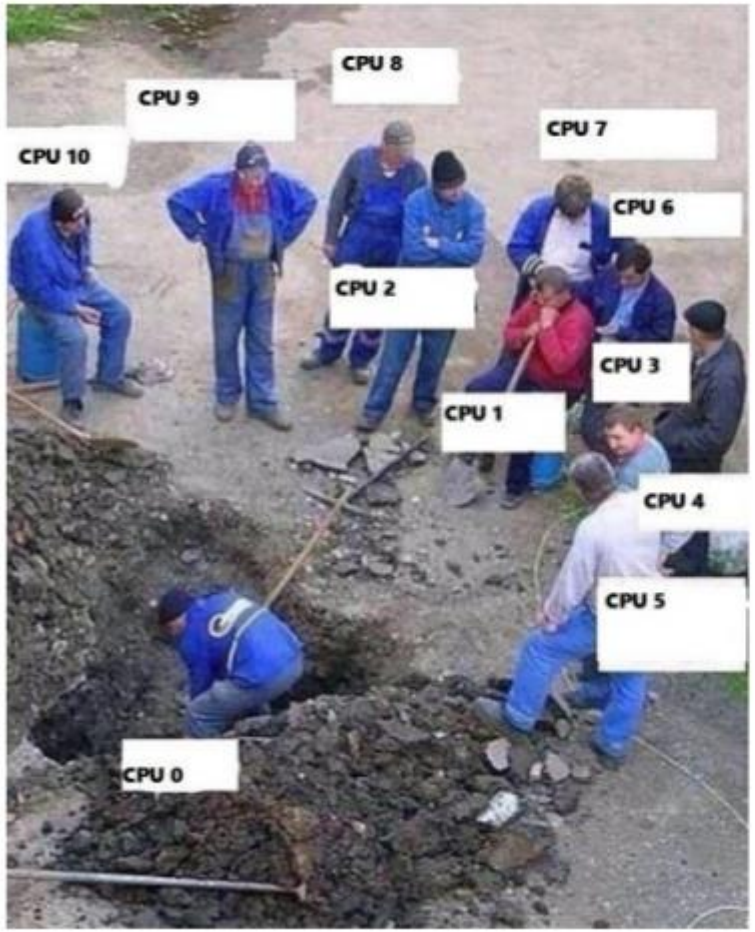
CHAPTER 12
Optimize Concurrency

Concurrency Refresher

동시성이란?

"컴퓨터 자원의 활용률을 증가시키기 위해 다수의 작업을 동시에 실행하는 것."



