Assignment 2:
By: Manoj Kumar

Bug No. 1:

Question: Description of the bug including details on how it manifests
Answer:

Mainly in the web server, it accept request from the clients on standard port and then binds the request of the client to the worker thread which runs on non standard ports so that it can accept and serves various request simultaneously.

People had seen from the log files that few of the worker threads under-perform as they were not getting chances to serve the request.

In the debugging process, It had been found that there is a variable 'busy" and whose values are increment and decrements by worker threads and which was not protected by any kind of locking mechanism as a result few worker threads suffer from race condition.

Question: Patch (difference in code) that was used to fix the bug.
Answer:

```
diff -uNr apache2-2.2.9-old/modules/proxy/mod_proxy_balancer.c apache2-
2.2.9/modules/proxy/mod_proxy_balancer.c
--- apache2-2.2.9-old/modules/proxy/mod_proxy_balancer.c        2008-11-14 14:39:43.000000000
-0800
+++ apache2-2.2.9/modules/proxy/mod_proxy_balancer.c    2008-11-14 15:23:12.000000000 -0800
@@ -553,7 +553,6 @@
                                proxy_server_conf *conf)
 {

-#if 0
    apr_status_t rv;

    if ((rv = PROXY_THREAD_LOCK(balancer)) != APR_SUCCESS) {
@@ -562,8 +561,9 @@
            balancer->name);
        return HTTP_INTERNAL_SERVER_ERROR;
    }
-    /* TODO: placeholder for post_request actions
-     */
+
+    if (worker && worker->s->busy)
+        worker->s->busy--;

     if ((rv = PROXY_THREAD_UNLOCK(balancer)) != APR_SUCCESS) {
         ap_log_error(APLOG_MARK, APLOG_ERR, rv, r->server,
@@ -573,10 +573,6 @@
     ap_log_error(APLOG_MARK, APLOG_DEBUG, 0, r->server,
                  "proxy_balancer_post_request for (%s)", balancer->name);

-#endif
-
-    if (worker && worker->s->busy)
-        worker->s->busy--;

     return OK;
```

Question: How was the bug detected ?
Answer:

In the debugging process, they had seen that 'busy' variable is shared by various threads, but not at all protected by any kind of locking mechanism.

Question: How frequently does the bug appear? Or how important is the bug?

Answer:

This bugs appears when large number of backend workers threads perform task simultaneously, a race condition can prevent the 'busy' counter from being decremented properly, causing some workers to be ignored completely and which leads to generate timeout error at client side.

Question: Online link to the bug id
Answer:

https://issues.apache.org/bugzilla/show_bug.cgi?id=46215

Question: Generate a simple reproducer that follows the original bug
Answer:

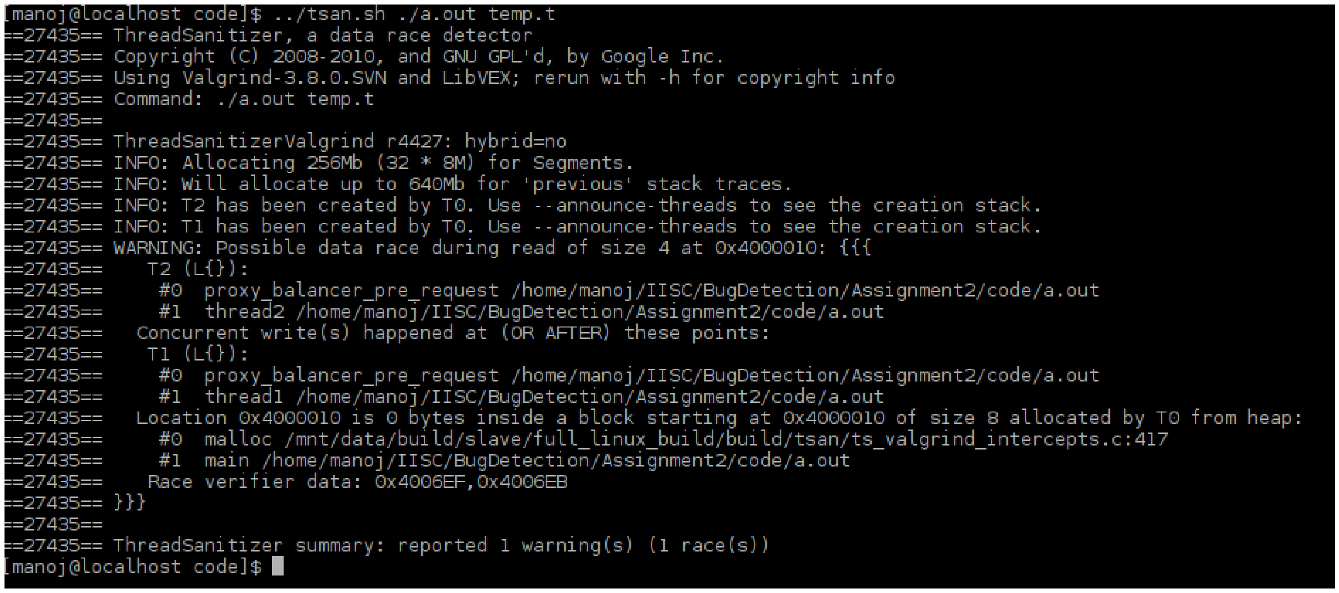
Please refer the following directory:
HW2-310-spring/Code/Bug1/bug1_45215.c

Question: Investigate whether any available program analysis tool can catch this bug automatically. If so, name the tool and a snapshot of its

report.

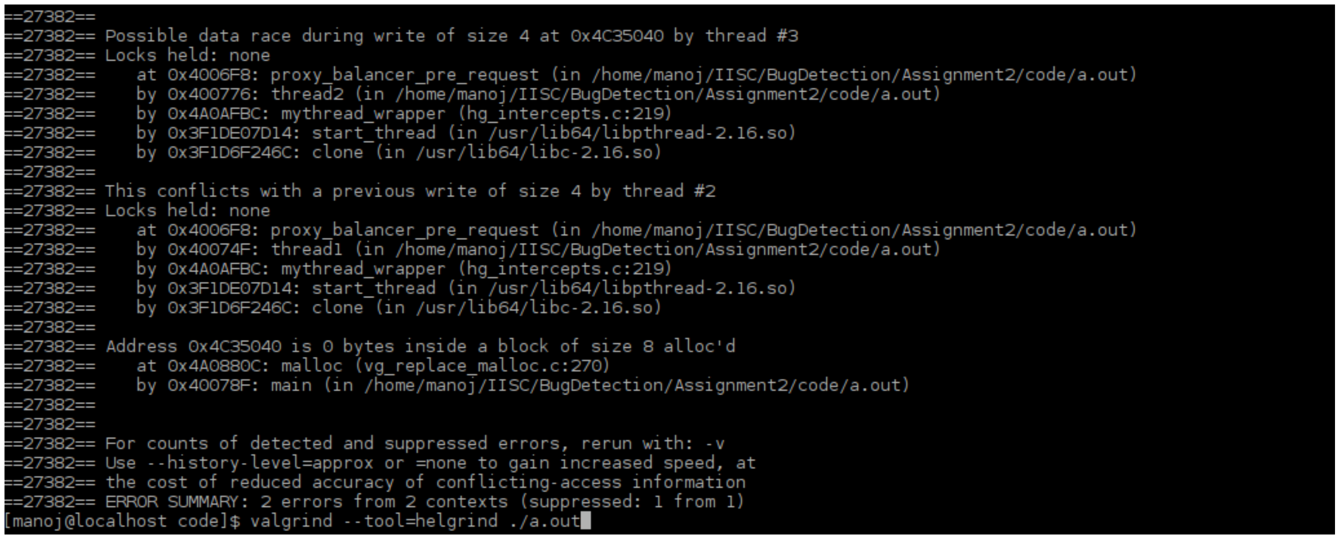
Answer:

Yes i have tried on two dynamic race detection tools and almost all have detected data race bug.



```
[manoj@localhost code]$ ../tsan.sh ./a.out temp.t
==27435== ThreadSanitizer, a data race detector
==27435== Copyright (C) 2008-2010, and GNU GPL'd, by Google Inc.
==27435== Using Valgrind-3.8.0.SVN and LibVEX; rerun with -h for copyright info
==27435== Command: ./a.out temp.t
==27435==
==27435== ThreadSanitizerValgrind r4427: hybrid=no
==27435== INFO: Allocating 256Mb (32 * 8M) for Segments.
==27435== INFO: Will allocate up to 640Mb for 'previous' stack traces.
==27435== INFO: T2 has been created by T0. Use --announce-threads to see the creation stack.
==27435== INFO: T1 has been created by T0. Use --announce-threads to see the creation stack.
==27435== WARNING: Possible data race during read of size 4 at 0x4000010: {{{
==27435==    T2 (L{}):
==27435==     #0  proxy_balancer_pre_request /home/manoj/IISC/BugDetection/Assignment2/code/a.out
==27435==     #1  thread2 /home/manoj/IISC/BugDetection/Assignment2/code/a.out
==27435==   Concurrent write(s) happened at (OR AFTER) these points:
==27435==    T1 (L{}):
==27435==     #0  proxy_balancer_pre_request /home/manoj/IISC/BugDetection/Assignment2/code/a.out
==27435==     #1  thread1 /home/manoj/IISC/BugDetection/Assignment2/code/a.out
==27435==   Location 0x4000010 is 0 bytes inside a block starting at 0x4000010 of size 8 allocated by T0 from heap:
==27435==     #0  malloc /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:417
==27435==     #1  main /home/manoj/IISC/BugDetection/Assignment2/code/a.out
==27435==    Race verifier data: 0x4006EF,0x4006EB
==27435== }}}
==27435==
==27435== ThreadSanitizer summary: reported 1 warning(s) (1 race(s))
[manoj@localhost code]$
```

Figure 1.1: bug detected by ThreadSanitizer tool



```
==27382==
==27382== Possible data race during write of size 4 at 0x4C35040 by thread #3
==27382== Locks held: none
==27382==    at 0x4006F8: proxy_balancer_pre_request (in /home/manoj/IISC/BugDetection/Assignment2/code/a.out)
==27382==    by 0x400776: thread2 (in /home/manoj/IISC/BugDetection/Assignment2/code/a.out)
==27382==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==27382==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==27382==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==27382==
==27382== This conflicts with a previous write of size 4 by thread #2
==27382== Locks held: none
==27382==    at 0x4006F8: proxy_balancer_pre_request (in /home/manoj/IISC/BugDetection/Assignment2/code/a.out)
==27382==    by 0x40074F: thread1 (in /home/manoj/IISC/BugDetection/Assignment2/code/a.out)
==27382==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==27382==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==27382==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==27382==
==27382== Address 0x4C35040 is 0 bytes inside a block of size 8 alloc'd
==27382==    at 0x4A0880C: malloc (vg_replace_malloc.c:270)
==27382==    by 0x40078F: main (in /home/manoj/IISC/BugDetection/Assignment2/code/a.out)
==27382==
==27382==
==27382== For counts of detected and suppressed errors, rerun with: -v
==27382== Use --history-level=approx or =none to gain increased speed, at
==27382== the cost of reduced accuracy of conflicting-access information
==27382== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 1 from 1)
[manoj@localhost code]$ valgrind --tool=helgrind ./a.out
```

Figure 1.2: bug detected by Valgrind –tool=helgrind

Question: Design an approach that will detect this bug automatically.

Answer:

Here, developer has not at all use locks to make shared variable thread safe hence detection data race detection is easy.

Approach: Static analysis

```
If( multiple_pthread_create_not_exist)
        return( no_data_race ) // If program is not multithread
else{

        1) Analysis the functions that are called inside the pthread_create and store the shared variables
        into a set.

        2) Calculate intersection of different sets and if intersection value is not    null between two sets
        and those shared variables are not protected by any lock then generates warning for data race.

}
```

Question: Does your approach compare with any existing approach? If so, how?

Answer: No

Question: Mention any drawbacks of your approach. Present clear examples where these drawbacks will manifest?

Answer:

My approach will work fine but when there are many threads then finding the intersection is time consuming and complexity goes in factorial. For example

{a, b ,c} {b, d} {e , f } { g, h} { a , b}

set 1 : Total comparisons are: 4 - {a,b,c} to {b,d}, {a,b,c} to {e,f}, {a,b,c} to {g,h}, {a,b,c} – {a,b}
set 2 : Total comparisons are: 3 - {b,d} to {e,f}, {b,d} to {g,h}, {a,b} to {a,b}
set 3 : Total comparisons are: 2 – {e,f} to {g,h}, {e,f} to {a,b}

Question: What were the reasons that you could not adjust your approach to handle the above mentioned drawbacks?

Answer:

No, i have generated the approach and that works fine on the above simplified version of same bug.

Bug No. 2:

Question: Description of the bug including details on how it manifests

Answer:

This bug comes under deadlock category. It occurs when multiple request comes from client as a http request to download files whose size is above 200MB. In the server side, main thread accepts request and binds the request to worker thread and worker thread serves the request. In the serving process worker thread captures some lock and perform some tasks. Since the serving time is very high as the consequence other thread starves and that leads to deadlock.

Question: Patch (difference in code) that was used to fix the bug.

Answer:

```
Index: event.c
===================================================================
--- event.c     (revision 572596)
+++ event.c     (working copy)
@@ -1070,14 +1070,17 @@
        cs = APR_RING_FIRST(&keepalive_timeout_head);
        timeout_time = time_now + TIMEOUT_FUDGE_FACTOR;
        while (!APR_RING_EMPTY(&keepalive_timeout_head, conn_state_t, timeout_list)
-              && cs->expiration_time < timeout_time
-              && get_worker(&have_idle_worker)) {
+               && cs->expiration_time < timeout_time) {

            cs->state = CONN_STATE_LINGER;

            APR_RING_REMOVE(cs, timeout_list);
            apr_thread_mutex_unlock(timeout_mutex);

+            if(!get_worker(&have_idle_worker)){
+                apr_thread_mutex_lock(timeout_mutex);
+                break;
+            }
            rc = push2worker(&cs->pfd, event_pollset);

            if (rc != APR_SUCCESS) {
@@ -1096,13 +1099,16 @@
        /* Step 2: write completion timeouts */
        cs = APR_RING_FIRST(&timeout_head);
        while (!APR_RING_EMPTY(&timeout_head, conn_state_t, timeout_list)
-              && cs->expiration_time < timeout_time
-              && get_worker(&have_idle_worker)) {
+               && cs->expiration_time < timeout_time) {
```

```
            cs->state = CONN_STATE_LINGER;
            APR_RING_REMOVE(cs, timeout_list);
            apr_thread_mutex_unlock(timeout_mutex);

+            if(!get_worker(&have_idle_worker)){
+                apr_thread_mutex_lock(timeout_mutex);
+                break;
+            }
            rc = push2worker(&cs->pfd, event_pollset);
            if (rc != APR_SUCCESS) {
                return NULL;
```

Question: How was the bug detected ?
Answer:

It is given that child process freezes with many downloading against MaxClients and it was detected by backtracking process. People had performed dummy setup and runs the server application on top of gdb bug detector tool.

Question: How frequently does the bug appear? Or how important is the bug?

Answer:

In the dummy setup, people had showed under 10 to 20 simultaneously client request of above 200MB file size creates deadlock.

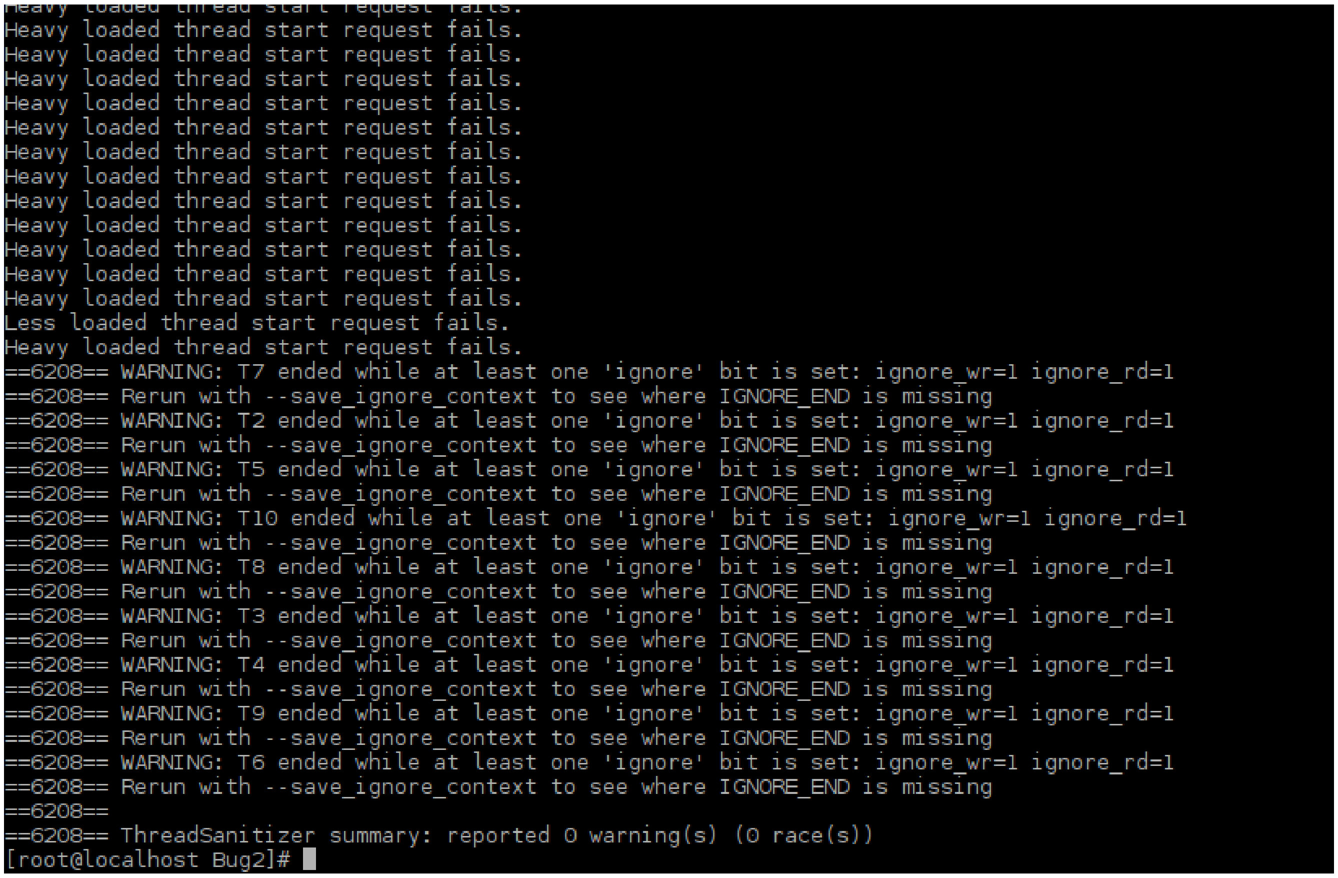
Question: Online link to the bug id
Answer:

https://issues.apache.org/bugzilla/show_bug.cgi?id=42031

Question: Generate a simple reproducer that follows the original bug
Answer:

