## Introduction to std::shared\_future

This lesson gives an introduction to std::shared\_future which is used in C++ for multithreading.

```
we'll cover the following ^
• std::shared_future
```

## std::shared\_future #

A future becomes shared by using <code>fut.share()</code>. A shared future is associated with its promise and can independently ask for the result. An <code>std::shared</code> future has the same interface as an <code>std::future</code>.

In addition to the std::future, a std::shared\_future enables us to query the
promise independently of the other associated futures.

There are two ways to create a std::shared\_future:

- 1. Invoke fut.share() on an std::future fut. Afterwards, the result is no longer available. That means valid == false.
- 2. Initialize an std::shared\_future from an std::promise:
   std::shared\_future<int> divResult= divPromise.get\_future().

The handling of an std::shared\_future is special.

```
// sharedFuture.cpp

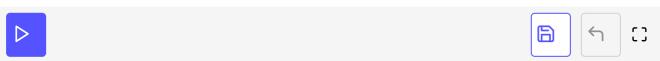
#include <future>
#include <iostream>
#include <thread>
#include <utility>

std::mutex coutMutex;

struct Div{

  void operator()(std::promise<int>&& intPromise, int a, int b){
    intPromise.set_value(a/b);
}
```

```
};
struct Requestor{
  void operator ()(std::shared_future<int> shaFut){
    // lock std::cout
    std::lock_guard<std::mutex> coutGuard(coutMutex);
    // get the thread id
    std::cout << "threadId(" << std::this_thread::get_id() << "): ";</pre>
    std::cout << "20/10= " << shaFut.get() << std::endl;</pre>
  }
};
int main(){
  std::cout << std::endl;</pre>
  // define the promises
  std::promise<int> divPromise;
  // get the futures
  std::shared_future<int> divResult = divPromise.get_future();
  // calculate the result in a separat thread
  Div div;
  std::thread divThread(div, std::move(divPromise), 20, 10);
  Requestor req;
  std::thread sharedThread1(req, divResult);
  std::thread sharedThread2(req, divResult);
  std::thread sharedThread3(req, divResult);
  std::thread sharedThread4(req, divResult);
  std::thread sharedThread5(req, divResult);
  divThread.join();
  sharedThread1.join();
  sharedThread2.join();
  sharedThread3.join();
  sharedThread4.join();
  sharedThread5.join();
  std::cout << std::endl;</pre>
```



Both work packages, that of the promise and that of the future, are function objects in this current example. In line 46, divPromise will be moved and

executed in the thread divThread. Accordingly, std::shared\_future s are copied in all five threads (lines 57 - 61). It's important to emphasize it once more: In contrast to an std::future object that can only be moved, we can copy an std::shared\_future object.

The main thread waits in lines 57 to 61 for its child threads to finish their jobs and to display their results.

Dividing a number by 0 shows undefined behavior.

This concludes our discussion of tasks.