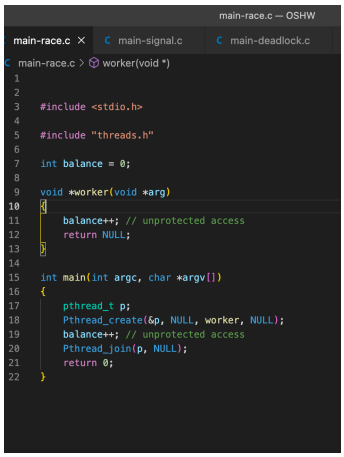
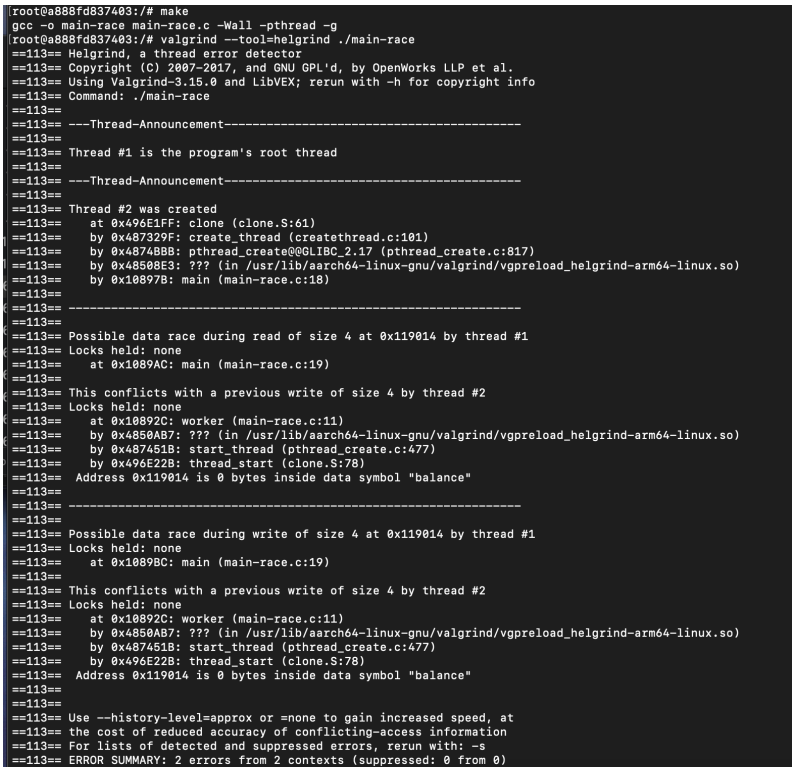
HW7 Thread API

Created	@April 17, 2022 2:15 PM
Tags	

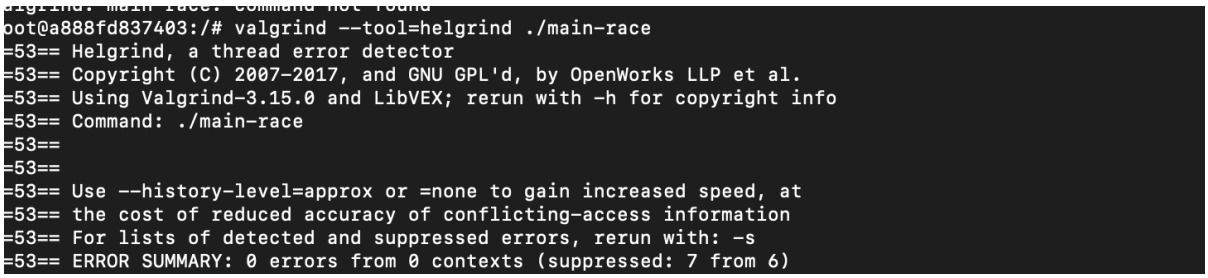
1️⃣ First build main-race.c. Examine the code so you can see the (hopefully
obvious) data race in the code. Now run helgrind (by typing valgrind
--tool=helgrind main-race) /to see how it reports the race. Does it
point to the right lines of code? What other information does it give to you?





