A ThreadPool implementation

Ratner Yuri, 22 May 2005

4.36 (10 votes) Rate this:

This article describes a ThreadPool implementation.

Download source - 8.67 Kb

Introduction

This article is about a thread pool. A pool manages requests from clients. A request is a pointer to a function or method, parameter for it, identity number, and priority number. Management is storing requests from clients and executing them in parallel by different threads. The order of execution is by priority, this scheduler is non-preemptive - when thread begins its execution nothing can stop it.

This code includes interesting subjects like OOD and polymorphism, multi-threading, synchronization, generic programming (templates) and STL containers.

I attached the source file of class pool and a simple `main` which shows how to use it.

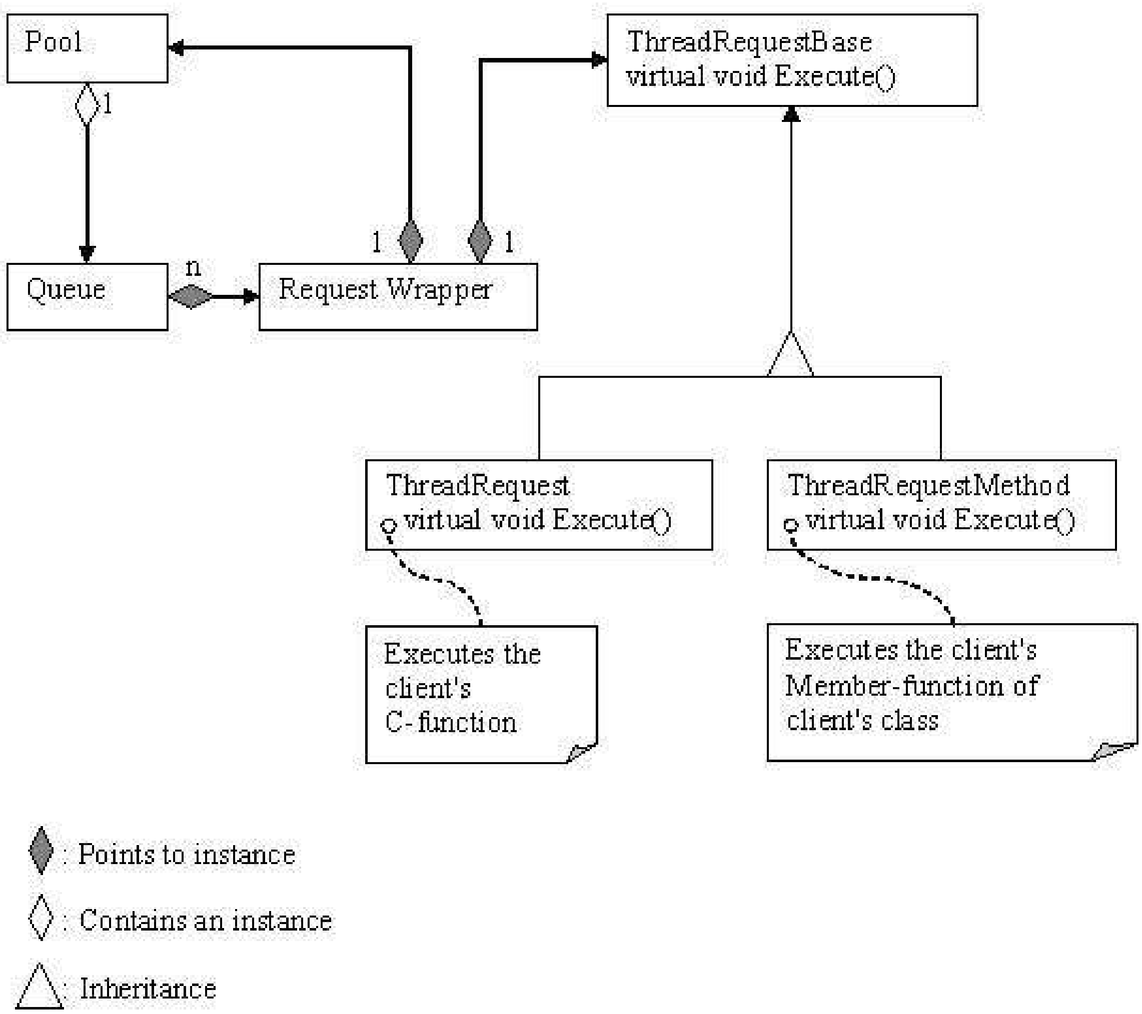
Interface

The interface of the pool class is very simple. It contains only a few functions.