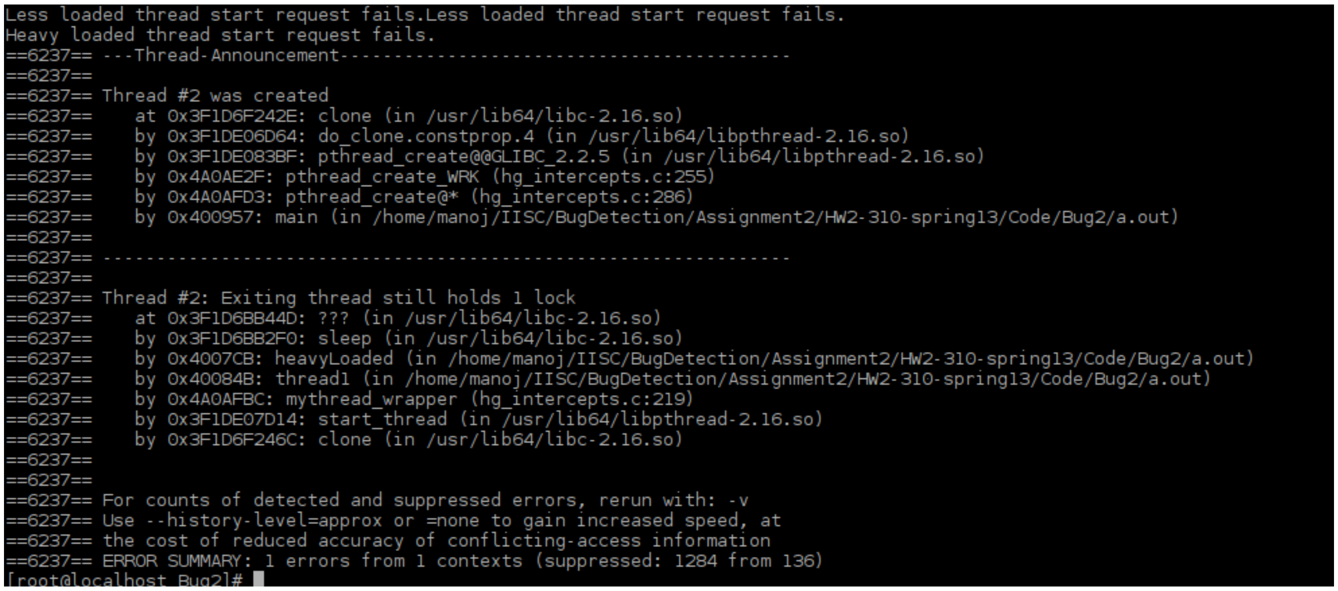
Please refer the following directory:
HW2-310-spring/Code/Bug2/bug2_42031.c

Question: Investigate whether any available program analysis tool can catch this bug automatically. If so, name the tool and a snapshot of its report.
Answer:



```
Heavy loaded thread start request fails.
Heavy loaded thread start request fails.
Heavy loaded thread start request fails.
Heavy loaded thread start request fails.
Heavy loaded thread start request fails.
Heavy loaded thread start request fails.
Heavy loaded thread start request fails.
Heavy loaded thread start request fails.
Heavy loaded thread start request fails.
Heavy loaded thread start request fails.
Heavy loaded thread start request fails.
Heavy loaded thread start request fails.
Less loaded thread start request fails.
Heavy loaded thread start request fails.
==6208== WARNING: T7 ended while at least one 'ignore' bit is set: ignore_wr=1 ignore_rd=1
==6208== Rerun with --save_ignore_context to see where IGNORE_END is missing
==6208== WARNING: T2 ended while at least one 'ignore' bit is set: ignore_wr=1 ignore_rd=1
==6208== Rerun with --save_ignore_context to see where IGNORE_END is missing
==6208== WARNING: T5 ended while at least one 'ignore' bit is set: ignore_wr=1 ignore_rd=1
==6208== Rerun with --save_ignore_context to see where IGNORE_END is missing
==6208== WARNING: T10 ended while at least one 'ignore' bit is set: ignore_wr=1 ignore_rd=1
==6208== Rerun with --save_ignore_context to see where IGNORE_END is missing
==6208== WARNING: T8 ended while at least one 'ignore' bit is set: ignore_wr=1 ignore_rd=1
==6208== Rerun with --save_ignore_context to see where IGNORE_END is missing
==6208== WARNING: T3 ended while at least one 'ignore' bit is set: ignore_wr=1 ignore_rd=1
==6208== Rerun with --save_ignore_context to see where IGNORE_END is missing
==6208== WARNING: T4 ended while at least one 'ignore' bit is set: ignore_wr=1 ignore_rd=1
==6208== Rerun with --save_ignore_context to see where IGNORE_END is missing
==6208== WARNING: T9 ended while at least one 'ignore' bit is set: ignore_wr=1 ignore_rd=1
==6208== Rerun with --save_ignore_context to see where IGNORE_END is missing
==6208== WARNING: T6 ended while at least one 'ignore' bit is set: ignore_wr=1 ignore_rd=1
==6208== Rerun with --save_ignore_context to see where IGNORE_END is missing
==6208==
==6208== ThreadSanitizer summary: reported 0 warning(s) (0 race(s))
[root@localhost Bug2]#
```

Figure 2.1: bug detected by ThreadSanitizer tool



```
Less loaded thread start request fails.Less loaded thread start request fails.
Heavy loaded thread start request fails.
==6237== ---Thread-Announcement------------------------------------------
==6237==
==6237== Thread #2 was created
==6237==    at 0x3F1D6F242E: clone (in /usr/lib64/libc-2.16.so)
==6237==    by 0x3F1DE06D64: do_clone.constprop.4 (in /usr/lib64/libpthread-2.16.so)
==6237==    by 0x3F1DE083BF: pthread_create@@GLIBC_2.2.5 (in /usr/lib64/libpthread-2.16.so)
==6237==    by 0x4A0AE2F: pthread_create_WRK (hg_intercepts.c:255)
==6237==    by 0x4A0AFD3: pthread_create@* (hg_intercepts.c:286)
==6237==    by 0x400957: main (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug2/a.out)
==6237==
==6237== ----------------------------------------------------------------
==6237==
==6237== Thread #2: Exiting thread still holds 1 lock
==6237==    at 0x3F1D6BB44D: ??? (in /usr/lib64/libc-2.16.so)
==6237==    by 0x3F1D6BB2F0: sleep (in /usr/lib64/libc-2.16.so)
==6237==    by 0x4007CB: heavyLoaded (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug2/a.out)
==6237==    by 0x40084B: thread1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug2/a.out)
==6237==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==6237==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==6237==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==6237==
==6237==
==6237== For counts of detected and suppressed errors, rerun with: -v
==6237== Use --history-level=approx or =none to gain increased speed, at
==6237== the cost of reduced accuracy of conflicting-access information
==6237== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 1284 from 136)
[root@localhost Bug2]#
```

Figure 2.2: bug detected by Valgrind –tool=helgrind

Question: Design an approach that will detect this bug automatically.
Answer:

Automatic tool wouldn't work because the above bug is not a software bug or buggy implementation. This happens because out of many, one thread is taking the lock and serving the client for so long that other thread have to wait even though scheduler switch one thread to another thread execution non of threads perform any productive tasks.

In the software patch they have introduced timeout variable that measures the time and release the lock if maximum timeout period exceeded by the particular thread execution.

Algorithm to detect deadlock bug

<u>In static analysis:</u>

1) Insert the instrumentation at the top of thread start and perform some more actions at the bottom of thread function.
2) It maintains a structure of link list in which it keeps information about, when that particular thread starts and ends that give us turn around time, serial number which give how many times a particular thread has been run or how many time it has already accessed, count of how many threads have starved when this particular thread was active, Unique thread id to identify thread and update data structure data structure at run time. At the close of instrumentation function, change the flag variable from active to inactive.

<u>In Dynamic analysis:</u>
1) When ever new thread runs, Instrumentation code will perform some checking then it search thread by its thread id in linklist, if it finds in the link list then set it active, increment the serial number and update the data structure that are connected by double link list for all the threads. All the threads can edit and modify double linklist anytime.
2) Suppose current thread not able to capture lock then it update the record of currently activated thread count value, which counts how many threads were waiting when current thread was performing the task.
3) At every start of the thread, it can be calculated that whether it is costly thread or normal thread, here costly thread means when it consume more time to complete task.

Costly factor can be calculated by = Total number of threads/ count value in which how many other threads starved at the time of current thread execution.

4) In this way, this bug can be detect and report warning message in the detection console.

Question: Does your approach compare with any existing approach? If so, how?
Answer:

No. The above algorithm will not be generic but specific to particular application.

Question: Mention any drawbacks of your approach. Present clear examples where these drawbacks will manifest?

Answer:

I am using double linklist and all the threads have access to change/modify or delete records (structure with double linklist) so here we need to make our program race and deadlock free. It program is badly written then tool may run out of memory. For example: Suppose user create many threads and does not close the thread properly, in that case our record size will increase gradually and at some point of time it may be run out of memory.

Question: What were the reasons that you could not adjust your approach to handle the above mentioned drawbacks?

Answer:

With my approach i can detect the above bug, but there is some implementation limitations and even i don't have much experience whether this type of instrumentation is possible or not.

Bug No. 3:

Question: Description of the bug including details on how it manifests
Answer:

This bug comes under data race. It occurs when apache server is shutting down and in the same time, it tries to log this information in log file that server has been successfully shutdown at a certain time and date, but sometime it fails because the mutex value has been cleaned up or reset by other threads. It then tries to log this error to an error log file, but the file handle has been invalidated by this time.

The problem happens on shutdown, but never at start-up because at shutdown other thread may resets the value as well as locking variables and if main thread tries to read some shared variable that is already reset by other thread, then it generates segmentation fault and crash the system.

Question: Patch (difference in code) that was used to fix the bug.
Answer:

To fix this problem it should check the mutex and file handler variable before use.

```
Index: readwrite.c
===================================================================
RCS file: /home/cvspublic/apr/file_io/win32/readwrite.c,v
retrieving revision 1.79
diff -u -r1.79 readwrite.c
--- readwrite.c 1 May 2003 03:16:30 -0000       1.79
+++ readwrite.c 18 Dec 2003 16:07:31 -0000
@@ -304,12 +304,18 @@
         apr_off_t offset = 0;
         apr_status_t rc;
         if (thefile->append) {
+            if (thefile->filehand == (HANDLE)0 || thefile->filehand == (HANDLE)-1) {
+                return (APR_OS_START_SYSERR + ERROR_INVALID_HANDLE);
+            }
             /* apr_file_lock will mutex the file across processes.
              * The call to apr_thread_mutex_lock is added to avoid
              * a race condition between LockFile and WriteFile
              * that occasionally leads to deadlocked threads.
              */
-            apr_thread_mutex_lock(thefile->mutex);
+            rc = apr_thread_mutex_lock(thefile->mutex);
+            if (rc != APR_SUCCESS) {
+                return rc;
+            }
             rc = apr_file_lock(thefile, APR_FLOCK_EXCLUSIVE);
             if (rc != APR_SUCCESS) {
                 apr_thread_mutex_unlock(thefile->mutex);
```

Question: How was the bug detected ?
Answer:

At the shutdown, most of the time server got crashed then in the debug process, developers come up with the conclusion that few of the shared variables value are reset by other threads and when current thread trying to use that variables then it encounters segmentation fault.

Question: How frequently does the bug appear? Or how important is the bug?

Answer:

At the shutdown process if mutex and file handler values are null. This is not so important bug as it was happening at shutdown (In my opinion)

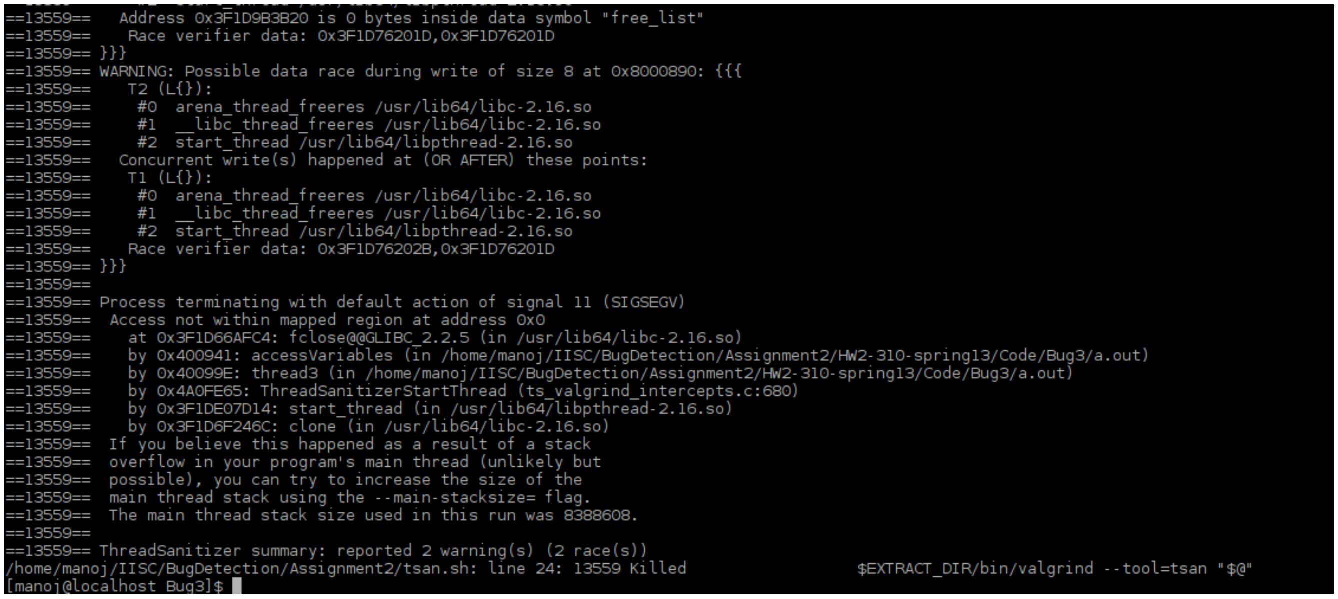
Question: Online link to the bug id
Answer:

https://issues.apache.org/bugzilla/show_bug.cgi?id=25575

Question: Generate a simple reproducer that follows the original bug
Answer:

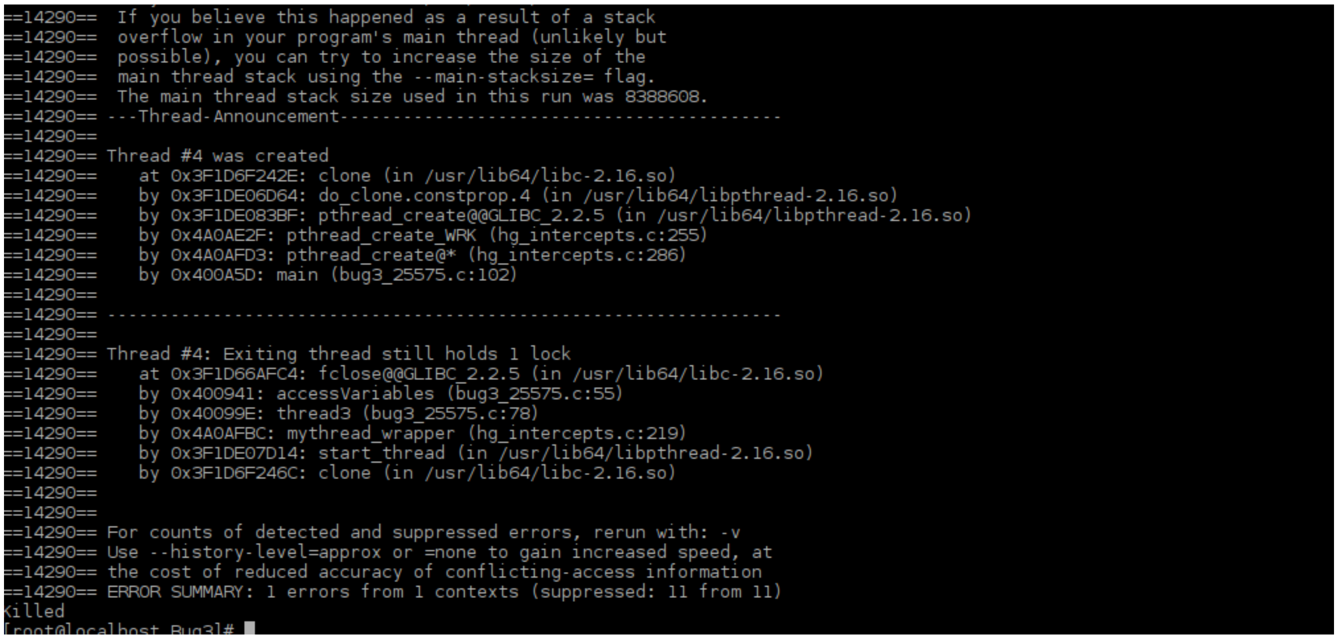
Please refer the following directory:
HW2-310-spring/Code/Bug3/bug3_25575.c

Question: Investigate whether any available program analysis tool can catch this bug automatically. If so, name the tool and a snapshot of its report.
Answer:



```
==13559==   Address 0x3F1D9B3B20 is 0 bytes inside data symbol "free_list"
==13559==    Race verifier data: 0x3F1D76201D,0x3F1D76201D
==13559== }}}
==13559== WARNING: Possible data race during write of size 8 at 0x8000890: {{{
==13559==    T2 (L{}):
==13559==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==13559==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==13559==     #2  start_thread /usr/lib64/libpthread-2.16.so
==13559==   Concurrent write(s) happened at (OR AFTER) these points:
==13559==    T1 (L{}):
==13559==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==13559==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==13559==     #2  start_thread /usr/lib64/libpthread-2.16.so
==13559==    Race verifier data: 0x3F1D76202B,0x3F1D76201D
==13559== }}}
==13559==
==13559== Process terminating with default action of signal 11 (SIGSEGV)
==13559==  Access not within mapped region at address 0x0
==13559==    at 0x3F1D66AFC4: fclose@@GLIBC_2.2.5 (in /usr/lib64/libc-2.16.so)
==13559==    by 0x400941: accessVariables (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug3/a.out)
==13559==    by 0x40099E: thread3 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug3/a.out)
==13559==    by 0x4A0FE65: ThreadSanitizerStartThread (ts_valgrind_intercepts.c:680)
==13559==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==13559==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==13559==  If you believe this happened as a result of a stack
==13559==  overflow in your program's main thread (unlikely but
==13559==  possible), you can try to increase the size of the
==13559==  main thread stack using the --main-stacksize= flag.
==13559==  The main thread stack size used in this run was 8388608.
==13559==
==13559== ThreadSanitizer summary: reported 2 warning(s) (2 race(s))
/home/manoj/IISC/BugDetection/Assignment2/tsan.sh: line 24: 13559 Killed                  $EXTRACT_DIR/bin/valgrind --tool=tsan "$@"
[manoj@localhost Bug3]$
```

Figure 3.1: bug detected by ThreadSanitizer tool



```
==14290==  If you believe this happened as a result of a stack
==14290==  overflow in your program's main thread (unlikely but
==14290==  possible), you can try to increase the size of the
==14290==  main thread stack using the --main-stacksize= flag.
==14290==  The main thread stack size used in this run was 8388608.
==14290== ---Thread-Announcement------------------------------------------
==14290==
==14290== Thread #4 was created
==14290==    at 0x3F1D6F242E: clone (in /usr/lib64/libc-2.16.so)
==14290==    by 0x3F1DE06D64: do_clone.constprop.4 (in /usr/lib64/libpthread-2.16.so)
==14290==    by 0x3F1DE083BF: pthread_create@@GLIBC_2.2.5 (in /usr/lib64/libpthread-2.16.so)
==14290==    by 0x4A0AE2F: pthread_create_WRK (hg_intercepts.c:255)
==14290==    by 0x4A0AFD3: pthread_create@* (hg_intercepts.c:286)
==14290==    by 0x400A5D: main (bug3_25575.c:102)
==14290==
==14290== ----------------------------------------------------------------
==14290==
==14290== Thread #4: Exiting thread still holds 1 lock
==14290==    at 0x3F1D66AFC4: fclose@@GLIBC_2.2.5 (in /usr/lib64/libc-2.16.so)
==14290==    by 0x400941: accessVariables (bug3_25575.c:55)
==14290==    by 0x40099E: thread3 (bug3_25575.c:78)
==14290==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==14290==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==14290==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==14290==
==14290==
==14290== For counts of detected and suppressed errors, rerun with: -v
==14290== Use --history-level=approx or =none to gain increased speed, at
==14290== the cost of reduced accuracy of conflicting-access information
==14290== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 11 from 11)
Killed
[root@localhost Bug3]#
```

Figure 3.2: bug detected by Valgrind –tool=helgrind

Question: Design an approach that will detect this bug automatically.

