Explanation about “pure virtual function call” on Win32 platform

作者：余祖波
Blog: http://blog.csdn.net/livelylittlefish, http://www.abo321.org

Content

0. Introduction

In this article, we will not explain why does it prompt “pure virtual function call” and how to prompt “pure virtual function call”, but detailed explain the program itself when we call a pure virtual directly/indirectly in a constructor/destructor on win32 platform. This issue on Linux platform will be introduced in another article.

At the beginning, a classical example will be shown, in this example, it will prompt a message box with “pure virtual function call”.

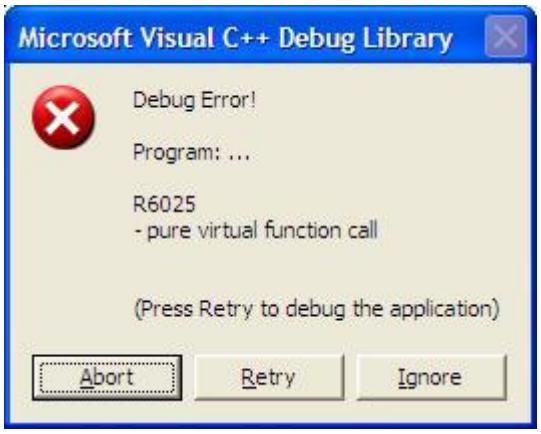
1. An example

```
/**
 * "pure virtual function call" on win32 platform
 * filename: testWin32PVFC.cpp
 */
#include <iostream>

#define PI 3.1415926
using namespace std;

class Shape
{
private:
```

```
    double ValuePerSquareUnit;

protected:
    Shape(double valuePerSquareUnit):
        ValuePerSquareUnit(valuePerSquareUnit)
    {
         //error LNK2001: unresolved external symbol "public: virtual double __thiscall Shape::area(void)const "
(?area@Shape@@UBENXZ)
        //std::cout << "creating shape, area = " << area() << std::endl;
        std::cout << "creating shape, value = " << value() << std::endl;  //indirectly call pure virtual function in
constructor
    }

public:
    virtual double area() const = 0;

    double value() const
    {
    return ValuePerSquareUnit * area();
    }

    virtual ~Shape()
    {
        printf("Shape::~Shape() is called");
    }

    double getPerSquareUnit()
    {
        return ValuePerSquareUnit;
    }
};

class Rectangle : public Shape
{
private:
    double Width;
    double Height;

public:
    Rectangle(double width, double height, double valuePerSquareUnit):
        Shape(valuePerSquareUnit),Width(width),Height(height)
    {
    }

    virtual ~Rectangle()  //can be removed
    {
    }

    virtual double area() const
```

```
    {
        return Width * Height;
    }

};

class Circle: public Shape
{
    double Radius;

public:
    Circle(double radius, double valuePerSquareUnit):
        Shape(valuePerSquareUnit),Radius(radius)
    {
    }

    virtual ~Circle()  //can be removed
    {
    }

    virtual double area() const
    {
        return PI * Radius * Radius;
    }
};

int main()
{
    Rectangle* pr = new Rectangle(30, 20, 10);
    Circle* pc = new Circle(15, 10);

    //invoke Rectangle::area()
    printf("rectangle: area = %.2f, PerSquareUnit = %.2f, value = %.2f\n", pr->area(), pr->getPerSquareUnit(),
pr->value());
    //invoke Circle::area()
    printf("circle   : area = %.2f, PerSquareUnit = %.2f, value = %.2f\n", pc->area(), pc->getPerSquareUnit(),
pc->value());

    Shape* shape;
    shape = pr;
    printf("rectangle: area = %.2f, PerSquareUnit = %.2f, value = %.2f\n", shape->area(), shape->getPerSquareUnit(),
shape->value());

    shape = pc;
    printf("circle   : area = %.2f, PerSquareUnit = %.2f, value = %.2f\n", shape->area(), shape->getPerSquareUnit(),
shape->value());

    return 0;
}
```

Running result:



From the example, we can conclude that,

In constructor/destructor,

- Direct call a pure virtual function, a compiling error will occur, such as error LNK2001: unresolved external symbol "public: virtual double __thiscall Shape::area(void)const " (?area@Shape@@UBENXZ)
- Indirect call a pure virtual function, it will prompt “pure virtual function call”

