温柔的暴力

C/C++中的段错误（Segmentation fault）[转]

Segment fault 之所以能够流行于世，是与Glibc库中基本所有的函数都默认型参指针为非空有着密切关系的。

来自:http://oss.lzu.edu.cn/blog/article.php?uid_7/tid_700.html#comment

背景

最近一段时间在linux下用C做一些学习和开发，但是由于经验不足，问题多多。而段错误就是让我非常头痛的一个问题。不过，目前写一个一千行左右的代码，也很少出现段错误，或者是即使出现了，也很容易找出来，并且处理掉。

那什么是段错误？段错误为什么是个麻烦事？以及怎么发现程序中的段错误以及如何避免发生段错误呢？

一方面为了给自己的学习做个总结，另一方面由于至今没有找到一个比较全面介绍这个虽然是"FREQUENTLY ASKED QUESTIONS"的问题，所以我来做个抛砖引玉吧。下面就从上面的几个问题出发来探讨一下"Segmentation

导航

博客园
首页
新随笔
联系
订阅XML
管理

公告

昵称：温柔的暴力
园龄：5年5个月
粉丝：29
关注：1
+加关注

< 2012年5月 >

日	一	二	三	四	五	六
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

统计

随笔 - 33
文章 - 0
评论 - 57
引用 - 0

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

Python核心编程(8)
练习题答案(8)
字典(4)
百度笔试面试题(3)
编程珠玑(2)

faults"吧。

目录

1。什么是段错误?
2。为什么段错误这么"麻烦"?
3。编程中通常碰到段错误的地方有哪些?
4。如何发现程序中的段错误并处理掉?

正文

1。什么是段错误?

下面是来自Answers.com的定义:

> A segmentation fault (often shortened to segfault) is a particular error condition that can occur during the operation of computer software. In short, a segmentation fault occurs when a program attempts to access a memory location that it is not allowed to access, or attempts to access a memory location in a way that is not allowed (e.g., attempts to write to a read-only location, or to overwrite part of the operating system). Systems based on processors like the Motorola 68000 tend to refer to these events as Address or Bus errors.
>
> Segmentation is one approach to memory management and

编程珠玑学习笔记(2)
思考题(2)
循环(2)
英文拼写纠错(1)
元祖tuple(1)
更多

随笔分类

C# .Net EntityFrameWork(1)
C/C++(6)
Java(2)
Python(15)
编程珠玑(2)
面试笔试题(15)
算法和数据结构(8)
移动互联网

随笔档案

2013年7月 (1)
2012年8月 (12)
2012年7月 (11)
2012年5月 (9)

最新评论

1. Re:《Python核心编程》第二版第七章练习题答案第二部分
这个第一题有问题
--0x02
2. Re:《Python核心编程》第二版第八章练习题答案第一部分
result.append(getprimefactors)[0]应该改为[1]吧，[0]处因子永远为1啊，初学者，感谢作者提供的答案的帮助
--hu1977
3. Re:C/C++中的段错误（Segmentation fault）[转]
总结的好。
--soul11201
4. Re:Entity Framework 学习之--Ling to entity实现分页
不想说什么
--goddices
5. Re:Entity Framework 学习之--Ling to entity实现分页
@通用OA开发平台thank you, 大家一起进步...
--温柔的暴力

阅读排行榜

1. C/C++中的段错误（Segmentation fault）[转](32440)
2. 2011百度校园招聘笔试题 C++类-附原创答案(5565)
3. Entity Framework 学习之--Ling to entity实现分页(2746)

protection in the operating system. It has been superseded by paging for most purposes, but much of the terminology of segmentation is still used, "segmentation fault" being an example. Some operating systems still have segmentation at some logical level although paging is used as the main memory management policy.

On Unix-like operating systems, a process that accesses invalid memory receives the SIGSEGV signal. On Microsoft Windows, a process that accesses invalid memory receives the STATUS_ACCESS_VIOLATION exception.

