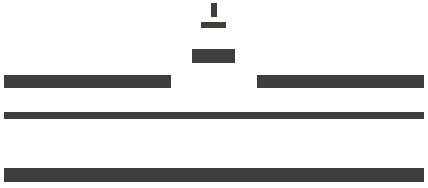
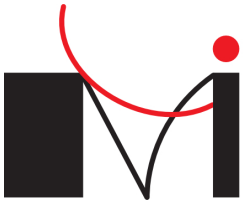


Westfälische Wilhelms-Universität Münster



Applied Mathematics Münster

std::async() in C++11

Basic Multithreading

living.knowledge
WWU Münster

Andreas Buhr

2. December 2015

std::thread

```
#include <iostream>
#include <thread>
void hello(){
  std::cout << "Hello from thread" << std::endl;
}
int main(){
  std::thread t1(hello);
  t1.join();
}
```

Compile: g++ -std=c++11 -pthread foo.cpp

std::async()

```
#include <iostream>
#include <future>
void hello(){
  std::cout << "Hello from thread" << std::endl;
}
int main(){
  std::future<void> f1 = std::async(hello);
  f1.wait();
}
```

Compile: g++ -std=c++11 -pthread foo.cpp

std::thread with Arguments

```
#include <iostream>
#include <thread>
void hello(std::string message){
  std::cout << message << " from thread" << std::endl;
}
int main(){
  std::thread t1(hello, "Guten Tag");
  t1.join();
}
```

Compile: g++ -std=c++11 -pthread foo.cpp

std::async() with Arguments

```
#include <iostream>
#include <future>
void hello(std::string message){
  std::cout << message << " from thread" << std::endl;
}
int main(){
  std::future<void> f1 = std::async(hello, "Guten Tag");
  f1.wait();
}
```

Compile: g++ -std=c++11 -pthread foo.cpp

std::future<T>

std::future<T>

Interface:

```
void wait();
T get();
```

std::async() with Result

```
#include <iostream>
#include <future>
double calculate(){
  double result = 42.; // very expensive calculation
  return result;
}
int main(){
  std::future<double> f1 = std::async(calculate);
  // do something else in the meantime
  std::cout << "The result is " << f1.get() << std::endl;
}
```

std::async() with Exception

```
#include <iostream>
#include <future>
double calculate(){
  throw std::runtime_error("end of the world");
}
int main(){
  std::future<double> f1 = std::async(calculate);
  try{
    std::cout << "Result: " << f1.get() << std::endl;
  } catch (std::exception& e){
    std::cout << "Error: " << e.what() << std::endl;
  }
}
```

Launch Policies

```
#include <iostream>
#include <future>
void hello(){
  std::cout << "Hello from thread" << std::endl;
}
int main(){
  std::future<void> f1 =
    std::async(std::launch::deferred, hello);
  f1.wait();
}
```

Policies

- std::launch::async
  - New thread launched
- std::launch::deferred
  - No thread
  - execution in “wait()” or “get()”
- Default: let implementation decide.

Pitfalls

- g++ default policy is “deferred”, so no parallel execution by default.
- Implicit “wait()” in destructor of future with “async” policy.

```
std::async(std::launch::async,myfunction);
```

This is a synchronous call.

Not parallel

```
#include <iostream>
#include <future>
#include <thread>
void wait(){
  std::cout << "start waiting" << std::endl;
  std::this_thread::sleep_for(std::chrono::seconds(2));
  std::cout << "stop waiting" << std::endl;
}
int main(){
  std::async(std::launch::async,wait);
  std::async(std::launch::async,wait);
}
```

Parallel

```
#include <iostream>
#include <future>
#include <thread>
void wait(){
  std::cout << "start waiting" << std::endl;
  std::this_thread::sleep_for(std::chrono::seconds(2));
  std::cout << "stop waiting" << std::endl;
}
int main(){
  auto f1 = std::async(std::launch::async,wait);
  auto f2 = std::async(std::launch::async,wait);
}
```

Summary

- std::async interface similar to std::thread.
- Result and/or exceptions are propagated back through std::future.
- Two start policies, “async” and “deferred”. Only “async” starts a thread.
- Make sure to save the future ;-).