- In `ctor` it receives number which limits the number of running threads in parallel. For example, if this number is 1 we get a sequential execution.
- `Run` function creates the main thread which does all the management. This function should not be called more then once; there is only one pool per object pool. If you want to create a lot of pools do it in the right way (`Pool p1 (2), p2 (2) ;`), the pool class is not singleton.
- `Stop` function kills the main thread. It saves requests that are being posted.
- `Enqueue` function adds a new request to a pool. The name enqueue is because requests are stored in the priority queue. This function is threading safe.
- `Wait` function waits until all requests are finished.
- `Wait` for request function waits until a specific request is finished.
- In `dtor` it stops the management, that is, there is no function `Stop`, `Pause` or something like that. If you want to stop a pool, kill it.

The order of calling functions is not important. You can `Run` and then `Enqueue` a new request.

Structure



Implementation

Request

Request is a function or method to be executed. Function means a C-Style function, also called "`__cdecl`". Method means a member function of some class - a non static function inside a class, also called "`__thiscall`". In the case of functions it is simple if you pass a pointer to a function and it works, but in the case of methods it does not work so easily. To execute a method you need an object of some class. This is a way to execute a method. See the *ThreadRequestMethod.h* file.

Hide Copy Code

```
template <typename ClassT, typename ParamT = LPVOID>
class ThreadRequestMethod : public ThreadRequestBase
{
      ...
      virtual void Execute() { (m_Client->*m_ClientMethod)(m_Param);
      ...
 private:
      ClassT *m_Client;
      void (ClassT::*m_ClientMethod)(ParamT *param);
      ParamT *m_Param;
};
```

Priority Queue

To implement the priority queue, I used STL. There is a ready priority queue. There is a way to "teach" the queue of STL to work with our priorities of the members in this queue.

The way is to define a functor: structure which is derived from "`binary_function`" object, and to override the `operator ()`. This functor is defined inside a private `struct` of a pool.

Hide Copy Code

```
// functor - used in stl container - priority queue.
template <class ClassT>
struct less_ptr : public binary_function<ClassT, ClassT, bool>
{
    bool operator()(ClassT x, ClassT y) const
    {
        return x->GetPriority() < y->GetPriority();
    }
};
```

This functor is good for every class which contains a function "`int GetPriority ()`". Now the definition of priority queue is as follows:

Hide Copy Code

```
priority_queue <
        RequestWrapper*,
        vector<RequestWrapper*>,
        less_ptr<RequestWrapper*>
    > RequestQueue;
```

The third template defines how to manage priorities.

The queue contains Request Wrappers. It wraps the request from clients and contains a pointer to the pool. It is necessary because a function which can be executed in a separate thread must be a C-function or a static function of some class. (I don't know another option.) So a Wrapper contains a pointer to "`this`".

Multi-threading and synchronization

Every request gets a thread to execute its function. There can be a lot of threads running in parallel and there are variables that every thread touches. To make safe those variables, I protected them by “Critical Sections” of Windows API. In the first version of this pool I used Mutexes but this way is not efficient as you can see from the following table from MSDN.

Table 1. Synchronization Objects Summary

Name	**Relative speed**	**Cross process**	**Resource counting**	**Supported platforms**
Critical Section	Fast	No	No (Exclusive Access)	95/NT/CE
Mutex	Slow	Yes	No (Exclusive Access)	95/NT/CE
Semaphore	Slow	Yes	Yes	95/NT
Event	Slow	Yes	Yes*	95/NT/CE
Metered Section	Fast	Yes	Yes	95/NT/CE

* Events can be used for resource counting, but they do not keep track of the count for you.

This is a way to ensure that only one thread will execute “// do something …” at the same time.

