



멀티쓰레드-1

게임서버프로그래밍

정내훈

한국공학대학교 게임공학과

내용

- 병렬 처리
- 멀티 쓰레드
- 멀티 쓰레드 프로그래밍

지금 까지 - 1

- 게임서버에서 가장 중요한 것
 - 안정성
 - 성능
- 성능을 높이려면?
 - 프로그램 최적화
 - 멀티코어 활용
- 멀티코어를 활용하려면
 - 멀티쓰레드 프로그래밍이 필요

지금 까지 - 2

- 온라인 게임을 만들려면
 - 소켓 프로그래밍 필요
- 다중 접속 서버를 만들려면
 - 서버에서 동접과 같은 수의 소켓을 관리하여야 함.
- 효율적인 다중 접속 관리는?
 - IOCP가 필수
- IOCP의 특징
 - 멀티쓰레드프로그래밍을 요구한다.

멀티쓰레드

- 하나의 프로그램의 여러 곳이 동시다발적으로 실행되는 프로그래밍 기법
- 최근에 가장 많이 사용되는 병렬처리 프로그래밍 기법
- 운영체제 수업시간에 쓰레드를 배움

병렬처리

- 하나의 작업을 여러 개의 콘텍스트에서 **동시** 수행하는 것
 - 콘텍스트 => CPU의 실행 상태 => PC를 포함한 모든 레지스터의 값
 - 여러 대의 컴퓨터를 사용하는 경우
 - 분산시스템, 클러스터
 - SETI
 - 한대의 컴퓨터를 사용하는 경우
 - **SMP** : 여러 개의 **CPU**
 - **Multi-Core** : 여러 개의 **core**

병렬처리

- 왜 병렬처리를 하는가?
 - 한 개의 CPU의 처리속도가 너무 느리기 때문
 - 프로그램의 구조가 깔끔해 지므로
- 왜 지금 병렬처리가 각광을 받고 있는가?
 - 발열의 한계에 부딪친 클럭속도 증가
 - 제작사의 사활을 건 멀티코어 CPU의 보급
 - 콘솔, 모바일 기기의 멀티 core화
- Game Server는 25년 전부터 병렬처리!

멀티 쓰레드 복습

- 프로세스와 쓰레드
 - 프로세스 : 실행 중인 프로그램
 - 하나의 프로세스는 하나의 실행화일에서 출발한다.
 - 쓰레드 : 프로그램 실행의 흐름
 - 프로세스실행 중 프로그램이 쓰레드 생성 명령 실행

