



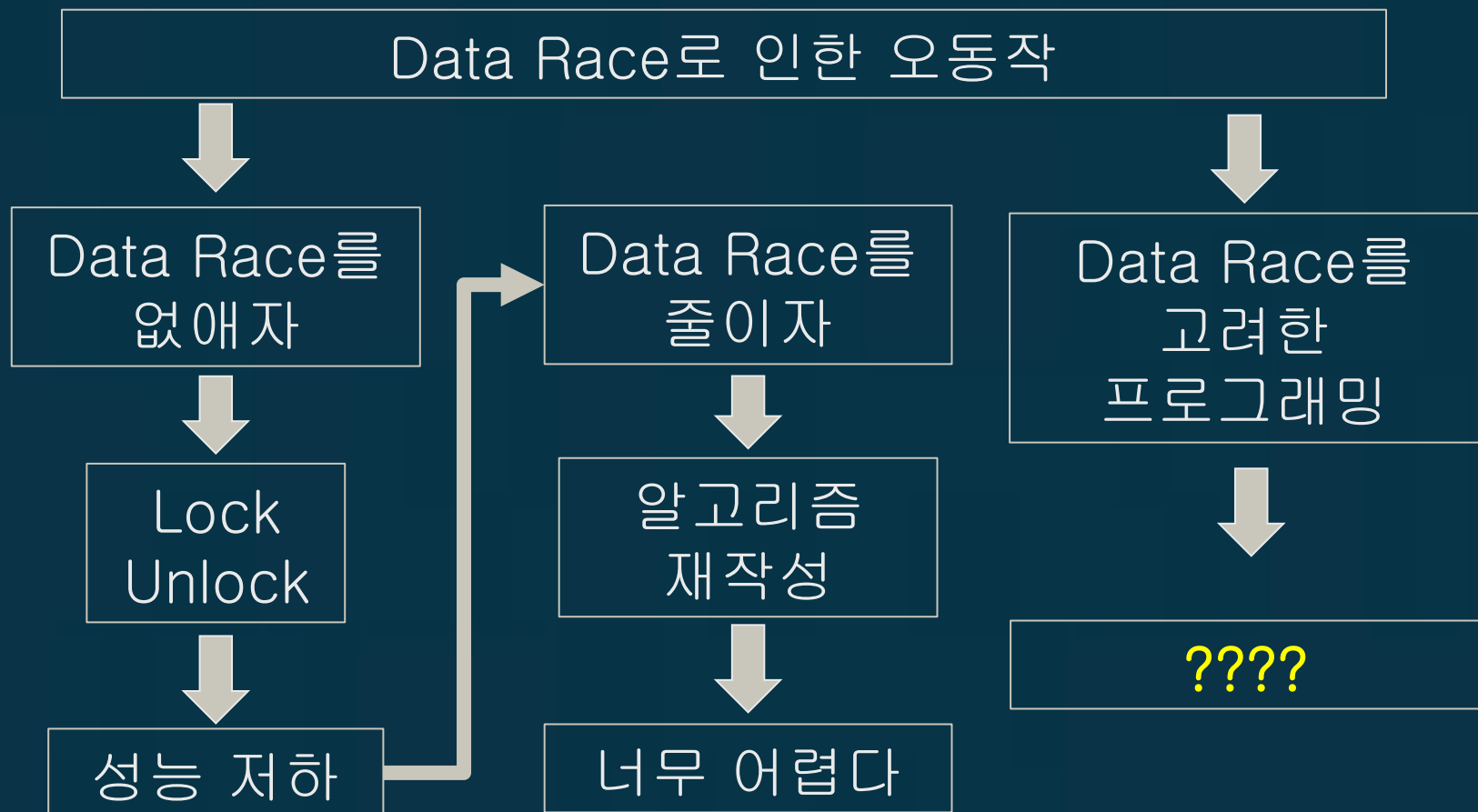
2. 프로그래밍

멀티쓰레드 프로그래밍

정내훈

정리

- 멀티쓰레드 프로그래밍



목차

- Data Race 길들이기??
 - 컴파일러
 - CPU
- Non-Blocking 프로그래밍
- 이론 시간 + CAS

Data Race 길들이기

- 질문 : Data Race가 있는 줄 알고 있고, Data Race때문에 어떤 실행결과가 나올지 알고 있으면 문제를 피할 수 있도록 프로그래밍 할 수 있지 않을까?
 - Data Race라 해도 전혀 엉뚱한 값이 나오지 않고, 나올 수 있는 값들이 예측 가능하다.
 - Lock을 추가해서 Data Race를 없애는 일을 하지 않으니 성능도 좋지 않을까?
- 해보자.

