aComment:
Mining Annotations from Comments and Code to Detect Interrupt-Related Concurrency Bugs

Lin Tan, University of Waterloo, lintan@uwaterloo.ca
Yuanyuan (YY) Zhou, University of California, San Diego
Yoann Padioleau, Facebook Inc.
OS Concurrency Bugs are a Problem

- Concurrency bugs are pervasive and hard-to-detect.

- Operating System (OS) concurrency bugs can bring down all applications running on top of it.

- OS has a higher percentage of concurrency bugs than application software. [TanTechReport’11]

- 19% of OS driver bugs are concurrency bugs. [RyzhykEuroSys’09]
Interrupts Complicate OS Synchronization

Thread (T1)

Thread (T2)

Lock Acquisition
Failed Lock Acquisition
Lock Release
Interrupts Complicate OS Synchronization

Thread (T1)

Thread (T2)

Lock Acquisition

Failed Lock Acquisition

Lock Release
Interrupts Complicate OS Synchronization

Thread (T1)

Lock Acquisition

Failed Lock Acquisition

Lock Release

Thread (T2)

Context Switch
Interrupts Complicate OS Synchronization

Thread (T1)

Thread (T2)

Lock Acquisition

Failed Lock Acquisition

Lock Release

Context Switch
Interrupts Complicate OS Synchronization

Thread (T1) — Lock Acquisition — Context Switch — Lock Release

Thread (T2) — Failed Lock Acquisition — Lock Release
Interrupts Complicate OS Synchronization

Thread (T1)

Thread (T2)

Lock Acquisition

Failed Lock Acquisition

Lock Release

Context Switch
Interrupts Complicate OS Synchronization

- Lock Acquisition
- Failed Lock Acquisition
- Lock Release

Thread (T1)

Thread (T2)

Context Switch
Interrupts Complicate OS Synchronization

- Lock Acquisition
- Failed Lock Acquisition
- Lock Release

Thread (T1)

Thread (T2)

Context Switch
Interrupts Complicate OS Synchronization

Thread (T1)

Interrupt Handler Thread (TH)

Lock Acquisition

Failed Lock Acquisition
Interrupts Complicate OS Synchronization

Thread (T1)

Interrupt Handler Thread (TH)

Lock Acquisition

Failed Lock Acquisition

Interrupt
Interrupts Complicate OS Synchronization

Thread (T1)

Interrupt Handler
Thread (TH)

Lock Acquisition

Failed Lock Acquisition

Interrupt
Interrupts Complicate OS Synchronization

Thread (T1)

Failed Lock Acquisition

Lock Acquisition

Interrupt Handler Thread (TH)

Interrupt

aComment
Interrupts Complicate OS Synchronization

Lock Acquisition

Failed Lock Acquisition

Thread (T1)

Interrupt Handler Thread (TH)

Interrupt
Interrupts Complicate OS Synchronization

Thread (T1)

Interrupt Handler Thread (TH)

Lock Acquisition

Failed Lock Acquisition

Interrupt

Deadlock
Interrupts Complicate OS Synchronization

Thread (T1) should disable interrupts.

Lock Acquisition
Failed Lock Acquisition

Interrupt Handler Thread (TH)

Deadlock

Interrupts Complicate OS Synchronization

Lock Acquisition
Failed Lock Acquisition

Deadlock
Interrupts Complicate OS Synchronization

- Interrupts can also cause other concurrency bugs.
- Hard to reason about interrupts because
  - Interrupts can happen at anytime.
  - Interrupts are relatively infrequent.
  - OS contains many interrupt handlers.
State-of-Art & Our Solution

- Most effective concurrency bug detection tools [Savage TOCS'97, Choi PLDI'02, Lu ASPLOS'06, Lu SOSP'07, Hammer ICSE'08, Jula OSDI'08, Naik ICSE'09, Burnim ICSE'10, Lai ICSE'10]

- do not consider interrupts

- are dynamic tools designed for user-level applications.