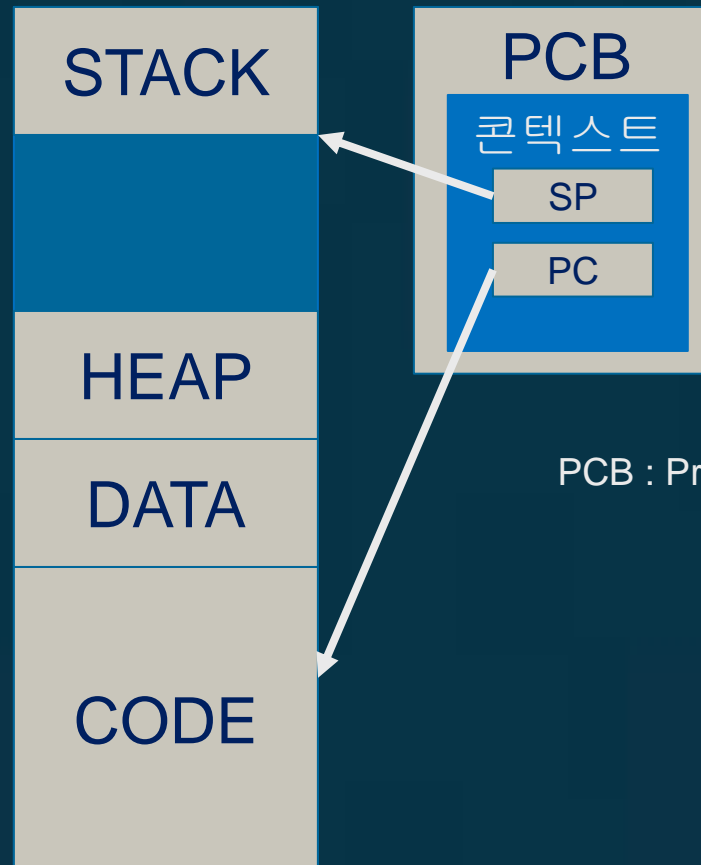
멀티 쓰레드 복습

- 프로세스와 쓰레드
 - 프로그램은 하나의 프로세스가 되어서 실행된다.
 - 처음에는 하나의 쓰레드로 실행
 - 쓰레드는 다른 쓰레드를 만들 수 있다.
 - 멀티쓰레드의 시작
 - 각각의 쓰레드는 자신의 스택을 가지고 있고, 같은 프로세스의 모든 쓰레드는 **Data**와 **Code**, **HEAP**을 공유한다.

멀티 쓰레드

- 프로세스

Process Image



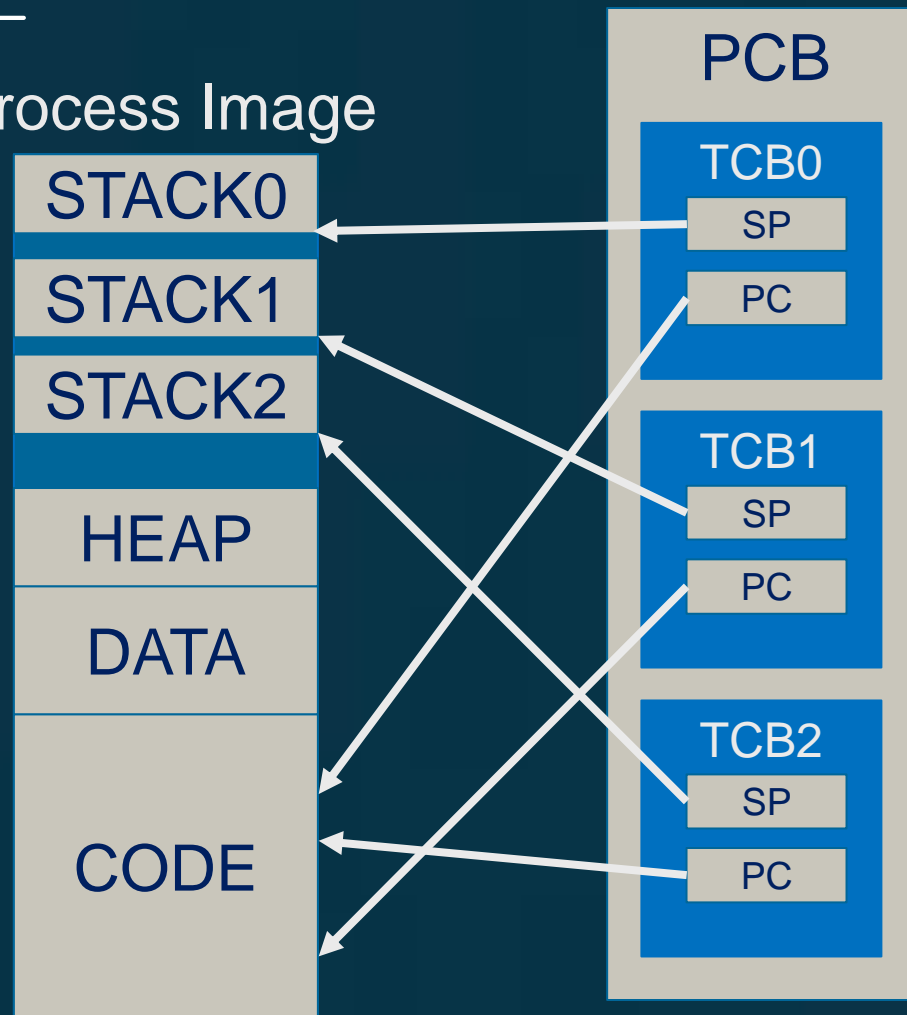
PCB : Process Control Block

멀티 쓰레드

- 멀티쓰레드

TCB : Thread Control Block

Process Image



멀티 쓰레드

- 멀티쓰레드에서의 메모리 접근
 - 메모리?? -> C의 경우 변수(variable)
 - 전역변수 : 모든 쓰레드가 공유한다.
 - 지역변수 : 쓰레드마다 따로 따로 존재한다.
 - 지역변수도 강제로 공유 가능
 - 지역변수의 주소를 전역변수에 저장하면 됨
 - 그러지 말자.
- 멀티쓰레드에서의 자원 공유
 - 모든 자원(메모리, 파일 핸들, 윈도우 핸들...)은 공유된다.

