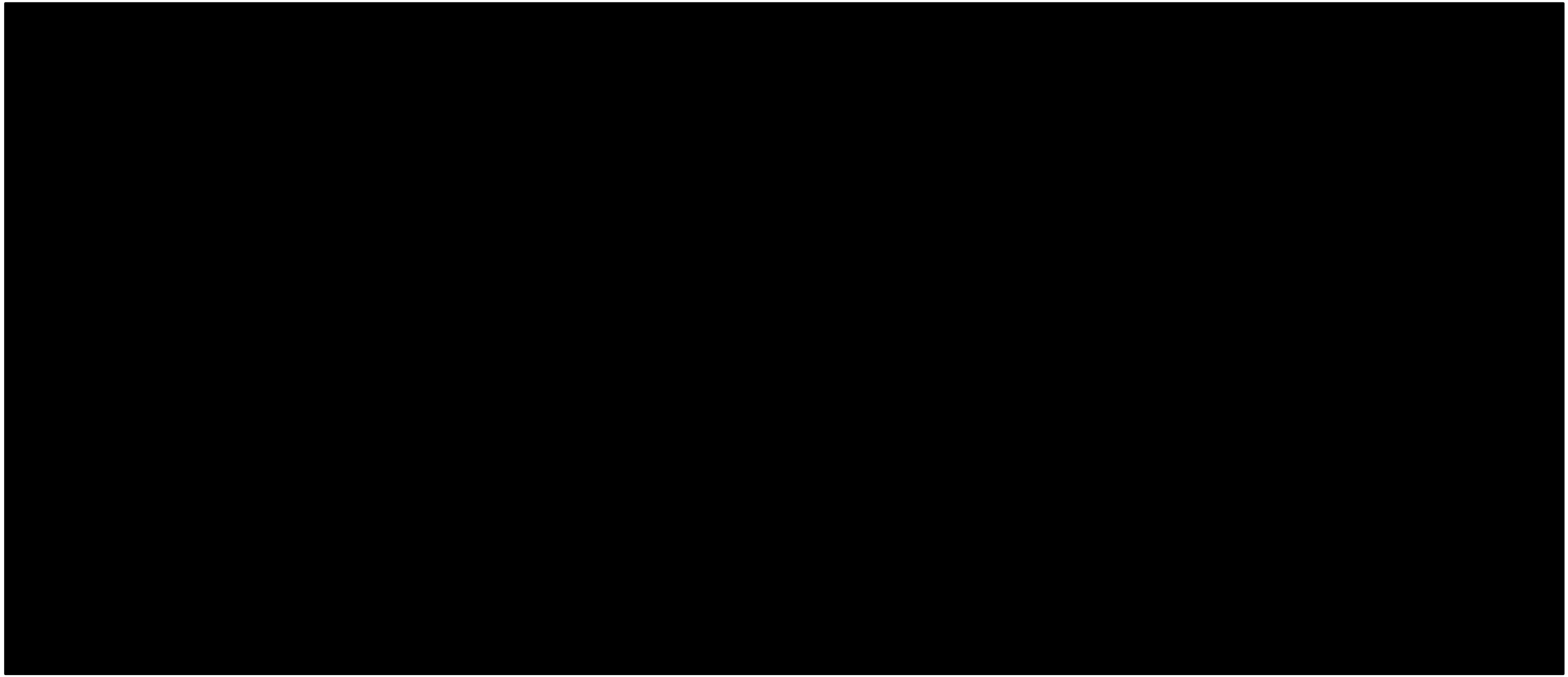
인트아이 C++ 심화 스터디

THREAD, MUTEX, SEMAPHORE

3회차 퀴즈 답안



Thread [스레드]

프로세스 = 메모리 안에서 실행 중인 프로그램

스레드 = 프로세스 안에서 코드를 동작시키는 단위

하나의 프로그램이 하나의 스레드만 사용하면 싱글 스레드, 여러 개의 스레드를 사용하면 멀티스레드

스레드는 전역 변수를 이용해 서로 값을 공유할 수 있음 (프로세스와의 차이점)

일반적인 C++ 코드는 하나의 스레드만 이용 → CPU 코어가 여러 개여도 하나 밖에 사용을 못함, 나머지 코어는 놀고 있음

프로그래머 직접 여러 개의 스레드를 사용할 수 있게 코드를 바꿔줘야 함

Thread

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

// Function Pointer
void f1() {
    for (int i = 0; i < 5; i += 1) {
        std::cout << "f1();\n";
    }
}

// Functor
struct f2 {
    void operator()(int n) {
        for (int i = 0; i < 5; i += 1) {
            std::cout << "f2(" << n << ");\n";
        }
    }
};

