


[articles](#) [Q&A](#) [forums](#) [lounge](#)Search for articles, questions, tips 

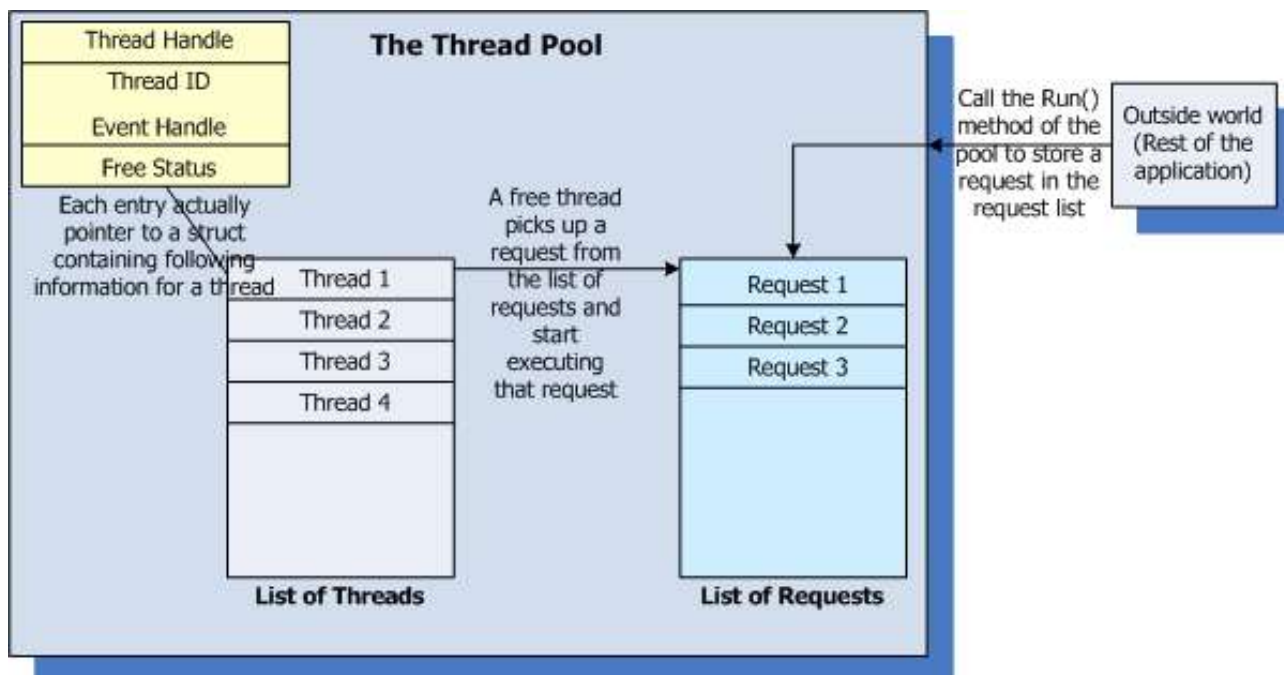
# Win32 Thread Pool

**Siddharth R Barman**, 7 Feb 2011

4.75 (41 votes)

Rate this:

An implementation of a Thread Pool in C++ for Windows

[Download demo project and source - 71.67 KB](#)

## Introduction

What is a thread pool? Exactly that, a pool of threads. You may have heard of terms like object pooling, thread pooling, car pooling (oops), anyway, the idea behind pooling is that you can re-use the objects which may be threads or database connections or instances of some class. Now why would we ever want to re-use such things? The most important reason would be that creating such an object might take up a lot of resources or time, so, we do the next best thing. We create a bunch of them initially and call the bunch a pool. Whenever someone (some code) wants to use such an object, instead of creating a new one, it gets it from the already existing bunch (pool).

# Latest Updates

The code for this was written quite a while back - Sept 2005! After seeing recent action on this article, I decided to update it. Thanks to everyone who has appreciated, asked questions and most of all pointed out defects in the code.

