



2. 프로그래밍

멀티쓰레드 프로그래밍

정내훈

목차

- 멀티쓰레드 프로그램 작성시 주의점
 - 컴파일러
 - CPU
- Non-Blocking 프로그래밍
- 이론 시간 + CAS

컴파일러

- 다음의 멀티쓰레드 프로그램에 어떠한 문제점이 있을까?

```
bool flag = false;
int data;

void ThreadFunc1()
{
    while(flag == false);
    int my_data = data;
}
```

```
void ThreadFunc2()
{
    data = 1;
    flag = true;
}
```

컴파일러

● 실습 #11

- 2개의 스레드를 실행 한 후 my_data의 값을 출력하라.


```
void ThreadFunc1()  
{  
    while(flag == false);  
    my_data = data;  
}
```

```
void ThreadFunc2()  
{  
    data = 1;  
    flag = true;  
}
```

컴파일러

● Visual Studio의 사기

```
void ThreadFunc1()  
{  
    while(flag == false);  
    my_data = data;  
}
```



```
void ThreadFunc1()  
{  
000612B0  mov     al,byte ptr ds:[00064494h]  
    while (flag == false);  
000612B5  test    al,al  
000612B7  je      thread_func1+5h (0612B5h)  
    int my_data = data;  
}  
000612B9  ret
```

싱글 쓰레드 프로그램이면?
VS는 무죄!

컴파일러

- 컴파일러의 사기를 피하는 방법
 - 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

- 2개의 스레드를 실행 한 후 my_data의 값을 출력하라.
- 무한 루프에 빠지지 않도록 변경 후 실행하라.

```
void ThreadFunc1()  
{  
    while(!flag);  
    my_data = data;  
}
```

```
void ThreadFunc2()  
{  
    data = 1;  
    flag = 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; // 순서를 지킴
- a = b; // 순서를 지키지 않는다.

—int * volatile a;

- *a = 1; // 순서를 지키지 않음,
- a = b; // 이것은 순서를 지킴

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.
- (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
 - 운영체제의 API를 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

- 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 :

- 빵집 알고리즘을 구현하시오
- 벤치마크 프로그램으로 1억 만들기 프로그램을 사용하시오.
 - 실습 #7의 프로그램을 사용
- 스레드 1, 2, 4, 8, 16개 일 때의 실행 시간을 구하시오.
- No Lock, With Lock 버전과 속도 비교를 하시오
- 제출물
 - .cpp 파일
 - 실행속도 비교표 (no Lock, mutex 사용, 빵집 알고리즘)
 - CPU의 종류 (모델명, 코어 개수, 클럭)
 - 실행시간이 30분 이상 걸리거나 컴퓨터가 버벅거리면 속도 측정 생략 가능
 - 이러한 이상 현상이 생기는 원인에 대한 예측
- 제출 : nhjung@kpu.ac.kr
 - 제목 : [2018 전공특강 숙제 1] 학번, 이름
 - 3월 27일 화요일 오전 10시까지 제출

The screenshot shows a Windows Task Manager window with the 'Performance' tab selected. The CPU usage is at 100% at 2.38 GHz. A list of processes is visible on the left, including 'C11_Compiler', 'C11_Hello', 'C11_LF_LIST', 'C11_LF_Stack', 'C11Test1', 'C11Test3', and 'ch10'. In the background, a command prompt window titled 'C:\Windows\system32\cmd.exe' displays the output of a program, showing statistics for 1, 2, 4, 8, 16, and 32 threads. The statistics include 'Sum' and 'Duration' in milliseconds.

Task Manager Performance:

- CPU:** 100% 2.38 GHz
- Memory:** 12/256 GB (5%)
- Ethernet:** Not connected
- Ethernet:** S: 8.0 Kbps R: 0 Kbps

Process List:

- C11_Compiler
- C11_Hello
- C11_LF_LIST
- C11_LF_Stack
- C11Test1
- C11Test3
- ch10

Command Prompt Output:

```

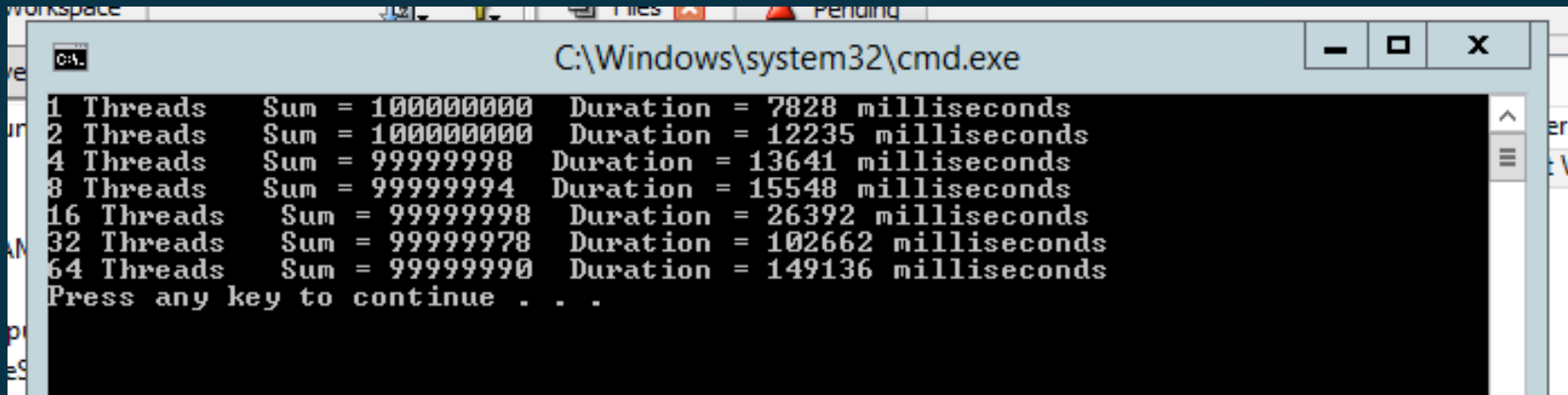
1 Threads Sum = 100000000 Duration = 7828 milliseconds
2 Threads Sum = 100000000 Duration = 12235 milliseconds
4 Threads Sum = 99999998 Duration = 13641 milliseconds
8 Threads Sum = 99999994 Duration = 15548 milliseconds
16 Threads Sum = 99999998 Duration = 26392 milliseconds
32 Threads Sum = 99999978 Duration = 102662 milliseconds
  
```

Task Manager Details:

- CPU:** Intel(R) Xeon(R) CPU E5-4620 0 @ 2.20GHz
- Logical processors:** 100% (all 10 processors are at 100% utilization)
- Utilization:** 100%
- Speed:** 2.38 GHz
- Maximum speed:** 2.20 GHz
- Sockets:** 4
- Cores:** 32
- Logical processors:** 64
- Virtualization:** Enabled
- L1 cache:** 2.0 MB
- L2 cache:** 8.0 MB
- L3 cache:** 64.0 MB
- Processes:** 47
- Threads:** 1135
- Handles:** 17638
- Up time:** 0:00:31:43

상호배제

- 뺑집알고리즘 실행 결과



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the execution results of the 뺑집알고리즘 (Bakjip algorithm) for different thread counts. The results are as follows:

Threads	Sum	Duration (milliseconds)
1	1000000000	7828
2	1000000000	12235
4	99999998	13641
8	99999994	15548
16	99999998	26392
32	99999978	102662
64	99999990	149136

Press any key to continue . . .

정리

- 멀티쓰레드에서 프로그램의 이상 동작
 - C 언어 자체의 문제
 - 컴파일러의 문제가 아님
 - volatile로 해결
- Volatile을 사용함에도 문제가 있음.
 - 어셈블리로 확인해 봐도 문제 없음
- Lock을 사용할 경우의 성능 저하
 - Custom하게 만든 simple한 lock도 성능상의 문제가 있음.
 - 결국은 상호배제를 포기해야???