1. First build main-race.c. Examine the code so you can see the (hopefully obvious) data race in the code. Now run helgrind (by typing valgrind --tool=helgrind main-race) to see how it reports the race. Does it point to the right lines of code? What other information does it give to you?

Main-race는 두 개의 스레드를 사용하는 프로그램이다. 이 두 개의 스레드를 사용하는 도중 동시 데이터 접근 문제가 발생한 것을 볼 수 있다.

```c
#include <stdio.h>

#include "mythreads.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 acces
    Pthread_join(p, NULL);
    return 0;
}
```

메인 스레드와 새로 만들어진 스레드가 모두 balance 값을 수정한다.

병렬적으로 실행 중인 두 스레드가 같은 데이터를 읽고 쓸 때 어떤 명령문이 먼저 실행될 지는 결정되지 않는다. 만약 한 스레드가 데이터를 읽고 다른 스레드는 수정한다면, 스레드는 원하지 않은 값을 읽거나 수정하게 될 수 있다. 이것이 바로 스레드의 동시성의 문제이다.

한 번 helgrind를 사용해 코드를 살펴보자.

```
==2176== ---Thread-Announcement------------------------------------------
==2176==
==2176== Thread #1 is the program's root thread
==2176==
==2176== ---Thread-Announcement------------------------------------------
==2176==
==2176== Thread #2 was created
==2176==    at 0x518287E: clone (clone.S:71)
==2176==    by 0x4E49EC4: create_thread (createthread.c:100)
==2176==    by 0x4E49EC4: pthread_create@@GLIBC_2.2.5 (pthread_create.c:797)
==2176==    by 0x4C36A27: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-lin
ux.so)
==2176==    by 0x108C54: Pthread_create (in /home/dasol/main-race)
==2176==    by 0x108D26: main (in /home/dasol/main-race)
==2176==
==2176== ----------------------------------------------------------------
==2176==
==2176== Possible data race during read of size 4 at 0x30A014 by thread #1
==2176== Locks held: none
==2176==    at 0x108D27: main (in /home/dasol/main-race)
==2176==
==2176== This conflicts with a previous write of size 4 by thread #2
==2176== Locks held: none
==2176==    at 0x108CDF: worker (in /home/dasol/main-race)
==2176==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-lin
ux.so)
==2176==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2176==    by 0x518288E: clone (clone.S:95)
==2176==  Address 0x30a014 is 0 bytes inside data symbol "balance"
==2176==
```

```
==2176==
==2176== Possible data race during write of size 4 at 0x30A014 by thread #1
==2176== Locks held: none
==2176==    at 0x108D30: main (in /home/dasol/main-race)
==2176==
==2176== This conflicts with a previous write of size 4 by thread #2
==2176== Locks held: none
==2176==    at 0x108CDF: worker (in /home/dasol/main-race)
==2176==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-lin
ux.so)
==2176==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2176==    by 0x518288E: clone (clone.S:95)
==2176==  Address 0x30a014 is 0 bytes inside data symbol "balance"
==2176==
==2176==
==2176== For counts of detected and suppressed errors, rerun with: -v
==2176== Use --history-level=approx or =none to gain increased speed, at
==2176== the cost of reduced accuracy of conflicting-access information
==2176== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
dasol@dasol-VirtualBox:~$
```

Helgrind에 따르면 메인 코드 스레드 1에서 balance 데이터를 읽으려고 하는 동안 work 코드 스레드 2에서 데이터를 수정하려고 하는 충돌이 발생하였다.

역시 메인 코드 스레드 1에서 balance 데이터를 수정하려고 하는 동안 work 코드 스레드 2에서 데이터를 수정하려고 하는 충돌이 발생하였다.

Balance 데이터에 스레드 오류가 발생했음을 알려준다.

