Homework7

Thread API

2016314364 박수헌

Question 1.

main-race.c 는 아래와 같았다.

```c
#include <stdio.h>

#include "common_threads.h"

int balance = 0;

void* worker(void* arg) {
    balance++; // unprotected access
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p;
    Pthread_create(&p, NULL, worker, NULL);
    balance++; // unprotected access
    Pthread_join(p, NULL);
    return 0;
}
```

14번째 줄에서, balance 라는 변수에 1을 더하는 thread 를 만든 것을 확인할 수 있다. 하지만 15번째 줄에서도 balance 에 1을 더해준다. 그리고 16번 째 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

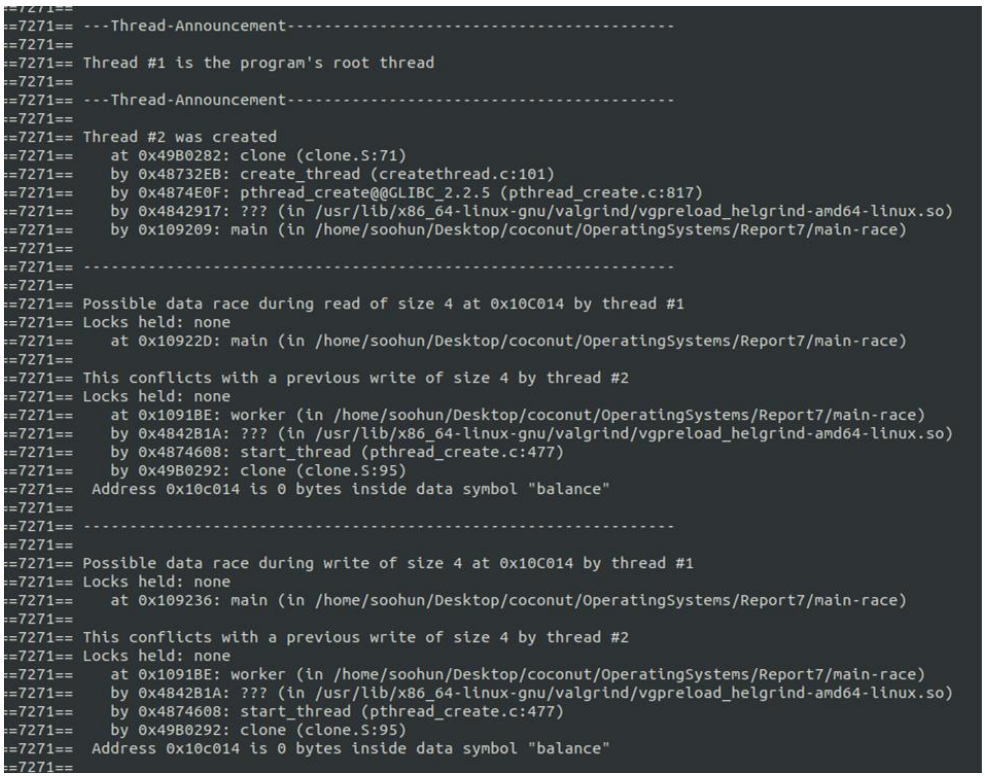
printf("balance : %d₩n", balance);

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.

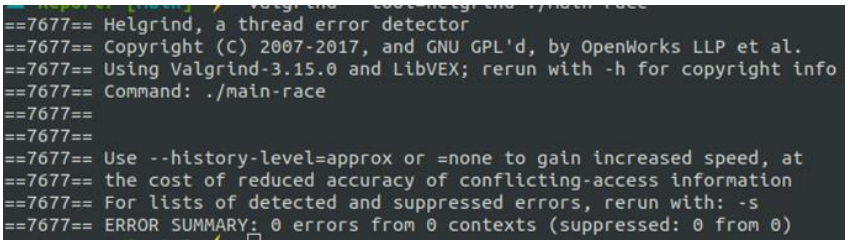


첫번 째 block 인 Thread-Announcement 에서 thread#1이 main 함수 thread 라는 것을 알려준다.

두 번 째 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.



Error summary 에 아무런 error 도 나타나지 않았다.

Question 3.

main-deadlock.c 의 코드는 아래와 같았다.

```c
#include <stdio.h>

#include "common_threads.h"

pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t m2 = PTHREAD_MUTEX_INITIALIZER;

void* worker(void* arg) {
    if ((long long) arg == 0) {
    Pthread_mutex_lock(&m1);
    Pthread_mutex_lock(&m2);
    } else {
    Pthread_mutex_lock(&m2);
    Pthread_mutex_lock(&m1);
    }
    Pthread_mutex_unlock(&m1);
    Pthread_mutex_unlock(&m2);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p1, p2;
    Pthread_create(&p1, NULL, worker, (void *) (long long) 0);
    Pthread_create(&p2, NULL, worker, (void *) (long long) 1);
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    return 0;
}
```

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 블럭에서는 이미 lock 이 되어있는 thread 를 lock 했다는 order 에 관한 에러를 표시했다.

Question 5.

main-deadlock-global.c 의 코드는 아래와 같았다.

```c
#include <stdio.h>

#include "common_threads.h"

pthread_mutex_t g = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t m2 = PTHREAD_MUTEX_INITIALIZER;

void* worker(void* arg) {
    Pthread_mutex_lock(&g);
    if ((long long) arg == 0) {
    Pthread_mutex_lock(&m1);
    Pthread_mutex_lock(&m2);
    } else {
    Pthread_mutex_lock(&m2);
    Pthread_mutex_lock(&m1);
    }
    Pthread_mutex_unlock(&m1);
    Pthread_mutex_unlock(&m2);
    Pthread_mutex_unlock(&g);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p1, p2;
    Pthread_create(&p1, NULL, worker, (void *) (long long) 0);
    Pthread_create(&p2, NULL, worker, (void *) (long long) 1);
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    return 0;
}
```

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.

```
==
== Thread #3: lock order "0x10C080 before 0x10C0C0" violated
==
== Observed (incorrect) order is: acquisition of lock at 0x10C0C0
==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==    by 0x109298: worker (main-deadlock-global.c:15)
==    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 0x1092C7: worker (main-deadlock-global.c:16)
==    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 0x10C080
==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==    by 0x10923A: worker (main-deadlock-global.c:12)
==    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 0x10C0C0
==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==    by 0x109269: worker (main-deadlock-global.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)
==
==  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-global.c:12)
==    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 "m1"
==
==  Lock at 0x10C0C0 was first observed
==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==    by 0x109269: worker (main-deadlock-global.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)
==  Address 0x10c0c0 is 0 bytes inside data symbol "m2"
==
```

Question 6.

main-signal.c 코드는 아래와 같았다.

```c
#include <stdio.h>

#include "common_threads.h"

int done = 0;

void* worker(void* arg) {
    printf("this should print first\n");
    done = 1;
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p;
    Pthread_create(&p, NULL, worker, NULL);
    while (done == 0)
    ;
    printf("this should print last\n");
    return 0;
}
```

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==
==9177== Possible data race during write of size 1 at 0x52841A5 by thread #1
==9177== Locks held: none
==9177==    at 0x48488A6: mempcpy (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==
this should print last
```

변수 done 의 read 와 write 의 data race 와 printf 구문의 data race 가 가능하다는 에러가 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.