KYLINSOFT
麒麟软件

KCSAN
Linux内核数据竞态探测器

2021.03

KCSAN

- 背景
- KCSAN简介
- KCSAN原理
- KCSAN特性
- KCSAN现状
- 示例演示

背景

- **什么是竞态？**
  - Multiple execution units access the same memory location
  - Those accesses are unordered — not protected by a lock, for example
  - At least one of those accesses is a write.
- **竞态的影响？**
  - 应用看到的视角不一样，导致程序运行逻辑混乱
  - 具有随机性，没有精准的测试用例可以复现
  - 调试困难，尤其是在内核空间

背景

- **常见的竞态场景**
  - 两个线程在未加锁的情况下并发读写同一个全局变量
  - 线程和中断在未加锁的情况下并发读写同一个全局变量

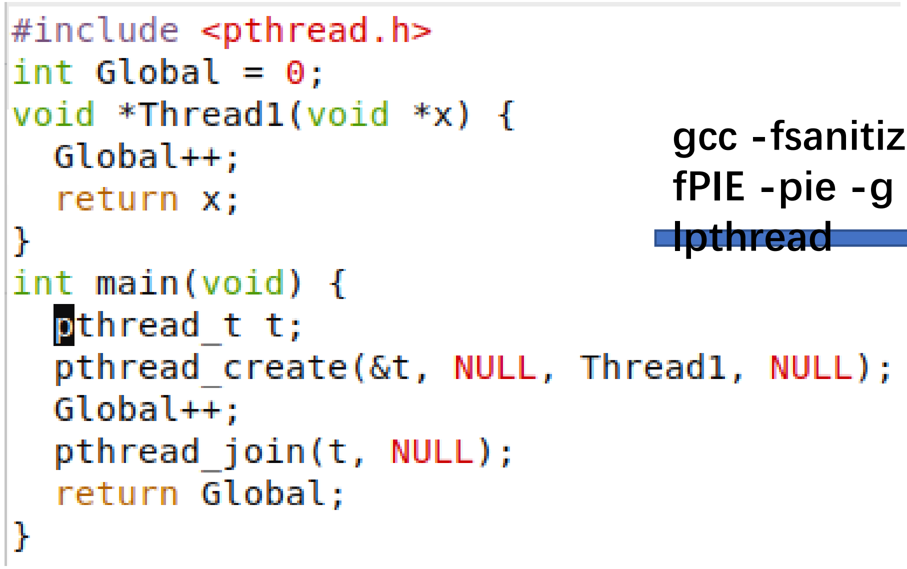
Picture from topic "Data-race detection in the Linux kernel" in 2020 LPC

背景

- **应用程序的竞态检测**
  - Static Analysis
  - Valgrind/Helgrind
  - ThreadSanitizer
    - 运行时动态检测
    - gcc内生支持（4.8以后）

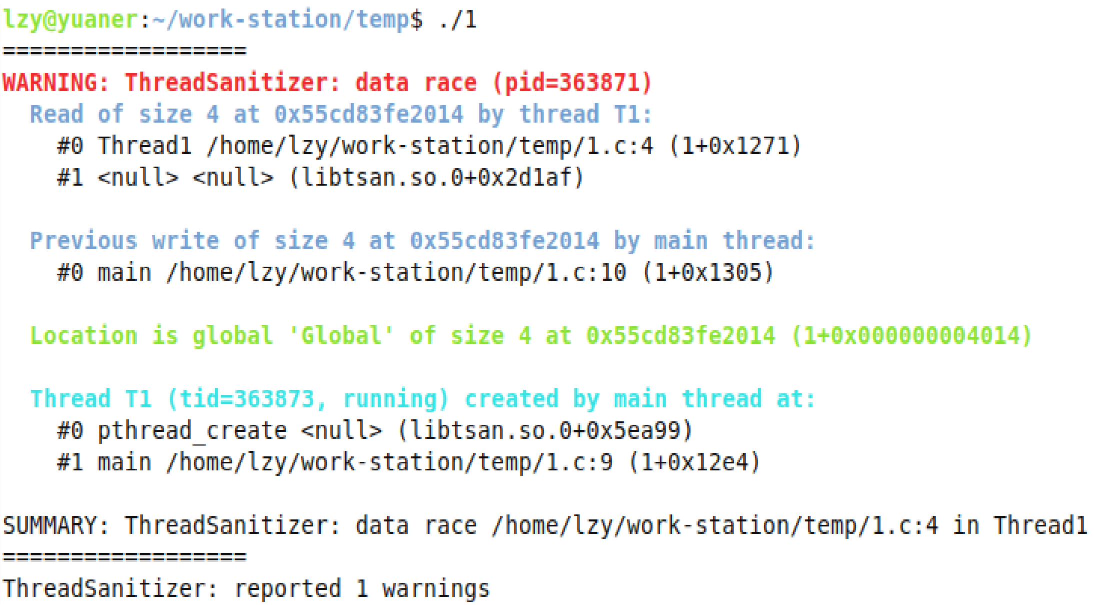
背景

- **应用程序的竞态检测-**ThreadSanitizer



```c
#include <pthread.h>
int Global = 0;
void *Thread1(void *x) {
  Global++;
  return x;
}
int main(void) {
  pthread_t t;
  pthread_create(&t, NULL, Thread1, NULL);
  Global++;
  pthread_join(t, NULL);
  return Global;
}
```