另外，这里有个基本上对照的中文解释，来自 http://www.linux999.org/html_sql/3/132559.htm

所谓的段错误 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

4. 百度笔试题，malloc/free与new/delete的区别与联系(2699)
5. 《Python核心编程》第二版第八章练习题答案 第一部分(1987)

评论排行榜

1. python 实现文件的递归拷贝(15)
2. 2011百度校园招聘笔试题 C++类-附原创答案(13)
3. Python代码性能优化(10)
4. Entity Framework 学习之--Ling to entity实现分页(6)
5. 编程珠玑学习笔记一 python实现(3)

推荐排行榜

1. Entity Framework 学习之--Ling to entity实现分页(5)
2. 2011百度校园招聘笔试题 C++类-附原创答案(4)
3. C/C++中的段错误（Segmentation fault）[转](4)
4. Python代码性能优化(3)
5. 百度笔试题，malloc/free与new/delete的区别与联系(1)

> 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

通过上面的解释，段错误应该就是访问了不可访问的内存，这个内存区要么是不存在的，要么是受到系统保护的。

2。为什么段错误这么麻烦?

中国linux论坛有一篇精华帖子《Segment fault 之永远的痛》(http://www.linuxforum.net/forum/gshowflat.php?Cat=&Board=program&Number=193239&page=2&view=collapsed&sb=5&o=all&fpart=1&vc=1)在主题帖子里头，作者这么写道:

> 写程序好多年了，Segment fault是许多C程序员头疼的提示。指针是好东西，但是随着指针的使用却诞生了这个同样威力巨大的恶魔。
>
> Segment fault 之所以能够流行于世，是与Glibc库中基本所有的函数都默认型参指针为非空有着密切关系的。
>
> 不知道什么时候才可以有能够处理NULL的glibc库诞生啊!
>
> 不得已，我现在为好多的函数做了衣

服，避免glibc的函数被NULL给感染，导致我的Mem访问错误，而我还不知道NULL这个病毒已经在侵蚀我的身体了。

Segment fault 永远的痛.....

后面有好多网友都跟帖了，讨论了Segmentation faults为什么这么"痛"，尤其是对于服务器程序来说，是非常头痛的，为了提高效率，要尽量减少一些不必要的段错误的"判断和处理"，但是不检查又可能会存在段错误的隐患。

那么如何处理这个"麻烦"呢?

就像人不可能"完美"一样，由人创造的"计算机语言"同样没有"完美"的解决办法。

我们更好的解决办法也许是:

通过学习前人的经验和开发的工具，不断的尝试和研究，找出更恰当的方法来避免、发现并处理它。对于一些常见的地方，我们可以避免，对于一些"隐藏"的地方，我们要发现它，发现以后就要及时处理，避免留下隐患。

下面我们可以通过具体的实验来举出一些经常出现段错误的地方，然后再举例子来发现和找出这类错误藏身之处，最后处理掉。

3。编程中通常碰到段错误的地方有哪些?

为了进行下面的实验，我们需要准备两个工具，一个是gcc，一个是gdb
我是在ubuntu下做的实验，安装这两个东西是比较简单的

```
sudo apt-get install gcc-4.0
libc6-dev
sudo apt-get install gdb
```

好了，开始进入我们的实验，我们粗略的分一下类

1）往受到系统保护的内存地址写数据

有些内存是内核占用的或者是其他程序正在使用，为了保证系统正常工作，所以会受到系统的保护，而不能任意访问。

例子1：

Code:

```
#include <stdio.h>
int main(){
  int i=0;
  scanf("%d",i);
  printf("%d\n",i);
  return 0;
}
```

编译和执行一下

```
$ gcc -o segerr segerr.c
$ ./segerr
10
段错误
```

咋一看，好像没有问题哦，不就是读取一个数据然后给输出来吗?

下面我们来调试一下，看看是什么原因?

```
$ gcc -g -o segerr segerr.c        --加-g选项查看调试信息
$ gdb ./segerr
(gdb) l                    --用l(list)显示我们的源代码
1       #i nclude <stdio.h>
2
3       int
4       main()
5       {
6               int i = 0;
7
8               scanf ("%d", i);
9               printf ("%d\n", i);
10              return 0;
(gdb) b 8                --用b(break)设置断点
Breakpoint 1 at 0x80483b7: file segerr.c, line 8.
(gdb) p i                --用p(print)打印变量i的值[看到没，这里i的值是0哦]
$1 = 0

(gdb) r                    --用r(run)运行，直到断点处
Starting program: /home/falcon/temp/segerr

Breakpoint 1, main () at segerr.c:8
8               scanf ("%d", i);  --[试图往地址0处写进一个值]
```

```
(gdb) n                    --用n(next)执行下一步
10

Program received signal SIGSEGV, Segmentation fault.
0xb7e9a1ca in _IO_vfscanf () from /lib/tls/i686/cmov/libc.so.6
(gdb) c                    --在上面我们接收到了SIGSEGV,然后用c(continue)继续执行
Continuing.

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
(gdb) quit                 --退出gdb
```