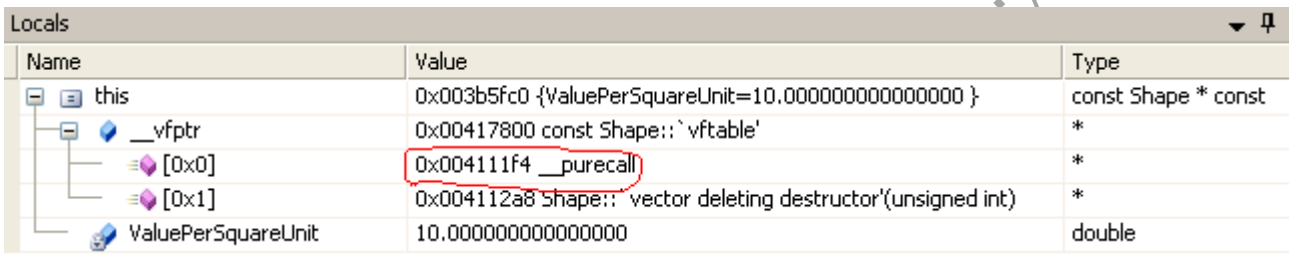
2. Disassemble code analysis

Debug this program, we can see the disassemble code of Shape::value() listed below, my comments embedded.

```
    double value() const
    {
004118F0  push        ebp
004118F1  mov         ebp,esp
004118F3  sub         esp,0CCh
004118F9  push        ebx
004118FA  push        esi
004118FB  push        edi
004118FC  push        ecx
004118FD  lea         edi,[ebp-0CCh]
00411903  mov         ecx,33h
00411908  mov         eax,0CCCCCCCCh
0041190D  rep stos    dword ptr es:[edi]
0041190F  pop         ecx
00411910  mov         dword ptr [ebp-8],ecx
    return ValuePerSquareUnit * area();
00411913  mov         eax,dword ptr [this]     //eax = 0x003b5fc0, move ‘this’ pointer to eax
00411916  mov         edx,dword ptr [eax]      //edx = 0x00417800, move vfptr to edx
00411918  mov         esi,esp
0041191A  mov         ecx,dword ptr [this]     //ecx = 0x003b5fc0, move ‘this’ pointer to ecx
0041191D  mov         eax,dword ptr [edx]      //eax = 0x004111f4, the address of __purecall, move the first virtual function address to eax
0041191F  call        eax                      //call this virtual function
```

```
00411921  cmp         esi,esp
00411923  call        @ILT+500(__RTC_CheckEsp) (4111F9h)
00411928  mov         ecx,dword ptr [this]
0041192B  fmul        qword ptr [ecx+8]
    }
0041192E  pop         edi
0041192F  pop         esi
00411930  pop         ebx
00411931  add         esp,0CCh
00411937  cmp         ebp,esp
00411939  call        @ILT+500(__RTC_CheckEsp) (4111F9h)
0041193E  mov         esp,ebp
00411940  pop         ebp
00411941  ret
```

the disassemble codes and my comments can be verified from the following figure, which was captured from debuging.