gcc -fsanitize=thread -fPIE -pie -g -o 1 1.c -lpthread



```
lzy@yuaner:~/work-station/temp$ ./1
==================
WARNING: ThreadSanitizer: data race (pid=363871)
  Read of size 4 at 0x55cd83fe2014 by thread T1:
    #0 Thread1 /home/lzy/work-station/temp/1.c:4 (1+0x1271)
    #1 <null> <null> (libtsan.so.0+0x2d1af)

  Previous write of size 4 at 0x55cd83fe2014 by main thread:
    #0 main /home/lzy/work-station/temp/1.c:10 (1+0x1305)

  Location is global 'Global' of size 4 at 0x55cd83fe2014 (1+0x000000004014)

  Thread T1 (tid=363873, running) created by main thread at:
    #0 pthread_create <null> (libtsan.so.0+0x5ea99)
    #1 main /home/lzy/work-station/temp/1.c:9 (1+0x12e4)

SUMMARY: ThreadSanitizer: data race /home/lzy/work-station/temp/1.c:4 in Thread1
==================
ThreadSanitizer: reported 1 warnings
```

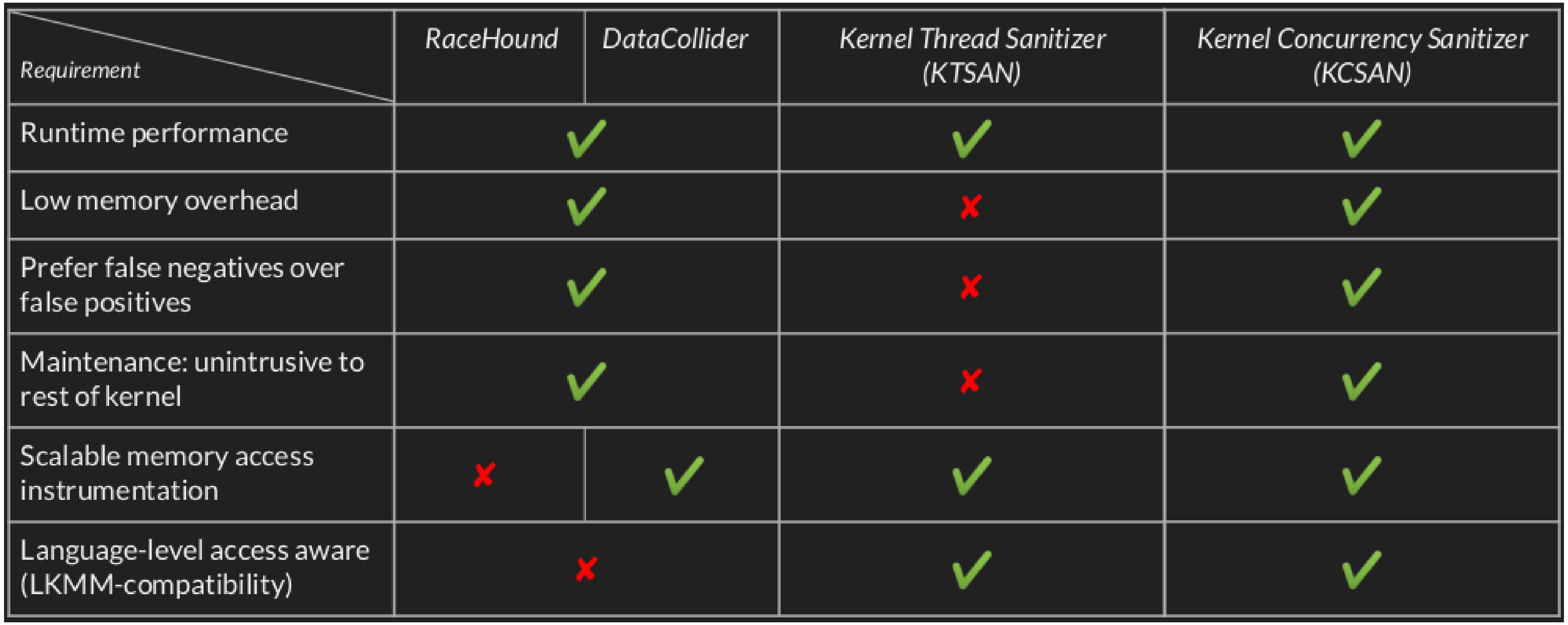
背景

KYLINSOFT
麒麟软件

- **内核数据竞态探测器的产生**
  - **内核竞态问题相比应用空间的竞态问题更隐晦，更难调**
    - 内核非常庞大，各个子系统交织在一起
    - 模块化开发，需要对模块上面的框架非常了解
    - 各种同步机制，锁，原子变量，内存栅，完成量等

背景

- 内核数据竞态探测器比较



Requirement	RaceHound	DataCollider	Kernel Thread Sanitizer (KTSAN)	Kernel Concurrency Sanitizer (KCSAN)
Runtime performance	✓		✓	✓
Low memory overhead	✓		✗	✓
Prefer false negatives over false positives	✓		✗	✓
Maintenance: unintrusive to rest of kernel	✓		✗	✓
Scalable memory access instrumentation	✗	✓	✓	✓
Language-level access aware (LKMM-compatibility)	✗		✓	✓

