**task 그룹**

**task 그룹은 여러 개의 task를 그룹에 추가하여 실행하고 종료를 대기하거나 취소 작업을 가능하게 하는 객체입니다.** 앞선 강좌에서 여러 개의 task가 모두 종료될 때까지 대기하기 위하여 각각의 task를 생성하여 실행하고 when\_all 함수를 사용하여 대기하도록 구현하였습니다. 이와 같은 동작을 task 그룹을 이용하면 task 그룹의 wait 메서드를 통하여 구현할 수 있으며 여러 개의 연관된 task의 취소 작업도 task 그룹의 cancel 메서드를 이용하여 구현이 가능합니다. task 그룹의 취소에 대해서는 PPL 강좌 후반부에서 자세히 설명하도록 하겠습니다.

**task\_handle**

**task 그룹에서는 그룹의 구성 요소로 task 객체를 직접 사용하지 않고 task를 캡슐화한 task\_handle 객체를 사용합니다.**task\_handle 객체는 make\_task 함수를 통하여 생성할 수 있으며 이는 task 객체를 생성하는 create\_task 함수의 사용법과 매우 유사합니다. **하지만 make\_task에 의하여 생성된 task\_handle은 task와는 달리 생성과 동시에 실행되지 않으며 task 그룹의 run 메소드를 통해 실행됩니다.** 일반적으로 task\_handle을 직접 사용하는 경우는 거의 없으므로 task\_handle은 task 그룹의 한 구성 요소라고 생각하시면 됩니다.

아래 예제에서는 task 그룹을 표현하는 클래스 중 하나인 structured\_task\_group 클래스를 이용하여 여러개의 task를 동시에 실행하고 대기하는 방법을 보여줍니다.

|  |
| --- |
| #include <ppl.h> #include <iostream>  using namespace concurrency;  int wmain() {      // make\_task 함수를 이용하여 몇 가지 task를 정의합니다.     **auto task1 = make\_task([] { wcout << L"Task 1" << endl; });**      auto task2 = make\_task([] { wcout << L"Task 2" << endl; });     auto task3 = make\_task([] { wcout << L"Task 3" << endl; });       // structured\_task\_group 객체를 생성하여 task를 동시에 실행합니다.     **structured\_task\_group tasks;**      **tasks.run(task1);**     tasks.run(task2);     tasks.run(task3);     **tasks.wait();** }    // 실행결과  Task 2 Task 1 Task 3 |

예제를 살펴보면 먼저 make\_task 함수를 이용하여 3개의 task\_handle을 생성합니다. make\_task 함수는 create\_task와 마찬가지로 함수형을 파라미터로 받기 때문에 여기서는 람다 함수를 이용하여 task의 처리 작업을 구현하였습니다. task\_handle 생성 이후에는 task 그룹 객체를 생성하고 run 메서드를 이용하여 각각의 task를 동시에 실행하고 wait 메서드를 호출하여 모든 작업이 끝날때까지 대기한 후 종료합니다. 위 예제의 출력 결과의 순서는 실행 시 마다 변경 될 수 있으며 task를 실행하고 대기하는 코드는 run\_and\_wait 메서드를 사용하여 다음과 같이 작성할 수도 있습니다.

|  |
| --- |
| tasks.run(task1); tasks.run(task2); **tasks.run\_and\_wait(task3);** |

**task\_group과 structured\_task\_group**

PPL에서는 task 그룹을 표현한 두 가지 클래스를 제공합니다. 바로 task\_group 클래스와 structured\_task\_group 클래스인데요, 지금부터는 두 가지 클래스의 차이점 위주로 설명하도록 하겠습니다. 일단 두 가지 클래스가 제공하는 메서드의 종류는 차이가 없습니다. **두 가지 클래스 모두 task의 실행과 대기를 위한 run, run\_and\_wait, wait 메서드와 취소를 위한 cancel, is\_canceling 메서드를 제공하며 각 메서드의 역할 또한 동일합니다.**run 메서드는 task를 task 그룹에 추가하여 실행하는 메서드이고 wait는 task 그룹에 포함된 모든 task의 종료를 대기하는 메서드입니다. run\_and\_wait 메서드는 이름만 봐도 run과 wait 메서드를 순서대로 수행하는 메서드임을 알 수 있습니다. 그렇다면 두 클래스 사이에 어떤 차이점이 있는지 알아보도록 하죠.

**첫 번째 차이점은 메서드 호출의 thread-safe 여부이고 이것은 task\_group과 structured\_task\_group를 구분하는 가장 큰 차이점입니다.** task\_group 클래스는 thread-safe 하기 때문에 여러 스레드에서 동시에 실행하고 대기하며 취소하는 코드를 작성할 수 있습니다. 반면 structured\_task\_group은 thread-safe 하지 않기 때문에 여러 스레드에서 동시에 실행하고 대기하는 코드를 작성할 수 없죠.**thread-safe의 의미는 task 그룹을 사용(메서드 호출)하는 스레드 사이의 동기화 안전성을 이야기하는 것이지 task 그룹 내에서 실행되는 task 간의 동기화 안전성을 이야기하는 것이 아님을 꼭 유념하시기 바랍니다**.

