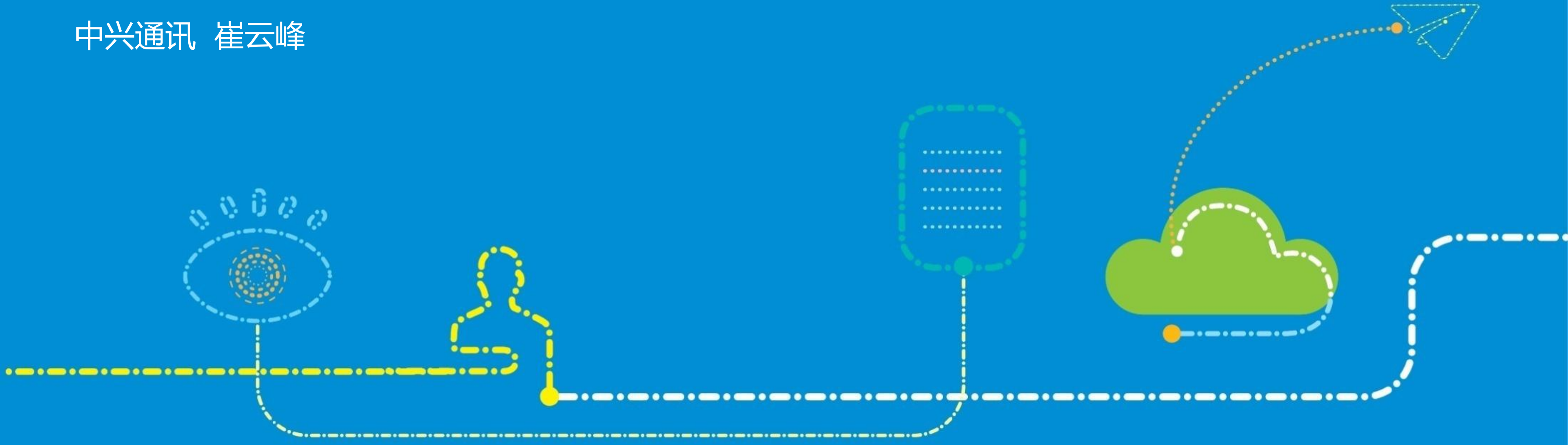
Linux 内核形式化验证实践

中兴通讯 崔云峰



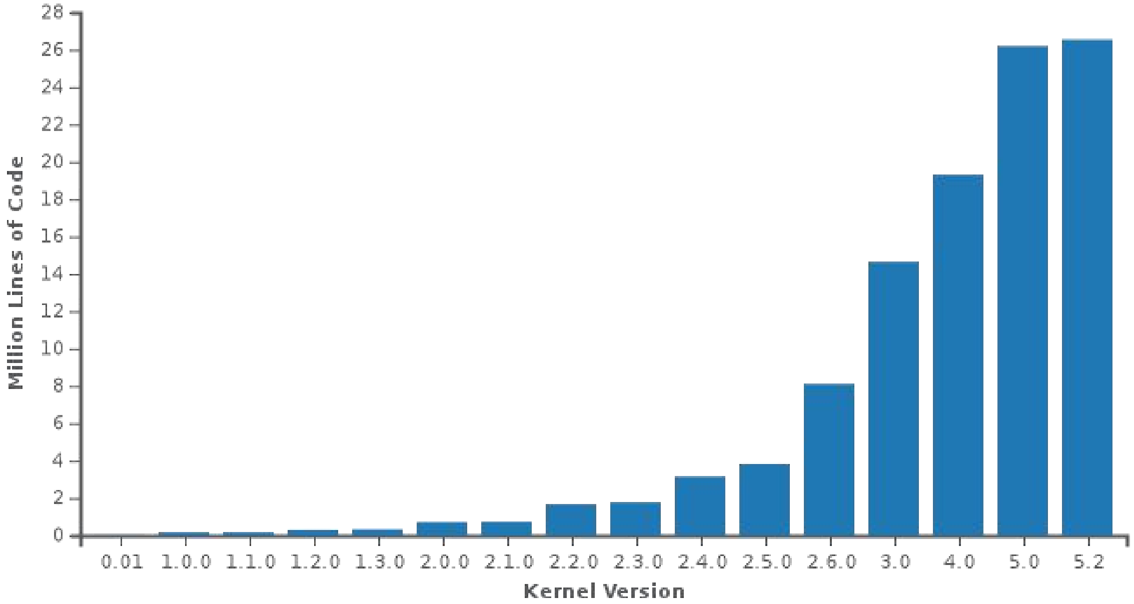
目录

- 形式化（模型检查）方法
  - 简介
  - 工具
  - 语言
- Linux Futex实践
