Dynamic Analysis Tool "Valgrind"

Andrew Peng edited this page Apr 11, 2016 · 3 revisions

Concept

In order to enhance our code quality some dynamic analysis tools should be consider to used.

Valgrind introduction

Valgrind is an instrumentation framework for building dynamic analysis tools. It comes with a set of tools each of which performs some kind of debugging, profiling, or similar task that helps you improve your programs. Valgrind is designed to be as non-intrusive as possible. It works directly with existing executables. You don’t need to recompile, relink, or otherwise modify the program to be checked. Your program will run on a synthetic CPU provided by the Valgrind core.

Please refer to the http://valgrind.org/docs/manual/manual-intro.html to get more information.

CPU support list

Pages 2

Home

Dynamic Analysis Tool "Valgrind"

Clone this wiki locally

https://github.com/Andrew1013/

Reference link: http://www.valgrind.org/info/platforms.html

- x86
- ARM
- MIPS
- PPC

OS support list

- linux
- solaris
- darwin
- Andriod

A number of useful tools

You can select different tool through the argument of "--tool" to check your program.

- **Memcheck** is a memory error detector. It helps you make your programs, particularly those written in C and C++, more correct.
- **Cachegrind** is a cache and branch-prediction profiler. It helps you make your programs run faster.
- **Callgrind** is a call-graph generating cache profiler. It has some overlap with Cachegrind, but also gathers some information that Cachegrind does not.
- **Helgrind** is a thread error detector. It helps you make your multi-threaded programs more correct.
- **DRD** is also a thread error detector. It is similar to Helgrind but uses different analysis techniques and so may find different problems.
- **Massif** is a heap profiler. It helps you make your programs use less memory.

- **DHAT** is a different kind of heap profiler. It helps you understand issues of block lifetimes, block utilisation, and layout inefficiencies.
- **SGcheck** is an experimental tool that can detect overruns of stack and global arrays. Its functionality is complementary to that of Memcheck: SGcheck finds problems that Memcheck can't, and vice versa..
- **BBV** is an experimental SimPoint basic block vector generator. It is useful to people doing computer architecture research and development.

How to build for x86 platform

- Decompress valgrind-3.11.0.tgz file
- Execute following commands to build

```
 $> ./configure
$> make
$> make install
```

- Execute command to analyze the application that you wanted

```
 $> ./vg-in-place --trace-children=yes --track-fds=yes ${APPLICATION} ${APPLICATION_ARG} ...
```

Note:

- vg-in-place script is used for run Valgrind without having to install it.
- Make sure the program is compiled with CFlag "-g -O0".

How to build for ARM platform

- Check your cross compile environment is ready
- Execute following commands to build

```
 $> ./configure --prefix=$PWD/_install --host=armv7-linux
$> make
$> make install
```

- Package related files

```
 $> cd  _install
$> tar cvzf ../valgrind-rfs.tgz .
```

- Put tar file to ARM platform and decompress the file under "/" directory
- Execute command to analyze the application that you wanted

```
 $> export VALGRIND_LIB=/lib/valgrind
$> valgrind --trace-children=yes --track-fds=yes ${APPLICATION} ${APPLICATION_ARG} ...
```

Useful command arguments

Argument	Description
--tool	Use the Valgrind tool named. The default tool is "memcheck"
--trace-children	Track child processes

--track-fds	Track open file descriptors
-d	Enable Valgrind debug message
--trace-mutex	Trace all mutex activity. This argument is depends on --tool=drd
--trace-rwlock	Trace all reader-writer activity. This argument is depends on --tool=drd
--trace-semaphore	Trace all semaphore activity. This argument is depends on --tool=drd (excluding SYS V)

Test cases

We used the "Memcheck" and "DRD" to defect the error of memory leak, file descriptor leak, heap overrun, stack overrun, stack overflow and thread race condition.

case1

Source code

```
 ......
void memLeak()
{
    void *p = malloc(1024);
    return;
}
```

Test result

```
==1987== HEAP SUMMARY:
==1987==     in use at exit: 1,592 bytes in 2 blocks
==1987==   total heap usage: 7 allocs, 6 frees, 11,833 bytes allocated
==1987==
==1987== LEAK SUMMARY:
==1987==    definitely lost: 1,024 bytes in 1 blocks
==1987==    indirectly lost: 0 bytes in 0 blocks
==1987==      possibly lost: 0 bytes in 0 blocks
==1987==    still reachable: 568 bytes in 1 blocks
==1987==         suppressed: 0 bytes in 0 blocks
```

case2

Source code

```
......
48: void memFree()
49: {
50:     delete malloc(1024);
51:     free(malloc(1024));
52:     delete (new int[1024]);
53:     free(new int[1024]);
54:     AAA *aaa = new AAA();
55:     delete aaa;
56:     delete aaa;
57:     return 0;
58: }
```