- Dynamic approaches are cumbersome for OS:
  - difficult to instrument OS, low level, many drivers, large code sizes, complexity, ...
State-of-Art & Our Solution

- Most effective concurrency bug detection tools [Savage TOCS'97, Choi PLDI'02, Lu ASPLOS'06, Lu SOSP'07, Hammer ICSE'08, Jula OSDI'08, Naik ICSE'09, Burnim ICSE'10, Lai ICSE'10]
  - do not consider interrupts
  - are dynamic tools designed for user-level applications.

- Dynamic approaches are cumbersome for OS:
  - difficult to instrument OS, low level, many drivers, large code sizes, complexity, ...

- **Our Solution**: Static approach with interrupts in mind
Goal

• Infer

• **Precondition**: If interrupts should have already been disabled or enabled upon entry to a function, and

• **Postcondition**: If interrupts should have already been disabled or enabled upon exit from the function
Goal

• Infer

  • **Precondition**: If interrupts should have already been disabled or enabled upon entry to a function, and
  
  • **Postcondition**: If interrupts should have already been disabled or enabled upon exit from the function
Goal

- Infer

  - **Precondition**: If interrupts should have already been disabled or enabled upon entry to a function, and
  
  - **Postcondition**: If interrupts should have already been disabled or enabled upon exit from the function

- From *comments and code*
Goal

• Infer

Annotations

• **Precondition**: If interrupts should have already been disabled or enabled upon entry to a function, and

• **Postcondition**: If interrupts should have already been disabled or enabled upon exit from the function

• From **comments and code**

• Detect violations to these annotations statically
Inferring Annotations from Comments & Code

linux/kernel/time/tick-oneshot.c:
/* ... Called with interrupts disabled. */
int tick_init_highres(void) {...}
Inferring Annotations from Comments & Code

```c
linux/kernel/time/tick-oneshot.c:
/* ... Called with interrupts disabled. */
int /*@IRQ(D, X)*/ tick_init_highres(void) {...}
```
Inferring Annotations from Comments & Code

```
linux/kernel/time/tick-oneshot.c:
/* ... Called with interrupts disabled. */
int /*@IRQ(D, X)*/ tick_init_highres(void) {...}
```

```
linux/kernel/posix-cpu-timers.c:
void run_posix_cpu_timers(...)
{ BUG_ON(!irqs_disabled()); ... }
```
Inferring Annotations from Comments & Code

**linux/kernel/time/tick-oneshot.c:**

```c
/* ... Called with interrupts disabled. */
int /*@IRQ(D, X)*/ tick_init_highres(void) {...}
```

**linux/kernel/posix-cpu-timers.c:**

```c
void /*@IRQ(D, X)*/ run_posix_cpu_timers(...)
{    BUG_ON(!irqs_disabled());    ...  }
```
Our Contributions

✧ Feasible to extract annotations from comments & code

● Designed new interrupt-related annotations

● Generated 96,821 *interrupt-related* annotations & automatically detected 9 true bugs in the Linux kernel

● These annotations can help developers avoid bugs.

✧ Combining comments & code help extract more annotations and detect more bugs than using comments or code alone.
Outline

• Motivation & Contributions
• Annotation Design
• Annotation Extraction
  • From comments
  • From code
• Annotation Propagation & Bug Detection
• Results: Bug Detection & Annotation Extraction
• Related Work
• Conclusions
Annotation Language Design

@IRQ (Precondition, Postcondition)
Annotation Language Design

@IRQ(D/E/X, D/E/X)

Read our paper for the meaning of value ‘P’.
**Annotation Language Design**

@IRQ (D/E/X, D/E/X)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Interrupts are disabled.</td>
</tr>
<tr>
<td>E</td>
<td>Interrupts are enabled.</td>
</tr>
<tr>
<td>X</td>
<td>Don’t care</td>
</tr>
</tbody>
</table>

Read our paper for the meaning of value ‘P’.
Annotation Language Design

@IRQ (D/E/X , D/E/X)