果然
我们“不小心”把&i写成了i
而我们刚开始初始化了i为0,这样我们不是试图向内存地址0存放一个值吗?
实际上很多情况下，你即使没有初始化为零，默认也可能是0，所以要特别注意。

补充:
可以通过man 7 signal查看SIGSEGV的信息。

```
$ man 7 signal | grep SEGV
Reformatting signal(7), please wait...
       SIGSEGV      11       Core    Invalid memory reference
```

例子2：

Code:

```
#include <stdio.h>
int main(){
  char *p;
  p = NULL;
  *p = 'x';
  printf("%c", *p);
  return 0;
}
```

很容易发现，这个例子也是试图往内存地址0处写东西。

这里我们通过gdb来查看段错误所在的行

```
$ gcc -g -o segerr segerr.c
$ gdb ./segerr
(gdb) r        --直接运行，我们看到抛出段错误以后，自动显示出了出现段错误的行，这就是一个找出段错误的方法
Starting program: /home/falcon/temp/segerr

Program received signal SIGSEGV, Segmentation fault.
0x08048516 in main () at segerr.c:10
10              *p = 'x';
(gdb)
```

2）内存越界(数组越界，变量类型不一致等)

例子3：

Code:

```
#include <stdio.h>
int main(){
   char test[1];
   printf("%c",
test[1000000000]);
   return 0;
}
```

这里是比较极端的例子，但是有时候可能是会出现的，是个明显的数组越界的问题

或者是这个地址是根本就不存在的

例子4：

Code:

```
#include <stdio.h>
 int main(){
     int b = 10;
     printf("%s\n", b);
     return 0;
}
```

我们试图把一个整数按照字符串的方式输出出去，这是什么问题呢?

由于还不熟悉调试动态链接库，所以我只是找到了printf的源代码的这里

```
声明部分：
   int pos =0
,cnt_printed_chars =0 ,i ;
     unsigned char *chptr ;
     va_list ap ;
%s格式控制部分：
case 's':
         chptr =va_arg (ap
,unsigned char *);
         i =0 ;
```

```
        while (chptr [i ])
        {...
           cnt_printed_chars
++;
           putchar (chptr [i
++]);
     }
```

仔 细看看，发现了这样一个问题，在打印字符串的时候，实际上是打印某个地址开始的所有字符，但是当你想把整数当字符串打印的时候，这个整数被当成了一个地 址，然后printf从这个地址开始去打印字符，直到某个位置上的值为\0。所以，如果这个整数代表的地址不存在或者不可访问，自然也是访问了不该访问的 内存——segmentation fault。

类似的，还有诸如：sprintf等的格式控制问题
比如，试图把char型或者是int的按照%s输出或存放起来，如：
Code:

```
#include <stdio.h>
#include <string.h>
 int main(){
    char c='c';
    int i=10;
    char buf[100];
    printf("%s", c); //试图把
char型按照字符串格式输出，这里的
字符会解释成整数，
                    //再解释成地
址，所以原因同上面那个例子
    printf("%s", i); //试图把int
型按照字符串输出
```

```
    memset(buf, 0, 100);
    sprintf(buf, "%s", c); 试图把char型按照字符串格式转换
    memset(buf, 0, 100);
    sprintf(buf, "%s", i);//试图把int型按照字符串转换
}
```

3）其他

其实大概的原因都是一样的，就是段错误的定义。但是更多的容易出错的地方就要自己不断积累，不段发现，或者吸纳前人已经积累的经验，并且注意避免再次发生。

例如：

<1>定义了指针后记得初始化，在使用的时候记得判断是否为NULL
<2>在使用数组的时候是否被初始化，数组下标是否越界，数组元素是否存在等
<3>在变量处理的时候变量的格式控制是否合理等

