使用 CreateThread() API 创建线程

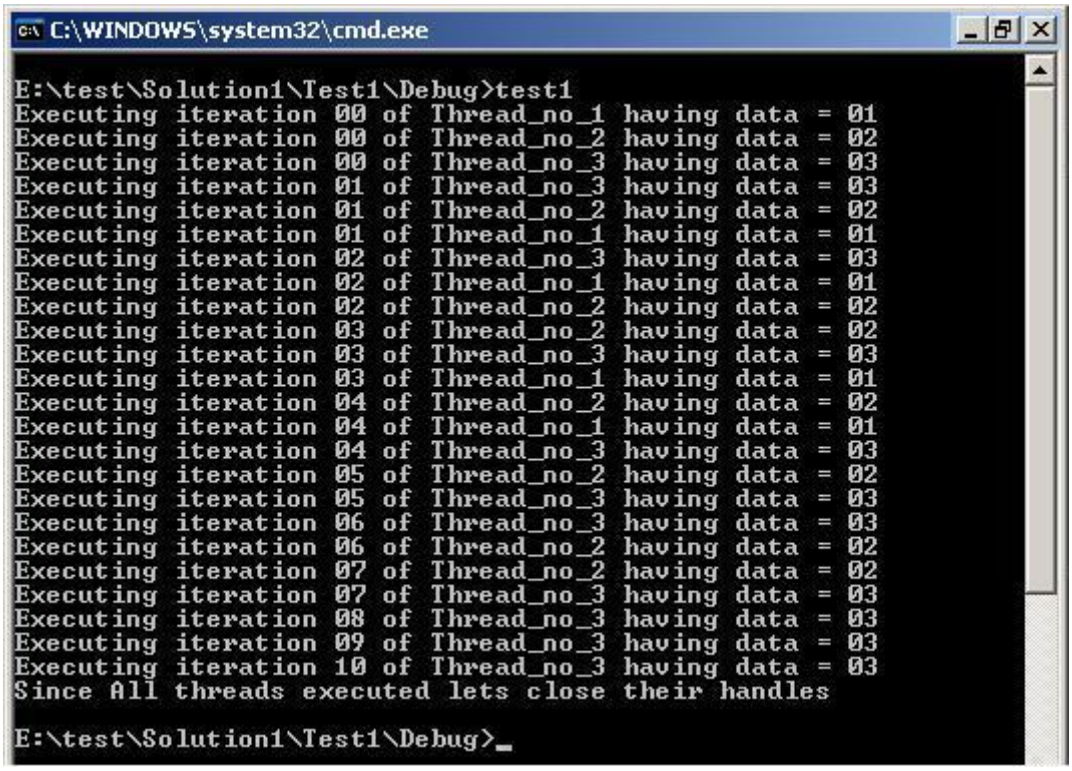
切坦·库达尔卡

2006 年 3 月 24 日

使用 CreateThread() API 创建线程。

下载演示项目 - 34.2 Kb

下载源 - 1.24 Kb



介绍

本文适用于希望参加创建线程的初步课程的初学者。本文介绍了如何使用该`CreateThread()`函数创建线程。当我开始学习多线程编程时，我很难找到简单和描述性的程序和文章，以简单明了的英语演示和解释与多线程相关的概念。这就是我为 CodeProject 撰写关于多线程的第一篇文章的动力。

背景

我正在展示一个程序，它将演示使用 Windows API 创建和并发执行三个线程`CreateThread()`。

使用代码

我在 zip 文件*source.zip* 中提供了源代码。我在*Test1.zip* 中提供了可执行文件“ *Test1.exe* ”。当您从 Windows 命令提示符运行*Test1.exe* 时，您将在 DOS 框中看到输出，如上图所示。

