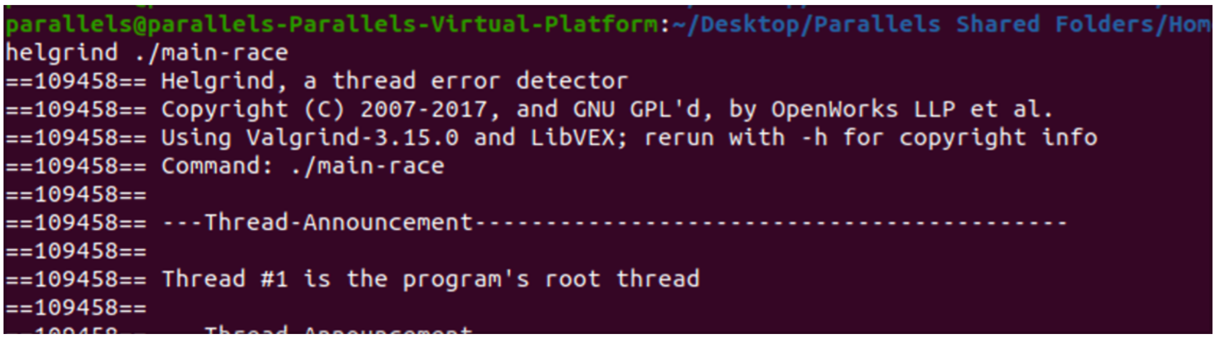
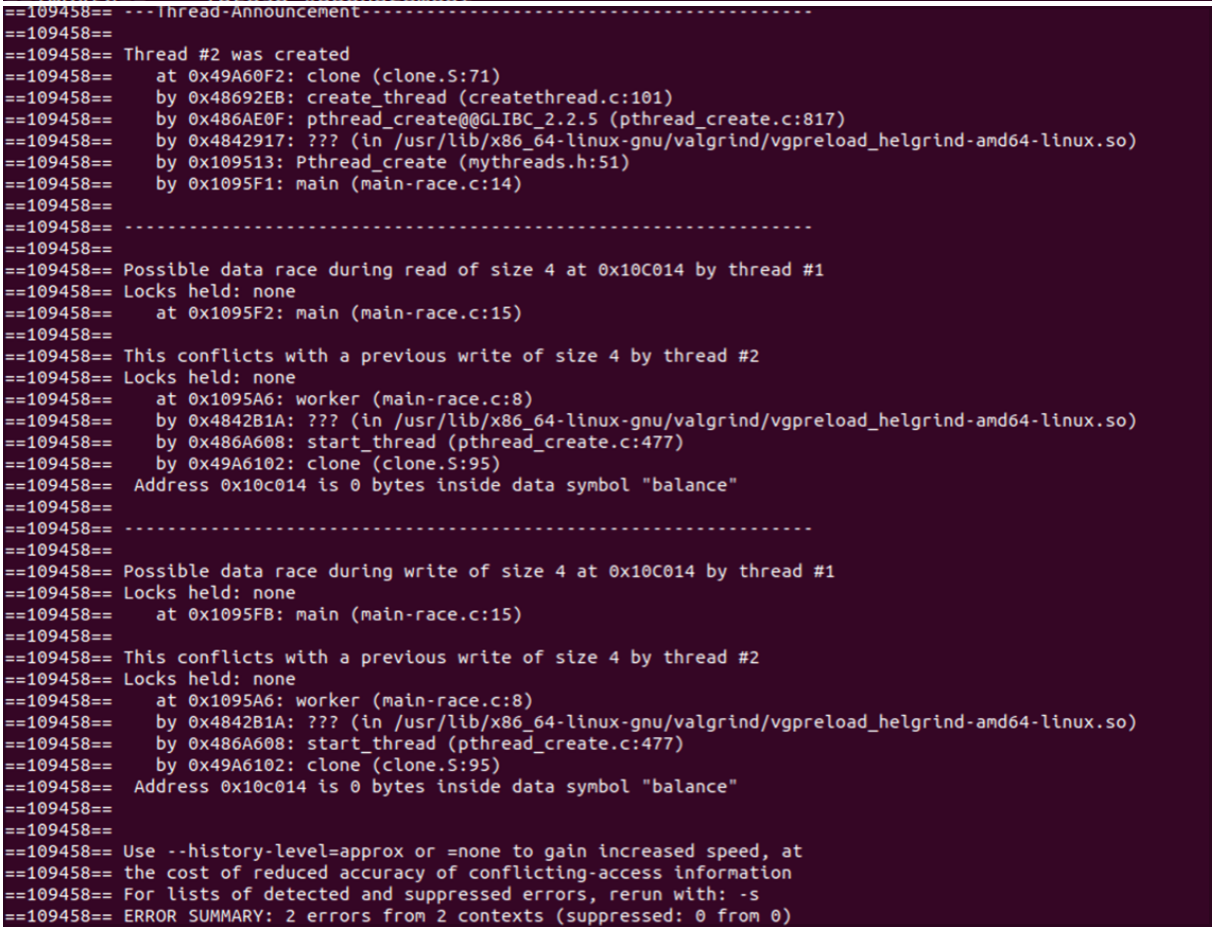
#uni/semester3/Betriebssysteme/chapter27/code-fragen