멀티 쓰레드

- 장점
 - 성능 향상
 - 빠른 응답 속도
 - 더 나은 자원 활용 (CPU Core)
 - 프로세스보다 효율적인
 - 통신 (=메모리 읽고 쓰기)
 - context switch
- 위험
 - 프로그램 복잡도 증가
 - 디버깅의 어려움 (data race, deadlock)

멀티 쓰레드

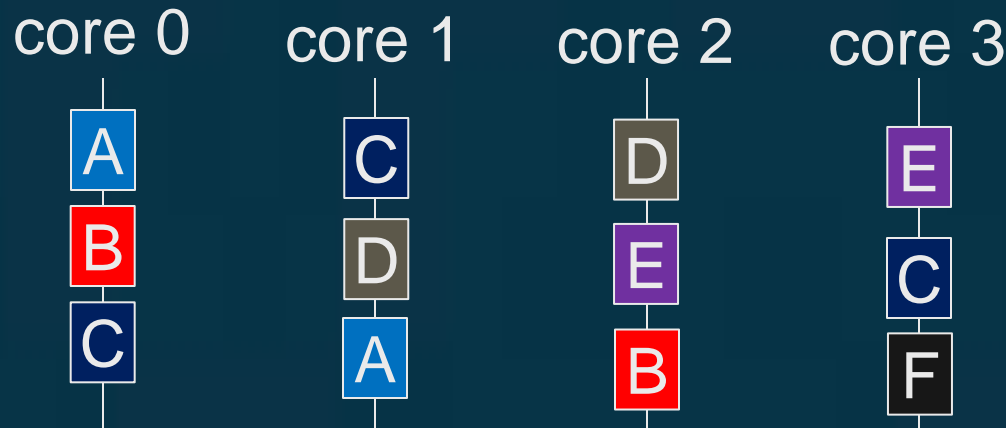
- 게임 서버에서 멀티 쓰레드를 사용하는 이유
 - 동접을 늘리기 위해서
 - 싱글 쓰레드는 1000명 정도가 한계
 - 5대의 기계로 5000명 만드는 것과의 차이?
 - 빠른 응답 속도를 위해서
 - 작업의 길이의 차이
- 실제 많은 게임 서버에서 멀티 쓰레드 사용
 - Lineage, Lineage2, AION, Lineage2 Revolution, ArchAge

멀티 쓰레드

- 두 종류의 프로그래밍 스타일
 - Heterogeneous
 - 작업을 역할 별로 나누어서 전용 쓰레드에게 맞기는 스타일
 - Homogenous (event driven, data driven)
 - 작업을 쪼개서 쓰레드 구분 없이 나누어 하는 스타일
- Game 서버는 Homogenous
 - 패킷 하나를 한 조각의 작업으로 생각
 - 동접 5000이면 최대 5000개의 조각
- Game 클라이언트는 Heterogeneous
 - 렌더링 쓰레드, 물리 엔진 쓰레드, 장면 구성 쓰레드..

멀티 쓰레드

- 주의점
 - 쓰레드의 개수가 많다고 좋은 것이 아니다
 - 프로세서/코어 의 개수에 맞추어라.
 - 코어보다 많은 쓰레드 -> 컨텍스트 스위치 부하, 반응속도 저하
 - 너무 많은 쓰레드 => 운영체제 및 하드웨어에 부담



멀티 쓰레드

- 주의점
 - 내가 사용하는 메모리의 내용이 내가 아닌 다른 쓰레드에 의해서 변경될 수 있음을 항상 유념해야 한다.
 - DATA RACE라고 부름
 - Single Core Computer도 마찬가지
 - Debugging의 어려움

멀티 쓰레드 프로그래밍

- Windows에서의 멀티 쓰레드 프로그래밍
 - Windows에서는 멀티쓰레드를 기본 지원
 - Kernel Level Thread
 - 사실상 멀티쓰레드에 특화된 OS
 - 추가 header파일, library 필요없음
 - 쓰레드를 멀티코어에 잘 분배
 - Affinity 적용
 - 쓰레드 문맥 전환
 - Windows의 scheduler가 알아서 함
 - x86 CPU에 쓰레드 전환 명령어 존재
 - 프로그램에서 OS에 요청할 수도 있음