This is what typically happens in your computer

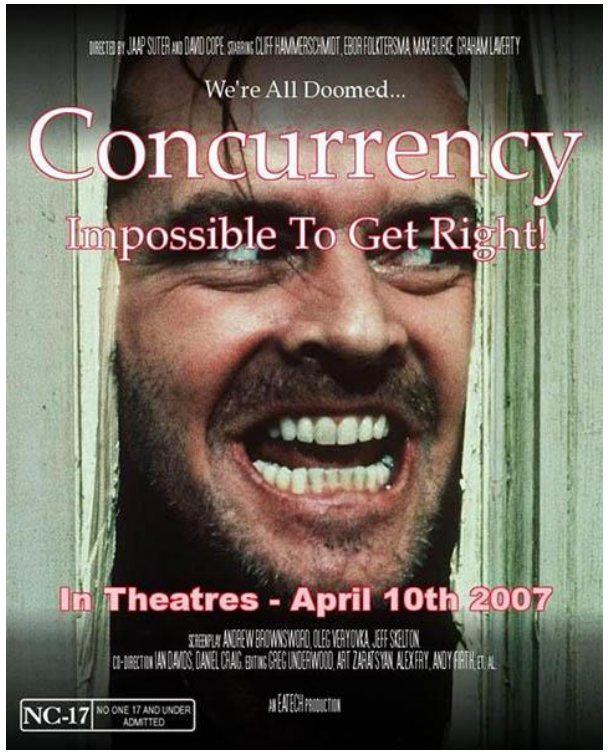
A Walk Through the Concurrency Zoo

Time slicing

Virtualization

Containerization

Symmetric multiprocessing



Simultaneous multithreading

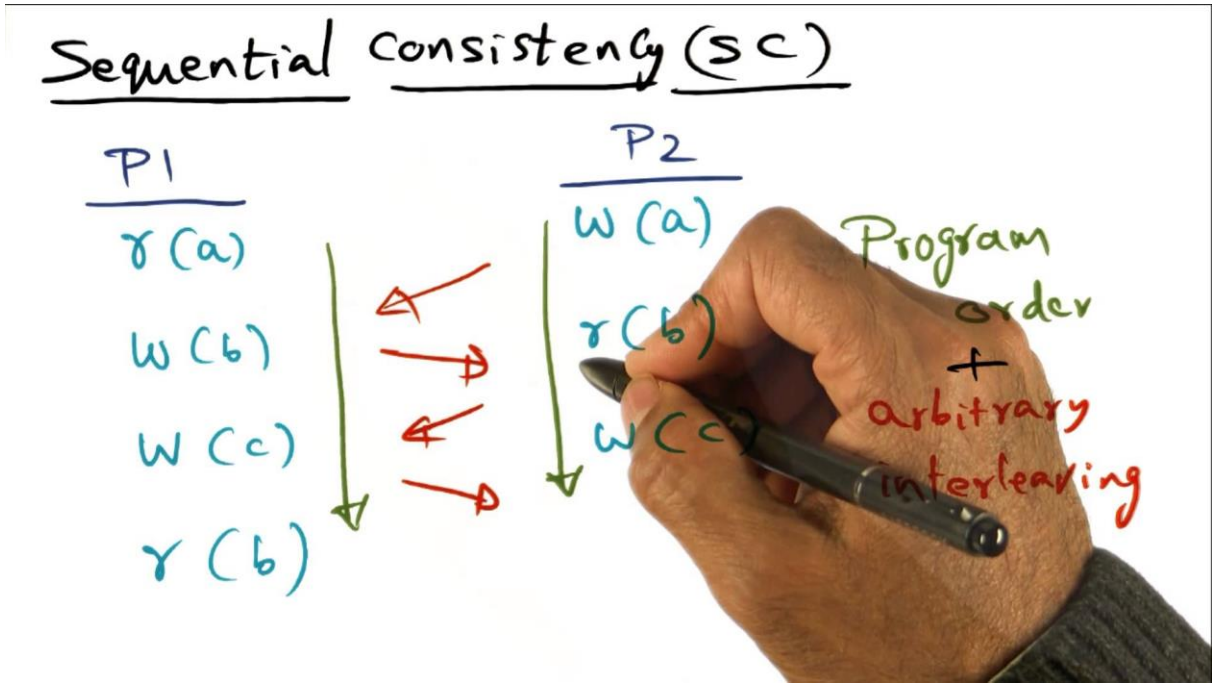
Multiple processes

Distributed processing

Threads

Tasks

Interleaved Execution, Sequential Consistency, Races



"다중 작업 환경에서는 순차적 일관성이 지켜지지 않을 수 있음."