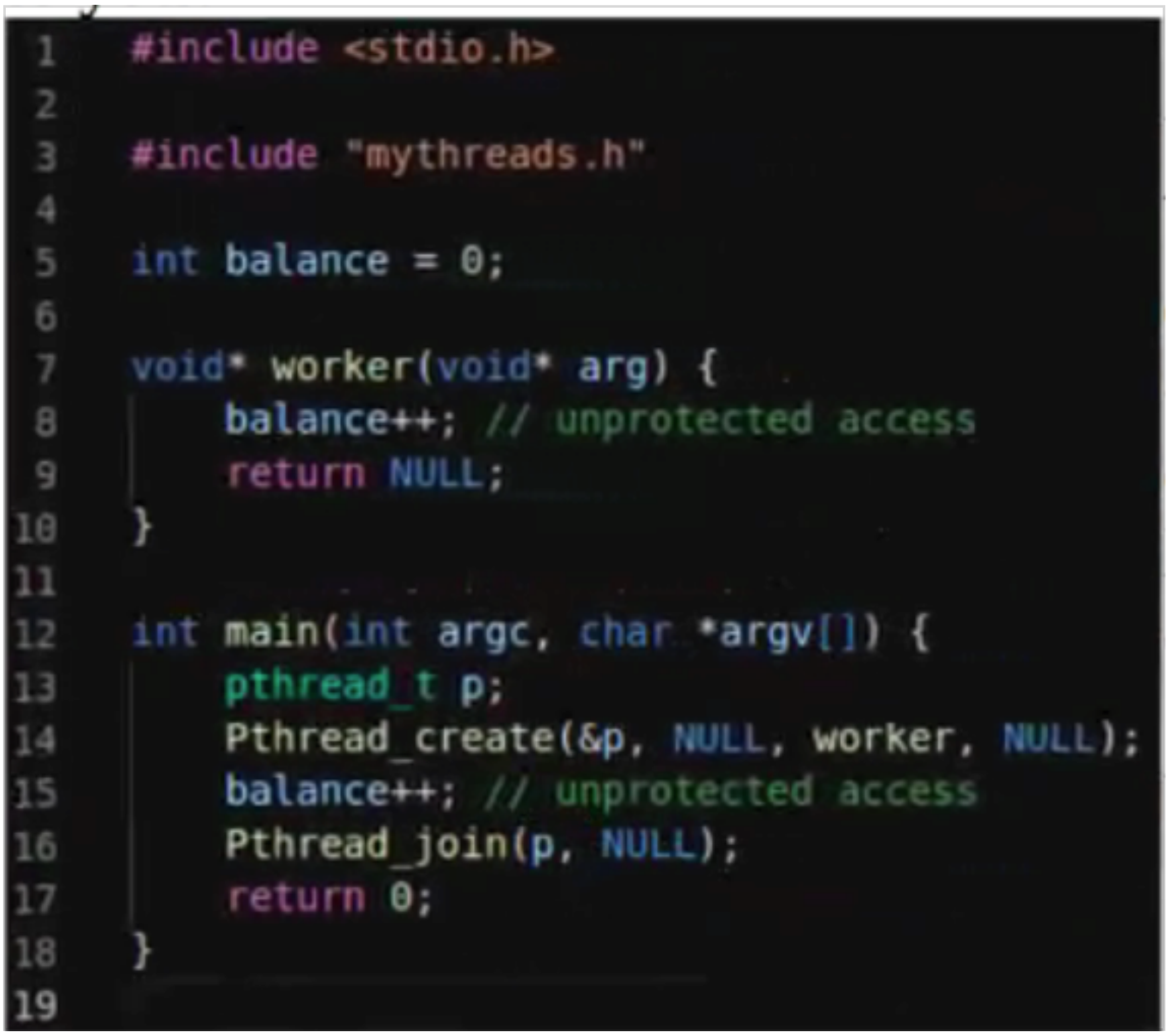
1. First build main-race.c. Examine the code so you can see the (hopefully obvious) data race in the code. Now run helgrind (by typing valgrind --tool=helgrind ./main-race) to see how it reports the race. Does it point to the right lines of code? What other information does it give to you?



```
parallels@parallels-Parallels-Virtual-Platform:~/Desktop/Parallels Shared Folders/Hom
helgrind ./main-race
==109458== Helgrind, a thread error detector
==109458== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==109458== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==109458== Command: ./main-race
==109458==
==109458== ---Thread-Announcement------------------------------------------
==109458==
==109458== Thread #1 is the program's root thread
==109458==
```



```
==109458== ---Thread-Announcement------------------------------------------
==109458==
==109458== Thread #2 was created
==109458==    at 0x49A60F2: clone (clone.S:71)
==109458==    by 0x48692EB: create_thread (createthread.c:101)
==109458==    by 0x486AE0F: pthread_create@@GLIBC_2.2.5 (pthread_create.c:817)
==109458==    by 0x4842917: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==109458==    by 0x109513: Pthread_create (mythreads.h:51)
==109458==    by 0x1095F1: main (main-race.c:14)
==109458==
==109458== ----------------------------------------------------------------
==109458==
==109458== Possible data race during read of size 4 at 0x10C014 by thread #1
==109458== Locks held: none
==109458==    at 0x1095F2: main (main-race.c:15)
==109458==
==109458== This conflicts with a previous write of size 4 by thread #2
==109458== Locks held: none
==109458==    at 0x1095A6: worker (main-race.c:8)
==109458==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==109458==    by 0x486A608: start_thread (pthread_create.c:477)
==109458==    by 0x49A6102: clone (clone.S:95)
==109458==  Address 0x10c014 is 0 bytes inside data symbol "balance"
==109458==
==109458== ----------------------------------------------------------------
==109458==
==109458== Possible data race during write of size 4 at 0x10C014 by thread #1
==109458== Locks held: none
==109458==    at 0x1095FB: main (main-race.c:15)
==109458==
==109458== This conflicts with a previous write of size 4 by thread #2
==109458== Locks held: none
==109458==    at 0x1095A6: worker (main-race.c:8)
==109458==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==109458==    by 0x486A608: start_thread (pthread_create.c:477)
==109458==    by 0x49A6102: clone (clone.S:95)
==109458==  Address 0x10c014 is 0 bytes inside data symbol "balance"
==109458==
==109458==
==109458== Use --history-level=approx or =none to gain increased speed, at
==109458== the cost of reduced accuracy of conflicting-access information
==109458== For lists of detected and suppressed errors, rerun with: -s
==109458== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```