멀티 쓰레드 프로그래밍

- Windows에서의 멀티 쓰레드 프로그래밍
 - Windows고유의 API가 존재하나 C++11의 표준을 따르는 프로그래밍 방식 권장

멀티 쓰레드 프로그래밍

- 쓰레드 만들기

- 쓰레드 객체를 생성하고 생성자에 초기값으로 실행할 함수를 넣어 준다.

```
#include <thread>

std::thread t1 { mythread };
```

- **mythread** : 새 쓰레드가 시작될 함수

```
void mythread()
{
    // 쓰레드가 실행할 프로그램
}
```

멀티 쓰레드 프로그래밍

- 쓰레드 종료 검사
 - 자식 쓰레드를 생성한 쓰레드는 생성한 쓰레드의 종료를 확인해야 한다.
 - new/delete, open/close 와 같은 개념
 - join() 메소드를 호출하면 된다.

```
std::thread t1 {mythread};  
t1.join();
```

멀티 쓰레드 프로그래밍 (실습)

- Thread를 만들어서 실행하는 프로그램 작성
 - 10개의 쓰레드를 만들어서 각자 자신의 쓰레드 ID를 출력

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

using namespace std;

void ThreadFunc(int threadid)
{
    cout << threadid << endl;
}

int main()
{
    vector <thread> my_thread;

    for (int i=0; i< 10; ++i)
        my_thread.emplace_back(ThreadFunc, i);

    for (auto &t:my_threads) t.join();
}
```

멀티 쓰레드 프로그래밍

- 멀티 쓰레드 프로그램의 성능 측정
 - `high_resolution_clock`으로 성능 측정
 - 2를 5000만번 더하는 프로그램
- 싱글 쓰레드 프로그램 부터 테스트
- 주의!! 모든 성능 측정은 **Release Mode**로 한다!

```
#include <chrono>
using namespace std::chrono;

auto t = high_resolution_clock::now();
// 측정하고 싶은 프로그램을 이곳에 위치시킨다.
auto d = high_resolution_clock::now() - t;
cout << duration_cast<milliseconds>(d).count() << " msecs\n";
```

멀티 쓰레드 프로그래밍

- 덧셈 프로그램 싱글 쓰레드

```
#include <iostream>
#include <chrono>

using namespace std;
using namespace std::chrono;

int sum;

int main()
{
    auto t = high_resolution_clock::now();
    for (auto i = 0; i < 50000000; ++i) sum = sum + 2;
    auto d = high_resolution_clock::now() - t;
    cout << "Sum = " << sum << " duration = ";
    cout << duration_cast<milliseconds>(d).count() << endl;
}
```


멀티 쓰레드 프로그래밍

- 덧셈 프로그램 싱글 쓰레드
 - 결과는?
 - 말도 안되는 실행 속도
 - 무엇이 문제인가?
 - 컴파일러가 너무 똑똑함

```
...  
  
volatile int sum;  
  
int main()  
{  
    auto t = high_resolution_clock::now();  
    for (auto i = 0; i < 50000000; ++i) sum = sum + 2;  
    auto d = high_resolution_clock::now() - t;  
    cout << "Sum = " << sum << " duration = ";  
    cout << duration_cast<milliseconds>(d).count() << endl;  
}
```

멀티 쓰레드 프로그래밍

- Thread 2개로 합을 구하는 프로그램 작성
 - 전역변수 `sum`
 - 쓰레드가 하는 일은 “`sum = sum + 2`” 오천만 **/ 2** 번 수행
 - 쓰레드 종료 후 `sum` 출력

멀티 쓰레드 프로그래밍

- 쓰레드 2개 합계 구하기 프로그램

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

using namespace std;
volatile int sum = 0;

void thread_func()
{
    for (auto i = 0; i < 25000000; ++i) sum = sum + 2;
}

int main()
{
    thread t1 { thread_func };
    thread t2 { thread_func };
    t1.join();    t2.join();
    cout << "Sum = " << sum << "\n";
}
```

멀티 쓰레드 프로그래밍

- 쓰레드2개 합계 구하기 프로그램의 결과
 - 속도는?
 - 결과는?
- 수정
 - 쓰레드 4개는?
 - 쿼드 코어 CPU에서는???

