[REPORT]

■과 목 명: 운영체제 (SWE3004-42)

■담 당 교 수: 신동군 교수님

■제 출 일: 2022년 4월 15일

■학　　과: 수학과

■학　　번: 2016314786

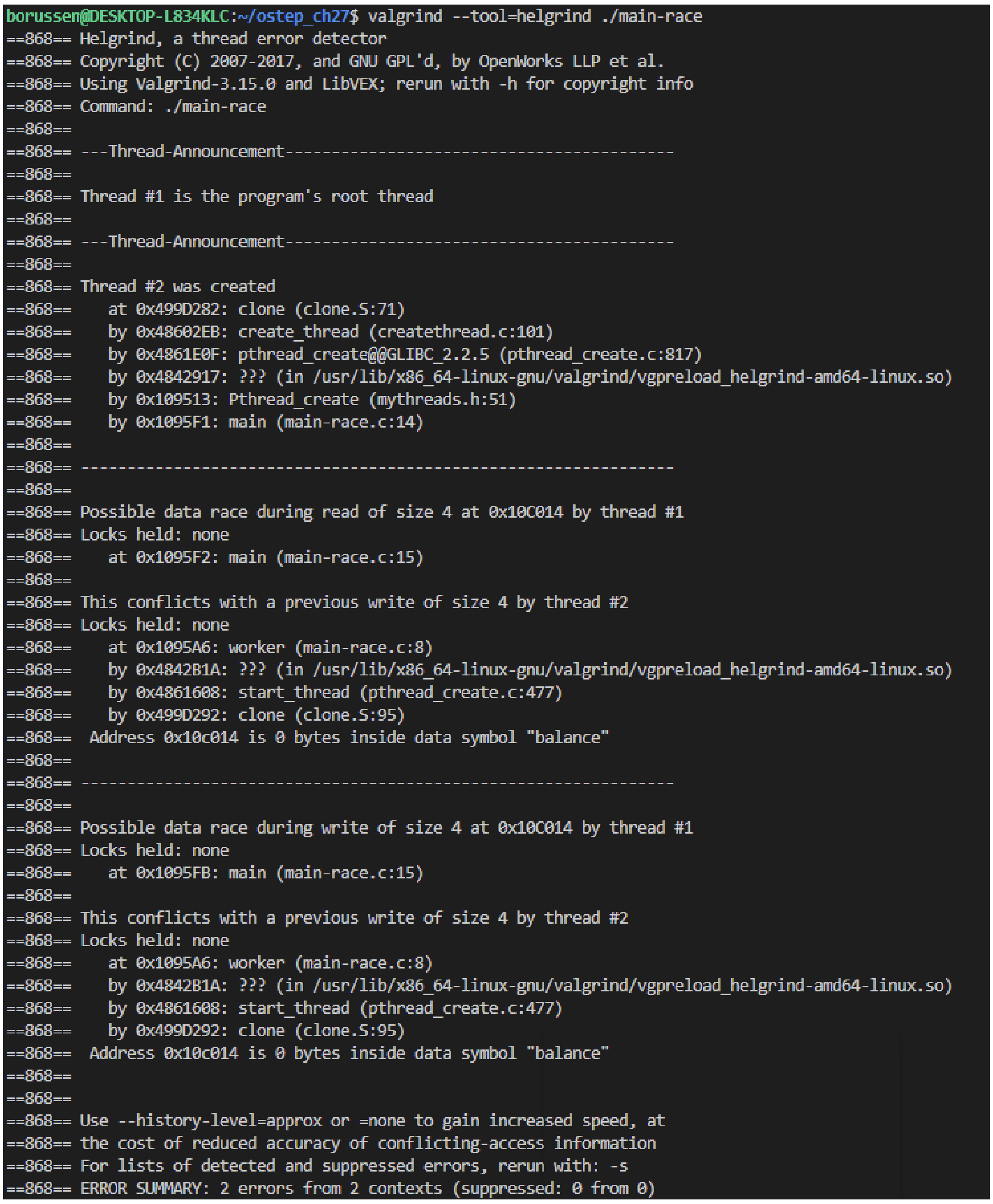
■성　　명: 김호진

Chap 27. Debugging Race/Deadlock w/ *helgrind*

수학과 김호진 (2016314786)