Test result

```
==1987== Memcheck, a memory error detector
```

```
==1987== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==1987== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==1987== Command: /home/clyde/build/sources/test
==1987==
==1987== Mismatched free() / delete / delete []
==1987==    at 0x4A0684A: operator delete(void*) (vg_replace_malloc.c:575)
==1987==    by 0x4009C5: memFree() (test1.cc:50)
==1987==    by 0x400A76: main (test1.cc:72)
==1987==  Address 0x4c27480 is 0 bytes inside a block of size 1,024 alloc'd
==1987==    at 0x4A0728A: malloc (vg_replace_malloc.c:299)
==1987==    by 0x4009BD: memFree() (test1.cc:50)
==1987==    by 0x400A76: main (test1.cc:72)
==1987==
==1987== Mismatched free() / delete / delete []
==1987==    at 0x4A0684A: operator delete(void*) (vg_replace_malloc.c:575)
==1987==    by 0x4009E9: memFree() (test1.cc:52)
==1987==    by 0x400A76: main (test1.cc:72)
==1987==  Address 0x4c27d00 is 0 bytes inside a block of size 4,096 alloc'd
==1987==    at 0x4A07A02: operator new[](unsigned long) (vg_replace_malloc.c:422)
==1987==    by 0x4009E1: memFree() (test1.cc:52)
==1987==    by 0x400A76: main (test1.cc:72)
==1987==
==1987== Mismatched free() / delete / delete []
==1987==    at 0x4A06C64: free (vg_replace_malloc.c:530)
==1987==    by 0x4009FB: memFree() (test1.cc:53)
==1987==    by 0x400A76: main (test1.cc:72)
==1987==  Address 0x4c28d40 is 0 bytes inside a block of size 4,096 alloc'd
==1987==    at 0x4A07A02: operator new[](unsigned long) (vg_replace_malloc.c:422)
==1987==    by 0x4009F3: memFree() (test1.cc:53)
==1987==    by 0x400A76: main (test1.cc:72)
==1987==
==1987== Invalid free() / delete / delete[] / realloc()
==1987==    at 0x4A0684A: operator delete(void*) (vg_replace_malloc.c:575)
==1987==    by 0x400A49: memFree() (test1.cc:57)
==1987==    by 0x400A76: main (test1.cc:72)
==1987==  Address 0x4c29d80 is 0 bytes inside a block of size 1 free'd
==1987==    at 0x4A0684A: operator delete(void*) (vg_replace_malloc.c:575)
```

```
==1987==    by 0x400A30: memFree() (test1.cc:56)
==1987==    by 0x400A76: main (test1.cc:72)
==1987==  Block was alloc'd at
==1987==    at 0x4A07E85: operator new(unsigned long) (vg_replace_malloc.c:333)
==1987==    by 0x400A05: memFree() (test1.cc:55)
==1987==    by 0x400A76: main (test1.cc:72)
```

case3

Source code

```
 ......
22: void fdLeak()
23: {
24:     int fd = open("/dev/null", O_RDWR);
25:     FILE *fd1 = fopen("/dev/null", "rw");
25:     return;
27: }
```

Test result

```
 ==1987== FILE DESCRIPTORS: 4 open at exit.
==1987== Open file descriptor 4: /dev/null
==1987==    at 0x3FFD2DB310: __open_nocancel (in /lib64/libc-2.12.so)
==1987==    by 0x3FFD272AEE: _IO_file_fopen@@GLIBC_2.2.5 (in /lib64/libc-2.12.so)
==1987==    by 0x3FFD266F25: __fopen_internal (in /lib64/libc-2.12.so)
==1987==    by 0x400921: fdLeak() (test1.cc:25)
==1987==    by 0x400A80: main (test1.cc:76)
==1987==
==1987== Open file descriptor 3: /dev/null
==1987==    at 0x3FFD2DB310: __open_nocancel (in /lib64/libc-2.12.so)
```

```
==1987==    by 0x40090F: fdLeak() (test1.cc:24)
==1987==    by 0x400A80: main (test1.cc:76)
```

case4

Source code

```
 ......
29: void fdClose()
30: {
31:     int fd1 = open("/dev/null", O_RDWR);
32:     int fd2 = open("/dev/console", O_RDWR); //This open will fail
33:     int fd3 = 1025;
34:
35:     close(fd1);
36:     close(fd1); //Double close will not be catch
37:     close(fd2);
38:     close(fd3);
39:     return;
40: }
```