```
 1  #include <stdio.h>
 2
 3  #include "mythreads.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  }
19
```

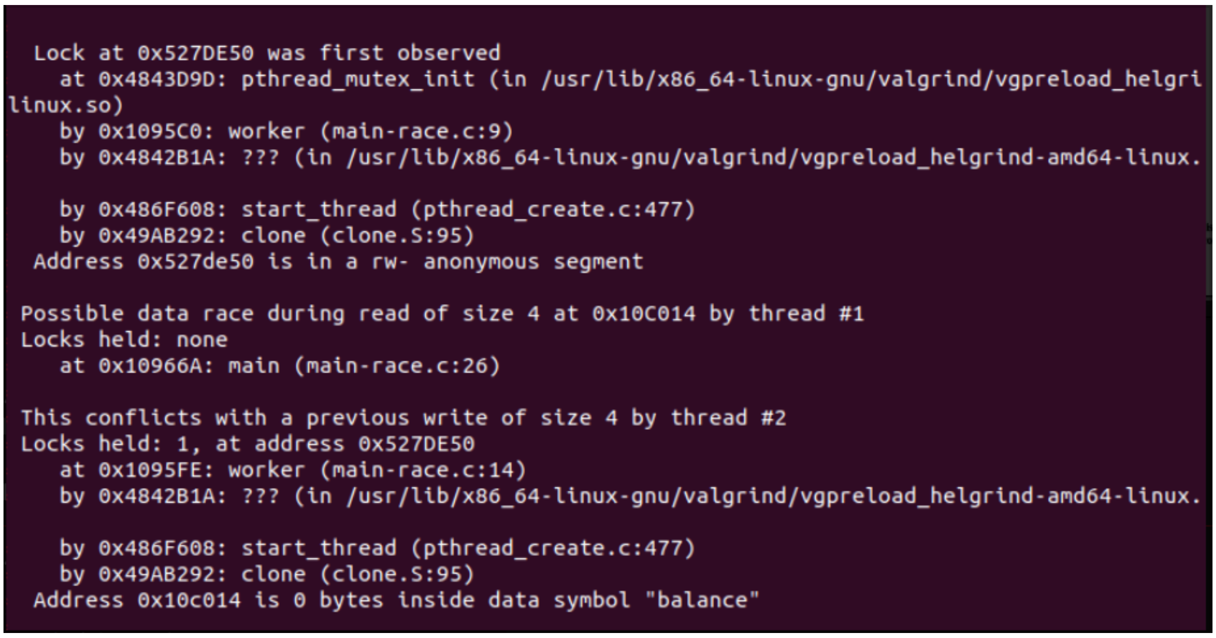
➡ Two errors are shown. balance++ means balance = balance + 1. So one error comes from trying to read balance by Thread 1 but it was before written by Thread 2. Second error comes from trying to write to balance by Thread 1 but it was written before by Thread 2. The error in Thread 1 happens at line 15. In this case Thread 2 executed before Thread 1.

➡ It shows when Threads were created.

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?

➡ If one of the lines 8 and 15 are commented, no errors will be shown. Both also results in no error.

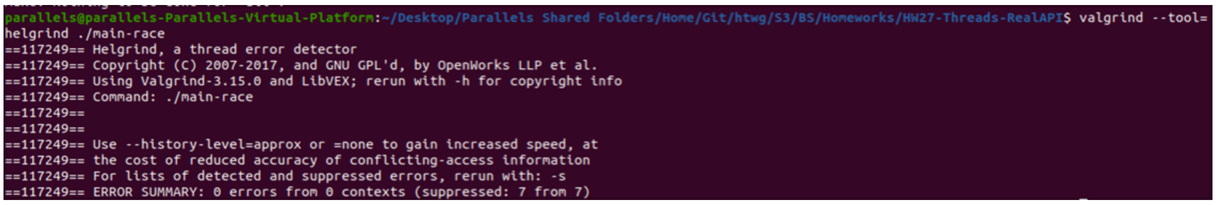
➡ If a lock is only put around a line of the above lines, helgrind shows additionally this:



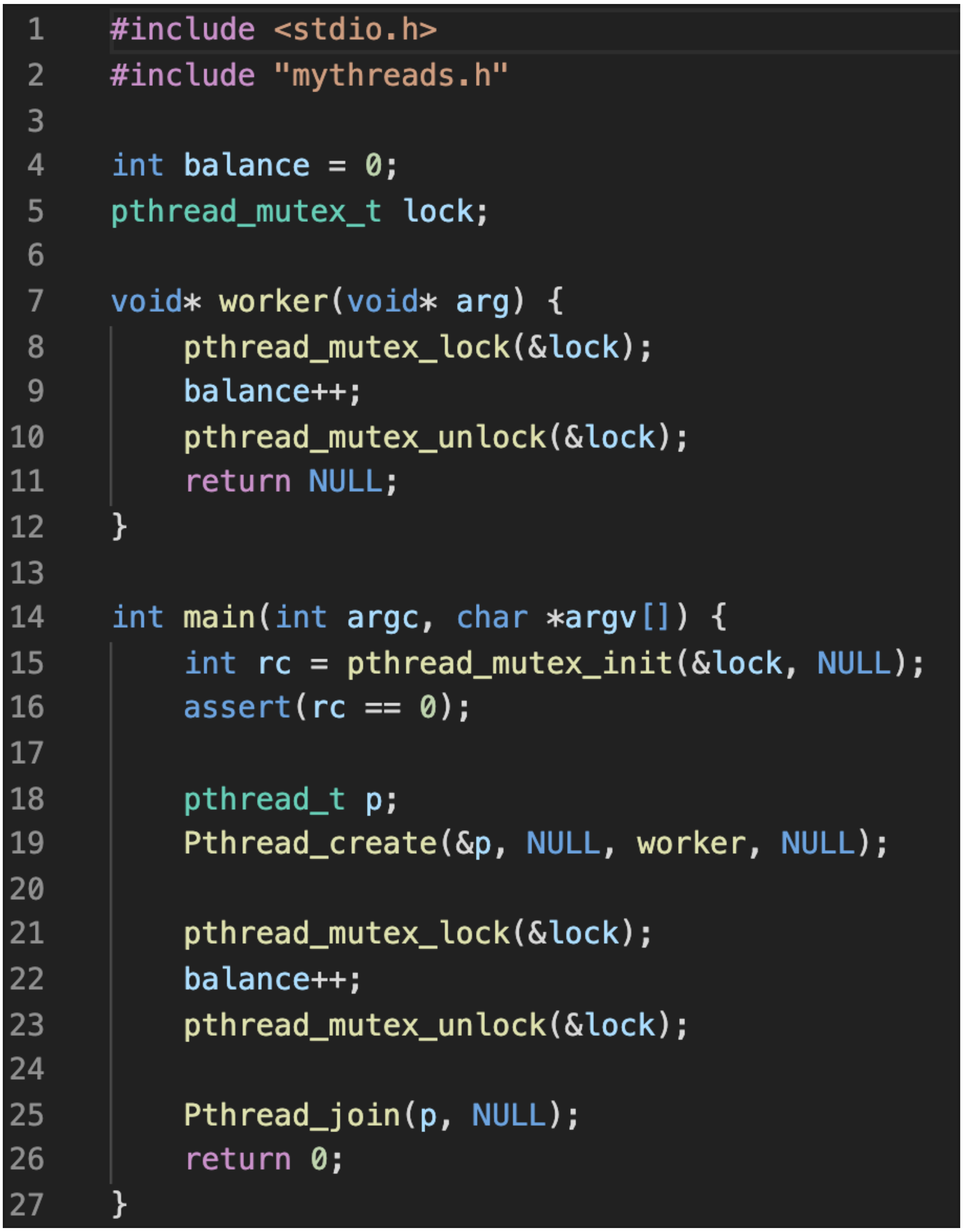
```
 Lock at 0x527DE50 was first observed
    at 0x4843D9D: pthread_mutex_init (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgri
linux.so)
    by 0x1095C0: worker (main-race.c:9)
    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.