Answer:

This is the problem of data race and hence it can be detected by Static analysis

```
If( multiple_pthread_create_not_exist)
        return( no_data_race ) // If program is not multithread
else{
```

1) Analysis the functions that are called inside the pthread_create and store the shared variables into a set.

2) Calculate intersection of different sets and if intersection value is not null between two sets and those shared variables are not protected by any lock then generates warning for data race.

}

but there is a situation in which variables are freed by other threads and when some other thread tries to access that location then it generates segmentation faults. This type of situation can be handle by dynamic analysis. But my doubt is that how program analysis tool or algorithm finds whether memory location is valid or invalid so it is hard to determine memory access is legal or illegal at run time by dynamic analysis tool.

Question: Does your approach compare with any existing approach? If so, how?

Answer:

No

Question: Mention any drawbacks of your approach. Present clear examples where these drawbacks will manifest?

Answer:

This approach finds data race but don't find illegal memory access.

Question: What were the reasons that you could not adjust your approach to handle the above mentioned drawbacks?

Answer:

Algorithm doesn't work on illegal memory access.

Bug No. 4:

Question: Description of the bug including details on how it manifests
Answer:

This bug comes under race condition. It occurs when file handler points to some particular file and in the mean while that file is replaced by some other file with different file size but same name. This happens between the stat and the open calls, as the result either file data will be truncated or generates segmentation fault because of illegal memory access.

File data truncates when new file size is more than old file size and segmentation faults when new file size is less than old file size.

Question: Patch (difference in code) that was used to fix the bug.
Answer:

```
Index: server/core.c
===================================================================
--- server/core.c       (revision 574858)
+++ server/core.c       (working copy)
@@ -3608,6 +3608,12 @@
             return HTTP_FORBIDDEN;
         }

+        if ((status = apr_file_info_get(&r->finfo, APR_FINFO_MIN, fd) ) != APR_SUCCESS) {
+            ap_log_rerror(APLOG_MARK, APLOG_ERR, status, r,
+                    "unable to read from fd of : %s", r->filename);
+            apr_file_close(fd);
+            return HTTP_INTERNAL_SERVER_ERROR;
+        }
         ap_update_mtime(r, r->finfo.mtime);
         ap_set_last_modified(r);
         ap_set_etag(r);
```

Question: How was the bug detected ?
Answer:

People had specified that they had requirement to update some file periodically at the web server and whenever both operation which is read and write takes place simultaneously then this bug may be appears.

Question: How frequently does the bug appear? Or how important is the bug?

Answer:

This bug occurs when worker thread is serving to the client and in the mean while requested file is replaced by some other file at the server side.

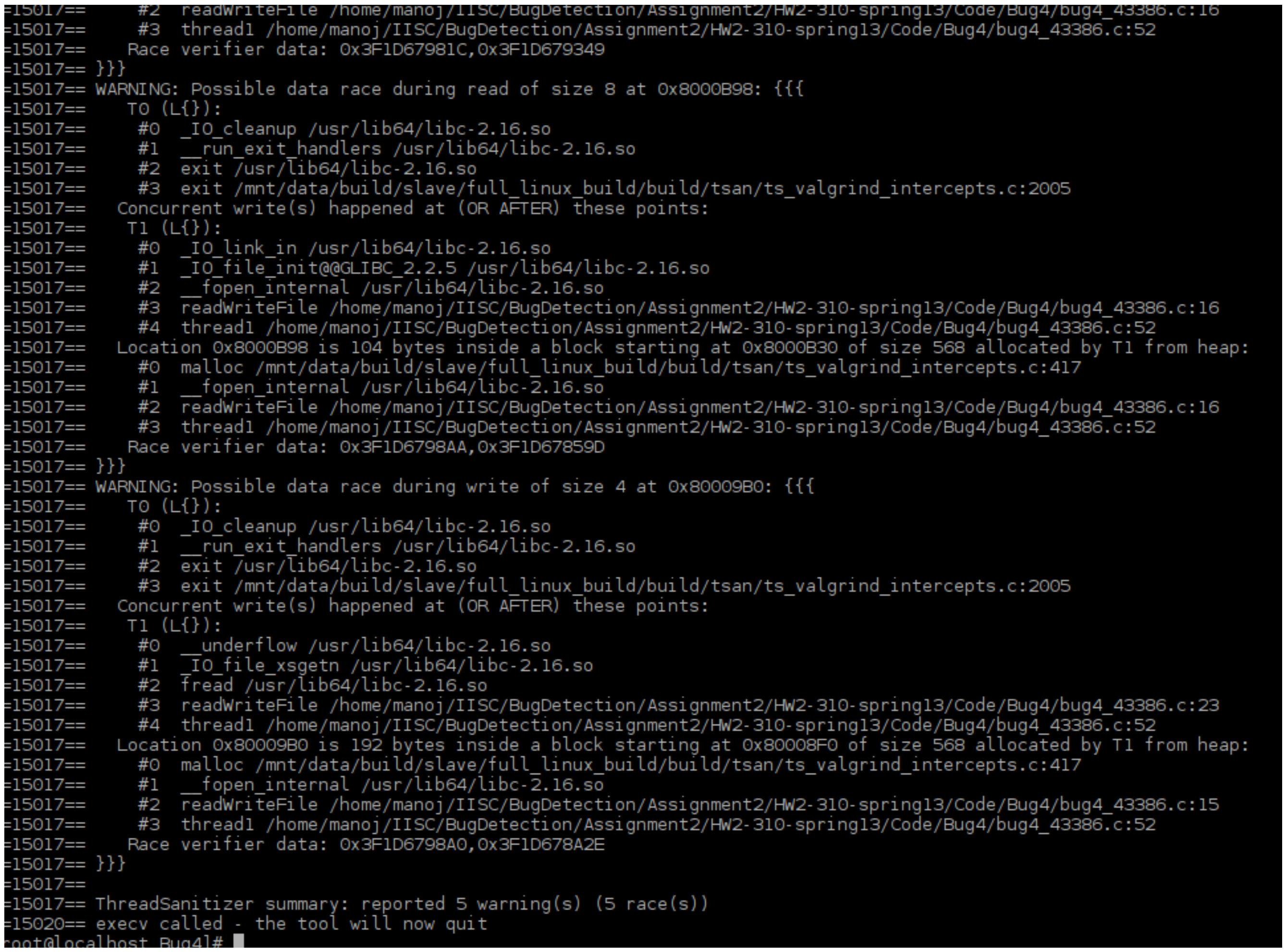
Question: Online link to the bug id
Answer:

https://issues.apache.org/bugzilla/show_bug.cgi?id=43386

Question: Generate a simple reproducer that follows the original bug
Answer:

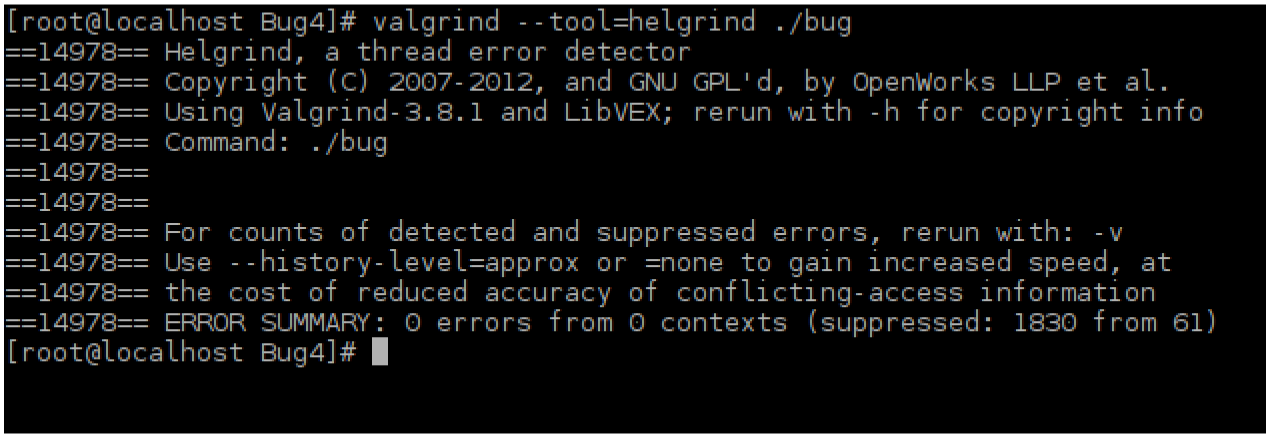
Please refer the following directory:
HW2-310-spring/Code/Bug4/bug4_43386.c

Question: Investigate whether any available program analysis tool can catch this bug automatically. If so, name the tool and a snapshot of its report.
Answer:



```
=15017==     #2  readWriteFile /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug4/bug4_43386.c:16
=15017==     #3  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug4/bug4_43386.c:52
=15017==    Race verifier data: 0x3F1D67981C,0x3F1D679349
=15017== }}}
=15017== WARNING: Possible data race during read of size 8 at 0x8000B98: {{{
=15017==    T0 (L{}):
=15017==     #0  _IO_cleanup /usr/lib64/libc-2.16.so
=15017==     #1  __run_exit_handlers /usr/lib64/libc-2.16.so
=15017==     #2  exit /usr/lib64/libc-2.16.so
=15017==     #3  exit /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:2005
=15017==   Concurrent write(s) happened at (OR AFTER) these points:
=15017==    T1 (L{}):
=15017==     #0  _IO_link_in /usr/lib64/libc-2.16.so
=15017==     #1  _IO_file_init@@GLIBC_2.2.5 /usr/lib64/libc-2.16.so
=15017==     #2  __fopen_internal /usr/lib64/libc-2.16.so
=15017==     #3  readWriteFile /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug4/bug4_43386.c:16
=15017==     #4  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug4/bug4_43386.c:52
=15017==   Location 0x8000B98 is 104 bytes inside a block starting at 0x8000B30 of size 568 allocated by T1 from heap:
=15017==     #0  malloc /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:417
=15017==     #1  __fopen_internal /usr/lib64/libc-2.16.so
=15017==     #2  readWriteFile /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug4/bug4_43386.c:16
=15017==     #3  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug4/bug4_43386.c:52
=15017==    Race verifier data: 0x3F1D6798AA,0x3F1D67859D
=15017== }}}
=15017== WARNING: Possible data race during write of size 4 at 0x80009B0: {{{
=15017==    T0 (L{}):
=15017==     #0  _IO_cleanup /usr/lib64/libc-2.16.so
=15017==     #1  __run_exit_handlers /usr/lib64/libc-2.16.so
=15017==     #2  exit /usr/lib64/libc-2.16.so
=15017==     #3  exit /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:2005
=15017==   Concurrent write(s) happened at (OR AFTER) these points:
=15017==    T1 (L{}):
=15017==     #0  __underflow /usr/lib64/libc-2.16.so
=15017==     #1  _IO_file_xsgetn /usr/lib64/libc-2.16.so
=15017==     #2  fread /usr/lib64/libc-2.16.so
=15017==     #3  readWriteFile /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug4/bug4_43386.c:23
=15017==     #4  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug4/bug4_43386.c:52
=15017==   Location 0x80009B0 is 192 bytes inside a block starting at 0x80008F0 of size 568 allocated by T1 from heap:
=15017==     #0  malloc /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:417
=15017==     #1  __fopen_internal /usr/lib64/libc-2.16.so
=15017==     #2  readWriteFile /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug4/bug4_43386.c:15
=15017==     #3  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug4/bug4_43386.c:52
=15017==    Race verifier data: 0x3F1D6798A0,0x3F1D678A2E
=15017== }}}
=15017==
=15017== ThreadSanitizer summary: reported 5 warning(s) (5 race(s))
=15020== execv called - the tool will now quit
root@localhost Bug4]#
```

Figure 4.1: bug detected by ThreadSanitizer tool



```
[root@localhost Bug4]# valgrind --tool=helgrind ./bug
==14978== Helgrind, a thread error detector
==14978== Copyright (C) 2007-2012, and GNU GPL'd, by OpenWorks LLP et al.
==14978== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==14978== Command: ./bug
==14978==
==14978==
==14978== For counts of detected and suppressed errors, rerun with: -v
==14978== Use --history-level=approx or =none to gain increased speed, at
==14978== the cost of reduced accuracy of conflicting-access information
==14978== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 1830 from 61)
[root@localhost Bug4]#
```

Figure 4.2: bug detected by Valgrind –tool=helgrind

Bug No. 5:

Question: Description of the bug including details on how it manifests
Answer:

This bug comes under data race. It occurs when variable mWriteThread which is called in function WaitOnWriteThread() was used in many threads and a shared variable mWriteThread was not protected by any kind of lock and at the end of the function it was reinitialize to null so in the mean while if there is any thread which is currently accessing that variable gets segmentation fault and program gets crash.

Question: Patch (difference in code) that was used to fix the bug.
Answer:

```
# HG changeset patch
# Parent 54a3038f2ca0a11f95b01432114af28dfcd88aab
# User Taras Glek <tglek@mozilla.com>

diff --git a/startupcache/StartupCache.cpp b/startupcache/StartupCache.cpp
--- a/startupcache/StartupCache.cpp
+++ b/startupcache/StartupCache.cpp
@@ -65,7 +65,9 @@
 #include "mozilla/Omnijar.h"
 #include "prenv.h"
 #include "mozilla/FunctionTimer.h"
-
+#include "nsThreadUtils.h"
+#include "nsXULAppAPI.h"
+
 #ifdef IS_BIG_ENDIAN
 #define SC_ENDIAN "big"
 #else
@@ -126,11 +128,9 @@ StartupCache::~StartupCache()
   }

   // Generally, the in-memory table should be empty here,
-  // but in special cases (like Talos Ts tests) we
-  // could shut down before we write.
-  // This mechanism will change when IO is moved off-thread
-  // (bug 586859) or when Talos first-run is changed to allow
-  // our timer to work (bug 591471).
+  // but an early shutdown means either mTimer didn't run
+  // or the write thread is still running.
+  WaitOnWriteThread();
   WriteToDisk();
   gStartupCache = nsnull;
 }
@@ -138,6 +138,11 @@ StartupCache::~StartupCache()
```

```
 nsresult
 StartupCache::Init()
 {
+  if (XRE_GetProcessType() != GeckoProcessType_Default) {
+    NS_WARNING("Startup cache is only available in the chrome process");
+    return NS_ERROR_NOT_AVAILABLE;
+  }
+
   nsresult rv;
   mTable.Init();
 #ifdef DEBUG
@@ -201,10 +206,12 @@ StartupCache::Init()
   return NS_OK;
 }

+/**
+ * LoadArchive can be called from the main thread or while reloading cache on write thread.
+ */
 nsresult
 StartupCache::LoadArchive()
 {
-  WaitOnWriteThread();
   PRBool exists;
   mArchive = NULL;
   nsresult rv = mFile->Exists(&exists);
@@ -220,6 +227,7 @@ StartupCache::LoadArchive()
 nsresult
 StartupCache::GetBuffer(const char* id, char** outbuf, PRUint32* length)
 {
+  NS_ASSERTION(NS_IsMainThread(), "Startup cache only available on main thread");
   WaitOnWriteThread();
   if (!mStartupWriteInitiated) {
     CacheEntry* entry;
@@ -260,6 +268,7 @@ StartupCache::GetBuffer(const char* id,
 nsresult
 StartupCache::PutBuffer(const char* id, const char* inbuf, PRUint32 len)
 {
+  NS_ASSERTION(NS_IsMainThread(), "Startup cache only available on main thread");
   WaitOnWriteThread();
   if (StartupCache::gShutdownInitiated) {
     return NS_ERROR_NOT_AVAILABLE;
@@ -320,10 +329,14 @@ CacheCloseHelper(const nsACString& key,
   return PL_DHASH_REMOVE;
 }

+
+/**
+ * WriteToDisk writes the cache out to disk. Callers of WriteToDisk need to call WaitOnWriteThread
+ * to make sure there isn't a write happening on another thread
+ */
 void
 StartupCache::WriteToDisk()
 {
-  WaitOnWriteThread();
   nsresult rv;
   mStartupWriteInitiated = PR_TRUE;

@@ -382,13 +395,12 @@ StartupCache::InvalidateCache()
 void
 StartupCache::WaitOnWriteThread()
 {
-  PRThread* writeThread = mWriteThread;
-  if (!writeThread || writeThread == PR_GetCurrentThread())
+  NS_ASSERTION(NS_IsMainThread(), "Startup cache should only wait for io thread on main
thread");
+  if (!mWriteThread || mWriteThread == PR_GetCurrentThread())
     return;

   NS_TIME_FUNCTION_MIN(30);
-  //NS_WARNING("Waiting on startupcache write");
-  PR_JoinThread(writeThread);
+  PR_JoinThread(mWriteThread);
   mWriteThread = NULL;
 }
```

Please read more about above functions see below:

NS_ASSERTION: It takes input as a function and corresponding error message.
https://developer.mozilla.org/en/docs/NS_ASSERTION

NS_IsMainThread: It ensure that callers only use StartupCache on the main thread. This function is giving mutual exclusion in the above patch.