Picture from topic “Data-race detection in the Linux kernel” in 2020 LPC

KCSAN简介

- KCSAN（the Kernel Concurrency Sanitizer）是google开发并开源的一种基于内存采样的内核竞态Sanitizer, 常见的Sanitizer还有KASAN, UBSAN等。

- KCSAN使用编译器Instrument，通过在内存访问前“插桩”的方式来检测内存并发访问问题。

- 2019年底已合入Linux 内核主线(v5.8)

- 项目地址：https://github.com/google/ktsan/wiki/KCSAN

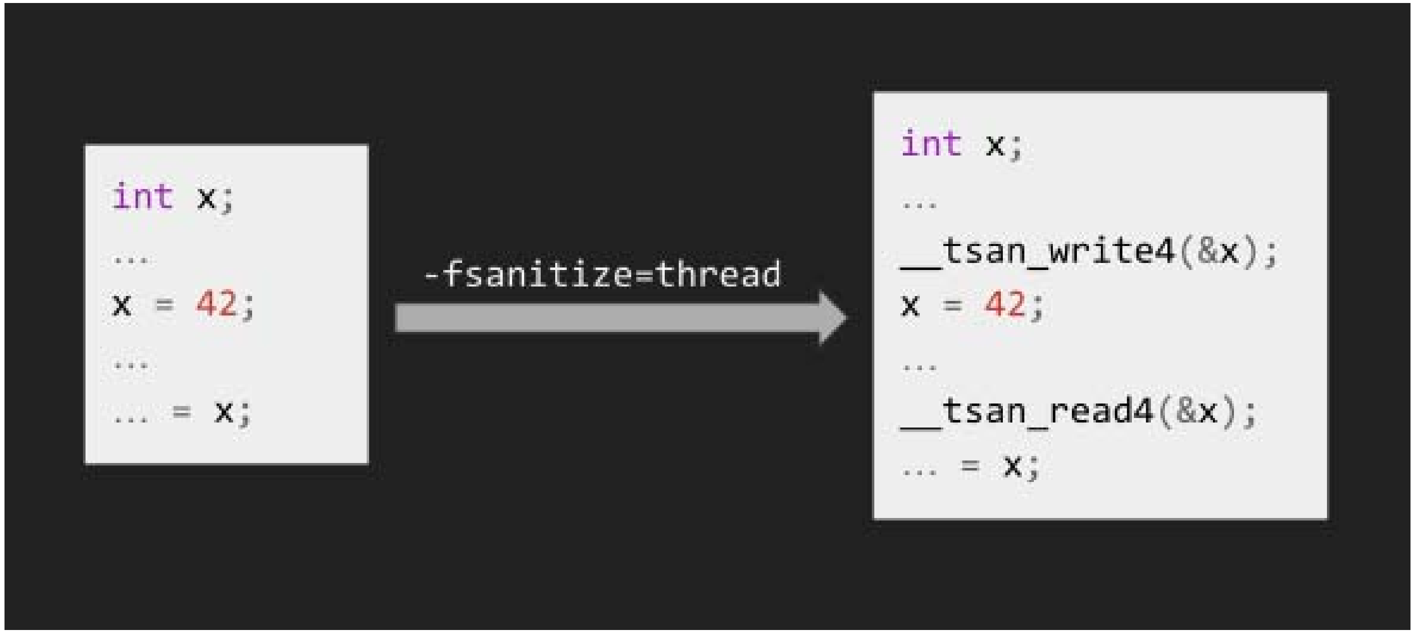
- 邮件列表：kasan-dev@googlegroups.com (open list:KCSAN)

KCSAN原理

- **编译时插桩**: Instrument Memory Accesses
- **运行时检测**： Runtime Check
  - Set watchpoint, and stall access.
  - If watchpoint already exists □ race.
  - If value changed □ race.
  - Stall accesses with random delays to increase chance to observe race.

