Methods of Threads in Practice

This lesson shows the application of commonly used thread methods such as get_id, hardware_concurrency, and joinable in C++.

Some of the most commonly used thread methods are in the following code widget:

```cpp
// threadMethods.cpp

#include <iostream>
#include <thread>

using namespace std;

int main(){

  cout << boolalpha << endl;

  cout << "hardware_concurrency()= "<< thread::hardware_concurrency() << endl;

  thread t1([]{cout << "t1 with id= " << this_thread::get_id() << endl;});
  thread t2([]{cout << "t2 with id= " << this_thread::get_id() << endl;});

  cout << endl;

  cout << "FROM MAIN: id of t1 " << t1.get_id() << endl;
  cout << "FROM MAIN: id of t2 " << t2.get_id() << endl;

  cout << endl;
  swap(t1,t2);

  cout << "FROM MAIN: id of t1 " << t1.get_id() << endl;
  cout << "FROM MAIN: id of t2 " << t2.get_id() << endl;

  cout << endl;

  cout << "FROM MAIN: id of main= " << this_thread::get_id() << endl;

  cout << endl;

  cout << "t1.joinable(): " << t1.joinable() << endl;

  cout << endl;

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

  cout << endl;
```

```
    cout << "t1.joinable(): " << t1.joinable() << endl;


    cout << endl;

}
```

In combination with the output, the program should be quite easy to follow.

Maybe it looks a little weird to have the threads `t1` and `t2` (lines 14 and 15) run at different points in time during the program execution. However, we have no guarantee as to when each thread will run; we only have the guarantee that both threads will run before `t1.join()` and `t2.join()` in lines 38 and 39.

The more mutable (non-const) variables the threads share, the more challenging multithreading becomes.

This chapter enlightened us with the basics of threads. In the next chapter, we'll discuss data sharing between threads.