운영체제 과제 1: 상호배제

컴퓨터과학부 2018920031 유승리 | 운영체제 과제 1 | 2020-04-09-목 제출

** 운영체제 과제 1: 상호배제

- 학번과 이름: 2018920031 유승리
- 제출일: 2020-04-09 목

#1 i와 j로 이루어진 루프의 횟수를 변화시키면서 shared_var의 출력값이 어떻게 되는지를 반복해서 관찰하고 그 결과를 레포트에 기술하라. 그리고 그 이유를 설명하여 레포트에 기술하라.

〉 실험 결과

[예 1-1] i〈100, j〈100

- 예상 출력: 2*100*100 = 20000
- 평균 출력: 20000.000000
- 오차율: 0%

1회	2회	3회
Main Thread completed Thread_1 completed 20000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 20000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 20000.000000 (컴퓨터과학부 2018920031 유승리)
20000.000000	20000.000000	20000.000000

[예 1-2] i〈1000, j〈1000

- 예상 출력: 2*1000*1000 = 2000000
- 평균 출력: 1188230. 000000
- 오차율: 40.589%

1회	2회	3회
Main Thread completed Thread_1 completed 1266027.000000 (컴퓨터과학부 2018920031 유승리)	Thread_1 completed Main Thread completed 1100680.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 1197983.000000 (컴퓨터과학부 2018920031 유승리)
1266027.000000	1100680.000000	1197983.000000

[예 1-3] i < 10000, j < 100000

- 예상 출력: 2*10000*100000 = 2000000000

- 평균 출력: 1025470245.666667

- 오차율: 48.726%

1회	2회	3회
Thread_1 completed Main Thread completed 1076399156.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 997901390.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 1002110191.000000 (컴퓨터과학부 2018920031 유승리)
1076399156.000000	997901390.000000	1002110191.000000

[예 1-4] i < 150, j < 150

- 예상 출력: 2*150*150 = 45000

- 평균 출력: 44841.666667

- 오차율: 0.352%

1회	2회	3회
Main Thread completed Thread_1 completed 44525.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 45000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 45000.000000 (컴퓨터과학부 2018920031 유승리)
44525.000000	45000.000000	45000.000000

[예 1-5] i < 200, j < 200

- 예상 출력: 2*200*200= 80000

- 평균 출력: 67762.000000

- 오차율: 15.298%

1회	2회	3회
Main Thread completed Thread_1 completed 57265.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 80000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 66021.000000 (컴퓨터과학부 2018920031 유승리)
57265.000000	80000.000000	66021.000000

[예 1-6] i < 10000, j < 10000

- 예상 출력: 2*10000*10000 = 200000000

- 평균 출력: 102920096.333333

- 오차율: 48.540%

1회	2회	3회
Thread_1 completed Main Thread completed 104991596.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 101590012.000000 (컴퓨터과학부 2018920031 유승리)	Thread_1 completed Main Thread completed 102178681.000000 (컴퓨터과학부 2018920031 유승리)
104991596.000000	101590012.000000	102178681.000000

〉 이유

이 프로그램은 공유변수 shared_var를 이중 for문 내에서 증가시키는 multithread 프로그램이므로 이 임계구역에서 race condition이 발생할 수 있다. 즉, 동시에 여러 thread가 임계구역에 진입할 수 있기 때문에 상호배제(mutual exclusion)가 지켜지지 않을 수 있다.

for문의 반복 횟수가 적을수록 임계구역 내(shared_var++;)에서 발생 가능한 race condition의 절대적인 수 또한 작기 때문에 [예 1-1], [예 1-4]와 같이 for문의 반복 횟수가 적은 조건에서는 오차율이 0% 또는 0%에 근접하게 출력되었다. 하지만 for문의 반복 횟수가 많아질수록 임계구역 내 공유변수 shared_var에 대해 발생할 수 있는 race condition의 절대적인 수 또한 커지기 때문에 [예 1-3], [예 1-6]에서는 오차율이 50%에 근접하게 출력되었다.

