Segmentation Fault错误原因总结

By Wangt

5月 11 2015 更新日期:5月 11 2015

最近在项目上遇到了Segmentation Fault的错误，一直调试不出来是哪里出了问题，对于刚接触嵌入式的，也不知道该如何去调试一个项目，定位内存问题，纠结了好几天，好阿红整理下自己的思路。从头开始。

文章目录

以下内容只为整理来自己使用的，大多来源于网络，感谢大家的分享：

http://www.cnblogs.com/no7dw/archive/2013/02/20/2918372.html

http://blog.chinaunix.net/uid-20780355-id-538814.html

一、什么是“Segmentation fault in Linux”

```
A segmentation fault (often shortened to SIGSEGV) is a particular

Segmentation is one approach to memory management and protection

On Unix-like operating systems, a process that accesses an invalid
```

就是：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。

即“当程序试图访问不被允许访问的内存区域（比如，尝试写一块属于操作系统的内存），或以错误的类型访问内存区域（比如，尝试写一块只读内存）。这个描述是

准确的。为了加深理解，我们再更加详细的概括一下SIGSEGV。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。

> SIGSEGV是在访问内存时发生的错误，它属于内存管理的范畴
>
> SIGSEGV是一个用户态的概念，是操作系统在用户态程序错误访问内存时所做出的处理。
>
> 当用户态程序访问（访问表示读、写或执行）不允许访问的内存时，产生SIGSEGV。
>
> 当用户态程序以错误的方式访问允许访问的内存时，产生SIGSEGV。

用户态程序地址空间，特指程序可以访问的地址空间范围。如果广义的说，一个进程的地址空间应该包括内核空间部分，只是它不能访问而已

二、SIGSEGV产生的可能情况

指针越界和SIGSEGV是最常出现的情况，经常看到有帖子把两者混淆，而这两者的关系也确实微妙。在此，我们把指针运算（加减）引起的越界、野指针、空指针都归为指针越界。SIGSEGV在很多时候是由于指针越界引起的，但并不是所有的指针越界都会引发SIGSEGV。一个越界的指针，如果不解引用它，是不会引起SIGSEGV的。而即使解引用了一个越界的指针，也不一定会引起SIGSEGV。这听上去让人发疯，而实际情况确实如此。SIGSEGV涉及到操作系统、C库、编译器、链接器各方面的内容，我们以一些具体的例子来说明。

（1）错误的访问类型引起

```
#include <stdlib.h>
int main()
{
     char *c = "hello world";
     c[1] = 'H';
}
```

上述程序编译没有问题，但是运行时弹出SIGSEGV。此例中，”hello world”作为一个常量字符串，在编译后会被放在rodata节（GCC），最后链接生成目标程序时.rodata节会被合并到text segment与代码段放在一起，故其所处内存区域是只读的。这就是错误的访问类型引起的SIGSEGV。

（2）访问了不属于进程地址空间的内存

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int* p = (int*)0xC0000fff;
     *p = 10;
}
```

还有另一种可能，往受到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：

```
int  i=0;
scanf ("%d", i);  /* should have used &i */
printf ("%d\n", i);
return 0;
```

（3）访问了不存在的内存

最常见的情况不外乎解引用空指针了，如：

int *p = null;* p = 1;

在实际情况中，此例中的空指针可能指向用户态地址空间，但其所指向的页面实际不存在。

（4）内存越界，数组越界，变量类型不一致等

```
#include <stdio.h>
int  main()
{
    char test[1];
    printf("%c", test[10]);
    return 0;
}
```

这就是明显的数组越界了，或者这个地址根本不存在。

（5）试图把一个整数按照字符串的方式输出

```
    int  main()
    {
        int b = 10;
```

```
        printf("%s\n", b);
        return 0;
    }
```

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：

```
#include <stdio.h>
#include <string.h>
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
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
```

（6）栈溢出了，有时SIGSEGV，有时却啥都没发生

大部分C语言教材都会告诉你，当从一个函数返回后，该函数栈上的内容会被自动“释放”。“释放”给大多数初学者的印象是free()，似乎这块内存不存在了，于是当他访问这块应该不存在的内存时，发现一切都好，便陷入了深深的疑惑。

三、调试定位SIGSEGV

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，使用gdb来快速定位这些”段错误”的语句。其实还有很多其他的方法。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。

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。gcc应该都有安装的。

首先安装gdb: sudo aot-get install gdb

下面是对某个小程序的的调试过程截图：

运行gcc的时候加上-g这个参数查看调试信息，

l：(list)显示我们的源代码

b 行号：在相应的行上设置断点，我在第六行设置

r : run 运行程序至断点

p：p(print)打印变量的值

n：n(next)执行下一步 出现错误信息了

c : continue 继续执行

quit ： 退出gdb

g c c

ggc

防止segmentation fault的出现就要注意：

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

Linux lab

Linux

PREVIOUS:
« Java异常处理机制
NEXT:
» 纪念汪国真--那些曾经伴我度过岁月的诗

标签

Design pattern 4 JUnit 1 Java 9 LPR 9 Linux 5 lab 24 maven 5 opencv 1 python 6 selenium 2 testNG 1 图像融合 7 心路历程 4 性能测试 4 总结 1 日志汇总 1 测试 2 测试杂感 1 测试用例 2 瑜伽 1

分类

Java	5
Linux	4
lab	22
python	6
心路历程	4
性能测试	4

归档

友情链接

Github
Hexo
tuicool
cnblog
淘测试
百度QA
51testing
虫师
华中大在线

RSS 订阅

Hello, I am a master at hust.
This is my blog, believe you will love it.