It throws an error and Helgrind detects the conflicted address and the corresponding memory size(in bytes).

2️⃣ 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?



There was no such error despite removing the offending lines(balance++ in my codeof the code.

3️⃣ 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?

There were 2 errors that were locked to each other

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

```
root@a888fd837403:/# valgrind --tool=helgrind ./main-deadlock
==60== Helgrind, a thread error detector
==60== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==60== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==60== Command: ./main-deadlock
==60==
^C==60==
==60== Process terminating with default action of signal 2 (SIGINT)
==60==    at 0x4875818: __pthread_clockjoin_ex (pthread_join_common.c:145)
==60==    by 0x484E12F: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==60==    by 0x108EBB: Pthread_join (threads.h:65)
==60==    by 0x108FD7: main (main-deadlock.c:30)
==60== ---Thread-Announcement------------------------------------------
==60==
==60== Thread #2 was created
==60==    at 0x496E1FF: clone (clone.S:61)
==60==    by 0x487329F: create_thread (createthread.c:101)
==60==    by 0x4874BBB: pthread_create@@GLIBC_2.17 (pthread_create.c:817)
==60==    by 0x48508E3: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==60==    by 0x108E63: Pthread_create (threads.h:59)
==60==    by 0x108FAF: main (main-deadlock.c:28)
==60==
==60== ----------------------------------------------------------------
==60==
==60== Thread #2: Exiting thread still holds 1 lock
==60==    at 0x487EBB0: __lll_lock_wait (lowlevellock.c:48)
==60==    by 0x4876BC7: pthread_mutex_lock (pthread_mutex_lock.c:80)
==60==    by 0x484E2EF: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==60==    by 0x108CAB: Pthread_mutex_lock (threads.h:26)
==60==    by 0x108F2F: worker (main-deadlock.c:13)
==60==    by 0x4850AB7: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==60==    by 0x487451B: start_thread (pthread_create.c:477)
==60==    by 0x496E22B: thread_start (clone.S:78)
==60==
==60== ---Thread-Announcement------------------------------------------
==60==
==60== Thread #3 was created
==60==    at 0x496E1FF: clone (clone.S:61)
==60==    by 0x487329F: create_thread (createthread.c:101)
==60==    by 0x4874BBB: pthread_create@@GLIBC_2.17 (pthread_create.c:817)
==60==    by 0x48508E3: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==60==    by 0x108E63: Pthread_create (threads.h:59)
==60==    by 0x108FCB: main (main-deadlock.c:29)
==60==
==60== ----------------------------------------------------------------
==60==
==60== Thread #3: Exiting thread still holds 1 lock
==60==    at 0x487EBB0: __lll_lock_wait (lowlevellock.c:48)
==60==    by 0x4876BC7: pthread_mutex_lock (pthread_mutex_lock.c:80)
==60==    by 0x484E2EF: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==60==    by 0x108CAB: Pthread_mutex_lock (threads.h:26)
==60==    by 0x108F4B: worker (main-deadlock.c:18)
==60==    by 0x4850AB7: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==60==    by 0x487451B: start_thread (pthread_create.c:477)
==60==    by 0x496E22B: thread_start (clone.S:78)
==60==
==60==
==60== Use --history-level=approx or =none to gain increased speed, at
==60== the cost of reduced accuracy of conflicting-access information
==60== For lists of detected and suppressed errors, rerun with: -s
==60== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

The threads that were lock to each other are elucidated via helgrind.

5️⃣ 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?