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?**

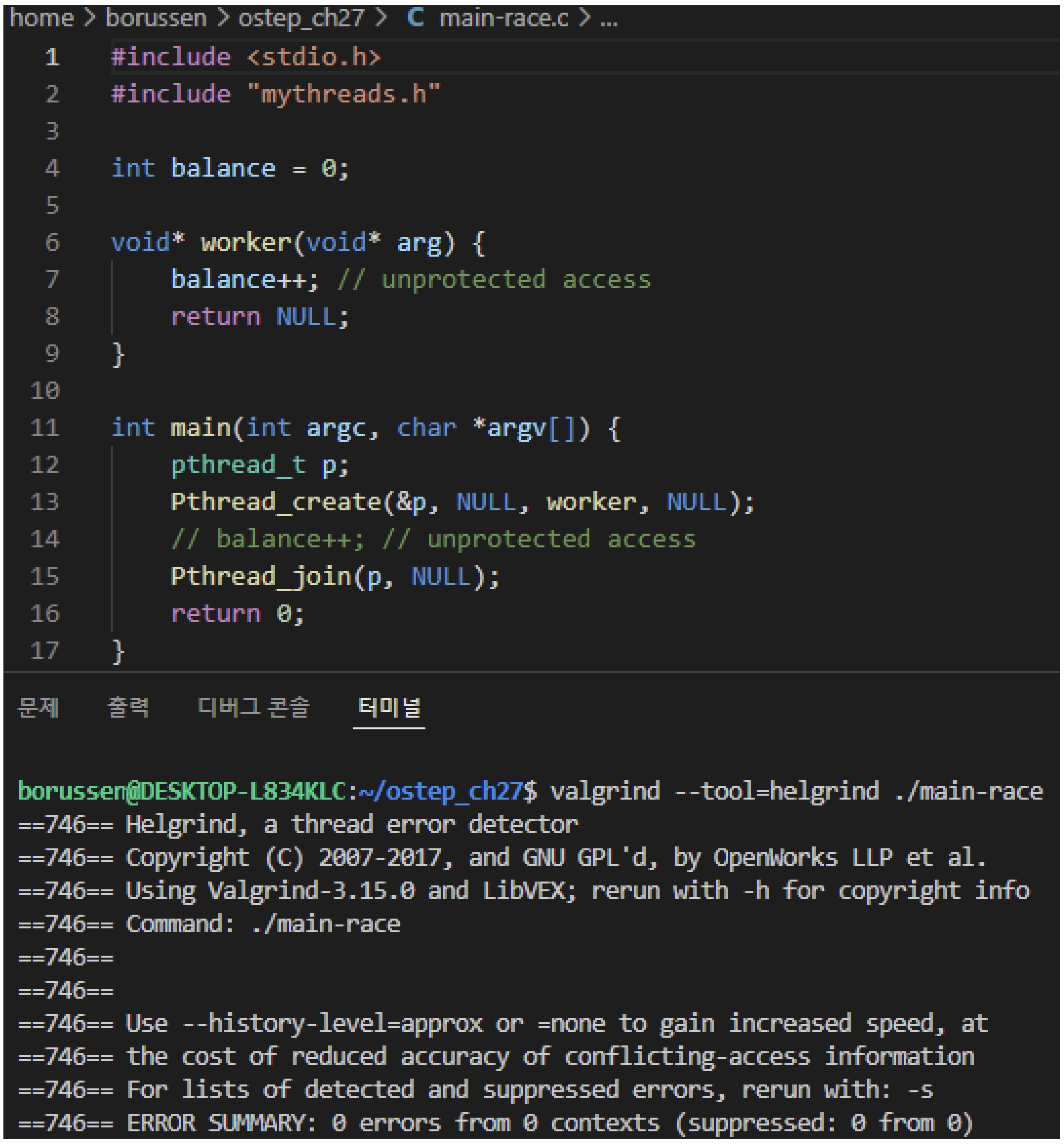
helgrind는 올바른 코드 라인을 가리키고 있으며, 스레드가 생성된 위치를 제공한다. 추가적으로, main-race.c의 data race 과정 중 line 8과 line 15에서 발생한 error에 대한 message를 보고한다.



```
borussen@DESKTOP-L834KLC:~/ostep_ch27$ valgrind --tool=helgrind ./main-race
==868== Helgrind, a thread error detector
==868== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==868== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==868== Command: ./main-race
==868==
==868== ---Thread-Announcement------------------------------------------
==868==
==868== Thread #1 is the program's root thread
==868==
==868== ---Thread-Announcement------------------------------------------
==868==
==868== Thread #2 was created
==868==    at 0x499D282: clone (clone.S:71)
==868==    by 0x48602EB: create_thread (createthread.c:101)
==868==    by 0x4861E0F: pthread_create@@GLIBC_2.2.5 (pthread_create.c:817)
==868==    by 0x4842917: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==868==    by 0x109513: Pthread_create (mythreads.h:51)
==868==    by 0x1095F1: main (main-race.c:14)
==868==
==868== ----------------------------------------------------------------
==868==
==868== Possible data race during read of size 4 at 0x10C014 by thread #1
==868== Locks held: none
==868==    at 0x1095F2: main (main-race.c:15)
==868==
==868== This conflicts with a previous write of size 4 by thread #2
==868== Locks held: none
==868==    at 0x1095A6: worker (main-race.c:8)
==868==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==868==    by 0x4861608: start_thread (pthread_create.c:477)
==868==    by 0x499D292: clone (clone.S:95)
==868==  Address 0x10c014 is 0 bytes inside data symbol "balance"
==868==
==868== ----------------------------------------------------------------
==868==
==868== Possible data race during write of size 4 at 0x10C014 by thread #1
==868== Locks held: none
==868==    at 0x1095FB: main (main-race.c:15)
==868==
==868== This conflicts with a previous write of size 4 by thread #2
==868== Locks held: none
==868==    at 0x1095A6: worker (main-race.c:8)
==868==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==868==    by 0x4861608: start_thread (pthread_create.c:477)
==868==    by 0x499D292: clone (clone.S:95)
==868==  Address 0x10c014 is 0 bytes inside data symbol "balance"
==868==
==868==
==868== Use --history-level=approx or =none to gain increased speed, at
==868== the cost of reduced accuracy of conflicting-access information
==868== For lists of detected and suppressed errors, rerun with: -s
==868== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

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?**

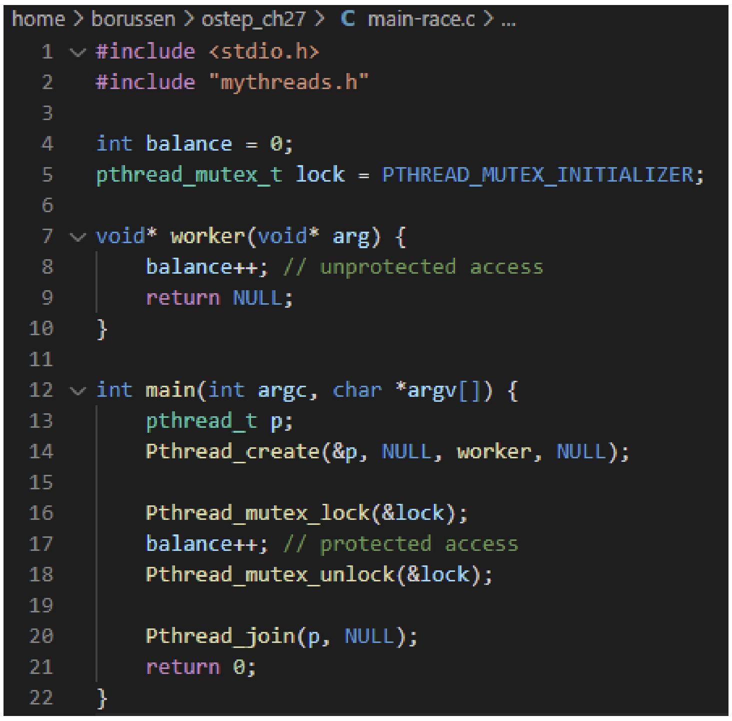
(1) 문제가 되는 코드의 line 중 하나를 제거하자 이전에 탐지되었던 error가 보고되지 않는다.


home > borussen > ostep_ch27 > C main-race.c > ...

