



고급프로그래밍

멀티스레딩

Professor Jeong, Mun-Ho

Robot Vision & Intelligence Laboratory
Kwangwoon University
(02-940-5625, mhjeong@kw.ac.kr)

Schedule

주차	주제		과제	퀴즈
1	과목소개	교과목 소개 (1), C++ 시작 (2)		
2	C++	C++ 프로그래밍의 기본 (3), 클래스와 객체 (4)	1	1
3		객체생성과 사용 (5)	2	2
4		함수와 참조 (6, 3/26), 복사 생성자와 함수중복(7)	3	3
5		static, friend, 연산자중복 (8, 4/2), 연산자중복 상속(9)	4	4
6		상속 (10, 4/9), 가상함수와 추상클래스 (11)		5
7		템플릿과 STL (12, 4/16), 입출력(13)	5	
8		중간고사		
9		파일 입출력(14), 예외처리 및 C 사용(15)		6
10		람다식(16, 5/7) , 멀티스레딩(17, 5/9)	6	7
11		멀티스레딩, 고급문법		
12		고급문법		
13	병렬프로그래밍	병렬프로그래밍		
14		병렬프로그래밍		
15		기말고사		

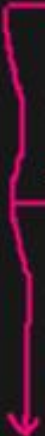
오늘의 학습내용

- 멀티스레딩

멀티스레딩

Revisit : auto

```
3 #include <iostream>
4 #include <vector>
5
6 using namespace std;
7
8 int main()
9 {
10     vector<int> vec;
11     vec.push_back(1);
12     vec.push_back(2);
13     vec.push_back(3);
14
15     for(auto &each : vec) // vector 요소 변경 가능
16     {
17         each++;
18     }
19
20     for(auto each : vec) // vector 요소 변경 불가
21     {
22         cout << each << endl;
23     }
24
25     return 0;
26 }
```



- 범위기반 for문
- for문은 STL과 배열에 접근 가능

Revisit : Function Object

- To behave like a function despite being an object
- Realized by operator overloading: operator ()

```
2 #include <iostream>
3 using namespace std;
4
5 class Plus
6 {
7 public:
8     int operator( )(int a, int b)
9     {
10         return a + b;
11     }
12 };
13
14 int main()
15 {
16     Plus pls;
17
18     cout << "pls(10, 20): " << pls(10, 20) << endl;
19     return 0;
20 }
```

explicit call: pls.operator()(10,20)

implicit call

멀티스레드

- 복수의 계산을 동시에 병렬로 수행가능
- 멀티프로세서, 멀티코어 장점 활용 가능
 - CPU 클럭을 높이면 전력소모와 발열이 커짐
 - CPU 코어 개수 향상 정책: 듀얼, 쿼드, 12-, 16-, 80-....
- 여태까지의 멀티스레드 **API**는 운영체제 또는 플랫폼에 종속적
 - pthread, boost::thread, ...
- **C++11** 부터 멀티스레드 라이브러리 포함

How to Start a Thread

- Using Function Pointer

```
std::thread variable_name ( function_name, param1, param2,... )
```

```
#include <iostream>
#include <thread>

using namespace std;

void counter(int id, int numIterations)
{
    for (int i = 0; i < numIterations; ++i)
    {
        cout << "Counter " << id << " has value ";
        cout << i << endl;
    }
}
```

```
int main()
{
    //make sure cout is
    //thread-safe
    cout.sync_with_stdio(true);

    thread t1(counter, 1, 6);
    thread t2(counter, 2, 4);

    t1.join(); //to wait for finishing t1
    t2.join();

    return 0;
}
```


How to Start a Thread

- Using a class method

```
class Task
{
public:
    void execute(string command)
    {
        for(int i = 0; i < 5; i++)
        {
            cout<<command<<" ::
                "<< i << endl;
        }
    }
};
```

```
int main()
{
    Task * taskPtr = new Task();

    // Create a thread using member function
    thread th(&Task::execute, taskPtr, "Sample Task");

    th.join();

    delete taskPtr;
}
```

How to Start a Thread

- Using a static class method

```
class Task
{
public:
    static void execute(string command)
    {
        for(int i = 0; i < 5; i++)
        {
            std::cout<<command<<" ::
                "<<i<<std::endl;
        }
    }
};
```

```
int main()
{
    //Task * taskPtr = new Task();

    // Create a thread using member function
    thread th(&Task::execute, "Sample Task");

    th.join();

    //delete taskPtr;
}
```