3. jump table of this program

To find the address of 0x004111f4, it is necessary to find the jump table of this program in the disassemble codes. Then, we find it listed below, which lists all jump items.

```
00411005  jmp         _setdefaultprecision (413E80h)
0041100A  jmp         _setargv (413F20h)
0041100F  jmp         std::ios_base::good (41283Ah)
00411014  jmp         DebugBreak (414B78h)
00411019  jmp         _RTC_GetErrDesc (413BE0h)
0041101E  jmp         Rectangle::area (411BD0h)
00411023  jmp         _p__fmode (413F94h)
00411028  jmp         __security_check_cookie (412870h)
0041102D  jmp         IsDebuggerPresent (414B6Ch)
00411032  jmp         std::basic_ostream<char,std::char_traits<char> >::sentry::operator bool (412630h)
00411037  jmp         type_info::operator= (412BA0h)
0041103C  jmp         _RTC_Terminate (413F60h)
00411041  jmp         WideCharToMultiByte (414B7Eh)
00411046  jmp         _RTC_AllocaHelper (412940h)
0041104B  jmp         _RTC_GetErrorFuncW (413CA0h)
00411050  jmp         _RTC_NumErrors (413BD0h)
00411055  jmp         std::basic_ios<char,std::char_traits<char> >::rdbuf (412810h)
0041105A  jmp         __setusermatherr (413F04h)
0041105F  jmp         Sleep (414B48h)
00411064  jmp         type_info::_type_info_dtor_internal_method (414B12h)
```

```
00411069  jmp         Circle::`scalar deleting destructor' (411DC0h)
0041106E  jmp         Rectangle::Rectangle (4119A0h)
00411073  jmp         std::basic_ios<char,std::char_traits<char> >::setstate (4127ECh)
00411078  jmp         GetModuleFileNameW (414BD2h)
0041107D  jmp         __security_init_cookie (414120h)
00411082  jmp         Shape::getPerSquareUnit (411960h)
00411087  jmp         Circle::`scalar deleting destructor' (411DC0h)
0041108C  jmp         SetUnhandledExceptionFilter (414B66h)
00411091  jmp         _cexit (41428Ch)
00411096  jmp         Shape::`scalar deleting destructor' (411AF0h)
0041109B  jmp         _CrtDbgReportW (414504h)
004110A0  jmp         VirtualQuery (414BD8h)
004110A5  jmp         atexit (4140E0h)
004110AA  jmp         MultiByteToWideChar (414B84h)
004110AF  jmp         FatalAppExitA (414BBAh)
004110B4  jmp         std::endl (4127E6h)
004110B9  jmp         _RTC_SetErrorType (413C00h)
004110BE  jmp         _except_handler4 (414520h)
004110C3  jmp         _lock (414B30h)
004110C8  jmp         std::basic_streambuf<char,std::char_traits<char> >::_Unlock (412852h)
004110CD  jmp         GetProcAddress (414B90h)
004110D2  jmp         std::char_traits<char>::length (412828h)
004110D7  jmp         _RTC_CheckStackVars (4128C0h)
004110DC  jmp         operator delete (412858h)
004110E1  jmp         std::char_traits<char>::eq_int_type (4127FEh)
004110E6  jmp         type_info::_type_info_dtor_internal_method (414B12h)
004110EB  jmp         std::uncaught_exception (412846h)
004110F0  jmp         __report_gsfailure (4130F0h)
004110F5  jmp         terminate (414B0Ch)
004110FA  jmp         _exit (414280h)
004110FF  jmp         GetCurrentThreadId (414BA8h)
00411104  jmp         _initterm (41450Ah)
00411109  jmp         std::basic_ios<char,std::char_traits<char> >::tie (412834h)
0041110E  jmp         std::ios_base::width (4127F2h)
00411113  jmp         GetCurrentProcess (414B5Ah)
00411118  jmp         Circle::~Circle (411E30h)
0041111D  jmp         std::basic_streambuf<char,std::char_traits<char> >::sputc (41280Ah)
00411122  jmp         std::basic_ostream<char,std::char_traits<char> >::operator<< (4127E0h)
00411127  jmp         _encode_pointer (414100h)
0041112C  jmp         std::ios_base::width (412822h)
00411131  jmp         _RTC_UninitUse (413A80h)
00411136  jmp         _RTC_Shutdown (412AD0h)
0041113B  jmp         type_info::`vector deleting destructor' (412B10h)
00411140  jmp         _FindPESection (414320h)
00411145  jmp         Rectangle::`scalar deleting destructor' (411C20h)
0041114A  jmp         _configthreadlocale (413E78h)
0041114F  jmp         _RTC_InitBase (412A90h)
00411154  jmp         _RTC_StackFailure (413700h)
00411159  jmp         LoadLibraryA (414B96h)
```

```
0041115E  jmp         RaiseException (414B72h)
00411163  jmp         _crt_debugger_hook (414550h)