2. What happens when you remove one of the offending lines of code? Now add a lock around one of the updates to the shared variable, and then around both. What does helgrind report in each of these cases?

```c
#include <stdio.h>

#include "mythreads.h"

pthread_mutex_t mutex_lock;
int balance = 0;

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

int main(int argc, char *argv[]) {

    pthread_t p;

    pthread_mutex_init(&mutex_lock, NULL);
    Pthread_create(&p, NULL, worker, NULL);
    Pthread_mutex_lock(&mutex_lock);
    balance++; // unprotected access
    Pthread_mutex_unlock(&mutex_lock);
    Pthread_join(p, NULL);
    return 0;
}
```

Mutex_lock을 설정하여 잠금을 만들어주었다. 이제 balance 변수에 접근하기 위해서는 lock을 가지고 있어야만 한다. 이러한 방법으로 변수에 한 번에 하나의 스레드만 접근하도록 함으로써 스레드의 동시성 문제를 해결할 수 있다.

한 번 helgrind를 사용해 코드를 살펴보자.

```
asol@dasol-VirtualBox:~$ gcc -o main-race1 main-race1.c -lpthread
asol@dasol-VirtualBox:~$ valgrind --tool=helgrind ./main-race1
==2397== Helgrind, a thread error detector
==2397== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==2397== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2397== Command: ./main-race1
==2397==
==2397==
==2397== For counts of detected and suppressed errors, rerun with: -v
==2397== Use --history-level=approx or =none to gain increased speed, at
==2397== the cost of reduced accuracy of conflicting-access information
==2397== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 7 from 7)
asol@dasol-VirtualBox:~$
```

오류가 발생하지 않은 것을 확인할 수 있다. 변수에 충돌된 접근이 없다고 한다.

3. Now let's look at main-deadlock.c. Examine the code. This code has a problem known as deadlock (which we discuss in much more depth in a forthcoming chapter). Can you see what problem it might have?

Main-deadlock는 두 개의 스레드를 사용하는 프로그램이다. 이 두 개의 스레드를 사용하는 도중 deadrock 문제가 발생한 것을 볼 수 있다.

```c
#include <stdio.h>

#include "mythreads.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;
}
```

Deadlock이란 한 스레드가 lock을 획득하기 위해 기다리는데, 이 lock을 잡고 있는 스레드도 똑같이 다른 lock을 획득하기 위해 기다리고 있는 상태를 말한다. 코드를 보면 두 개의 스레드가 unlock하지 않은 채로 다른 lock을 획득하려는 상태가 발생할 가능성이 있다. 이러한 경우 unlock이 되지 않았기에 교착상태에 들어가게 된다.

4. Now run helgrind on this code. What does helgrind report?

한 번 helgrind를 사용해 코드를 살펴보자.

```
==2443== ---Thread-Announcement------------------------------------------
==2443==
==2443== Thread #3 was created
==2443==    at 0x518287E: clone (clone.S:71)
==2443==    by 0x4E49EC4: create_thread (createthread.c:100)
==2443==    by 0x4E49EC4: pthread_create@@GLIBC_2.2.5 (pthread_create.c:797)
==2443==    by 0x4C36A27: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-l
ux.so)
==2443==    by 0x108C54: Pthread_create (in /home/dasol/main-deadlock)
==2443==    by 0x108D89: main (in /home/dasol/main-deadlock)
==2443==
==2443== ----------------------------------------------------------------
==2443==
==2443== Thread #3: lock order "0x30A040 before 0x30A080" violated
==2443==
==2443== Observed (incorrect) order is: acquisition of lock at 0x30A080
==2443==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-l
ux.so)
==2443==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock)
==2443==    by 0x108D06: worker (in /home/dasol/main-deadlock)
==2443==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-l
ux.so)
==2443==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2443==    by 0x518288E: clone (clone.S:95)
==2443==
==2443==  followed by a later acquisition of lock at 0x30A040
==2443==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-l
ux.so)
==2443==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock)
==2443==    by 0x108D12: worker (in /home/dasol/main-deadlock)
==2443==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-l
ux.so)
==2443==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2443==    by 0x518288E: clone (clone.S:95)
==2443==
==2443== Required order was established by acquisition of lock at 0x30A040
==2443==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-l
ux.so)
==2443==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock)
==2443==    by 0x108CEC: worker (in /home/dasol/main-deadlock)
==2443==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-l
ux.so)
==2443==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2443==    by 0x518288E: clone (clone.S:95)
==2443==  followed by a later acquisition of lock at 0x30A080
==2443==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-lin
ux.so)
==2443==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock)
==2443==    by 0x108CF8: worker (in /home/dasol/main-deadlock)
==2443==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-lin
ux.so)
==2443==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2443==    by 0x518288E: clone (clone.S:95)
==2443==
==2443==  Lock at 0x30A040 was first observed
==2443==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-lin
ux.so)
==2443==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock)
==2443==    by 0x108CEC: worker (in /home/dasol/main-deadlock)
==2443==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-lin
ux.so)
==2443==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2443==    by 0x518288E: clone (clone.S:95)
==2443==  Address 0x30a040 is 0 bytes inside data symbol "m1"
==2443==
==2443==  Lock at 0x30A080 was first observed
==2443==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-lin
ux.so)
==2443==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock)
==2443==    by 0x108CF8: worker (in /home/dasol/main-deadlock)
==2443==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-lin
ux.so)
==2443==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2443==    by 0x518288E: clone (clone.S:95)
==2443==  Address 0x30a080 is 0 bytes inside data symbol "m2"
==2443==
==2443==
==2443==
==2443== For counts of detected and suppressed errors, rerun with: -v
==2443== Use --history-level=approx or =none to gain increased speed, at
==2443== the cost of reduced accuracy of conflicting-access information
==2443== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 7 from 7)
```

