Subject: Re: Job package: A lightweight multithreading tool, using promise/future mechanism
Posted by mirek on Sun, 10 Sep 2017 13:56:16 GMT
View Forum Message <> Reply to Message

BTW, as exercise:

```
#include <CtrlLib/CtrlLib.h>

#include <future>

using namespace Upp;

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 {
  p.set_value(f(args...));
 });
 return ftr;
}

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

Still not sure about real world scenarion where I would prefer using future/promise over CoWork.

Maybe my problem with future/promise really is that fact that usually the "result" of async operation as a change in some data that gets into it as reference. future forces me to do a copy to store the function result.