00411168  jmp         _ValidateImageBase (4142A0h)
0041116D  jmp         Shape::value (4118F0h)
00411172  jmp         InterlockedCompareExchange (414B4Eh)
00411177  jmp         Rectangle::~Rectangle (411C90h)
0041117C  jmp         Shape::Shape (411A30h)
00411181  jmp         std::basic_streambuf<char,std::char_traits<char> >::_Lock (41284Ch)
00411186  jmp         std::char_traits<char>::eof (412804h)
0041118B  jmp         std::basic_ostream<char,std::char_traits<char> >::sentry::~sentry (412560h)
00411190  jmp         Shape::~Shape (411B60h)
00411195  jmp         GetProcessHeap (414BCCh)
0041119A  jmp         _RTC_SetErrorFuncW (413C60h)
0041119F  jmp         _onexit (413FA0h)
004111A4  jmp         NtCurrentTeb (412FF0h)
004111A9  jmp         HeapFree (414BC0h)
004111AE  jmp         std::operator<<<std::char_traits<char> > (411E90h)
004111B3  jmp         _RTC_SetErrorFunc (413C30h)
004111B8  jmp         _invoke_watson_if_error (413ED0h)
004111BD  jmp         std::basic_ostream<char,std::char_traits<char> >::operator<< (4127DAh)
004111C2  jmp         std::basic_ostream<char,std::char_traits<char> >::_Sentry_base::~_Sentry_base (412730h)
004111C7  jmp         TerminateProcess (414B54h)
004111CC  jmp         std::basic_ostream<char,std::char_traits<char> >::flush (41282Eh)
004111D1  jmp         mainCRTStartup (412CF0h)
004111D6  jmp         QueryPerformanceCounter (414B9Ch)
004111DB  jmp         __p__commode (413F8Eh)
004111E0  jmp         _unlock (414B24h)
004111E5  jmp         GetCurrentProcessId (414B4Eh)
004111EA  jmp         _RTC_CheckStackVars2 (412980h)
004111EF  jmp         __set_app_type (414106h)
004111F4  jmp         _purecall (412BB4h)
004111F9  jmp         _RTC_CheckEsp (412890h)
004111FE  jmp         main (4115B0h)
00411203  jmp         Rectangle::`scalar deleting destructor' (411C20h)
00411208  jmp         _RTC_Initialize (413F30h)
0041120D  jmp         _controlfp_s (414B18h)
00411212  jmp         GetSystemTimeAsFileTime (414BB4h)
00411217  jmp         _decode_pointer (414B36h)
0041121C  jmp         _invoke_watson (414B1Eh)
00411221  jmp         _RTC_GetSrcLine (414560h)
00411226  jmp         _CRT_RTC_INITW (413CA6h)
0041122B  jmp         GetTickCount (414BA2h)
00411230  jmp         std::basic_streambuf<char,std::char_traits<char> >::sputn (4127F8h)
00411235  jmp         _IsNonwritableInCurrentImage (4143B0h)
0041123A  jmp         __CxxFrameHandler3 (41286Ah)
0041123F  jmp         HeapAlloc (414BC6h)
00411244  jmp         _amsg_exit (41410Ch)
00411249  jmp         operator new (412864h)
0041124E  jmp         _XcptFilter (414286h)
```

```
00411253  jmp         _CrtSetCheckCount (414298h)
00411258  jmp         InterlockedExchange (414B42h)
0041125D  jmp         UnhandledExceptionFilter (414B60h)
00411262  jmp         std::basic_ostream<char,std::char_traits<char> >::sentry::sentry (412400h)
00411267  jmp         type_info::type_info (412AF0h)
0041126C  jmp         printf (41285Eh)
00411271  jmp         Circle::Circle (411CF0h)
00411276  jmp         _except_handler4_common (414B3Ch)
0041127B  jmp         _matherr (413F10h)
00411280  jmp         std::basic_ios<char,std::char_traits<char> >::fill (412816h)
00411285  jmp         __getmainargs (414112h)
0041128A  jmp         __ArrayUnwind (413D90h)
0041128F  jmp         Circle::area (411D70h)
00411294  jmp         lstrlenA (414B8Ah)
00411299  jmp         _RTC_Failure (413300h)
0041129E  jmp         std::ios_base::flags (41281Ch)
004112A3  jmp         _RTC_AllocaFailure (413870h)
004112A8  jmp         Shape::`scalar deleting destructor' (411AF0h)
004112AD  jmp         DebuggerKnownHandle (413230h)
004112B2  jmp         exit (414292h)
004112B7  jmp         std::basic_ostream<char,std::char_traits<char> >::_Sentry_base::_Sentry_base (412670h)
004112BC  jmp         __dllonexit (414B2Ah)
004112C1  jmp         FreeLibrary (414BDEh)
004112C6  jmp         `eh vector destructor iterator' (413CD0h)
004112CB  jmp         _initterm_e (414510h)
004112D0  jmp         std::basic_ostream<char,std::char_traits<char> >::_Osfx (412840h)
004112D5  jmp         _RTC_GetErrorFunc (413C90h)
```