Interleaved Execution, Sequential Consistency, Races

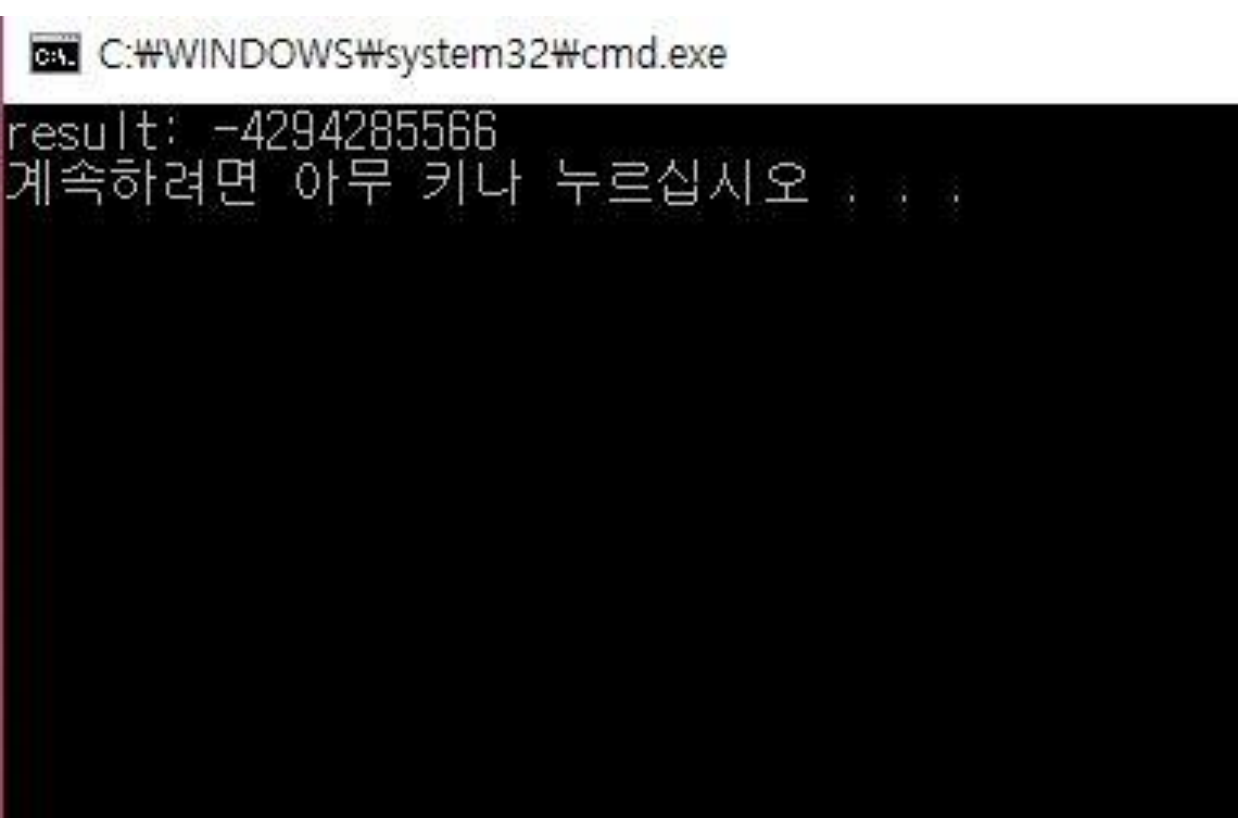
```
HANDLE  ThreadHandle[50];

for ( int i = 0; i < 50; ++i )
{
    if ( i % 2 )
        ThreadHandle[i] = (HANDLE)_beginthreadex(nullptr, 0, ThreadFunctionInc, nullptr, 0, nullptr);
    else
        ThreadHandle[i] = (HANDLE)_beginthreadex(nullptr, 0, ThreadFunctionDes, nullptr, 0, nullptr);
}

WaitForMultipleObjects( 50, ThreadHandle, TRUE, INFINITE );

printf("result: %lld\n", Count);
```

