[REPORT]

■과 목 명: 운영체제 (SWE3004-42)

■담 당 교 수: 신동군 교수님

■제 출 일: 2022년 5월 7일

■학　　과: 수학과

■학　　번: 2016314786

■성　　명: 김호진

Chap 32. Deadlock

수학과 김호진 (2016314786)

1. First let's make sure you understand how the programs generally work, and some of the key options. Study the code in vector-deadlock.c**, as well as in** main-common.c **and related files. Now, run** ./vector-deadlock -n 2 -l 1 -v**, which instantiates two threads (**-n 2**), each of which does one vector add (**-l 1**), and does so in verbose mode (**-v**). Make sure you understand the output. How does the output change from run to run?**

프로그램 'vector-deadlock'을 거듭하여 실행하다 보면, 다음과 같이 실행되는 thread의 순서가 변하는 경우가 발생한다. 다만, '-n 2 -l 1 -v'에서는 circular wait이 나타나지 않으므로 deadlock에 빠지지 않는다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-deadlock -n 2 -l 1 -v
->add(0, 1)
<-add(0, 1)
                ->add(0, 1)
                <-add(0, 1)
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-deadlock -n 2 -l 1 -v
->add(0, 1)
<-add(0, 1)
                ->add(0, 1)
                <-add(0, 1)
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-deadlock -n 2 -l 1 -v
->add(0, 1)
<-add(0, 1)
                ->add(0, 1)
                <-add(0, 1)
```

2. Now add the -d **flag, and change the number of loops (**-l**) from 1 to higher numbers. What happens? Does the code (always) deadlock?**

-d flag를 추가하고 루프의 수(-l)를 1보다 더 큰 수인 10000으로 변경한 './vector-deadlock -n 2 -l 10000 -v -d'를 실행하면, 다음과 같이 deadlock이 간헐적으로 발생하는 것을 확인할 수 있다. 그러나 deadlock이 항상 발생하는 것은 아니며, 대다수의 경우에서는 정상적으로 실행된다.

```
                ->add(1, 0)
                <-add(1, 0)
                ->add(1, 0)
                <-add(1, 0)
                ->add(1, 0)
->add(0, 1)
```

3. How does changing the number of threads (-n) change the outcome of the program? Are there any values of -n that ensure no deadlock occurs?

Thread의 수(-n)를 늘릴수록 circular wait이 발생하는 경우가 증가하기 때문에 프로그램이 deadlock에 빠질 확률이 높아진다. 만약 thread의 수(-n)가 1개라면, 오직 하나의 thread만이 vector에 접근하므로 deadlock이 절대 발생하지 않는다.

4. Now examine the code in vector-global-order.c. First, make sure you understand what the code is trying to do; do you understand why the code avoids deadlock? Also, why is there a special case in this vector_add() routine when the source and destination vectors are the same?

vector-global-order.c 코드를 살펴보면, lock address에 따라 lock ordering을 적용하여 deadlock을 방지하는 것을 알 수 있다. 즉, 해당 코드에서는 lock을 획득했을 때 global order를 사용하여 deadlock을 방지하고 circular wait이 발생하지 않도록 만든다. 그런데 source vector와 destination vector가 동일하면, same lock을 두 번 기다리게 되어 deadlock이 발생하기 때문에 이를 방지하기 위해서 vector_add()의 routine에 special case가 존재한다.

5. Now run the code with the following flags: -t -n 2 -l 100000 -d. How long does the code take to complete? How does the total time change when you increase the number of loops, or the number of threads?

flag: -t -n 2 -l 100000 -d를 사용하여 코드를 실행하면 완료까지 0.03초가 소요된다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 100000 -d
Time: 0.03 seconds
```

Loop의 수를 늘리게 되면, 소요시간은 Loop 수에 따라 선형적으로 증가한다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 100000 -d
Time: 0.03 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 1000000 -d
Time: 0.26 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 2000000 -d
Time: 0.51 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 4000000 -d
Time: 1.06 seconds
```

Thread의 수를 늘리게 되면, 소요시간은 thread의 수에 따라 선형적으로 증가한다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 100000 -d
Time: 0.03 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 20 -l 100000 -d
Time: 1.27 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 40 -l 100000 -d
Time: 2.51 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 80 -l 100000 -d
Time: 5.02 seconds
```

6. What happens if you turn on the parallelism flag (-p)? How much would you expect performance to change when each thread is working on adding different vectors (which is what -p enables) versus working on the same ones?

