

C++ Core dump问题定位方法

张翔

一、 预备知识

1、 Core

在使用半导体作为内存的材料前，人类是利用线圈当作内存的材料（发明者为王安），线圈就叫作 Core ，用线圈做的内存就叫作 Core memory。如今，半导体工业蓬勃发展，已经没有人用 Core memory 了，不过在许多情况下，人们还是把内存叫作 Core。

2、 何谓 Core dump

我们在开发（或使用）一个程序时，最怕的就是程序莫名其妙地当掉。虽然系统没事，但我们下次仍可能遇到相同的问题。于是这时操作系统就会把程序当掉时的内存内容 dump 出来（现在通常是写在一个叫 core 的文件里面，不同操作系统下的名称可能不同），让我们或是调试器做为参考，进行问题定位。这个动作就叫作 Core dump。

3、 为何会发生 Core dump

前面说过，在程序当掉时出错。C/C++语言中，最常发生错误的地方就是指针有问题。您可以利用 core 文件和调试器把错误找出来。

4、 Core dump 适用范围

Core dump 适用于定位 C++程序内存错误、线程死锁等复杂问题，特别是当程序在用户机子上出现内存问题时，通过日志一般也很难定位出来（由于程序当了，写日志的线程也跟着程序结束了，错误信息很可能没有写到日志里），这种情况只有通过获取当时的 Core dump 文件来定位，没有第二种方法了。