3개의 스레드가 만들어지고 m1(0x30a040), m2(0x30a080)의 2개의 lock이 존재한다.

먼저, 스레드 1은 m1를 얻고 m2을 얻고자 요청을 보낸다. 그러나 스레드 2는 m2를 얻고 m1을 얻고자 요청을 보낸다. 즉, 두 스레드는 lock을 unlock하지 않은 채로 다른 lock을 원하고 있는 것이다. 이러한 경우 교착상태에 빠지며 deadlock 문제가 발생한다.

5. Now run helgrind on main-deadlock-global.c. Examine the code; does it have the same problem that main-deadlock.c has? Should helgrind be reporting the same error? What does this tell you about tools like helgrind?

main-deadlock-global에서는 g라는 lock을 하나 더 설정하여 한 번의 하나의 스레드만이 m1과 m2를 얻는 코드에 접근할 수 있도록 했다. 즉, lock을 요구하기 전에 자신이 가지고 있는 lock을 반드시 unlock하도록 설정한 것이다.

```
dasol@dasol-VirtualBox:~$ valgrind --tool=helgrind ./main-deadlock-global
==2608== Helgrind, a thread error detector
==2608== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==2608== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2608== Command: ./main-deadlock-global
==2608==
==2608== ---Thread-Announcement------------------------------------------
==2608==
==2608== Thread #3 was created
==2608==    at 0x518287E: clone (clone.S:71)
==2608==    by 0x4E49EC4: create_thread (createthread.c:100)
==2608==    by 0x4E49EC4: pthread_create@@GLIBC_2.2.5 (pthread_create.c:797)
==2608==    by 0x4C36A27: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x108C54: Pthread_create (in /home/dasol/main-deadlock-global)
==2608==    by 0x108DA1: main (in /home/dasol/main-deadlock-global)
==2608==
==2608== ----------------------------------------------------------------
==2608==
==2608== Thread #3: lock order "0x30A080 before 0x30A0C0" violated
==2608==
==2608== Observed (incorrect) order is: acquisition of lock at 0x30A0C0
==2608==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock-global)
==2608==    by 0x108D12: worker (in /home/dasol/main-deadlock-global)
==2608==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2608==    by 0x518288E: clone (clone.S:95)
==2608==
==2608==  followed by a later acquisition of lock at 0x30A080
==2608==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock-global)
==2608==    by 0x108D1E: worker (in /home/dasol/main-deadlock-global)
==2608==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2608==    by 0x518288E: clone (clone.S:95)
==2608==
==2608== Required order was established by acquisition of lock at 0x30A080
==2608==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock-global)
==2608==    by 0x108CF8: worker (in /home/dasol/main-deadlock-global)
==2608==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2608==    by 0x518288E: clone (clone.S:95)
```

```
==2608==
==2608==  followed by a later acquisition of lock at 0x30A0C0
==2608==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock-global)
==2608==    by 0x108D04: worker (in /home/dasol/main-deadlock-global)
==2608==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2608==    by 0x518288E: clone (clone.S:95)
==2608==
==2608==  Lock at 0x30A080 was first observed
==2608==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock-global)
==2608==    by 0x108CF8: worker (in /home/dasol/main-deadlock-global)
==2608==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2608==    by 0x518288E: clone (clone.S:95)
==2608==  Address 0x30a080 is 0 bytes inside data symbol "m1"
==2608==
==2608==  Lock at 0x30A0C0 was first observed
==2608==    at 0x4C3403C: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x108AD7: Pthread_mutex_lock (in /home/dasol/main-deadlock-global)
==2608==    by 0x108D04: worker (in /home/dasol/main-deadlock-global)
==2608==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==2608==    by 0x4E496DA: start_thread (pthread_create.c:463)
==2608==    by 0x518288E: clone (clone.S:95)
==2608==  Address 0x30a0c0 is 0 bytes inside data symbol "m2"
==2608==
==2608==
==2608==
==2608== For counts of detected and suppressed errors, rerun with: -v
==2608== Use --history-level=approx or =none to gain increased speed, at
==2608== the cost of reduced accuracy of conflicting-access information
==2608== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 7 from 7)
dasol@dasol-VirtualBox:~$
```