Where,

```
004111F4  jmp         _purecall (412BB4h)
```

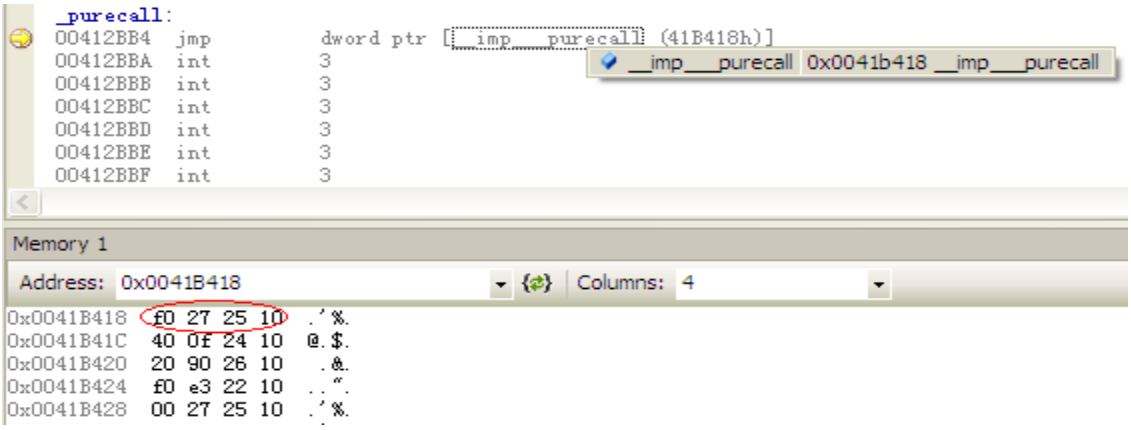
It indicates the program jump to address 0x412BB4.