Question: How was the bug detected ?
Answer:

It was detected with the help of bug detection tool which is known as valgrind –tool=helgrind and example is copied from the bug site and pasted below

```
==22869== Possible data race during write of size 8 at 0x162f6d10 by thread #1
==22869==    at 0x5739380: mozilla::scache::StartupCache::WriteTimeout(nsITimer*,
void*) (StartupCache.cpp:415)
==22869==    by 0x63A37C4: nsTimerImpl::Fire() (nsTimerImpl.cpp:425)
==22869==    by 0x63A38A8: nsTimerEvent::Run() (nsTimerImpl.cpp:517)
==22869==    by 0x639FFFC: nsThread::ProcessNextEvent(int, int*)
(nsThread.cpp:633)
==22869==    by 0x63604F9: NS_ProcessNextEvent_P(nsIThread*, int)
(nsThreadUtils.cpp:250)
==22869==    by 0x6259882:
mozilla::ipc::MessagePump::Run(base::MessagePump::Delegate*) (MessagePump.cpp:110)
==22869==    by 0x63E0ECA: MessageLoop::RunInternal() (message_loop.cc:219)
==22869==    by 0x63E0ED6: MessageLoop::RunHandler() (message_loop.cc:202)
==22869==    by 0x63E0F45: MessageLoop::Run() (message_loop.cc:176)
==22869==    by 0x6178787: nsBaseAppShell::Run() (nsBaseAppShell.cpp:192)
==22869==    by 0x5FE21D4: nsAppStartup::Run() (nsAppStartup.cpp:220)
==22869==    by 0x563AC11: XRE_main (nsAppRunner.cpp:3781)
==22869==    by 0x400F7E: main (nsBrowserApp.cpp:158)

==22869==  This conflicts with a previous read of size 8 by thread #26
==22869==    at 0x573939B: mozilla::scache::StartupCache::WaitOnWriteThread()
(StartupCache.cpp:385)
==22869==    by 0x573960C: mozilla::scache::StartupCache::WriteToDisk()
(StartupCache.cpp:326)
==22869==    by 0x57397DF: mozilla::scache::StartupCache::ThreadedWrite(void*)
(StartupCache.cpp:398)
==22869==    by 0x7A43A1C: _pt_root (ptthread.c:189)
==22869==    by 0x4C2CBE7: mythread_wrapper (hg_intercepts.c:221)
==22869==    by 0x4E369C9: start_thread (pthread_create.c:300)
==22869==    by 0xB2DA70C: clone (clone.S:112)
```

Question: How frequently does the bug appear? Or how important is the bug?

Answer:

StartupCache class initializes the cache variables and write cache data into the disk by calling the WriteToDisk() and inside WriteToDisk() function it uses WaitOnWriteThread() function and WaitOnWriteThread() usually used shared variable to keep current thread id and which was reset to null at the end of WaitOnWriteThread() function. This bug occurs whenever multiple thread tries to access same shared variable which is known as mWriteThread. This leads to generate segmentation fault if mWriteThread is pointing to NULL.

This bug occurs whenever WriteToDisk() function call since WriteToDisk() function is very frequent so i guess this bug occurs very frequently in the multithreaded environment.

Question: Online link to the bug id

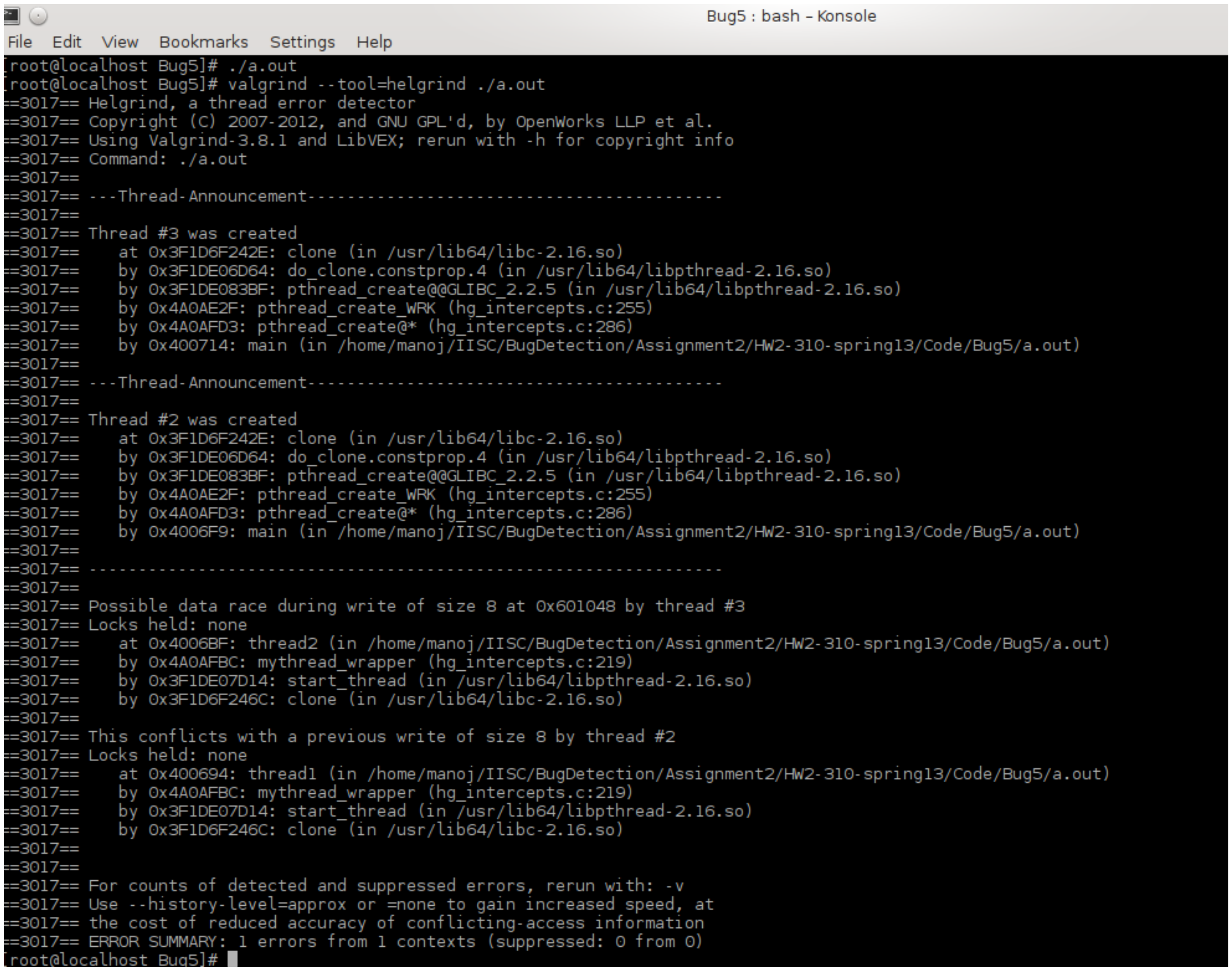
Answer:

https://bugzilla.mozilla.org/show_bug.cgi?id=637461

Question: Generate a simple reproducer that follows the original bug
Answer:

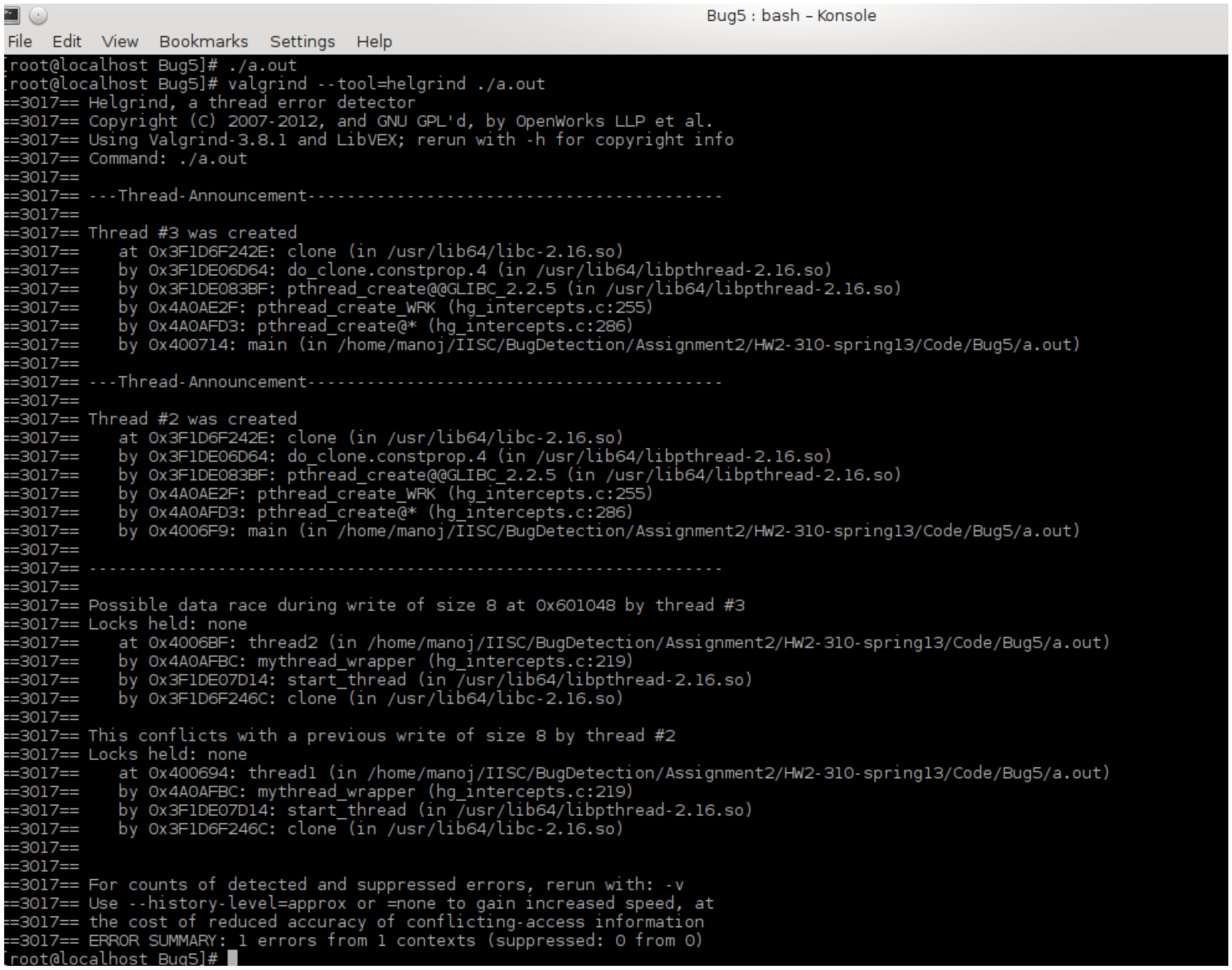
Please refer the following directory:
HW2-310-spring/Code/Bug5/bug5_637461.c

Question: Investigate whether any available program analysis tool can catch this bug automatically. If so, name the tool and a snapshot of its report.
Answer:



```
[root@localhost Bug5]# ./a.out
[root@localhost Bug5]# valgrind --tool=helgrind ./a.out
==3017== Helgrind, a thread error detector
==3017== Copyright (C) 2007-2012, and GNU GPL'd, by OpenWorks LLP et al.
==3017== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==3017== Command: ./a.out
==3017==
==3017== ---Thread-Announcement------------------------------------------
==3017==
==3017== Thread #3 was created
==3017==    at 0x3F1D6F242E: clone (in /usr/lib64/libc-2.16.so)
==3017==    by 0x3F1DE06D64: do_clone.constprop.4 (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x3F1DE083BF: pthread_create@@GLIBC_2.2.5 (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x4A0AE2F: pthread_create_WRK (hg_intercepts.c:255)
==3017==    by 0x4A0AFD3: pthread_create@* (hg_intercepts.c:286)
==3017==    by 0x400714: main (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug5/a.out)
==3017==
==3017== ---Thread-Announcement------------------------------------------
==3017==
==3017== Thread #2 was created
==3017==    at 0x3F1D6F242E: clone (in /usr/lib64/libc-2.16.so)
==3017==    by 0x3F1DE06D64: do_clone.constprop.4 (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x3F1DE083BF: pthread_create@@GLIBC_2.2.5 (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x4A0AE2F: pthread_create_WRK (hg_intercepts.c:255)
==3017==    by 0x4A0AFD3: pthread_create@* (hg_intercepts.c:286)
==3017==    by 0x4006F9: main (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug5/a.out)
==3017==
==3017== ----------------------------------------------------------------
==3017==
==3017== Possible data race during write of size 8 at 0x601048 by thread #3
==3017== Locks held: none
==3017==    at 0x4006BF: thread2 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug5/a.out)
==3017==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==3017==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==3017==
==3017== This conflicts with a previous write of size 8 by thread #2
==3017== Locks held: none
==3017==    at 0x400694: thread1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug5/a.out)
==3017==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==3017==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==3017==
==3017==
==3017== For counts of detected and suppressed errors, rerun with: -v
==3017== Use --history-level=approx or =none to gain increased speed, at
==3017== the cost of reduced accuracy of conflicting-access information
==3017== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
[root@localhost Bug5]#
```

Figure 5.1: bug detected by ThreadSanitizer tool



```
[root@localhost Bug5]# ./a.out
[root@localhost Bug5]# valgrind --tool=helgrind ./a.out
==3017== Helgrind, a thread error detector
==3017== Copyright (C) 2007-2012, and GNU GPL'd, by OpenWorks LLP et al.
==3017== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==3017== Command: ./a.out
==3017==
==3017== ---Thread-Announcement------------------------------------------
==3017==
==3017== Thread #3 was created
==3017==    at 0x3F1D6F242E: clone (in /usr/lib64/libc-2.16.so)
==3017==    by 0x3F1DE06D64: do_clone.constprop.4 (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x3F1DE083BF: pthread_create@@GLIBC_2.2.5 (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x4A0AE2F: pthread_create_WRK (hg_intercepts.c:255)
==3017==    by 0x4A0AFD3: pthread_create@* (hg_intercepts.c:286)
==3017==    by 0x400714: main (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug5/a.out)
==3017==
==3017== ---Thread-Announcement------------------------------------------
==3017==
==3017== Thread #2 was created
==3017==    at 0x3F1D6F242E: clone (in /usr/lib64/libc-2.16.so)
==3017==    by 0x3F1DE06D64: do_clone.constprop.4 (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x3F1DE083BF: pthread_create@@GLIBC_2.2.5 (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x4A0AE2F: pthread_create_WRK (hg_intercepts.c:255)
==3017==    by 0x4A0AFD3: pthread_create@* (hg_intercepts.c:286)
==3017==    by 0x4006F9: main (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug5/a.out)
==3017==
==3017== ----------------------------------------------------------------
==3017==
==3017== Possible data race during write of size 8 at 0x601048 by thread #3
==3017== Locks held: none
==3017==    at 0x4006BF: thread2 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug5/a.out)
==3017==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==3017==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==3017==
==3017== This conflicts with a previous write of size 8 by thread #2
==3017== Locks held: none
==3017==    at 0x400694: thread1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug5/a.out)
==3017==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==3017==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==3017==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==3017==
==3017==
==3017== For counts of detected and suppressed errors, rerun with: -v
==3017== Use --history-level=approx or =none to gain increased speed, at
==3017== the cost of reduced accuracy of conflicting-access information
==3017== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
[root@localhost Bug5]#
```

Figure 5.2: bug detected by Valgrind –tool=helgrind

Bug No. 6:

Question: Description of the bug including details on how it manifests
Answer:

This bug occurs whenever multiple thread tries to write log information into a log file. Actually developers tried to optimized IO operating by keep the log information in a buffer, rather writing into a file directly. When the buffer size exceeds max buffer length, it writes into a file, reset buffer and buffer count variable.

Data race occurs when one thread reads used buffer size and other thread updates the used buffer size.

Atomicity violating occurs when first thread reads buffer used size and if buffer used size is not equal to or greater than MAX log size, then it starts writing on the buffer and once it completes writing, it updates the used buffer size, but in the interleaving of other thread which checks the size of used buffer variable and do same operations, hence that leads to Atomicity violation.

Illegal Memory access occurs when first thread reads buffer used size and compares whether that memory is sufficient to store the log data into the buffer, but in the mean time other thread fills some data into buffer and increments used buffer size, now when the first thread tries to write log data into buffer, then it crosses the buffer max size, as a result, system generates segmentation fault error.

Question: Patch (difference in code) that was used to fix the bug.
Answer:

```
typedef struct {
    apr_file_t *handle;
    apr_size_t outcnt;
    char outbuf[LOG_BUFSIZE];
+   apr_anylock_t mutex;
} buffered_log;
static apr_status_t ap_buffered_log_writer(request_rec *r,
                                           void *handle,
                                           const char **strs,
                                           int *strl,
                                           int nelts,
                                           apr_size_t len)
{
    char *str;
    char *s;
    int i;
    apr_status_t rv;
    buffered_log *buf = (buffered_log*)handle;

 +   if ((rv = APR_ANYLOCK_LOCK(&buf->mutex)) != APR_SUCCESS) {
 +       return rv;
 +   }

    if (len + buf->outcnt > LOG_BUFSIZE) {
        flush_log(buf);
    }
    if (len >= LOG_BUFSIZE) {
        apr_size_t w;
        str = apr_palloc(r->pool, len + 1);
        for (i = 0, s = str; i < nelts; ++i) {
            memcpy(s, strs[i], strl[i]);
            s += strl[i];
        }
        w = len;
        rv = apr_file_write(buf->handle, str, &w);
    }
    else {
        for (i = 0, s = &buf->outbuf[buf->outcnt]; i < nelts; ++i) {
            memcpy(s, strs[i], strl[i]);
            s += strl[i];
        }
        buf->outcnt += len;
        rv = APR_SUCCESS;
    }
    // Unlocking
+   APR_ANYLOCK_UNLOCK(&buf->mutex);
    return rv;
}
```

Question: How was the bug detected ?

