Subject: Re: RE: Job package: A scope-bound worker thread for non-blocking operations.
Posted by Oblivion on Tue, 19 Sep 2017 09:12:03 GMT
View Forum Message <> Reply to Message

Hello Mirek,
Thanks for all the corrections. I find your opinions really valuable and useful. I was suspicios there was something wrong with that benchmark too.

Quote:This is not quite true (unlike Job, which in fact requires you to move the data if you need it outside the Job).

Not necessarily. I can always use a pointer or Ref(), or std::ref. (In it's current state it wouldn't be as clean as capturing, I admit, but can be improved)
Or I can simply pass reference, using lambda capture. That is not forbidden. But then I have to take into account the object's life time too.
Job solves some of those head aches. (Of course, tt may introduce its own).

For the sake of discussion (ugly):
In Job.h

```
template<class V = T>
Job(V v) : Job()       { *data = v; }
```

then, in Example.cpp:

```
int i;

Job<int*> job(&i);
job.Start([]{ auto& n = *Job<int*>::Data() = 100;});
job.Finish();
Cout() << "Number is " << i << '\n';
```

Quote:future / promise also seems to have superior error handling facilities (exceptions can easily be passed from thread to caller).

Not really hard to add. I can: process JobErrors, and re-throw plain Upp::Exc, or std::exception category.

Quote:
The one thing that seems to be missing is future -> promise abort.

Job has a per-job cancel mechanism.

So what do you suggest?

Your Async wrapper is really simple and nice. Are you going to add it to U++?

Or should I improve Job? (add exception forwarding, easier way to pass references and pointers...)

My favourite :) : Or should I re-introduce Future/promise pair as was before? After all Job was inially a future/promise wrapper. But it lacked RAII and worker thead model.
Now they are in place. All I need to do is to incorporate it to the current model using the Async example you provided (except cowork).
(Without changing the public api of Job much. IMO It looks familiar and acts fine.)

Or should I simply abandon it? This tool is born out of need, which plain future/promise mechanism couldn't satisfy (i.e. a fine-grained control over threads for non-blocking behaviour.)

I don't see a reson to hang onto it if there will be a better solution.

Again, thank you very much for taking time to discuss the issue.

Job currently lacks shared states but it is not what Job is meant for anyway. IMO CoWork is there for it (and for bunch of other things).

Best regards,
Oblivion