  - 建模实践
  - 故障检测
- 小结

形式化方法的意义

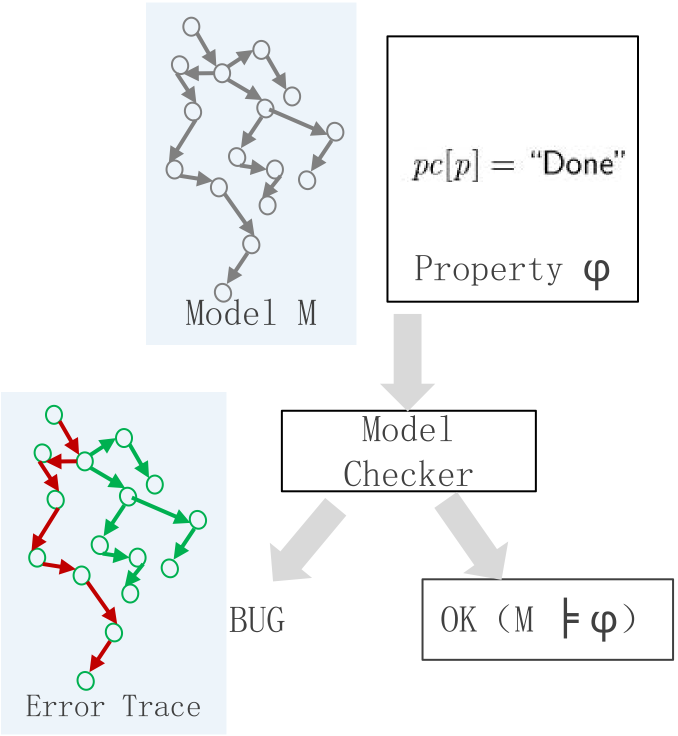
内核越来越复杂，应用面越来越广
- 规模大：超过2600万行；
- 高复杂：多核、RCU、实时补丁等；
- 应用广：服务器、汽车、电信设备、手机等。