再举一个比较不错的例子：

我在进行一个多线程编程的例子里头，定义了一个线程数组
#define THREAD_MAX_NUM
pthread_t
thread[THREAD_MAX_NUM];
用pthread_create创建了各个线程，然后用pthread_join来等待线程的结束

刚 开始我就直接等待，在创建线程都成功的时候，pthread_join能够顺利

等待各个线程结束，但是一旦创建线程失败，那用pthread_join来 等待那个本不存在的线程时自然会存在访问不能访问的内存的情况，从而导致段错误的发生，后来，通过不断调试和思考，并且得到网络上资料的帮助，找到了上面 的原因和解决办法：

在创建线程之前，先初始化我们的线程数组，在等待线程的结束的时候，判断线程是否为我们的初始值
如果是的话，说明我们的线程并没有创建成功，所以就不能等拉。否则就会存在释放那些并不存在或者不可访问的内存空间。

上面给出了很常见的几种出现段错误的地方，这样在遇到它们的时候就容易避免拉。但是人有时候肯定也会有疏忽的，甚至可能还是会经常出现上面的问题或者其他常见的问题，所以对于一些大型一点的程序，如何跟踪并找到程序中的段错误位置就是需要掌握的一门技巧拉。

4。如何发现程序中的段错误?

有个网友对这个做了比较全面的总结，除了感谢他外，我把地址弄了过来。文章名字叫《段错误bug的调试》(http://www.cublog.cn/u/5251/showart.php?id=173718),应该说是很全面的。

而我常用的调试方法有：

1）在程序内部的关键部位输出

(printf)信息，那样可以跟踪 段错误在代码中可能的位置

为了方便使用这种调试方法，可以用条件编译指令#ifdef DEBUG和#endif把printf函数给包含起来，编译的时候加上-DDEBUG参数就可以查看调试信息。反之，不加上该参数进行调试就可以。

2）用gdb来调试，在运行到段错误的地方，会自动停下来并显示出错的行和行号

这 个应该是很常用的，如果需要用gdb调试，记得在编译的时候加上-g参数，用来显示调试信息,对于这个，网友在《段错误bug的调试》文章里创造性的使用 这样的方法，使得我们在执行程序的时候就可以动态扑获段错误可能出现的位置：通过扑获SIGSEGV信号来触发系统调用gdb来输出调试信息。如果加上上 面提到的条件编译，那我们就可以非常方便的进行段错误的调试拉。

3）还有一个catchsegv命令
通过查看帮助信息，可以看到

> Catch segmentation faults in programs

这个东西就是用来扑获段错误的，它通过动态加载器（ld-linux.so）的预加载机制（PRELOAD）把一个事先写好的库（/lib/libSegFault.so）加载上，用于捕捉断错误的出错信息。

到这里，“初级总结篇”算是差不多完成拉。欢迎指出其中表达不当甚至错误的地方，先谢过!

参考资料[具体地址在上面的文章中都已经给出拉]:

1。段错误的定义
Ansers.com
http://www.answers.com
Definition of "Segmentation fault"
http://www.faqs.org/qa/qa-673.html
2。《什么是段错误》
http://www.linux999.org/html_sql/3/132559.htm
3。《Segment fault 之永远的痛》
http://www.linuxforum.net/forum/gshowflat.php?Cat=&Board=program&Number=193239&page=2&view=collapsed&sb=5&o=all&fpart=
4。《段错误bug的调试》
http://www.cublog.cn/u/5251/showart.php?id=173718

后记

虽然感觉没有写什么东西,但是包括查找资料和打字，也花了好些几个小时，不过总结一下也是值得的，欢迎和我一起交流和讨论，也欢迎对文章中表达不当甚至是错误的地方指正一下。

分类: C/C++,面试笔试题

温柔的暴力
关注 - 1
粉丝 - 29
+加关注

« 上一篇：<编程之美>计算0到N中包含数字1的个数[转]
» 下一篇：函数返回指针和返回数组名有什么区别

posted on 2012-05-31 13:57 温柔的暴力 阅读(32440) 评论(1) 编辑 收藏

评论

#1楼 2014-06-01 14:27 soul11201
总结的好。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。