그러나 helgrind로 분석한 결과 오류는 똑같이 나타났다. 스레드 1이 m1를 얻고 m2을 얻고자 요청을 보내고, 스레드 2가 m2를 얻고 m1을 얻고자 요청을 보내는 문제가 해결되지 않았기에 그대로 오류가 있는 것으로 보인다. 즉, 잠금 순서에 문제가 있는 것으로 보인다.

6. Let's next look at main-signal.c. This code uses a variable (done) to signal that the child is done and that the parent can now continue. Why is this code inefficient? (what does the parent end up spending its time doing, particularly if the child thread takes a long time to complete?)

```c
#include <stdio.h>

#include "mythreads.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;
}
```

main-signal.c는 부모 스레드가 자식 스레드를 만든 후 자식 스레드가 종료될 때까지 기다리는 형태의 코드이다. Done 변수는 0으로 초기 설정되어 있으며 자식 스레드가 종료될 때 1로 바뀐다. 부모 스레드는 그 동안 while문으로 done이 1로 바뀌었는지 확인한다.

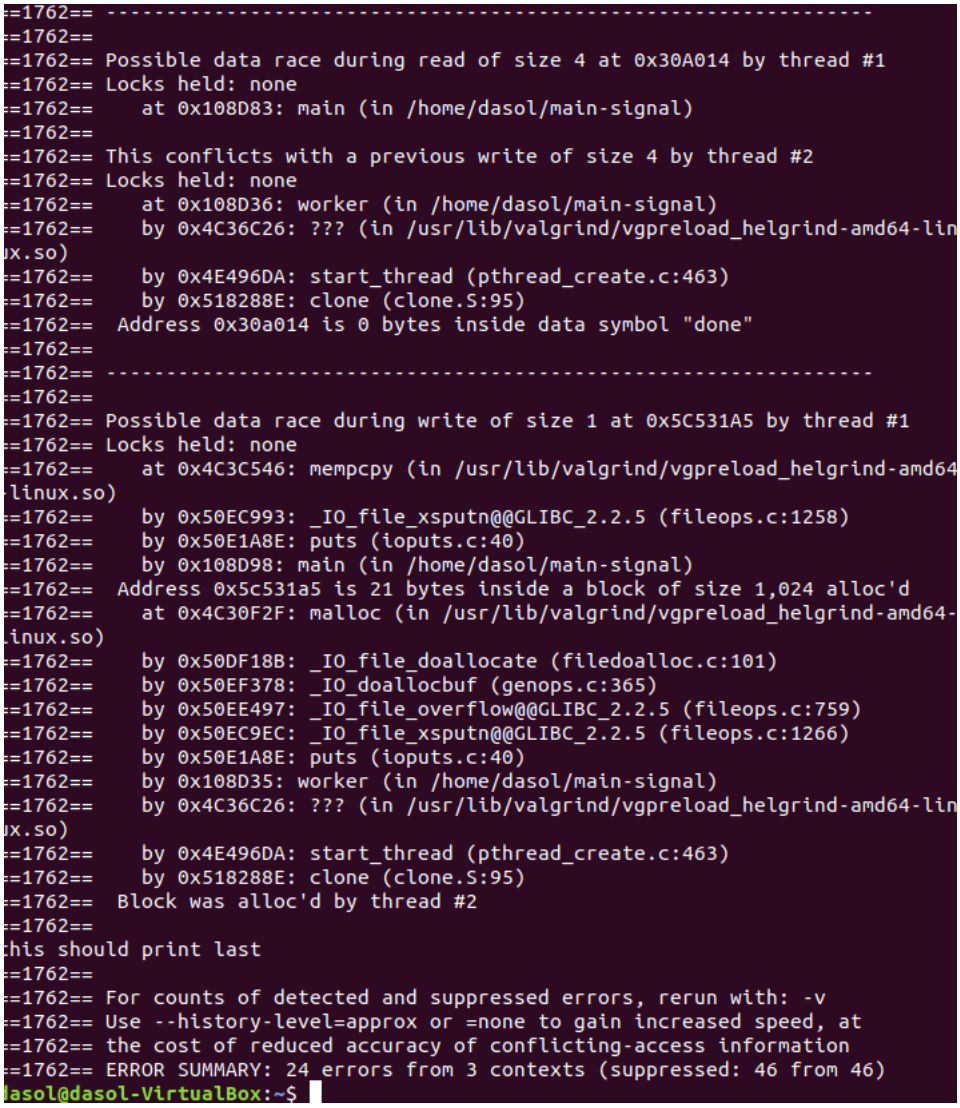
이러한 형태의 코드는 부모 스레드가 자식 스레드가 종료될 때까지 계속 while문을 돌아야 한다. 만약 자식 스레드가 종료되는 데에 많은 시간이 소요된다면 그 동안 부모 스레드는 cpu와 같은 resource를 낭비하게 되는 것이다. 그렇기 때문에 inefficient하다.

7. Now run helgrind on this program. What does it report? Is the code correct?

```
asol@dasol-VirtualBox:~$ valgrind --tool=helgrind ./main-signal
=1762== Helgrind, a thread error detector
=1762== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
=1762== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
=1762== Command: ./main-signal
=1762==
his should print first
=1762== ---Thread-Announcement------------------------------------------