Core dump 不适用于定位程序的逻辑错误。

5、本文范围

本文主要介绍 Windows 系统下，如何获取 C++程序的 Core dump 和如何通过 Core dump 文件定位问题。

二、 准备工作

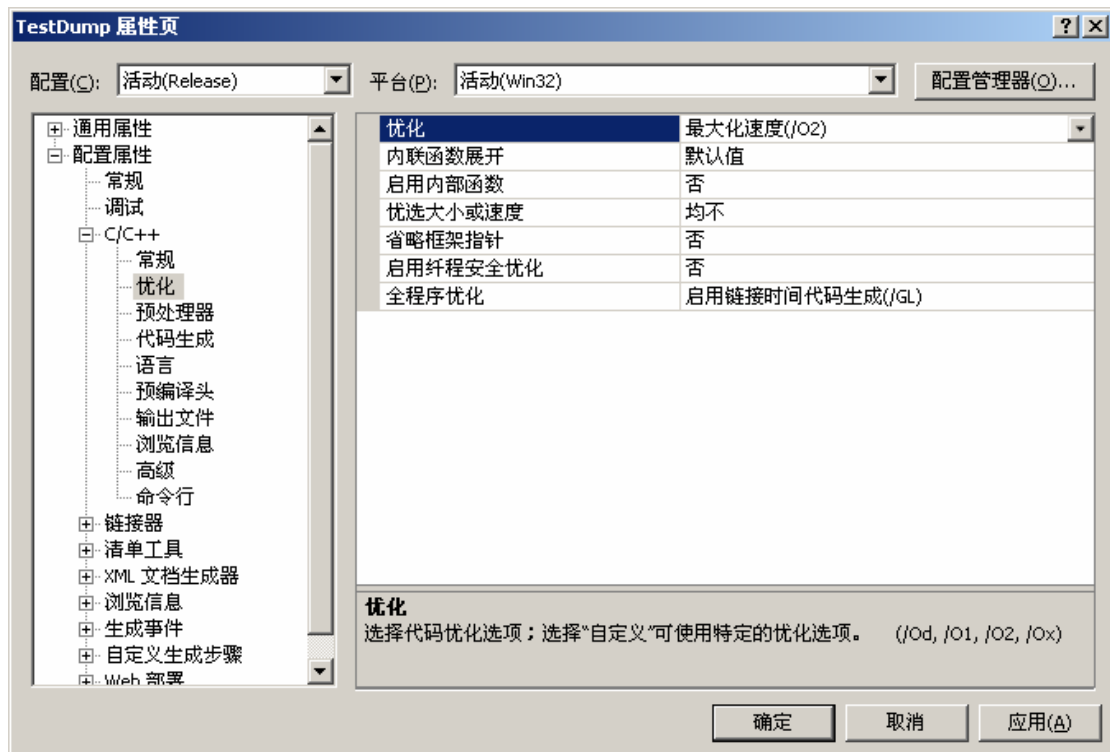
1、 调试符号

要使用 Core dump 来定位问题，程序必须要经过编译器的特殊处理，因为 Core dump 将出现问题时刻的内存映像写入文件。我们可以通过 Core dump 文件获取出现问题时的变量名称、变量值、函数调用堆栈信息（这些被称为调试符号）。但是要获取这些信息，程序在编译时要带有调试符号才可以，原因是程序编译后是二进制文件，不会携带有任何调试符号，在 windows 下，所有调试符号将写入对应的 pdb 文件。因此在使用 Core dump 时 pdb 文件是相当重要的，如果缺少 pdb 的话，即使有 Core dump 你也无法进行定位，如下图所示，由于缺少调试符号只会显示堆栈地址信息，无法显示地址对应的函数名称，这样对于定位问题没有任何帮助。

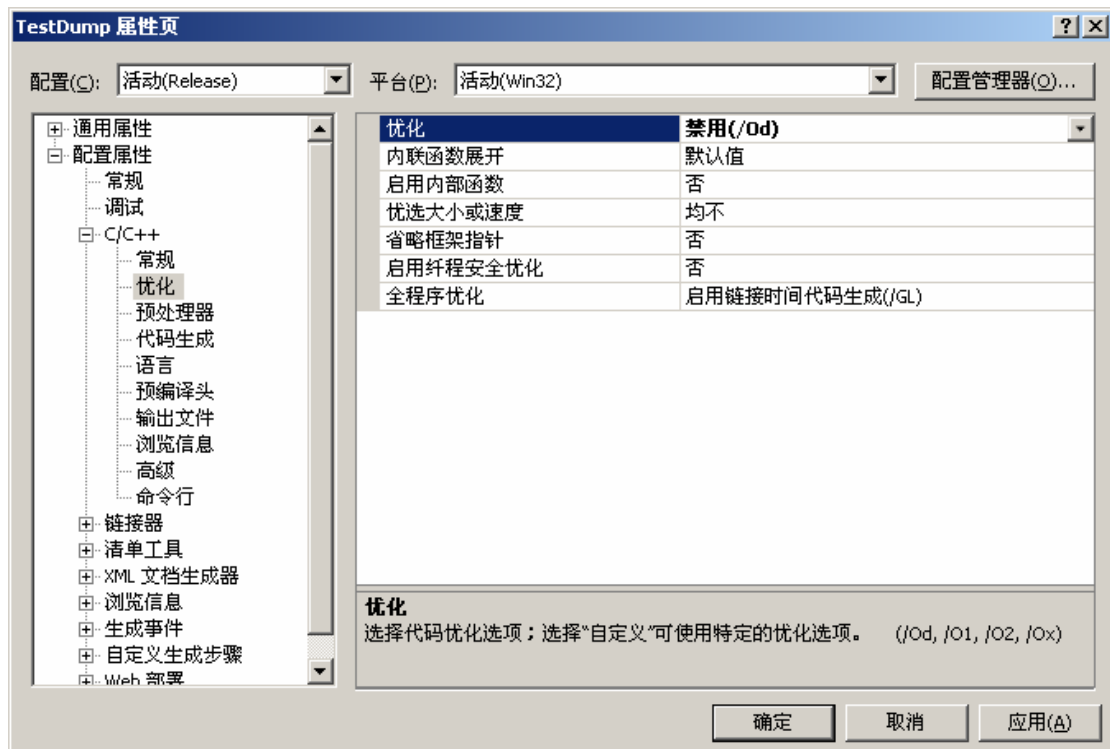
调用堆栈	
名称	
➔ TestDump.exe!004017f4()	
[下面的框架可能不正确和/或缺失，没有为 TestDump.exe 加载符号]	
TestDump.exe!00401169()	
ntdll.dll!7c93005d()	
kernel32.dll!7c816fe7()	
ntdll.dll!7c93005d()	