Answer:

It occurs when web server writes log data into disk and some time log data are truncated or write arbitrarily garbage data because of change in the buffer length by some other thread. People have seen that log data was not proper and in the debugging process it was detected.

Question: How frequently does the bug appear? Or how important is the bug?

Answer:

When ever server writes log data into disk and in the mean while some other thread alter shared variables that produce inconsistency in the log record.

Question: Online link to the bug id

Answer:

https://issues.apache.org/bugzilla/show_bug.cgi?id=25520

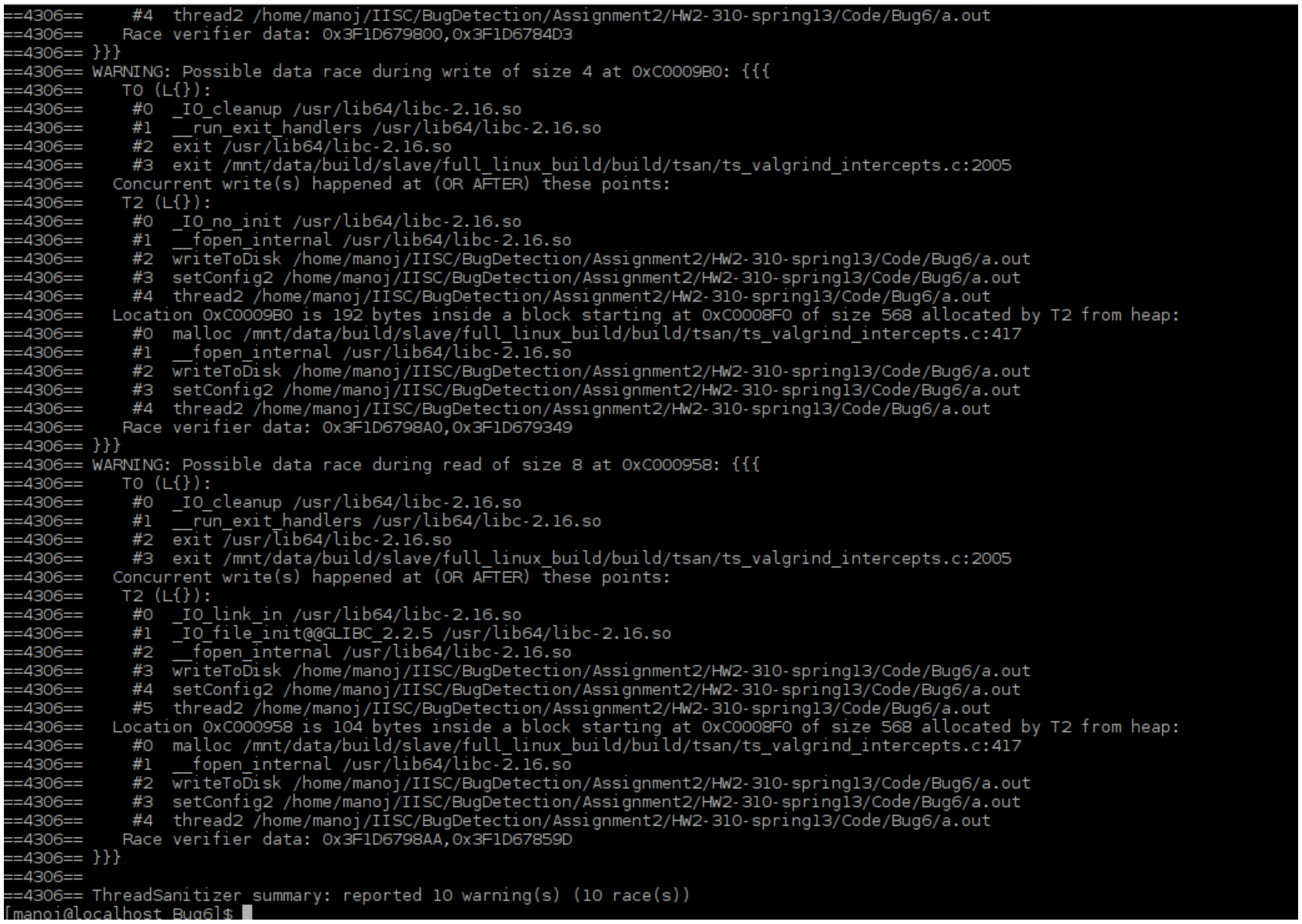
Question: Generate a simple reproducer that follows the original bug

Answer:

Please refer the following directory:
HW2-310-spring/Code/Bug6/bug6_25520.c

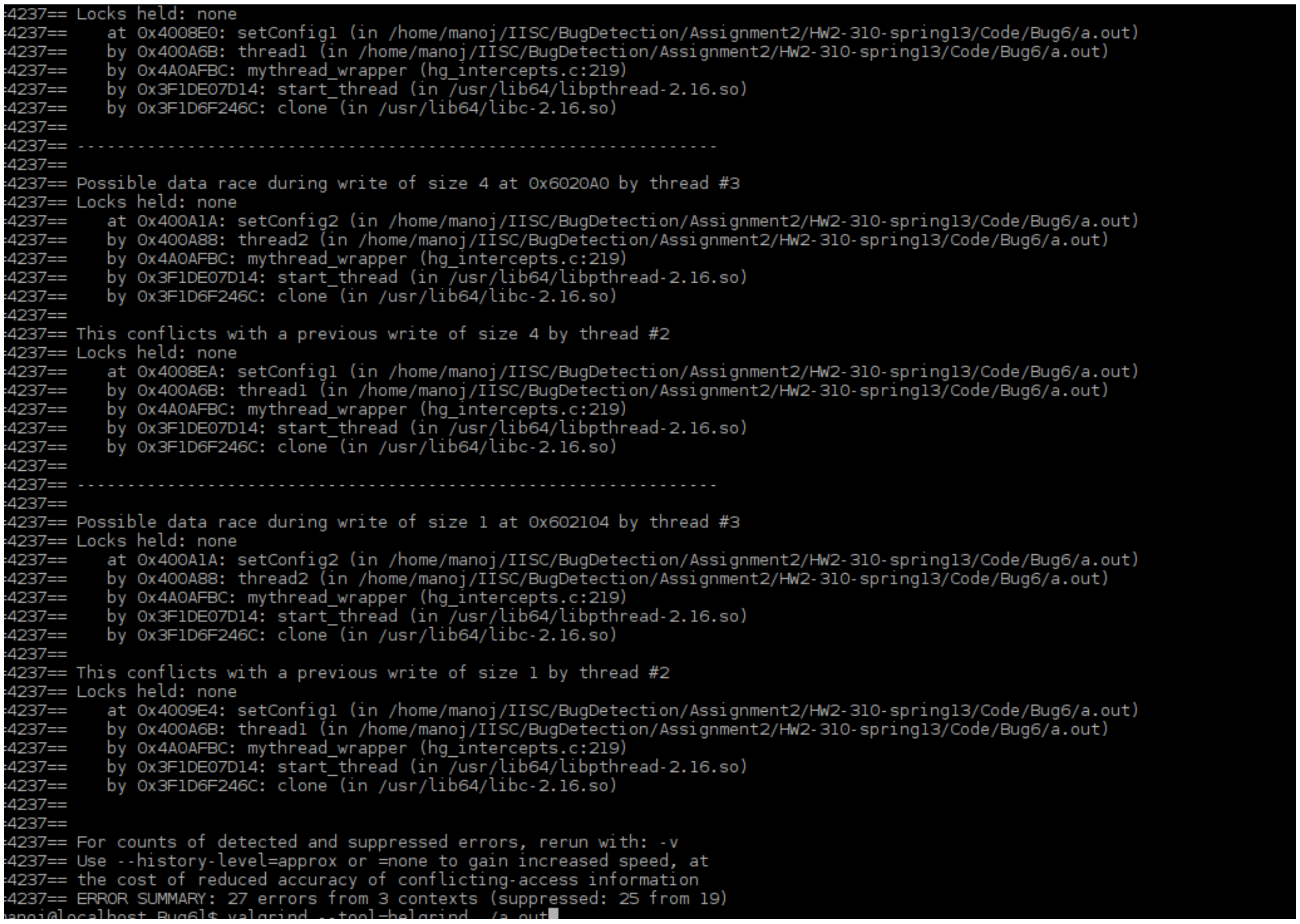
Question: Investigate whether any available program analysis tool can catch this bug automatically. If so, name the tool and a snapshot of its report.

Answer:



```
==4306==     #4  thread2 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==    Race verifier data: 0x3F1D679800,0x3F1D6784D3
==4306== }}}
==4306== WARNING: Possible data race during write of size 4 at 0xC0009B0: {{{
==4306==    T0 (L{}):
==4306==     #0  _IO_cleanup /usr/lib64/libc-2.16.so
==4306==     #1  __run_exit_handlers /usr/lib64/libc-2.16.so
==4306==     #2  exit /usr/lib64/libc-2.16.so
==4306==     #3  exit /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:2005
==4306==   Concurrent write(s) happened at (OR AFTER) these points:
==4306==    T2 (L{}):
==4306==     #0  _IO_no_init /usr/lib64/libc-2.16.so
==4306==     #1  __fopen_internal /usr/lib64/libc-2.16.so
==4306==     #2  writeToDisk /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==     #3  setConfig2 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==     #4  thread2 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==   Location 0xC0009B0 is 192 bytes inside a block starting at 0xC0008F0 of size 568 allocated by T2 from heap:
==4306==     #0  malloc /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:417
==4306==     #1  __fopen_internal /usr/lib64/libc-2.16.so
==4306==     #2  writeToDisk /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==     #3  setConfig2 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==     #4  thread2 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==    Race verifier data: 0x3F1D6798A0,0x3F1D679349
==4306== }}}
==4306== WARNING: Possible data race during read of size 8 at 0xC000958: {{{
==4306==    T0 (L{}):
==4306==     #0  _IO_cleanup /usr/lib64/libc-2.16.so
==4306==     #1  __run_exit_handlers /usr/lib64/libc-2.16.so
==4306==     #2  exit /usr/lib64/libc-2.16.so
==4306==     #3  exit /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:2005
==4306==   Concurrent write(s) happened at (OR AFTER) these points:
==4306==    T2 (L{}):
==4306==     #0  _IO_link_in /usr/lib64/libc-2.16.so
==4306==     #1  _IO_file_init@@GLIBC_2.2.5 /usr/lib64/libc-2.16.so
==4306==     #2  __fopen_internal /usr/lib64/libc-2.16.so
==4306==     #3  writeToDisk /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==     #4  setConfig2 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==     #5  thread2 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==   Location 0xC000958 is 104 bytes inside a block starting at 0xC0008F0 of size 568 allocated by T2 from heap:
==4306==     #0  malloc /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:417
==4306==     #1  __fopen_internal /usr/lib64/libc-2.16.so
==4306==     #2  writeToDisk /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==     #3  setConfig2 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==     #4  thread2 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out
==4306==    Race verifier data: 0x3F1D6798AA,0x3F1D67859D
==4306== }}}
==4306==
==4306== ThreadSanitizer summary: reported 10 warning(s) (10 race(s))
[manoj@localhost Bug6]$
```

Figure 6.1: bug detected by ThreadSanitizer tool



```
=4237== Locks held: none
=4237==    at 0x4008E0: setConfig1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out)
=4237==    by 0x400A6B: thread1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out)
=4237==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
=4237==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
=4237==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
=4237==
=4237== ----------------------------------------------------------------
=4237==
=4237== Possible data race during write of size 4 at 0x6020A0 by thread #3
=4237== Locks held: none
=4237==    at 0x400A1A: setConfig2 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out)
=4237==    by 0x400A88: thread2 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out)
=4237==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
=4237==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
=4237==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
=4237==
=4237== This conflicts with a previous write of size 4 by thread #2
=4237== Locks held: none
=4237==    at 0x4008EA: setConfig1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out)
=4237==    by 0x400A6B: thread1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out)
=4237==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
=4237==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
=4237==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
=4237==
=4237== ----------------------------------------------------------------
=4237==
=4237== Possible data race during write of size 1 at 0x602104 by thread #3
=4237== Locks held: none
=4237==    at 0x400A1A: setConfig2 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out)
=4237==    by 0x400A88: thread2 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out)
=4237==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
=4237==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
=4237==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
=4237==
=4237== This conflicts with a previous write of size 1 by thread #2
=4237== Locks held: none
=4237==    at 0x4009E4: setConfig1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out)
=4237==    by 0x400A6B: thread1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug6/a.out)
=4237==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
=4237==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
=4237==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
=4237==
=4237==
=4237== For counts of detected and suppressed errors, rerun with: -v
=4237== Use --history-level=approx or =none to gain increased speed, at
=4237== the cost of reduced accuracy of conflicting-access information
=4237== ERROR SUMMARY: 27 errors from 3 contexts (suppressed: 25 from 19)
anoj@localhost Bug6]$ valgrind --tool=helgrind ./a.out
```

Figure 6.2: bug detected by Valgrind –tool=helgrind

Bug No. 7:

Question: Description of the bug including details on how it manifests
Answer:

This bug occurs when cache size becomes negative and which produces segmentation fault. The main implementation logic is as follows:

A dynamically generated objects are first inserted in the cache as temporary object with a default size and once the caching completes and gets the actual size, cache data should re-insert with true size and updates the size variable which keeps the information about the cache buffer used size.

For example: default cache size = 100, actual size = 50 then before update, size variable value = 150 and after updates size value = 50.

When re-insertion happens with actual size then the size variable value subtracts by default value and updates with actual size.

Now the problem starts when there is a cache buffer limit, and buffer size exceeds the max limit of cache buffer, then some portion of cache buffer data will be removed and updates the size variable.

Suppose first thread inserts default size cache data and then actual size cache data and before updating the size variable, in the mean time, other thread from other function finds that cache buffer size is full and it removes data from cache buffer and updates size variable and in the same time first thread also update the size variable and size variable value could be become negative.

Eventually producing a negative cache size and a negative cache size quickly produces segmentation fault in cache_insert function.

Question: Patch (difference in code) that was used to fix the bug.
Answer:

```
--- mod_mem_cache.1.99.c        Thu Dec 11 15:36:06 2003
+++ mod_mem_cache.1.100.c       Thu Dec 11 16:50:40 2003
@@ -1028,6 +1028,7 @@
              * correct size and copy the streamed response into that
              * buffer */
             char *buf = malloc(obj->count);
+            cache_object_t *tmp_obj = NULL;
             if (!buf) {
                 return APR_ENOMEM;
             }
@@ -1043,7 +1044,18 @@
             if (sconf->lock) {
                 apr_thread_mutex_lock(sconf->lock);
             }
-            cache_remove(sconf->cache_cache, obj);
```

```
+                               /* We need to check if the object has been removed/replaced
+                                * from/in the cache, this could happen in some cases, because
+                                * the current request is a streaming response that is handled
+                                * in multiple individual pieces. - fixing Bugzilla Defect 21285
+                                */
+                           if((tmp_obj = (cache_object_t *) cache_find(sconf->cache_cache, obj->key)) &&
+                               (tmp_obj == obj)) {
+                       cache_remove(sconf->cache_cache, obj);
+                               }
+                               else {
+                                       obj->cleanup = 0;
+                               }
            mobj->m_len = obj->count;
            cache_insert(sconf->cache_cache, obj);
            sconf->cache_size -= (mobj->m_len - obj->count);
```

Question: How was the bug detected ?

Answer:

It occurs when web server tries to cache most frequent data into the cache memory and in the same time other thread seen that cache memory is full and it removes some portion of cache buffer data from cache memory and which produces data inconsistency since the first thread was coping data and other thread has removed data from cache memory. When cache size variable becomes negative then it generates segmentation fault and during the debugging process this bug was detected.

Question: How frequently does the bug appear? Or how important is the bug?

Answer:

Since caching frequency is high but this bug occurs when cache count variable becomes negative.

