**task 클래스**

**task는 병렬 프로그래밍에서 하나의 작업 단위를 표현하는 템플릿 클래스 입니다.** PPL에서 가장 기본이 되는 개념으로 사용자는 직접적 또는 간접적 방법에 의해 task를 사용하게 됩니다. 간접적 방법이라 언급한 것은 이후에 나오게 될 병렬 알고리즘 같은 경우 겉으로 드러나진 않지만 내부적으로는 task\_handle을 이용하여 구현이 되어 있기 때문에 사용자가 직접 task 클래스를 생성하여 사용하지 않아도 간접적으로 task를 이용하게 된다는 의미입니다. 결국 task를 빼놓고는 PPL을 설명할 수 없다는 얘기겠죠? ^^ 그럼 task를 어떻게 생성하고 사용할 수 있는지 간단한 예제를 통해 알아보겠습니다.

**※ task 클래스가 포함된 ppltasks.h 파일은 Visual Studio 2010 버전에는 존재하지 않습니다. 그러므로 ppltask.h 파일을 사용하는 예제들은 Visual Studio 2012 이상 버전에서만 컴파일됩니다.**

|  |
| --- |
| // task 클래스를 사용하려면 ppltasks.h 파일을 포함시켜야 합니다. **#include <ppltasks.h>** #include <iostream>  // PPL 관련 클래스와 함수들은 모두 concurrency 네임스페이스에 포함되어 있습니다. **using namespace concurrency;** using namespace std;  int wmain() {     // task 객체 생성     **task<int> t([]()** {         return 42;     });      // task 종료 대기, get 메서드도 task의 종료를 대기하기 때문에 생략 가능     **t.wait();**      // 결과 출력     wcout << **t.get()** << endl; }  //실행결과  42 |

위의 예제에서는 **정수값 42를 리턴하는 매우 단순한 task 객체인 t를 생성하고 있습니다**. 여기서 주목해야할 두 가지는 템플릿 파라미터인  int 타입과 생성자의 파라미터인 람다 함수 입니다. **먼저 템플릿 파라미터에는 task의 리턴 타입을 지정합니다.** 모든 task는 리턴 타입을 가지며 이는 함수의 리턴 타입의 개념과 유사하다고 할 수 있습니다. 만약 리턴 타입을 가지지 않는 task는 task<void>로 지정할 수 있습니다. **다음으로 task 클래스 생성자의 파라미터에는 '함수형'을 전달할 수 있는데 이는 곧 task가 실행되면 처리하게될 작업이 됩니다.** 앞서 함수형이라고 설명한 것은 위 예제에서 사용한 람다 함수와 함수 객체, 함수 포인터를 모두 포함한 것이며 앞으로 이를 묶어 함수 형으로 부르도록 하겠습니다. 그리고 이 강좌에서는 코드 가독성을 위해 대부분의 예제에서 람다 함수를 사용하도록 하겠습니다.

**task를 생성하는 또 한 가지의 방법은 create\_task 함수를 사용하는 방법입니다.** 위에서는 task의 타입을 task<int>와 같이 명시적으로 지정해 주었지만 auto 키워드와 create\_task 함수를 조합하면 좀 더 깔끔한 코드를 만들어 낼 수 있습니다. 아래 코드는 앞의 task 객체 생성 코드와 정확히 같은 동작을 하게됩니다.

|  |
| --- |
| **auto t = create\_task([]() -> int** {     return 42; }); |

다음 코드를 보면 task 클래스의 wait 메서드를 이용하여 task가 처리 될 때까지 기다리고 있습니다. 마지막으로는 get 메서드를 호출하여 task의 리턴값을 받아 출력하고 있구요. 주석을 통하여 설명하였지만 get 메서드는 task 처리가 종료될때까지 기다렸다가 리턴되기 때문에 위의 경우에는 wait 메서드를 생략하더라도 결과는 같습니다.

자, 여기서 잠깐 문제를 하나 내보겠습니다. 위 예제에서 task가 실행되는 시점은 언제일까요? 참고로 task 클래스에는 task를 실행 하기 위한 메서드가 존재하지 않습니다. 그렇다면 wait 또는 get 메서드를 호출 함으로서 task가 실행이 되는 것일까요?  **정답은 'task 생성과 동시에 실행될 수 있는 가장 빠른 시점' 입니다.** 이처럼 어려운 표현을 쓴 이유는 task는 병렬로 처리되기 때문이죠. **쉽게 얘기하면 task 생성 직후에 실행되어 처리된다고 할 수 있겠습니다.**

사실 위의 예제 같은 경우 하나의 task를 생성하고 task의 처리 종료를 대기하기 때문에 병렬 처리의 의미는 없습니다. 만약 wait와 get으로 task의 종료를 기다리는 대신 다른 어떤 처리를 수행한다면 병렬 처리의 의미가 있게 되겠죠. 여기서는 task의 생성 방법과 간단한 사용법을 익히는 정도로 넘어가도록 하겠습니다.