```
root@a888fd837403:/# valgrind --tool=helgrind ./main-deadlock-global
==121== Helgrind, a thread error detector
==121== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==121== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==121== Command: ./main-deadlock-global
==121==
==121== ---Thread-Announcement------------------------------------------
==121==
==121== Thread #3 was created
==121==    at 0x496E1FF: clone (clone.S:61)
==121==    by 0x487329F: create_thread (createthread.c:101)
==121==    by 0x4874BBB: pthread_create@@GLIBC_2.17 (pthread_create.c:817)
==121==    by 0x48508E3: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x108C03: main (main-deadlock-global.c:32)
==121==
==121== ----------------------------------------------------------------
==121==
==121== Thread #3: lock order "0x11A048 before 0x11A078" violated
==121==
==121== Observed (incorrect) order is: acquisition of lock at 0x11A078
==121==    at 0x484E348: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x108A7B: worker (main-deadlock-global.c:19)
==121==    by 0x4850AB7: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x487451B: start_thread (pthread_create.c:477)
==121==    by 0x496E22B: thread_start (clone.S:78)
==121==
==121==  followed by a later acquisition of lock at 0x11A048
==121==    at 0x484E348: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x108AAF: worker (main-deadlock-global.c:20)
==121==    by 0x4850AB7: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x487451B: start_thread (pthread_create.c:477)
==121==    by 0x496E22B: thread_start (clone.S:78)
==121==
==121== Required order was established by acquisition of lock at 0x11A048
==121==    at 0x484E348: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x108A13: worker (main-deadlock-global.c:14)
==121==    by 0x4850AB7: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x487451B: start_thread (pthread_create.c:477)
==121==    by 0x496E22B: thread_start (clone.S:78)
==121==
==121==  followed by a later acquisition of lock at 0x11A078
==121==    at 0x484E348: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x108A47: worker (main-deadlock-global.c:15)
==121==    by 0x4850AB7: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x487451B: start_thread (pthread_create.c:477)
==121==    by 0x496E22B: thread_start (clone.S:78)
==121==
==121==  Lock at 0x11A048 was first observed
==121==    at 0x484E348: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x108A13: worker (main-deadlock-global.c:14)
==121==    by 0x4850AB7: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x487451B: start_thread (pthread_create.c:477)
==121==    by 0x496E22B: thread_start (clone.S:78)
==121==  Address 0x11a048 is 0 bytes inside data symbol "m1"
==121==
==121==  Lock at 0x11A078 was first observed
==121==    at 0x484E348: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x108A47: worker (main-deadlock-global.c:15)
==121==    by 0x4850AB7: ??? (in /usr/lib/aarch64-linux-gnu/valgrind/vgpreload_helgrind-arm64-linux.so)
==121==    by 0x487451B: start_thread (pthread_create.c:477)
==121==    by 0x496E22B: thread_start (clone.S:78)
==121==  Address 0x11a078 is 0 bytes inside data symbol "m2"
==121==
==121==
==121==
==121== Use --history-level=approx or =none to gain increased speed, at
==121== the cost of reduced accuracy of conflicting-access information
==121== For lists of detected and suppressed errors, rerun with: -s
==121== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 7 from 6)
```

there are same errors like the answer of question #4. However, helgrind shows not the same result. Thus, we can conclude, that helgrind is not 100% precisely working.

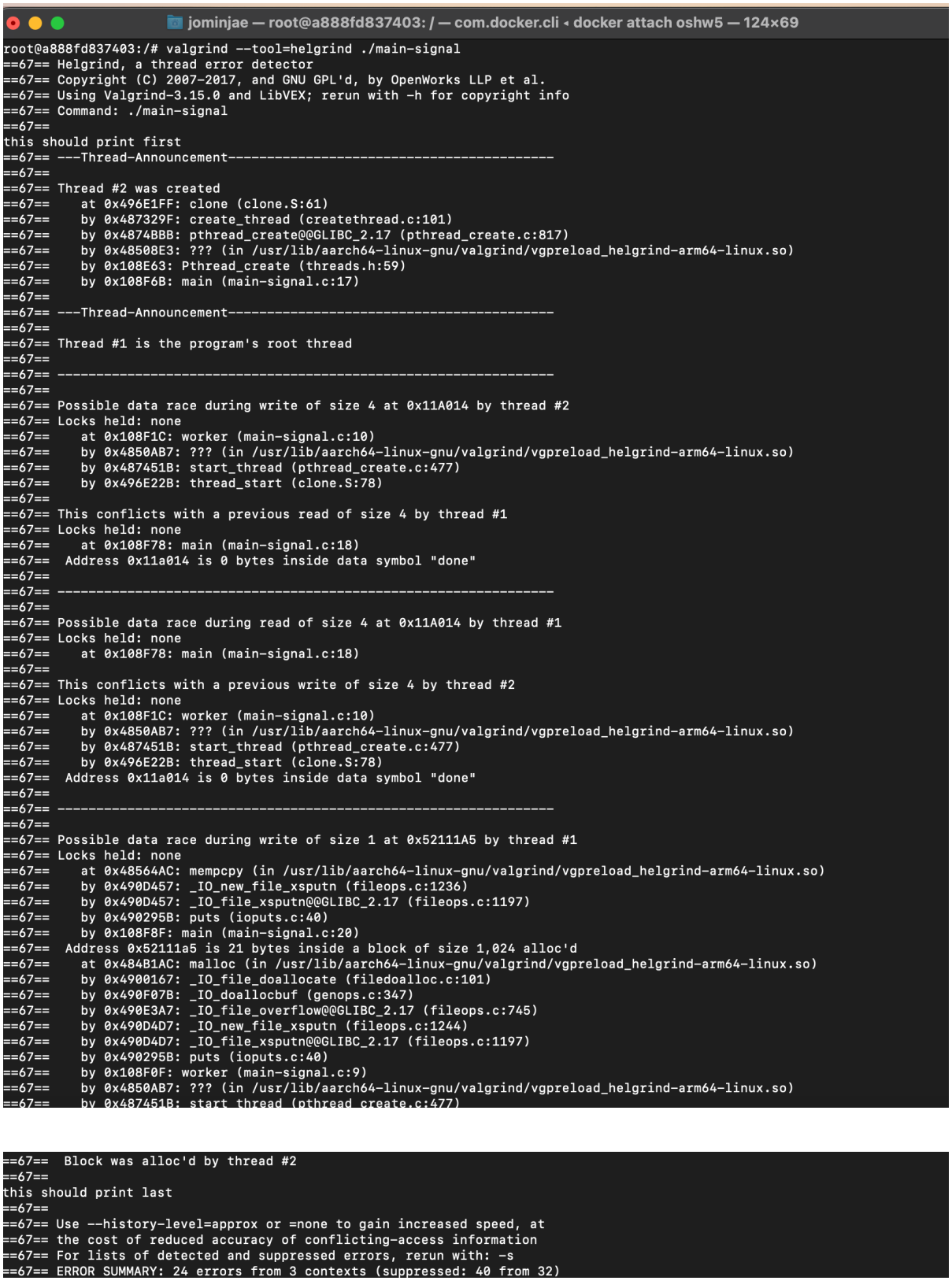
6️⃣ Let’s next look at main-signal.c. This code uses a variable (done) to
signal that the child is done and that the parent can now continue. Why is
this code inefficient? (what does the parent end up spending its time doing,
particularly if the child thread takes a long time to complete?)