컴파일러

- 다음 멀티쓰레드 프로그램에 어떠한 문제점이 있을까?

```
bool g_ready = false;
int g_data = 0;

void Receiver()
{
    while (false == g_ready);
    std::cout << "I got " << g_data << std::endl;
}
```

```
void Sender()
{
    cin >> g_data;
    g_ready = true;
}
```

컴파일러

● 실습 #11

- 실행 후 결과 값을 확인해 보시오.
- <주의> 10번 정도 실행해 보시오.

```
bool g_ready = false;
int g_data = 0;

void Receiver()
{
    while (false == g_ready);
    std::cout << "I got " << g_data << std::endl;
}
```

```
void Sender()
{
    cin >> g_data;
    g_ready = true;
}
```

컴파일러

● 컴파일 결과

```
void Receiver()
{
    while (false == g_ready);
    std::cout << "I got " << g_data << std::endl;
}
```



```
--- C:\Wdepot\WProjects\W\Wlecture\WMP\Wsync_test\Wsync_test\Wsync.cpp -----
while (false == g_ready);
std::cout << "I got " << g_data << std::endl;
00631010 mov     ecx,dword ptr [__imp_std::cout (0633068h)]
00631016 push    offset std::endl<char,std::char_traits<char> > (06314A0h)
0063101B push    dword ptr [g_data (06353F8h)]
00631021 call     std::operator<<<std::char_traits<char> > (0631280h)
00631026 mov     ecx,eax
00631028 call     dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (0633038h)]
0063102E mov     ecx,eax
00631030 call     dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (0633034h)]
}
00631036 ret
```

컴파일러

● Visual Studio의 사기?



★ Pinned



Microsoft Resolution - [Karen Huang \[MSFT\]](#)

Closed - Not a Bug

...

After our investigation, this is not a bug. It's by design. Please see comment by Gratian Lup.

OK, good to know 0 ↑

Reconsider 0 ↓

Show reactions 0

Jun 10, 2021

View timeline by [All Posts \(8\)](#) [Solutions & workarounds \(2\)](#)



정내훈

New

Apr 15, 2021



컴파일러

- Release 최적화 오류
 - 불안정한 결과
 - 원인 : while loop 자체를 없애 버림
 - 버그 레포트 결과 : “not a bug, 다른 컴파일러도 마찬가지 ㅋㅋㅋ.”

컴파일러

- 다음 멀티쓰레드 프로그램에 어떠한 문제점이 있을까?

```
void Receiver()  
{  
    int count = 0;  
    while (false == g_ready) count++;  
    std::cout << "I got " << g_data << std::endl;  
    std::cout << "count  = " << count << std::endl;  
}
```

```
void Sender()  
{  
    cin >> g_data;  
    g_ready = true;  
}
```

컴파일러

● 컴파일 결과

```
void Receiver()
{
    int count = 0;
    while (false == g_ready) count++;
    std::cout << "I got " << g_data << std::endl;
    std::cout << "count  = " << count << std::endl;
}
```



00007FF6D54D1020	48 83 EC 28	sub	rsp,28h
int count = 0;			
while (false == g_ready) {			
00007FF6D54D1024	80 3D E5 46 00 00 00	cmp	byte ptr [g_ready (07FF6D54D5710h)],0
00007FF6D54D102B	75 05	jne	Receiver+12h (07FF6D54D1032h)
00007FF6D54D102D	0F 1F 00	nop	dword ptr [rax]
→ 00007FF6D54D1030	EB FE	jmp	Receiver+10h (07FF6D54D1030h)
count++;			
}			
std::cout << "I got " << g_data << std::endl;			

컴파일러

● Release 모드 최적화 버그???

— 무한 루프

— 원인 : g_ready의 변경값을 읽지 않음

— 버그??? => 아니다

- MS에 소송을 걸면 우리가 짐
- C 언어 자체가 멀티쓰레드 언어가 아님
- Release는 single thread에 맞춰서 만 최적화 해도 문제 없음.
- 멀티쓰레드에서 오동작은 다 프로그래머 책임
 - MS가 멀티쓰레드하라고 협박한 적 없음
- C++11에서 <thread> 관련 도우미들이 있다고 착각하면 안됨

컴파일러

- 컴파일러의 사기를 피하는 방법
 - lock/unlock을 사용한다.
 - volatile을 사용하면 된다.
 - 반드시 메모리를 읽고 쓴다.
 - 변수를 레지스터에 할당하지 않는다.
 - 읽고 쓰는 순서를 지킨다.
 - 참 쉽죠?
 - “어셈블리를 모르면 Visual Studio의 사기를 알 수 없다” ...

