Snowboard: Finding Kernel Concurrency Bugs through Systematic Inter-thread Communication Analysis

Sishuai Gong (Purdue University)  Deniz Altınbüken (Google Research)
Pedro Fonseca (Purdue University)  Petros Maniatis (Google Research)
A Linux kernel concurrency bug

Delete, modify and re-insert a node

```
list_delete(&node, ...);
... // modify a node
list_add(&node, ...);
```

Walk over each node and check

```
list_for_each_entry(..., node){
... // checks on every node
}
```
A Linux kernel concurrency bug

Kernel thread 1—running seek()

1. list_delete(&node, ...);
   ... // modify a node
   list_add(&node, ...);

Kernel thread 2—running lookup()

2. list_for_each_entry(..., node){
   ... // checks on every node
}

Kernel thread 1: Modifies a node on the list.
Kernel thread 2: Iterates through the list, checking each node.
A Linux kernel concurrency bug

Kernel thread 1—running seek()

1. `list_delete(&node, ...);
   ... // modify a node
   list_add(&node, ...);`

Kernel thread 2—running lookup()

2. `list_for_each_entry(..., node){
   ... // checks on every node
   }

Kernel panic: Null pointer deference

This bug existed in the kernel for over 14 years until Snowboard found it :)

until Snowboard found it :)
Challenges in finding concurrency bugs

**Challenge 1:**
Find error-inducing concurrent inputs

**Challenge 2:**
Find error-inducing interleavings
Finding concurrent inputs is challenging

Concurrent input

Kernel

Thread A
Thread B

Concurrent input

syscall_A(X, Y, Z)
systemcall_F(P, Q, R)
systemcall_O(K, D)

S=syscall_X(H, M)
systemcall_E(G, S, S)
systemcall_Z(A, L, M)

400+ system calls
Various parameters
Data/control dependencies
Finding concurrent inputs + interleavings is even more challenging.

Concurrent input:
- syscall_A(X, Y, Z)
- syscall_F(P, Q, R)
- syscall_O(K, D)
- syscall_X(H, M)
- syscall_E(G, S, S)
- syscall_Z(A, L, M)

Thread A
- P = syscall_O(K, D)
- syscall_F(P, Q, R)

Thread B
- S = syscall_X(H, M)
- syscall_E(G, S, S)
- syscall_Z(A, L, M)

Kernel

Too many possible interleavings

Only a few interleavings expose the bug

Average # of instructions in 2 threads vs. # of possible interleavings

- 1,000,000
- 10,000,000
- 100,000,000
- 1,000,000,000
- 10,000,000,000
- 100,000,000,000
- 1,000,000,000,000

Graph:
- X-axis: Average # of instructions in 2 threads
- Y-axis: # of possible interleavings

Legend:
- Too many possible interleavings
- Only a few interleavings expose the bug
How does Snowboard find concurrency bugs?

1. Predict thread interactions

2. Explore interaction interleavings
Potential memory communication

Kernel thread 1
list_del(&node, ...);
...

Kernel thread 2
list_for_each_entry(..., node) {
...
}

Shared memory
write
read

a PMC

“write before read” or “write after read”?
PMC interleaving

Interleavings of the PMC can lead to concurrency issues

Kernel thread 1

```
list_delete(&node, ...);
...
```

write

Kernel thread 2

```
list_for_each_entry(..., node) {
...
}
```

read

New data/control flow

write before read

write after read

Inconsistent data

write before read

write after read
Snowboard finds concurrency bugs by testing PMCs

1. Find PMCs
   Dynamic sequential input analysis

2. Prioritize PMCs
   Clustering strategy
   - Cluster A
   - Cluster B

3. Test PMCs
   PMC interleaving exploration
   - thread 1
     1. write
   - thread 2
     2. read
Find kernel PMCs—Possible approaches

Approach 1: Brute-force search
- Concurrent input slot machine
- syscall_A
- syscall_B
- syscall_D
- syscall_E
- syscall_F

Does not scale

Approach 2: Static analysis
- list_delete(&s_sibling);
  ...
- list_for_each_entry(sibling, s_sibling);
  ...