**task 연결**

프로그램을 비동기로 구현 하다 보면 어떤 작업이 먼저 수행되고 그 결과값을 이용하여 다음 작업을 수행 해야 할 경우가 많이 있습니다.**이러한 경우에 먼저 실행될 task와 해당 task 종료 이후 실행될 task를 연결할 수 있습니다.** (MSDN 문서에서는 **Continuation Tasks** 라는 단어로 표현하고 있습니다만, 저는 **'task 연결'** 이라는 단어로 표현하겠습니다. 또한 이해를 돕기 위해 먼저 실행될 task를 '부모 task'로, 부모 task 종료 후 실행될 task를 '자식 task'로 표현하도록 하겠습니다. 이는 MSDN 문서에서는 antecedent와 continuation으로 표현하고 있습니다.) 그럼 예제를 통해 task를 어떻게 연결하는지 알아보도록 하죠.

|  |
| --- |
| #include <ppltasks.h> #include <iostream>  using namespace concurrency; using namespace std;  int wmain() {     auto t = create\_task([]() -> int     {         return 42;     });         **t.then([](int result)**     {         wcout << result << endl;     }).wait(); }    //실행결과  42 |

위 예제에서는 두 개의 task 객체를 생성합니다. 첫 번째 task는 정수 값 42를 리턴 하는 task이고 두 번째 task는 정수 값을 받아 출력하는 task 입니다. 그리고 task 클래스의 then 메서드를 통해서 매우 직관적인 방법으로 두 개의 task를 연결합니다. **then 메서드의 형식을 살펴보면 메서드의 파라미터로 함수형을 요구하는데 이것이 바로 자식 task가 처리하게될 작업입니다.** 그리고 여기서 주목해야 할 것은 람다 함수의 파라미터인데요, **부모 task의 리턴 타입이 int 타입이기 때문에 자식 task의 함수 파라미터는 int 타입을 지정해야 합니다.** 만약 부모 task의 리턴 타입과 다르면 컴파일 오류가 나게 되죠. 그리고 then 메서드는 결과값으로 자식 task 객체를 리턴합니다. 그러므로 마지막에 호출된 wait 메서드는 자식 task에 의해 호출된 것임을 알 수 있습니다. task 연결 작업의 코드를 풀어서 써보면 다음과 같이 표현됩니다.

|  |
| --- |
| **auto t2 = t.then([](int result)** {     wcout << result << endl; }); **t2.wait();** |

이번에는 task의 then 메서드를 통하여 task의 연결 체인을 구성해 보도록 하겠습니다. 연결 체인이라고 해서 별 다른 것은 없구요, task를 여러 번에 걸쳐서 연결하는 것을 이야기합니다.**task를 연결 하는 데에는 횟수의 제한이 없기 때문에 상황에 따라 부모 task - 자식 task 구조가 연속적으로 발생하는 체인 구조로 설계할 수 있습니다.** 아래 예제에서는 정수 값의 증가 처리를 task의 연결 체인으로 구현한 간단한 예를 보여줍니다.

|  |
| --- |
| #include <ppltasks.h> #include <iostream>  using namespace concurrency; using namespace std;  int wmain() {     auto t = create\_task([]() -> int     {         return 0;     });      // 입력된 정수를 +1 증가시키는 람다 함수 생성     auto increment = [](int n) { return n + 1; };      // task 체인을 실행하고 결과값을 출력    **int result = t.then(increment).then(increment).then(increment).get();**     wcout << result << endl; }  //실행결과  3 |

task의 then 메서드가 자식 task의 객체를 리턴하기 때문에 위 예제와 같이 then 메서드를 연속적으로 사용함으로서 연결 체인을 만들어나갈 수 있습니다. 여기서는 정수값 증가 처리가 3개의 연결된 task에 의해 처리되어 결과값 3을 출력하게 됩니다. 이해하기 어렵지 않죠? ^^

**value 기반의 연결과 task 기반의 연결**

앞서 task 클래스의 then 메서드 사용 시 자식 task의 함수 파라미터 타입은 부모 task의 리턴 타입에 의해 결정된다고 설명하였고 자식 task 구현 시 int 타입을 파라미터 타입으로 전달받아 사용하였습니다. 설명이 조금 늦었지만 **이러한 방식을 value 기반의 연결(Value-Based Continuations)이라 부릅니다.**그리고 또 한 가지의 방법은 부모 task의 task 타입을 전달받는 방법입니다. 만약 부모 task의 타입이 task<int>라면 자식 task의 함수 파라미터로 int 대신 task<int> 타입을 사용할 수 있습니다.**이 방식은 task 기반의 연결(Task-Based Continuations) 라고 부르며 다음과 같이 사용합니다.**

|  |
| --- |
| #include <ppltasks.h> #include <iostream>  using namespace concurrency; using namespace std;  int wmain() {     auto t = create\_task([]() -> int     {         return 42;     });      **t.then([](task<int> result)**     {         wcout << **result.get()** << endl;     }).wait(); }    //실행결과  42 |

위 예제와 달라진 것은 자식 task의 함수 파라미터가 int에서 task<int>로 변경되었다는 것과 부모 task의 결과 값을 이용하기 위해 get 메서드를 호출했다는 점입니다. 이 외에도 value 기반의 연결과 task 기반의 연결에는 큰 차이점이 있습니다. 그것은 부모의 task의 취소 또는 예외 발생 시에 동작 방식의 차이인데요, **value 기반의 연결은 부모 task가 취소되거나 예외가 발생하면 자식 task는 실행되지 않지만 task 기반의 연결에서는 부모 task가 취소되거나 예외가 발생하더라도 자식 task는 실행됩니다.** task 기반의 연결에서 부모 task가 취소되었을 경우 자식 task에서 get 메서드를 호출하면 concurrency::task\_canceled 예외가 발생하게 되므로 이를 처리하는 코드을 넣어줘야 합니다. task의 취소와 예외에 관한 이야기는 이후 강좌에서 다룰 예정이니 여기서는 이 정도 차이점만 알고 넘어가시면 됩니다.