    by 0x486F608: start_thread (pthread_create.c:477)
    by 0x49AB292: clone (clone.S:95)
 Address 0x527de50 is in a rw- anonymous segment

Possible data race during read of size 4 at 0x10C014 by thread #1
Locks held: none
    at 0x10966A: main (main-race.c:26)

This conflicts with a previous write of size 4 by thread #2
Locks held: 1, at address 0x527DE50
    at 0x1095FE: worker (main-race.c:14)
    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.

    by 0x486F608: start_thread (pthread_create.c:477)
    by 0x49AB292: clone (clone.S:95)
 Address 0x10c014 is 0 bytes inside data symbol "balance"
```

➡ (von dani)

➡ If the lock is made global and put around both, the helgrind shows now error.



```
parallels@parallels-Parallels-Virtual-Platform:~/Desktop/Parallels Shared Folders/Home/Git/htwg/S3/BS/Homeworks/HW27-Threads-RealAPI$ valgrind --tool=
helgrind ./main-race
==117249== Helgrind, a thread error detector
==117249== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==117249== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==117249== Command: ./main-race
==117249==
==117249==
==117249== Use --history-level=approx or =none to gain increased speed, at
==117249== the cost of reduced accuracy of conflicting-access information
==117249== For lists of detected and suppressed errors, rerun with: -s
==117249== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 7 from 7)
```

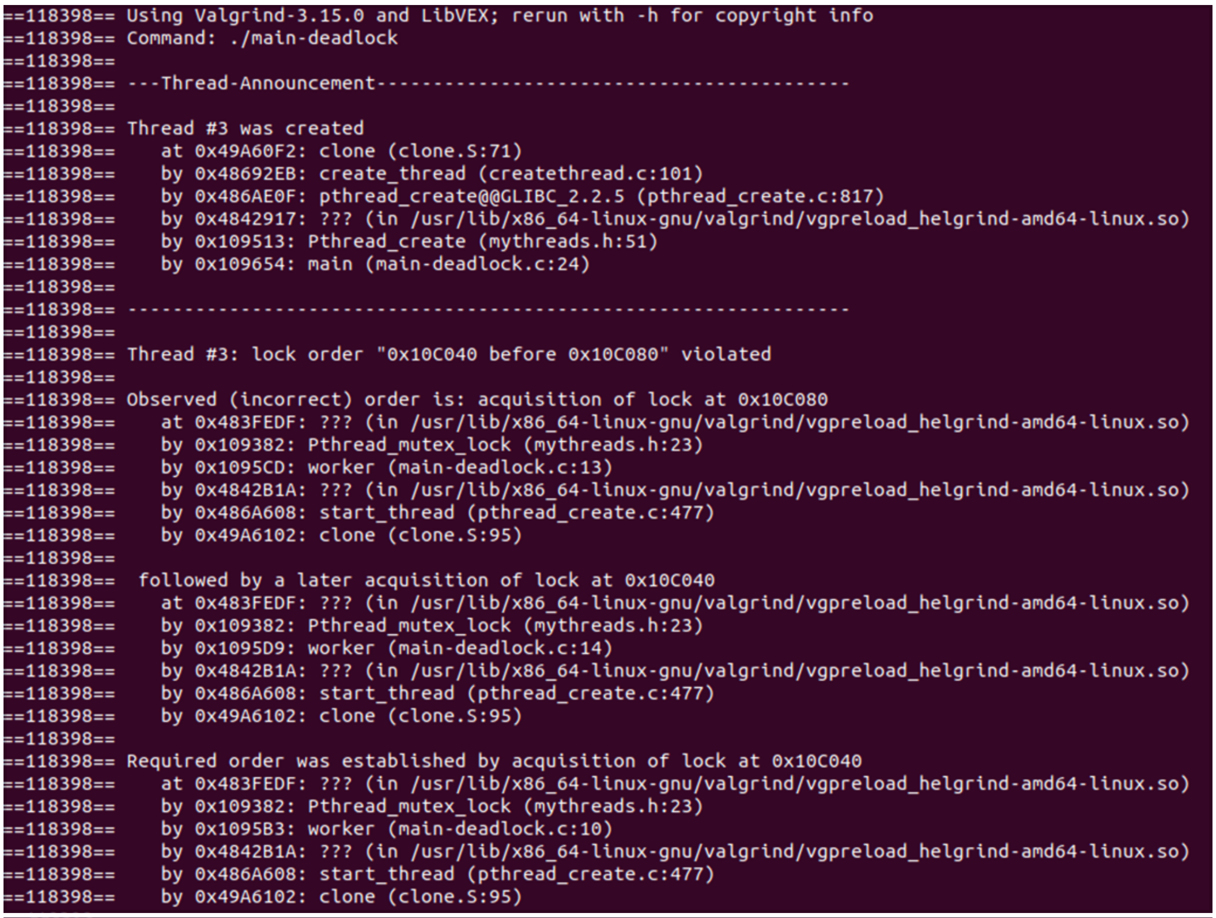