Test result

```
 ==1987== Warning: invalid file descriptor -1 in syscall close()
==1987== Warning: invalid file descriptor 1025 in syscall close()
```

case5

Source code

```
......
void tArray1()
{
    int aaa[3];

    aaa[4] = 10;
    return;
}
```

Test result

```
==1987== Process terminating with default action of signal 11 (SIGSEGV)
==1987==  Access not within mapped region at address 0xF0000000A
==1987==    at 0x400A95: main (test1.cc:81)
==1987==  If you believe this happened as a result of a stack
==1987==  overflow in your program's main thread (unlikely but
==1987==  possible), you can try to increase the size of the
==1987==  main thread stack using the --main-stacksize= flag.
==1987==  The main thread stack size used in this run was 10485760.
==1987== Invalid write of size 8
==1987==    at 0x4801661: _vgnU_freeres (vg_preloaded.c:58)
==1987==  Address 0xf00000002 is on thread 1's stack
```

case6

Source code

```
......
int produceStack(unsigned int u32Deep)
{
```

```
11:    /*produce 1MB Stack*/
12:    char stack[1024*1024];
13:
14:    /*print start address of stack*/
15:    printf("%02d: stack = 0x%010lx\n",u32Deep,stack);
16:
17:    if( u32Deep == 1)
18:        return 0;
19:
20:    produceStack((--u32Deep));
21:    return 0;
22: }
```

Test result

```
 produceStack(10);
==28761==
==28761== Process terminating with default action of signal 11 (SIGSEGV)
==28761==  Access not within mapped region at address 0xFFE6004AC
==28761==    at 0x4006EB: produceStack (Valgrind_Memory.c:10)
==28761==  If you believe this happened as a result of a stack
==28761==  overflow in your program's main thread (unlikely but
==28761==  possible), you can try to increase the size of the
==28761==  main thread stack using the --main-stacksize= flag.
==28761==  The main thread stack size used in this run was 10485760.
==28761==
==28761== Process terminating with default action of signal 11 (SIGSEGV)
==28761==  Access not within mapped region at address 0xFFE600498
==28761==    at 0x4A246A8: _vgnU_freeres (vg_preloaded.c:58)
==28761==  If you believe this happened as a result of a stack
==28761==  overflow in your program's main thread (unlikely but
==28761==  possible), you can try to increase the size of the
==28761==  main thread stack using the --main-stacksize= flag.
==28761==  The main thread stack size used in this run was 10485760.
```

```
==28761==
```

case7

Source code

```
 ......

48:    MemoryAlloc = (char*) malloc(sizeof(char) * 1024);
49:    /*Heap Overrun*/
50:    memset(MemoryAlloc,0,1025);
51:    /*Double Free*/
52:    free(MemoryAlloc);
53:    free(MemoryAlloc);
```

