

Homework7

Thread API

2016314364 박수헌

Question 1.

main-race.c 는 아래와 같았다.

```
1 #include <stdio.h>
2
3 #include "common_threads.h"
4
5 int balance = 0;
6
7 void* worker(void* arg) {
8     balance++; // unprotected access
9     return NULL;
10 }
11
12 int main(int argc, char *argv[]) {
13     pthread_t p;
14     pthread_create(&p, NULL, worker, NULL);
15     balance++; // unprotected access
16     pthread_join(p, NULL);
17     return 0;
18 }
```

14번째 줄에서, balance 라는 변수에 1을 더하는 thread 를 만든 것을 확인할 수 있다. 하지만 15번째 줄에서도 balance 에 1을 더해준다. 그리고 16번째 줄에서 join 을 통해 thread 가 끝나길 기다리며 프로그램의 실행이 종료된다. 결과적으로 balance 의 값은 2가 될 것을 알 수 있다. 16번과 17번 줄 사이에

printf("balance : %d\n", balance);

를 추가하고 컴파일해 실행하자 역시 balance : 2 라는 결과가 나왔다.

```
Report7 [main] ./main
balance : 2
```

즉, thread 안에 있는 변수가 다른 곳에서도 접근되었다는 뜻이다.

valgrind -tool=helgrind ./main-race 를 통해 확인해보았을 때 결과는 아래와 같았다.

```

==7271==
==7271== ---Thread-Announcement-----
==7271==
==7271== Thread #1 is the program's root thread
==7271==
==7271== ---Thread-Announcement-----
==7271==
==7271== Thread #2 was created
==7271==   at 0x49B0282: clone (clone.S:71)
==7271==   by 0x48732EB: create_thread (createthread.c:101)
==7271==   by 0x4874E0F: pthread_create@@GLIBC_2.2.5 (pthread_create.c:817)
==7271==   by 0x4842917: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==7271==   by 0x109209: main (in /home/soohun/Desktop/coconut/OperatingSystems/Report7/main-race)
==7271==
==7271== -----
==7271== Possible data race during read of size 4 at 0x10C014 by thread #1
==7271== Locks held: none
==7271==   at 0x10922D: main (in /home/soohun/Desktop/coconut/OperatingSystems/Report7/main-race)
==7271==
==7271== This conflicts with a previous write of size 4 by thread #2
==7271== Locks held: none
==7271==   at 0x1091BE: worker (in /home/soohun/Desktop/coconut/OperatingSystems/Report7/main-race)
==7271==   by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==7271==   by 0x4874608: start_thread (pthread_create.c:477)
==7271==   by 0x49B0292: clone (clone.S:95)
==7271== Address 0x10C014 is 0 bytes inside data symbol "balance"
==7271==
==7271== -----
==7271== Possible data race during write of size 4 at 0x10C014 by thread #1
==7271== Locks held: none
==7271==   at 0x109236: main (in /home/soohun/Desktop/coconut/OperatingSystems/Report7/main-race)
==7271==
==7271== This conflicts with a previous write of size 4 by thread #2
==7271== Locks held: none
==7271==   at 0x1091BE: worker (in /home/soohun/Desktop/coconut/OperatingSystems/Report7/main-race)
==7271==   by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==7271==   by 0x4874608: start_thread (pthread_create.c:477)
==7271==   by 0x49B0292: clone (clone.S:95)
==7271== Address 0x10C014 is 0 bytes inside data symbol "balance"
==7271==

```

첫 번째 block 인 Thread-Announcement 에서 thread#1이 main 함수 thread 라는 것을 알려준다.

두 번째 block 에서는 worker thread 가 생겨난 것을 나타내는 것을 볼 수 있다. Thread1의 0x10C014에서 4byte 크기의 변수 read 가 thread#2에서의 4byte 의 write 와 충돌이 일어난다고 표시해주고 있다.

Question 2.

main-race.c 의 코드에서 15번째 줄인 main 함수에서 balance 함수에 1을 더해주는 줄을 주석처리한 후 helgrind 로 다시 확인해 보았다. 결과는 아래와 같았다.

```

==7677== Helgrind, a thread error detector
==7677== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==7677== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==7677== Command: ./main-race
==7677==
==7677== Use --history-level=approx or =none to gain increased speed, at
==7677== the cost of reduced accuracy of conflicting-access information
==7677== For lists of detected and suppressed errors, rerun with: -s
==7677== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Error summary 에 아무런 error 도 나타나지 않았다.

Question 3.

main-deadlock.c 의 코드는 아래와 같았다.

