

C++ Threads

STL (Standard Template Library)

Mini-introduction

Les bases

C++ thread

Introduction: basics

C++ thread: basics

Création d'un thread effectuant un traitement.
=> très proche de Python.

```
// THREAD en C++  
//  
// Compile : g++ -std=c++0x xxx.cpp -o xxx_program -lpthread  
//           OU -std=c++11 OU -std=c++14 OU -std=c++17 OU -std=c++20
```

```
#include <thread>
```

```
void hello() {  
    std::cout << "Hello from thread " << std::this_thread::get_id() << std::endl;  
}
```

NOTE : `std::this_thread::get_id()` n'est pas un index comme en Python mais un identifiant unique du thread (l'utilisateur n'en a pas le contrôle). On verra comment passer un paramètre au thread, comme un index.

Declare the
"thread function"

```
void main()
```

```
{
```

```
    const unsigned int nbMaxThreads = std::thread::hardware_concurrency();  
    std::cout << nbMaxThreads << " concurrent threads are supported.\n";
```

```
    // Declare and start the thread  
    std::thread t1( hello );
```

```
    // DO some other stuff  
    // ...
```

```
    // Wait for end of thread t1  
    t1.join();
```

```
    // ...
```

```
}
```

Thread creation:
Pass the "thread
function" to the
thread's constructor

C++ thread: basics

Création de plusieurs threads s'exécutant en parallèle.
Stockage des threads dans un conteneur, ici un tableau (STL vector).
=> très proche de Python.

```
#include <thread>
```

```
void hello() {  
    std::cout << "Hello from thread " << std::this_thread::get_id() << std::endl;  
}
```

Declare the “thread function” called by each thread

```
void main()  
{
```

```
// Declare and start several threads (stored in a container [STL vector])  
std::vector< std::thread > threads;  
for ( int i = 0; i < 5; ++i )  
{  
    threads.push_back( std::thread( hello ) );  
}
```

Threads creation:
Pass the “thread function” to the thread's constructor

```
// DO some other stuff  
// ...
```

Rappel : STL vector
⇒ ajout d'un élément à la fin avec push_back().

```
// Wait for end of all threads  
for ( auto& thread : threads )  
{  
    thread.join();  
}
```

Rappel : STL vector
⇒ parcours d'un conteneur de la STL (plusieurs syntaxes possibles de la boucle FOR).
⇒ “auto” permet d'éviter de taper le type.

```
// ...  
}
```

C++ thread: basics

Passage de paramètres à des threads.
=> très proche de Python.

```
#include <thread>

void hello( int i ) {
    std::cout << "Hello from thread: " << i << std::endl;
}

void main()
{
    // Declare and start several threads (stored in a container [STL vector])
    std::vector< std::thread > threads;
    for ( int i = 0; i < 5; ++i )
    {
        // Thread creation, passing an argument to its function
        threads.push_back( std::thread( hello, i ) );
    }

    // DO some other stuff
    // ...

    // Wait for end of all threads
    for ( auto& thread : threads )
    {
        thread.join();
    }

    // ...
}
```

Declare the “thread function” called by each thread.
Function can have several arguments.

Threads creation:
Pass the “thread function” to the thread's constructor,
with parameters separated by commas.

Possibilité de passer des paramètres aux threads, comme son indice (très utile), en les séparant par des virgules.

NOTE : si grosse donnée comme un tableau, utiliser la fonction **std::ref()** pour passer le paramètre afin d'éviter un passage par copie coûteux.

C++ thread: basics

Création de threads avec une "lambda function".

```
#include <thread>

void main()
{
    // Declare and start several threads (stored in a container [STL vector])
    std::vector< std::thread > threads;
    for ( int i = 0; i < 5; ++i )
    {
        // Thread creation, passing a "lambda function" (modern C++)
        threads.push_back( std::thread( []() {
            std::cout << "Hello from thread " << std::this_thread::get_id() << std::endl; } )
        );
    }

    // DO some other stuff
    // ...

    // Wait for end of all threads
    for ( auto& thread : threads )
    {
        thread.join();
    }

    // ...
}
```

Threads creation:
Pass the "thread function" to
the thread's constructor,
as a lambda function.

C++ thread concurrent programming

```
struct Counter {  
    int value;  
  
    Counter() : value(0){}  
  
    void increment(){  
        ++value;  
    }  
};
```

```
// Declare a counter  
Counter counter;
```

```
std::vector< std::thread > threads;  
for ( int i = 0; i < 5; ++i )  
{  
    threads.push_back( std::thread( [&counter]() {  
        for ( int i = 0; i < 100; ++i ) {  
            counter.increment();  
        }  
    })  
);  
}
```

```
for ( auto& thread : threads ){  
    thread.join();  
}
```

```
std::cout << counter.value << std::endl;
```

5 threads incrémentent, chacun, 100 fois un compteur.
Chaque exécution du code produit, au lieu de 500,
une valeur différente.

Ex : 442, 500, 477, 400, 422, 487...

PROBLEME :

Race condition => section critique de code.

Le paramètre externe "counter" est passé par référence (avec le &) à la lambda function (il pourra être modifié).


```
struct ConcurrentCounter {
```

```
    std::mutex mutex;  
    int value;
```

```
    ConcurrentCounter() : value(0) {}
```

```
    void increment() {  
        mutex.lock();  
        ++value;  
        mutex.unlock();  
    }
```

```
};
```

```
ConcurrentCounter concurrentCounter;  
std::vector< std::thread > threads;
```

```
for ( int i = 0; i < 5; ++i )
```

```
{  
    threads.push_back( std::thread( [&concurrentCounter ]() {  
        for ( int i = 0; i < 100; ++i ) {  
            concurrentCounter.increment();  
        }  
    })  
);  
}
```

```
for ( auto& thread : threads ) {  
    thread.join();  
}
```

```
std::cout << concurrentCounter.value << std::endl;
```

Utilisation d'un verrou, ici un mutex (**std::mutex**), pour éviter la race condition. Mutex : couple de fonctions **lock()**, **unlock()**. => proche de Python (acquire(), release())

NOTE:
Code non optimisé.
Il sert d'illustration.

C++ Threads

On peut faire bien plus avec les threads de la STL en C++
(ex : synchronisations [condition], async, future / promise, etc...)
On a juste vu les notions de base.