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Interrupts are disabled.</td>
</tr>
<tr>
<td>E</td>
<td>Interrupts are enabled.</td>
</tr>
<tr>
<td>X</td>
<td>Don’t care</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@IRQ (D, D)</td>
<td>Interrupts are disabled on entry and remain disabled on exit.</td>
</tr>
<tr>
<td>@IRQ (X, E)</td>
<td>Don’t-care on entry and interrupts are enabled on exit.</td>
</tr>
<tr>
<td>@IRQ (X, X)</td>
<td>Our design choice: Either @IRQ (D, D) or @IRQ (E, E)</td>
</tr>
</tbody>
</table>

Read our paper for the meaning of value ‘P’.
### Annotation Extraction From Comments

<table>
<thead>
<tr>
<th>Software</th>
<th>LOC</th>
<th>Sentence</th>
<th>IRQSent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>5.2M</td>
<td>1,024,624</td>
<td>23,662</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>2.4M</td>
<td>420,013</td>
<td>11,117</td>
</tr>
<tr>
<td>NetBSD</td>
<td>3.3M</td>
<td>680,650</td>
<td>23,942</td>
</tr>
<tr>
<td>OpenSolaris</td>
<td>3.7M</td>
<td>535,073</td>
<td>8,074</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>14.6M</strong></td>
<td><strong>2,660,360</strong></td>
<td><strong>66,795</strong></td>
</tr>
</tbody>
</table>

- Millions of lines of comments exist in OSs.
- We analyze comments as is: No need to rewrite comments.
Annotation Extraction From Comments

- /* Neither are the interrupt status bits */ (Linux)
- /* Called with interrupts disabled. */ (OpenSolaris)
- /* Disables interrupts before calling this function */ (NetBSD)
- /* Must be called with interrupts locked out */ (FreeBSD)
Annotation Extraction From Comments

- /* Neither are the interrupt status bits */ (Linux)
- /* Called with interrupts disabled. */ (OpenSolaris)
- /* Disables interrupts before calling this function */ (NetBSD)
- /* Must be called with interrupts locked out */ (FreeBSD)

Contains no annotations
### Annotation Extraction From Comments

Contains no annotations

- /* Neither are the interrupt status bits */ (Linux)
- /* Called with interrupts disabled. */ (OpenSolaris)
- /* Disables interrupts before calling this function */ (NetBSD)
- /* Must be called with interrupts locked out */ (FreeBSD)

<table>
<thead>
<tr>
<th>ID</th>
<th>Heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;call&gt; &amp; &lt;with&gt; &amp; &lt;interrupt&gt; (ordered)</td>
</tr>
<tr>
<td>2</td>
<td>&lt;before&gt; &amp; &lt;disable/enable&gt; &amp; &lt;interrupt&gt; (orderless)</td>
</tr>
<tr>
<td>3</td>
<td>&lt;assume&gt; &amp; &lt;<strong>disable</strong>/enable&gt; &amp; &lt;interrupt&gt; (orderless)</td>
</tr>
</tbody>
</table>
Annotation Extraction From Comments

- /* Neither are the interrupt status bits */ (Linux)
- /* Called with interrupts disabled. */ (OpenSolaris)
- /* Disables interrupts before calling this function */ (NetBSD)
- /* Must be called with interrupts locked out */ (FreeBSD)

<table>
<thead>
<tr>
<th>ID</th>
<th>Heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;call&gt; &amp; &lt;with&gt; &amp; &lt;interrupt&gt; (ordered)</td>
</tr>
<tr>
<td>2</td>
<td>&lt;before&gt; &amp; &lt;disable/enable&gt; &amp; &lt;interrupt&gt; (orderless)</td>
</tr>
<tr>
<td>3</td>
<td>&lt;assume&gt; &amp; &lt;<strong>disable</strong>/enable&gt; &amp; &lt;interrupt&gt; (orderless)</td>
</tr>
</tbody>
</table>

“disable”, “turn off”, “block”, “lock out”, ...
Annotation Extraction From Comments

