，并在后面的程序使用该指针的时候，先判断是否为 NULL，再进行使用. 例如:

```
int *p=(int *) malloc(sizeof(int));
free(p);
p=NULL;
if(p!=NULL)
{
    *p=13;
   printf("%d ",*p);
}
```

另外一种“指针指向的内存已经释放”的错误是: 如果指针指向比自身作用域要小的变量时，当该变量的生命周期结束，该变量的内存将被系统回收，指向该变量的指针成为野指针，若访问该指针将非法访问内存.

示例程序如下:

```
void main()
{
int *p;
{
    int i=10;
    p=&i;
}
printf("%d\n",*p);
}
```

同样, VC 6.0 不会报错，程序可以正常运行，上述程序的输出结果为“10”. 但这个程序是错误的，因为指针 p 指向变量 i，但 i 在局部程序块退出后，该变量指向的内存被操作系统回收，此时指针 p 指向了无效的地址. 如果误把 p 当作合法指针来使用，并且程序后面的操作覆盖了这个内存空间，程序可能崩溃.

因此，为避免此类指针错误的发生，需要养成良好的编程习惯，注意不要使用作用域大的指针变量指向作用域小的变量[9].

2.3 指针指向的地址越界

由于 C 语言的编译器不作指针的越界检查，因此在编程过程中，稍有疏忽就有可能引起指针访问越界[2].

示例程序如下:

```
void main()
{
int arr[2]={1,2};
int *p;
p=arr+2;
*p=100;
printf("%d ",*p);
}
```

以上的程序是错误的，因为 arr 数组的长度只有 2，而 p 指向的是 arr+2 的地址，已经越界. 运行上述程序，VC 6.0 会报错如上面的图 1 所示:

点击调试，进入调试环境，弹出错误窗口如上面的图 2 所示，点击确定. 弹出 Find Source 窗口:

点击确定，弹出窗口如下:

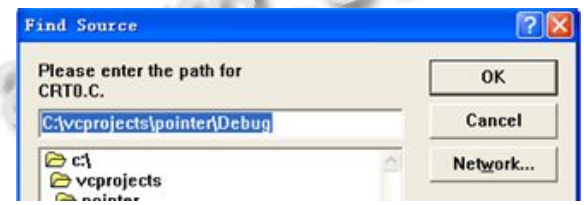

图 4 Find Source 窗口

点击 Cancel，将进入反汇编，无法定位到错误行:

图 5 反汇编窗口

这时，我们可以通过使用 Call Stack 调用堆栈窗口来获得调试信息. 首先介绍一下什么叫调用堆栈: 假设我们有几个函数，分别是 function1, function2, function3，且 function1 调用 function2，function2 调用 function3. 在 function3 运行过程中，我们可以从线程当前堆栈中了解到是哪几个函数调用它. 从函数的顺序关系看, function3、function2、function1 呈现出一种“堆栈”的特征，最后被调用的函数 function3 出现在最上方. 因此称呼这种关系为调用堆栈 (call stack). VC 6.0 调试环境下的 Call Stack 窗口会显示出调用堆栈的相关信息，例如，程序运行当前所在行数，函数参数的当前取值，将要调用的其他函数[11]. 在 Call Stack 窗口中显示了一个调用系列，最上面的是当前函数，往下依次是调用函数的上级函数，双击这些函数名可以跳到对应的函数中去.

在 Debug 工具栏中，打开 Call Stack 窗口.

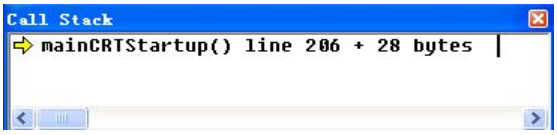

图 6 Call Stack 窗口 1

Call Stack 窗口中的 mainCRTSartup()是什么函数? 我们把断点设在刚进入 main()函数时, 来观察一下 Call Stack 窗口的情况.