KCSAN原理

- **编译时插桩：** Instrument Memory Accesses

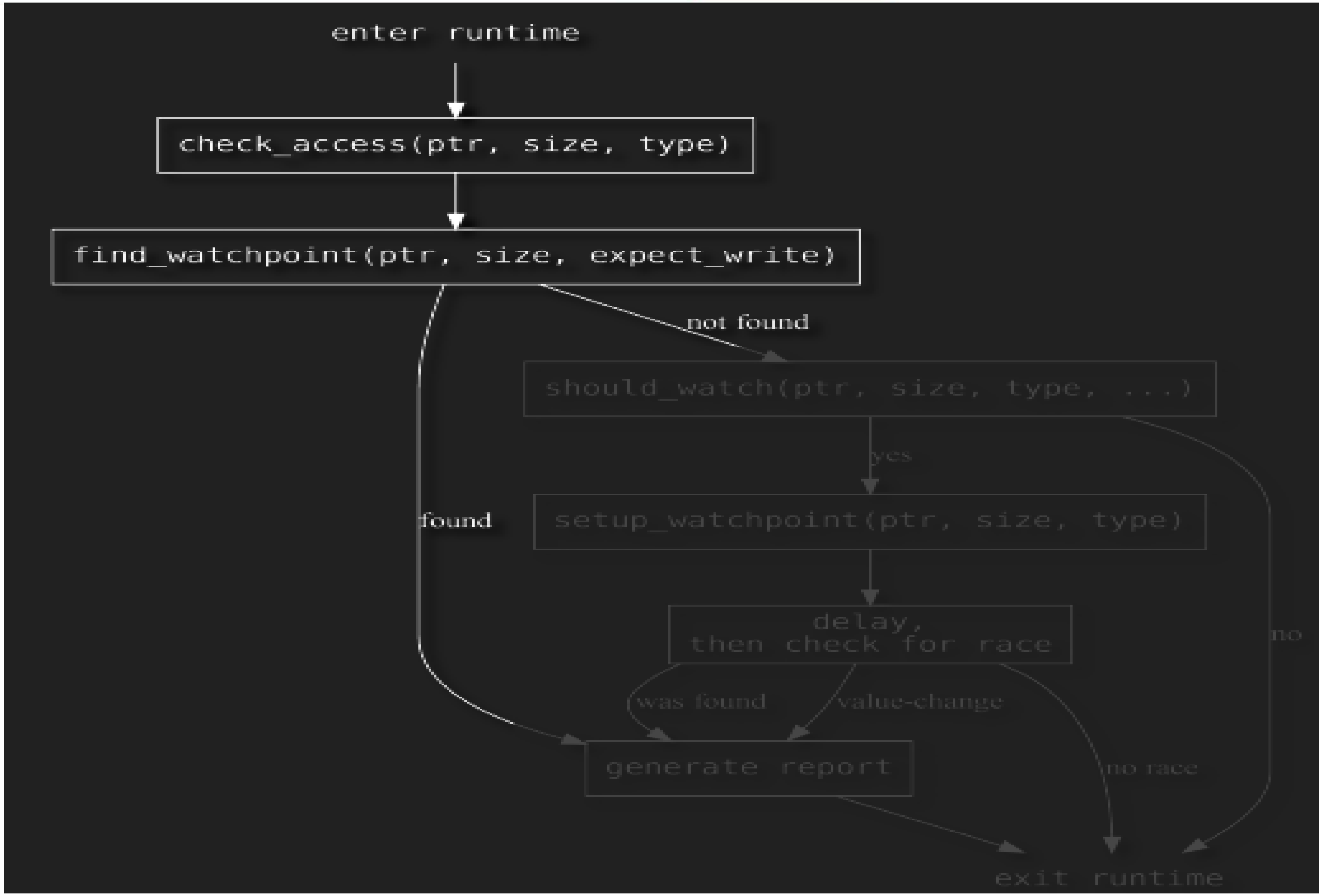


```
int x;
...
x = 42;
...
... = x;
```

```
int x;
...
__tsan_write4(&x);
x = 42;
...
__tsan_read4(&x);
... = x;
```