```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 access
    Pthread_join(p, NULL);
    return 0;
}
```

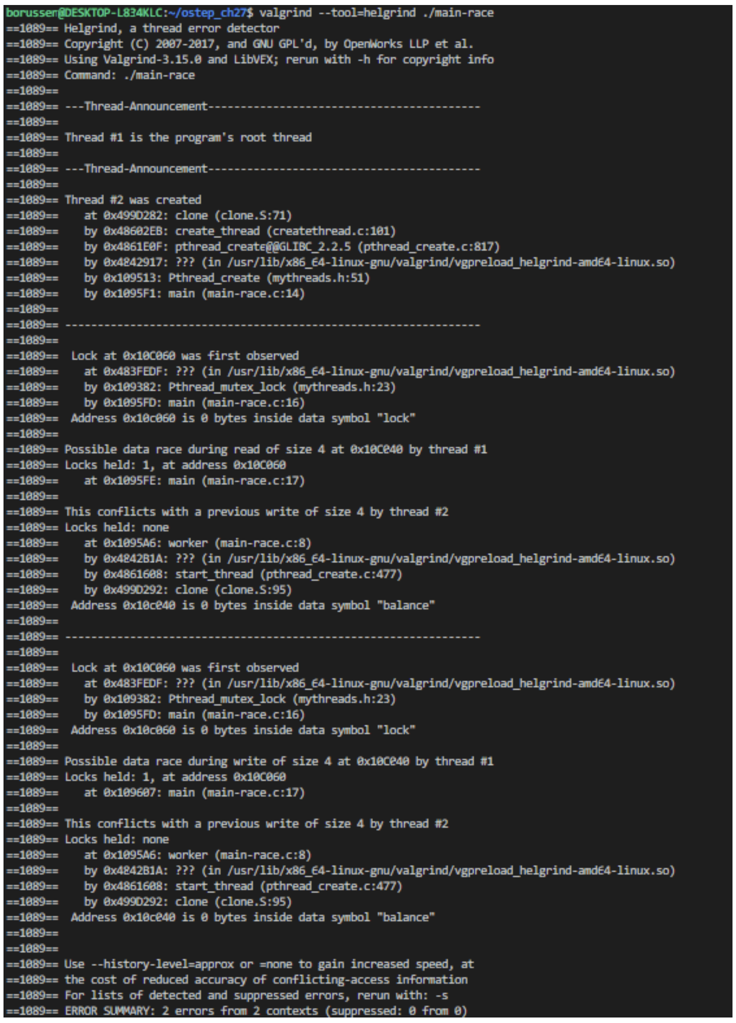
```
borussen@DESKTOP-L834KLC:~/ostep_ch27$ valgrind --tool=helgrind ./main-race
==746== Helgrind, a thread error detector
==746== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==746== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==746== Command: ./main-race
==746==
==746==
==746== Use --history-level=approx or =none to gain increased speed, at
==746== the cost of reduced accuracy of conflicting-access information
==746== For lists of detected and suppressed errors, rerun with: -s
==746== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

(2) shared variable의 update 작업 중 하나에만 lock을 추가하였을 때는 error를 보고한다.



