2010년 8월 28일

Visual Studio Camp #1

About Concurrency Runtime

임준환

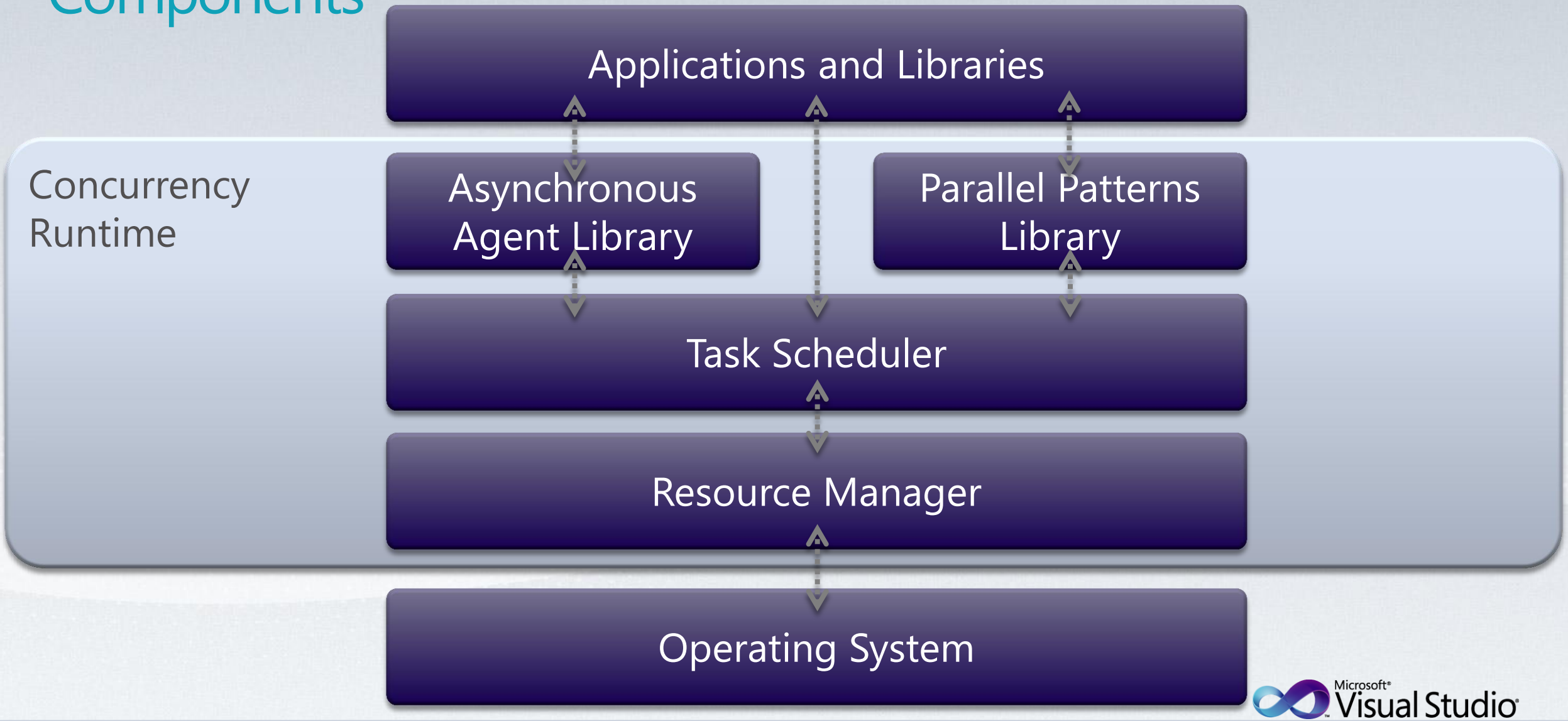
Concurrency Runtime 을 이용한 멀티 코어 활용하기

Agenda

- Concurrency Runtime
- Parallel Patterns Library(PPL)
- PPL – Parallel Algorithms
- PPL – Parallel Containers & Object
- PPL – Task Parallelism
- Asynchronous Agents Library(AAL)
- Task Scheduler
- Resource Manager