内核失效的影响面越来越大，后果也更严重；传统方法越来越难以保障质量。



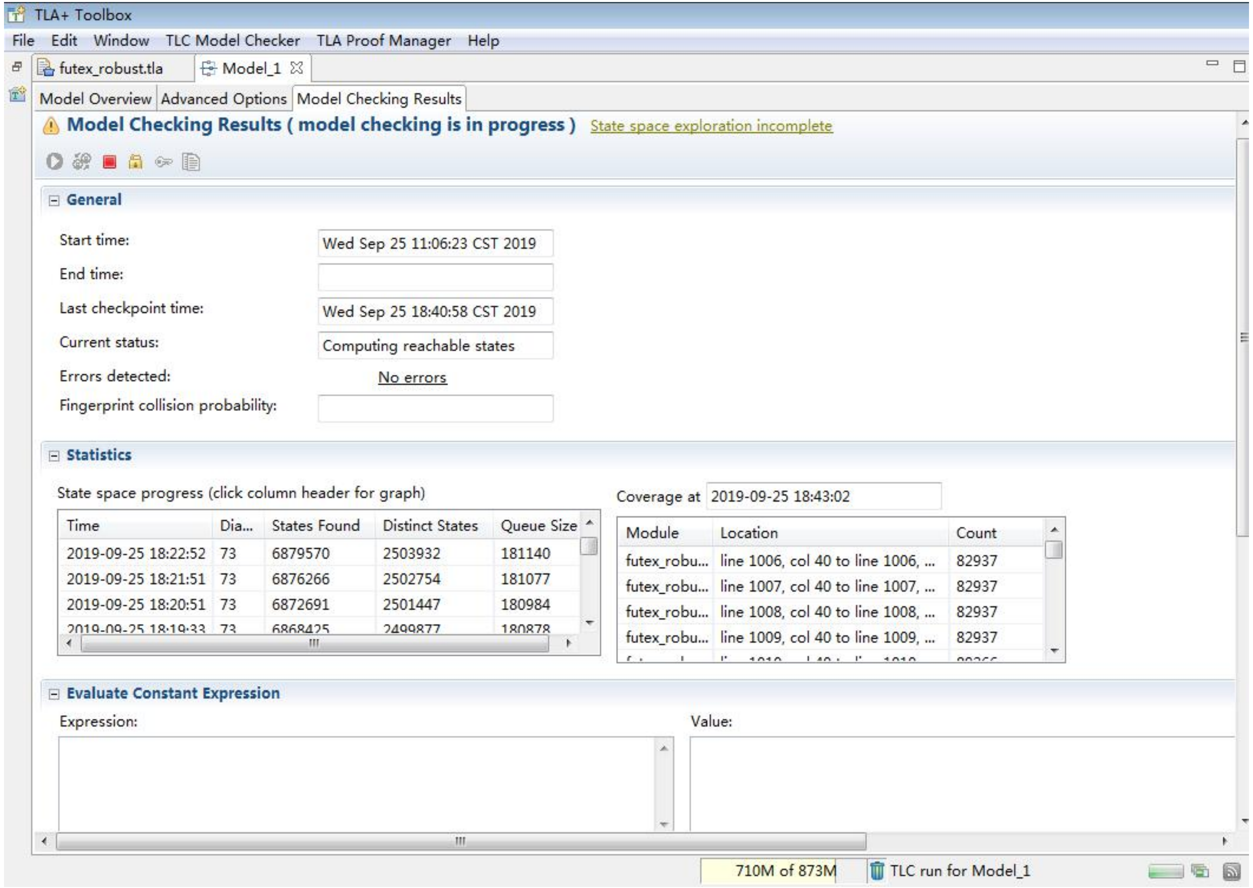
形式化方法概述

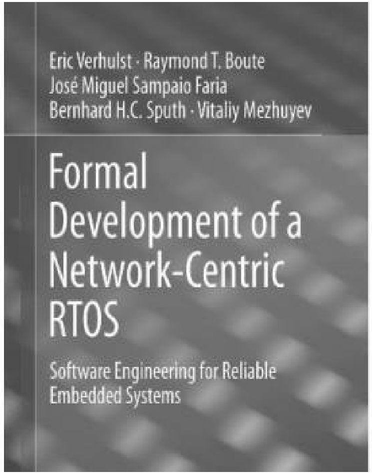
形式化方法：
定理证明（formal proof）：高阶逻辑、复杂、成本高
模型检查（model checking）：一阶逻辑、易于落地



模型检查过程：
1. 根据设计或实现编写模型（M）；
2. 根据设计需求编写属性约束（φ）；
3. 检查器通过穷举状态空间：
   - 若满足则证明：M ⊨ φ
   - 否则给出错误路径

模型检查工具及应用案例





单核RTOS ->多核、分布式 RTOS：
TLA+设计新方案尺寸小10倍

模型语言：命题逻辑

命题逻辑（Propositional logic）：

$\wedge$ conjunction(并且)　　$\vee$ disjunction(或)　　$\rightarrow$ implication(蕴含，如果..那么)

$\equiv$ equivalence(等价于)　　$\neg$ negation(非)

并且

p	q	$(p \wedge q)$
T	T	T
T	F	F
F	T	F
F	F	F

或

p	q	$(p \vee q)$
T	T	T
T	F	T
F	T	T
F	F	F

非

p	$\neg p$
T	F
F	T

等价于

p	q	$(p \equiv q)$
T	T	T
T	F	F
F	T	F
F	F	T

蕴含

p	q	$(p \rightarrow q)$
T	T	T
T	F	F
F	T	T
F	F	T

模型语言：谓词逻辑

谓词逻辑（Predicate Logic）：

- 全称谓词“∀x”（Any）：指集合中个体的全部，等价于：每一个、所有、凡。
  范例：假设 S == {1， 2， 3}　则 $\forall x \in S : x\%2 = 0$ 为成假命题

- 存在谓词“∃x”（Exists）：指集合中个体至少有一个存在，等价于：存在、有些、有。
  范例：假设 S == {1， 2， 3}　则 $\exists x \in S : x\%2 = 0$ 为成真命题

模型语言：时态逻辑

时态逻辑（Temporal Logic）：

- □ (Always，总是)：

  □ P is ture of a behavior if P is true in every state of the behavior.

