

---

Subject: Re: AsyncWork and pick/std::move problems...

Posted by [mirek](#) on Sat, 21 Sep 2024 12:50:09 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

For reference, this is my latest version

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

using namespace Upp;

#define AsyncWork AsyncWork2

template <class Ret>
class AsyncWork {
    template <class Ret2>
    struct Imp {
        CoWork co;
        Ret2 ret;

        template<class Function, class... Args>
        void Do(Function&& f, Args&&... args) {
            co.Do([=, a = std::make_tuple(std::forward<Args>(args) ...)]() mutable {
                std::apply([=](auto&& ... as) { ret = f(std::forward<Args>(as)...); }, std::move(a));
            });
        }

        const Ret2& Get() { return ret; }
        Ret2 Pick() { return pick(ret); }
    };
};

struct ImpVoid {
    CoWork co;

    template<class Function, class... Args>
    void Do(Function&& f, Args&&... args) {
        co.Do([=, a = std::make_tuple(std::forward<Args>(args) ...)]() mutable {
            std::apply([f](auto&& ... as) { f(std::forward<Args>(as)...); }, std::move(a));
        });
    }

    void Get() {}
    void Pick() {}
};

using ImpType = typename std::conditional<std::is_void<Ret>::value, ImpVoid, Imp<Ret>>::type;

One<ImpType> imp;

public:
    template<class Function, class... Args>
```

```

void Do(Function&& f, Args&&... args)      { imp.Create().Do(f, std::forward<Args>(args)...); }

void      Cancel()                      { if(imp) imp->co.Cancel(); }
static bool IsCanceled()                { return CoWork::IsCanceled(); }
bool     IsFinished()                  { return imp && imp->co.IsFinished(); }
Ret     Get()                         { ASSERT(imp); imp->co.Finish(); return imp->Get(); }
Ret     operator~()                   { return Get(); }
Ret     Pick()                        { ASSERT(imp); imp->co.Finish(); return imp->Pick(); }

AsyncWork& operator=(AsyncWork&&) = default;
AsyncWork(AsyncWork&&) = default;

AsyncWork()                           {}
~AsyncWork()                          { if(imp) imp->co.Cancel(); }
};

template< typename Function, class... Args>
AsyncWork<std::invoke_result_t<Function, Args&&...>> Async2(Function&& f, Args&&... args)
{
    AsyncWork2<std::invoke_result_t<Function, Args&&...>> h;
    h.Do(f, std::forward<Args>(args)...);
    return pick(h);
}

CONSOLE_APP_MAIN
{
    Vector<double> data;
    data << 1 << 2 << 3;
    int test = 0;

    auto job = Async2([](Vector<double>&& data) -> double {
        return Sum(data);
    }, pick(data));

    auto job2 = Async2([&] { test = 3141592; });
    auto job3 = Async2([](String s) { return Atof(s) * 3; }, String("1.1"));
    auto job4 = Async2([](String s) { return Atof(s) * 3; }, "1.1");
    auto job5 = Async2([](double x) { return x; }, 1.234);
    auto job6 = Async2([](double x) { return x; }, 1);
    auto job7 = Async2([](const char *s) { return Atof(s) * 3; }, "1.1");

    DDUMP(job.Get());
    job2.Get();
    DDUMP(test);
}

```

job4 and job7 do not work - basically, it fails with string literals.

Mirek

---