멀티 쓰레드 프로그래밍

- 다중 쓰레드

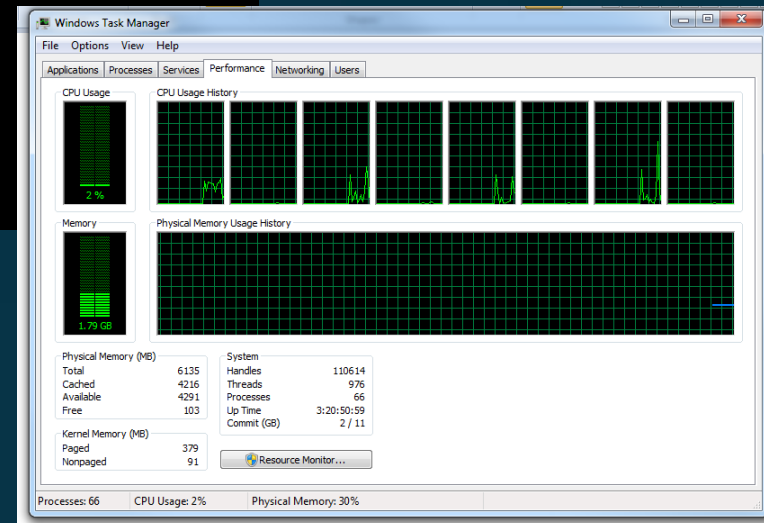
```
#include <iostream>
#include <thread>
#include <chrono>
#include <vector>
const auto MAX_THREADS = 64;
using namespace std;
using namespace std::chrono;
volatile int sum;
void thread_func(int num_threads) {
    for (auto i = 0; i < 50000000 / num_threads; ++i)    sum = sum + 2;
}
int main() {
    vector<thread> threads;
    for (auto i = 1; i <= MAX_THREADS; i *= 2) {
        sum = 0;
        threads.clear();
        auto start = high_resolution_clock::now();
        for (auto j = 0; j < i; ++j) threads.push_back(thread{ thread_func, i });
        for (auto &tmp : threads) tmp.join();
        auto duration = high_resolution_clock::now() - start;
        cout << i << " Threads,    Sum = " << sum;
        cout << "    Duration = " << duration_cast<milliseconds>(duration).count() << " milliseconds\n";
    } }
```

멀티 쓰레드 프로그래밍

- 다중 쓰레드 - 결과

```
C:\Windows\system32\cmd.exe

1 Threads, Time 283989, Result is 1000000000
2 Threads, Time 151488, Result is 50440770
4 Threads, Time 114538, Result is 26425132
8 Threads, Time 121798, Result is 28521652
16 Threads, Time 113896, Result is 25475290
Press any key to continue . . .
```



멀티쓰레드

- 왜 틀린 결과가 나왔을까?
 – “**sum = sum + 2**”가 문제이다.

쓰레드 1

쓰레드 2

MOV EAX, SUM

ADD EAX, 2

MOV SUM, EAX

MOV EAX, SUM

ADD EAX, 2

MOV SUM, EAX

sum = 200

sum = 200

sum = 202

sum = 202

멀티 쓰레드

- 틀리게 나오는 이유
 - Data Race 때문
- Data Race란?
 - 같은 메모리를 한 개 이상의 쓰레드가 동시에 읽고 쓰는 경우
 - 그 중 적어도 한 개는 반드시 쓰기 일것

멀티 쓰레드

• Data Race 해결

- Data Race 없애기 : 한번에 하나만 수행할 수 있도록 한다.
- Lock (L) : L을 다른 쓰레드에서 사용하고 있으면 대기, 없으면 사용 중 표시
- Unlock(L) : 사용 중 표시를 지움
- 용어 (mutual exclusion, critical section....)