在编译程序时要获取正确的 pdb，需要进行如下设置（此处建议选择 release 的编译方式，对于发布给局方的版本也使用 release 编译的发布）。

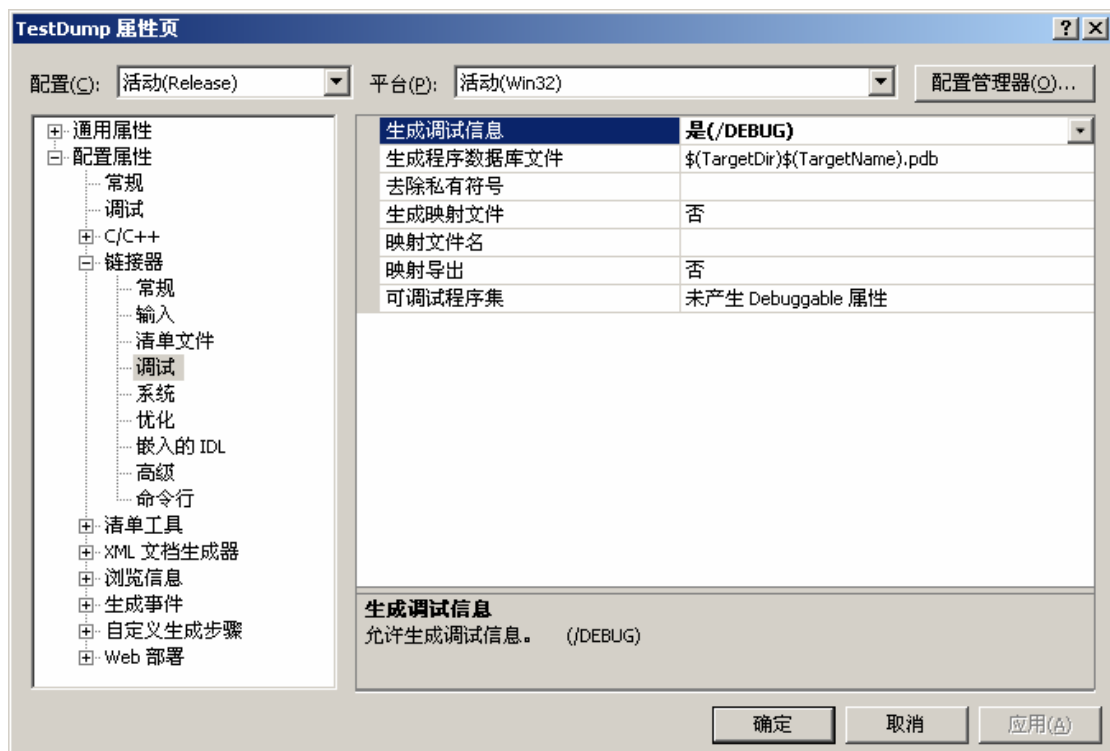
a、关闭优化开关，这种汇编级的优化对于我们目前绝大部分的程序应用没有任何帮助，而且容易造成无法通过 pdb 定位问题。



将优化方式设为“禁用”如下图所示：



b、打开调试符号生成开关，如下图所示：



2、windbg

在 Windows 下捕获程序运行时生成的 Core dump 有多种方式，比如系统自带的 Dr. Watson、Visual C++ 和其他的第三方工具等，这里使用的工具是