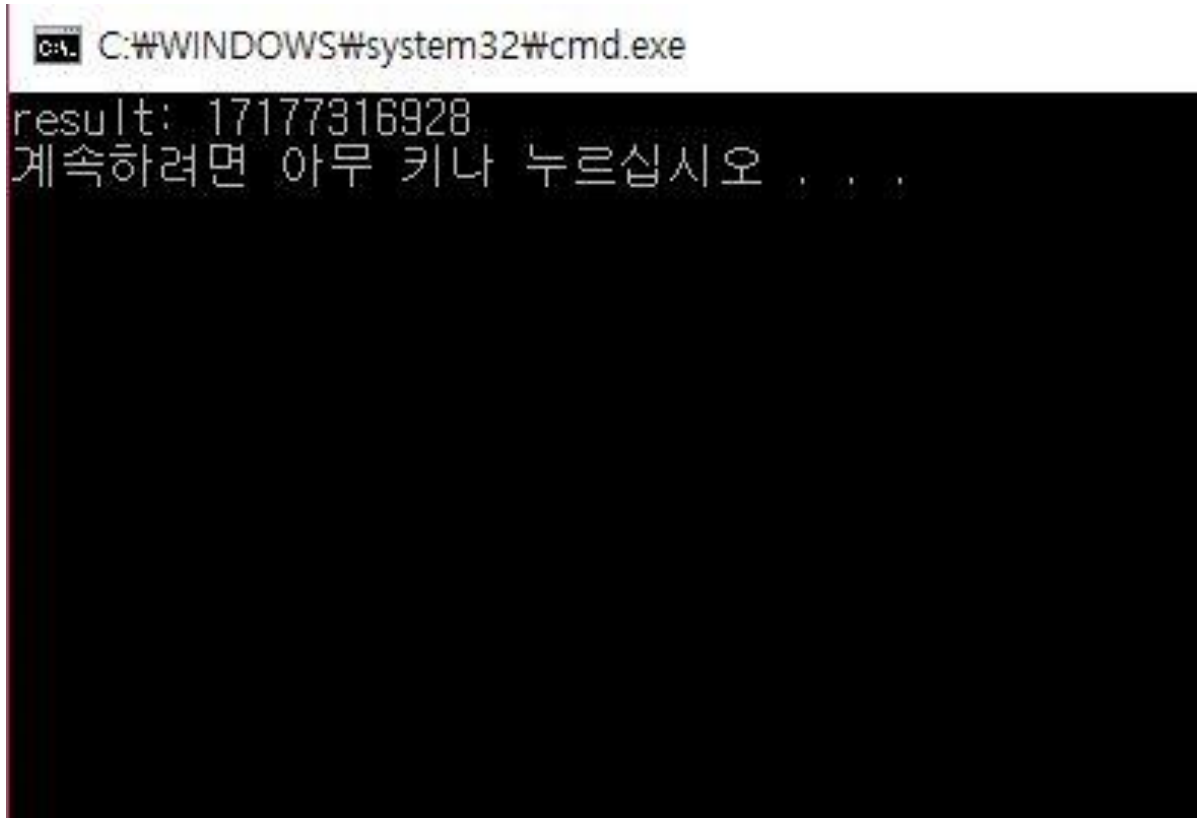
WIN32API를 이용한 스레드 생성

Interleaved Execution, Sequential Consistency, Races

```
// 공유 자원
long long Count = 0;

unsigned    WINAPI  ThreadFunctionInc(void* arg)
{
    for (int i = 0; i < 5000000; ++i)
    {
        ++Count;
    }

    return 0;
}

unsigned    WINAPI  ThreadFunctionDes(void* arg)
{
    for (int i = 0; i < 5000000; ++i)
    {
        --Count;
    }

    return 0;
}
```