컴파일러

- 실습 #12

- volatile을 추가한 후 실행하라. 결과는?

```
volatile bool g_ready = false;
volatile int g_data = 0;

void Receiver()
{
    while (false == g_ready);
    std::cout << "I got " << g_data << std::endl;
}
```

```
void Sender()
{
    cin >> g_data;
    g_ready = true;
}
```

고생길

● 정말 쉬운가???

```
struct Qnode {  
    volatile int data;  
    volatile Qnode* next;  
};  
  
void ThreadFunc1()  
{  
    ...  
    while ( qnode->next == NULL ) { }  
    my_data = qnode->next->data;  
    ...  
}
```

무엇이 문제일까??

고생길

● volatile의 사용법

—volatile int * a;

- *a = 1; // volatile 적용, 프로그램 그대로 컴파일
- a = b; // 컴파일러의 최적화 대상

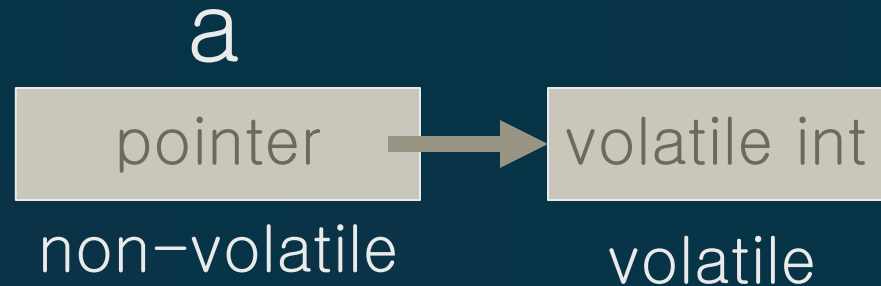
—int * volatile a;

- *a = 1; // 컴파일러의 최적화 대상
- a = b; // volatile 적용, 프로그램 그대로 컴파일

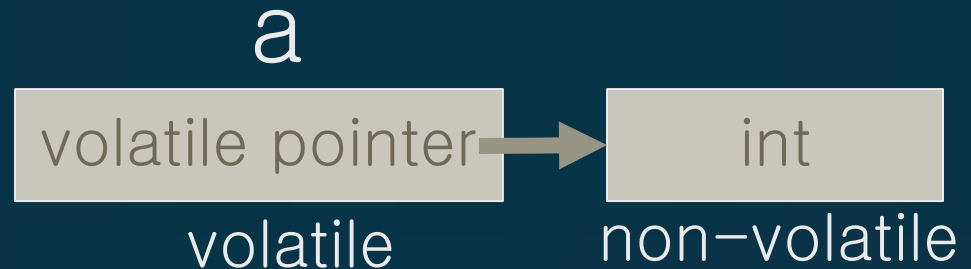
고생길

- volatile의 사용법

- volatile int * a;



- int * volatile a;



컴파일러

● Volatile 위치 오류의 예

`volatile Qnode* next;`



`Qnode * volatile next;`

```
void UnLock() {
    Qnode *qnode;
    qnode = myNode;
    if (qnode->next == NULL) {
        LONG long_qnode = reinterpret_cast<LONG>(qnode);
        volatile LONG *long_tail = reinterpret_cast<volatile LONG*>(&tail);
        if ( CAS(long_tail, NULL, long_qnode) ) return;
        while ( qnode->next == NULL ) { }
    }
    qnode->next->locked = false;
    qnode->next = NULL;
}
```

```
011F1089  mov     eax,dword ptr [esi+4]
011F108C  lea     esp,[esp]
011F1090  cmp     eax,ebx
011F1092  je      ThreadFunc+90h (11F1090h)
```

```
01191090  mov     eax,dword ptr [esi+4]
01191093  test    eax,eax
01191095  je      ThreadFunc+90h (1191090h)
```

컴파일러

● 정리

- 여러 개의 스레드가 공유하는 변수는 volatile을 사용 해야 한다.
- volatile을 사용하면 컴파일러는 프로그래머가 지시한 대로 메모리에 접근한다.
 - Volatile이 없으면 컴파일러는 싱글쓰레드를 기준으로 프로그램을 최적화 한다
- (지금은 SKIP) 하지만, CPU는 volatile을 모른다...