#2 Peterson's Algorithm을 이용하여 shared_var을 증가시킬 때 상호배제가 되도록 수정하여 실험한 결과를 레포트에 제시하라.

(#1번에서 상호배제가 지켜지지 않았던 경우 중 [예 1-2]와 [예 1-6]의 경우를 사용하였다.)

enter_region(), leave_region() 위치	
for문 밖	for문 안

enter_region(0);
for (i = 0; i < 10000; i++) {
for (j = 0; j < 10000; j++) {
shared_var++;
}
}
leave_region(0);
	for (i = 0; i < 10000; i++) {
	for (j = 0; j < 10000; j++) {
	enter_region(0);
	shared_var++;
	leave_region(0);
	}
	}

〉실험 결과

[예 2-1] i 〈 1000, j 〈 1000 // for문 밖

- 예상 출력: 2*1000*1000 = 2000000

- 평균 출력: 2000000.000000

- 오차율: 0%

1회	2회	3회
Main Thread completed Thread_1 completed 2000000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 2000000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 2000000.000000 (컴퓨터과학부 2018920031 유승리)
2000000.000000	2000000.000000	2000000.000000

[예 2-2] i 〈 10000, j 〈 10000 // for문 밖

- 예상 출력: 2*10000*10000 = 200000000

- 평균 출력: 200000000.000000

- 오차율: 0%

1회	2회	3회
Main Thread completed Thread_1 completed 200000000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 200000000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 200000000.000000 (컴퓨터과학부 2018920031 유승리)
200000000.000000	200000000.000000	200000000.000000

[예 2-3] i 〈 1000, j 〈 1000 // for문 안

- 예상 출력: 2*1000*1000 = 2000000

- 평균 출력: 1998061.666667

- 오차율: 0.097%

1회	2회	3회
Main Thread completed Thread_1 completed 1997561.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 1997433.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 1999191.000000 (컴퓨터과학부 2018920031 유승리)
1997561.000000	1997433.000000	1999191.000000

[예 2-4] i 〈 10000, j 〈 10000 // for문 안

- 예상 출력: 2*10000*10000 = 200000000

- 평균 출력: 199944100.666667

- 오차율: 0.028%

1회	2회	3회
Main Thread completed Thread_1 completed 199950331.000000 (컴퓨터과학부 2018920031 유승리)	Thread_1 completed Main Thread completed 199963325.000000 (컴퓨터과학부 2018920031 유승리)	Thread_1 completed Main Thread completed 199918646.000000 (컴퓨터과학부 2018920031 유승리)
199950331.000000	199963325.000000	199918646.000000

〉 분석

Peterson's Algorithm의 `enter_region()`과 `leave_region()`을 for문 밖에서 호출한 경우에는 오차율 0%로 출력되었다. 하지만 `enter_region()`과 `leave_region()`을 for문 안에서 호출한 경우에는 0.1% 미만의 오차율이 발생하였다. 이는 [예 1-2], [예 1-6]에서의 오차율인 40~50%에 비하여 매우 작은 오차율이므로 Peterson's Algorithm이 상호배제를 효과적으로 구현하지만, 실제 프로그램에서는 상호배제가 완벽하게 구현되지 못할 수 있음을 보여준다.

#3 경쟁상황을 줄이기 위해 shared_var의 선언에 volatile을 추가하고, volatile 키워드의 기능은 무엇인지 기술하라. 그리고 적절한 지점에 메모리 배리어 __asm mfence를 추가하여 실험하라. 이 배리어의 역할은 무엇인지, 어느 부분에 이 배리어를 추가해야 하는지, 그리고 그 결과는 어떻게 되는지 적어라.

(#2번에서 오차가 발생했던, for문 안에서 enter_region(), leave_region()하는 경우인 [예 2-3]와 [예 2-4]의 경우를 사용하였다.)