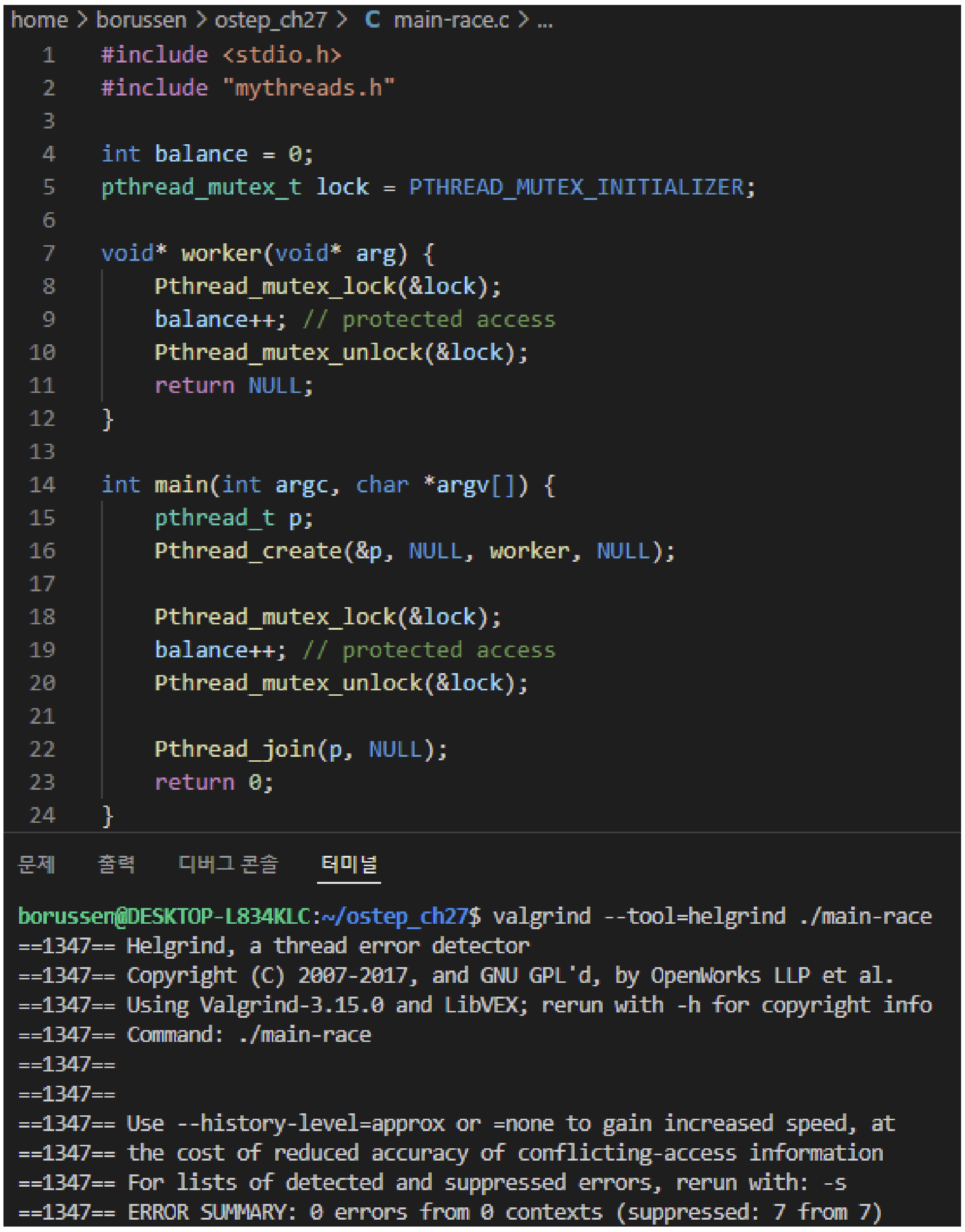
home > borussen > ostep_ch27 > C main-race.c > ...

```c
#include <stdio.h>
#include "mythreads.h"

int balance = 0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

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

int main(int argc, char *argv[]) {
    pthread_t p;
    Pthread_create(&p, NULL, worker, NULL);

    Pthread_mutex_lock(&lock);
    balance++; // protected access
    Pthread_mutex_unlock(&lock);

    Pthread_join(p, NULL);
    return 0;
}
```



```
borussen@DESKTOP-L834KLC:~/ostep_ch27$ valgrind --tool=helgrind ./main-race
==1089== Helgrind, a thread error detector
==1089== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==1089== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==1089== Command: ./main-race
==1089==
==1089== ---Thread-Announcement------------------------------------------
==1089==
==1089== Thread #1 is the program's root thread
==1089==
==1089== ---Thread-Announcement------------------------------------------
==1089==
==1089== Thread #2 was created
==1089==    at 0x499D282: clone (clone.S:71)
==1089==    by 0x48602EB: create_thread (createthread.c:101)
==1089==    by 0x4861E0F: pthread_create@@GLIBC_2.2.5 (pthread_create.c:817)
==1089==    by 0x4842917: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1089==    by 0x109513: Pthread_create (mythreads.h:51)
==1089==    by 0x1095F1: main (main-race.c:14)
==1089==
==1089== ----------------------------------------------------------------
==1089==
==1089==  Lock at 0x10C060 was first observed
==1089==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1089==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1089==    by 0x1095FD: main (main-race.c:16)
==1089==  Address 0x10c060 is 0 bytes inside data symbol "lock"
==1089==
==1089== Possible data race during read of size 4 at 0x10C040 by thread #1
==1089== Locks held: 1, at address 0x10C060
==1089==    at 0x1095FE: main (main-race.c:17)
==1089==
==1089== This conflicts with a previous write of size 4 by thread #2
==1089== Locks held: none
==1089==    at 0x1095A6: worker (main-race.c:8)
==1089==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1089==    by 0x4861608: start_thread (pthread_create.c:477)
==1089==    by 0x499D292: clone (clone.S:95)
==1089==  Address 0x10c040 is 0 bytes inside data symbol "balance"
==1089==
==1089== ----------------------------------------------------------------
==1089==
==1089==  Lock at 0x10C060 was first observed
==1089==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1089==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1089==    by 0x1095FD: main (main-race.c:16)
==1089==  Address 0x10c060 is 0 bytes inside data symbol "lock"
==1089==
==1089== Possible data race during write of size 4 at 0x10C040 by thread #1
==1089== Locks held: 1, at address 0x10C060
==1089==    at 0x109607: main (main-race.c:17)
==1089==
==1089== This conflicts with a previous write of size 4 by thread #2
==1089== Locks held: none
==1089==    at 0x1095A6: worker (main-race.c:8)
==1089==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1089==    by 0x4861608: start_thread (pthread_create.c:477)
==1089==    by 0x499D292: clone (clone.S:95)
==1089==  Address 0x10c040 is 0 bytes inside data symbol "balance"
==1089==
==1089==
==1089== Use --history-level=approx or =none to gain increased speed, at
==1089== the cost of reduced accuracy of conflicting-access information
==1089== For lists of detected and suppressed errors, rerun with: -s
==1089== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

(3)두 작업 모두 lock을 추가하면 error를 보고하지 않는다.


home > borussen > ostep_ch27 > C main-race.c > ...

```c
#include <stdio.h>
#include "mythreads.h"

int balance = 0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

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

int main(int argc, char *argv[]) {
    pthread_t p;
    Pthread_create(&p, NULL, worker, NULL);

    Pthread_mutex_lock(&lock);
    balance++; // protected access
    Pthread_mutex_unlock(&lock);

