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