Interleaved Execution, Sequential Consistency, Races





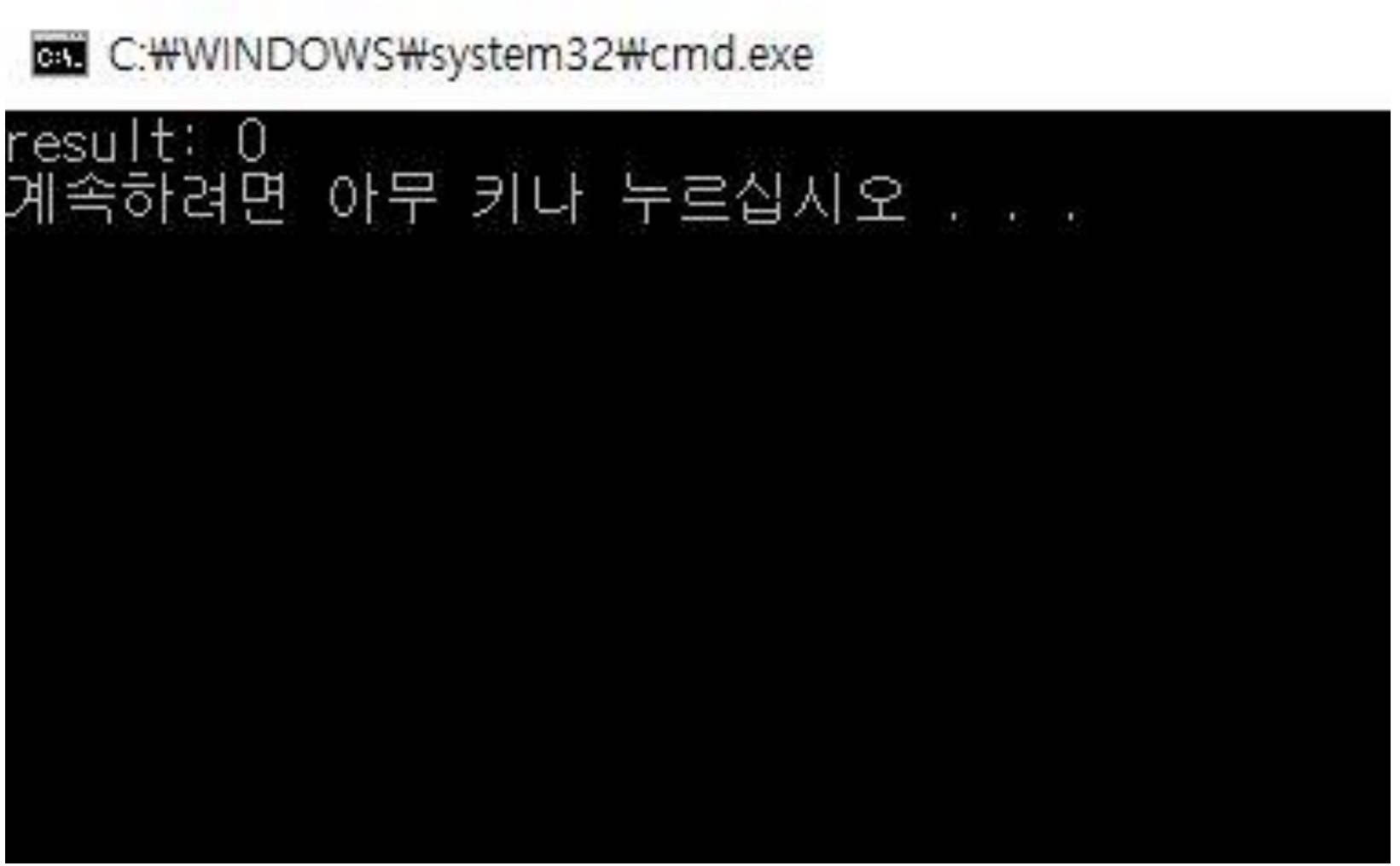
결과는 실행마다 달라진다 -> 순차적 일관성의 실패

Synchronization

```
CRITICAL_SECTION    cs;

unsigned    WINAPI  ThreadFunctionInc(void* arg)
{
    EnterCriticalSection(&cs)
    for (int i = 0; i < 5000000; ++i)
    {
        ++Count;
    }
    LeaveCriticalSection(&cs)

    return 0;
}

unsigned    WINAPI  ThreadFunctionDes(void* arg)
{
    EnterCriticalSection(&cs)
    for (int i = 0; i < 5000000; ++i)
    {
        --Count;
    }
    LeaveCriticalSection(&cs);

    return 0;
}
```

```
HANDLE  ThreadHandle[50];
InitializeCriticalSection(&cs);

for ( int i = 0; i < 50; ++i )
{
    if ( i % 2 )
        ThreadHandle[i] = (HANDLE)_beginthr
    else
        ThreadHandle[i] = (HANDLE)_beginthr
}

WaitForMultipleObjects( 50, ThreadHandle,

printf("result: %lld\n", Count);
DeleteCriticalSection(&cs);
```

Synchronization



Atomicity

```
unsigned    WINAPI  ThreadFunctionInc(void* arg)
{
    for (int i = 0; i < 5000000; ++i)
    {
        InterlockedIncrement(&Count);
    }

    return 0;
}

unsigned    WINAPI  ThreadFunctionDes(void* arg)
{
    for (int i = 0; i < 5000000; ++i)
    {
        InterlockedDecrement(&Count);
    }

    return 0;
}
```