목차

- 병렬 프로그램 작성시 주의점
 - 컴파일러
 - CPU
 - 상호배제의 구현
 - 메모리 일관성 문제
- Non-Blocking 프로그래밍
- 이론 시간 + CAS

상호배제

- 멀티 스레드 프로그램에서의 문제는 하나의 자원을 여러 스레드에서 동시에 사용해서 생기는 경우가 대부분
- 해결책
 - 공유 자원을 업데이트 하는 부분은 한번에 하나의 스레드에서만 실행할 수 있도록 하자
 - 이것을 상호배제(mutual exclusion)라 부른다.

상호배제

- 임계영역
 - Critical Section
 - 프로그램중 상호배제로 보호받고 있는 구간
 - 오직 하나의 스레드만 실행할 수 있음.
- 임계영역 구현
 - Lock & Unlock을 사용해서 Lock과 Unlock사이에 임계영역을 둔다.
 - Lock은 다른 스레드가 Lock을 통과했고 Unlock을 하기 전이라면 Unlock을 실행할 때 까지 프로그램의 실행을 멈춘다.

상호배제

- 실제로 구현해 보자
 - C++11의 라이브러리를 사용하지 않고.
 - 최대한 단순하고 가볍게
- Lock & Unlock의 구현
 - 공유 메모리를 통해서 구현한다.
 - 여러 가지 알고리즘이 있다.
 - 피터슨, 데커, 빵집...

상호배제

- 피터슨 알고리즘

- 2개의 쓰레드사이의 Lock과 Unlock을 구현하는 알고리즘
- 매개 변수로 쓰레드 아이디를 전달 받으며 값은 0과 1이라고 가정
- 운영체제 시간에 배움, 유명한 알고리즘

```
volatile int victim = 0;
volatile bool flag[2] = {false, false};

Lock(int myID)
{
    int other = 1 - myID;
    flag[myID] = true;
    victim = myID;
    while(flag[other] && victim == myID) {}
}

Unlock (int myID)
{
    flag[myID] = false;
}
```