    Pthread_join(p, NULL);
    return 0;
}
```

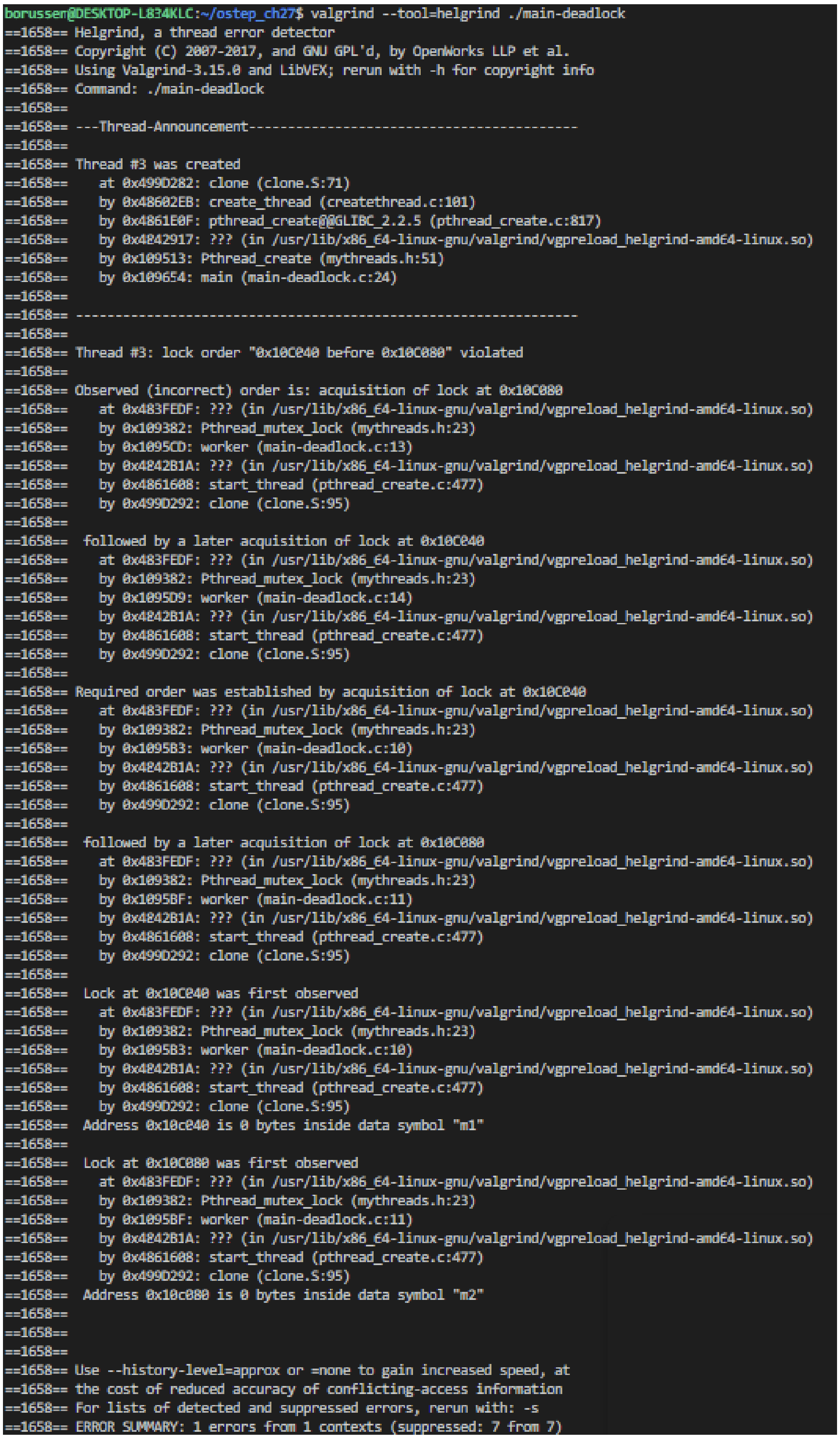
```
borussen@DESKTOP-L834KLC:~/ostep_ch27$ valgrind --tool=helgrind ./main-race
==1347== Helgrind, a thread error detector
==1347== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==1347== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==1347== Command: ./main-race
==1347==
==1347==
==1347== Use --history-level=approx or =none to gain increased speed, at
==1347== the cost of reduced accuracy of conflicting-access information
==1347== For lists of detected and suppressed errors, rerun with: -s
==1347== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 7 from 7)
```

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-deadlock.c에서 두 개의 Thread가 동시에 실행되기 시작하면 문제가 발생한다. Thread 1이 m1에서 Lock이 되고, Thread가 m2에서 Lock이 되는 경우(혹은 Thread 1이 m2에서 Lock이 되고, Thread 2가 m1에서 Lock이 되는 경우), 각 Thread는 상대방의 Lock이 풀리기를 기다리게 되면서 교착 상태에 빠지게 되고, 그 결과 deadlock이 발생한다.

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

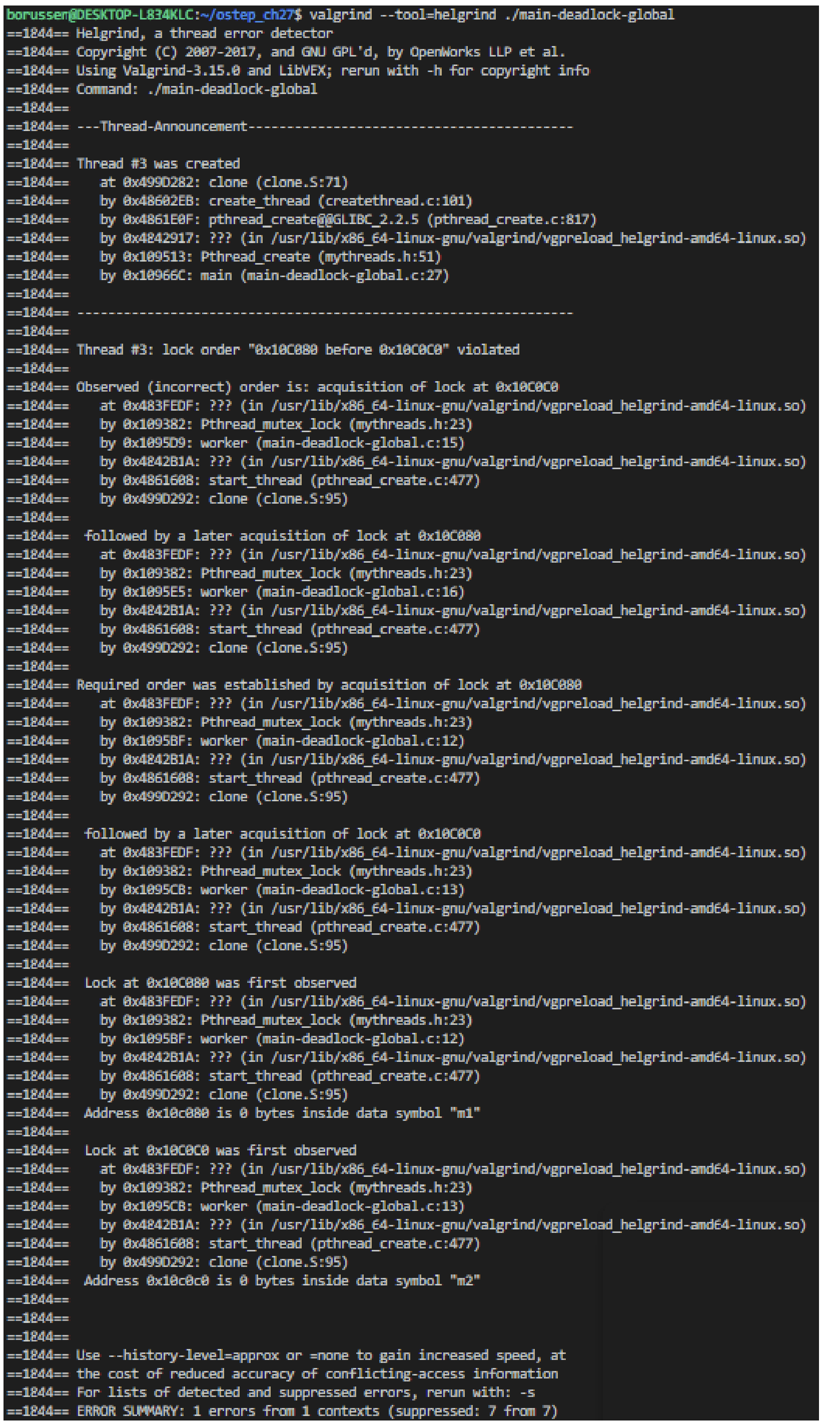
helgrind는 main-deadlock.c에서 lock order가 violated 되었다고 보고한다.