병렬 flag(-p)를 사용하면, thread가 lock을 얻기 위해 wait 할 필요가 없어지므로 lock contention 이 발생하지 않아 프로그램을 완료하는데 소요되는 시간이 줄어든다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 100000 -d
Time: 0.03 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 100000 -d -p
Time: 0.02 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 1000000 -d
Time: 0.29 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 1000000 -d -p
Time: 0.14 seconds
```

```
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 100000 -d
Time: 0.03 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 100000 -d -p
Time: 0.02 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 20 -l 100000 -d
Time: 1.24 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 20 -l 100000 -d -p
Time: 0.05 seconds
```

7. Now let's study vector-try-wait.c**. First make sure you understand the code. Is the first call to** pthread_mutex_trylock() **really needed? Now run the code. How fast does it run compared to the global order approach? How does the number of retries, as counted by the code, change as the number of threads increases?**

vector-try-wait.c에서 pthread_mutex_trylock()에 대한 first call은 필요하지 않다. 해당 방식은 global order approach 방식과 비교했을 때 느린 실행속도를 보이며, thread의 수가 증가함에 따라 재시도 횟수가 많아지는 것을 확인할 수 있다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 100000 -d
Time: 0.05 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-try-wait -t -n 2 -l 100000 -d
Retries: 1063534
Time: 0.07 seconds
```

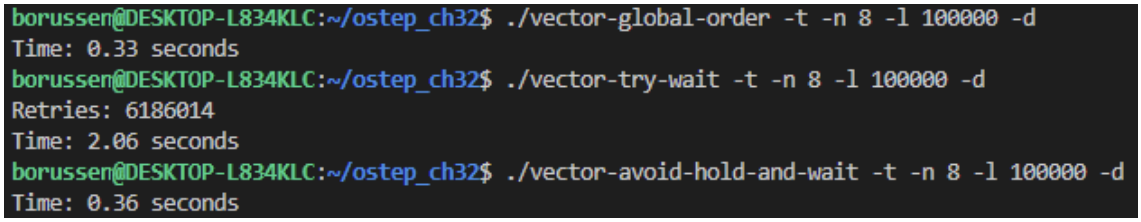
```
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 8 -l 100000 -d
Time: 0.35 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-try-wait -t -n 8 -l 100000 -d
Retries: 6143833
Time: 2.06 seconds
```

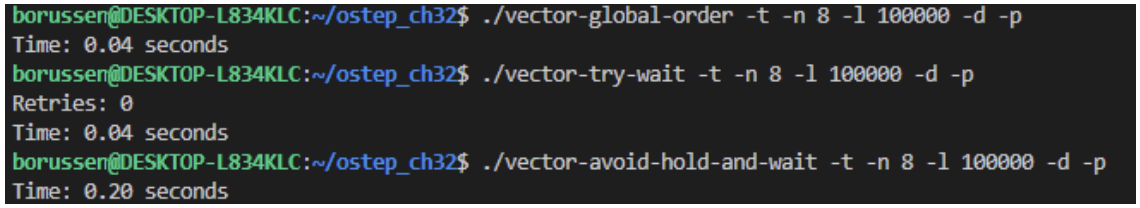
-p flag를 사용하면, 재시도 횟수가 0이 되어 global order approach 방식과 동일한 성능을 가진다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 8 -l 100000 -d -p
Time: 0.04 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-try-wait -t -n 8 -l 100000 -d -p
Retries: 0
Time: 0.04 seconds
```

8. Now let's look at vector-avoid-hold-and-wait.c**. What is the main problem with this approach? How does its performance compare to the other versions, when running both with** -p **and without it?**

vector-avoid-hold-and-wait.c에서 -p flag를 사용하는 경우, 모든 threads가 서로 다른 vector에서 작업을 하더라도 global lock을 기다려야 하기 때문에 global order approach 방식이나 try wait approach 방식과 같은 다른 접근법에 비해 성능이 저하된다. 그러나, -p flag를 사용하지 않는 경우에는 다른 접근법에 비해 좋은 성능을 보인다.





9. Finally, let's look at vector-nolock.c**. This version doesn't use locks at all; does it provide the exact same semantics as the other versions? Why or why not?**

vector-nolock.c에서는 lock을 사용하지 않는 대신 fetch_and_add를 사용한다. 이는 다른 버전과 동일한 semantics을 제공하기는 하지만, 메모리에 있는 주소를 잠그기 위해 mutex 대신 assembly에 의존하면서 프로그램을 훨씬 더 단순하게 만든다.

10. Now compare its performance to the other versions, both when threads are working on the same two vectors (no -p**) and when each thread is working on separate vectors (**-p**). How does this no-lock version perform?**

thread가 동일한 두 vectors (no-p)에서 작동할 때와 각 thread가 별도의 vector(-p)에서 작동할 때 모두 no-lock 버전은 다른 버전에 비해 좋지 못한 성능을 보인다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 200000 -d
Time: 0.07 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-try-wait -t -n 2 -l 200000 -d
Retries: 1972768
Time: 0.14 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-avoid-hold-and-wait -t -n 2 -l 200000 -d
Time: 0.14 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-nolock -t -n 2 -l 200000 -d
Time: 0.89 seconds
```

```
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-global-order -t -n 2 -l 200000 -d -p
Time: 0.05 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-try-wait -t -n 2 -l 200000 -d -p
Retries: 0
Time: 0.04 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-avoid-hold-and-wait -t -n 2 -l 200000 -d -p
Time: 0.09 seconds
borussen@DESKTOP-L834KLC:~/ostep_ch32$ ./vector-nolock -t -n 2 -l 200000 -d -p
Time: 0.14 seconds
```