인자가 없으면 생략가능

How to Start a Thread

- Using Function Object

Overloading `operator()` in the class declaration,

```
std::thread variable_name { class_constructor( ... ) }
```

```
class_name object_name( ... );  
std::thread thread_name ( object_name )
```

```
std::thread thread_name ( class_constructor( ... ) )
```

```
std::thread thread_name ( class_constructor( ) ) → compile error
```

How to Start a Thread

- Using Function Object

```
#include <iostream>
#include <thread>

using namespace std;

class Counter
{
    int _nID, _nItr;

public:
    Counter(int id, int nItr) : _nID(id), _nItr(nItr){ }
    void operator()() const
    {
        for(int i = 0; i < _nItr; ++i)
        {
            cout << "Counter " << _nID << " has value ";
            cout << i << endl;
        }
    }
};
```

```
int main()
{
    cout.sync_with_stdio(true);

    //c++11 initialization syntax
    thread t1{Counter(1,20)};

    //using named variable
    Counter c(2,12);
    thread t2(c);

    //using temporary
    thread t3(Counter(3,10));

    t1.join();
    t2.join();
    t3.join();

    return 0;
}
```

How to Start a Thread

- Using Function Object

```
#include <iostream>
#include <thread>

using namespace std;

class Counter
{
    int    _nID, _nItr;

public:
    Counter(int id, int nItr) : _nID(id), _nItr(nItr){ }
    void operator()() //const
    {
        for(int i = 0; i < _nItr; ++i)
        {
            cout << "Counter " << _nID << " has value ";
            cout << i << endl;
        }
        _nItr ++;
    }
    int Itr( ){ return _nItr; }
};
```

```
int main()
{
    cout.sync_with_stdio(true);

    Counter c(2,12);
    thread t1(c);
    //thread t1(ref(c))

    t1.join(); //to wait for finishing t1

    cout << "Itr = " << c.Itr() << endl;
}
```

How to Start a Thread

- Using Function Object

```
#include <thread>
#include <iostream>
#include <functional> // for std::ref
using namespace std;

class PrintThis
{
public:
    void operator( )( ) const
    {
        cout << "this=" << this << endl;
    }
};
```

```
int main()
{
    PrintThis x;
    x( );
    thread t(ref(x));
    t.join();

    thread t2(x);
    t2.join();
}
```



C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
this=0x62fe37
this=0x62fe37
this=0x761758
Press <RETURN> to close this window...
```

How to Start a Thread

- Using Lambda Expression

```
#include <iostream>
#include <thread>

using namespace std;

int main()
{
    //make sure cout is
    //thread-safe
    cout.sync_with_stdio(true);

    thread t1( [](int id, int num) {
        for (int i = 0; i < num; ++i)
        {
            cout << "Counter " << id << " has value ";
            cout << i << endl;
        }
    }, 1, 5);

    t1.join(); //to wait for finishing t1
}
```

실습

- auto와 람다함수를 이용하여 아래를 완성하시오.

```
int main()
{
    cout.sync_with_stdio(true); //make sure cout is thread-safe

    [redacted] = [ ](int id, int num)
    {
        for (int i = 0; i < num; ++i) {
            cout << "Counter " << id << " has value ";
            cout << i << endl;
        }
    };

    //thread t1생성 및 시작 (id = 1, num = 5)
    [redacted]

    t1.join(); //to wait for finishing t1
}
```


Race Condition

- 여러 스레드가 공유메모리를 동시에 읽기/쓰기 할 때 발생
- 이때 읽기 혹은 쓰기 명령이 실행되지 않는 경우

mutex

- 공유메모리에서 `race condition` 없이 읽기/쓰기를 위해 필요
- `mutex`로 공유메모리에 락을 구하고 공유메모리를 점유함
- 다른 스레드가 락을 하고 있으면 대기함
- 공유메모리에서의 작업이 끝나면 락 해제함
- `std::mutex`
 - `lock()`, `try_lock()`, `unlock()`
 - `try_lock` : 락을 얻어내지 못하면 `false`를 반환
- `std::timed_mutex`
 - `lock()`, `try_lock()`, `unlock()`
 - `try_lock(rel_time)`, `try_lock_until(abs_time)`

예제: mutex & lock

```
#include <iostream>
#include <thread>
#include <mutex>

using namespace std;

class Counter
{
    int _nID, _nItr;
    static mutex _oMutex;

public:
    Counter(int id, int nItr) : _nID(id), _nItr(nItr)
    {}
    void operator()() const
    {
        for(int i = 0; i < _nItr; ++i)
        {
            _oMutex.lock();
            cout << "Counter " << _nID << " has value ";
            cout << i << endl;
            _oMutex.unlock();
        }
    }
};

mutex Counter::_oMutex;
```

```
int main()
{
    cout.sync_with_stdio(true);