쓰레드 1

쓰레드 2

```
Lock (LA)
A+=2;
Unlock (LA);
```

```
Lock (LA)
A+=2;
Unlock (LA);
```

결과는 항상

A = 4

멀티 쓰레드 프로그래밍

- Lock과 Unlock
 - C++11 표준에 존재
 - Mutex 클래스의 객체 생성 후 lock(), unlock() 메소드 호출

```
#include <mutex>

using namespace std;
mutex mylock;

...

mylock.lock();
// Critical Section
mylock.unlock();
```

멀티 쓰레드 프로그래밍

- Lock과 Unlock : 주의점
 - Mutex객체는 전역 변수로.
 - 같은 객체 사이에서만 Lock/Unlock이 동작.
 - 다른 mutex객체는 상대방을 모름.
 - 서로 동시에 실행돼도 괜찮은 critical section이 있다면 다른 mutex객체로 보호하는 것이 성능이 좋음
 - 같은 mutex객체로 보호하면 동시에 실행이 안됨

멀티 쓰레드 프로그래밍

- Why

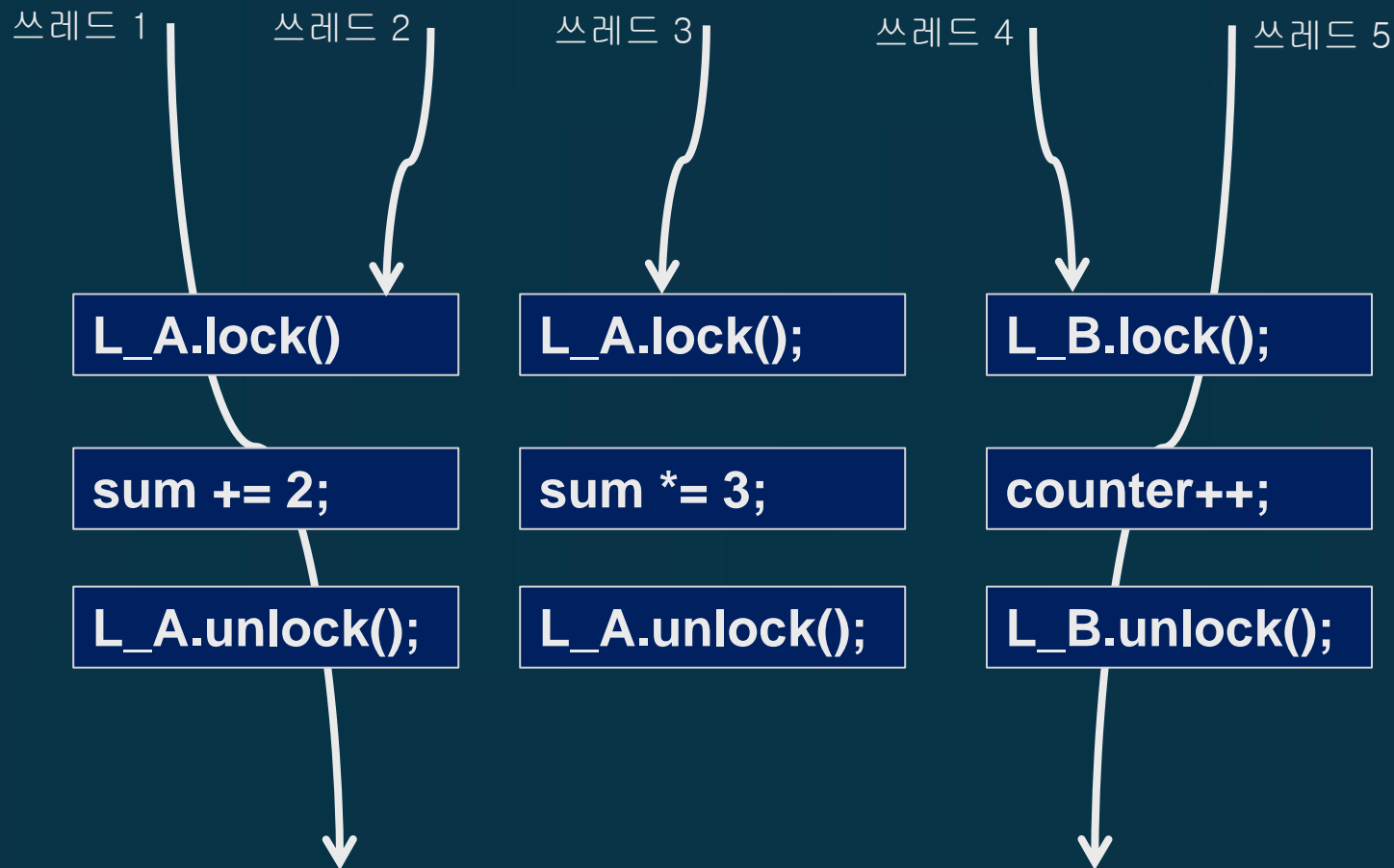
```
mylock.lock();
```

- Not

```
lock();
```

멀티 쓰레드 프로그래밍

- Mutex 객체가 필요한 이유.