```
1 #include <stdio.h>
2
3 #include "common_threads.h"
4
5 pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER;
6 pthread_mutex_t m2 = PTHREAD_MUTEX_INITIALIZER;
7
8 void* worker(void* arg) {
9     if ((long long) arg == 0) {
10         Pthread_mutex_lock(&m1);
11         Pthread_mutex_lock(&m2);
12     } else {
13         Pthread_mutex_lock(&m2);
14         Pthread_mutex_lock(&m1);
15     }
16     Pthread_mutex_unlock(&m1);
17     Pthread_mutex_unlock(&m2);
18     return NULL;
19 }
20
21 int main(int argc, char *argv[]) {
22     pthread_t p1, p2;
23     Pthread_create(&p1, NULL, worker, (void *) (long long) 0);
24     Pthread_create(&p2, NULL, worker, (void *) (long long) 1);
25     Pthread_join(p1, NULL);
26     Pthread_join(p2, NULL);
27     return 0;
28 }
```

23번째 줄에서 생성한 p1 thread 의 arg 는 0이다. 따라서 m1, m2 순서로 lock 한 후, m1, m2 순서로 unlock 한다.

그 후 24번째 줄에서 생성한 p2 thread 의 arg 는 1이므로 m2, m1 순서로 lock 한 후, m1, m2 순서로 unlock 한다.

즉, 두 개의 thread 가 서로를 lock 하는 것을 볼 수 있다.

Question 4.

valgrind -tool=helgrind ./main-deadlock 의 결과는 아래와 같았다.

```
= Thread #3: lock order "0x10C040 before 0x10C080" violated
=
= Observed (incorrect) order is: acquisition of lock at 0x10C080
=   at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x109269: worker (main-deadlock.c:13)
=   by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x4874608: start_thread (pthread_create.c:477)
=   by 0x49B0292: clone (clone.S:95)
=
= followed by a later acquisition of lock at 0x10C040
=   at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x109298: worker (main-deadlock.c:14)
=   by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x4874608: start_thread (pthread_create.c:477)
=   by 0x49B0292: clone (clone.S:95)
=
= Required order was established by acquisition of lock at 0x10C040
=   at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x10920B: worker (main-deadlock.c:10)
=   by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x4874608: start_thread (pthread_create.c:477)
=   by 0x49B0292: clone (clone.S:95)
=
= followed by a later acquisition of lock at 0x10C080
=   at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x10923A: worker (main-deadlock.c:11)
=   by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x4874608: start_thread (pthread_create.c:477)
=   by 0x49B0292: clone (clone.S:95)
=
= Lock at 0x10C040 was first observed
=   at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x10920B: worker (main-deadlock.c:10)
=   by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x4874608: start_thread (pthread_create.c:477)
=   by 0x49B0292: clone (clone.S:95)
= Address 0x10C040 is 0 bytes inside data symbol "m1"
=
= Lock at 0x10C080 was first observed
=   at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x10923A: worker (main-deadlock.c:11)
=   by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
=   by 0x4874608: start_thread (pthread_create.c:477)
=   by 0x49B0292: clone (clone.S:95)
= Address 0x10C080 is 0 bytes inside data symbol "m2"
=
```

0x10C040과 0x10C080에서 lock 이 실행되었지만, 밑의 두 블록에서는 이미 lock 이 되어있는 thread 를 lock 했다는 order 에 관한 에러를 표시했다.

Question 5.

main-deadlock-global.c 의 코드는 아래와 같았다.

```
1 #include <stdio.h>
2
3 #include "common_threads.h"
4
5 pthread_mutex_t g = PTHREAD_MUTEX_INITIALIZER;
6 pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER;
7 pthread_mutex_t m2 = PTHREAD_MUTEX_INITIALIZER;
8
9 void* worker(void* arg) {
10     pthread_mutex_lock(&g);
11     if ((long long) arg == 0) {
12         pthread_mutex_lock(&m1);
13         pthread_mutex_lock(&m2);
14     } else {
15         pthread_mutex_lock(&m2);
16         pthread_mutex_lock(&m1);
17     }
18     pthread_mutex_unlock(&m1);
19     pthread_mutex_unlock(&m2);
20     pthread_mutex_unlock(&g);
21     return NULL;
22 }
23
24 int main(int argc, char *argv[]) {
25     pthread_t p1, p2;
26     pthread_create(&p1, NULL, worker, (void *) (long long) 0);
27     pthread_create(&p2, NULL, worker, (void *) (long long) 1);
28     pthread_join(p1, NULL);
29     pthread_join(p2, NULL);
30     return 0;
31 }
```

위의 코드에서는 p1 thread 가 우선 g mutex 의 lock 을 한 후, 위와 같은 과정을 거치고 unlock 한다.. 그 후, p2 thread 또한 g 를 lock 한 후 같은 과정을 거친 후 unlock 한다.

결국 main-deadlock 과 같은 lock order error 을 helgrind 를 통해 확인했다.