Question: Online link to the bug id

Answer:

https://issues.apache.org/bugzilla/show_bug.cgi?id=21285

Question: Generate a simple reproducer that follows the original bug

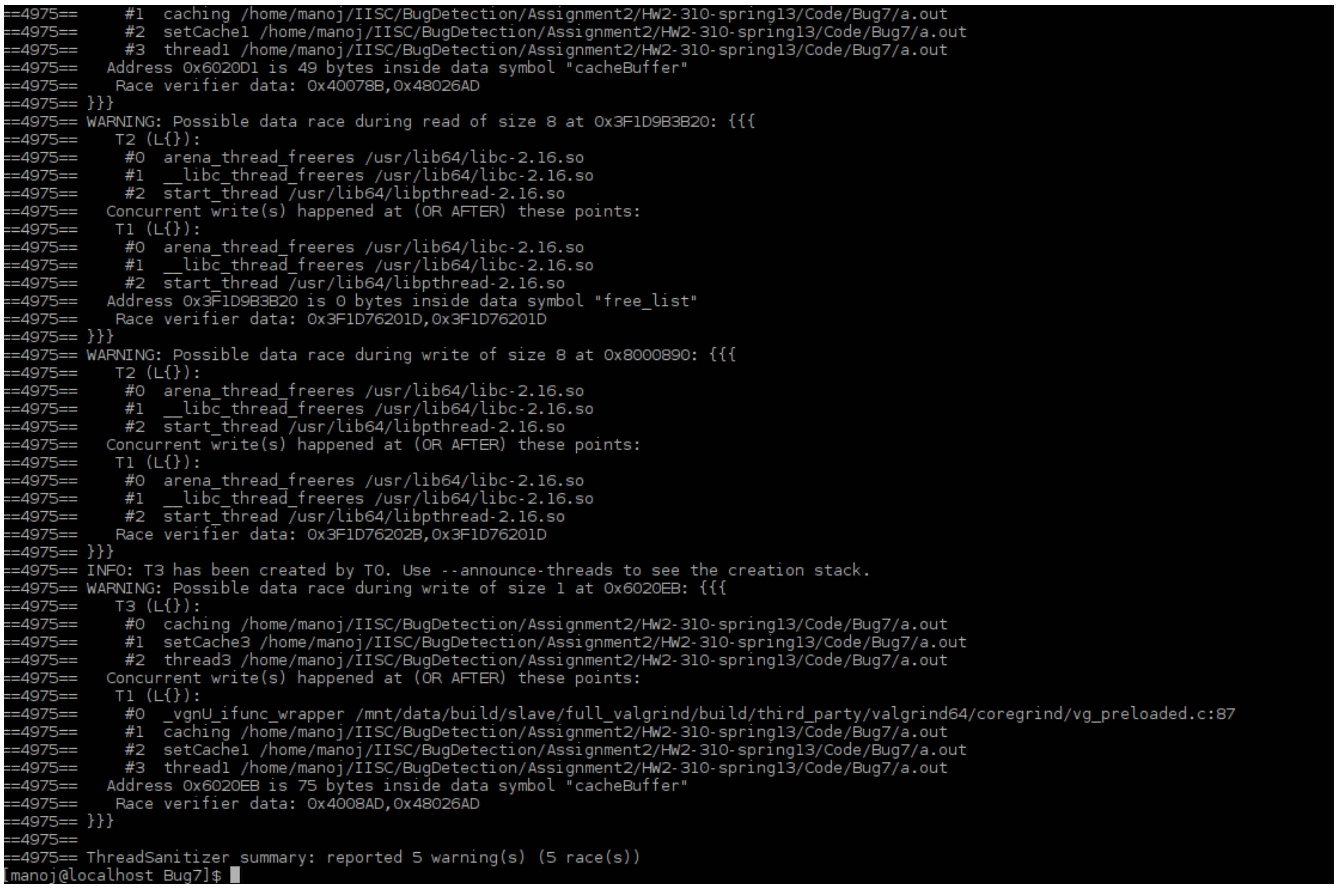
Answer:

Please refer the following directory:
HW2-310-spring/Code/Bug7/bug7_21285.c

Question: Investigate whether any available program analysis tool can

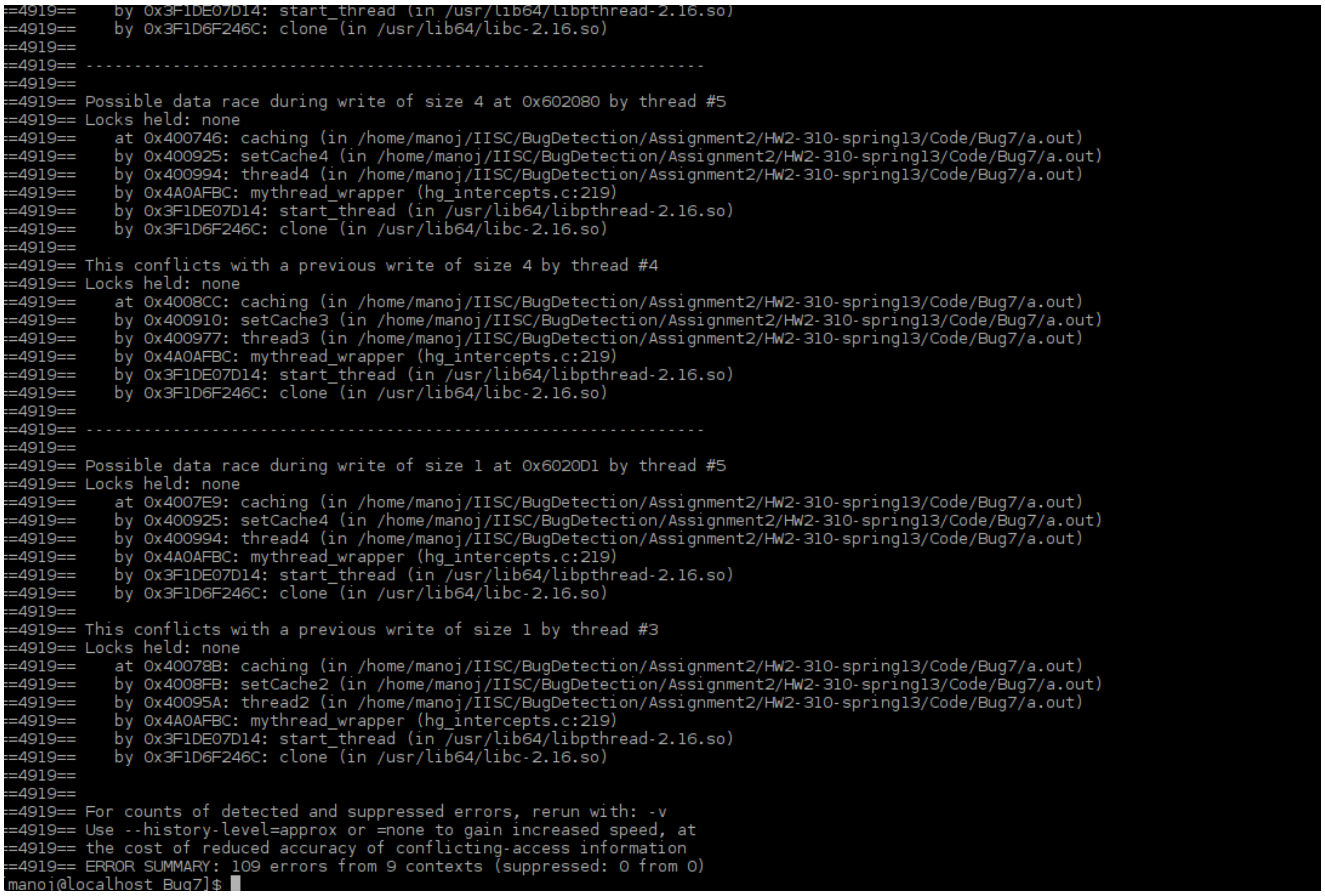
catch this bug automatically. If so, name the tool and a snapshot of its report.

Answer:



```
==4975==     #1  caching /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out
==4975==     #2  setCache1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out
==4975==     #3  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out
==4975==   Address 0x6020D1 is 49 bytes inside data symbol "cacheBuffer"
==4975==    Race verifier data: 0x40078B,0x48026AD
==4975== }}}
==4975== WARNING: Possible data race during read of size 8 at 0x3F1D9B3B20: {{{
==4975==    T2 (L{}):
==4975==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==4975==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==4975==     #2  start_thread /usr/lib64/libpthread-2.16.so
==4975==   Concurrent write(s) happened at (OR AFTER) these points:
==4975==    T1 (L{}):
==4975==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==4975==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==4975==     #2  start_thread /usr/lib64/libpthread-2.16.so
==4975==   Address 0x3F1D9B3B20 is 0 bytes inside data symbol "free_list"
==4975==    Race verifier data: 0x3F1D76201D,0x3F1D76201D
==4975== }}}
==4975== WARNING: Possible data race during write of size 8 at 0x8000890: {{{
==4975==    T2 (L{}):
==4975==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==4975==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==4975==     #2  start_thread /usr/lib64/libpthread-2.16.so
==4975==   Concurrent write(s) happened at (OR AFTER) these points:
==4975==    T1 (L{}):
==4975==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==4975==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==4975==     #2  start_thread /usr/lib64/libpthread-2.16.so
==4975==    Race verifier data: 0x3F1D76202B,0x3F1D76201D
==4975== }}}
==4975== INFO: T3 has been created by T0. Use --announce-threads to see the creation stack.
==4975== WARNING: Possible data race during write of size 1 at 0x6020EB: {{{
==4975==    T3 (L{}):
==4975==     #0  caching /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out
==4975==     #1  setCache3 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out
==4975==     #2  thread3 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out
==4975==   Concurrent write(s) happened at (OR AFTER) these points:
==4975==    T1 (L{}):
==4975==     #0  _vgnU_ifunc_wrapper /mnt/data/build/slave/full_valgrind/build/third_party/valgrind64/coregrind/vg_preloaded.c:87
==4975==     #1  caching /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out
==4975==     #2  setCache1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out
==4975==     #3  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out
==4975==   Address 0x6020EB is 75 bytes inside data symbol "cacheBuffer"
==4975==    Race verifier data: 0x4008AD,0x48026AD
==4975== }}}
==4975==
==4975== ThreadSanitizer summary: reported 5 warning(s) (5 race(s))
[manoj@localhost Bug7]$
```

Figure 7.1: bug detected by ThreadSanitizer tool



```
==4919==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==4919==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==4919==
==4919== ----------------------------------------------------------------
==4919==
==4919== Possible data race during write of size 4 at 0x602080 by thread #5
==4919== Locks held: none
==4919==    at 0x400746: caching (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x400925: setCache4 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x400994: thread4 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==4919==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==4919==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==4919==
==4919== This conflicts with a previous write of size 4 by thread #4
==4919== Locks held: none
==4919==    at 0x4008CC: caching (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x400910: setCache3 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x400977: thread3 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==4919==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==4919==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==4919==
==4919== ----------------------------------------------------------------
==4919==
==4919== Possible data race during write of size 1 at 0x6020D1 by thread #5
==4919== Locks held: none
==4919==    at 0x4007E9: caching (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x400925: setCache4 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x400994: thread4 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==4919==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==4919==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==4919==
==4919== This conflicts with a previous write of size 1 by thread #3
==4919== Locks held: none
==4919==    at 0x40078B: caching (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x4008FB: setCache2 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x40095A: thread2 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug7/a.out)
==4919==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==4919==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==4919==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==4919==
==4919==
==4919== For counts of detected and suppressed errors, rerun with: -v
==4919== Use --history-level=approx or =none to gain increased speed, at
==4919== the cost of reduced accuracy of conflicting-access information
==4919== ERROR SUMMARY: 109 errors from 9 contexts (suppressed: 0 from 0)
[manoj@localhost Bug7]$
```

Figure 7.2: bug detected by Valgrind –tool=helgrind

Bug No. 8:

Question: Description of the bug including details on how it manifests
Answer:

This bug occurs when one thread sets some value to a global variable and in the mean while another thread alters the variable value and when again first thread takes decision based on the global variable value then this bug occurs. This is again an example of simple data race and in the patch they had imposed locking and unlocking mechanism.

Question: Patch (difference in code) that was used to fix the bug.
Answer:

```
--- mod_mem_cache.1.99.c        Fri Dec 12 12:09:49 2003
+++ mod_mem_cache.c     Fri Dec 12 12:52:08 2003
@@ -352,33 +352,26 @@
             sconf->cache_size -= mobj->m_len;
             obj->cleanup = 1;
         }
-        if (sconf->lock) {
-            apr_thread_mutex_unlock(sconf->lock);
-        }
+    }
+        else if (sconf->lock) {
+        apr_thread_mutex_lock(sconf->lock);
     }

     /* Cleanup the cache object */
 #ifdef USE_ATOMICS
-    if (!apr_atomic_dec32(&obj->refcount)) {
-        if (obj->cleanup) {
-            cleanup_cache_object(obj);
-        }
-    }
+    if (!apr_atomic_dec32(&obj->refcount) && (obj->cleanup)) {
 #else
-    if (sconf->lock) {
-        apr_thread_mutex_lock(sconf->lock);
-    }
     obj->refcount--;
     /* If the object is marked for cleanup and the refcount
      * has dropped to zero, cleanup the object
      */
     if ((obj->cleanup) && (!obj->refcount)) {
+#endif
         cleanup_cache_object(obj);
     }
```

```
    if (sconf->lock) {
        apr_thread_mutex_unlock(sconf->lock);
    }
-#endif
    return APR_SUCCESS;
 }
 static apr_status_t cleanup_cache_mem(void *sconfv)
@@ -761,26 +754,18 @@
          * protection of the lock
          */
        apr_atomic_inc32(&obj->refcount);
+        obj->cleanup = 1;
+        if (!apr_atomic_dec32(&obj->refcount)) {
 #else
+        obj->cleanup = 1;
        if (!obj->refcount) {
-            cleanup_cache_object(obj);
-            obj = NULL;
-        }
 #endif
-        if (obj) {
-            obj->cleanup = 1;
-        }
+            cleanup_cache_object(obj);
+              }
    }
    if (sconf->lock) {
        apr_thread_mutex_unlock(sconf->lock);
    }
-#ifdef USE_ATOMICS
-    if (obj) {
-        if (!apr_atomic_dec32(&obj->refcount)) {
-            cleanup_cache_object(obj);
-        }
-    }
-#endif
    return OK;
 }
```

Question: How was the bug detected ?

Answer:

There are no mutex lock protection in decrement_refcount if it is defined USE_ATOMICS. This bug was detected in the cleanup_cache_object function, when two threads are trying to clean-up the same object which is no longer referenced in the cache.

Question: How frequently does the bug appear? Or how important is the bug?

Answer:

This bug is important as it crash the either server. This problem is very simple example of data races because i guess they forgotten to use mutex lock to protect decrement_refcount variable.

Question: Online link to the bug id

Answer:

http://issues.apache.org/bugzilla/show_bug.cgi?id=21287

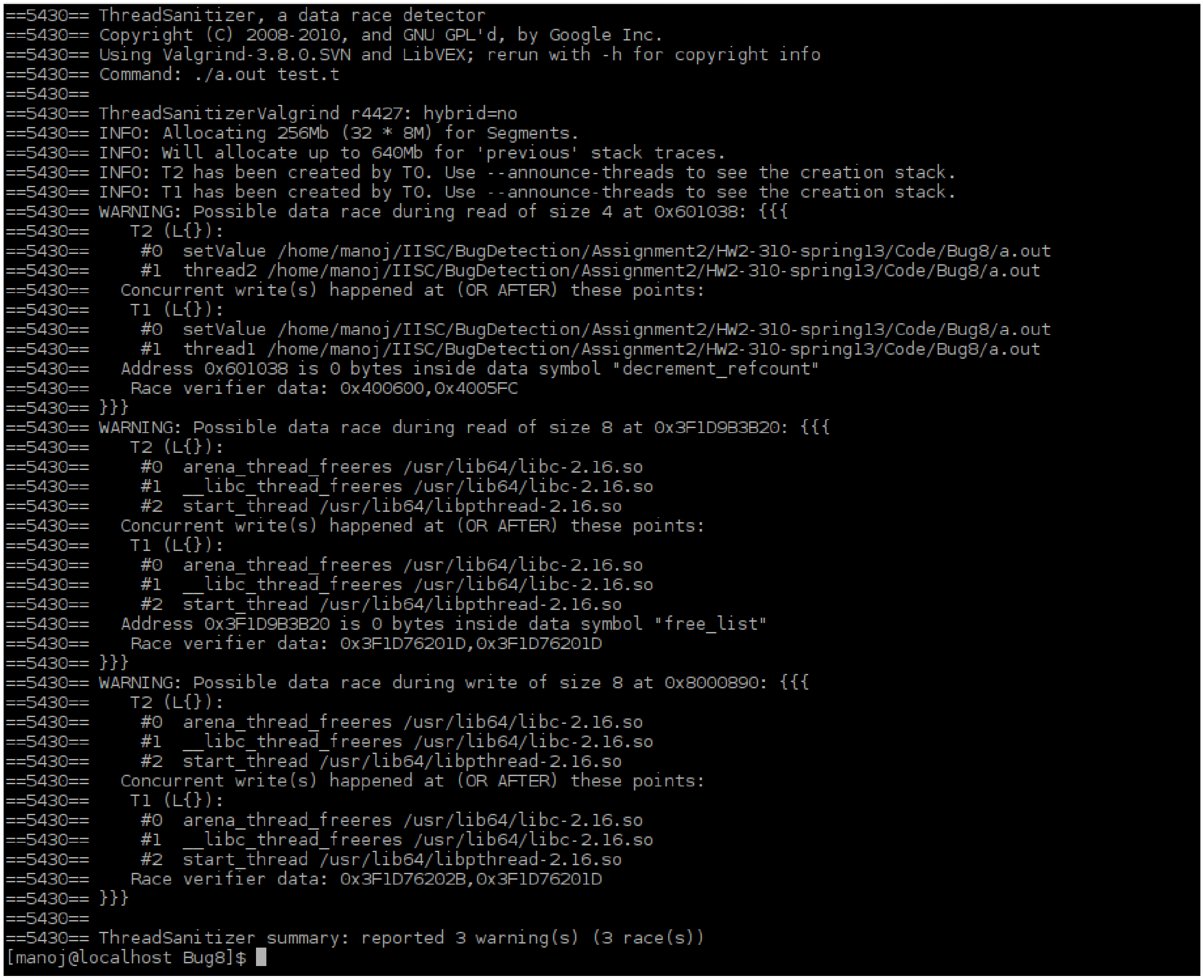
Question: Generate a simple reproducer that follows the original bug

Answer:

Please refer the following directory:
HW2-310-spring/Code/Bug8/bug8_21287.c

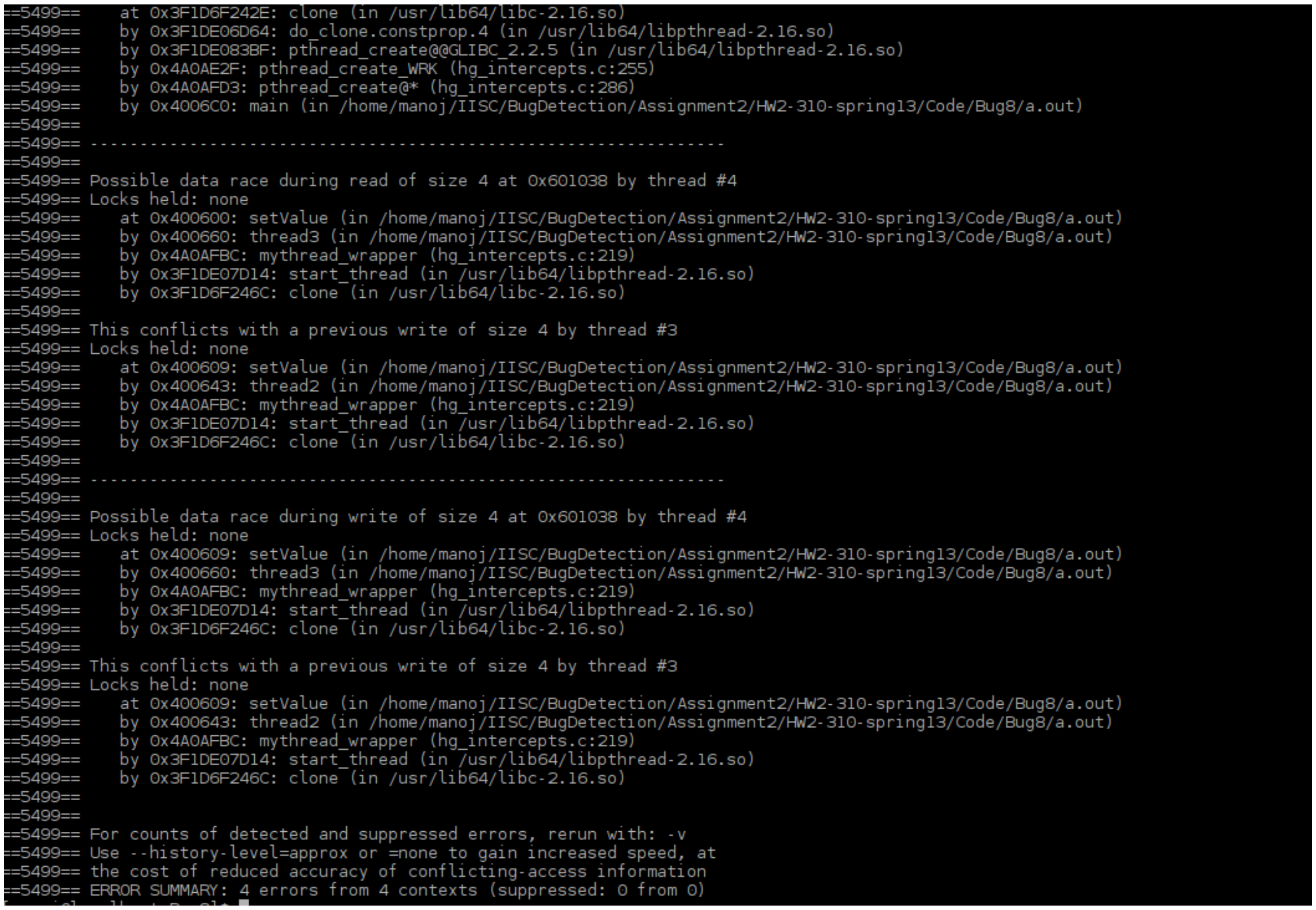
Question: Investigate whether any available program analysis tool can catch this bug automatically. If so, name the tool and a snapshot of its report.