  如果对于系统中所有行为P一直都为真，则“□ P”针对这个系统行为是真命题。

- ◇ (Eventually，最终)：

  ◇ F asserts that F is not always flase, which means that F is true at some time eventually.

  “◇ F”是指F不是总是为假，而是在某个时刻之后最终为真。

- ⇝ (Leads to，导致):

  F ⇝ G asserts that whenever F is ture, G is eventually true (G is ture then or at some later time).

  “F ⇝ G”是指从F为真的时刻开始，G最终一定为真（G不一定立即为真，可能晚一点为真）

状态机模型

将程序看作动态离散系统，提取变量并用TLA+语言描述变量的状态变迁。

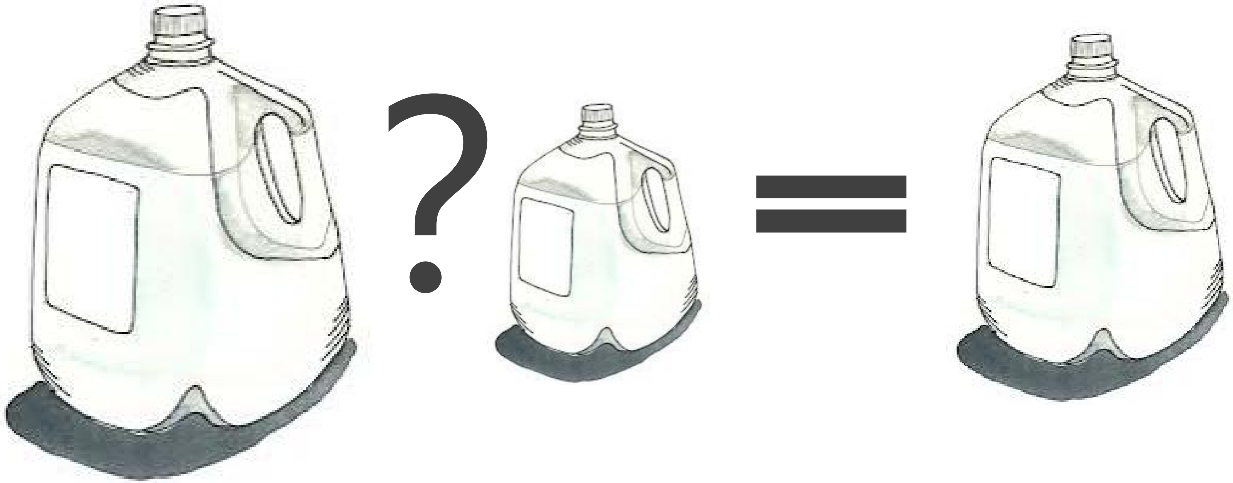
任何一个程序都可采用TLA+时态逻辑表达：

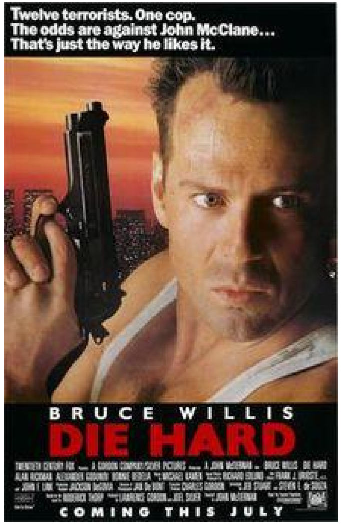
$$Init \bigwedge \Box[Next]_{var} \bigwedge \Box\Diamond \langle Liveness \rangle$$

- NEXT $\equiv$ BEHAIOR1 $\vee$ BEHAIOR2 $\vee$ ...
- BEHAVIOR $\equiv$ ENABLED $\wedge$ ACTION
- Liveness Property : Define a correct behavior that must eventually hold
  定义过程中最终的正确结果

案例：Die Hard Problem

有一个5加仑的罐子和3加仑的罐子，如何准确测量出4加仑的水？





虎胆龙威

案例：Die Hard Problem

- 状态变量：

```
VARIABLES big, small
```

- 初始状态：

```
Init == /\ big = 0
        /\ small = 0
```

- Next状态：

```
Next ==  \/ FillSmallJug
         \/ FillBigJug
         \/ EmptySmallJug
         \/ EmptyBigJug
         \/ SmallToBig
         \/ BigToSmall
```