KCSAN原理

- 运行时监测
  Runtime Check

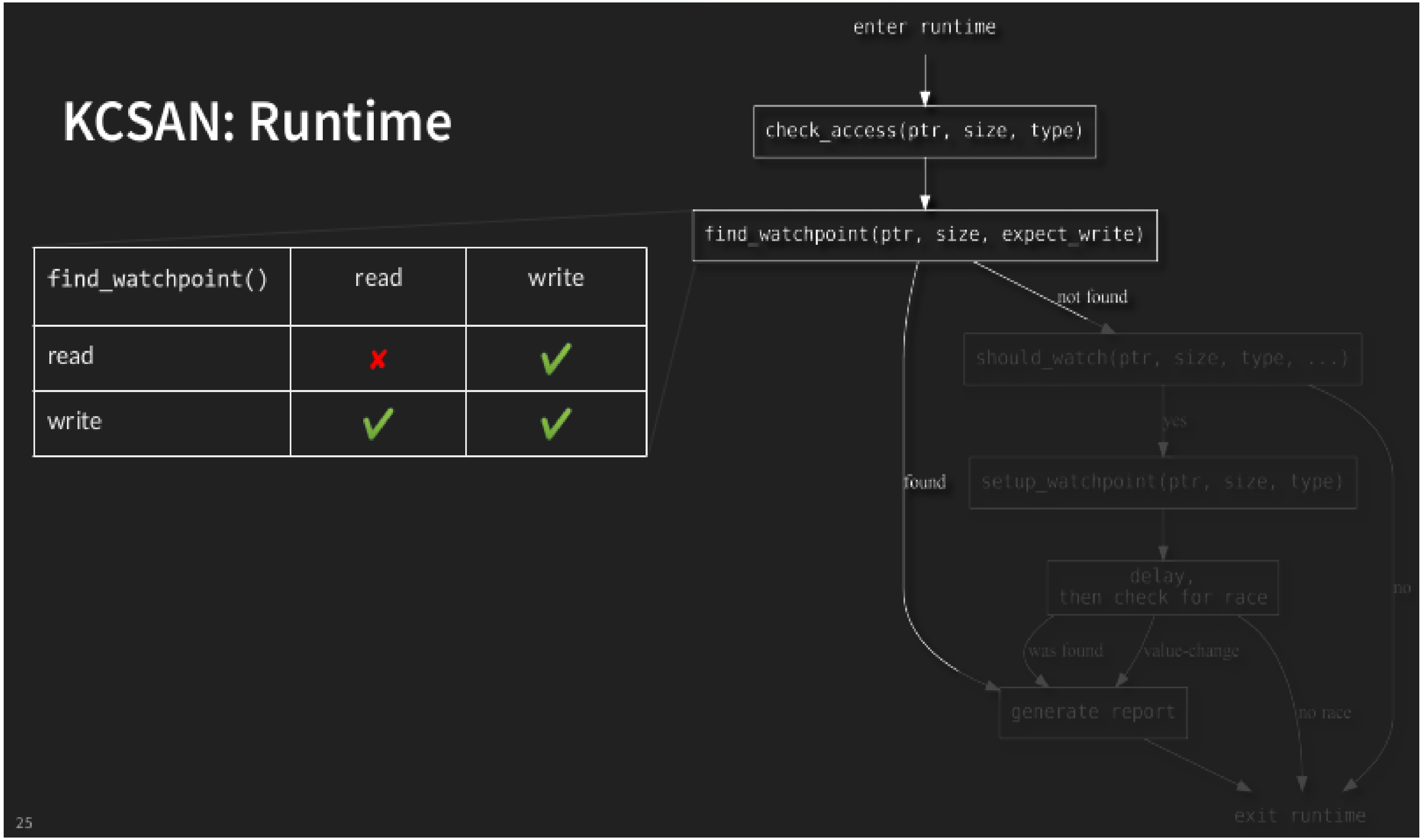

enter runtime
check_access(ptr, size, type)
find_watchpoint(ptr, size, expect_write)
not found
should_watch(ptr, size, type, ...)
yes
found
setup_watchpoint(ptr, size, type)
delay,
then check for race
no
was found
value-change
generate report
no race
exit runtime

Picture from topic “Data-race detection in the Linux kernel” in 2020 LPC

KCSAN原理

- **运行时监测：**

find_watchpoint()



KCSAN: Runtime

find_watchpoint()	read	write
read	✗	✓
write	✓	✓

enter runtime

check_access(ptr, size, type)

find_watchpoint(ptr, size, expect_write)

not found

should_watch(ptr, size, type, ...)

yes

found

setup_watchpoint(ptr, size, type)

delay,
then check for race

no

was found

value-change

generate report

no race

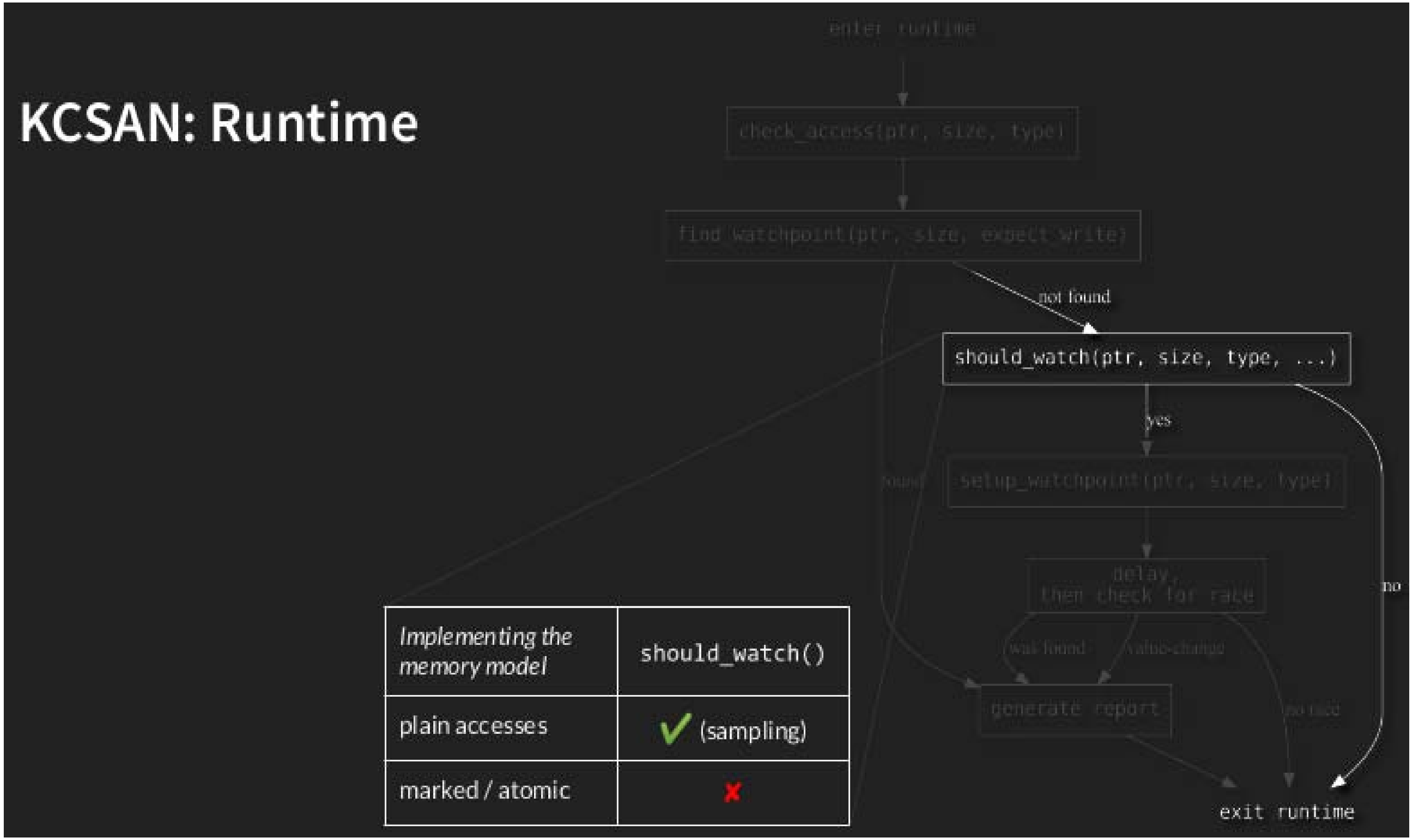
exit runtime

25

Picture from topic “Data-race detection in the Linux kernel” in 2020 LPC

KCSAN原理

- 运行时监测：
  should_watch()