=1762==
=1762== Thread #2 was created
=1762==    at 0x518287E: clone (clone.S:71)
=1762==    by 0x4E49EC4: create_thread (createthread.c:100)
=1762==    by 0x4E49EC4: pthread_create@@GLIBC_2.2.5 (pthread_create.c:797)
=1762==    by 0x4C36A27: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-lin
x.so)
=1762==    by 0x108CA4: Pthread_create (in /home/dasol/main-signal)
=1762==    by 0x108D81: main (in /home/dasol/main-signal)
=1762==
=1762== ---Thread-Announcement------------------------------------------
=1762==
=1762== Thread #1 is the program's root thread
=1762==
=1762== ----------------------------------------------------------------
=1762==
=1762== Possible data race during write of size 4 at 0x30A014 by thread #2
=1762== Locks held: none
=1762==    at 0x108D36: worker (in /home/dasol/main-signal)
=1762==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-lin
x.so)
=1762==    by 0x4E496DA: start_thread (pthread_create.c:463)
=1762==    by 0x518288E: clone (clone.S:95)
=1762==
=1762== This conflicts with a previous read of size 4 by thread #1
=1762== Locks held: none
=1762==    at 0x108D83: main (in /home/dasol/main-signal)
=1762==  Address 0x30a014 is 0 bytes inside data symbol "done"
=1762==
=1762== ----------------------------------------------------------------
```

스레드 1은 부모 스레드이며 스레드 2는 자식 스레드이다.