Test result

```
 ==28773== Invalid write of size 1
==28773==    at 0x4C30120: memset (vg_replace_strmem.c:1224)
==28773==    by 0x40081D: main (Valgrind_Memory.c:50)
==28773==    by 0x4E57B14: __libc_start_main (in /usr/lib64/libc-2.17.so)
==28773==    by 0x400618: ??? (in /home/jeff/Valgrind/Valgrind_Memory)
==28773==    by 0xFFF0006E7: ???
==28773==    by 0x1B: ???
==28773==    by 0x1: ???
==28773==    by 0xFFF0008B2: ???
==28773==    by 0xFFF0008C4: ???
==28773==  Address 0x51f7440 is 0 bytes after a block of size 1,024 alloc'd
==28773==    at 0x4C29C3D: malloc (vg_replace_malloc.c:299)
==28773==    by 0x400803: main (Valgrind_Memory.c:48)
```

```
==28773==    by 0x4E57B14: __libc_start_main (in /usr/lib64/libc-2.17.so)
==28773==    by 0x400618: ??? (in /home/jeff/Valgrind/Valgrind_Memory)
==28773==    by 0xFFF0006E7: ???
==28773==    by 0x1B: ???
==28773==    by 0x1: ???
==28773==    by 0xFFF0008B2: ???
==28773==    by 0xFFF0008C4: ???
==28773==
==28773== Invalid free() / delete / delete[] / realloc()
==28773==    at 0x4C2AD57: free (vg_replace_malloc.c:530)
==28773==    by 0x400835: main (Valgrind_Memory.c:53)
==28773==    by 0x4E57B14: __libc_start_main (in /usr/lib64/libc-2.17.so)
==28773==    by 0x400618: ??? (in /home/jeff/Valgrind/Valgrind_Memory)
==28773==    by 0xFFF0006E7: ???
==28773==    by 0x1B: ???
==28773==    by 0x1: ???
==28773==    by 0xFFF0008B2: ???
==28773==    by 0xFFF0008C4: ???
==28773==  Address 0x51f7040 is 0 bytes inside a block of size 1,024 free'd
==28773==    at 0x4C2AD57: free (vg_replace_malloc.c:530)
==28773==    by 0x400829: main (Valgrind_Memory.c:52)
==28773==    by 0x4E57B14: __libc_start_main (in /usr/lib64/libc-2.17.so)
==28773==    by 0x400618: ??? (in /home/jeff/Valgrind/Valgrind_Memory)
==28773==    by 0xFFF0006E7: ???
==28773==    by 0x1B: ???
==28773==    by 0x1: ???
==28773==    by 0xFFF0008B2: ???
==28773==    by 0xFFF0008C4: ???
==28773==  Block was alloc'd at
==28773==    at 0x4C29C3D: malloc (vg_replace_malloc.c:299)
==28773==    by 0x400803: main (Valgrind_Memory.c:48)
==28773==    by 0x4E57B14: __libc_start_main (in /usr/lib64/libc-2.17.so)
==28773==    by 0x400618: ??? (in /home/jeff/Valgrind/Valgrind_Memory)
==28773==    by 0xFFF0006E7: ???
==28773==    by 0x1B: ???
==28773==    by 0x1: ???
==28773==    by 0xFFF0008B2: ???
```

```
==28773==    by 0xFFF0008C4: ???
==28773==
```

case8

Source code

```
......
void* func1(void *arg)
{
    int op;
    op = *(int*)arg;

    switch(op)
     {
        case eLOCK_PTHREAD_MUTEX:
            pthread_mutex_lock(&lock1);
            sleep(3);
            pthread_mutex_lock(&lock2);
            /*do something*/
            pthread_mutex_unlock(&lock2);
            pthread_mutex_unlock(&lock1);
           break;
        case eLOCK_SYSV_SEMAPHORE:
            sysv_down(gSemid,0);
            sleep(3);
            sysv_down(gSemid,1);
            /*do something*/
            sysv_up(gSemid,1);
            sysv_up(gSemid,0);
           break;
        case eLOCK_POSIX_SEMAPHORE:
            sem_wait(&posix_sem1);
            sleep(3);
            sem_wait(&posix_sem2);
            /*do something*/
            sem_post(&posix_sem2);
            sem_post(&posix_sem1);
           break;
       default:
           printf("Input error: wrong number (0:pthread mutex,1:SYS V Semaphore,2:POSIX Sem
   }
    pthread_exit(0);
}

void* func2(void *arg)
{
   int op;
   op = *(int*)arg;

   switch(op)
    {
       case eLOCK_PTHREAD_MUTEX:
            pthread_mutex_lock(&lock2);
            sleep(3);
            pthread_mutex_lock(&lock1);
            /*do something*/
            pthread_mutex_unlock(&lock1);
            pthread_mutex_unlock(&lock2);
           break;
       case eLOCK_SYSV_SEMAPHORE:
            sysv_down(gSemid,1);
            sleep(3);
            sysv_down(gSemid,0);
            /*do something*/
            sysv_up(gSemid,0);
            sysv_up(gSemid,1);
           break;
       case eLOCK_POSIX_SEMAPHORE:
            sem_wait(&posix_sem2);
            sleep(3);
            sem_wait(&posix_sem1);
            /*do something*/
            sem_post(&posix_sem1);
            sem_post(&posix_sem2);
           break;
       default:
           printf("Input error: wrong number (0:pthread mutex,1:SYS V Semaphore,2:POSIX Sem
   }
   pthread_exit(0);
}
...
   op = atoi(argv[1]);
    /*initial lock*/
   switch(op)
   {
       /*Tool: helgrind*/
       case eLOCK_PTHREAD_MUTEX:
           pthread_mutex_init(&lock1,NULL);
           pthread_mutex_init(&lock2,NULL);
           break;
       /*Tool: None*/
       case eLOCK_SYSV_SEMAPHORE:
           gSemid = sysv_semphore_create();
           break;
       /*Tool: DRD with --trace-semaphore=yes*/
       case eLOCK_POSIX_SEMAPHORE:
           sem_init(&posix_sem1,0,1);
           sem_init(&posix_sem2,0,1);
           break;
       default:
           printf("Input error: wrong number (0:pthread mutex,1:SYS V Semaphore,2:POSIX Ser
           return -1;
   }

   /*create pthread */
```

```
    pthread_create(&pid1,NULL,func1,&op);
    pthread_create(&pid2,NULL,func2,&op);

    pthread_join(pid1,NULL);
    pthread_join(pid2,NULL);