KCSAN: Runtime

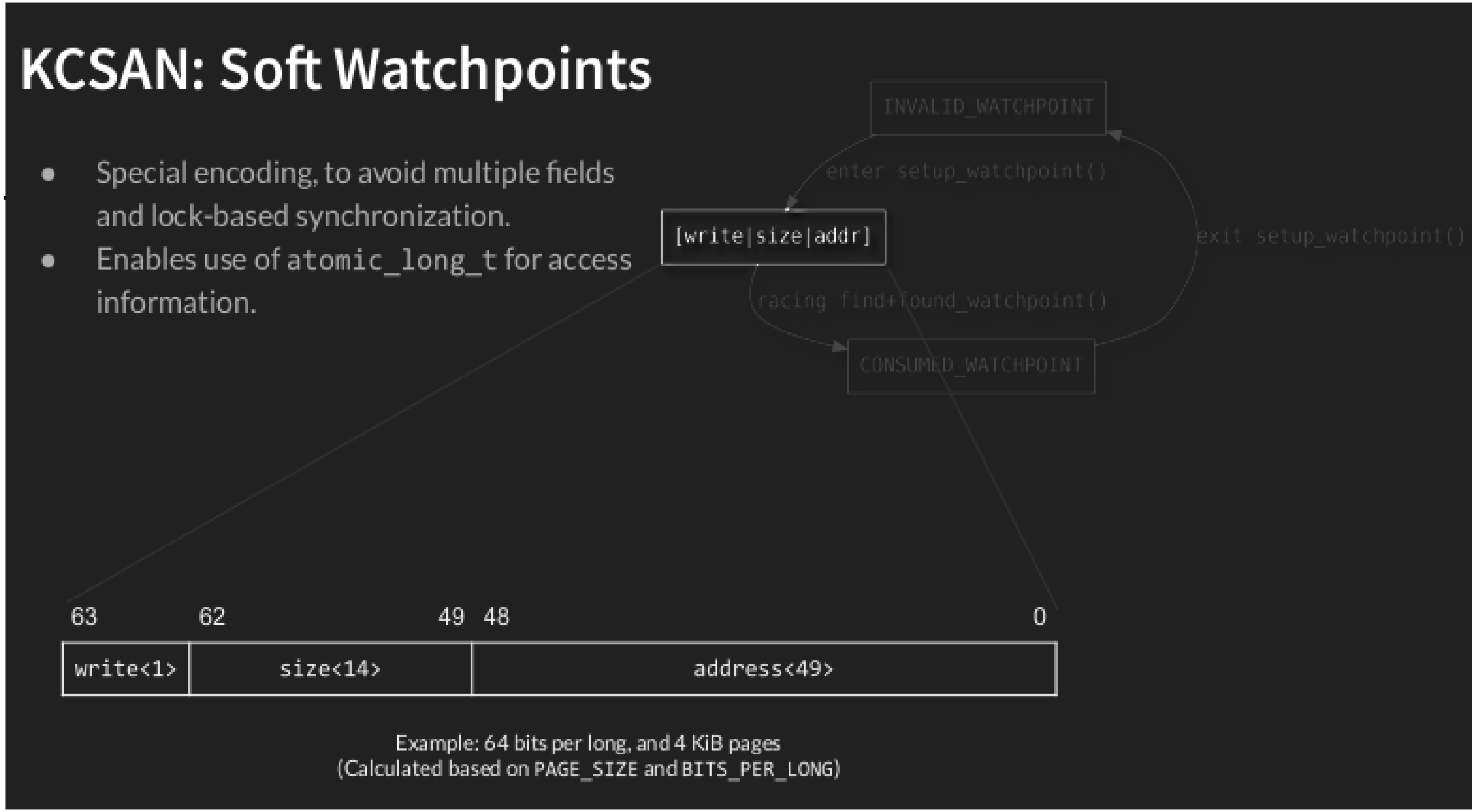
enter runtime
check_access(ptr, size, type)
find_watchpoint(ptr, size, expect_write)
not found
should_watch(ptr, size, type, ...)
yes
found
setup_watchpoint(ptr, size, type)
delay,
then check for race
wat found
value-change
no race
no
generate report
exit runtime

Implementing the memory model	should_watch()
plain accesses	✓ (sampling)
marked / atomic	✗

Picture from topic “Data-race detection in the Linux kernel” in 2020 LPC

KCSAN原理

- 运行时监测：setup_watchpoin



KCSAN: Soft Watchpoints

- Special encoding, to avoid multiple fields and lock-based synchronization.
- Enables use of atomic_long_t for access information.