Imprecise
Find kernel PMCs—Our approach

Input
- Sequential input 1
  - syscall_A(X, Y, Z)
  - syscall_F(P, Q, R)
  - syscall_O(K, D)
- Sequential input 2
- Sequential input 3

Snowboard
- Dynamic sequential input analysis

Output
- Kernel PMC list
  - PMC-1
  - PMC-2
  - PMC-3
  - ...

Dynamic sequential input analysis

Single-thread execution profile

syscall_A(X, Y, Z)
syscall_F(P, Q, R)
syscall_O(K, D)

Kernel

Execution profile

Shared memory

Arbitrary kernel states?

Single-thread execution profile

syscall_X(H, M)
syscall_E(G, S, S)
syscall_Z(A, L, M)

Kernel

Execution profile

Shared memory

write memory access
read memory access
Snowboard finds concurrency bugs by testing PMCs

1. Find PMCs

Dynamic sequential input analysis

2. Prioritize PMCs

Clustering strategy

Cluster A

Cluster B

3. Test PMCs

PMC interleaving exploration

thread 1

1 write

thread 2

2 read
Prioritize PMCs

Why do we need to prioritize PMCs?

1. Too many PMCs in the kernel
   e.g., we identified 161B PMCs in Linux

2. Testing PMCs is expensive
   e.g., controlling kernel interleavings is expensive
Cluster similar PMCs

Since testing similar execution is less rewarding
Clustering strategy

1. **Cluster similar PMCs**
   Since testing similar execution is less rewarding

2. **Prioritize small clusters**
   Since these are less likely to be tested

Cluster A
1st cluster to test

Cluster B
2nd cluster to test

a unique PMC
Clustering strategy

1. Cluster similar PMCs
   Since testing similar execution is less rewarding

2. Prioritize small clusters
   Since these are less likely to be tested

3. Sample a PMC from each cluster
   Since the rest of the PMCs are similar
Snowboard finds concurrency bugs by testing PMCs

1. Find PMCs
   Dynamic sequential input analysis

2. Prioritize PMCs
   Clustering strategy
   - Cluster A
   - Cluster B

3. Test PMCs
   PMC interleaving exploration
   - Thread 1
     1. Write
   - Thread 2
     2. Read
Test PMCs

PMC testing (Concurrent execution)

- syscall_A(X, Y, Z)
- syscall_F(P, Q, R)
- syscall_O(K, D)
- syscall_X(H, M)
- syscall_E(G, S, S)
- syscall_Z(A, L, M)

Fixed kernel state

Thread 1

- write

Thread 2

- read

Shared memory

Interleaving exploration

Bug detection
Evaluation

We applied Snowboard to recent Linux kernel releases

1. Many bugs have serious impact (e.g. kernel panics, filesystem error).
2. Some bugs existed for years.
Evaluation

Number of concurrency issues

- Snowboard: 7 issues
- Random Concurrent input generation: 2 issues
Evaluation

See paper for more details :)
**Snowboard**  
**Kernel concurrency bugs**

**Potential memory communication (PMC)**  
Pair of write and read accesses to shared resources

1. Find PMCs  
Dynamic sequential input analysis

2. Prioritize PMCs  
Clustering strategy

- Cluster A
- Cluster B

3. Test PMCs  
PMC testing

- thread 1
  - 1. write
- thread 2
  - 2. read

https://github.com/rssys/snowboard

Effective in finding new concurrency bugs

**Snowboard**

**Kernel concurrency bugs**

**Potential memory communication (PMC)**  
Pair of write and read accesses to shared resources

1. Find PMCs  
Dynamic sequential input analysis

2. Prioritize PMCs  
Clustering strategy

- Cluster A
- Cluster B

3. Test PMCs  
PMC testing

- thread 1
  - 1. write
- thread 2
  - 2. read

https://github.com/rssys/snowboard

Effective in finding new concurrency bugs

**Snowboard**

**Kernel concurrency bugs**

**Potential memory communication (PMC)**  
Pair of write and read accesses to shared resources

1. Find PMCs  
Dynamic sequential input analysis

2. Prioritize PMCs  
Clustering strategy

- Cluster A
- Cluster B

3. Test PMCs  
PMC testing

- thread 1
  - 1. write
- thread 2
  - 2. read

https://github.com/rssys/snowboard

Effective in finding new concurrency bugs