windbg（微软官方调试器）。进行 Core dump 的捕获。

名称约定：

客户环境：安装有应用程序的PC，比如：测试人员使用的PC、局方用户使用的工作PC，一般不会安装开发环境；

开发环境：开发人员使用的，安装有开发环境（visual c++）的PC；

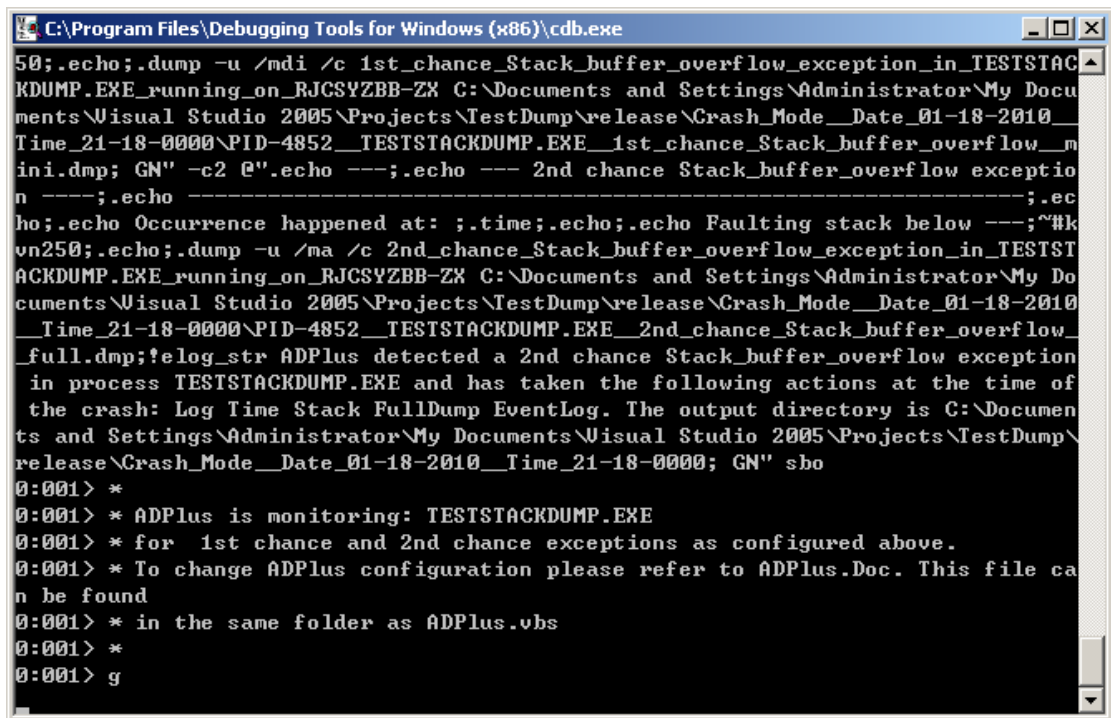
使用 Core dump 进行定位时，在客户环境上安装 windbg 即可，无须在开发环境上进行安装，因为开发环境上 windbg 的功能可以通过 visual c++ 来完成。windbg 程序小（10M 左右）、安装简单，按照默认方式安装即可。安装完目录为：“C:\Program Files\Debugging Tools for Windows (x86)”

三、 捕获 Core dump

windbg 可以使用两种工作模式：crash 和 hang 来捕获，其中在程序运行时，可以开启 windbg，以 crash 方式跟随程序一起运行，用于捕获程序运行过程中出现的随机错误，比如突然弹出的内存访问越界对话框，程序突然退出的错误等，hang 方式，主要用来获取程序运行时某个时刻的内存快照，常用于捕获线程死锁。

1、获取 Core dump

通过 windbg 里面自带的一个 vbs 脚本工具：adplus.vbs，其捕获 Core dump 步骤非常简单，先运行可执行文件（比如：DGISPowerModel.exe，可以不用登陆进去，只要是进程运行了就可以），然后进入 windbg 安装目录，执行如下命令：adplus.vbs -crash -pn DGISPowerModel.exe -o “d:\PDGPMS”，执行该命令后会出现两个提示框，直接点击掉即可，启动完 vbs 后会出现一个窗口，如下所示，这个窗口不能关闭，如果关闭程序也跟着退出，当出现这个窗口后，就以登陆建模程序，进行相关操作，当出现问题时，程序自然会退出，并产生 Core dump：



```
C:\Program Files\Debugging Tools for Windows (x86)\cdb.exe
50;.echo;.dump -u /mdi /c 1st_chance_stack_buffer_overflow_exception_in_TESTSTACKDUMP.EXE_running_on_RJCSYZBB-ZX C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects\TestDump\release\Crash_Mode__Date_01-18-2010__Time_21-18-0000\PID-4852__TESTSTACKDUMP.EXE__1st_chance_stack_buffer_overflow_m
ini.dmp; GN" -c2 0".echo ---;.echo --- 2nd chance Stack_buffer_overflow exception ---;.echo -----;.echo Occurrence happened at: ;.time;.echo;.echo Faulting stack below ---;"#kvn250;.echo;.dump -u /ma /c 2nd_chance_stack_buffer_overflow_exception_in_TESTSTACKDUMP.EXE_running_on_RJCSYZBB-ZX C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects\TestDump\release\Crash_Mode__Date_01-18-2010__Time_21-18-0000\PID-4852__TESTSTACKDUMP.EXE__2nd_chance_stack_buffer_overflow_full.dmp;!elog_str ADPlus detected a 2nd chance Stack_buffer_overflow exception in process TESTSTACKDUMP.EXE and has taken the following actions at the time of the crash: Log Time Stack FullDump EventLog. The output directory is C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects\TestDump\release\Crash_Mode__Date_01-18-2010__Time_21-18-0000; GN" sbo
0:001> *
0:001> * ADPlus is monitoring: TESTSTACKDUMP.EXE
0:001> * for 1st chance and 2nd chance exceptions as configured above.
0:001> * To change ADPlus configuration please refer to ADPlus.Doc. This file can be found
0:001> * in the same folder as ADPlus.vbs
0:001> *
0:001> g
```