编码

缩小▲ 复制代码

```
#include <windows.h> <WINDOWS.H>
#include <strsafe.h> <STRSAFE.H>
#include <stdio.h><STDIO.H>

#define BUF_SIZE 255

//----------------------------------------
// A function to Display the message
// indicating in which tread we are
//----------------------------------------
void DisplayMessage (HANDLE hScreen,
     char *ThreadName, int Data, int Count)
{

    TCHAR msgBuf[BUF_SIZE];
    size_t cchStringSize;
    DWORD dwChars;

    // Print message using thread-safe functions.
    StringCchPrintf(msgBuf, BUF_SIZE,
       TEXT("Executing iteration %02d of %s"
       " having data = %02d \n"),
       Count, ThreadName, Data);
    StringCchLength(msgBuf, BUF_SIZE, &cchStringSize);
    WriteConsole(hScreen, msgBuf, cchStringSize,
                 &dwChars, NULL);
    Sleep(1000);
}

//----------------------------------------
// A function that represents Thread number 1
//----------------------------------------
DWORD WINAPI Thread_no_1( LPVOID lpParam )
{

    int     Data = 0;
    int     count = 0;
    HANDLE  hStdout = NULL;

    // Get Handle To screen.
    // Else how will we print?
    if( (hStdout =
         GetStdHandle(STD_OUTPUT_HANDLE))
         == INVALID_HANDLE_VALUE )
    return 1;

    // Cast the parameter to the correct
    // data type passed by callee i.e main() in our case.
    Data = *((int*)lpParam);

    for (count = 0; count <= 4; count++ )
    {
```

```
        DisplayMessage (hStdout, "Thread_no_1", Data, count);
    }

    return 0;
}

//---------------------------------------
// A function that represents Thread number 2
//---------------------------------------
DWORD WINAPI Thread_no_2( LPVOID lpParam )
{

    int     Data = 0;
    int     count = 0;
    HANDLE  hStdout = NULL;

    // Get Handle To screen. Else how will we print?
    if( (hStdout =
         GetStdHandle(STD_OUTPUT_HANDLE)) ==
         INVALID_HANDLE_VALUE )
    return 1;

    // Cast the parameter to the correct
    // data type passed by callee i.e main() in our case.
    Data = *((int*)lpParam);

    for (count = 0; count <= 7; count++ )
    {
      DisplayMessage (hStdout, "Thread_no_2", Data, count);
    }

    return 0;
}

//---------------------------------------
// A function that represents Thread number 3
//---------------------------------------
DWORD WINAPI Thread_no_3( LPVOID lpParam )
{
    int     Data = 0;
    int     count = 0;
    HANDLE  hStdout = NULL;

    // Get Handle To screen. Else how will we print?
    if( (hStdout =
         GetStdHandle(STD_OUTPUT_HANDLE))
         == INVALID_HANDLE_VALUE )
    return 1;

    // Cast the parameter to the correct
    // data type passed by callee i.e main() in our case.
    Data = *((int*)lpParam);

    for (count = 0; count <= 10; count++ )
    {
     DisplayMessage (hStdout, "Thread_no_3", Data, count);
    }

    return 0;
}


void main()
{
    // Data of Thread 1
    int Data_Of_Thread_1 = 1;
    // Data of Thread 2
    int Data_Of_Thread_2 = 2;
    // Data of Thread 3
```

```
    int Data_Of_Thread_3 = 3;
    // variable to hold handle of Thread 1
    HANDLE Handle_Of_Thread_1 = 0;
    // variable to hold handle of Thread 1
    HANDLE Handle_Of_Thread_2 = 0;
    // variable to hold handle of Thread 1
    HANDLE Handle_Of_Thread_3 = 0;
    // Aray to store thread handles
    HANDLE Array_Of_Thread_Handles[3];

    // Create thread 1.
    Handle_Of_Thread_1 = CreateThread( NULL, 0,
           Thread_no_1, &Data_Of_Thread_1, 0, NULL);
    if ( Handle_Of_Thread_1 == NULL)
        ExitProcess(Data_Of_Thread_1);

    // Create thread 2.
    Handle_Of_Thread_2 = CreateThread( NULL, 0,
           Thread_no_2, &Data_Of_Thread_2, 0, NULL);
    if ( Handle_Of_Thread_2 == NULL)
        ExitProcess(Data_Of_Thread_2);

    // Create thread 3.
    Handle_Of_Thread_3 = CreateThread( NULL, 0,
           Thread_no_3, &Data_Of_Thread_3, 0, NULL);
    if ( Handle_Of_Thread_3 == NULL)
        ExitProcess(Data_Of_Thread_3);


    // Store Thread handles in Array of Thread
    // Handles as per the requirement
    // of WaitForMultipleObjects()
    Array_Of_Thread_Handles[0] = Handle_Of_Thread_1;
    Array_Of_Thread_Handles[1] = Handle_Of_Thread_2;
    Array_Of_Thread_Handles[2] = Handle_Of_Thread_3;

    // Wait until all threads have terminated.
    WaitForMultipleObjects( 3,
        Array_Of_Thread_Handles, TRUE, INFINITE);

    printf("Since All threads executed"
           " lets close their handles \n");

    // Close all thread handles upon completion.
    CloseHandle(Handle_Of_Thread_1);
    CloseHandle(Handle_Of_Thread_2);
    CloseHandle(Handle_Of_Thread_3);
}
```