    //c++11 initialization syntax
    thread t1{Counter(1,20)};

    //using named variable
    Counter c(2,12);
    thread t2(c);

    //using temporary
    thread t3(Counter(3,10));

    t1.join();
    t2.join();
    t3.join();

    return 0;
}
```

- ① for 문 바깥으로 뮤텍스 이동해보시오
- ② 좋은 점과 나쁜 점은 ?

lock_guard vs. unique_lock

- `mutex`를 사용하기 쉽게 하기 위한 클래스
- `lock` 객체의 소멸시에 `mutex` 락을 해제함 : 예외처리 등을 통해 `unlock`를 호출 못하는 위험 회피 가능

```
void function( ){  
    mutex.lock( );  
    if(true)  
        throw 1;  
    mutex.unlock( );  
}
```

```
void function( )  
{  
    lock_guard<mutex> lock1(mutex);  
    if(true)  
        throw 1;  
}
```

- `std::lock_guard`
 - 생성자에서 뮤텝스 락 점유 시작
 - 소멸자에 의해 `unlock`
 - 중간에 다시 `lock`을 걸거나 `unlock` 할 수 없다.
- `std::unique_lock`
 - 객체 먼저 생성 뒤, 필요한 시점에 락 점유 시도 가능
 - 소멸하기 전에 다시 `lock` 혹은 `unlock`을 할 수 있다.
 - `unique_lock<mutex> lock1(mut1);`
 - `unique_lock<mutex> lock2(mut1, defer_lock_t());`

예제: lock_guard

```
#include <iostream>
#include <thread>
#include <mutex>

using namespace std;

class Counter
{
    int _nID, _nItr;
    static mutex _oMutex;

public:
    Counter(int id, int nItr) : _nID(id), _nItr(nItr)
    {}
    void operator()() const
    {
        for(int i = 0; i < _nItr; ++i)
        {
            lock_guard<mutex> lock(_oMutex);
            cout << "Counter " << _nID << " has value ";
            cout << i << endl;
        }
    }
};

mutex Counter::_oMutex;
```

예제: unique_lock

```
1 #include <iostream>
2 #include <thread>
3 #include <mutex>
4
5 using namespace std;
6
7 class Counter
8 {
9     int _nID, _nItr;
10     static timed_mutex _oTimedMutex;
11
12 public:
13     Counter(int id, int nItr) : _nID(id), _nItr(nItr)
14     {
15     }
16     void operator()() const
17     {
18         for(int i = 0; i < _nItr; ++i)
19         {
20             unique_lock<timed_mutex> lock(_oTimedMutex, chrono::milliseconds(200));
21
22             if(lock)
23             {
24                 cout << "Counter " << _nID << " has value ";
25                 cout << i << endl;
26             }
27             else{
28                 //lock not obtained in 200 ms
29             }
30         }
31     }
32 };
33
34 timed_mutex Counter::_oTimedMutex;
```

예제: try_to_lock

- Locking을 시도하고 바로 리턴

```
:  
thread t( [&]()  
{  
    while( temp!= -1)  
    {  
        this_thread::sleep_for(std::chrono::seconds(5));  
  
        unique_lock<mutex> lock( _mutex, try_to_lock);  
        if(lock.owns_lock())  
        {  
            //do something  
            temp=0;           // 스레드 종료  
        }  
    }  
});
```

스레드와 auto

```
class MyPrint
{
    string myWord;
public:
    MyPrint(const string& text)
    {
        myWord = text;
    }
    void operator( )(const string& szText) const
    {
        cout << myWord << endl;
        cout << szText << endl;
    }
};

int main()
{
    auto th1 = thread{ MyPrint("Hello,"), "World!" };
    th1.join();
    return 0;
}
```


실습

- 다음 프로그램을 수행하면 어떤 문제가 발생하는가? 프로그램을 수정하여 문제를 해결하시오.

```
int value;

void increase_value()
{
    value++;
    cout << value << endl;
}
```

```
int main()
{
    value = 0;
    thread t[10];

    for (auto i = 0; i < 1000; i++)
    {
        // 스레드 생성
        t[i] = thread(increase_value);
    }

    for (int i = 0; i < 1000; i++)
    {
        // 스레드 대기 종료
        t[i].join();
    }
}
```

