ThreadSanitizer

Andre Meyering

24. Februar 2019

Inhaltsverzeichnis

Aktuelle Situation

- parallele Ausführung wird immer wichtiger
- Code wird immer paralleler

Aktuelle Situation

- parallele Ausführung wird immer wichtiger
- Code wird immer paralleler
- immer mehr Abstraktionen
- Code wird komplexer
- Race-Conditions schwer zu entdecken

Wie Race-Conditions finden?

- viel Nachdenken

Wie Race-Conditions finden?

- viel Nachdenken
- viel Kopfzerbrechen

Wie Race-Conditions finden?

- viel Nachdenken
- viel Kopfzerbrechen
- viele Tränen

Lösung - Thead Sanitizer

ThreadSanitizer is a tool that detects data races. It consists of a compiler instrumentation module and a run-time library.

- von Google als Teil von LLVM entwickelt
- `https://clang.llvm.org/docs/ThreadSanitizer.html`

Anforderungen

- Linux, NetBSD, FreeBSD — 64 Bit

[1] https://www.phoronix.com/scan.php?page=news_item&px=MTIzOTU

Anforderungen

- Linux, NetBSD, FreeBSD — 64 Bit
- ab GCC 4.8 (2012)[1]
- ab Clang ~3.5

[1] https://www.phoronix.com/scan.php?page=news_item&px=MTIzOTU

Anforderungen

- Linux, NetBSD, FreeBSD — 64 Bit
- ab GCC 4.8 (2012)[1]
- ab Clang ~3.5
- Erfordert die ThreadSanitizer Library
- `sudo apt install libtsan0`
- benötigt Position Independent Code (`-fPIE`)

[1] https://www.phoronix.com/scan.php?page=news_item&px=MTIzOTU

Funktionsweise

- fügt eigene Instruktionen in das Binary ein

Funktionsweise

- fügt eigene Instruktionen in das Binary ein
- `export CFLAGS="-g -O2 -fsanitize=thread -fno-omit-frame-pointer"`
  - Debug Informationen
  - Optimierung für bessere Performance
  - Frame Pointer für besseren Stacktrace

Funktionsweise

- fügt eigene Instruktionen in das Binary ein
- `export CFLAGS="-g -O2 -fsanitize=thread -fno-omit-frame-pointer"`
  - Debug Informationen
  - Optimierung für bessere Performance
  - Frame Pointer für besseren Stacktrace
- Error zur Runtime falls Race-Condition entdeckt wurde

pthread — Ausgabe

```
1  #include <pthread.h>
2
3  int Global;
4
5  void* Thread1(void* x) {
6      Global = 42;
7      return x;
8  }
9  int main() {
10     pthread_t t;
11     pthread_create(&t, NULL, Thread1, NULL);
12     Global = 43;
13     pthread_join(t, NULL);
14     return Global;
15 }
```

```
gcc -g -O2 -fsanitize=thread
   -fno-omit-frame-pointer
   -o pthread_error pthread_error.c
```

pthread

```
1  ==================
2  WARNING: ThreadSanitizer: data race (pid=19409)
3    Write of size 4 at 0x556de1400014 by main thread:
4      #0 main <path>/basic_pthread.c:12 (basic_pthread+0x913)
5
6    Previous write of size 4 at 0x556de1400014 by thread T1:
7      #0 Thread1 <path>/basic_pthread.c:6 (basic_pthread+0xa90)
8      #1 <null> <null> (libtsan.so.0+0x296bd)
9
10   Location is global 'Global' of size 4 at 0x556de1400014 (basic_pthread
         +0x000000201014)
11
12   Thread T1 (tid=19411, finished) created by main thread at:
13     #0 pthread_create <null> (libtsan.so.0+0x2bcfe)
14     #1 main <path>/basic_pthread.c:9 (basic_pthread+0x907)
15
16 SUMMARY: ThreadSanitizer: data race <path>/basic_pthread.c:12 in main
17 ==================
18 ThreadSanitizer: reported 1 warnings
```

OpenMP

```c
#include <stdio.h>
#include <omp.h>

int main () {
    int i = 0;
    #pragma omp parallel
    {
        #pragma omp critical
        {
            ++i;
        }
    }
    printf("Ausgabe: %d\n", i);
    return i;
}
```

Wo ist die Race-Condition?