代码说明

我们的目标是使用 Windows API 创建并同时执行三个线程`CreateThread()`。设三个线程为`Thread_no_1`、`Thread_no_2`、和`Thread_no_3`。每个线程由一个函数表示。所以让我们命名函数。对应的函数`Thread_no_1`命名为`Thread_no_1()`。对应的函数`Thread_no_2`名为`Thread_no_2()`。对应的函数`Thread_no_3`命名为`Thread_no_3()`。因此，我们有三个函数，每个函数代表一个特定的线程。因此，当我们说三个线程并发运行时，我们的意思是三个函数同时执行。换句话说，当我们说三个线程`Thread_no_1`, `Thread_no_2`, 和`Thread_no_3`并发运行时，我们指的是三个函数`Thread_no_1()`，`Thread_no_2()Thread_no_3()`正在并发执行。因此，我们在程序中定义了三个函数。这三个功能是：

- `Thread_no_1()`
- `Thread_no_2()`
- `Thread_no_3()`

每个线程处理提供给它的一段数据或全局可用的数据。在我们的例子中，我们没有使用任何全局数据。因此，我们示例中的线程将使用提供给它们的数据。而我们程序中的线程是由一个函数来表示的。因此，向线程提供数据意味着向函数提供数据。我们如何向函数

提供数据？通过将参数传递给函数。因此，我们表示线程的函数通过参数接受数据。所以现在，代表线程的每个函数看起来像这样.....

- `Thread_no_1(LPVOID lpParam)`
- `Thread_no_2(LPVOID lpParam)`
- `Thread_no_3(LPVOID lpParam)`

其中“`LPVOID lpParam`”是指向 a 的长指针`void`。

谁必须将数据传递给这些线程，即传递给线程函数？嗯....它是由 Windows API 完成的`CreateThread()`。稍后将讨论如何做到这一点。想知道谁调用了`CreateThread()`API？它`main()`是调用`CreateThread()`API 以简单地创建线程的程序。程序执行从哪里开始？它从 开始`main()`。因此，`main()`调用`CreateThread()`. 该函数`CreateThread()`创建线程。线程并发执行并终止。这就是故事！

现在让我们了解线程的实现，即线程函数。考虑第一个线程函数，`Thread_no_1`。

Thread_no_1()：

该函数的原型是“`DWORD WINAPI Thread_no_1( LPVOID lpParam )`”。请不要对此提出任何问题！它必须是这样的。让我们以初学者的水平接受它。该函数接受以 long void 指针变量形式提供给它的数据`lpParam`。它定义了两个整数变量，`Data`和`count`. 它定义了一个`hStdout`数据类型的变量`HANDLE`。接下来，该函数使用函数“`GetStdHandle()`”获取屏幕句柄（标准输出）。如果无法获得屏幕句柄，则线程函数返回。接下来，该函数从中提取数据`lpParam`并将其存储在变量 中`Data`。该数据`Thread_no_1()`通过由`CreateThread()`调用的函数传递给`main()`。接下来，该函数实现了一个`for`运行四次的循环。功能`DisplayMessage()`从`for`循环内调用。因此该函数`DisplayMessage`执行四次。它的工作是显示一个字符串，该字符串指示正在执行的线程、迭代次数以及传递给线程的数据。

Thread_no_2()：

该函数的原型是“`DWORD WINAPI Thread_no_2( LPVOID lpParam )`”。请不要对此提出任何问题！它必须是这样的。让我们以初学者的水平接受它。该函数接受以 long void 指针变量形式提供给它的数据`lpParam`。它定义了两个整数变量，`Data`和`count`. 它定义了一个`hStdout`数据类型的变量`HANDLE`。接下来，该函数使用函数“`GetStdHandle()`”获取屏幕句柄（标准输出）。如果无法获得屏幕句柄，则线程函数返回。接下来，该函数从中提取数据`lpParam`并将其存储在变量 中`Data`。该数据`Thread_no_2()`通过由`CreateThread()`调用的函数传递给`main()`。接下来，该函数实现了一个`for`运行七次的循环。功能`DisplayMessage()`从`for`循环内调用。因此该函数`DisplayMessage`执行七次。它的工作是显示一个字符串，该字符串指示正在执行的线程、迭代次数以及传递给线程的数据。