Concurrency Runtime

Components



Concurrency Runtime
Introduce

- 협조적 스케줄링
  - 바쁜 리소스의 작업을 한가로운 리소스의 작업으로 이동
  - OS 의 선점형 스케줄링과 협조적 스케줄링의 조합
- 작업의 추상화
  - 멀티 스레드 프로그래밍 라이브러리지만 스레드를 제어하지 않음
- C++ 스러운 구현
  - 객체 지향과 일반적 프로그래밍 모델 사용
  - 차세대 C++ 표준인 tr1 과 C++0x 문법을 사용

Concurrency Runtime

Parallel Patterns Library

Concurrency Runtime

Parallel Patterns Library

- Parallel Algorithms
  - parallel_for
  - parallel_for_each
  - parallel_invoke
- Parallel Containers & Object
  - concurrent_vector
  - concurrent_queue
  - combinable
- Task Parallelism
  - task_group
  - structured_task_group

Parallel Patterns Library

Parallel Algorithms

parallel_for

parallel_for_each

parallel_invoke

Parallel Algorithms

parallel_for

for

```
for( int i = 0; i < 100; ++i )
{
        DoSomething( i );
}
```

parallel_for

```
parallel_for( 0, 100, []( int i )
{
        DoSomething( i );
} );
```

Parallel Algorithms

parallel_for_each

```
for_each( v.begin(); v.end(); []( int i )
{
        DoSomething( i );
} );
```

for_each

parallel_for_each

```
parallel_for_each( v.begin(); v.end(); []( int i )
{
        DoSomething( i );
} );
```

Parallel Algorithms

parallel_invoke

?

```
prime_count1 = count_prime( 0, 10000 );
prime_count2 = count_prime( 10000, 20000 );
prime_count3 = count_prime( 20000, 30000 );
prime_count4 = count_prime( 30000, 40000 );
```

```
parallel_invoke(
        [&]{ prime_count1 = count_prime( 0, 10000 ); },
        [&]{ prime_count2 = count_prime( 10000, 20000 ); },
        [&]{ prime_count3 = count_prime( 20000, 30000 ); },
        [&]{ prime_count4 = count_prime( 30000, 40000 ); } );
```

parallel_invoke

Parallel Patterns Library

Parallel Containers & Object

concurrent_vector

concurrent_queue

combinable

Parallel Containers & Object

concurrent_vector & concurrent_queue

concurrent_vector

- 크기가 변경되어도 반복자 유효
- 중간 요소의 삽입과 제거 금지
- 비연속적인 메모리
- 요소의 값 수정은 동시성 보장 안 함

concurrent_queue

- 삽입과 제거는 동시성 보장
- front() + pop() = try_pop()
- back() 제공 안 함
- 반복자는 동시성 보장 안 함

Parallel Containers & Object

combinable

- 병렬 처리 중 각 스레드 별 로컬 저장소
  - local() 을 이용한 접근
- 병렬 처리 후 연산 결과가 저장된 로컬 저장소 병합
  - combine() 을 이용한 병합

```
combinable< unsigned int > result;
parallel_invoke(
        [&]{ result.local() += count_prime( 0, 10000 ); },
        [&]{ result.local() += count_prime( 10000, 20000 ); },
        [&]{ result.local() += count_prime( 20000, 30000 ); },
        [&]{ result.local() += count_prime( 30000, 40000 ); } );
prime_count = result.combine( plus< unsigned int >() );
```

Parallel Patterns Library

Task Parallelism

task_group

structured_task_group

Task Parallelism

task_group

task_group	structured_task_group
스레드로부터 안전함	스레드로부터 안전하지 않음
task 를 직접 관리할 필요 없음	task 를 직접 관리해야 함
재사용 가능	재사용 불가능

Task Parallelism

task_group

```
task_group tasks;
tasks.run( [&] { prime_count1 = count_primes( 0, 10000 ); } );
tasks.run( [&] { prime_count2 = count_primes( 10000, 20000 ); } );
tasks.run( [&] { prime_count3 = count_primes( 20000, 30000 ); } );
tasks.run( [&] { prime_count4 = count_primes( 30000, 40000 ); } );
tasks.wait();
```

Task Parallelism

structured_task_group

```
auto task1 = make_task( [&] { prime_count1 = count_primes( 0, 10000 ); } );
auto task2 = make_task( [&] { prime_count2 = count_primes( 10000, 20000 ); } );
auto task3 = make_task( [&] { prime_count3 = count_primes( 20000, 30000 ); } );
auto task4 = make_task( [&] { prime_count4 = count_primes( 30000, 40000 ); } );
structured_task_group tasks;
tasks.run( task1 );
tasks.run( task2 );
tasks.run( task3 );
tasks.run( task4 );
tasks.wait();
```