```c
#include <stdio.h>
#include "mythreads.h"

int balance = 0;
pthread_mutex_t lock;

void* worker(void* arg) {
    pthread_mutex_lock(&lock);
    balance++;
    pthread_mutex_unlock(&lock);
    return NULL;
}

int main(int argc, char *argv[]) {
    int rc = pthread_mutex_init(&lock, NULL);
    assert(rc == 0);

    pthread_t p;
    Pthread_create(&p, NULL, worker, NULL);

    pthread_mutex_lock(&lock);
    balance++;
    pthread_mutex_unlock(&lock);

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

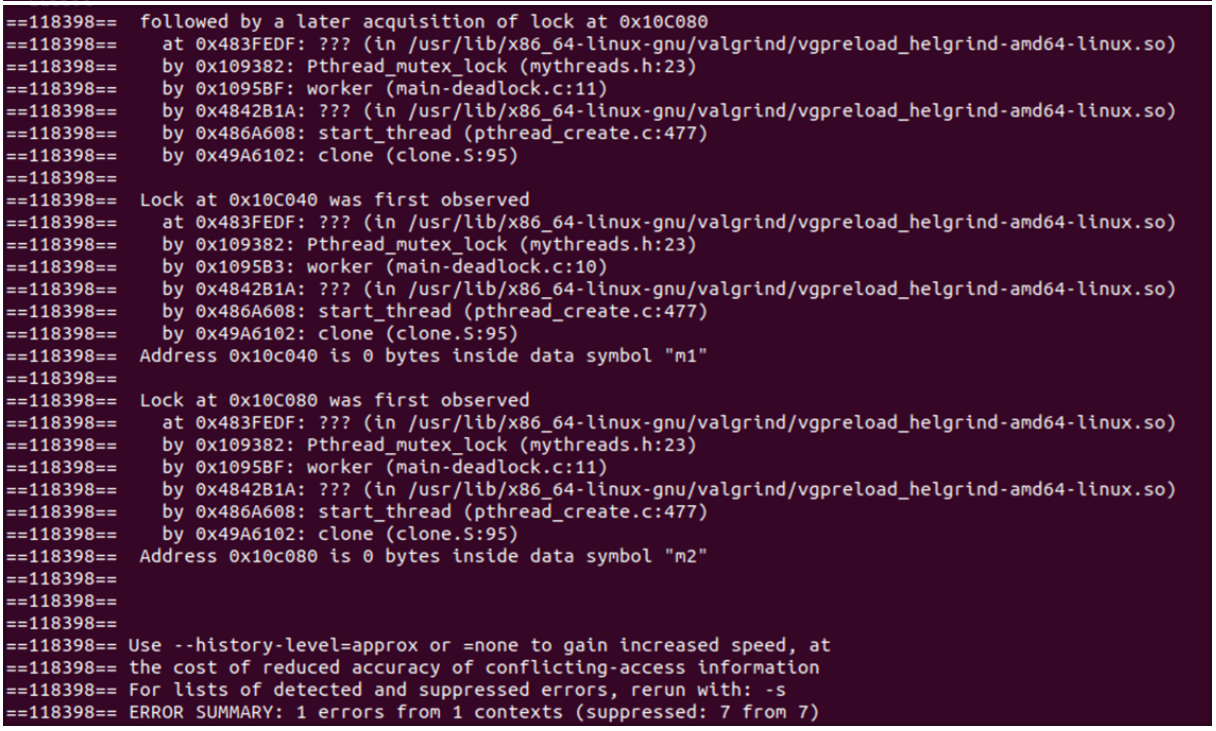
3. Now let's look at main-deadlock.c. Examine the code. This code has a problem known as deadlock (which we discuss in much more depth in a forthcoming chapter). Can you see what problem it might have?
➡ helgrind detects a potential deadlock, namely when p1 and p2 run in parallel, and p1 locks m1 and p2 locks m2, it results in a deadlock.
➡ it doesn't mean that the error happened, but only that valgrind analyzes the code and shows potential errors. 7. Helgrind: a thread error detector