```
borussen@DESKTOP-L834KLC:~/ostep_ch27$ valgrind --tool=helgrind ./main-deadlock
==1658== Helgrind, a thread error detector
==1658== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==1658== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==1658== Command: ./main-deadlock
==1658==
==1658== ---Thread-Announcement------------------------------------------
==1658==
==1658== Thread #3 was created
==1658==    at 0x499D282: clone (clone.S:71)
==1658==    by 0x48602EB: create_thread (createthread.c:101)
==1658==    by 0x4861E0F: pthread_create@@GLIBC_2.2.5 (pthread_create.c:817)
==1658==    by 0x4842917: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x109513: Pthread_create (mythreads.h:51)
==1658==    by 0x109654: main (main-deadlock.c:24)
==1658==
==1658== ----------------------------------------------------------------
==1658==
==1658== Thread #3: lock order "0x10C040 before 0x10C080" violated
==1658==
==1658== Observed (incorrect) order is: acquisition of lock at 0x10C080
==1658==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1658==    by 0x1095CD: worker (main-deadlock.c:13)
==1658==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x4861608: start_thread (pthread_create.c:477)
==1658==    by 0x499D292: clone (clone.S:95)
==1658==
==1658==  followed by a later acquisition of lock at 0x10C040
==1658==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1658==    by 0x1095D9: worker (main-deadlock.c:14)
==1658==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x4861608: start_thread (pthread_create.c:477)
==1658==    by 0x499D292: clone (clone.S:95)
==1658==
==1658== Required order was established by acquisition of lock at 0x10C040
==1658==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1658==    by 0x1095B3: worker (main-deadlock.c:10)
==1658==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x4861608: start_thread (pthread_create.c:477)
==1658==    by 0x499D292: clone (clone.S:95)
==1658==
==1658==  followed by a later acquisition of lock at 0x10C080
==1658==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1658==    by 0x1095BF: worker (main-deadlock.c:11)
==1658==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x4861608: start_thread (pthread_create.c:477)
==1658==    by 0x499D292: clone (clone.S:95)
==1658==
==1658==  Lock at 0x10C040 was first observed
==1658==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1658==    by 0x1095B3: worker (main-deadlock.c:10)
==1658==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x4861608: start_thread (pthread_create.c:477)
==1658==    by 0x499D292: clone (clone.S:95)
==1658==  Address 0x10c040 is 0 bytes inside data symbol "m1"
==1658==
==1658==  Lock at 0x10C080 was first observed
==1658==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1658==    by 0x1095BF: worker (main-deadlock.c:11)
==1658==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1658==    by 0x4861608: start_thread (pthread_create.c:477)
==1658==    by 0x499D292: clone (clone.S:95)
==1658==  Address 0x10c080 is 0 bytes inside data symbol "m2"
==1658==
==1658==
==1658==
==1658== Use --history-level=approx or =none to gain increased speed, at
==1658== the cost of reduced accuracy of conflicting-access information
==1658== For lists of detected and suppressed errors, rerun with: -s
==1658== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 7 from 7)
```

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-global.c에서는 lock-ordering 문제로 인해 발생하는 교착 상태를 방지하기 위해서 global lock 'g'를 추가적으로 사용하면서 main-deadlock.c가 기존에 가지고 있던 문제를 해결한다. 그러나 main-deadlock-global.c에 대해 helgrind를 실행하면 해당 문제가 해결되었음에도 불구하고 여전히 같은 error가 보고된다. 이처럼 helgrind는 특정 anti-pattern에 대한 오류를 가지고 있으며, 그로 인해 몇몇 결과에 대해서 신뢰하기 어렵다.



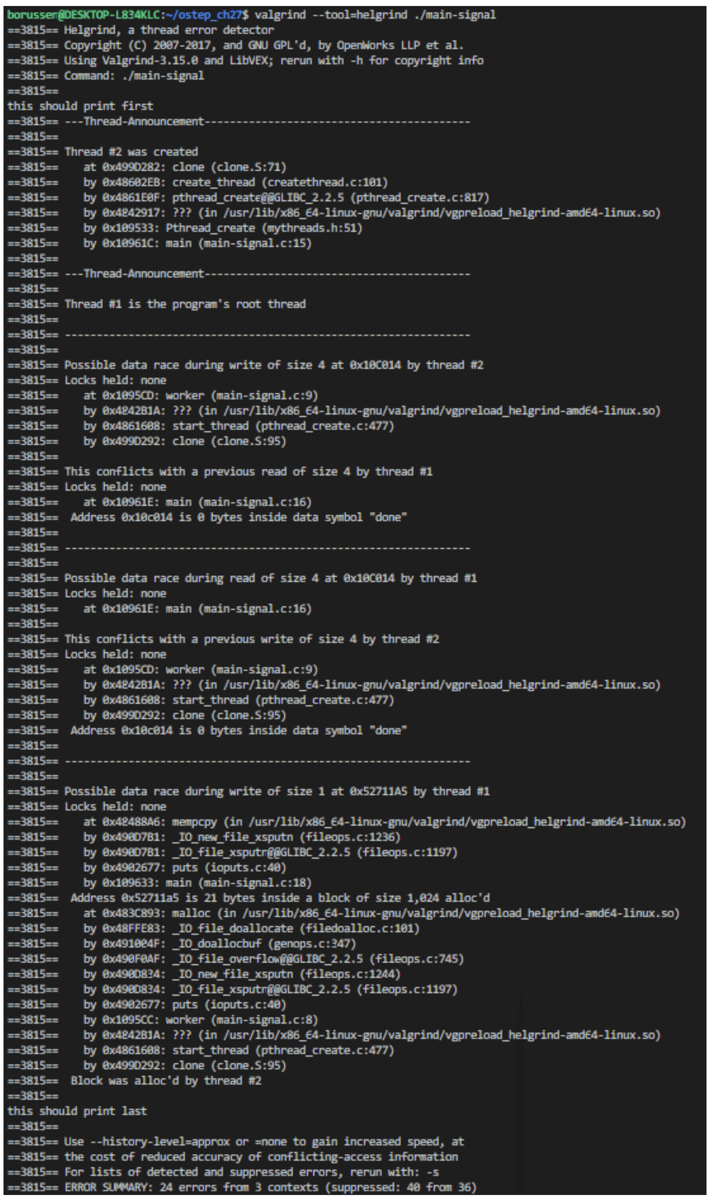
```
borussen@DESKTOP-L834KLC:~/ostep_ch27$ valgrind --tool=helgrind ./main-deadlock-global
==1844== Helgrind, a thread error detector
==1844== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==1844== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==1844== Command: ./main-deadlock-global
==1844==
==1844== ---Thread-Announcement------------------------------------------
==1844==
==1844== Thread #3 was created
==1844==    at 0x499D282: clone (clone.S:71)