〉volatile 키워드의 기능

volatile 키워드는 데이터를 메모리에서 CPU 레지스터로 로드 및 연산 후 다시 메모리에 결과값을 저장하는 과정을 컴파일러에서 생략 및 변경하지 못하도록 하여 원칙적인 수행 절차를 거치도록 하는 키워드이다. 즉, 프로그램의 문맥 상에서는 레지스터만을 이용하여 동작할 수 있는 경우이더라도, 컴파일러가 해당 작업을 메모리에도 저장하도록 하여 컴파일러의 재량을 제한하는 것이다. 또한, volatile로 선언된 변수는 컴파일러 최적화 기법 중 하나인 명령어 재배치(instruction reordering)의 대상에서 제외되는데, 명령어 재배치란 연산의 빠른 수행을 위해 일부 연산의 순서를 바꾸는 최적화 기법이다.

(source | - https://ko.wikipedia.org/wiki/Volatile_변수

- http://summerlight-textcube.blogspot.com/2009/11/volatile과-메모리-배리어.html

- https://zapiro.tistory.com/entry/volatile-변수의-쓰임)

〉 메모리 배리어(__asm mfence)의 역할과 위치

메모리 배리어(memory barrier)는 중앙 처리 장치나 컴파일러에게 특정 연산의 순서를 강제하도록 하는 기능이다. 변수에 volatile 속성을 부여하면 컴파일러에 의한 명령어 재배치는 막을 수 있으나, CPU에 의한 비순차적 명령어 처리 기법(out-of-order execution)에 의해 서로 종속성이 없는 연산의 순서가 바뀌는 것은 해결할 수 없다. 이는 속도 향상을 위한 기술로 컴파일과 무관하게 런타임에 이루어지는 것이기 때문이며, 이에 따라 multi thread 환경에서는 잘못된 결과가 나올 수 있다. 이와 같이 특정 메모리 연산의 순서가 보장되지 않을 때, mfence 명령어를 통해 해당 명령어 이전의 모든 메모리 연산에 대해 직렬화 연산을 수행함으로써 이 실행 순서를 강제할 수 있다.

Peterson's Algorithm이 적용된 이 프로그램의 경우, 자신의 interested flag를 TRUE로 설정하고 turn을 자신으로 설정하는 것이 중요한 연산이므로 다른 연산으로 인해 순서가 바뀌면 안된다. 따라서 mfence 명령어는 enter_region() 함수 내부에서 해당 연산의 다음 위치에 추가되어야 한다. 하지만 while문이나 leave_region()에서 interested flag를 FALSE로 설정하는 것은 조금 늦춰지더라도 괜찮기 때문에 메모리 배리어는 해당 위치에만 추가되면 된다.

(source | - https://ko.wikipedia.org/wiki/Volatile_변수

- http://summerlight-textcube.blogspot.com/2009/11/volatile과-메모리-배리어.html

- https://talkingaboutme.tistory.com/entry/Distributed-System-lock

- https://spcl.inf.ethz.ch/Teaching/2017-dphpc/assignments/locks_assignment.pdf

- http://egloos.zum.com/studyfoss/v/5454726

- https://ko.wikipedia.org/wiki/메모리_배리어

- https://popcorntree.tistory.com/15

- https://code.i-harness.com/ko-kr/q/b0d3a2)

```
void enter_region(int process)
{
	int other;

	other = 1 - process;
	interested[process] = TRUE;
	turn = process;
	__asm mfence; // 메모리 배리어
	while (turn == process && interested[other] == TRUE);
}

void leave_region(int process)
{
	interested[process] = FALSE;
}
```

〉실험 결과

[예 3-1] i〈1000, j〈1000 // for문 안 // volatile

- 예상 출력: 2*1000*1000 = 2000000

- 평균 출력: 1998343. 000000

- 오차율: 0.083%

1회	2회	3회
Main Thread completed Thread_1 completed 1997380.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 1999765.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 1997884.000000 (컴퓨터과학부 2018920031 유승리)
1997380.000000	1999765.000000	1997884.000000