```c
#include <stdio.h>

#include "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;
}
```

This is code of main-signal.c. This program will print first or last, so that user(programmer) could know which code was working first. Parent code was executed within the loop.

```
root@a888fd837403:/#
root@a888fd837403:/#
root@a888fd837403:/# ./main-signal
this should print first
this should print last
root@a888fd837403:/#
```

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



Read/write of the variable done and printf()-internal function data seem to be traced.

8️⃣ 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?

```c
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;
}

int main(int argc, char *argv[])
{
    pthread_t p;
    signal_init(&s);
    Pthread_create(&p, NULL, worker, NULL);
    signal_wait(&s);
    printf("this should print last\n");

    return 0;
}
```

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

```
root@a888fd837403:/# valgrind --tool=helgrind ./main-signal-cv
==128== Helgrind, a thread error detector
==128== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==128== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==128== Command: ./main-signal-cv
==128==
this should print first
this should print last
==128==
==128== Use --history-level=approx or =none to gain increased speed, at
==128== the cost of reduced accuracy of conflicting-access information
==128== For lists of detected and suppressed errors, rerun with: -s
==128== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 12 from 11)
```

⇒ Based on the result from Question 9, the answer for question 8 is: main-signal-cv performs better in “both” performance and accuracy.

code: https://github.com/minjaeingithub/2022_Operating_Systems