- /* Neither are the interrupt status bits */ (Linux)
- /* Called with interrupts disabled. */ (OpenSolaris)
- /* Disables interrupts before calling this function */ (NetBSD)
- /* Must be called with interrupts locked out */ (FreeBSD)

<table>
<thead>
<tr>
<th>ID</th>
<th>Heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;call&gt; &amp; &lt;with&gt; &amp; &lt;interrupt&gt; (ordered)</td>
</tr>
<tr>
<td>2</td>
<td>&lt;before&gt; &amp; &lt;disable/enable&gt; &amp; &lt;interrupt&gt; (orderless)</td>
</tr>
<tr>
<td>3</td>
<td>&lt;assume&gt; &amp; &lt;disable/enable&gt; &amp; &lt;interrupt&gt; (orderless)</td>
</tr>
</tbody>
</table>

“disable”, “turn off”, “block”, “lock out”, ...

- Automatically extract function names and the preconditions (D or E).
Annotation Extraction From Code Assertions

Linux/kernel/posix-cpu-timers.c:
void /*@IRQ(D, X)*/ run_posix_cpu_timers(...)
{ BUG_ON(!irqs_disabled()); ... }

- Learn from dynamic assertions
- Can learn invariants from the majority of code

[ErnstICSE'00], [EnglerSOSP'01], [HangalCSE'02], [LiFSE'05], [LivshitsFSE'05], [TanSecurity'08] ...
Annotation Extraction From Code Assertions

- Learn from dynamic assertions
- Can learn invariants from the majority of code

Linux/kernel/posix-cpu-timers.c:

```c
void /*@IRQ(D, X)*/ run_posix_cpu_timers(...) {
  BUG_ON(!irqs_disabled()); ... }
```

Seed function
Annotation Extraction From Code Assertions

```c
linux/kernel/posix-cpu-timers.c:
void /*@IRQ(D, X)*/ run_posix_cpu_timers(...) {
  BUG_ON(!irqs_disabled()); ... }
```

- Learn from dynamic assertions
- Can learn invariants from the majority of code
  
  [ErnstICSE'00], [EnglerSOSP'01], [HangallCSE'02], [LiFSE'05], [LivshitsFSE'05], [TanSecurity'08] ...

- We directly extract annotations from seed functions’ code and comments.
- Challenge: Scarceness of seed functions
void update_process_times(int user_tick) {
    struct task_struct p = get_current();
    ...

    account_process_tick(p, user_tick);
    run_local_timers();
    if (rcu_pending(cpu))
        rcu_check_callbacks(cpu, user_tick);
    scheduler_tick();
    run_posix_cpu_timers(p);
}
Annotation Propagation

```
int kernel/timer.c:
1 void update_process_times(int user_tick)
2 {
3     struct task_struct p = get_current();
4     ...
5
6     account_process_tick(p, user_tick);
7     run_local_timers();
8     if (rcu_pending(cpu))
9         rcu_check_callbacks(cpu, user_tick);
10    scheduler_tick();
11    run_posix_cpu_timers(p);
12 }
```
void update_process_times(int user_tick) {
    struct task_struct p = get_current();
    ...
    account_process_tick(p, user_tick);
    run_local_timers();
    if (rcu_pending(cpu))
        rcu_check_callbacks(cpu, user_tick);
    scheduler_tick();
    run_posix_cpu_timers(p);
}
void update_process_times(int user_tick) {
    struct task_struct p = get_current();
    ...
    account_process_tick(p, user_tick);
    run_local_timers();
    if (rcu_pending(cpu))
        rcu_check_callbacks(cpu, user_tick);
    scheduler_tick();
    run_posix_cpu_timers(p);
}
void update_process_times(int user_tick)
{
    struct task_struct p = get_current();
    ...
    account_process_tick(p, user_tick);
    run_local_timers();
    if (rcu_pending(cpu))
        rcu_check_callbacks(cpu, user_tick);
    scheduler_tick();
    run_posix_cpu_timers(p);
}
void update_process_times(int user_tick) {
    struct task_struct p = get_current();
    ...
    account_process_tick(p, user_tick);
    run_local_timers();
    if (rcu_pending(cpu))
        rcu_check_callbacks(cpu, user_tick);
    scheduler_tick();
    run_posix_cpu_timers(p);
}
void update_process_times(int user_tick) {
    struct task_struct p = get_current();
    ...
    account_process_tick(p, user_tick);
    run_local_timers();
    if (rcu_pending(cpu))
        rcu_check_callbacks(cpu, user_tick);
    scheduler_tick();
    run_posix_cpu_timers(p);
}
void update_process_times(int user_tick) {
    struct task_struct p = get_current();
    ...
    account_process_tick(p, user_tick);
    run_local_timers();
    if (rcu_pending(cpu))
        rcu_check_callbacks(cpu, user_tick);
    scheduler_tick();
    run_posix_cpu_timers(p);
}
Annotation Propagation