[예 3-2] i < 10000, j < 10000 // for문 안 // volatile

- 예상 출력: 2*10000*10000 = 200000000

- 평균 출력: 199869114.000000

- 오차율: 0.065%

1회	2회	3회
Main Thread completed Thread_1 completed 199892104.000000 (컴퓨터과학부 2018920031 유승리)	Thread_1 completed Main Thread completed 199782061.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 199933177.000000 (컴퓨터과학부 2018920031 유승리)
199892104.000000	199782061.000000	199933177.000000

[예 3-3] i < 1000, j < 1000 // for문 안 // volatile + 메모리 배리어

- 예상 출력: 2*1000*1000 = 2000000

- 평균 출력: 2000000.000000

- 오차율: 0% → race condition X, 상호배제가 완벽하게 이루어짐

1회	2회	3회
Main Thread completed Thread_1 completed 2000000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 2000000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 2000000.000000 (컴퓨터과학부 2018920031 유승리)
2000000.000000	2000000.000000	2000000.000000

[예 3-4] i < 10000, j < 10000 // for문 안 // volatile + 메모리 배리어

- 예상 출력: 2*10000*10000 = 200000000

- 평균 출력: 200000000.000000

- 오차율: 0% → race condition X, 상호배제가 완벽하게 이루어짐

1회	2회	3회
Main Thread completed Thread_1 completed 200000000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 200000000.000000 (컴퓨터과학부 2018920031 유승리)	Main Thread completed Thread_1 completed 200000000.000000 (컴퓨터과학부 2018920031 유승리)
200000000.000000	200000000.000000	200000000.000000

 # 소스 코드 ([예 3-4])

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <conio.h>

#define FALSE 0
#define TRUE  1
#define N     2

/*
** 운영체제 과제 1:     상호배제
** 학번과 이름:         2018920031 유승리
** 제출일:              2020-04-09-목
*/

DWORD WINAPI thread_func_1(LPVOID lpParam);
void enter_region(int process);
void leave_region(int process);

volatile double     shared_var = 0.0;
volatile int job_complete[2] = {0, 0};
int    turn;
int    interested[N] = { FALSE, FALSE };


int main(void)
{
       DWORD dwThreadId_1, dwThrdParam_1 = 1;
       HANDLE hThread_1;
       int    i, j;

       // Create Thread 1
       hThread_1 = CreateThread(
       NULL,                       // default security attributes
       0,                         // use default stack size
       thread_func_1,                  // thread function
       &dwThrdParam_1,                 // argument to thread function
       0,                         // use default creation flags
       &dwThreadId_1);                 // returns the thread identifier

   // Check the return value for success.

    if (hThread_1 == NULL)
    {
       printf("Thread 1 creation error\n");
          exit(0);
    }
    else
    {
       CloseHandle( hThread_1 );
    }

       /* I am main thread */
       /* Now Main Thread and Thread 1 runs concurrently */
       for (i = 0; i < 10000; i++) {
              for (j = 0; j < 10000; j++) {
                     enter_region(0);
                     shared_var++;
                     leave_region(0);
              }
       }

       printf("Main Thread completed\n");
       job_complete[0] = 1;
       while (job_complete[1] == 0) ;
```

```
	printf("%f\n", shared_var);
	printf("\n(컴퓨터과학부 2018920031 유승리)\n");
	_getch();
	ExitProcess(0);
}


DWORD WINAPI thread_func_1(LPVOID lpParam)
{
	int	i, j;

	for (i = 0; i < 10000; i++) {
		for (j = 0; j < 10000; j++) {
			enter_region(1);
			shared_var++;
			leave_region(1);
		}
	}

	printf("Thread_1 completed\n");
	job_complete[1] = 1;
	ExitThread(0);
}


void enter_region(int process)
{
	int	other;

	other = 1 - process;
	interested[process] = TRUE;
	turn = process;
	__asm mfence;// 메모리 배리어
	while (turn == process && interested[other] == TRUE);
}


void leave_region(int process)
{
	interested[process] = FALSE;
}
```