Thread_no_3()：

该函数的原型是“`DWORD WINAPI Thread_no_3( LPVOID lpParam )`”。请不要对此提出任何问题！它必须是这样的。让我们以初学者的水平接受它。该函数接受以 long void 指针变量形式提供给它的数据`lpParam`。它定义了两个整数变量，`Data`和`count`. 它定义了一个`hStdout`数据类型的变量`HANDLE`。接下来，该函数使用函数“`GetStdHandle()`”获取屏幕句柄（标准输出）。如果无法获得屏幕句柄，则线程函数返回。接下来，该函数从中提取数据`lpParam`并将其存储在变量 中`Data`。该数据`Thread_no_3()`通过由`CreateThread()`调用的函数传递给`main()`。接下来，该函数实现了`forDisplayMessage()`从内部调用的`for`环形。因此该函数`DisplayMessage`执行十次。它的工作是显示一个字符串，该字符串指示正在执行的线程、迭代次数以及传递给线程的数据。

现在，让我们考虑函数`DisplayMessage()`。

显示消息（）：

此函数不返回任何内容。该函数接受四个参数。第一个参数是屏幕句柄，第二个参数是线程的名称，第三个参数是线程的数据，最后一个参数是线程正在执行的迭代次数。此函数声明一个缓冲区`msgbuff`来保存要在屏幕上显示的消息。该函数声明`cchStringSize`保存要显示的消息的大小。该函数声明`dwChars`为`DWORD`. 它是`WriteConsole`函数所必需的。该函数`StringCchPrintf()`复制要在 中显示的消息`msgbuff`。该函数`StringCchLength()`计算要显示的消息的长度。该函数`WriteConsole()`显示消息。显示消息后，它会休眠一秒钟。

重申一下，我们程序中每个线程的工作是每秒显示一条消息。

转到`main()`程序。

主程序：

主程序的目标是创建三个线程并让它们并发运行直到它们终止。使用该CreateThread()函数创建线程。当CreateThread()函数创建一个线程时，它返回一个线程句柄。我们的主程序必须存储这个句柄。由于我们创建了三个线程，我们的程序需要存储三个线程句柄，所以我们的程序定义了三个句柄变量，即。Handle_Of_Thread_1、Handle_Of_Thread_2、 和Handle_Of_Thread_3。我们的程序还定义了三个数据变量，它们将包含要传递给线程的数据。因此，变量的内容Data_Of_Thread_1将传递给第一个线程，即Thread_no_1(). 变量的内容Data_Of_Thread_2会被传递给第二个线程，即Thread_no_2()，以及变量的内容Data_Of_Thread_3将传递给第三个线程，即Thread_no_3。接下来，我们声明一个数组来保存三个线程句柄。数组的名称是Array_Of_Thread_Handles[3]。稍后将讨论此数组的用途。

接下来，尝试通过调用函数来创建第一个线程CreateThread()。CreateThread接受六个参数。我们的目标是创建一个简单的线程。因此，我们将重点关注CreateThread()函数的第三个和第四个参数。函数的地址Thread_no_1()作为第三个参数传递。变量的地址Data_Of_Thread_1作为第四个参数传递。这就是我们向线程传递数据的方式。该CreateThread()函数创建一个线程，该线程开始执行。该函数CreateThread()返回Thread_no_1的句柄。此句柄收集在句柄变量中Handle_Of_Thread_1。如果NULL返回一个值，程序将退出，退出值为Data_Of_Thread_1。

接下来，尝试通过调用函数来创建第二个线程CreateThread()。我们的目标是创建一个简单的线程。函数的地址Thread_no_2()作为第三个参数传递。变量的地址Data_Of_Thread_2作为第四个参数传递。这就是我们向线程传递数据的方式。该CreateThread()函数创建一个线程，该线程开始执行。该函数CreateThread()返回Thread_no_2的句柄。此句柄收集在句柄变量中Handle_Of_Thread_2。如果NULL返回一个值，程序将退出，退出值为Data_Of_Thread_2。

接下来，尝试通过调用函数来创建第三个线程CreateThread()。函数的地址Thread_no_3()作为第三个参数传递。变量的地址Data_Of_Thread_3作为第四个参数传递。这就是我们向线程传递数据的方式。该CreateThread()函数创建一个线程，该线程开始执行。该函数CreateThread()返回Thread_no_3的句柄。此句柄收集在句柄变量中Handle_Of_Thread_3。如果NULL返回一个值，程序将退出，退出值为Data_Of_Thread_3。