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

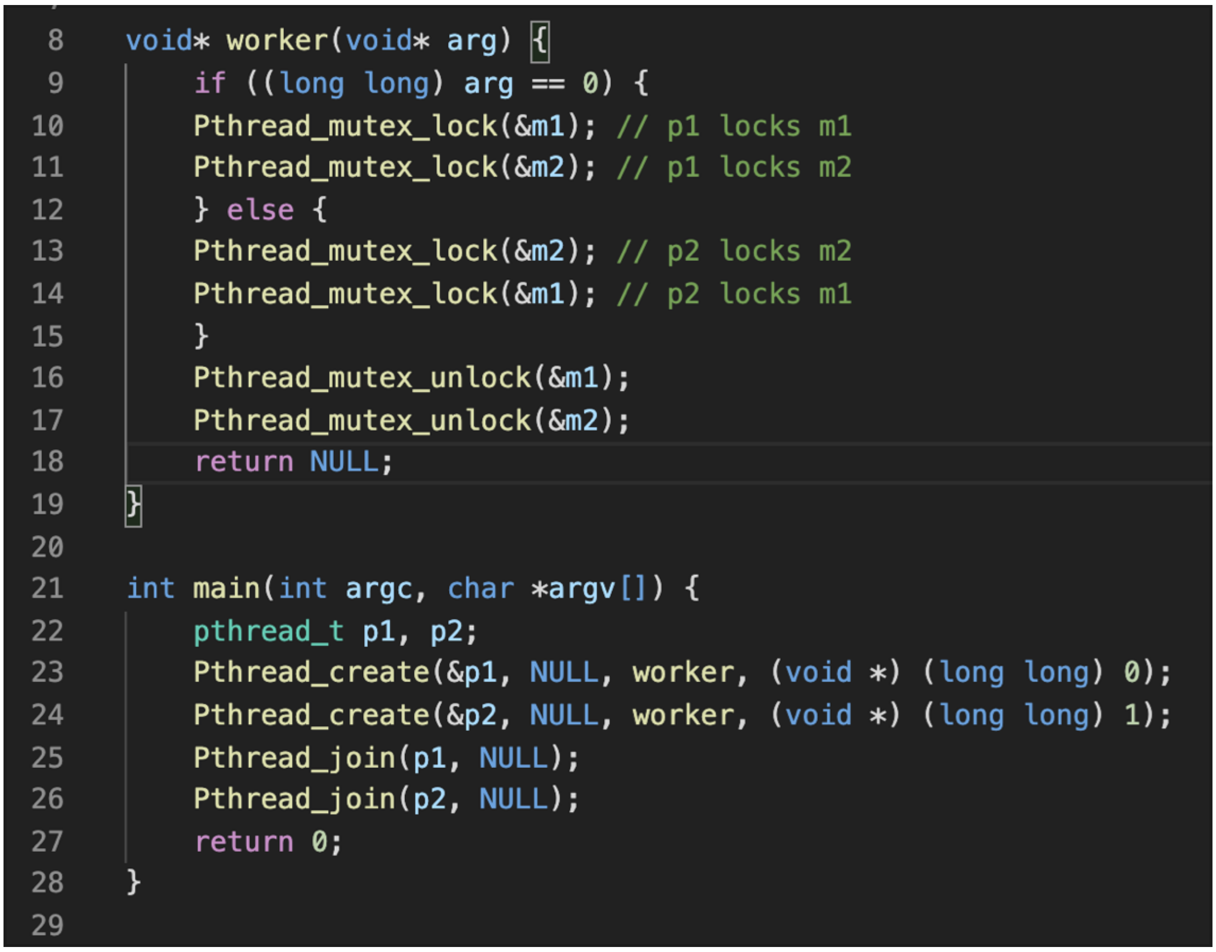
➡ Order of lines: 24, 13, 14, 10, 11, 10, 11



```
==118398== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==118398== Command: ./main-deadlock
==118398==
==118398== ---Thread-Announcement------------------------------------------
==118398==
==118398== Thread #3 was created
==118398==    at 0x49A60F2: clone (clone.S:71)
==118398==    by 0x48692EB: create_thread (createthread.c:101)
==118398==    by 0x486AE0F: pthread_create@@GLIBC_2.2.5 (pthread_create.c:817)
==118398==    by 0x4842917: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x109513: Pthread_create (mythreads.h:51)
==118398==    by 0x109654: main (main-deadlock.c:24)
==118398==
==118398== ----------------------------------------------------------------
==118398==
==118398== Thread #3: lock order "0x10C040 before 0x10C080" violated
==118398==
==118398== Observed (incorrect) order is: acquisition of lock at 0x10C080
==118398==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==118398==    by 0x1095CD: worker (main-deadlock.c:13)
==118398==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x486A608: start_thread (pthread_create.c:477)
==118398==    by 0x49A6102: clone (clone.S:95)
==118398==
==118398==  followed by a later acquisition of lock at 0x10C040
==118398==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==118398==    by 0x1095D9: worker (main-deadlock.c:14)
==118398==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x486A608: start_thread (pthread_create.c:477)
==118398==    by 0x49A6102: clone (clone.S:95)
==118398==
==118398== Required order was established by acquisition of lock at 0x10C040
==118398==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==118398==    by 0x1095B3: worker (main-deadlock.c:10)
==118398==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x486A608: start_thread (pthread_create.c:477)
==118398==    by 0x49A6102: clone (clone.S:95)
```



```
==118398==  followed by a later acquisition of lock at 0x10C080
==118398==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==118398==    by 0x1095BF: worker (main-deadlock.c:11)
==118398==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x486A608: start_thread (pthread_create.c:477)
==118398==    by 0x49A6102: clone (clone.S:95)
==118398==
==118398==  Lock at 0x10C040 was first observed
==118398==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==118398==    by 0x1095B3: worker (main-deadlock.c:10)
==118398==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x486A608: start_thread (pthread_create.c:477)
==118398==    by 0x49A6102: clone (clone.S:95)
==118398==  Address 0x10c040 is 0 bytes inside data symbol "m1"
==118398==
==118398==  Lock at 0x10C080 was first observed
==118398==    at 0x483FEDF: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x109382: Pthread_mutex_lock (mythreads.h:23)
==118398==    by 0x1095BF: worker (main-deadlock.c:11)
==118398==    by 0x4842B1A: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==118398==    by 0x486A608: start_thread (pthread_create.c:477)
==118398==    by 0x49A6102: clone (clone.S:95)
==118398==  Address 0x10c080 is 0 bytes inside data symbol "m2"
==118398==
==118398==
==118398==
==118398== Use --history-level=approx or =none to gain increased speed, at
==118398== the cost of reduced accuracy of conflicting-access information
==118398== For lists of detected and suppressed errors, rerun with: -s
==118398== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 7 from 7)
```