OpenMP

```c
#include <stdio.h>
#include <omp.h>

int main () {
    int i = 0;
    #pragma omp parallel
    {
        #pragma omp critical
        {
            ++i;
        }
    }
    printf("Ausgabe: %d\n", i);
    return i;
}
```

Wo ist die Race-Condition?

- es gibt keine

OpenMP

```
1  #include <stdio.h>
2  #include <omp.h>
3
4  int main () {
5      int i = 0;
6      #pragma omp parallel
7      {
8          #pragma omp critical
9          {
10             ++i;
11         }
12     }
13     printf("Ausgabe: %d\n", i);
14     return i;
15 }
```

Wo ist die Race-Condition?

- es gibt keine
- dennoch Fehlermeldung

OpenMP - ThreadSanitizer

- OpenMP hat TSAN Unterstützung
- `https://xrunhprof.wordpress.com/2018/08/27/tsan-with-openmp/`
- CMake Flag: `-DLIBOMP_TSAN_SUPPORT=ON`

Beispiel

```
clang -fno-omit-frame-pointer -g
   -isystem/path/to/omp/include -L/path/to/omp/lib
   -fopenmp -fsanitize=thread file.c -o out
```

Beispiel

```
1  #include <stdio.h>
2  #include <omp.h>
3
4  #define N 10000
5
6  int main (int argc, char **argv)
7  {
8       int a[N];
9       int j = N;
10      // [...] initialisiere array
11      #pragma omp parallel for
12      for (int i = 0; i < N - 2; i++) {
13           #pragma omp critical
14           a[i] = a[i] + a[j];
15           j--;
16      }
17 }
```

Wer findet die Race-Condition?

Fehlermeldung

```
1  ==================
2  WARNING: ThreadSanitizer: data race (pid=13641)
3    Read of size 4 at 0x7fff79a01170 by main thread:
4      #0 .omp_outlined. openmp_error.c:14 (openmp_error+0x00000049b5a2)
5      #1 __kmp_invoke_microtask <null> (libomp.so+0x000000077842)
6      #2 __libc_start_main /build/glibc-t3gR2i/glibc-2.23/csu/libc-start.c
           :291 (libc.so.6+0x00000002082f)
7
8    Previous write of size 4 at 0x7fff79a01170 by thread T1:
9      #0 .omp_outlined. openmp_error.c:14 (openmp_error+0x00000049b5d6)
10     #1 __kmp_invoke_microtask <null> (libomp.so+0x000000077842)
11
12   Location is stack of main thread.
13
14   Thread T1 (tid=13643, running) created by main thread at:
15     #0 pthread_create tsan_interceptors.cc:902:3 (openmp_error+0
           x00000043db75)
16     #1 __kmp_create_worker <null> (libomp.so+0x00000006c364)
17     #2 __libc_start_main /build/glibc-t3gR2i/glibc-2.23/csu/libc-start.c
           :291 (libc.so.6+0x00000002082f)
18
19 SUMMARY: ThreadSanitizer: data race openmp_error.c:14 in .omp_outlined.
20 ==================
21 ThreadSanitizer: reported 1 warnings
```

Achtung

Normales OpenMP

- ThreadSanitizer meckert bei jedem Durchlauf
- false-positives

ThreadSanitizer OpenMP

- ThreadSanitizer meckert beim pthread Beispiel gar nicht
- keine false-positives
- meckert beim OpenMP Beispiel
- viele false-negatives (läuft nicht immer in race-condition)

Fazit

ThreadSanitizer

- sehr nützlich um Race-Conditions zu finden
- verlangsamt Code um das 10 bis 15 fache
- ein Muss für neue Projekte

OpenMP + ThreadSanitizer

- nützlich bei OpenMP Projekten
- keine false-positives bei `#pragma omp parallel`
- meldet nicht immer Race-Conditions
  (es muss eine stattfinden)

Ausblick

Weitere Sanitizer

- AddressSanitizer
  - use-after-free error
  - LeakSanitizer
- MemorySanitizer
  - unitialized memory read
- UndefinedBehaviorSanitizer
  - signed integer overflow
  - integer divide by zero

Danke

- Präsentation und Projekt unter:
  `https://github.com/bugwelle/bugwelle-lightning-talks`

Fragen?