// Lambda
auto f3 = [](int n, int m) {
    for (int i = 0; i < 5; i += 1) {
        std::cout << "f3(" << n << ", " << m << ");\n";
    }
};

int main() {
    std::thread t1(f1);
    std::thread t2(f2(), 10);
    std::thread t3(f3, 20, 30);
    t1.join();
    t2.join();
    t3.join();
    return 0;
}
```

std::thread를 이용해 새로운 스레드 생성

스레드의 생성자에는 실행할 함수를 넣어 줌

2번째 인자부터는 함수의 인자를 전달

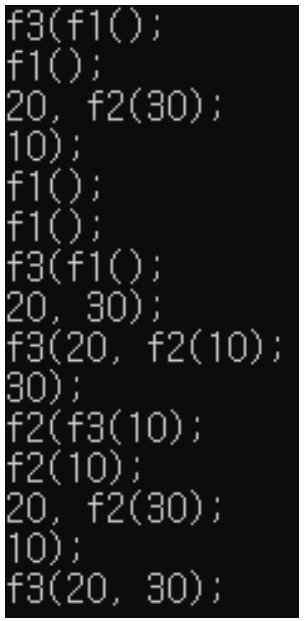
스레드를 생성했으면 join해야 함

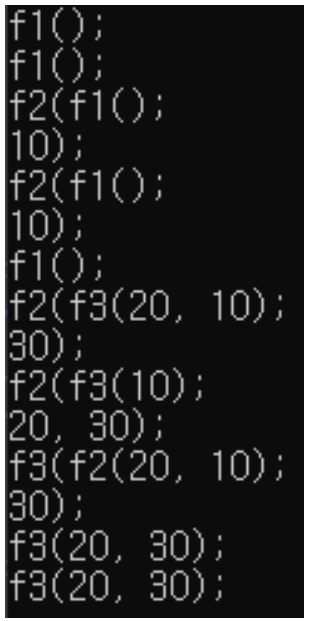
join은 생성한 스레드가 끝날 때까지 기다리는 역할

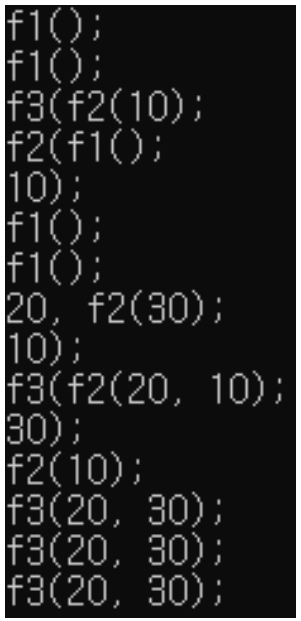
https://en.cppreference.com/w/cpp/thread/thread

Thread

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

// Function Pointer
void f1() {
    for (int i = 0; i < 5; i += 1) {
        std::cout << "f1();\n";
    }
}

// Functor
struct f2 {
    void operator()(int n) {
        for (int i = 0; i < 5; i += 1) {
            std::cout << "f2(" << n << ");\n";
        }
    }
};

// Lambda
auto f3 = [](int n, int m) {
    for (int i = 0; i < 5; i += 1) {
        std::cout << "f3(" << n << ", " << m << ");\n";
    }
};

int main() {
    std::thread t1(f1);
    std::thread t2(f2(), 10);
    std::thread t3(f3, 20, 30);
    t1.join();
    t2.join();
    t3.join();
    return 0;
}
```







매번 실행 결과가 달라짐

3개의 스레드가 경쟁하며 cout을 사용하면서 출력 결과가 엉망이 됨

Mutex [뮤텍스]

상호 배제(mutual exclusion)의 약자

여러 개의 스레드가 공유자원(cout 등)을 경쟁적으로 사용할 때, 특정 스레드만 공유자원을 독점해서 사용할 수 있도록 하는 기능

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

std::mutex mutex; 전역 변수로 mutex 선언

// Function Pointer
void f1() {
    for (int i = 0; i < 5; i += 1) {
        mutex.lock(); t1 스레드만 사용할 수 있게 mutex를 잠금, 나머지 스레드는 unlock 될 때까지 기다림
        std::cout << "f1();\n";
        mutex.unlock(); cout 사용이 끝나면 unlock 호출
    }
}
```

Mutex

```
// Functor
struct f2 {
    void operator()(int n) {
        for (int i = 0; i < 5; i += 1) {
            std::lock_guard<std::mutex> lock(mutex);
            std::cout << "f2(" << n << ");\n";
        }
    }
};

