Multi-Threading

Multi-Threading in C++

Stefano Quer and Alessandro Savino
Dipartimento di Automatica e Informatica
Politecnico di Torino

Introduction

- The header thread defines the class std::thread that can be used to start new threads
  - Using this class is the best way to use platform-independent threads

```
#include <thread>
```

  - Using it may require additional compiler flags
    - For gcc and clang, use -pthread

```
set(CMAKE_CXX_FLAGS
  "${CMAKE_CXX_FLAGS} -std=c++14 -pthread")
```

Introduction

- The library is based on objects of type std::thread
  - The operator std::thread works with any callable object like a function, an instance of a class, a lambda expression

Main thread primitives

- The library covers all main functionalities
  - But … there is no way to automatically capture the data computed by a thread

Type	Main characteristics
std::thread t;	Creates a thread object t.
std::thread t(f);	Creates a thread object t associated with the thread function f.
std::thread t(f,p1,p2,…); std::thread t{f(p1,p2,…)};	Creates a thread object t associated with the thread function f which receives the parameters p1, p2, etc.
t2=std::move(t1);	Move the thread associated to the thread object t1 to object t2.
t.detach()	Makes the thread t as detached.
t.join()	Waits the thread t for joining.

Other primitives

- The thread library also contains useful functions related to starting and stopping threads

Type	Main characteristics
std::this_thread::sleep_for	Stop the current thread for a given amount of time.
std::this_thread::sleep_until	Stop the current thread until a given point in time.
std::this_thread::yield	Let the operating system schedule another thread.
std::this_thread::get_id	Get the (operating-system-specific) id of the current thread.
std::thread::hardware_concurrency	Reports the actual max number of threads based on your architecture.

Examples

```
void f1() {
   ...
}

void f2(int a, int b) {
   ...
}
```

Function definitions (without and with parameters)

```
std::thread t1;

std::thread t2(f1);

std::thread t3(f2, 123, 456);

std::thread t4([] { f2(123, 456); });
```

Creates an object that does not refer to a thread

Starts an object thread that calls f1()

Starts a thread that calls f2(123, 456)

Works also with lambda functions

Join and Move

- The member function **join** can be used to wait for a thread to finish
  - Function **join** must be called exactly once for each thread
- Standard threads are not copyable, but movable, so they can be used in containers
  - Moving an **std::thread** transfers all resources associated with the running thread
  - Only the new thread can be joined

Examples

```
void f1() {
   ...
}

void f2(int a, int b) {
   ...
}
```

Function definitions
(without and with parameters)

```
std::thread t1;
std::thread t2(f2, 123, 456);
// Alternative syntax
std::thread t3{f2(123, 456)};

t1 = std::move(t2);
t1.detach();
t3.join();
```

Starts a thread that
calls f2(123, 456)

std::thread is moveable not copyable
Associate f2 to thread object t1

Examples

```
void f(int &result) {
   ...
   result = ...;
}
```

Function definition
(with a paramter by reference)

```
int main() {
   ...
   std::thread t (f, std::ref(i));
   ...
   t.join();
}
```

Parameter by reference

Variable i will assume the value once the execution is terminated

Examples

```
std::thread t1([] { std::cout << "Hi\n"; });

std::thread t2 = std::move(t1);

t2.join();
```

t1 is now empty

The thread originally started in t1 is joined

```
std::thread t1(some_function);
std::thread t2 = std::move(t1);
std::thread t1(some_other_function);
std::thread t3;
t3 = std::move(t2);
t1 = std::move(t3);
```

Error: t1 is running some_other_function
The assignment terminates the program

Examples

```
void my_thread () {
  std::cout << "TID" <<
    std::this_thread_::get_id() << endl;
}

main() {
  std::thread t1{mythread};
  std::thread t2{mythread};

  t1.join();
  t2.join();

  return 1;
}
```

The output operation shoud be protected

Examples

```
void safe_print (int i) {
  ... Enter critical section ...
  std::cout << i;
  ... Exit critical section ...
  return;
}
```

For example, use a mutex (see, unit 06)

```
std::vector<std::thread> threadPool;

for (int i = 1; i <= 9; ++i) {
  threadPool.emplace_back([i] { safe_print(i); });
}

for (auto& t : threadPool) {
  t.join();
}
```

Digits 1 to 9 are printed (unordered)