==1844==    by 0x48602EB: create_thread (createthread.c:101)
==1844==    by 0x4861E0F: pthread_create@@GLIBC_2.2.5 (pthread_create.c:817)
==1844==    by 0x4842917: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x109513: Pthread_create (mythreads.h:51)
==1844==    by 0x10966C: main (main-deadlock-global.c:27)
==1844==
==1844== ----------------------------------------------------------------
==1844==
==1844== Thread #3: lock order "0x10C080 before 0x10C0C0" violated
==1844==
==1844== Observed (incorrect) order is: acquisition of lock at 0x10C0C0
==1844==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1844==    by 0x1095D9: worker (main-deadlock-global.c:15)
==1844==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x4861608: start_thread (pthread_create.c:477)
==1844==    by 0x499D292: clone (clone.S:95)
==1844==
==1844==  followed by a later acquisition of lock at 0x10C080
==1844==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1844==    by 0x1095E5: worker (main-deadlock-global.c:16)
==1844==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x4861608: start_thread (pthread_create.c:477)
==1844==    by 0x499D292: clone (clone.S:95)
==1844==
==1844== Required order was established by acquisition of lock at 0x10C080
==1844==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1844==    by 0x1095BF: worker (main-deadlock-global.c:12)
==1844==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x4861608: start_thread (pthread_create.c:477)
==1844==    by 0x499D292: clone (clone.S:95)
==1844==
==1844==  followed by a later acquisition of lock at 0x10C0C0
==1844==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1844==    by 0x1095CB: worker (main-deadlock-global.c:13)
==1844==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x4861608: start_thread (pthread_create.c:477)
==1844==    by 0x499D292: clone (clone.S:95)
==1844==
==1844==  Lock at 0x10C080 was first observed
==1844==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1844==    by 0x1095BF: worker (main-deadlock-global.c:12)
==1844==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x4861608: start_thread (pthread_create.c:477)
==1844==    by 0x499D292: clone (clone.S:95)
==1844==  Address 0x10c080 is 0 bytes inside data symbol "m1"
==1844==
==1844==  Lock at 0x10C0C0 was first observed
==1844==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==1844==    by 0x1095CB: worker (main-deadlock-global.c:13)
==1844==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==1844==    by 0x4861608: start_thread (pthread_create.c:477)
==1844==    by 0x499D292: clone (clone.S:95)
==1844==  Address 0x10c0c0 is 0 bytes inside data symbol "m2"
==1844==
==1844==
==1844==
==1844== Use --history-level=approx or =none to gain increased speed, at
==1844== the cost of reduced accuracy of conflicting-access information
==1844== For lists of detected and suppressed errors, rerun with: -s
==1844== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 7 from 7)
```

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?)**

main-signal.c 상에서 child thread가 작업을 완료하는 데 오랜 시간이 걸리는 경우, parent thread는 child thread가 끝나기를 기다리는 데 대부분의 시간을 소비하고 해당 기간동안 CPU resource 역시 소모되기 때문에 코드는 비효율적이다.

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

main-signal.c에 대해 helgrind를 실행하면, done = 1과 print()의 write/read로 인해 발생하는 data race를 보고한다. 해당 코드는 다음과 같이 많은 에러가 보고되기 때문에 올바르지 않다.



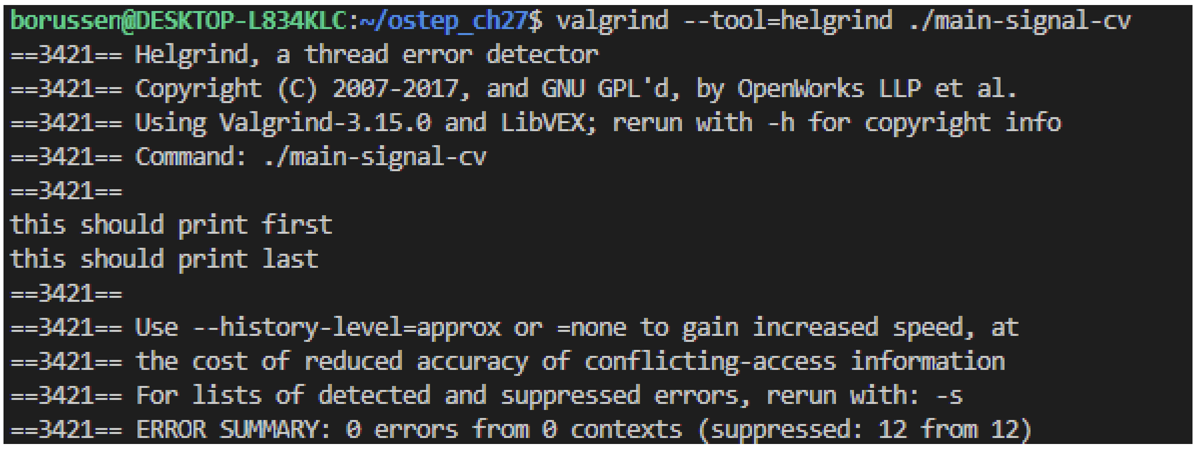
```
borusser@DESKTOP-L834KLC:~/ostep_ch27$ valgrind --tool=helgrind ./main-signal
==3815== Helgrind, a thread error detector
==3815== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==3815== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3815== Command: ./main-signal
==3815==
this should print first
==3815== ---Thread-Announcement------------------------------------------
==3815==
==3815== Thread #2 was created
==3815==    at 0x499D282: clone (clone.S:71)
==3815==    by 0x48602EB: create_thread (createthread.c:101)
==3815==    by 0x4861E0F: pthread_create@@GLIBC_2.2.5 (pthread_create.c:817)
==3815==    by 0x4842917: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==3815==    by 0x109533: Pthread_create (mythreads.h:51)
==3815==    by 0x10961C: main (main-signal.c:15)