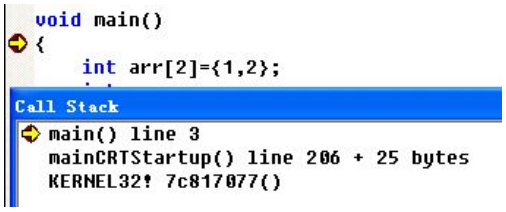

图 7 Call Stack 窗口 2

可以看到 mainCRTSartup()是退出 main()函数后要进入的函数, 根据调用堆栈的原理, 是 mainCRTSartup()调用了 main()函数. 实际上, 操作系统装载应用程序后, 做完初始化工作就转到程序的入口点执行. 程序的默认入口点由连接程序设置, 不同的连接器选择的入口函数也不尽相同. 在 VC 6.0 下, 连接器对控制台程序设置的入口函数是 mainCRTStartup, mainCRTStartup 再调用 main 函数.

因此, 通过图 6 可判断出, 错误应该是发生在 main 函数.

此时, 一般查找错误的方法是在 main 函数里, 在怀疑出错的地方设置断点, 然后单步执行程序, 看看到底哪一行出错.

然而, 此程序在调试环境中, 从 main 函数的一开始一直到结束, 单步调试都没有报错. 但是, 在编程环境中, 一运行程序就报错. 如上面的图 1 所示.

这是由于 C 语言的编译器不对数组越界进行检查, 而在单步调试过程中, 虽然 p 指针变量超越了 arr 数组的边界进行赋值, 但 p 越界访问内存在此程序中没有出现错误, 所以没有报错.. 为什么在编程环境中, 一运行就报错? 这是由于在编程环境中, main 函数生命周期结束后, 会释放 arr 数组的内存, VC 6.0 在此时报错.

由于此类指针错误只在函数退出后报错, 比较难以定位到出错行. 但“指针指向的地址越界”出现的机率还是比较大的, 即使是资深的程序员都有可能犯此错误, 就例如使用指针通过 for 循环操作数组的时候, 可能数组的长度算错了, 或者笔误, 都可能导致指针越界操作.

因此, 当程序出现图 1 或者图 2 的报错窗口时, 可通过 Call Stack 窗口或者设置断点, 定位到出错函数, 如果报错只在函数退出时出现, 就可判断有可能出现指针越界操作了, 就在出错函数中找指针操作数组等可能指针越界操作的代码. 最好的预防方法, 还是养成良好的编程习惯, 例如在进行数组操作时, 熟知数组的长度, 合理使用指针, 不要越界操作数组.

3 关于指针的良好编程习惯

养成使用指针的良好编程习惯, 有助于预防指针错误的发生: 1) 谁分配谁释放的原则, 编程风格上, 在内存使用的分配释放上一定要配对, 函数之间传递动态分配的地址是很危险的, 尽量申请, 释放内存在一个函数中. 2) 尽量不要使用作用域大的指针指向作用域小的变量. 3) 删除指针后立即赋空(NULL), 这可以保护其他程序使用已经删除后的指针. 4) 使用指针之前先确保指针不为空, 不要依赖任何其他程序来保证你使用的指针不为空. 5) 熟知指针所操作变量的内存范围, 不要越界操作指针[12].

4 结语

总的来说, 本文所述的指针错误产生的原因是指针非法访问了内存, 可通过养成良好的编程习惯来预防指针错误的产生, 和使用 VC 6.0 强大的调试工具来修正指针错误. 除此之外, 还可使用 NuMega 系列的 Bounds Checker 等调试工具来调试指针错误. Bounds Checker 安装后, 集成到 VC6.0 中, 使用很方便. BoundsChecker 是一种运行时错误检测工具, 它主要用于高级程序在运行时期发生的各种错误. BoundsChecker 能检测的错误有: 1) 指针操作及资源泄露错误, 例如: 内存泄露, 对指针变量的错误操作. 2)内存操作方面的错误, 例如: 内存读、写溢出,使用未初始化的内存. 3)API 函数使用错误[13].

参考文献

1 谭浩强.C 程序设计.第 4 版.北京:清华大学出版社,2010.9–10.
2 姜青山,洪心兰.C 语言程序设计的健壮性与安全性研究.工矿自动化,2007,5:125–126.
3 郭曦,何炎祥,张焕国,等.一种改进的指针安全分析算法.武汉大学学报(理工版),2010,56(2):170–174.
4 马国峰,杨俊红,李元臣.高职 C 语言教材建设的改革与实践.教育与职业,2012,2:116–117.
5 马英.C 语言程序设计课教学之我见.山西财经大学学报,2011,33(2):131–132.

(下转第 181 页)

级检修信息.