4. _purecall function & its disassemble code

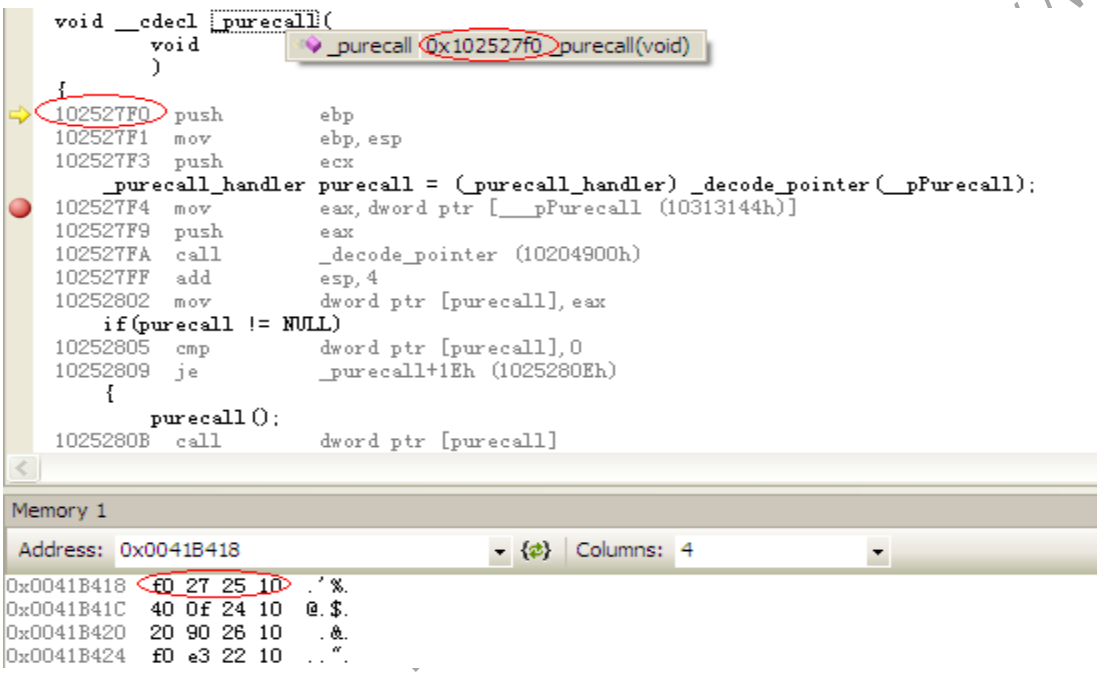
The code from 0x00412BB4 listed below, where, we can see that it is indirect addressing (间接寻址). It will jump to the content of 0x0041B418.

```
_purecall:
00412BB4  jmp         dword ptr [__imp___purecall (41B418h)]
00412BBA  int         3
00412BBB  int         3
00412BBC  int         3
00412BBD  int         3
00412BBE  int         3
00412BBF  int         3
```

From the following figure, we can see that, the content of 0x0041B418 is 0x102527f0, which is the start address of _purecall.



We go on step, then, it will jump to 0x102527f0, the start address of _purecall. From the following figure, we can see it clearly.



The disassemble code of _purecall is as follows.

```
void __cdecl _purecall(
        void
        )
{
102527F0  push        ebp
102527F1  mov         ebp, esp
102527F3  push        ecx
    _purecall_handler purecall = (_purecall_handler) _decode_pointer(__pPurecall);
102527F4  mov         eax, dword ptr [___pPurecall (10313144h)]
102527F9  push        eax
102527FA  call        _decode_pointer (10204900h)
102527FF  add         esp, 4
10252802  mov         dword ptr [purecall], eax
    if(purecall != NULL)
10252805  cmp         dword ptr [purecall], 0
```

```
10252809  je          _purecall+1Eh (1025280Eh)
    {
        purecall();
1025280B  call        dword ptr [purecall]

        /*  shouldn't return, but if it does, we drop back to
            default behaviour
        */
    }

    _NMSG_WRITE(_RT_PUREVIRT);
1025280E  push        19h
10252810  call        _NMSG_WRITE (10202AA0h)
10252815  add         esp,4
    /* do not write the abort message */
    _set_abort_behavior(0, _WRITE_ABORT_MSG);
10252818  push        1
1025281A  push        0
1025281C  call        _set_abort_behavior (10218780h)
10252821  add         esp,8
    abort();
10252824  call        abort (10218640h)
}
10252829  mov         esp,ebp
1025282B  pop         ebp
1025282C  ret
```

But who initialze __pPurecall? And what things does _decode_pointer do? This is the important thing, which will explained in the next article.

Its source code is as follows.

```
////////////////////////////////////////////////////////////////////////
//
// The global variable:
//

extern _purecall_handler __pPurecall;

/***
*void _purecall(void) -
*
*Purpose:
*       The compiler calls this if a pure virtual happens
*
*Entry:
*       No arguments
*
*Exit:
*       Never returns
```

```
*
*Exceptions:
*
*******************************************************************************/

void __cdecl _purecall(
        void
        )
{
    _purecall_handler purecall = (_purecall_handler) _decode_pointer(__pPurecall);
    if(purecall != NULL)
    {
        purecall();

        /*  shouldn't return, but if it does, we drop back to
            default behaviour
        */
    }

    _NMSG_WRITE(_RT_PUREVIRT);
    /* do not write the abort message */
    _set_abort_behavior(0, _WRITE_ABORT_MSG);
    abort();
}
```

5. where is the message from?