Concurrency Runtime
Synchronization

critical_section

reader_writer_lock

event

Synchronization

critical_section

- 임계영역을 나타내는 객체
- 블록 단위 잠금 제공

```
critical_section cs;
cs.lock();
...
cs.unlock();

{
        critical_section::scoped_lock auto_lock( cs );
        ...
}
```

Synchronization

reader_writer_lock

- 쓰기 작업과 읽는 작업을 구분하는 잠금 객체
- Windows API 의 Slim Reader Writer Lock 과 같은 기능

```
reader_writer_lock rwl;
rwl.lock();
...
rwl.unlock();
{
        reader_writer_lock::scoped_lock
        auto_lock( cs );
...
}
```

```
reader_writer_lock rwl;
rwl.lock_read();
...
rwl.unlock_read();
{
        reader_writer_lock::scoped_lock_read
        auto_read_lock( swl );
        ...
}
```

Synchronization

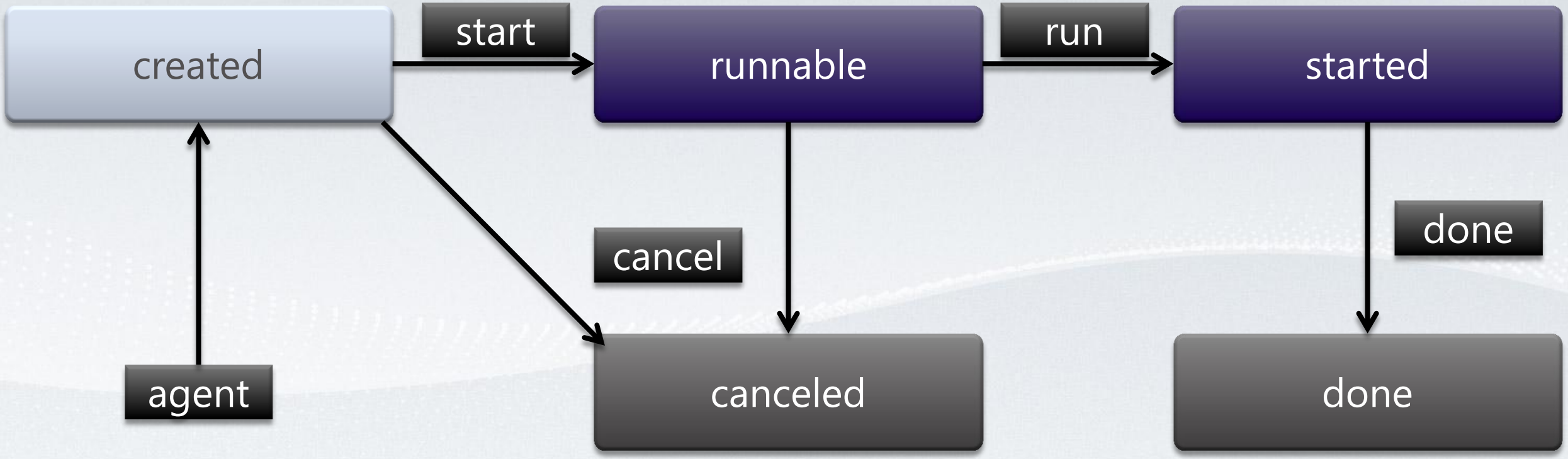
event

- 상태 변화를 감시하는 객체
- Windows API 의 EVENT 객체와 동일
- 협조적 대기 사용

```
event e1;
e1.set();
e1.wait();
e1.reset();
event e2;
event* events[2] = { &e1, &e2 };
event::wait_for_multiple( events, 2, true );
```

Concurrency Runtime
Asynchronous Agents Library

- 스레드를 객체화



Concurrency Runtime

Task Scheduler

- 컴퓨팅 리소스에 관련된 정책 설정
- 예약된 작업의 그룹화
- 메모리 할당 및 해제
- Context 객체를 통한 대기와 해제

Concurrency Runtime

Resource Manager

- Task Scheduler 와 통신
- 컴퓨팅 리소스 초기화
- 컴퓨팅 리소스 재할당

Demo

Thank you.

mumbi@daum.net

한국 Visual Studio 공식 팀
http://vsts2010.net
@vsts2010