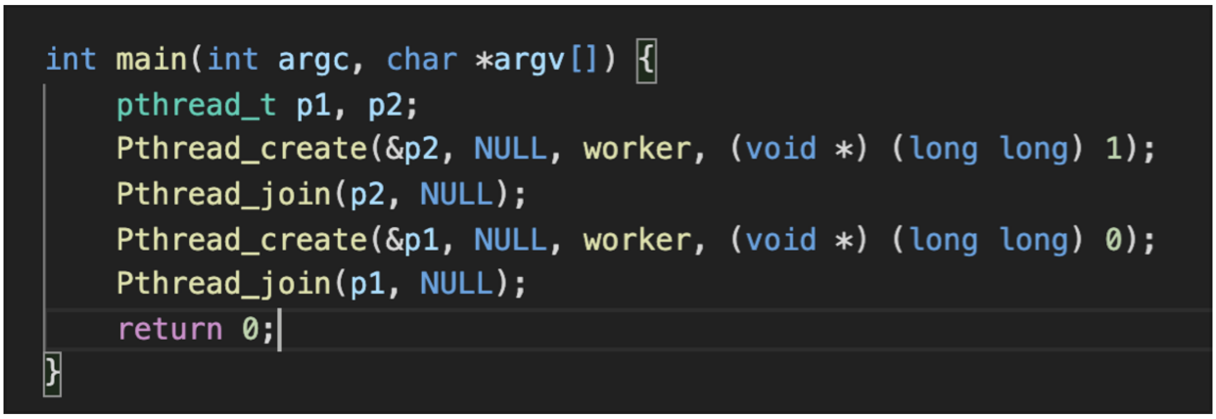


```c
void* worker(void* arg) {
    if ((long long) arg == 0) {
    Pthread_mutex_lock(&m1); // p1 locks m1
    Pthread_mutex_lock(&m2); // p1 locks m2
    } else {
    Pthread_mutex_lock(&m2); // p2 locks m2
    Pthread_mutex_lock(&m1); // p2 locks m1
    }
    Pthread_mutex_unlock(&m1);
    Pthread_mutex_unlock(&m2);
    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;
}
```

➡ Thread lock order 40 before 80.

➡ In this example p1 runs before p2. This is shown by Observed (incorrect order) is: acquisition of lock 80 and then the two lines from p2.

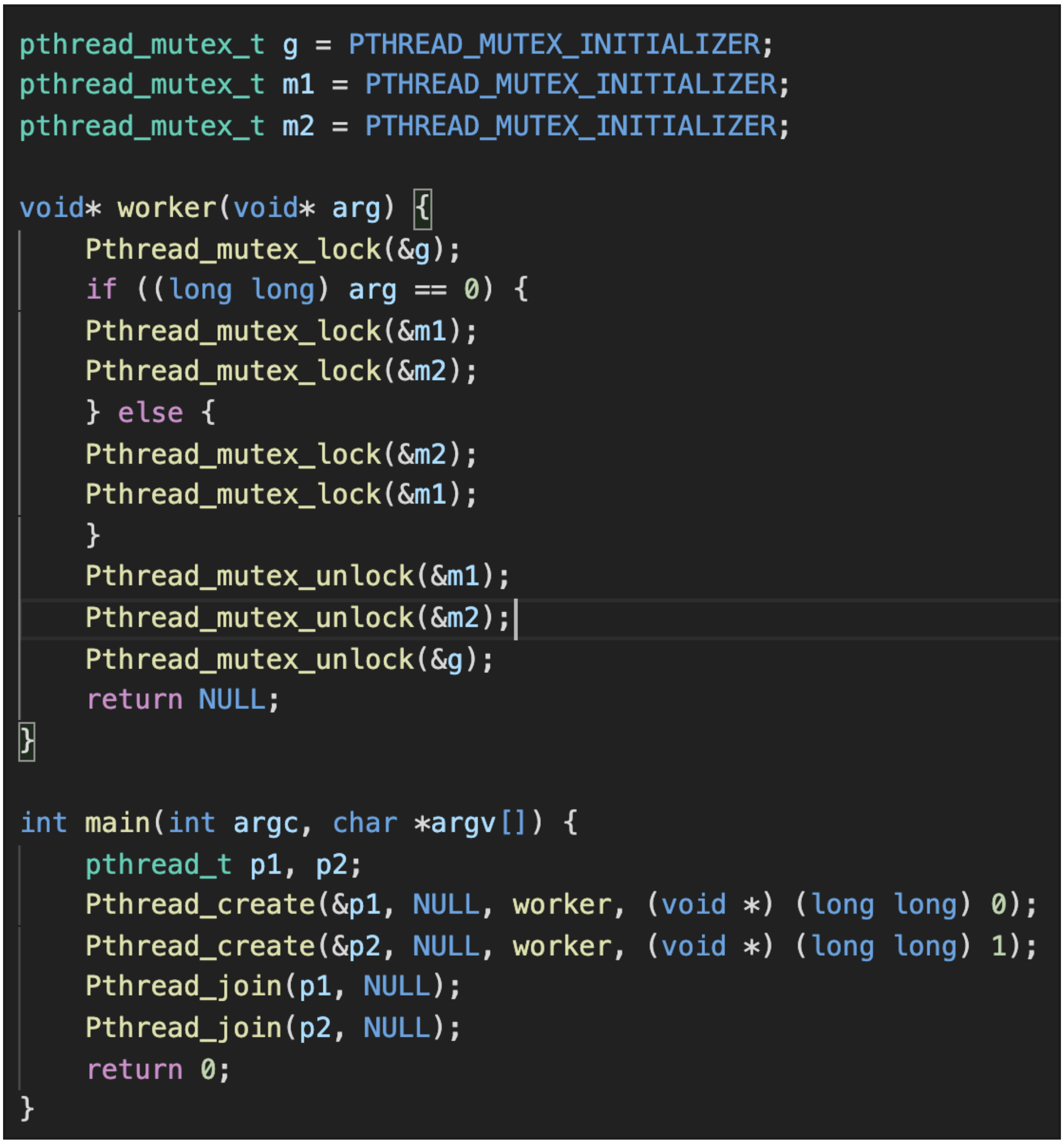
➡ The correct order is that of p1, shown by Required Order showed was established by acquisition of lock 40, followed by the two lines of p1.