**The latest updates are listed below:**

1. The **shutdown** event is not a named event to allow creation of multiple thread pools across processes.
2. Individual thread wait handles are named as **PID:** <processid />, **TDX:** <threadindex />.
3. Fixed **Create()** method to check **INVALID\_HANDLE\_VALUE** when using **\_beginthreadex**.
4. Changed **Create()** method to return **false** if we fail to create any of the threads.
5. Added virtual destructor to **IRunObject** structure.
6. Modified **Destroy()** to delete **IRunObject** object is **AutoDelete()** is **true**.
7. The sample application has been revamped. I must admit it wasn't very understandable. It's been made lot more simpler and demonstrates the use of the thread-pool better.

## Background

Usually when I develop something, there is a fairly good reason behind it like my project demanded it or I did it for someone who asked me about it, but this thread pool was a result of neither of these. Actually, around a year back, I had attended an interview with a company where I was asked this question. Well at that time, I didn't really think of it seriously, but a few days back the question came back to me, and this time I decided it was time to get my answer working.

## Using the Code

### Main Files

- *threadpool.h* & *threadpool.cpp* - defines the **CThreadPool** class. This class is the main class which provides the thread pooling facilities.
- *ThreadPoolAppDlg.cpp* - see the **OnOK()** method. This method makes use of the thread pool.

This example was written using VC++ 6.0. The source has now been updated for Visual Studio 2008. The thread pool class itself does not make use of any MFC classes so you should be able to use the thread pool for pure Win32 applications also.

### A Little Explanation of How the Pool Works

First, you need to create an instance of the thread pool. While creating it, you have the option of specifying the number of threads which will live in the pool. The default is **10**. You can also specify whether to create the pool right now or later. If you choose to create it later, you will need to call the **Create()** method to actually create the threads in the pool.

Once the pool is created, the following things have already happened:

1. The specified number of threads have been created.
2. These threads have been added to an internal list of threads.
3. A list has been created which will store the user requests for execution.
4. All the threads are waiting eagerly for someone to add a work item for execution.

Once a new item has been given to the thread pool, the pool looks for the first available (free) thread. It sets off an event which makes the thread go and pop off the work item from the list of requests and immediately go about executing the request. Once execution finishes, the thread marks itself as 'free' and checks the request-list once for any pending requests. This keeps happening for the entire lifetime of the pool.

## Using the Pool

Once the pool is created, you can give it work to do using two different ways.

## Method 1

Write a function of the following form:

Hide Copy Code

```
DWORD WINAPI MyThreadFunc1(LPVOID param)
{
    // VERY IMPORTANT: param is a pointer to UserPoolData
    UserPoolData* poolData = (UserPoolData*)param;
    // NOTE: To get to YOUR data, you need to retrieve it from poolData
    LPVOID myData = poolData->pData;
    // do my work
    // ...
    // If you are doing Long running processing, you should check if the pool
    // is getting destroyed:
    while(poolData->pThreadPool->CheckThreadStop() == false)
    {
        // keep doing work as long as CheckThreadStop() keeps returning false
    }
}
```

Once this is done, suppose the instance of the thread pool is **gThreadPool**, then call the **Run()** method of the pool.  
E.g.:

Hide Copy Code

```
CMyOwnData* PointerToMyOwnData = new CMyOwnData();
gThreadPool.Run(MyThreadFunc1, (void*)PointerToMyOwnData);
```

So, in this case what we specify are:

- the function to execute
- the data which will be passed into the function while it executes in a 'new' thread.

See the **LPVOID param** in the **MyThreadFunc1** definition?

## Method 2

Write your own class but make sure it derives from **IRunObject** (declared in *RunObject.h*). E.g.:

Hide Shrink ▲ Copy Code

```
class CRunner : public IRunObject
{
public:
    HWND m_hWnd;

    void Run()
    {
        CListBox list;
        list.Attach(m_hWnd);
        list.ResetContent();
        for(int nIndex=0; nIndex < 10000; nIndex++)
        {
            // Check if the pool want us to stop
            if(this->pThreadPool->CheckThreadStop())
            {
                break;
            }

            list.AddString(_T("Item"));
            Sleep(1000);
        }
    }
}
```

```

        list.Detach();

        delete this;
    }

    bool AutoDelete()
    {
        return false;
    }
};