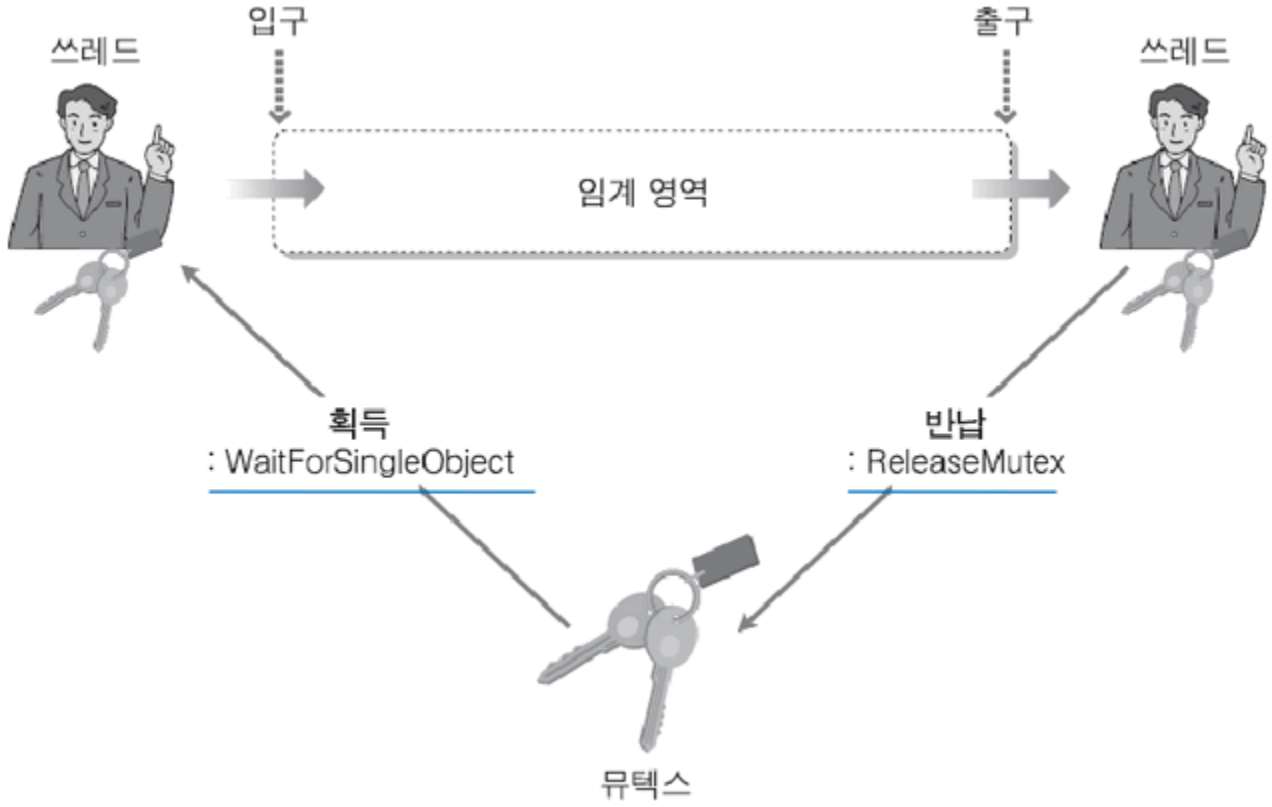
더 이상 쪼개질 수 없는 성질. 각각의 명령어 단위는 실행하는 도중에 중단될 수 없음

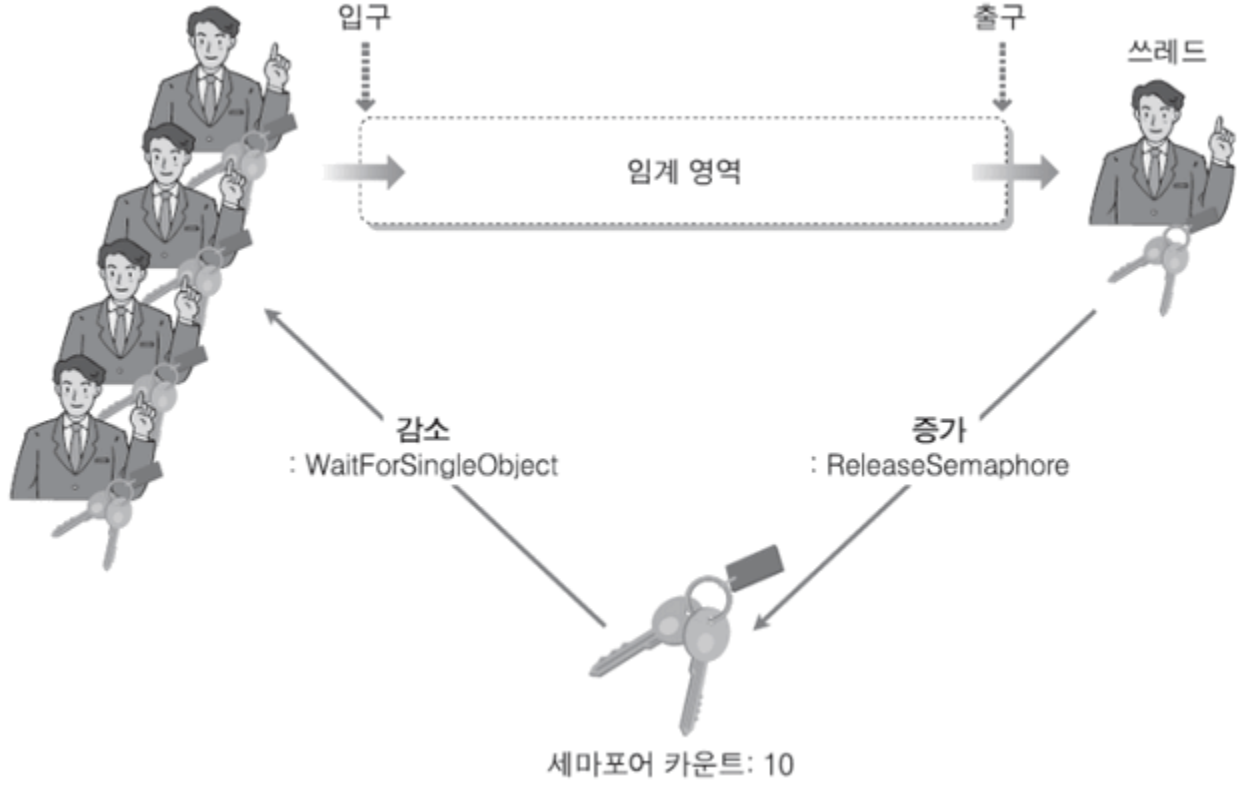
Synchronization(WIN32)