스레드 2가 done을 바꾸려고 하는데 스레드 1이 같은 변수를 읽으며 충돌이 발생하였다.



스레드 1이 done을 읽으려고 하는데 스레드 2이 같은 변수를 수정하며 충돌이 발생하였다.

문제1과 마찬가지로 스레드의 동시성 문제가 발생했음을 알 수 있다.

즉. 스레드 1은 자원을 낭비하고 있을 뿐만 아니라, 스레드 1이 while문을 도는 동안 계속해서 동시성 문제가 발생하고 있는 것이다.

8. Now look at a slightly modified version of the code, which is found in main-signal-cv.c. This version uses a condition variable to do the signaling (and associated lock). Why is this code preferred to the previous version? Is it correctness, or performance, or both?

main-signal-cv는 condition variable와 associated lock를 사용하여 main-signal의 문제점을 해결한 코드이다.

```c
#include <stdio.h>

#include "mythreads.h"

//
// simple synchronizer: allows one thread to wait for another
// structure "synchronizer_t" has all the needed data
// methods are:
//   init (called by one thread)
//   wait (to wait for a thread)
//   done (to indicate thread is done)
//
typedef struct __synchronizer_t {
    pthread_mutex_t lock;
    pthread_cond_t cond;
    int done;
} synchronizer_t;

synchronizer_t s;

void signal_init(synchronizer_t *s) {
    Pthread_mutex_init(&s->lock, NULL);
    Pthread_cond_init(&s->cond, NULL);
    s->done = 0;
}

void signal_done(synchronizer_t *s) {
    Pthread_mutex_lock(&s->lock);
    s->done = 1;
    Pthread_cond_signal(&s->cond);
    Pthread_mutex_unlock(&s->lock);
}

void signal_wait(synchronizer_t *s) {
    Pthread_mutex_lock(&s->lock);
    while (s->done == 0)
   Pthread_cond_wait(&s->cond, &s->lock);
    Pthread_mutex_unlock(&s->lock);
}

void* worker(void* arg) {
    printf("this should print first\n");
    signal_done(&s);
    return NULL;
}
"main-signal-cv.c" 55L, 1205C                                 1,1           꼭대기
```

main-signal-cv에서는 condition variable을 사용한다. 자식 스레드가 만들어지면 부모 스레드는 signal wait로 가서 lock을 획득한다. 그 후 condition variable를 통해서 자식 스레드가 종료될 때까지 wait 상태가 된다. 만약 자식 스레드가 종료된다면 역시 condition variable를 통해 시그널을 보내 종료되었음을 알려준다. 그러면 condition variable를 통해 그 시그널을 받고 wait 상태에서 벗어나 lock을 unlock한다.

9. Once again run helgrind on main-signal-cv. Does it report any errors?

```
dasol@dasol-VirtualBox:~$ valgrind --tool=helgrind ./main-signal-cv
==2013== Helgrind, a thread error detector
==2013== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==2013== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2013== Command: ./main-signal-cv
==2013==
this should print first
this should print last
==2013==
==2013== For counts of detected and suppressed errors, rerun with: -v
==2013== Use --history-level=approx or =none to gain increased speed, at
==2013== the cost of reduced accuracy of conflicting-access information
==2013== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 16 from 15)
dasol@dasol-VirtualBox:~$
```

Helgrind로 돌려본 결과 아무런 문제가 발생하지 않은 것을 확인할 수 있다.

Lock을 사용해 공통으로 사용하는 condition variable를 안전하게 보호하였기에 스레드의 동시성 문제 또한 해결된 것이다.

즉, condition variable를 사용하여 기존 main-signal의 두 가지 문제, 자원 낭비와 스레드의 동시성 문제를 condition variable와 associated lock을 사용하여 해결한 것을 볼 수 있다.