Hide Copy Code

```
CRITICAL_SECTION m_CSecQueue;
InitializeCriticalSection(&m_CSecQueue);

EnterCriticalSection(&m_CSecQueue);
// do something …
LeaveCriticalSection(&m_CSecQueue);
DeleteCriticalSection(&m_CSecQueue);
```

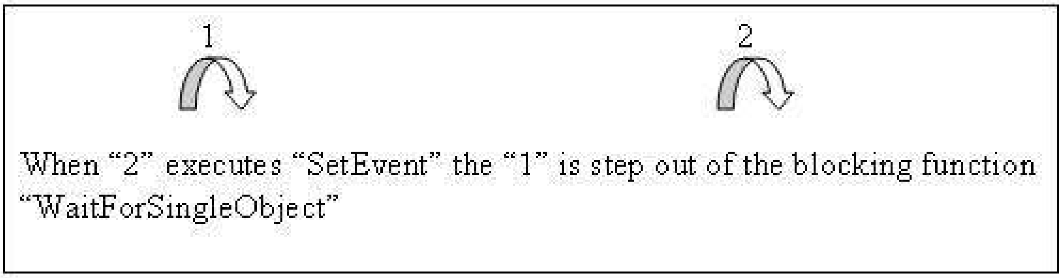
Another feature of Windows API that I used is events. The reason why I used it is to prevent wasting of CPU. This is a way to create an event:

Hide Copy Code

```
HANDLE event = CreateEvent (NULL, false, false, "");
```

This is a way to use an event:

1. `WaitForSingleObject (event, INFINITE);`
2. `SetEvent (event);`



See function "`RunMainThread`" to see how it saves CPU time.

How does it work

This is the life cycle of a request. First a client side:

Creation

Hide Copy Code

```
ThreadRequestBase *r = new ThreadRequest<Param>(&func, param, priority);
```

Submitting to a pool

Hide Copy Code

```
pool->Enqueue(r);
```

Start a Pool

Hide Copy Code

```
bool res = Pool->Run();
```

Management

This is the pool side (or inside of pool).

The request is added to the priority queue in function `Enqueue`. In the main thread function there is an infinite loop and inside there is a check if the pool can execute another request. It `Dequeue`s a request from the queue and runs some pool function, not the client request function, and increases the variable which counts the number of running threads. Inside a pool function, the client function is executed and after this it decreases the number of running threads, and signals by `SetEvent` to the main thread that a request was finished. Using of this event prevents the main thread from wasting CPU. Then the main thread cannot execute another request, it waits for this signal in a blocking function.

To be improved

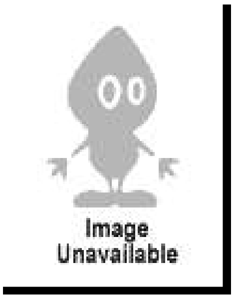
The pool creates a new thread for each request. When the request is finished the thread is dead. This is not an effective way but it is very simple. So the effective solution for pool management is to create the correct number of threads and use them to execute requests.

License

Share

TWITTER FACEBOOK

About the Author



Ratner Yuri

Web Developer

Israel

I am a student at the end of MCs computer science, work in Polycom as a C++ programmer.

You may also be interested in...

.NET's ThreadPool Class - Behind The Scenes

Optimizing Traffic for Emergency Vehicles using IOT and Mobile Edge Computing

Full Multi-thread Client/Server Socket Class with ThreadPool

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

My vote of 1
Member 4733739 3-May-11 4:36

Could you explain?
WREY 16-May-05 14:51

Re: Could you explain?
Ratner Yuri 16-May-05 22:03

Re: Could you explain?
WREY 19-May-05 12:02

Re: Could you explain?
Ratner Yuri 20-May-05 7:14

Alternative to wrapper class
Joe Pizzi 11-May-05 18:05

Why a priority queue?
yafan 28-Apr-05 6:10

Re: Why a priority queue?
Ratner Yuri 29-Apr-05 7:55

Re: Why a priority queue?
staceyw 1-May-05 6:45

Re: Why a priority queue?
yafan 23-May-05 11:01

Some issues
Tim Smith **28-Apr-05 4:58**

Re: Some issues
Ratner Yuri 29-Apr-05 8:32

Re: Some issues
Paolo Vernazza 3-May-05 5:50

+ InterlockedIncrement
BorisKoltsov 3-May-05 22:22

Re: Some issues
Paolo Vernazza 4-May-05 0:08

nice.
Angus He **28-Apr-05 0:52**

Re: nice.
Ratner Yuri 29-Apr-05 8:33

Refresh 1

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.