此时，所有三个线程都在并发执行，您可以在屏幕上看到如上图所示的输出。Thread_no_1有一个for循环四次。Thread_no_2有一个for循环七次的循环。Thread_no_3有一个for循环十次的循环。因此，Thread_no_1将首先Thread_no_2完成，然后完成，Thread_no_3最后完成。

WaitForMultipleObjects()

现在所有三个线程都在并发执行，您可以在屏幕上看到如上图所示的输出。我们等待让所有线程执行，然后我们应该退出我们的程序。为此，我们需要调用函数WaitForMultipleObjects()。对于这个函数，我们需要传递我们希望等待的所有线程的句柄。这个函数要求我们将这些句柄存储在一个数组中，并将这个数组作为第二个参数传递给函数WaitForMultipleObjects()。所以我们定义了一个数组Array_Of_Thread_Handles[3]。我们将三个线程的句柄存储在这个数组中，并将这个数组作为第二个参数传递给函数WaitForMultipleObjects()。这个函数的第一个参数表明我们正在等待三个线程。第三个参数是TRUE. 它表示等待所有线程。最后一个参数作为INFINITE. 这意味着，等待所有线程完成。

因此此时，主程序等待所有线程完成它们的执行。同时，如上图所示，线程同时运行。当所有线程都完成后，控制权转移到主程序。程序打印语句“既然所有线程都执行了，让我们关闭它们的句柄”，然后继续关闭线程的句柄并退出。

我希望这个程序使线程的创建变得清晰。如果您有任何问题，请告诉我。我希望我能回答他们。

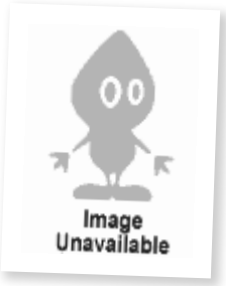
历史

- 2006 年 3 月 19 日：初始版本。

执照

分享

关于作者



切坦·库达尔卡
软件开发人员（高级）
印度
手表该会员

我是一名软件工程师，拥有大约 7 年以上的经验。我的大部分经验是在存储技术方面。

评论和讨论

添加评论或问题 电子邮件提醒 Search Comments

第一 页上一页 下一页

谢谢
steph78_0 26-Nov-14 20:34

谢谢你
Ibrahim Alwawi 19-Jan-14 22:17

我的3票
Stefano Capelli 6-Nov-12 21:53

Handle/WaitForMultiple...() 逻辑比它需要的更复杂
H.Brydon 29-Jun-12 10:16

回复：Handle/WaitForMultiple...() 逻辑比它需要的更复杂
Chetan Kudalkar 16-Oct-12 19:45

回复：Handle/WaitForMultiple...() 逻辑比它需要的更复杂
H.Brydon 25-Oct-12 11:26

我的5票
MadhureshK 28-Jun-12 19:19

请在顶部添加提示以使用 _beginthreadex
Seikilos 30-Mar-12 15:17

错误：
gniktihsrah 13-Jun-11 18:26

我的一票
Rajesh R Subramanian 3-Nov-10 1:02

使用 Borland C++ 编译
Kelly Cristina Mara 12-Mar-08 7:02

非常感谢
abo_obayd 12-Feb-08 18:55

回复：非常感谢
Christiaan Rakowski 3-Apr-09 18:31

线程ID
cooquoolive 14-Jul-07 20:08

如何检查线程的状态?
AlexZherdev 8-Feb-07 10:48

Re: 如何查看线程的状态?
azonenberg 20-Jan-09 1:17

暂停，优先
Kumar Sundaram 4-Jan-07 21:01

伟大的
Kumar Sundaram 4-Jan-07 20:58

回复：太好了
Chetan Kudalkar 8-Jan-07 15:32

CloseHandle 函数问题
Like2Byte 2-Nov-06 5:37

Re: CloseHandle 函数问题
azonenberg 20-Jan-09 1:19

优秀
Waldermort 27-Mar-06 0:27

回复：优秀
Chetan Kudalkar 12-Aug-06 21:17

较新的 WIN32 函数来创建和关闭线程
technoway 24-Mar-06 23:45

回复：较新的 WIN32 函数来创建和关闭线程
Nemanja Trifunovic 25-Mar-06 2:24

刷新 1 2 下一个 ▷

一般 新闻 建议 问题 错误 答案 笑话 赞美 咆哮 管理员

使用Ctrl+Left/Right 切换消息，Ctrl+Up/Down 切换主题，Ctrl+Shift+Left/Right 切换页面。