위에서 설명한 바와 같이 task\_group의 모든 메서드는 여러 스레드에서 동시에 호출이 가능하고 structured\_task\_group의 메서드는 반드시 하나의 스레드에서만 호출되어야 합니다. **여기서 예외 사항 한 가지는 structured\_task\_group를 사용하더라도 cancel 메서드와 is\_canceling 메서드는 다른 스레드에서 호출할 수 있다는 것입니다.** 이 예외 규칙에 의해 task 그룹에 포함된 자식 task가 task 내부에서 task 그룹을 cancel 하는것이 가능해집니다.

**두 번째 차이점은 task\_group은 wait 메서드 또는 run\_and\_wait 메서드를 호출하여 모든 task의 종료를 대기하는 중에도 다른 스레드에서 또 다른 task를 추가하여 실행할 수 있다는 점입니다.** structured\_task\_group에서는 다른 스레드에서 run 메서드를 호출할 수 없기 때문에 이러한 특징은 어떻게 보면 당연한 결과일 수 있겠네요.

**세 번째 차이점은 run 메서드의 파라미터입니다. 두 가지 클래스 모두 run 메서드의 파라미터로 task\_handle 타입을 전달 받지만 task\_group 클래스는 task\_handle 타입 이외에도 함수형을 전달 받는 오버로드 함수가 있습니다**. 만약 run 메서드의 파라미터로 직접 함수형을 전달하여 호출하면 task\_group은 내부적으로 task\_handle을 생성하여 관리합니다. run\_and\_wait 메서드의 경우에는 두 가지 클래스 모두 함수형을 전달받는 오버로드 함수가 있어서 함수형을 직접 전달하여 호출이 가능합니다.

PPL에서는 내부적으로 structured\_task\_group를 이용하여 구현 task의 병렬 처리를 구현한 parallel\_invoke 함수를 제공합니다. **parallel\_invoke 함수를 이용하면 structured\_task\_group를 직접 이용하여 구현한 것보다 간결하고 직관적인 코드를 작성할 수 있으므로 parallel\_invoke 함수로 구현될 수 없는 경우를 제외하고는 structured\_task\_group 대신에 parallel\_invoke 함수를 이용하는것이 좋습니다.** parallel\_invoke 함수는 이후 PPL 병렬 알고리즘 강좌에서 자세히 설명하도록 하겠습니다.

지금까지 task\_group과 structured\_task\_group에 대하여 살펴보았습니다. 딱딱하게 설명만 길게 늘어놓아서 잘 이해가 되셨는지 모르겠네요. 그럼 마지막으로 두 가지 클래스를 이용한 간단한 예제를 살펴보면서 추가적인 설명을 드리도록 하겠습니다.

|  |
| --- |
| #include <ppl.h> #include <sstream> #include <iostream>  using namespace concurrency; using namespace std;  // 메시지를 화면에 출력 template<typename T> void print\_message(T t) {     wstringstream ss;     ss << L"Message from task: " << t << endl;     wcout << ss.str(); }  int wmain() {     // structured\_task\_group 객체와 task\_group 객체를 생성합니다.     structured\_task\_group tg1;     **task\_group tg2;**      auto task1 = make\_task([&tg2]()     {         // task1 내부에서 tg2에 추가될 task를 생성하여 실행합니다.      **tg2.run([] { print\_message(L"Hello"); });**         tg2.run([] { print\_message(42); });     });     auto task2 = make\_task([&tg2]()     {         // task2 내부에서 tg2에 추가될 task를 생성하여 실행합니다.         tg2.run([] { print\_message(3.14); });     });      // tg1을 실행하고 모든 task가 종료될 때까지 대기합니다.     tg1.run(task1);     tg1.run\_and\_wait(task2);      //tg2의 모든 task가 종료될 때까지 대기합니다.     tg2.wait(); }  // 실행결과  Message from task: Hello Message from task: 3.14 Message from task: 42 |

이번 강좌를 차근차근 보신 분들이라면 이번 예제도 읽어 내려가는 수준 정도로 충분히 이해하셨으리라 생각합니다. 앞서 모두 설명 드린 내용이지만 중요한 부분을 한 번 더 짚어보겠습니다. 이번 예제에서 주목해야 할 것은 task\_group 객체인 tg2 인데요, 먼저 run 메서드를 호출 할 때 task\_handle 객체가 아닌 람다 함수를 직접 파라미터로 전달하고 있습니다. 이것은 앞서 설명 드린 task\_group 클래스의 특징이라고 할 수 있습니다. 그리고 run 메서드가 호출되는 위치를 보면 메인 스레드가 아닌 별도의 스레드인(작업 스케줄링에 따라 메인스레드에서 동작할 수도 있음) task1과 task2 내부에서 호출되고 있음을 볼 수 있는데 이 것 역시 task\_group의 메서드가 thread-safe 하게 사용될 수 있기 때문에 가능한 것이죠. 이번 예제의 출력 결과의 순서 역시 실행할 때 마다 달라질 수 있습니다.