命令解释如下:

参数 crash: 用于捕获 Coredump;

pn: 用于指定所需监视的进程名称;

o (小写字母): 用于指示所捕获 Core dump 文件输出目录;

由于 Core dump 写入的是出现问题时内存的信息, 因此输出的 dump 文件可能会比较大从几 M 到几 G 不定, 根据当时出现问题的具体情况, 因此在指定 Core dump 输出目录时, 最好指定在空间比较大的磁盘下。当执行 adplus 后, 在“o”参数指定的路径下, 可以看到多了一个目录, 一般目录名称为: “Crash_Mode__Date_01-18-2010__Time_18-49-4343”, 即 Crash 开头, 后接 Core dump 产生的日期、时间。里面会有“ADPlus_report.txt”、“PID-3324__DGISPOWERMODEL.EXE__Date_01-18-2010__Time_19-11-2424.log”、“Process_List.txt”三个文件, 记录了程序运行时的进程信息等, 可以用记事本打开查看。

当发生 Core dump 后, 这个目录下就会多三个以“.dmp”为后缀名的文件文件 (一般是三个, 有时也可能只有一个)。三个文件一般两个里面有包含“1st”、一个包含“2nd”的字样, 这个是跟 windows 系统异常捕获机制分为两阶段处理有关, 具体可以 Google 下, 不具体解释。这三个文件就是 Core

dump 文件，里面内容就是程序发生错误那个时刻的内存内容。

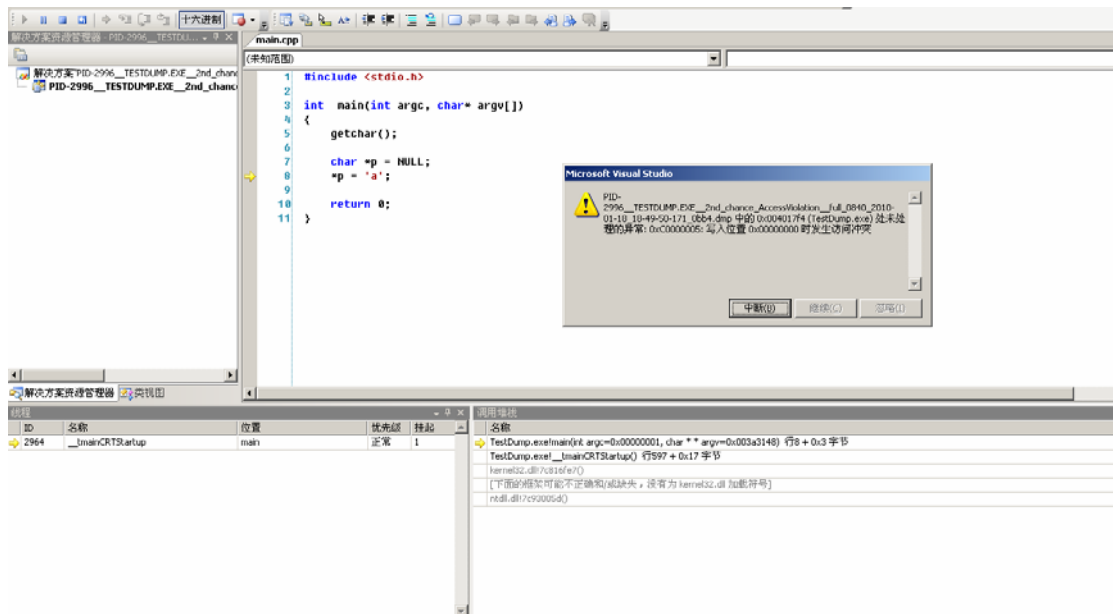
2、使用 Core dump 文件进行定位

有了 Core dump 文件后就可以进行问题定位了，Core dump 文件可以使用 Visual C++ 打开，但是光使用 VC 还是无法定位到那行代码出现问题，必须使用 pdb 文件（调试符号）+源码。具体使用步骤是：