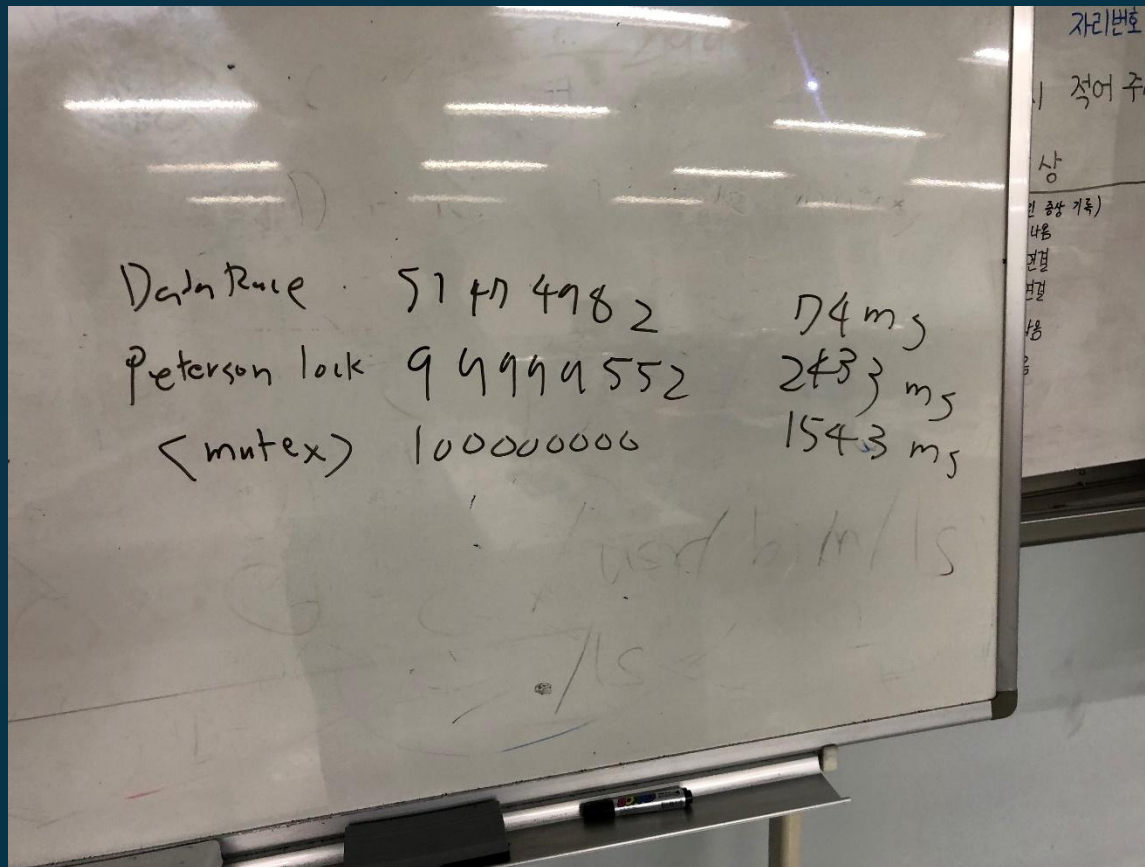

상호배제

● 실습 #13

- 앞 페이지의 피터슨 알고리즘을 사용하여 2개의 쓰레드로 1억 만들기를 구현하라.
- 실습 #8의 with lock 버전과 속도 비교를 하라.
- lock()이 없는 버전과 속도비교를 하라.

상호배제

● 실습 #13



상호배제

```
#include <iostream>
#include <thread>
using namespace std;

volatile int victim = 0;
volatile bool flag[2] = { false, false };
volatile int sum;

void Lock(int myID)
{
    int other = 1 - myID;
    flag[myID] = true;
    victim = myID;
    while (flag[other] && victim == myID) {}
}

void Unlock(int myID)
{
    flag[myID] = false;
}
```

```
void thread_func(int thid)
{
    for (auto i = 0; i < 250000000; ++i) {
        Lock(thid);
        sum = sum + 2;
        Unlock(thid);
    }
}

int main()
{
    thread t1 = thread{ thread_func, 0 };
    thread t2 = thread{ thread_func, 1 };
    t1.join();
    t2.join();
    cout << "Sum = " << sum << "Wn";
}
```

상호배제

- 피터슨 알고리즘의 문제
 - 빈번한 메모리 참조로 인한 성능 문제
 - 실제 컴퓨터에서 오동작을 일으킴
 - 원인은 ?? => 다음장에서
- 해결책?
 - C++11의 `mutex` 라이브러리를 사용
 - 여러 가지 편리한 기능
 - 오버헤드로 인한 성능문제

상호배제

- N개의 쓰레드에서의 동기화 구현
 - 빵집 알고리즘 : 교재 2.6 절

```
1  class Bakery implements Lock {
2      boolean[] flag;
3      Label[] label;
4      public Bakery (int n) {
5          flag = new boolean[n];
6          label = new Label[n];
7          for (int i = 0; i < n; i++) {
8              flag[i] = false; label[i] = 0;
9          }
10     }
11     public void lock() {
12         int i = ThreadID.get();
13         flag[i] = true;
14         label[i] = max(label[0], ..., label[n-1]) + 1;
15         while (( $\exists k \neq i$ )(flag[k] && (label[k],k) << (label[i],i))) {};
16     }
17     public void unlock() {
18         flag[ThreadID.get()] = false;
19     }
20 }
```

Figure 2.9 The Bakery lock algorithm.

상호배제

- 숙제 #1 :
 - 앞 페이지의 빵집 알고리즘을 C++로 구현하시오
 - 다른 곳에 있는 빵집 알고리즘을 사용하지 마시오!
 - Volatile을 사용해서 공유메모리 컴파일 최적화를 막으시오
 - 벤치마크 프로그램으로 천만 만들기 프로그램을 사용하시오.
 - 실습 #7의 프로그램을 사용
 - 피터슨 알고리즘과 마찬가지로 천만이 나오지 않을 수 있음.
 - 스레드 1, 2, 4, 8개 일 때의 실행 시간을 측정하시오.
 - Release모드에서 실행
 - 아무것도 사용하지 않을 때/ <mutex>사용할 때/빵집 알고리즘을 사용할 때 결과 값과 속도를 비교 하시오.
 - Volatile을 atomic으로 변경 후 결과값과 속도를 비교 하시오
 - 제출물
 - .cpp 파일
 - 실행속도 비교표 (no Lock, mutex 사용, 빵집 알고리즘(volatile, atomic))
 - CPU의 종류 (모델명, 코어 개수, 클럭)

정리

- 멀티쓰레드에서 프로그램의 이상 동작
 - <mutex>를 사용하지 않을 경우
 - C 언어 자체의 문제
 - 컴파일러의 최적화도 문제 있음
 - volatile로 해결
- <mutex>를 사용할 경우의 성능 저하
 - Custom하게 만든 simple한 lock도 성능상의 문제가 있음.
 - 결국은 멀티쓰레드 프로그래밍을 포기해야???