未定义拷贝构造函数导致进程重启

秦新良

October 30, 2013

目录

1 问题描述

今天版本在测试过程中发现这样一个问题，在某个业务流程下，进程必然重启。最终定位的原因是由于某个类没有定义拷贝构造函数而引起了进程的异常退出。该问题比较隐晦，且涉及到一些编译器的默认形为，有必要记录一下，以免后续也犯同样的错误。

2 问题分析

进程异常退出，首先分析异常退出时进程的堆栈信息。通过分析确定异常是由于某个类在销毁时，其在构造函数中 delete 某个指针成员变量[1]时抛出异常，导致进程退出。释放内存系统抛出异常一般有两个原因：野指针和内存重复释放。进一步排查代码发现，该成员不可能出现野指针的情况。除此之外，那只能是内存重复释放了[2]。虽然只剩这一种情况了，但继续排查代码后，始终未发现是什么原因导致了内存的重复释放。

说了这么多，你可能看的一头雾水，让我们把修改前后的代码贴出来就一目了然了。

```
class TDataNode
{
  public:
    TDataNode()
    {
        // initialization of data members
    }

    virtual ~TDataNode()
    {
        // resource free
    }

  public:
    // declaration of member function

  private:
    // delcaraion of data members
};
```

代码 1: 修改前的类定义

```
class TDataNode
{
  public:
    TDataNode()
```

[1]该指针成员变量是新版本修改引入，未引入该变量前，业务正常。

[2]两个指针指向同一片内存，内存通过其中的一个指针释放后，再通过另一个指针释放就会导致该情况。

```
5       {
6           // initialization of data members
7
8           ...
9
10          pData = NULL;
11      }
12
13      virtual ~TDataNode()
14      {
15          // resource free
16
17          ...
18
19          if (NULL != pData)
20          {
21              delete pData;
22          }
23      }
24
25    public:
26      // declaration of member function
27
28      // assignment operator newly added
29      TDataNode& operator= (const TDataNode &dataNode)
30      {
31          if (this == &dataNode)
32          {
33              return *this;
34          }
35
36          // assignment of other data members
37
38          ...
39
40          if (NULL == pData)
41          {
42              pData = new DataType;
43              // make sure memory is successfully allocated
44          }
45
46          // copy the content of the data member
47          *pData = *(dataNode.pData);
48      }
49
50    private:
51      // delcaraion of data members
52
53      // data member newly added
54      DataType *pData;
55  };
```

代码 2: 修改后的类定义

从代码1和代码2的前后对比中可以看出，修改后的代码在类 *TDataNode* 中新增了指针类型的成员变量 *pData*，而且为了防止类的不同对象之间调用默认的赋值运算符相互赋值导致 *pData*

指向同一片内存，同时显式的重载了类的赋值运算符。排查代码过程中也发现，确实有一处代码使用了该赋值操作，如下：

```
TDataNode dataNode = dataNodeInput;
```

而其它使用到该类的场景都是通过指针或引用操作的。这样分析看来，这个赋值操作是重点怀疑对象，但还不能确定就是它引起的。

进一步重现问题和走读代码发现，该业务流程恰好会走到这个使用赋值的分支并出现异常，把代码范围缩小到这一行上之后才明白到底出了什么问题：虽然这里是调用类 *TDataNode* 的赋值运算符将 *dataNodeInput* 赋值给 *dataNode* 来完成 *dataNode* 的初始化，但在这种场景下，为了提高效率，编译器自动将该操作优化成了调用类 *TDataNode* 的拷贝构造函数直接构造出 dataNode，而不是先调用类的构造函数构造出 *dataNode*，再调用赋值运算符初始化 dataNode。我们的代码中没有显式的定义类的拷贝构造函数，这种场景下，编译器认为类 *TDataNode* 需要一个拷贝构造函数，于是自动生成一个默认的拷贝构造函数。而编译器默认生成的拷贝构造函数对指针成员变量的操作方式是浅拷贝，这就导致了两个对象的成员变量同时指向了一块内存，两个对象最终销毁时就出现了对同一块内存的重复释放，导致进程异常重启。

3 关于C++

C++ 语言强大而灵活，但为了支持这种强大和灵活，C++ 的编译器为我们做了很多工作，如生成默认构造函数、拷贝构造函数、虚函数表等。像该案例中编译器的这种优化行为并不是程序员想要的，结果导致严重的后果。如果程序员不了解编译器的这种行为，出现这种错误就再所难免，而且一旦出现，问题也极难定位。

最后感叹一句：C++ 的水真是太深了。