멀티 쓰레드 프로그래밍 (실습)

- Thread 2개를 만드는 프로그램 작성
 - 전역변수 `sum`
 - 쓰레드가 하는 일은 “`sum += 2`” 오천만 **/ 2** 번 수행
 - 쓰레드 종료 후 `sum` 출력
 - **맞는 결과가 나오도록 프로그램 수정**

```
mutex mylock;  
  
mylock.lock();  
mylock.unlock();
```

멀티 쓰레드 프로그래밍 (실습)

- Lock을 사용한 멀티 쓰레드
 - 결과는 맞는가?
 - 수행 시간은?

멀티 쓰레드

- Lock 사용시 주의점

- Lock의 부하

- lock 호출 자체가 상당한 부하를 유발한다.
 - 호출한 코어 뿐만 아니라 다른 코어와 다른 CPU에도 Delay를 발생시킨다.

- Lock의 크기 (임계영역의 크기)

- 의미 : 하나의 Lock으로 보호 받는 변수의 크기 또는 프로그램의 길이
 - 너무 작다 : lock이 자주 호출되어 성능 저하가 심해진다.
 - 너무 크다 : lock을 얻지 못해 오랫동안 대기하는 쓰레드로 인한 성능 감소가 커진다. 병렬성이 떨어진다.

멀티 쓰레드 프로그래밍 (실습)

- Thread 여러개로 성능향상
 - 올바른 결과가 나와야 함.
 - Single Thread 프로그램보다 빨라야 함.
 - 멀티 쓰레드 프로그램은 싱글코어에서 Single Thread 프로그램보다 느릴 수 있음.
 - 방법 : **Lock**의 개수를 최소화 하고, 병렬 수행을 최대화 해야 함. => 알고리즘 수정 (재작성)

멀티 쓰레드 프로그래밍 (실습)

- 쿼드 코어(with Hyperthread)에서의 결과

```

void optimal_thread_func(int num_threads)
{
    volatile int local_sum = 0;
    for (auto i = 0; i < 50000000 / num_threads; ++i) local_sum += 2;
    mylock.lock();
    sum += local_sum;
    mylock.unlock();
}

int main()
{
    vector<thread*> threads;
    for (auto i = 1; i <= MAX_THREAD
    {
        sum = 0;
        threads.clear();
        auto start = high_resolution
        for (auto j = 0; j < i; ++j)
        for (auto tmp : threads) tmp
        auto duration = high_resolut
        cout << i << " Threads" << "
        cout << " Duration = " << d
    }

```

C:\Windows\system32\cmd.exe

```

1 Threads    Sum = 1000000000 Duration = 116 milliseconds
2 Threads    Sum = 1000000000 Duration = 58 milliseconds
4 Threads    Sum = 1000000000 Duration = 30 milliseconds
8 Threads    Sum = 1000000000 Duration = 32 milliseconds
16 Threads   Sum = 1000000000 Duration = 30 milliseconds
32 Threads   Sum = 1000000000 Duration = 28 milliseconds
64 Threads   Sum = 1000000000 Duration = 23 milliseconds

```

계속하려면 아무 키나 누르십시오 . . .

멀티 쓰레드 프로그래밍 (실습)

- 수행시간 비교
 - 싱글 쓰레드 수행시간
 - 멀티 쓰레드 수행시간
 - 멀티 쓰레드 + **lock**수행시간
 - (성능개선) 멀티 쓰레드 + **local** 계산 후 합산

정리

- 게임서버의 성능향상을 위해서는 멀티스레드 프로그래밍이 필수이다.
- 멀티 코어환경에서는 메모리 공유에 주의해야 한다. (Data Race)
- Lock은 성능저하를 초래한다.
- Lock을 최소한도로 사용하도록 프로그램을 재작성해야 한다.

Server의 요구사항

- Data Race의 위험성
- Mutex의 성능 저하
- 고성능 Atomic 자료 구조 필요
 - 고성능 : 상호배제 없이 동시 실행
 - Atomic : 호출 시 Data Race로 인한 오동작 없음
 - 자료구조 : int, bool 만이 아닌 Queue, Stack, Binary Tree, Hash Map등등.

다음시간

- 멀티 쓰레드 IOCP 실습