linux/kernel/timer.c:
1  void update_process_times(int user_tick)
2  {
3         struct task_struct p = get_current();
4         ...
5
6         account_process_tick(p, user_tick);
7         run_local_timers();
8         if (rcu_pending(cpu))
9             rcu_check_callbacks(cpu, user_tick);
10        scheduler_tick();
11        run_posix_cpu_timers(p);
12 }

• Initialize
  • only 8 IRQ functions (e.g., local_irq_disable) with (X, E), (X, D), etc.
  • seed functions with annotations extracted from comments and code
Bug Detection - Unsatisfiable Annotations

Seed function

```c
linux//arch/x86/mm/pageattr.c:
static void /* @IRQ (E, E) */ cpa_flush_array(...)
{   ... BUG_ON(irqs_disabled());   ... }
```
Bug Detection - Unsatisfiable Annotations

drivers/ssb/pcmcia.c:
static void ssb_pcmcia_write16(...) {
    ...
    \texttt{spin\_lock\_irqsave}(...);
    err = \texttt{select\_core\_and\_segment}(...);
    ...
}

linux/arch/x86/mm/pageattr.c:
static void /* @IRQ (E, E) */ cpa_flush_array(...) {
    ... \texttt{BUG\_ON(irqs\_disabled())}; ...
}
Bug Detection - Unsatisfiable Annotations

```
drivers/ssb/pcmcia.c:
static void ssb_pcmcia_write16(...) {
    ...
    spin_lock_irqsave(...);
    err = select_core_and_segment(...);
    ...
}
```

```
linux/arch/x86/mm/pageattr.c:
static void /* @IRQ (E, E) */ cpa_flush_array(...) {
    ... BUG_ON(irqs_disabled()); ... 
}
```

Seed function
Bug Detection - Unsatisfiable Annotations

Drivers/SSB/PCMCIA.C:
static void ssb_pcmcia_write16(...) {
  ...
  spin_lock_irqsave(...);
  err = select_core_and_segment(...);
  ...
}

LINUX/ARCH/X86/MM/PAGEATTR.C:
static void /*@IRQ (E, E)*/ cpa_flush_array(...) {
  ... BUG_ON(irqs_disabled()); ... }

Violation! A real bug in the Linux kernel

Seed function

Call*
Bug Detection - Root Function Annotations

```c
linux/kernel/posix-cpu-timers.c:
void /*@IRQ (D, D)/ run_posix_cpu_timers(...) {
  ... BUG_ON(!irqs_disabled()); ...
}
```

Seed function
Bug Detection - Root Function Annotations

```c
linux/arch/alpha/kernel/irq_alpha.c
asmlinkage /* @IRQ (D, D) */ void do_entInt(...) {
    ...
    smp_percpu_timer_interrupt(...); ...
}
```

```c
linux/kernel/posix-cpu-timers.c:
void /* @IRQ (D, D) */ run_posix_cpu_timers(...) {
    ... BUG_ON(!irqs_disabled()); ...
}
```

Seed function
Bug Detection - Root Function Annotations

- Root function do_entInt has no callers within a module.
- No guaranteed that external callers will disable interrupts.
Bug Detection - Root Function Annotations

- Root function `do_entInt` has no callers within a module.
- No guaranteed that external callers will disable interrupts.