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

➡ If we let p2 run first, then its acquisition order will be the correct one. So we will get: Thread lock order 80 before 40.

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?



```
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;
}
```

➡ No, it doesn't have the same error, because the whole block is being locked by an extra lock 'g'. Helgrind still shows the same error with orderings.

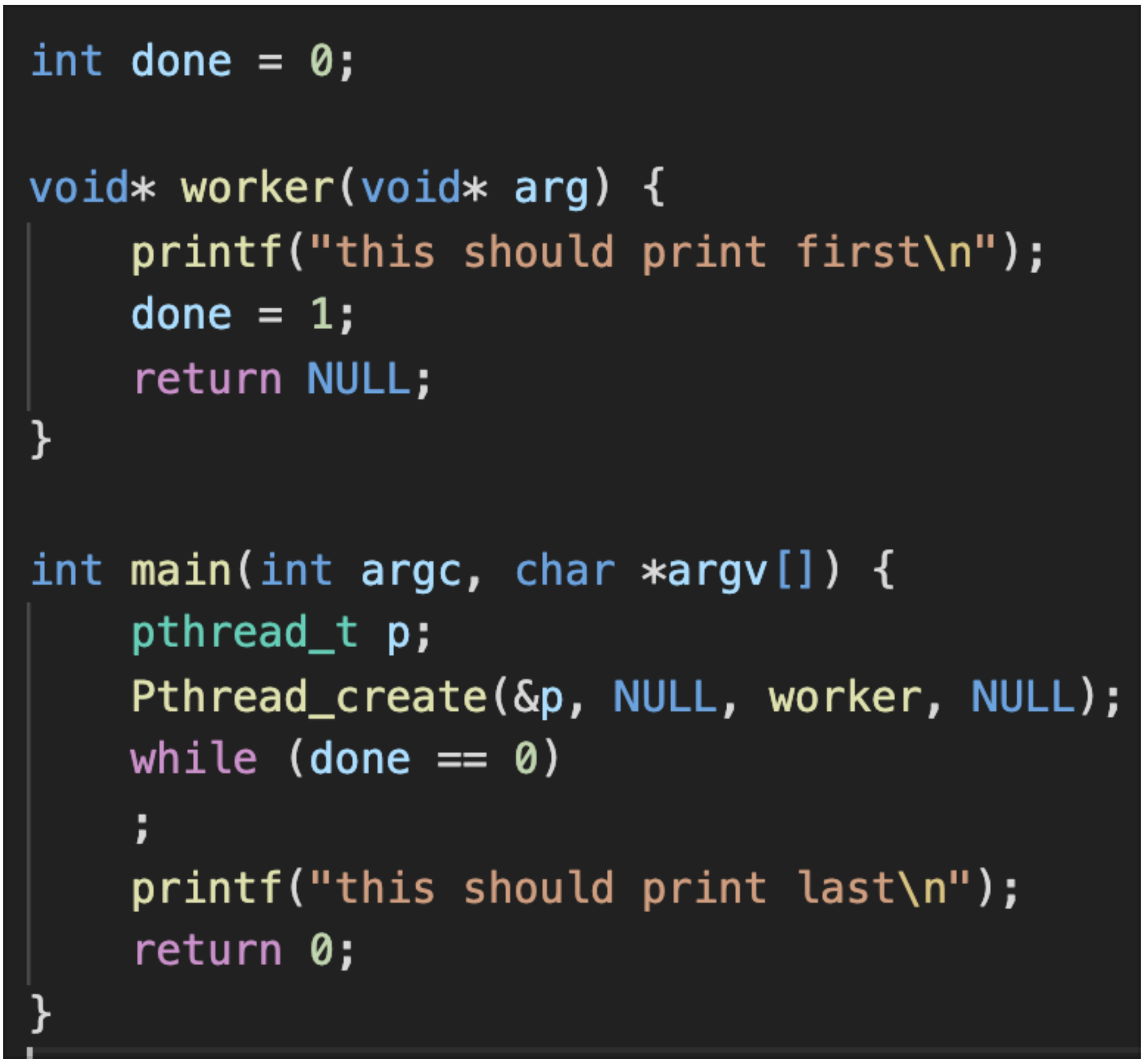
➡ No it shouldn't show an error. This is a kind of false positive, where helgrind doesn't understand, that it can't come to a deadlock.

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?)

➡ It is inefficient, because the main thread just loops in the while loop till the child sets the flag. This is wasting CPU cycles. The main thread isn't in a blocked state (as it is when the join routine is used).

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

➡ No it throws an error again, about possible data race. Global variable is accessed without the usage of locks.



```c
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;
}
```

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?

➡ Performance, because the main thread is going to sleep.

➡ And it is more correct code.

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

➡ No error, everything is perfect.