```

Once you derive a class from **IRunObject** you need to write code for the **void Run()** method and also for the **bool AutoDelete()** method. Write your business logic in the **Run** method. When the pool picks up your object, it will call its **Run()** method. After the **Run()** method finishes, the pool will call the **AutoDelete()** method. If this method returns **true**, the pool will use C++ **delete** to free the object. If the method returns **false**, the pool will not do anything to the object. Finally, you need to add the object to the queue. E.g.:

Hide Copy Code

```

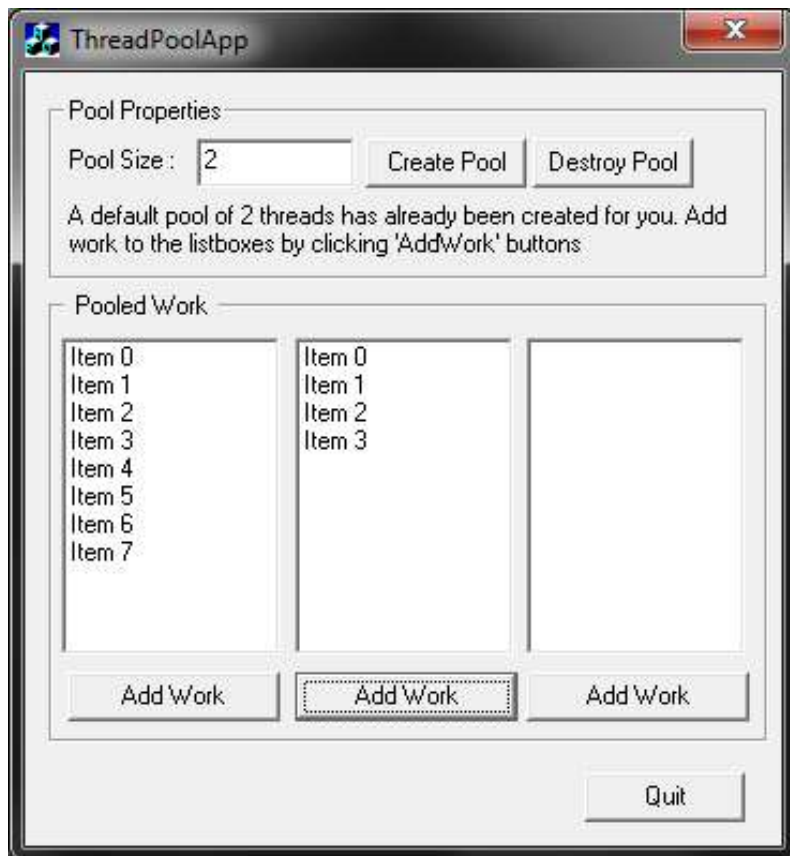
CRunner* runner = new CRunner();
runner->m_hWnd = hListBox3;
gThreadPool.Run(runner); // adding to the pool

```

To run the demo application, go to the *bin* folder. Rename the file *ThreadPoolApp.exe.txt* to *ThreadPoolApp.exe* and double-click it.

If for some reason the executable is not present, please build the source code. The source has been compiled using Visual Studio 2008 SP1.

## Explanation of the Demo



The demo application is a simple dialog based MFC application. It's got three buttons and three list boxes:

- The dialog has a thread-pool of size 2 threads created. The work that is submitted is filling the list boxes with numbers from 0 to 9. To add work to fill the first list-box, click the 'Add Work' button under it. Similarly, it is possible

to add work for the other list-boxes. Since the initial threadpool contains only 2 threads, if you queue up work items for all the 3 list-boxes, you see the last list-box getting filled only after the first two list-boxes have been filled.

- 'Create Pool' button: Clicking this button just destroys the existing pool and creates a new one. Enter the number of thread you want for the new pool and click this button. This will cause the existing threadpool to go into a 'destroying' state. The user-code which fills the listboxes check for the state by calling `CheckThreadStop()` on the threadpool object. If `true`, the user-code halts its execution.
- 'Destroy Pool' button: Clicking this button destroys the existing pool. All work items stop their execution. All threads are destroyed. Adding new work items have no effect since the pool itself is destroyed and work-items are no longer queued.
- 'Quit' button: Closes the application.