(2) 利用 Direct Media-Xtra 插件, 调用 Sprite 函数, 实现了装配仿真视频的播放、暂停、停止以及音量调节功能. 用户可以通过拖动滑动条, 快进到任意感兴趣的装配画面, 随时控制视频的播放和音量大小. 控制播放语句如下:

CallSprite(@"DirectMedia Xtra ", #videoplay)—播放

CallSprite(@"DirectMedia Xtra ", #videopause)—暂停

CallSprite(@"DirectMedia Xtra ", #videoseek, 0)—停止

CallSprite(@"DirectMedia Xtra ", #setvolume, volume)—音量调节

(3) 利用 ActiveX 控件, 调用 Sprite 函数, 实现虚拟环境的旋转、平移、缩放、复位交互功能, 可以浏览转向架各组成部件的三维模型, 全方位地观察其结构特征. 语句如下:

CallSprite(@"ActiveX",#Loadcult3d,FileLocation^"*.co")

(4) 调用 GoTo 和 Quit 函数,实现翻页、返回和退出按钮响应, 实现页面间的跳转以及系统退出.

2.5 数据压缩

由于系统采用多媒体形式表达丰富的转向架技术信息, 所以数据量也比较大, 尤其是其中的静态和动态图像信息, 信息中存在的冗余会直接影响系统运行速度, 有必要利用数据压缩技术进行优化压缩处理[6].

JPEG 是静态图像压缩标准, 既可以压缩灰度图像和彩色图像,还可以压缩视频序列中的帧内图像. 系统中的图像均采用 JPEG 格式存储, 从而大大减少了文件的存储空间.

MPEG 是动态图像压缩标准, 采用帧内和帧间图像数据压缩技术, 以减少空域和时域冗余. 系统利用 WinAVI Video Converter 专业视频编、解码软件, 采用 XviD MPEG-4, 将转向架总体和各组成部件的装配仿真及检修分解流程视频压缩后, 不仅图像压缩比很高, 数据失真很小, 还保持了原有的视频质量. 比如拖车转向架整体装配仿真动画文件原为 1.8G, 压缩后为 144M.

3 结语

针对 CRH380BL 高速动车组转向架, 设计开发的虚拟仿真检修支持系统, 采用虚拟现实、多媒体、仿真、交互和数据压缩等技术, 实现了转向架各组成部件的动态三维浏览、多媒体信息集成、交互式信息链接、总体和各组成部件的虚拟装配及检修拆卸过程的仿真、以及数据的压缩处理. 系统具有良好的用户界面, 交互性强, 信息表达清晰, 仿真动画形象和生动. 经过系统测试表明, 采用的实现技术是实用和有效的, 达到了系统的功能与性能要求.

参考文献

1 徐良军,章建,蒋毅,等.基于虚拟现实技术的电力安监仿真培训系统.计算机系统应用,2010,19(11):162–165.

2 张卫华,王伯铭.中国高速列车的创新发展.机车电传动,2010,(1):8–12.

3 周建新,王丹虹.基于 Web 的产品信息发布系统的实现.工程图学学报,2008,(5):146–149.

4 姜海涛,邓友银.虚拟装配技术在雷达装配中的应用.机械工程与自动化,2007,(5):32–34.

5 张智.虚拟现实技术在多媒体教学中的应用.现代计算机,2006,(11): 87–89.

6 周碧英.多媒体数据压缩技术.电脑知识与技术,2008,(11):332–333.

(上接第 156 页)

6 叶幼林.对计算机 C 语言教学的探讨与研究.中南民族大学学报(人文社会科学版),2004,24(4):189–190.

7 陈婷.C 语言程序设计实验教学改革探究.实验技术与管理,2010,27(10):182–184.

8 高海昌,冯博琴,何杭军,等.Linux 平台下基于源代码插装的动态内存检测.小型微型计算机系统,2006,27(9):1647–1651.

9 彭程,杨春生.C 语言指针操作技巧探讨.中国高新技术企业,2008,10:107–108.

10 张广梅,李晓维.动态内存错误的动态检测.计算机辅助设计与图形学学报,2005,17(3):400–406.

11 张俊.基于递归树的递归调用分析.实验室研究与探索,2010,29(3):83–87.

12 何灵敏,许翔,陆慧娟,等.C++教学中编程习惯的养成.计算机教育,2011,9:64–67.

13 封亮,严少清.软件白盒测试的方法和实践.计算机工程,2000,26(12):87–90.