USER MODE	KERNEL MODE
CRITICAL_SECTION Interlocked Function	Mutex Semaphore

애플리케이션 코드가 실행되는 유저모드 / 운영체제 코드가 실행되는 커널모드

Synchronization(WIN32)





출처 – 뇌자극 시스템 프로그래밍(윤성우 저)

std::thread

```
HANDLE  ThreadHandle[50];

for ( int i = 0; i < 50; ++i )
{
    if ( i % 2 )
        ThreadHandle[i] = (HANDLE)_beginthreadex(nullptr, 0, ThreadFunctionInc, nullptr, 0, nullptr);
    else
        ThreadHandle[i] = (HANDLE)_beginthreadex(nullptr, 0, ThreadFunctionDes, nullptr, 0, nullptr);
}

WaitForMultipleObjects( 50, ThreadHandle, TRUE, INFINITE );

printf("result: %lld\n", Count);
```

운영체제(WIN32API)에 종속적인 스레드 사용

std::thread

```
std::thread Th[50];
InitializeCriticalSection(&cs);

for ( int i = 0; i < 50; ++i )
{
    if (i % 2)
        Th[i] = std::thread(ThreadFunctionInc, nullptr);
    else
        Th[i] = std::thread(ThreadFunctionDes, nullptr);
}

for (int i = 0; i < 50; ++i)
    Th[i].join();

printf("result: %lld\n", Count);
DeleteCriticalSection(&cs);
```

운영체제의 한계를 넘어 언어 차원에서의 스레드 사용

std::thread

Example 12-3 shows a simple example of `std::thread` use.

Example 12-3. Starting a few simple threads

```
void f1(int n) {
    std::cout << "thread " << n << std::endl;
 }

void thread_example() {
    std::thread t1;           // thread variable, not a thread
    t1 = std::thread(f1, 1); // assign thread to a thread variable
    t1.join();               // wait for thread to complete

    std::thread t2(f1, 2);
    std::thread t3(std::move(t2));
    std::thread t4([]() { return; });// works with lambdas too
    t4.detach();
    t3.join();
}
```

std::promise, std::future

Example 12-4. Promises, futures, and threads

```
void promise_future_example() {
    auto meaning = [](std::promise<int>& prom) {
        prom.set_value(42); // compute the meaning of life
    };

    std::promise<int> prom;
    std::thread(meaning, std::ref(prom)).detach();

    std::future<int> result = prom.get_future();
    std::cout << "the meaning of life: " << result.get() << "\n";
}
```

std::packaged_task, std::async

Example 12-5 is a simplified version of Example 12-4 that uses a packaged_task.

Example 12-5. packaged_task and thread

```
void promise_future_example_2() {
    auto meaning = std::packaged_task<int(int)>(
                        [](int n) { return n; });
    auto result  = meaning.get_future();
    auto t       = std::thread(std::move(meaning), 42);

    std::cout << "the meaning of life: " << result.get() << "\n";
    t.join();
}
```

std::packaged_task, std::async

Example 12-6. tasks and async()

```
void promise_future_example_3() {
    auto meaning = [](int n) { return n; };
    auto result = std::async(std::move(meaning), 42);
    std::cout << "the meaning of life: " << result.get() << "\n";
}
```