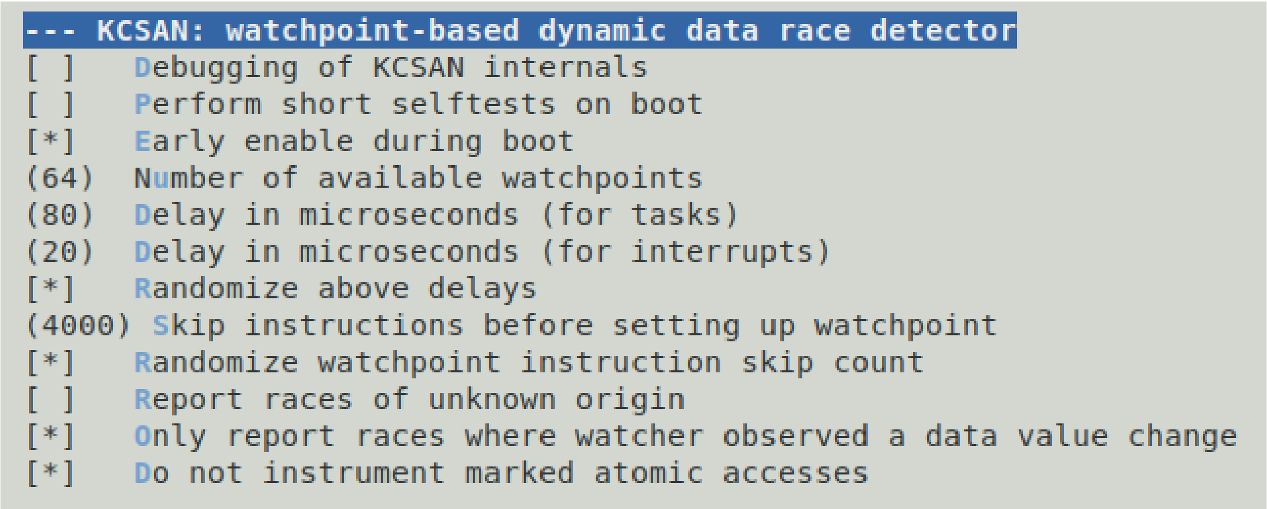
63	62 … 49	48 … 0
write<1>	size<14>	address<49>

Example: 64 bits per long, and 4 KiB pages
(Calculated based on PAGE_SIZE and BITS_PER_LONG)

Picture from topic “Data-race detection in the Linux kernel” in 2020 LPC

KASAN特性

- 支持debugfs动态过滤
- 支持Annotations和ASSERT宏
- 模块化设计，各种参数可调：



```
--- KCSAN: watchpoint-based dynamic data race detector
[ ]    Debugging of KCSAN internals
[ ]    Perform short selftests on boot
[*]    Early enable during boot
(64)   Number of available watchpoints
(80)   Delay in microseconds (for tasks)
(20)   Delay in microseconds (for interrupts)
[*]    Randomize above delays
(4000)  Skip instructions before setting up watchpoint
[*]    Randomize watchpoint instruction skip count
[ ]    Report races of unknown origin
[*]    Only report races where watcher observed a data value change
[*]    Do not instrument marked atomic accesses
```

KASAN现状

- 只能用于调试环境，不能用于生产环境
  - Memory Overhead： 需跟踪每一次全局资源的访问
  - Slowdown System Performance: 5-10X

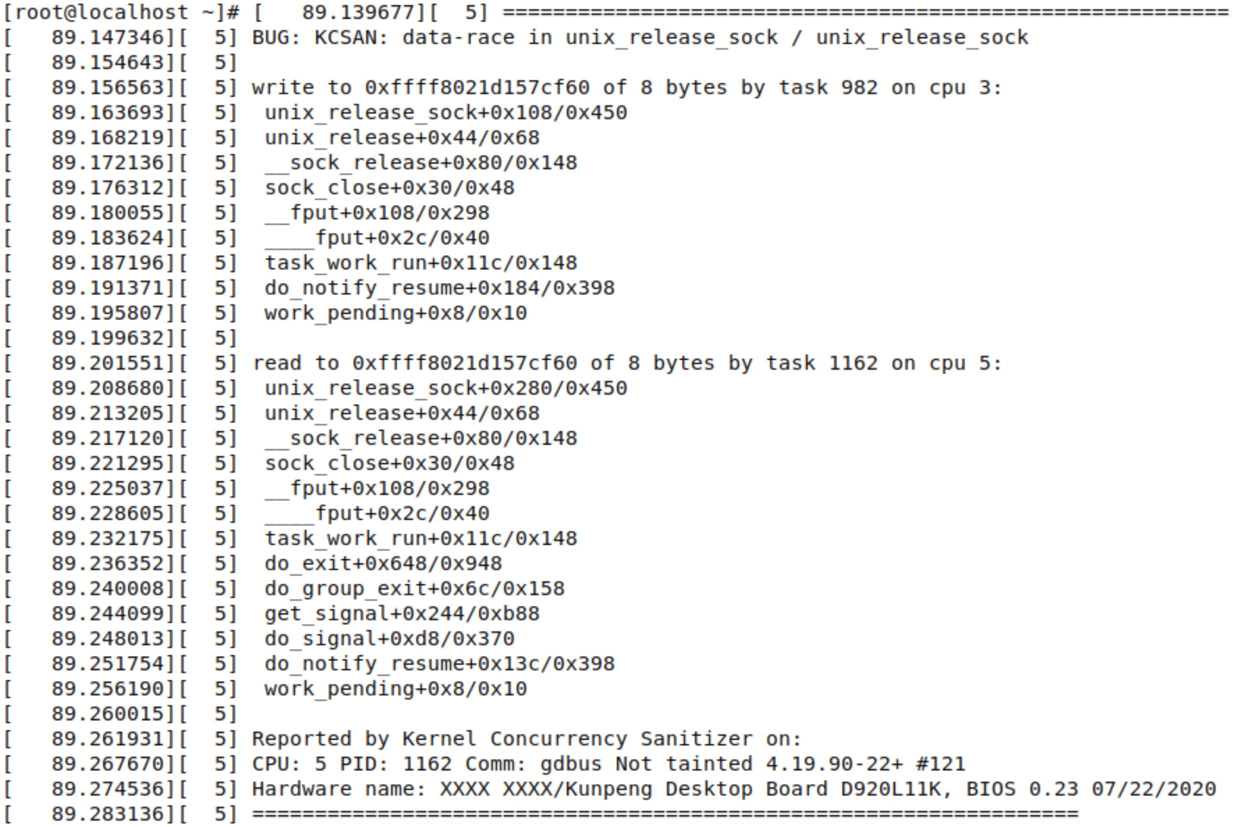
- 探测效果具有一定的局限性
  - 由于是运行时动态探测，对于未运行的代码中存在的竞态无法探测
  - 基于采样机制，不能保证每次都能捕获到竞态