```

Test result pthread mutex

```
 ==28791==
==28791== [1] mutex_init      mutex 0x6020e0                (init lock1)
==28791== [1] mutex_init      mutex 0x602140                (init lock2)
==28791== [1] mutex_init      mutex 0xfff0004e0
==28791== [1] mutex_ignore_ordering mutex 0xfff0004e0
==28791== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 0
==28791== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 0
==28791== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28791== [2] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 1
==28791== [2] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 1
==28791== [2] mutex_unlock    mutex 0xfff0004e0 rc 1
==28791== [2] mutex_trylock   mutex 0x6020e0 rc 0 owner 0      (fun1  pthread_mutex_lock(&lo
==28791== [2] post_mutex_lock mutex 0x6020e0 rc 0 owner 0
==28791== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 2
==28791== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 2
==28791== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28791== [1] mutex_destroy   mutex 0xfff0004e0 rc 0 owner 1
==28791== [1] mutex_init      mutex 0xfff0004e0
==28791== [1] mutex_ignore_ordering mutex 0xfff0004e0
==28791== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 0
==28791== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 0
==28791== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28791== [3] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 1
==28791== [3] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 1
```

```
==28791== [3] mutex_unlock    mutex 0xfff0004e0 rc 1
==28791== [3] mutex_trylock   mutex 0x602140 rc 0 owner 0      (fun2  pthread_mutex_lock(&lo
==28791== [3] post_mutex_lock mutex 0x602140 rc 0 owner 0
==28791== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 3
==28791== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 3
==28791== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28791== [1] mutex_destroy   mutex 0xfff0004e0 rc 0 owner 1
```

SYS V Semaphore (can't handle with SYS V semaphore)

```
 ==28799==
==28799== [1] mutex_init      mutex 0xfff0004e0
==28799== [1] mutex_ignore_ordering mutex 0xfff0004e0
==28799== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 0
==28799== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 0
==28799== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28799== [2] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 1
==28799== [2] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 1
==28799== [2] mutex_unlock    mutex 0xfff0004e0 rc 1
==28799== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 2
==28799== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 2
==28799== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28799== [1] mutex_destroy   mutex 0xfff0004e0 rc 0 owner 1
==28799== [1] mutex_init      mutex 0xfff0004e0
==28799== [1] mutex_ignore_ordering mutex 0xfff0004e0
==28799== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 0
==28799== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 0
==28799== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28799== [3] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 1
==28799== [3] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 1
==28799== [3] mutex_unlock    mutex 0xfff0004e0 rc 1
==28799== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 3
==28799== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 3
```

```
==28799== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28799== [1] mutex_destroy   mutex 0xfff0004e0 rc 0 owner 1
```

POSIX Semaphore

```
 ==28807==
==28807== [1] sem_init      0x602120 value 1
==28807== [1] sem_init      0x602180 value 1
==28807== [1] mutex_init      mutex 0xfff0004e0
==28807== [1] mutex_ignore_ordering mutex 0xfff0004e0
==28807== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 0
==28807== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 0
==28807== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28807== [2] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 1
==28807== [2] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 1
==28807== [2] mutex_unlock    mutex 0xfff0004e0 rc 1
==28807== [2] sem_wait      0x602120 value 1 -> 0
==28807== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 2
==28807== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 2
==28807== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28807== [1] mutex_destroy   mutex 0xfff0004e0 rc 0 owner 1
==28807== [1] mutex_init      mutex 0xfff0004e0
==28807== [1] mutex_ignore_ordering mutex 0xfff0004e0
==28807== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 0
==28807== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 0
==28807== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28807== [3] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 1
==28807== [3] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 1
==28807== [3] mutex_unlock    mutex 0xfff0004e0 rc 1
==28807== [3] sem_wait      0x602180 value 1 -> 0
==28807== [1] mutex_trylock   mutex 0xfff0004e0 rc 0 owner 3
==28807== [1] post_mutex_lock mutex 0xfff0004e0 rc 0 owner 3
==28807== [1] mutex_unlock    mutex 0xfff0004e0 rc 1
==28807== [1] mutex_destroy   mutex 0xfff0004e0 rc 0 owner 1
```