5.1 _RT_PUREVIRT macro

```
//file: src\rterr.h

#define _RT_PUREVIRT   25    /* pure virtual function call attempted (C++ error) */
```

5.2 _RT_PUREVIRT_TXT macro

```
//file: src\cmsgs.h

#define EOL "\r\n"
#define _RT_PUREVIRT_TXT   "R6025" EOL "- pure virtual function call" EOL
```

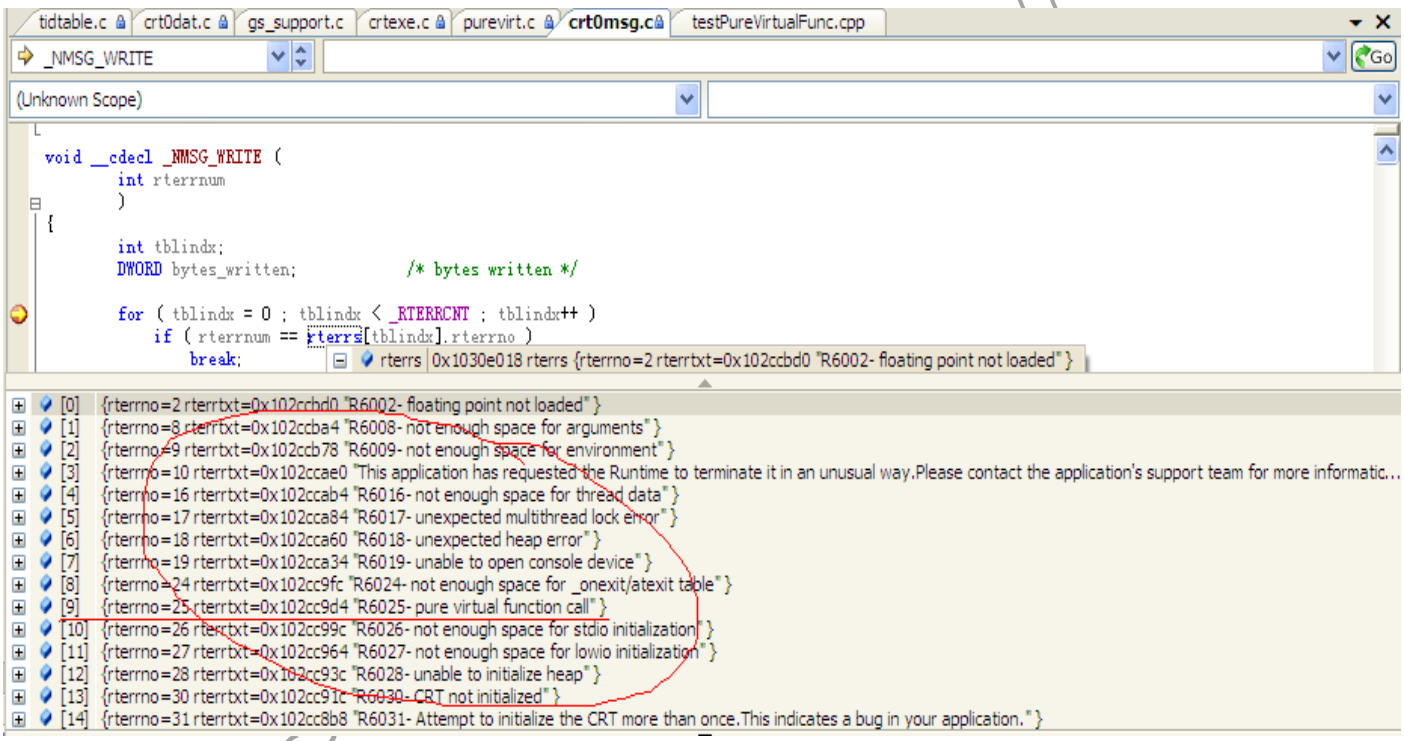
5.3 rterrs array

```
//file: src\crt0msg.c

/* struct used to lookup and access runtime error messages */
struct rterrmsgs {
        int rterrno;        /* error number */
        char *rterrtxt;     /* text of error message */
```

```
};

/* runtime error messages */
static struct rterrmsgs rterrs[] = {
        /* 2 */
        { _RT_FLOAT, _RT_FLOAT_TXT },
        /* 8 */
        { _RT_SPACEARG, _RT_SPACEARG_TXT },
        /* 9 */
        { _RT_SPACEENV, _RT_SPACEENV_TXT },
        /* 10 */
        { _RT_ABORT, _RT_ABORT_TXT },
        /* 16 */
        { _RT_THREAD, _RT_THREAD_TXT },
        /* 17 */
        { _RT_LOCK, _RT_LOCK_TXT },
        /* 18 */
        { _RT_HEAP, _RT_HEAP_TXT },
        /* 19 */
        { _RT_OPENCON, _RT_OPENCON_TXT },
        /* 22 */
        /* { _RT_NONCONT, _RT_NONCONT_TXT }, */
        /* 23 */
        /* { _RT_INVALDISP, _RT_INVALDISP_TXT }, */
        /* 24 */
        { _RT_ONEXIT, _RT_ONEXIT_TXT },
        /* 25 */
        { _RT_PUREVIRT, _RT_PUREVIRT_TXT },
        /* 26 */
        { _RT_STDIOINIT, _RT_STDIOINIT_TXT },
        /* 27 */
        { _RT_LOWIOINIT, _RT_LOWIOINIT_TXT },
        /* 28 */
        { _RT_HEAPINIT, _RT_HEAPINIT_TXT },
        ///* 29 */
        //{ _RT_BADCLRVERSION, _RT_BADCLRVERSION_TXT },
        /* 30 */
        { _RT_CRT_NOTINIT, _RT_CRT_NOTINIT_TXT },
        /* 31 */
        { _RT_CRT_INIT_CONFLICT, _RT_CRT_INIT_CONFLICT_TXT},
        /* 32 */
        { _RT_LOCALE, _RT_LOCALE_TXT},
        /* 33 */
        { _RT_CRT_INIT_MANAGED_CONFLICT, _RT_CRT_INIT_MANAGED_CONFLICT_TXT},
        /* 34 */
        { _RT_CHECKMANIFEST, _RT_CHECKMANIFEST_TXT},
        ///* 35 - not for _NMSG_WRITE, text passed directly to FatalAppExit */
        //{ _RT_COOKIE_INIT, _RT_COOKIE_INIT_TXT},
        /* 120 */
```

```
        { _RT_DOMAIN, _RT_DOMAIN_TXT },
        /* 121 */
        { _RT_SING, _RT_SING_TXT },
        /* 122 */
        { _RT_TLOSS, _RT_TLOSS_TXT },
        /* 252 */
        { _RT_CRNL, _RT_CRNL_TXT },
        /* 255 */
        { _RT_BANNER, _RT_BANNER_TXT }
};

/* number of elements in rterrs[] */
#define _RTERRCNT   ( sizeof(rterrs) / sizeof(struct rterrmsgs) )
```