- a、将 Core dump 文件所在的整个目录从客户环境取回；
- b、在开发环境中，创建跟客户环境一样的应用程序路径，客户环境上的应用程序放置在该路径下；然后将应用程序对应的 pdb，也一样放置在该路径下；比如客户环境下应用程序路径在“D:\GPMS 配电基层版”，那么在开发环境创建同样的目录，并把客户环境上的同样版本的应用程序放在该目录下，如果是 dll，dll 也一样放置该目录下。
- c、源码放置在当初对应得编译路径下，比如当时编译源码是工程目录为“D:\test\GPMS 配电基层版 2.1.1.0\Renderer ”，那么现在的工程也要放置在这个目录下；
- d、确认 b 和 c 的部署没有问题后，点击包含有“2nd”字样的 Core dump 文件，如果需要的话可以指定其跟 Visual C++ 编译器关联；VC 将打开该文件；
- e、打开 dmp 文件后，按 F5 或“启动调试”按钮，如果相关路径设置正确的话，程序将定位在发生错误的哪一行代码上，且发生错误的代码的相关的上下文信息，比如局部变量值等都可以查看得到。

注意：为了保证能够正确定位，防止出现调试信息未找到或未匹配的错误提示，要保证源码、pdb 和程序是同一份。

如下面这幅图：



这个是一个很简单的空指针访问错误的代码，通过打开这个程序的 Core dump 可以很容易定位到是哪个函数，那行代码出现问题，然后再分析下，这行代码附件的局部变量信息等，就可以很容易发现问题所在。

3、使用 Core dump 文件定位线程死锁问题

在 C++ 程序中有另外一种问题，就是线程死锁问题，发生线程死锁时，需要从程序逻辑和 Core dump 两方面来进行定位。当程序界面出现“挂住”情况，比如鼠标处于长时间等待状态，无法做任何操作，在排除网络通讯等问题后，可以使用 Core dump 获取线程死锁时的线程堆栈信息，在结合程序逻辑来进行判断。

比如下面这段简单代码：

```
#include <windows.h>
```

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

```
BOOL signal1 = TRUE;
```

```
BOOL signal2 = TRUE;
```

```
DWORD WINAPI ThreadProc1(LPVOID lpParameter)
```

```
{
```



```
while(signal1)
{
    if(signal2 == FALSE)
    {
        signal1 = FALSE;

        break;
    }
    else
    {
        Sleep(1);
    }
}

printf("ThreadFunc1 exit!\n");

return 0;
}
```

```
DWORD WINAPI ThreadProc2(LPVOID lpParameter)
```

```
{
    while(signal2)
    {
        if(signal1 == FALSE)
        {
            signal2 = FALSE;

            break;
        }
        else
        {
            Sleep(1);
        }
    }
}
```

```

    }
}

printf("ThreadFunc2 exit\n");

return 0;
}

int main(int argc, char* argv[])
{
    getchar();

    HANDLE thread1 = CreateThread(NULL, 0, ThreadProc1, NULL, 0, NULL);
    HANDLE thread2 = CreateThread(NULL, 0, ThreadProc2, NULL, 0, NULL);

    WaitForSingleObject(thread1, INFINITE);

    return 0;
}

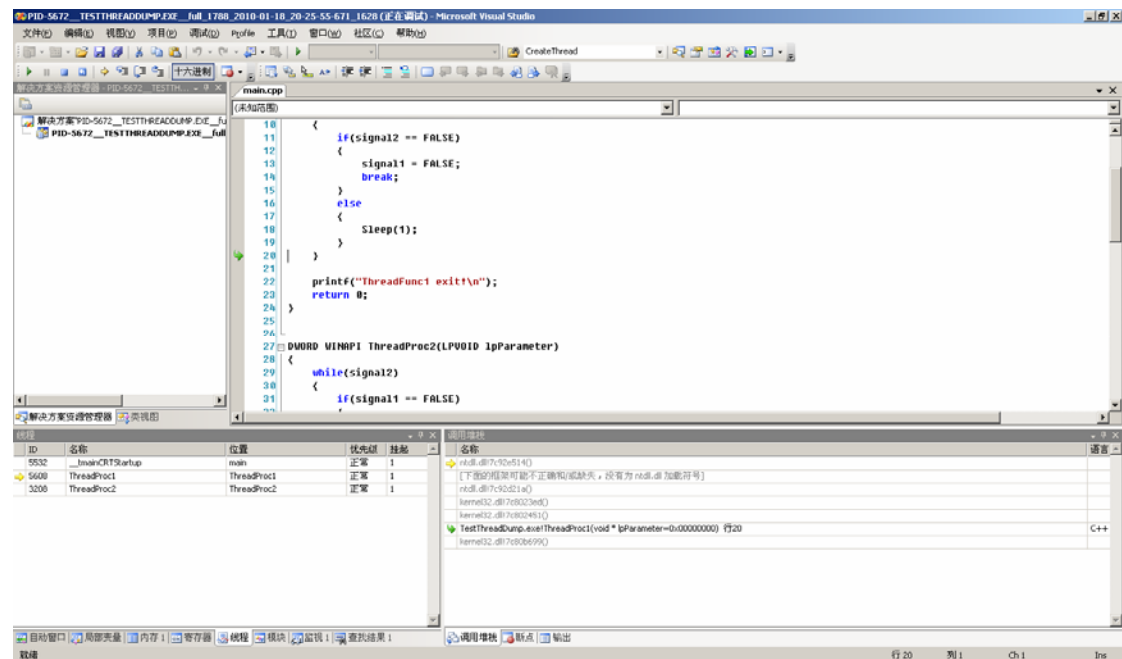
```