## History

- 17<sup>th</sup> October, 2005: Initial post
- 31<sup>st</sup> March, 2010: Article updated
- 8<sup>th</sup> January, 2011: Article updated
  - Fixed some memory related bugs
  - Added ability for user-code to react to pool-destruction event
  - Added new functions `int GetWorkingThreadCount()` and `bool CheckThreadStop()` to `CThreadPool` class
  - Updated code to run on Visual Studio 2008
- 4<sup>th</sup> February, 2011: Updated source code

C++ is great fun. Hope you all enjoyed this article as much as I had fun writing it.

Please mail your comments to [siddharth\\_b@yahoo.com](mailto:siddharth_b@yahoo.com). I'd love to hear from you.

## License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOl\)](#)

## Share

TWITTER

FACEBOOK

## About the Author



# Siddharth R Barman

Software Developer (Senior)

United States 

My personal website is at <http://sbytestream.pythonanywhere.com>

## You may also be interested in...

[Smart Thread Pool](#)

[Optimizing Traffic for Emergency Vehicles using IOT and Mobile Edge Computing](#)

[Win32 Thread Pools and C++11 : A quick wrapper](#)

[Get Started Turbo-Charging Your Applications with Intel® Parallel Studio XE](#)

[Build an Autonomous Mobile Robot with the Intel® RealSense™ Camera, ROS, and SAWR](#)

[SAPrefs - Netscape-like Preferences Dialog](#)

## Comments and Discussions

You must [Sign In](#) to use this message board.

Search Comments



[First](#) [Prev](#) [Next](#)

---

**Found a issue: release compile with vs2012 on windows 8.1.**   
**spring410** 10-Jul-14 17:26

Re: Found a issue: release compile with vs2012 on windows 8.1.   
**spring410** 10-Jul-14 21:27

---

**Does this program support Parallel Programming means if you have multicore machines**   
**Shailspa** 8-Apr-14 7:21

---

**My vote of 5**   
**lawsonshi** 16-May-13 23:52

---

**QueueUserWorkItem** 

shariqmuhammad 29-Jan-13 12:42

Re: QueueUserWorkItem 

Dave Calkins 27-Feb-13 5:08

Re: QueueUserWorkItem 

Siddharth R Barman 27-Feb-13 6:05

Re: QueueUserWorkItem 

Dave Calkins 27-Feb-13 6:26

Re: QueueUserWorkItem 

Siddharth R Barman 27-Feb-13 6:30

---

When all threads are busy, does it work well? 

saintwang 19-May-11 17:26

---

Multiple thread pools in a single application... Is it possible? 

Dragan Knezevic 2-Feb-11 7:33

Re: Multiple thread pools in a single application... Is it possible? 

Siddharth R Barman 4-Feb-11 7:13

Re: Multiple thread pools in a single application... Is it possible? 

tomajabvs 5-Feb-11 4:44

---

My vote of 2 

sheds 11-Jan-11 23:21

---

A little refactoring 

glabute 29-Sep-10 9:09

Re: A little refactoring 

pophelix 30-Dec-10 15:47

Re: A little refactoring 

lijianli 11-Apr-12 20:46

Re: A little refactoring 

Siddharth R Barman 13-Apr-12 6:00

Re: A little refactoring 

BianChengNan 25-Mar-13 21:08

---

Shutdown Event 

DarkOne1 26-Sep-10 20:59

Re: Shutdown Event 

glabute 29-Sep-10 8:05

Re: Shutdown Event 

Siddharth R Barman 4-Feb-11 7:15

---

My vote of 5 

crysis168 8-Aug-10 21:41

---








thread state error 

Re: thread state error 

**Siddharth R Barman** 21-Jul-08 6:35

[Refresh](#)

1 2 Next »

 General  News  Suggestion  Question  Bug  Answer  Joke  Praise  Rant  Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

[Permalink](#) | [Advertise](#) | [Privacy](#) | [Terms of Use](#) | Mobile    Layout: [fixed](#)    Article Copyright 2005 by Siddharth R Barman  
Web04 | 2.8.180227.1 | Last Updated 8 Feb 2011    | [fluid](#)    Everything else Copyright © [CodeProject](#), 1999-2018