Answer:



```
==5430== ThreadSanitizer, a data race detector
==5430== Copyright (C) 2008-2010, and GNU GPL'd, by Google Inc.
==5430== Using Valgrind-3.8.0.SVN and LibVEX; rerun with -h for copyright info
==5430== Command: ./a.out test.t
==5430==
==5430== ThreadSanitizerValgrind r4427: hybrid=no
==5430== INFO: Allocating 256Mb (32 * 8M) for Segments.
==5430== INFO: Will allocate up to 640Mb for 'previous' stack traces.
==5430== INFO: T2 has been created by T0. Use --announce-threads to see the creation stack.
==5430== INFO: T1 has been created by T0. Use --announce-threads to see the creation stack.
==5430== WARNING: Possible data race during read of size 4 at 0x601038: {{{
==5430==    T2 (L{}):
==5430==     #0  setValue /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out
==5430==     #1  thread2 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out
==5430==   Concurrent write(s) happened at (OR AFTER) these points:
==5430==    T1 (L{}):
==5430==     #0  setValue /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out
==5430==     #1  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out
==5430==   Address 0x601038 is 0 bytes inside data symbol "decrement_refcount"
==5430==    Race verifier data: 0x400600,0x4005FC
==5430== }}}
==5430== WARNING: Possible data race during read of size 8 at 0x3F1D9B3B20: {{{
==5430==    T2 (L{}):
==5430==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==5430==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==5430==     #2  start_thread /usr/lib64/libpthread-2.16.so
==5430==   Concurrent write(s) happened at (OR AFTER) these points:
==5430==    T1 (L{}):
==5430==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==5430==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==5430==     #2  start_thread /usr/lib64/libpthread-2.16.so
==5430==   Address 0x3F1D9B3B20 is 0 bytes inside data symbol "free_list"
==5430==    Race verifier data: 0x3F1D76201D,0x3F1D76201D
==5430== }}}
==5430== WARNING: Possible data race during write of size 8 at 0x8000890: {{{
==5430==    T2 (L{}):
==5430==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==5430==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==5430==     #2  start_thread /usr/lib64/libpthread-2.16.so
==5430==   Concurrent write(s) happened at (OR AFTER) these points:
==5430==    T1 (L{}):
==5430==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==5430==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==5430==     #2  start_thread /usr/lib64/libpthread-2.16.so
==5430==    Race verifier data: 0x3F1D76202B,0x3F1D76201D
==5430== }}}
==5430==
==5430== ThreadSanitizer summary: reported 3 warning(s) (3 race(s))
[manoj@localhost Bug8]$ 
```

Figure 8.1: bug detected by ThreadSanitizer tool



```
==5499==    at 0x3F1D6F242E: clone (in /usr/lib64/libc-2.16.so)
==5499==    by 0x3F1DE06D64: do_clone.constprop.4 (in /usr/lib64/libpthread-2.16.so)
==5499==    by 0x3F1DE083BF: pthread_create@@GLIBC_2.2.5 (in /usr/lib64/libpthread-2.16.so)
==5499==    by 0x4A0AE2F: pthread_create_WRK (hg_intercepts.c:255)
==5499==    by 0x4A0AFD3: pthread_create@* (hg_intercepts.c:286)
==5499==    by 0x4006C0: main (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out)
==5499==
==5499== ----------------------------------------------------------------
==5499==
==5499== Possible data race during read of size 4 at 0x601038 by thread #4
==5499== Locks held: none
==5499==    at 0x400600: setValue (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out)
==5499==    by 0x400660: thread3 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out)
==5499==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==5499==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==5499==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==5499==
==5499== This conflicts with a previous write of size 4 by thread #3
==5499== Locks held: none
==5499==    at 0x400609: setValue (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out)
==5499==    by 0x400643: thread2 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out)
==5499==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==5499==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==5499==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==5499==
==5499== ----------------------------------------------------------------
==5499==
==5499== Possible data race during write of size 4 at 0x601038 by thread #4
==5499== Locks held: none
==5499==    at 0x400609: setValue (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out)
==5499==    by 0x400660: thread3 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out)
==5499==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==5499==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==5499==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==5499==
==5499== This conflicts with a previous write of size 4 by thread #3
==5499== Locks held: none
==5499==    at 0x400609: setValue (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out)
==5499==    by 0x400643: thread2 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug8/a.out)
==5499==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==5499==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==5499==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==5499==
==5499==
==5499== For counts of detected and suppressed errors, rerun with: -v
==5499== Use --history-level=approx or =none to gain increased speed, at
==5499== the cost of reduced accuracy of conflicting-access information
==5499== ERROR SUMMARY: 4 errors from 4 contexts (suppressed: 0 from 0)
```

Figure 8.2: bug detected by Valgrind –tool=helgrind

Bug No. 9:

Question: Description of the bug including details on how it manifests

Answer:

This bug occurs when one thread reallocate or reset or frees memory location and another thread tries to get some information from that memory location so the end result, system generates illegal memory access and segmentation fault. In the patch they have put mutux lock and make these operation atomic.

This bug occurs in the following functions
In printf_thd() function which is in the sql/ha_innodb.cc file, has accessibility to change the value of thd->proc_info and in do_commond() which is in the sql/parse.cc file, when it checks whether thd->proc_info == NULL or not if not then function proceeds to next step and in the mean time print_thd() function reset thd->proc_info variable and application gets crash.

Question: Patch (difference in code) that was used to fix the bug.

Answer:

```
diff -Naur mysql-4.0.19-original/sql/ha_innodb.cc mysql-4.0.19-modified/sql/ha_innodb.cc
--- mysql-4.0.19-original/sql/ha_innodb.cc      2004-05-03 17:10:28.000000000 -0400
+++ mysql-4.0.19-modified/sql/ha_innodb.cc      2009-10-31 01:10:29.000000000 -0400
@@ -346,6 +346,9 @@

        if (thd->proc_info) {
                putc(' ', f);
+
+                sleep(1);
+
                fputs(thd->proc_info, f);
        }

diff -Naur mysql-4.0.19-original/sql/sql_parse.cc mysql-4.0.19-modified/sql/sql_parse.cc
--- mysql-4.0.19-original/sql/sql_parse.cc      2004-05-03 17:10:20.000000000 -0400
+++ mysql-4.0.19-modified/sql/sql_parse.cc      2009-10-31 01:12:12.000000000 -0400
@@ -1310,6 +1310,9 @@
   thd->proc_info="cleaning up";
   VOID(pthread_mutex_lock(&LOCK_thread_count)); // For process list
   thd->proc_info=0;
+
+      sleep(2);
+
   thd->command=COM_SLEEP;
   thd->query=0;
   thread_running--;
```

Question: How was the bug detected ?

Answer:

When MySQL starts, InnoDB creates a file innodb.status.<pid>. When it is shut down normally, InnoDB removes the file. The file should contain the same information as SHOW INNODB STATUS, and it should be updated every 15 seconds. So to investigate, developers safely remove the files, or maybe have a look at them first, in case there is some hint on what is causing the crashes.

Question: How frequently does the bug appear? Or how important is the bug?

Answer:

This bug occurs when multiple queries come from various clients then there may be high chances to crashing of mysql server.

Question: Online link to the bug id

Answer:

http://bugs.mysql.com/bug.php?id=3596

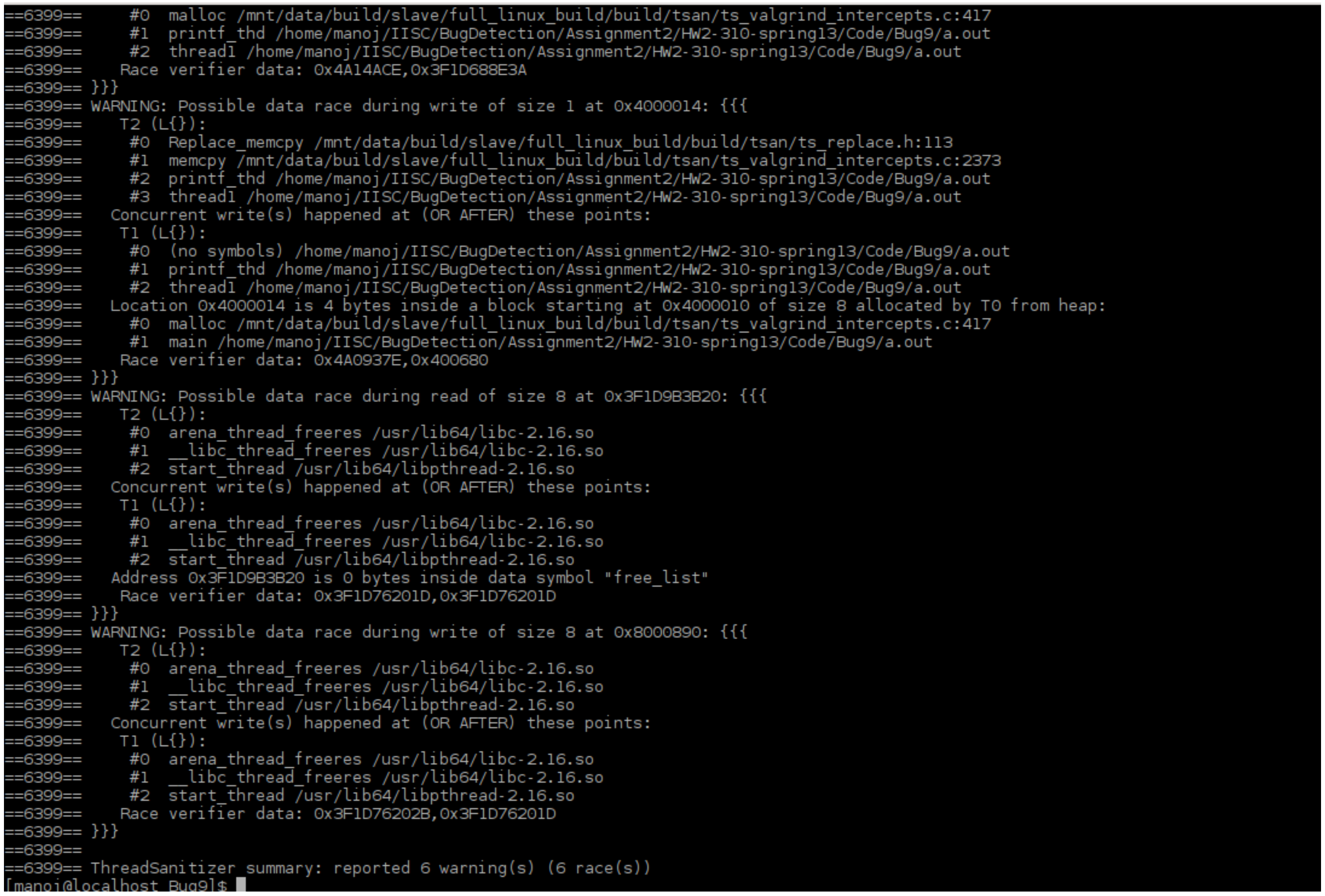
Question: Generate a simple reproducer that follows the original bug

Answer:

Please refer the following directory:
HW2-310-spring/Code/Bug9/bug9_3295.c

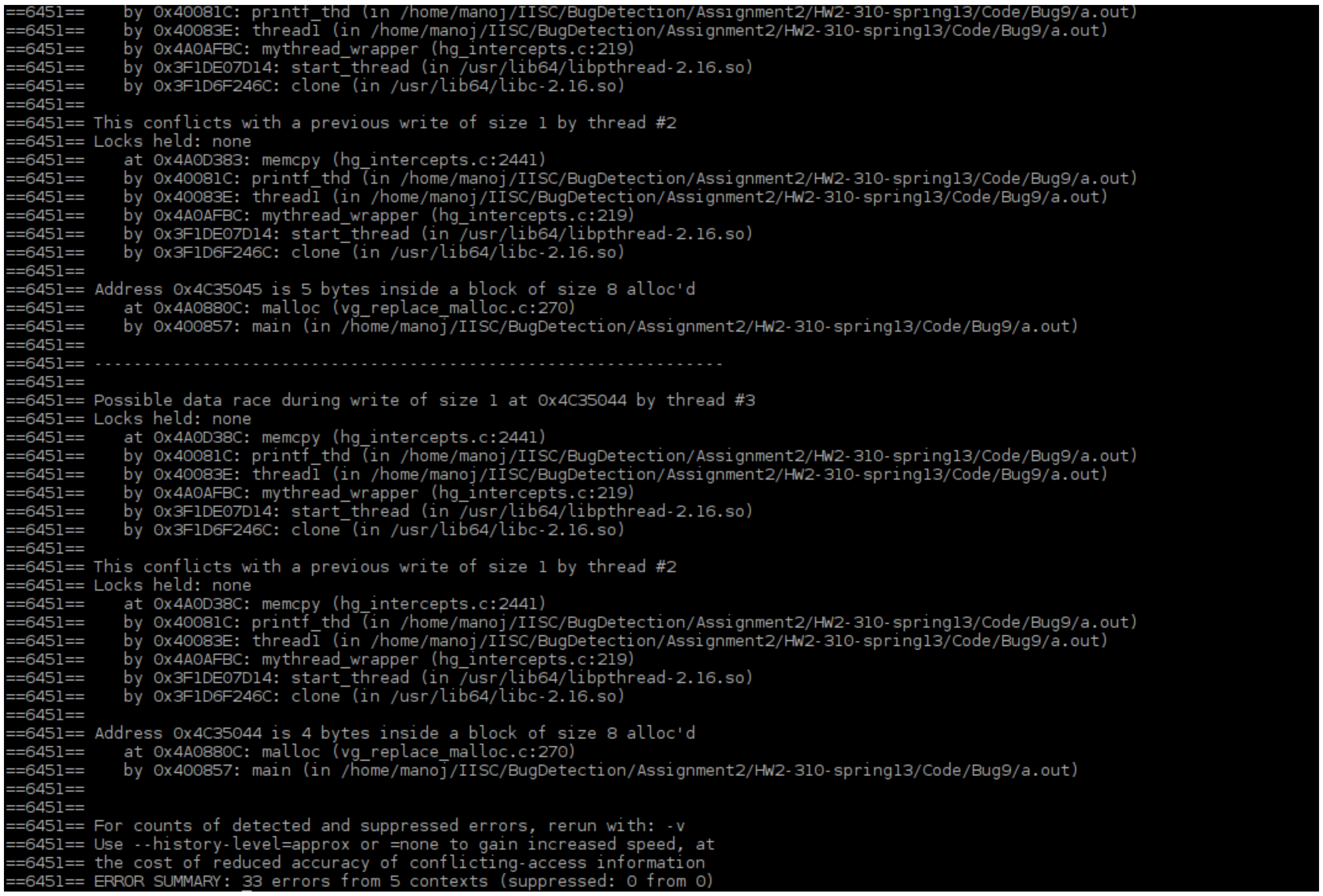
Question: Investigate whether any available program analysis tool can catch this bug automatically. If so, name the tool and a snapshot of its report.