该程序分别启动两个线程，线程 1 和线程 2 由于都在等待对方设置信号量为 FLASE，后退出线程，互相等待导致死锁。

这时抓取 Core dump 的方法跟上面提到的一样，不同的是，这时只需要抓取内存的快照，不需要一开始就运行 adplus，只要在发现有线程死锁现象时运行 adplus，并把-crash 参数改为-hang，即可。

通过该方式将在指定目录下获取一个以 hang 开头的目录，获取目录里面的 dmp 文件即可进行定位。

下面是这个程序的定位示例：



关键是在定位时，使用左下角的线程窗口，配合旁边的堆栈窗口，获取线程死锁时有哪些线程在执行，以及他们的堆栈信息，结合程序逻辑判断即可定位线程死锁原因了。

4、定位堆栈被破坏的程序

在 C++ 程序中有一类问题最难定位就是函数堆栈被破坏的程序，这种程序没有完整的堆栈信息，即使你使用 Core dump 也不一定能完整定位出来。但是通过 Core dump 最少可以定出堆栈被破坏的范围。

下面一段程序：

```
#include <stdio.h>

#include <memory.h>

void stack(char* str)

{

    char a[10];

    memcpy(a, str, 100);

}
```

```

int main(int argc, char* argv[])
{

    getchar();