```
3== Thread #3: lock order "0x10C080 before 0x10C0C0" violated
3==
3== Observed (incorrect) order is: acquisition of lock at 0x10C0C0
3== at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x109298: worker (main-deadlock-global.c:15)
3== by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x4874608: start_thread (pthread_create.c:477)
3== by 0x4980292: clone (clone.S:95)
3==
3== followed by a later acquisition of lock at 0x10C080
3== at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x1092C7: worker (main-deadlock-global.c:16)
3== by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x4874608: start_thread (pthread_create.c:477)
3== by 0x4980292: clone (clone.S:95)
3==
3== Required order was established by acquisition of lock at 0x10C080
3== at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x10923A: worker (main-deadlock-global.c:12)
3== by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x4874608: start_thread (pthread_create.c:477)
3== by 0x4980292: clone (clone.S:95)
3==
3== followed by a later acquisition of lock at 0x10C0C0
3== at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x109269: worker (main-deadlock-global.c:13)
3== by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x4874608: start_thread (pthread_create.c:477)
3== by 0x4980292: clone (clone.S:95)
3==
3== Lock at 0x10C080 was first observed
3== at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x10923A: worker (main-deadlock-global.c:12)
3== by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x4874608: start_thread (pthread_create.c:477)
3== by 0x4980292: clone (clone.S:95)
3== Address 0x10C080 is 0 bytes inside data symbol "m1"
3==
3== Lock at 0x10C0C0 was first observed
3== at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x109269: worker (main-deadlock-global.c:13)
3== by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
3== by 0x4874608: start_thread (pthread_create.c:477)
3== by 0x4980292: clone (clone.S:95)
3== Address 0x10C0C0 is 0 bytes inside data symbol "m2"
3==
```


Question 6.

main-signal.c 코드는 아래와 같았다.

```
1 #include <stdio.h>
2
3 #include "common_threads.h"
4
5 int done = 0;
6
7 void* worker(void* arg) {
8     printf("this should print first\n");
9     done = 1;
10    return NULL;
11 }
12
13 int main(int argc, char *argv[]) {
14     pthread_t p;
15     Pthread_create(&p, NULL, worker, NULL);
16     while (done == 0)
17         ;
18     printf("this should print last\n");
19     return 0;
20 }
```

15번 줄에서 p thread 가 만들어진다. 우선 8번 째 줄의 print 문이 print 되고 done 변수는 1이된다. 그 후 16번 째 줄에 while 문에서 done 이 1일 때, 18번째 줄의 print 문을 print 한 후 종료된다. 이 구조에서는 p thread 가 실행되는 동안 main thread 는 그저 무한 while loop 를 돌게 되므로 비효율적이다. 만약 p thread 가 매우 오랜 시간이 걸리는 함수라면 main thread 는 기약없이 기다려야만 한다.

Question 7.

```
==9177== -----
==9177==
==9177== Possible data race during read of size 4 at 0x10C014 by thread #1
==9177== Locks held: none
==9177==    at 0x109239: main (main-signal.c:16)
==9177==
==9177== This conflicts with a previous write of size 4 by thread #2
==9177== Locks held: none
==9177==    at 0x1091C5: worker (main-signal.c:9)
==9177==   by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==9177==   by 0x4874608: start_thread (pthread_create.c:477)
==9177==   by 0x49B0292: clone (clone.S:95)
==9177== Address 0x10C014 is 0 bytes inside data symbol "done"
==9177== -----
==9177==
==9177== Possible data race during write of size 1 at 0x52841A5 by thread #1
==9177== Locks held: none
==9177==    at 0x48488A6: memcpy (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==9177==   by 0x49207B1: _IO_new_file_xsputn (fileops.c:1236)
==9177==   by 0x49207B1: _IO_file_xsputn@GLIBC_2.2.5 (fileops.c:1197)
==9177==   by 0x4915677: puts (ioputs.c:40)
==9177==   by 0x10924E: main (main-signal.c:18)
==9177== Address 0x52841A5 is 21 bytes inside a block of size 1,024 alloc'd
==9177==    at 0x483C893: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==9177==   by 0x4912E83: _IO_file_doallocate (filedoalloc.c:101)
==9177==   by 0x492304F: _IO_doallocbuf (genops.c:347)
==9177==   by 0x49220AF: _IO_file_overflow@GLIBC_2.2.5 (fileops.c:745)
==9177==   by 0x4920834: _IO_new_file_xsputn (fileops.c:1244)
==9177==   by 0x4920834: _IO_file_xsputn@GLIBC_2.2.5 (fileops.c:1197)
==9177==   by 0x4915677: puts (ioputs.c:40)
==9177==   by 0x1091C4: worker (main-signal.c:8)
==9177==   by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==9177==   by 0x4874608: start_thread (pthread_create.c:477)
==9177==   by 0x49B0292: clone (clone.S:95)
==9177== Block was alloc'd by thread #2
==9177==
==9177== this should print last
==9177== -----
```

변수 done 의 read 와 write 의 data race 와 printf 구문의 data race 가 가능하다는 예러가 표시되었다. 따라서 code 는 잘못되었다.

Question 8.

위의 main-signal.c 코드처럼 thread 가 끝날 때까지 loop 를 돌며 기다리는 것이 아니라, signal 을 이용해 thread 가 끝났음을 main thread 가 알 수 있다. 따라서 thread 가 실행되는 중에도 main thread 는 자원을 낭비하지 않고 다른 일련의 코드들을 실행할 수 있으므로, 성능적인 면에서도 뛰어나고, 올바른 코드로 볼 수 있다.

Question 9.

아무런 오류도 발견되지 않았다.