Subject: [PROPOSAL]: AsyncQueue class (a single threaded synchronization tool) for U++
Posted by Oblivion on Sun, 28 Feb 2016 18:47:31 GMT
View Forum Message <> Reply to Message

Hello guys,

I don't really want to flood this forum with proposals every other day. But AsyncQueue is the last class I'll share with you before I publish the SSH package.
In fact AsyncQueue is what made the SSH package for U++ possible.

I'm uploading the package here.

New version is even simple and it uses callbacks,allows lambdas.

It also contains both the API docs and an article/tutorial covering the rationale, aim, features, highlights and usage of the AsyncQueue package.

AsyncQueue is actually a generic version of non-blocking/async interface first used in HttpRequest. (Then I used the same model in NetworkProxy package, which I am now refactoring using AsyncQueue)

Excerpt from the article:

Rationale
U++ provides a rich set of core classes, including synchronisation tools and primitives designed mainly with multithreading in mind. Considering the general trends in, and demands of, modern computing, this is perfectly reasonable, if not imperative. When done properly, multithreading can and usually does give the best performance/cost ratio with a negligable overhead. Hovewer, multithreading is not always the optimal solution, and has its own disadvantages. For one, it is relatively difficult to write and debug a multithreaded code, as multithreading bring in its own set of problems such as classic concurrency problems which must be taken care of with extreme caution. Hence the increase of complexity. Moreover, multithreading models does not always scale well. Not every asynchronous operation or applicaton requires, or benefits from multithreading. Enter AsyncQueue.

Aim

AsyncQueue helper class does not aspire to provide an alternative to the existing multithreading classes in U++. Rather it is meant to be a small addition to the arsenal of synchronisation tools in the Core package, providing through a standardized interface a simple yet flexible asynchronous model targeting single-threaded component and application development, including but not limited to common socket operations, where multithreading either is not desirable or can easily get costly.

...

Please take your time to read the article.

Package has a BSD license, so feel free to use/modify it.

What is lacking? Example, of course Its example code will be SSH package, I guess...

Any suggestions, bug reports, criticism is always welcome.

Regards,

Oblivion.

File Attachments

1) AsyncQueue.zip, downloaded 443 times

Subject: Re: [PROPOSAL]: AsyncQueue class (a single threaded synchronization tool) for U++
Posted by Mindtraveller on Mon, 29 Feb 2016 15:36:05 GMT
View Forum Message <> Reply to Message

Hi, did you see Bazaar/MtAlt package? It implements exactly the asynchronous queue.
http://www.ultimatepp.org/forums/index.php?t=msg&th=4515 &goto=25097&#msg_25097

Subject: Re: [PROPOSAL]: AsyncQueue class (a single threaded synchronization tool) for U++
Posted by Oblivion on Mon, 29 Feb 2016 16:21:43 GMT
View Forum Message <> Reply to Message

Quote:Hi, did you see Bazaar/MtAlt package? It implements exactly the asynchronous queue.
http://www.ultimatepp.org/forums/index.php?t=msg&th=4515 &goto=25097&#msg_25097

Hello Mindtraveller!

Thanks for the information. Yes, I saw the MtAlt package, and I actually use it in some of my projects. It works well.
Though I never had the time to examine its implementation details.

Correct me if I'm wrong but afaik, MtAlt and AsyncQueue has different targets and scopes. Former is a multithreading tool, while the latter is meant to provide a standard interface for synchronizing non-blocking/async calls (including but not limited to socket operations) without multithreading.