Violation!
Forgot to call `local_irq_disable()`;
A real bug in the Linux kernel

Seed function
Outline

• Motivation & Contributions
• Annotation Design
• Annotation Extraction
  • From comments
  • From code
• Annotation Propagation & Bug Detection
• Results: Bug Detection & Annotation Extraction
• Related Work
• Conclusions
Overall Results On Linux

<table>
<thead>
<tr>
<th>Source</th>
<th>Seed Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>226</td>
</tr>
<tr>
<td>Assertion</td>
<td>24</td>
</tr>
<tr>
<td>Total</td>
<td>245</td>
</tr>
</tbody>
</table>

- Annotations can help detect and avoid bugs.
- Comments and code complement each other for annotation extraction and bug detection.
- We propagate seed annotations to generate 96,821 annotations.
Overall Results On Linux

<table>
<thead>
<tr>
<th>Source</th>
<th>Seed Annotation</th>
<th>Seed Checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>226</td>
<td>119</td>
</tr>
<tr>
<td>Assertion</td>
<td>24</td>
<td>17</td>
</tr>
<tr>
<td>Total</td>
<td>245</td>
<td>133</td>
</tr>
</tbody>
</table>

- Annotations can help detect and avoid bugs.
- Comments and code complement each other for annotation extraction and bug detection.
- We propagate seed annotations to generate 96,821 annotations.
Overall Results On Linux

<table>
<thead>
<tr>
<th>Source</th>
<th>Seed Annotation</th>
<th>Seed Checked</th>
<th>True Bugs</th>
<th>False Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>226</td>
<td>119</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Assertion</td>
<td>24</td>
<td>17</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>245</td>
<td>133</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

- Annotations can help **detect** and **avoid** bugs.
- Comments and code complement each other for annotation extraction and bug detection.
- We propagate seed annotations to generate 96,821 annotations.
## Annotation Extraction Results

<table>
<thead>
<tr>
<th>Software</th>
<th>LOC</th>
<th>Sentence</th>
<th>IRQSent</th>
<th>HeuSent</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>5.2M</td>
<td>1,024,624</td>
<td>23,662</td>
<td>423</td>
<td>226</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>2.4M</td>
<td>420,013</td>
<td>11,117</td>
<td>80</td>
<td>43</td>
</tr>
<tr>
<td>NetBSD</td>
<td>3.3M</td>
<td>680,650</td>
<td>23,942</td>
<td>108</td>
<td>62</td>
</tr>
<tr>
<td>OpenSolaris</td>
<td>3.7M</td>
<td>535,073</td>
<td>8,074</td>
<td>71</td>
<td>24</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>14.6M</strong></td>
<td><strong>2,660,360</strong></td>
<td><strong>66,795</strong></td>
<td><strong>682</strong></td>
<td><strong>355</strong></td>
</tr>
</tbody>
</table>

- Reduce the # of annotations to be manually read from 66,795 to 682.
- The annotation generation accuracy is 90-100%.
Limitations & Future Work

- Automatically learn paraphrases, e.g., “disable” = “block”
  - Promising preliminary results [LinNLE’01, GlickmanRANLP’03, HillMSR’08]
- Consider different types of interrupts, different interrupt context, and conditional annotations
- Send annotations to developers for confirmation
- To detect annotations extracted from outdated comments
Conclusions

✧ Feasible to extract annotations from comments & code

• Generated 96,821 interrupt-related annotations & automatically detected 9 bugs in the Linux kernel

• These annotations can help developers avoid bugs.

✧ Combining comments & code help extract more annotations and detect more bugs than using comments or code alone.

• Apply to non-OS code and for extracting other types of annotations