// Lambda
auto f3 = [](int n, int m) {
    for (int i = 0; i < 5; i += 1) {
        std::lock_guard<std::mutex> lock(mutex);
        std::cout << "f3(" << n << ", " << m << ");\n";
    }
};
```

std::lock_guard를 이용하면 코드가 더 깔끔해짐

```
f1();
f1();
f1();
f1();
f1();
f2(10);
f2(10);
f2(10);
f2(10);
f2(10);
f3(20, 30);
f3(20, 30);
f3(20, 30);
f3(20, 30);
f3(20, 30);
```

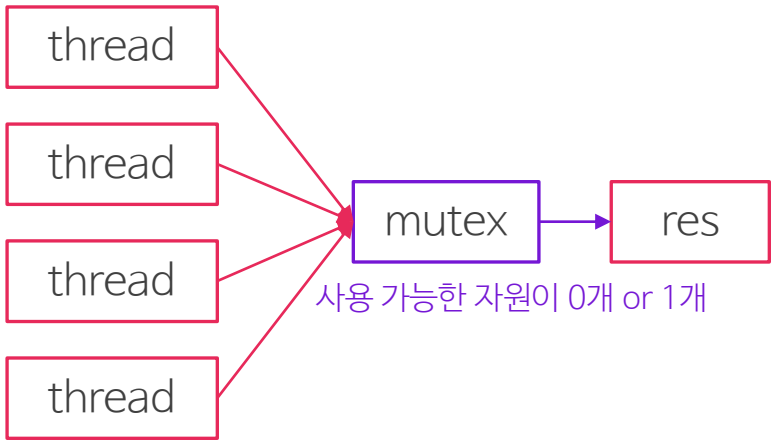
4개의 스레드로 0~100 합 계산

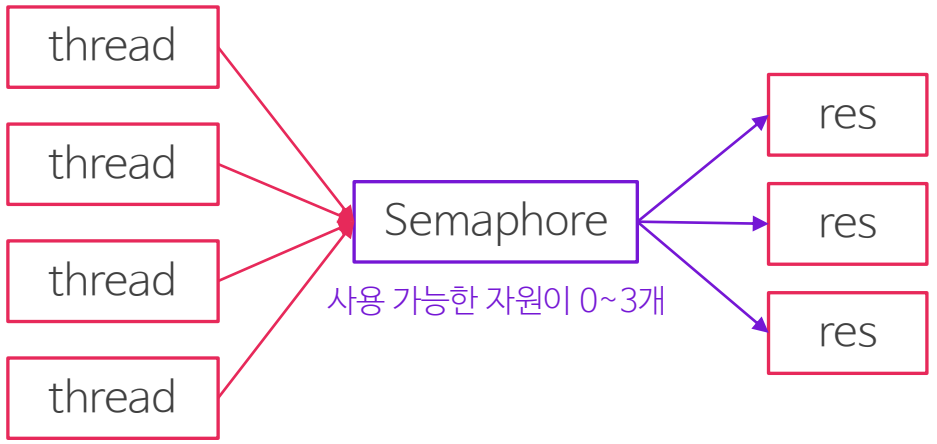
```
std::mutex mutex;
int N;

void sum(int a, int b) {
    for (int n = a; n < b; n += 1) {
        std::lock_guard<std::mutex> lock(mutex);
        N += n;
    }
}

int main() {
    std::thread t1(sum, 0, 25);
    std::thread t2(sum, 25, 50);
    std::thread t3(sum, 50, 75);
    std::thread t4(sum, 75, 101);
    t1.join();
    t2.join();
    t3.join();
    t4.join();
    std::cout << N;
    return 0;
}
```

Semaphore [세마포어]

Mutex는 가질 수 있는 상태가 2가지 (locked, unlocked)

Semaphore는 가질 수 있는 상태가 N가지인 Mutex (바꿔 말하면 Mutex는 Binary Semaphore)





Semaphore

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

std::counting_semaphore<1> mutex(1); 〈N〉에 사용 가능한 최대 자원 개수, (n)에 현재 사용 가능한 자원 개수
int N;

void sum(int a, int b) {
    for (int n = a; n < b; n += 1) {
        mutex.acquire();  lock 대신 acquire
        N += n;
        mutex.release();  unlock 대신 release
    }
}

int main() {
    std::thread t1(sum, 0, 25);
    std::thread t2(sum, 25, 50);
    std::thread t3(sum, 50, 75);
    std::thread t4(sum, 75, 101);
    t1.join();
    t2.join();
    t3.join();
    t4.join();
    std::cout << N;
    return 0;
}
```

5050

std::counting_semaphore, std::binary_semaphore

C++20 이상에서만 사용 가능