Regards,

Oblivion

Subject: Re: [PROPOSAL]: AsyncQueue class (a single threaded synchronization tool) for U++
Posted by mirek on Mon, 29 Feb 2016 16:23:46 GMT
View Forum Message <> Reply to Message

Just a small nitpicking for now...

```
  for(int i  = 0; i < test_stack.GetCount(); i++) {
.......
    test_stack.Remove(i);
```

this combo is generally incorrect - it skips the element after the removed one. It probably works OK here (the next element gets tested/removed in the pass), but still...

```
if(WhenDo) WhenDo()
```

No need for if, it is OK to call unassigned Callback - it is NOP.

```
    template<class T>T& GetJobArg(int i)                    { return job_queue[i].Get<T>(i); }
```

I smell the bug here - is 'i' index of VectorMap or index of VarArgs?

Overall, I am pretty unsure what is that VectorMap useful for. You are never using it as map,except for RemoveJob. Not that all those VectorMap::Insert/Remove operations are quite slow.

In practice, I am not quite sure you really need a queue there. Do we really need more than single level?

Also, we are going full C++11 soon. Lambdas would make superior alternative for both DoJob and VarArgs IMO.

Mirek

Subject: Re: [PROPOSAL]: AsyncQueue class (a single threaded synchronization tool) for U++
Posted by Oblivion on Mon, 29 Feb 2016 17:25:05 GMT
View Forum Message <> Reply to Message

Quote:Just a small nitpicking for now...

```
  for(int i  = 0; i < test_stack.GetCount(); i++) {
.......
    test_stack.Remove(i);
```

this combo is generally incorrect - it skips the element after the removed one. It probably works OK here (the next element gets tested/removed in the pass), but still...

No problem. I'll change this one.

Quote:
I smell the bug here - is 'i' index of VectorMap or index of VarArgs?

Yes, that's a bug. Thanks!

It should be:

```
template<class T>T& GetJobArg(int i)                    { return job_queue[0].Get<T>(i); } // Implies that it is of current job's.
```

Quote:In practice, I am not quite sure you really need a queue there. Do we really need more than single level?

Well, if I understand you correctly, yes. I'll take my example from the SFtp class, which shows a common case I encounter.

SFtp class has several file manipulation methods (e.g. open, fstat, close).

1) If we want to retrieve info/stats of a certain file asynchronously, we can open() a file, pass its handle to fstat(), read the result, and then close() the file when our job is finished. We can do it all by ourselves. Using a loop in our source code. But doing this can , and most of the time actually do get tedious. Since it'll require extra coding, state tracking and error checking in most cases. (In SFtp class, or in the next version of my Ftps class, where non-blocking I/O is default, reading and writing data is even more complicated).

OR

2) We can simply program a async convenience method StartGetInfo() to do such operation (open() + fstat() + ... + close()) in just one step for us.

(There are use cases for both and SFtp allows both, by the way).

The bottom line is, my experience with asyncronous I/O showed me that programmable queue allows (not imposes) simplification with a very little cost.

Quote:Also, we are going full C++11 soon. Lambdas would make superior alternative for both DoJob and VarArgs IMO.

I have no objections to this. In fact lambdas are like god-send. Also, the first version of AsyncQueue actually utilized U++ callbacks (not lambda compatible ones). But I decided against it, since it resulted in other problems that DoJob() can easily solve.

But I have mainly two concerns: AFAIK, C++11 is not yet default in U++ and it would break backward compatibility.
Yet ,if you'd like me to implement it using lambdas, I'll definitely give it a go.

Regards,
Oblivion

Subject: Re: [PROPOSAL]: AsyncQueue class (a single threaded synchronization tool) for U++
Posted by mirek on Mon, 29 Feb 2016 17:34:58 GMT
View Forum Message <> Reply to Message

Oblivion wrote on Mon, 29 February 2016 18:25
But I have mainly two concerns: AFAIK, C++11 is not yet default in U++ and it would break backward compatibility.
Yet ,if you'd like me to implement it using lambdas, I'll definitely give it a go.

"C++11 required" will happen very soon now, probably before the end of March. We cannot be stuck in "compatibility mode" forever...