This can be verified from the following figure captured from debuging.



6. which function does it prompt the message?

6.1 _NMSG_WRITE function

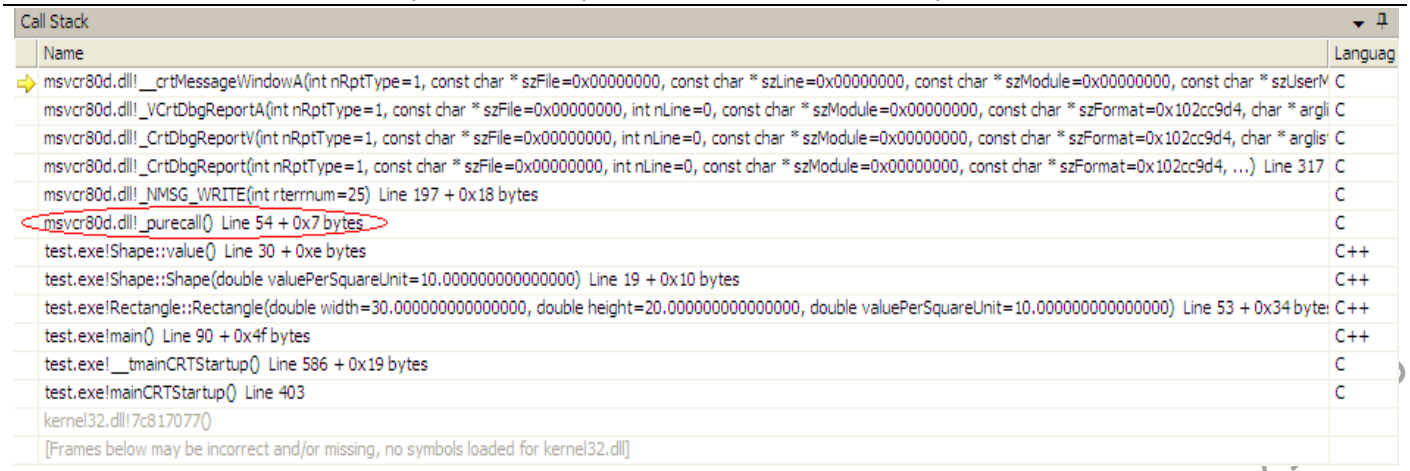
```
//file: src\crt0msg.c

/***
*__NMSG_WRITE(message) - write a given message to handle 2 (stderr)
*
*Purpose:
*       This routine writes the message associated with rterrnum
*       to stderr.
*
*Entry:
```

```
*       int rterrnum - runtime error number
*
*Exit:
*       no return value
*
*Exceptions:
*       none
*
*******************************************************************************/

void __cdecl _NMSG_WRITE (
        int rterrnum
        )
{
        int tblindx;
        DWORD bytes_written;            /* bytes written */

        for ( tblindx = 0 ; tblindx < _RTERRCNT ; tblindx++ )
            if ( rterrnum == rterrs[tblindx].rterrno )     //in rterrs array, find the mapped message
                break;

        if ( tblindx < _RTERRCNT )
        {
#ifdef _DEBUG
            /*
             * Report error.
             *
             * If _CRT_ERROR has _CRTDBG_REPORT_WNDW on, and user chooses
             * "Retry", call the debugger.
             *
             * Otherwise, continue execution.
             *
             */

            if (rterrnum != _RT_CRNL && rterrnum != _RT_BANNER && rterrnum != _RT_CRT_NOTINIT)
            {
                if (1 == _CrtDbgReport(_CRT_ERROR, NULL, 0, NULL, rterrs[tblindx].rterrtxt))
                    _CrtDbgBreak();
            }
#endif  /* _DEBUG */
            if ( (_set_error_mode(_REPORT_ERRMODE) == _OUT_TO_STDERR) ||
                 ((_set_error_mode(_REPORT_ERRMODE) == _OUT_TO_DEFAULT) &&
                  (__app_type == _CONSOLE_APP)) )
            {
                HANDLE hStdErr = GetStdHandle(STD_ERROR_HANDLE);
                if (hStdErr && hStdErr!=INVALID_HANDLE_VALUE)
                {
                    WriteFile( hStdErr,
                               rterrs[tblindx].rterrtxt,
```

```
                                (unsigned long)strlen(rterrs[tblindx].rterrtxt),
                                &bytes_written,
                                NULL );
                    }
                }
                else if (rterrnum != _RT_CRNL)
                {
                    #define MSGTEXTPREFIX "Runtime Error!\n\nProgram: "
                    static char outmsg[sizeof(MSGTEXTPREFIX) + _MAX_PATH + 2 + 500];
                        // runtime error msg + progname + 2 newline + runtime error text.
                    char * progname = &outmsg[sizeof(MSGTEXTPREFIX)-1];
                    size_t progname_size = _countof(outmsg) - (progname - outmsg);
                    char * pch = progname;

                    _ERRCHECK(strcpy_s(outmsg, _countof(outmsg), MSGTEXTPREFIX));

                    progname[MAX_PATH] = '\0';
                    if (!GetModuleFileName(NULL, progname, MAX_PATH))
                        _ERRCHECK(strcpy_s(progname, progname_size, "<program name unknown>"));

                    #define MAXLINELEN 60
                    if (strlen(pch) + 1 > MAXLINELEN)
                    {
                        pch += strlen(progname) + 1 - MAXLINELEN;
                        _ERRCHECK(strncpy_s(pch, progname_size - (pch - progname), "...", 3));
                    }

                    _ERRCHECK(strcat_s(outmsg, _countof(outmsg), "\n\n"));
                    _ERRCHECK(strcat_s(outmsg, _countof(outmsg), rterrs[tblindx].rterrtxt));

                    __crtMessageBoxA(outmsg,
                            "Microsoft Visual C++ Runtime Library",
                            MB_OK|MB_ICONHAND|MB_SETFOREGROUND|MB_TASKMODAL);
                }
            }
}
```

6.2 call stack

Step into some key functions, we can see the callstack as follows.



7. how does it be when the pure virtual function is implemented explicitly?

7.1 implement it inside of class

Modifiy the class like:

```
class Shape
{
...
public:
    virtual double area() const = 0
    {
        std::cout << "pure virtual area() called" << std::endl;
        return 0;
  }
...
};
```

There is no compiler error, it will prompt “pure virtual function call”.

7.2 implement it outside of class

Modifiy the code like:

```
class Shape
{
...
public:
    virtual double area() const = 0;
...
};

double Shape::area() const
{
    std::cout << "pure virtual area() called" << std::endl;
    return 0;
}
```

There is no compiler error, it will prompt “pure virtual function call”, too.

8. Summary

In this article, detailed explain the program itself when we call a pure virtual function directly/indirectly in a constructor/destructor on win32 platform, through a classical example. Some msvc crt source code is listed and _purecall function and its disassemble code is analyzed.

Besides, we introduced the jump table of this program, and where the prompt message is from, which is defined in a array (rterrs) and some macros such as _RT_PUREVIRT and _RT_PUREVIRT_TXT. At the last, we give a two implementations of the pure virtual function to verify whether it works or not, from the result, we find that even though there is implementation of pure virtual function, the compiler will ignore the implemented codes explicitly, _purecall still will be called, and prompt “pure virtual function call”.

Reference

msvc-crt source code
<Effective C++>