Subject: Re: RE: Job package: A scope-bound worker thread for non-blocking operations.
Posted by mirek on Tue, 19 Sep 2017 08:15:33 GMT
View Forum Message <> Reply to Message

Oblivion wrote on Sun, 17 September 2017 23:20
[code]
1) future/promise pair requires at least moving of the resulted data, which can be relatively expensive depending on the object type. On the other hand, Job acts as a simple container and uses a reference based result gathering method. This makes it possible to reduce move/copy overhead involved (nearly down to zero).

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

```
#include <Core/Core.h>
#include <future>

using namespace Upp;

template <class D, class P, class F, class... Args>
void set_future_value(D *dummy, P&& p, F&& f, Args&&... args)
{
 p.set_value(f(args...));
}

template <class P, class F, class... Args>
void set_future_value(void *dummy, P&& p, F&& f, Args&&... args)
{
 f(args...);
 p.set_value();
}

template< class Function, class... Args>
std::future<std::result_of_t<std::decay_t<Function>(std::decay_t<Args>...)>>
Async(Function&& f, Args&&... args )
{
 std::promise<std::result_of_t<std::decay_t<Function>(std::decay_t<Args>...)>> p;
 auto ftr = p.get_future();
 CoWork::Schedule([=, p = pick(p)]() mutable {
  std::result_of_t<std::decay_t<Function>(std::decay_t<Args>...)> *dummy = NULL;
  set_future_value(dummy, pick(p), pick(f));
 });
 return ftr;
}
```

```
CONSOLE_APP_MAIN
{
// with reference
 String h;
 auto f = Async([&h] { h = "Hello world"; });
 f.get();
 DDUMP(h);

// using return value
 DDUMP(Async([] { return "Hello world"; }).get());
}
```

BTW, I am mostly arguing there because it is a good way for me to investigate these issues....