Subject: Re: [PROPOSAL]: AsyncQueue class (a single threaded synchronization tool) for U++
Posted by mr_ped on Mon, 29 Feb 2016 20:55:26 GMT
View Forum Message <> Reply to Message

Besides that the C++11 is quite backward compatible for ordinary source code...

Subject: Re: [PROPOSAL]: AsyncQueue class (a single threaded synchronization tool) for U++
Posted by Oblivion on Mon, 29 Feb 2016 21:09:47 GMT
View Forum Message <> Reply to Message

Quote:Besides that the C++11 is quite backward compatible for ordinary source code...

Sure, but it seems that I'm going to redesign it using std::tuple, variadic templates and lambdas. Now that U++ will require or at least officially switch to c++11 before the end of March, it won't be a big problem I guess...

Regards,

Oblivion

Subject: Re: [PROPOSAL]: AsyncQueue class (a single threaded synchronization tool) for U++
Posted by Oblivion on Tue, 01 Mar 2016 23:22:31 GMT
View Forum Message <> Reply to Message

Hello Mirek,

I've got rid of VarArgs, VectorMap, DoJob(), and re-based AsyncQueue class as a thin wrapper of vector containing callbacks.
Now it can also utilize lambda callbacks in-place (e.g. this way any StartFoo() can be both used for programming the queue, and as the place to write the actual non-blocking code.), and have backward compatibility.

While U++ callbacks allow up to 5 args, using std::tuple and std::get it can be increased by the user, if necessary.

If you approve this one, I'll rewrite the document accordingly and upload the package asap...

class AsyncQueue : Moveable<AsyncQueue> {

```
    Vector<Callback> job_queue;
    bool halted;

protected:
    Callback&          AddJob()                          { return job_queue.Add(); }
    AsyncQueue&        AddJob(Callback cb)                    { job_queue.Add(cb); return *this; }
    Callback&          InsertJob(int i)                  { return job_queue.Insert(i); }
    AsyncQueue&        InsertJob(int i, Callback cb)          { job_queue.Insert(i, cb); return *this; }
    Callback&          GetJob(int i)                     { return job_queue.At(i); }
    void               RemoveJob(int i)                  { job_queue.Remove(i); }
    void               ProcessQueue()                     { if(!job_queue.IsEmpty())
job_queue.Remove(0); }
    void               ClearQueue()                     { if(!job_queue.IsEmpty()) job_queue.Clear();
halted = false; }
    bool               QueueHasJobs() const                { return job_queue.GetCount() > 1; }
    bool               QueueIsHalted() const              { return halted; }
    bool               QueueIsEmpty() const                { return job_queue.IsEmpty(); }
    bool               GetJobCount() const               { return job_queue.GetCount(); }
    void               Halt()                          { job_queue.Clear(); halted = true; }

public:
    virtual bool       Do()                             { if(!job_queue.IsEmpty()) GetJob(0)(); WhenDo();
return InProgress(); }

    bool               InProgress() const                { return !job_queue.IsEmpty();  }
    bool               IsSuccess() const                 { return job_queue.IsEmpty() && !halted; }
    bool               IsFailure() const                { return halted;  }

    Callback           WhenDo;

    AsyncQueue() : halted(false)    {}
    virtual ~AsyncQueue()          {}

    AsyncQueue(AsyncQueue rval_ a)                     { job_queue = pick(a.job_queue); halted =
a.halted; }
    void operator=(AsyncQueue rval_ a)                 { ClearQueue();  job_queue =
pick(a.job_queue); halted = a.halted; }
};
```

What do you think?

Regards,
Oblivion

Subject: Re: [PROPOSAL]: AsyncQueue class (a single threaded synchronization tool) for U++
Posted by Oblivion on Wed, 02 Mar 2016 19:15:36 GMT
View Forum Message <> Reply to Message

Hello guys,

I uploaded the package (see the first post), reflecting the changes.

New version is based on callbacks (allowing lambdas under c++11), and is simpler.

Regards,
Oblivion