    char *a="1234567890""1234567890""1234567890""1234567890""1234567890"

        "1234567890""1234567890""1234567890""1234567890""123456789";

    stack(a);

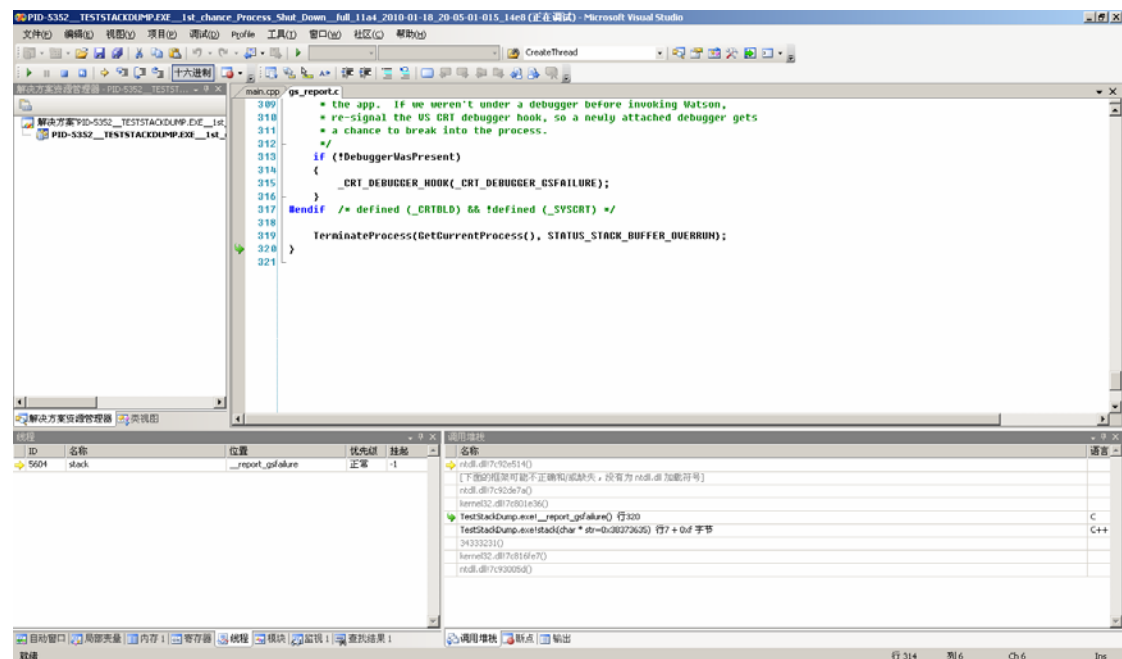
    printf("success!");

    return 0;
}

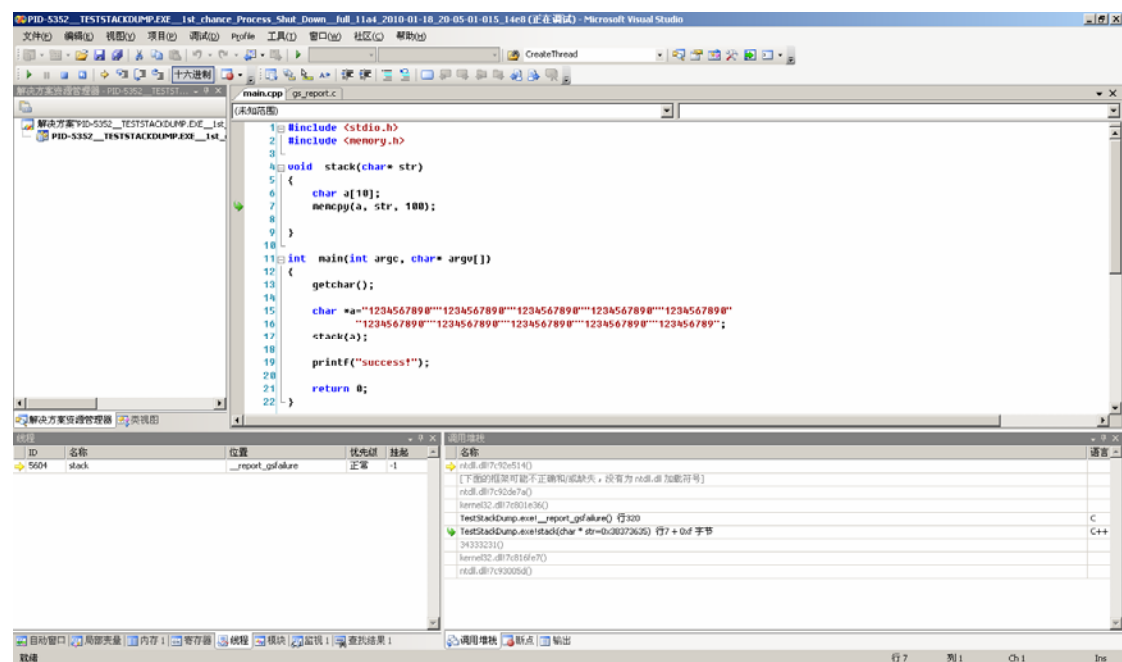
```

在 stack 函数中，进行内存拷贝时，超过了 a 数组的范围，由于 a 数组是局部变量，因此将破坏堆栈信息。当程序运行时，出现错误后，通过 adplus 可能只能获取一个 dmp 文件（根据经验一般 adplus 使用 crash 参数时，只能获取一个 dump 文件的，基本都是堆栈被破坏了）。

按照上面所说方法在开发环境打开 dmp 文件后，将获得如下信息：



程序将会定位到一个非你本工程的文件里面，通过点击下面的堆栈函数，就能基本定位出造成堆栈破坏的代码在哪一行了。



点击右下角调用堆栈，里面下一个函数（绿色箭头标示），就可以定位到哪一行了。（注：微软在新的 C 标准库中有提供这个功能保护堆栈，防止堆栈信息被破坏，如果使用 VC6.0 的编译器堆栈被破坏了就可能定位不到了）