Answer:



```
==6399==     #0  malloc /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:417
==6399==     #1  printf_thd /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out
==6399==     #2  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out
==6399==    Race verifier data: 0x4A14ACE,0x3F1D688E3A
==6399== }}}
==6399== WARNING: Possible data race during write of size 1 at 0x4000014: {{{
==6399==    T2 (L{}):
==6399==     #0  Replace_memcpy /mnt/data/build/slave/full_linux_build/build/tsan/ts_replace.h:113
==6399==     #1  memcpy /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:2373
==6399==     #2  printf_thd /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out
==6399==     #3  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out
==6399==   Concurrent write(s) happened at (OR AFTER) these points:
==6399==    T1 (L{}):
==6399==     #0  (no symbols) /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out
==6399==     #1  printf_thd /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out
==6399==     #2  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out
==6399==   Location 0x4000014 is 4 bytes inside a block starting at 0x4000010 of size 8 allocated by T0 from heap:
==6399==     #0  malloc /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:417
==6399==     #1  main /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out
==6399==    Race verifier data: 0x4A0937E,0x400680
==6399== }}}
==6399== WARNING: Possible data race during read of size 8 at 0x3F1D9B3B20: {{{
==6399==    T2 (L{}):
==6399==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==6399==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==6399==     #2  start_thread /usr/lib64/libpthread-2.16.so
==6399==   Concurrent write(s) happened at (OR AFTER) these points:
==6399==    T1 (L{}):
==6399==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==6399==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==6399==     #2  start_thread /usr/lib64/libpthread-2.16.so
==6399==   Address 0x3F1D9B3B20 is 0 bytes inside data symbol "free_list"
==6399==    Race verifier data: 0x3F1D76201D,0x3F1D76201D
==6399== }}}
==6399== WARNING: Possible data race during write of size 8 at 0x8000890: {{{
==6399==    T2 (L{}):
==6399==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==6399==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==6399==     #2  start_thread /usr/lib64/libpthread-2.16.so
==6399==   Concurrent write(s) happened at (OR AFTER) these points:
==6399==    T1 (L{}):
==6399==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==6399==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==6399==     #2  start_thread /usr/lib64/libpthread-2.16.so
==6399==    Race verifier data: 0x3F1D76202B,0x3F1D76201D
==6399== }}}
==6399==
==6399== ThreadSanitizer summary: reported 6 warning(s) (6 race(s))
[manoj@localhost Bug9]$ 
```

Figure 9.1: bug detected by ThreadSanitizer tool



```
==6451==    by 0x40081C: printf_thd (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out)
==6451==    by 0x40083E: thread1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out)
==6451==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==6451==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==6451==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==6451==
==6451== This conflicts with a previous write of size 1 by thread #2
==6451== Locks held: none
==6451==    at 0x4A0D383: memcpy (hg_intercepts.c:2441)
==6451==    by 0x40081C: printf_thd (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out)
==6451==    by 0x40083E: thread1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out)
==6451==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==6451==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==6451==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==6451==
==6451== Address 0x4C35045 is 5 bytes inside a block of size 8 alloc'd
==6451==    at 0x4A0880C: malloc (vg_replace_malloc.c:270)
==6451==    by 0x400857: main (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out)
==6451==
==6451== ----------------------------------------------------------------
==6451==
==6451== Possible data race during write of size 1 at 0x4C35044 by thread #3
==6451== Locks held: none
==6451==    at 0x4A0D38C: memcpy (hg_intercepts.c:2441)
==6451==    by 0x40081C: printf_thd (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out)
==6451==    by 0x40083E: thread1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out)
==6451==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==6451==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==6451==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==6451==
==6451== This conflicts with a previous write of size 1 by thread #2
==6451== Locks held: none
==6451==    at 0x4A0D38C: memcpy (hg_intercepts.c:2441)
==6451==    by 0x40081C: printf_thd (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out)
==6451==    by 0x40083E: thread1 (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out)
==6451==    by 0x4A0AFBC: mythread_wrapper (hg_intercepts.c:219)
==6451==    by 0x3F1DE07D14: start_thread (in /usr/lib64/libpthread-2.16.so)
==6451==    by 0x3F1D6F246C: clone (in /usr/lib64/libc-2.16.so)
==6451==
==6451== Address 0x4C35044 is 4 bytes inside a block of size 8 alloc'd
==6451==    at 0x4A0880C: malloc (vg_replace_malloc.c:270)
==6451==    by 0x400857: main (in /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug9/a.out)
==6451==
==6451==
==6451== For counts of detected and suppressed errors, rerun with: -v
==6451== Use --history-level=approx or =none to gain increased speed, at
==6451== the cost of reduced accuracy of conflicting-access information
==6451== ERROR SUMMARY: 33 errors from 5 contexts (suppressed: 0 from 0)
```

Figure 9.2: bug detected by Valgrind –tool=helgrind

Bug No. 10:

Question: Description of the bug including details on how it manifests
Answer:

This bug occurs when a thread is performing cache resizing while the other thread is storing sql queries into the same cache. The cache resizing (in function 'resize') is not atomic, making it possible for other threads to read an intermediate state, leading to a crash.

This bug occurs in the following functions
query_cache_size variable was used in various functions, but it was not at all protected by any lock. For instance, suppose store_query() function is performing some task and in the mean time some other thread calls resize() function and alter query_cache_size variable value and hence this leads to data inconsistency.

Question: Patch (difference in code) that was used to fix the bug.
Answer:

```
--- 1.65/sql/sql_cache.cc       2005-08-10 18:58:47 +03:00
+++ 1.66/sql/sql_cache.cc     2005-08-30 02:25:29 +03:00
@@ -563,13 +563,18 @@
 {
   DBUG_ENTER("query_cache_insert");

-#ifndef DBUG_OFF
-  // Check if we have called query_cache.wreck() (which disables the cache)
-  if (query_cache.query_cache_size == 0)
+  STRUCT_LOCK(&query_cache.structure_guard_mutex);
+  /*
+    It is very unlikely that following condition is TRUE (it is possible
+    only if other thread is resizing cache), so we check it only after guard
+    mutex lock
+  */
+  if (unlikely(query_cache.query_cache_size == 0))
+  {
+    STRUCT_UNLOCK(&query_cache.structure_guard_mutex);
     DBUG_VOID_RETURN;
-#endif
+  }

-  STRUCT_LOCK(&query_cache.structure_guard_mutex);
   Query_cache_block *query_block = ((Query_cache_block*)
                                    net->query_cache_query);
   if (query_block)
@@ -612,14 +617,20 @@
 {
   DBUG_ENTER("query_cache_abort");
```

```
-#ifndef DBUG_OFF
-  // Check if we have called query_cache.wreck() (which disables the cache)
-  if (query_cache.query_cache_size == 0)
-    DBUG_VOID_RETURN;
-#endif
   if (net->query_cache_query != 0)   // Quick check on unlocked structure
   {
     STRUCT_LOCK(&query_cache.structure_guard_mutex);
+    /*
+      It is very unlikely that following condition is TRUE (it is possible
+      only if other thread is resizing cache), so we check it only after guard
+      mutex lock
+    */
+    if (unlikely(query_cache.query_cache_size == 0))
+    {
+      STRUCT_UNLOCK(&query_cache.structure_guard_mutex);
+      DBUG_VOID_RETURN;
+    }
+
     Query_cache_block *query_block = ((Query_cache_block*)
                                        net->query_cache_query);
     if (query_block)                           // Test if changed by other thread
@@ -641,14 +652,20 @@
 {
   DBUG_ENTER("query_cache_end_of_result");

-#ifndef DBUG_OFF
-  // Check if we have called query_cache.wreck() (which disables the cache)
-  if (query_cache.query_cache_size == 0) DBUG_VOID_RETURN;
-#endif
-
   if (net->query_cache_query != 0)   // Quick check on unlocked structure
   {
     STRUCT_LOCK(&query_cache.structure_guard_mutex);
+    /*
+      It is very unlikely that following condition is TRUE (it is possible
+      only if other thread is resizing cache), so we check it only after guard
+      mutex lock
+    */
+    if (unlikely(query_cache.query_cache_size == 0))
+    {
+      STRUCT_UNLOCK(&query_cache.structure_guard_mutex);
+      DBUG_VOID_RETURN;
+    }
+
     Query_cache_block *query_block = ((Query_cache_block*)
                                       net->query_cache_query);
     if (query_block)
@@ -728,9 +745,15 @@
   DBUG_ENTER("Query_cache::resize");
   DBUG_PRINT("qcache", ("from %lu to %lu",query_cache_size,
                        query_cache_size_arg));
-  free_cache(0);
+  if (!initialized)
+    init();
+  STRUCT_LOCK(&structure_guard_mutex);
+  if (query_cache_size > 0)
+    free_cache();
   query_cache_size= query_cache_size_arg;
-  DBUG_RETURN(::query_cache_size= init_cache());
+  ::query_cache_size= init_cache();
+  STRUCT_UNLOCK(&structure_guard_mutex);
+  DBUG_RETURN(::query_cache_size);
 }


@@ -1307,7 +1330,7 @@
   }
   else
   {
-    free_cache(1);
+    free_cache();
     pthread_mutex_destroy(&structure_guard_mutex);
     initialized = 0;
   }
@@ -1336,8 +1359,6 @@
   int align;

   DBUG_ENTER("Query_cache::init_cache");
-  if (!initialized)
-    init();
   approx_additional_data_size = (sizeof(Query_cache) +
                                  sizeof(gptr)*(def_query_hash_size+
                                                def_table_hash_size));
@@ -1395,14 +1416,9 @@
     goto err;
   query_cache_size -= additional_data_size;

-  STRUCT_LOCK(&structure_guard_mutex);
-
-  if (!(cache = (byte *)
-          my_malloc_lock(query_cache_size+additional_data_size, MYF(0))))
-  {
-    STRUCT_UNLOCK(&structure_guard_mutex);
+  if (!(cache= (byte *)
+        my_malloc_lock(query_cache_size+additional_data_size, MYF(0))))
     goto err;
```

```
-  }

   DBUG_PRINT("qcache", ("cache length %lu, min unit %lu, %u bins",
                         query_cache_size, min_allocation_unit, mem_bin_num));
@@ -1482,7 +1498,6 @@

   queries_in_cache = 0;
   queries_blocks = 0;
-  STRUCT_UNLOCK(&structure_guard_mutex);
   DBUG_RETURN(query_cache_size +
               additional_data_size + approx_additional_data_size);

@@ -1509,14 +1524,11 @@
 }


-void Query_cache::free_cache(my_bool destruction)
+void Query_cache::free_cache()
 {
   DBUG_ENTER("Query_cache::free_cache");
   if (query_cache_size > 0)
   {
-    if (!destruction)
-      STRUCT_LOCK(&structure_guard_mutex);
-
     flush_cache();
 #ifndef DBUG_OFF
     if (bins[0].free_blocks == 0)
@@ -1538,8 +1550,6 @@
     make_disabled();
     hash_free(&queries);
     hash_free(&tables);
-    if (!destruction)
-      STRUCT_UNLOCK(&structure_guard_mutex);
   }
   DBUG_VOID_RETURN;
 }
@@ -2149,7 +2159,19 @@
   }

   if (!under_guard)
+  {
     STRUCT_LOCK(&structure_guard_mutex);
+    /*
+      It is very unlikely that following condition is TRUE (it is possible
+      only if other thread is resizing cache), so we check it only after
+      guard mutex lock
+    */
+    if (unlikely(query_cache.query_cache_size == 0))
+    {
+      STRUCT_UNLOCK(&structure_guard_mutex);
+      DBUG_RETURN(0);
+    }
+  }

   /* Free old queries until we have enough memory to store this block */
   Query_cache_block *block;
@@ -2606,6 +2628,17 @@
 {
   DBUG_ENTER("Query_cache::pack_cache");
   STRUCT_LOCK(&structure_guard_mutex);
+  /*
+    It is very unlikely that following condition is TRUE (it is possible
+    only if other thread is resizing cache), so we check it only after
+    guard mutex lock
+  */
+  if (unlikely(query_cache_size == 0))
+  {
+    STRUCT_UNLOCK(&structure_guard_mutex);
+    DBUG_VOID_RETURN;
+  }
+
   DBUG_EXECUTE("check_querycache",query_cache.check_integrity(1););

   byte *border = 0;
@@ -2913,7 +2946,7 @@
   DBUG_ENTER("Query_cache::join_results");

   STRUCT_LOCK(&structure_guard_mutex);
-  if (queries_blocks != 0)
+  if (query_cache_size > 0 && queries_blocks != 0)
   {
     Query_cache_block *block = queries_blocks;
     do
@@ -3209,7 +3242,19 @@
     DBUG_RETURN(0);
   }
   if (!not_locked)
+  {
     STRUCT_LOCK(&structure_guard_mutex);
+    /*
+      It is very unlikely that following condition is TRUE (it is possible
+      only if other thread is resizing cache), so we check it only after
+      guard mutex lock
+    */
+    if (unlikely(query_cache_size == 0))
+    {
+      STRUCT_UNLOCK(&query_cache.structure_guard_mutex);
+     DBUG_RETURN(0);
+   }
+  }

   if (hash_check(&queries))
   {

--- 1.22/sql/sql_cache.h        2004-10-21 23:56:02 +03:00
+++ 1.23/sql/sql_cache.h        2005-08-30 02:25:30 +03:00
@@ -313,7 +313,7 @@
   void init();
   ulong init_cache();
   void make_disabled();
-  void free_cache(my_bool destruction);
+  void free_cache();
   Query_cache_block *write_block_data(ulong data_len, gptr data,
                                       ulong header_len,
                                       Query_cache_block::block_type type)
```

Question: How was the bug detected ?

Answer:

This bug occurs when a thread is performing cache resizing while the other thread is storing sql queries into the same cache. The cache resizing (in function 'resize') is not atomic, making it possible for other threads to read an intermediate state, leading to a crash.

Question: How frequently does the bug appear? Or how important is the bug?

Answer:

It is given in the bug site that: Running query_cache test case from regular mysql-test suite in multithread environment (stress test) for some time (less than 1-2 min) will lead to crash.

Question: Online link to the bug id

Answer:

http://bugs.mysql.com/bug.php?id=12848

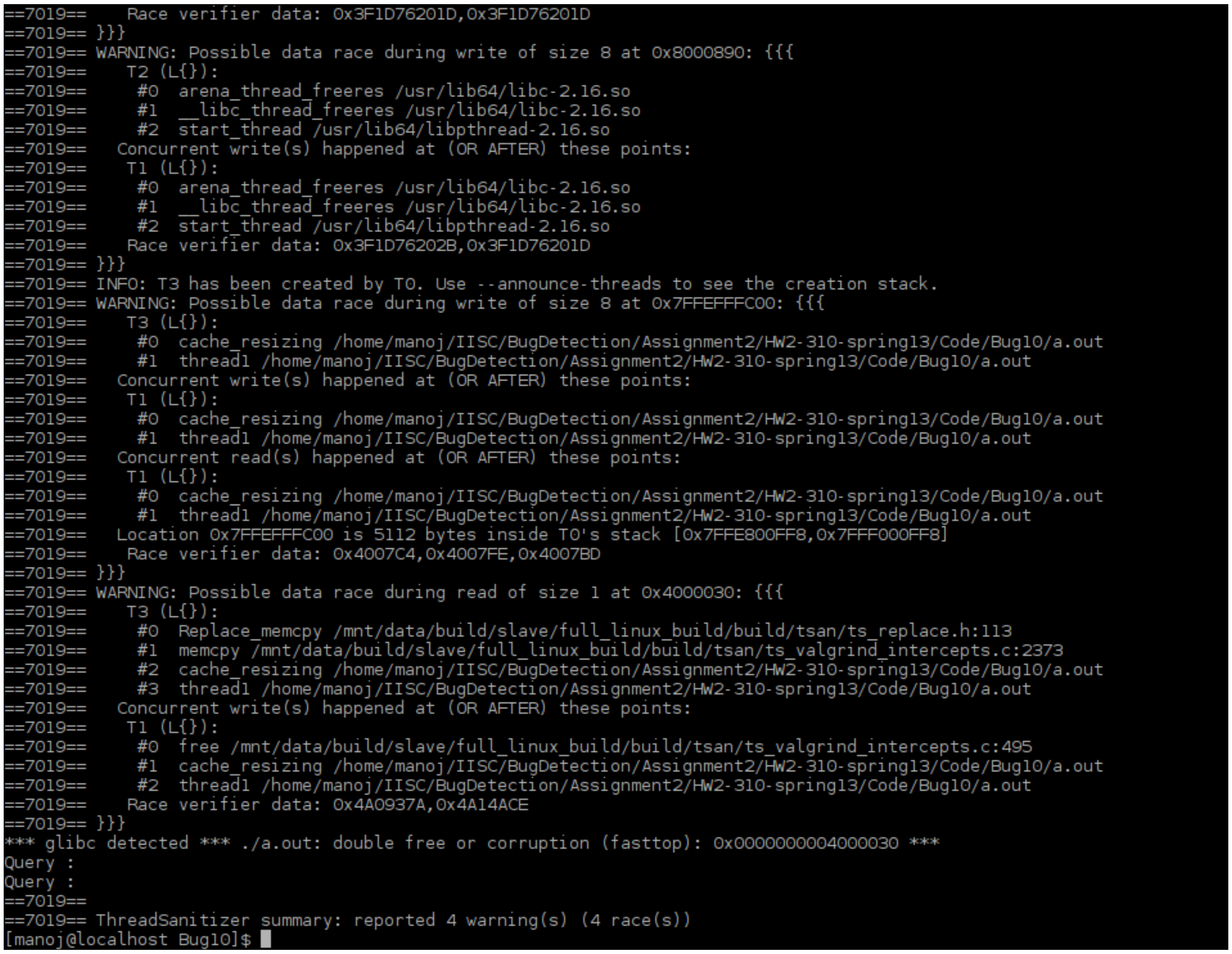
Question: Generate a simple reproducer that follows the original bug

Answer:

Please refer the following directory:
HW2-310-spring/Code/Bug10/bug10_12848.c

Question: Investigate whether any available program analysis tool can catch this bug automatically. If so, name the tool and a snapshot of its report.

Answer:



```
==7019==    Race verifier data: 0x3F1D76201D,0x3F1D76201D
==7019== }}}
==7019== WARNING: Possible data race during write of size 8 at 0x8000890: {{{
==7019==    T2 (L{}):
==7019==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==7019==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==7019==     #2  start_thread /usr/lib64/libpthread-2.16.so
==7019==   Concurrent write(s) happened at (OR AFTER) these points:
==7019==    T1 (L{}):
==7019==     #0  arena_thread_freeres /usr/lib64/libc-2.16.so
==7019==     #1  __libc_thread_freeres /usr/lib64/libc-2.16.so
==7019==     #2  start_thread /usr/lib64/libpthread-2.16.so
==7019==    Race verifier data: 0x3F1D76202B,0x3F1D76201D
==7019== }}}
==7019== INFO: T3 has been created by T0. Use --announce-threads to see the creation stack.
==7019== WARNING: Possible data race during write of size 8 at 0x7FFEFFFC00: {{{
==7019==    T3 (L{}):
==7019==     #0  cache_resizing /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug10/a.out
==7019==     #1  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug10/a.out
==7019==   Concurrent write(s) happened at (OR AFTER) these points:
==7019==    T1 (L{}):
==7019==     #0  cache_resizing /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug10/a.out
==7019==     #1  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug10/a.out
==7019==   Concurrent read(s) happened at (OR AFTER) these points:
==7019==    T1 (L{}):
==7019==     #0  cache_resizing /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug10/a.out
==7019==     #1  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug10/a.out
==7019==   Location 0x7FFEFFFC00 is 5112 bytes inside T0's stack [0x7FFE800FF8,0x7FFF000FF8]
==7019==    Race verifier data: 0x4007C4,0x4007FE,0x4007BD
==7019== }}}
==7019== WARNING: Possible data race during read of size 1 at 0x4000030: {{{
==7019==    T3 (L{}):
==7019==     #0  Replace_memcpy /mnt/data/build/slave/full_linux_build/build/tsan/ts_replace.h:113
==7019==     #1  memcpy /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:2373
==7019==     #2  cache_resizing /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug10/a.out
==7019==     #3  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug10/a.out
==7019==   Concurrent write(s) happened at (OR AFTER) these points:
==7019==    T1 (L{}):
==7019==     #0  free /mnt/data/build/slave/full_linux_build/build/tsan/ts_valgrind_intercepts.c:495
==7019==     #1  cache_resizing /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug10/a.out
==7019==     #2  thread1 /home/manoj/IISC/BugDetection/Assignment2/HW2-310-spring13/Code/Bug10/a.out
==7019==    Race verifier data: 0x4A0937A,0x4A14ACE
==7019== }}}
*** glibc detected *** ./a.out: double free or corruption (fasttop): 0x0000000004000030 ***
Query :
Query :
==7019==
==7019== ThreadSanitizer summary: reported 4 warning(s) (4 race(s))
[manoj@localhost Bug10]$ █
```

Figure 10.1: bug detected by ThreadSanitizer tool