- 目前只支持X86架构

示例

• 演示

- 内核开启KCSAN: CONFIG_KCAN=y
- 目前官方只支持x86架构
- 编译器版本要求： GCC 11



```
[root@localhost ~]# [   89.139677][  5] ==========================================================
[   89.147346][  5] BUG: KCSAN: data-race in unix_release_sock / unix_release_sock
[   89.154643][  5]
[   89.156563][  5] write to 0xffff8021d157cf60 of 8 bytes by task 982 on cpu 3:
[   89.163693][  5]  unix_release_sock+0x108/0x450
[   89.168219][  5]  unix_release+0x44/0x68
[   89.172136][  5]  __sock_release+0x80/0x148
[   89.176312][  5]  sock_close+0x30/0x48
[   89.180055][  5]  __fput+0x108/0x298
[   89.183624][  5]  ____fput+0x2c/0x40
[   89.187196][  5]  task_work_run+0x11c/0x148
[   89.191371][  5]  do_notify_resume+0x184/0x398
[   89.195807][  5]  work_pending+0x8/0x10
[   89.199632][  5]
[   89.201551][  5] read to 0xffff8021d157cf60 of 8 bytes by task 1162 on cpu 5:
[   89.208680][  5]  unix_release_sock+0x280/0x450
[   89.213205][  5]  unix_release+0x44/0x68
[   89.217120][  5]  __sock_release+0x80/0x148
[   89.221295][  5]  sock_close+0x30/0x48
[   89.225037][  5]  __fput+0x108/0x298
[   89.228605][  5]  ____fput+0x2c/0x40
[   89.232175][  5]  task_work_run+0x11c/0x148
[   89.236352][  5]  do_exit+0x648/0x948
[   89.240008][  5]  do_group_exit+0x6c/0x158
[   89.244099][  5]  get_signal+0x244/0xb88
[   89.248013][  5]  do_signal+0xd8/0x370
[   89.251754][  5]  do_notify_resume+0x13c/0x398
[   89.256190][  5]  work_pending+0x8/0x10
[   89.260015][  5]
[   89.261931][  5] Reported by Kernel Concurrency Sanitizer on:
[   89.267670][  5] CPU: 5 PID: 1162 Comm: gdbus Not tainted 4.19.90-22+ #121
[   89.274536][  5] Hardware name: XXXX XXXX/Kunpeng Desktop Board D920L11K, BIOS 0.23 07/22/2020
[   89.283136][  5] ==================================================================
```

在麒麟操作系统上支持KCSAN

银河麒麟高级服务器操作系统同源支持飞腾、龙芯、申威、兆芯、海光、鲲鹏等自主CPU及x86平台， 最新v10 sp1基于红帽R系构建。在麒麟操作系统上支持KCSAN可能需要考虑的点有以下几个方面：

- 内核版本跨度较大： v10 sp1基于4.19内核，KCSAN基于社区5.8开发，内核接口变化较大
- GCC版本： v10 sp1 自带GCC版本为7.3，KCSAN要求GCC 11
- 多平台支持： 目前KCSAN只支持x86平台，需要考虑其他平台上的支持。

参考资源

1. Data-race detection in the Linux kernel：LPC2020-KCSAN.pdf
2. ThreadSanitizer – data race detection in practice
3. Finding race conditions with KCSAN: lwn.net/Articles/802128/
4. Concurrency bugs should fear the big bad data-race detector: lwn.net/Articles/816850/
5. Kernel Concurrency Sanitizer (KCSAN): github.com/google/ktsan/wiki/KCSAN

KYLINSOFT
麒麟软件

麒麟软件有限公司

谢 谢