==3815==
==3815== ---Thread-Announcement------------------------------------------
==3815==
==3815== Thread #1 is the program's root thread
==3815==
==3815== ----------------------------------------------------------------
==3815==
==3815== Possible data race during write of size 4 at 0x10C014 by thread #2
==3815== Locks held: none
==3815==    at 0x1095CD: worker (main-signal.c:9)
==3815==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==3815==    by 0x4861608: start_thread (pthread_create.c:477)
==3815==    by 0x499D292: clone (clone.S:95)
==3815==
==3815== This conflicts with a previous read of size 4 by thread #1
==3815== Locks held: none
==3815==    at 0x10961E: main (main-signal.c:16)
==3815==  Address 0x10c014 is 0 bytes inside data symbol "done"
==3815==
==3815== ----------------------------------------------------------------
==3815==
==3815== Possible data race during read of size 4 at 0x10C014 by thread #1
==3815== Locks held: none
==3815==    at 0x10961E: main (main-signal.c:16)
==3815==
==3815== This conflicts with a previous write of size 4 by thread #2
==3815== Locks held: none
==3815==    at 0x1095CD: worker (main-signal.c:9)
==3815==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==3815==    by 0x4861608: start_thread (pthread_create.c:477)
==3815==    by 0x499D292: clone (clone.S:95)
==3815==  Address 0x10c014 is 0 bytes inside data symbol "done"
==3815==
==3815== ----------------------------------------------------------------
==3815==
==3815== Possible data race during write of size 1 at 0x52711A5 by thread #1
==3815== Locks held: none
==3815==    at 0x48488A6: mempcpy (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==3815==    by 0x490D7B1: _IO_new_file_xsputn (fileops.c:1236)
==3815==    by 0x490D7B1: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1197)
==3815==    by 0x4902677: puts (ioputs.c:40)
==3815==    by 0x109633: main (main-signal.c:18)
==3815==  Address 0x52711a5 is 21 bytes inside a block of size 1,024 alloc'd
==3815==    at 0x483C893: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==3815==    by 0x48FFE83: _IO_file_doallocate (filedoalloc.c:101)
==3815==    by 0x491024F: _IO_doallocbuf (genops.c:347)
==3815==    by 0x490F0AF: _IO_file_overflow@@GLIBC_2.2.5 (fileops.c:745)
==3815==    by 0x490D834: _IO_new_file_xsputn (fileops.c:1244)
==3815==    by 0x490D834: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1197)
==3815==    by 0x4902677: puts (ioputs.c:40)
==3815==    by 0x1095CC: worker (main-signal.c:8)
==3815==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==3815==    by 0x4861608: start_thread (pthread_create.c:477)
==3815==    by 0x499D292: clone (clone.S:95)
==3815==  Block was alloc'd by thread #2
==3815==
this should print last
==3815==
==3815== Use --history-level=approx or =none to gain increased speed, at
==3815== the cost of reduced accuracy of conflicting-access information
==3815== For lists of detected and suppressed errors, rerun with: -s
==3815== ERROR SUMMARY: 24 errors from 3 contexts (suppressed: 40 from 36)
```

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?**

정확도와 성능 모두 main-signal-cv.c가 main-signal.c보다 선호된다. main-signal-cv.c에서는 waiting thread를 sleep 시키다가, 필요시 signal을 보내 sleep 상태의 thread를 wake-up 시킬 수 있다. 따라서 thread가 불필요하게 CPU를 사용하는 것을 방지할 수 있어 main-signal.c에 비해 더 좋은 성능을 가진다. 또한 main-signal-cv.c에서는 thread를 동기화하기 위해서 locks을 모두 추가하였기 때문에 에러가 발생할 확률이 낮고 기존보다 더 높은 정확도를 보인다.

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

helgrind는 main-signal-cv에 대해 error가 존재하지 않는다고 보고한다.


```
borussen@DESKTOP-L834KLC:~/ostep_ch27$ valgrind --tool=helgrind ./main-signal-cv
==3421== Helgrind, a thread error detector
==3421== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==3421== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3421== Command: ./main-signal-cv
==3421==
this should print first
this should print last
==3421==
==3421== Use --history-level=approx or =none to gain increased speed, at
==3421== the cost of reduced accuracy of conflicting-access information
==3421== For lists of detected and suppressed errors, rerun with: -s
==3421== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 12 from 12)
```