- 模型Spec：

```
Spec == Init /\ [][Next]_<<big, small>>
```

```
FillSmallJug  == /\ small' = 3
                 /\ big' = big

EmptyBigJug   == /\ big' = 0
                 /\ small' = small

Min(m,n) == IF m < n THEN m ELSE n

SmallToBig == /\ big'   = Min(big + small, 5)
              /\ small' = small - (big' - big)

BigToSmall == /\ small' = Min(big + small, 3)
              /\ big'   = big - (small' - small)
```

案例：Die Hard Problem

- 设定目标为不变式：
  φ： big # 4
- 穷举状态空间验证(M ⊧ φ)是否成立：
  9个步骤得出违例情况（4加仑水）

Invariants

Formulas true in every reachable state.

☑ big # 4

Add

Edit

Remove

Model_1

`Invariant big # 4 is violated.`

⊞ **Error-Trace Exploration**

Error-Trace

Name	Value
<Initial predicate>	State (num = 1)
big	0
small	0
<Action line 65, col	State (num = 2)
big	0
small	3
<Action line 94, col	State (num = 3)
big	3
small	0
<Action line 65, col	State (num = 4)
big	3
small	3
<Action line 94, col	State (num = 5)
big	5
small	1
<Action line 74, col 18 to line	State (num = 6)
big	0
small	1
<Action line 94, col 15 to line	State (num = 7)
big	1
small	0
<Action line 65, col 18 to line	State (num = 8)
big	1
small	3
<Action line 94, col 15 to line	State (num = 9)
big	4
small	0

```
big = 4
small = 0
```

模型M 不满足于 φ，给出错误路径

PlusCal语言

- PlusCal 为TLA+的语法糖，提供类似C语言的形式。
- PlusCal与TLA+逻辑上对等，并可以自动转换。
- 每一个标号（Label）对应与一个行为，TLC穷举所有行为的状态空间。

```
---------------------------MODULE name_of_file-------------------
EXTENDS <M1>, <M2>, ...
CONSTANTS <C1>, <C2>, ...
(***********************************************************************
***
 --algorithm <algorithm>
        variable v1, v2, v3, ...;
        define{
                Op == <Exp>
         }
         macro Macro(<var1>, <var2>, ...)
         {
               <UnlabeledPlusCal>
         }
         process(Proc \in Set)
               variables v1, v2 ;
         {
          L1:
                <LabeledPlusCal 1>
          L2:
                <LabeledPlusCal 2>
          }
end algorithm;
***********************************************************************)
```

抽象结构：Function

- Function：描述了一种从定义域到值域的映射关系

  结构：Function == [s \in S |-> foo]

  范例：Doubles == [n \in Nat |-> 2 * n]

  Sum == [x, y \in S |-> x + y]

  调用：Doubles[2] = 4，Sum[1, 2] = 3

Futex模型

MODULE *futex_robust*

Linux futex robust model

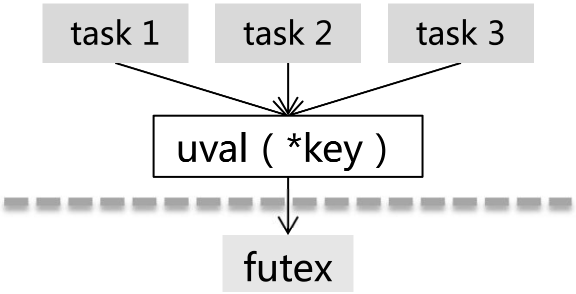
EXTENDS *Naturals*, *Sequences*, *FiniteSets*, *TLC*

CONSTANTS *HASHSIZE*,
UNKNOWN,
TASK, $\{t1, t2\}$
FUTEX_SET, $\{futex1, futex2\}$
SEM_ARRAY, $\{mutex1, mutex2\}$
MAX_TEST_CYCLE

ASSUME

$$
\begin{aligned}
&\land HASHSIZE \geq Cardinality(SEM\_ARRAY) \\
&\land Cardinality(SEM\_ARRAY) = Cardinality(FUTEX\_SET) \\
&\land Cardinality(FUTEX\_SET) = Cardinality(TASK)
\end{aligned}
$$

Futex模型: 状态变量uval、key



Function

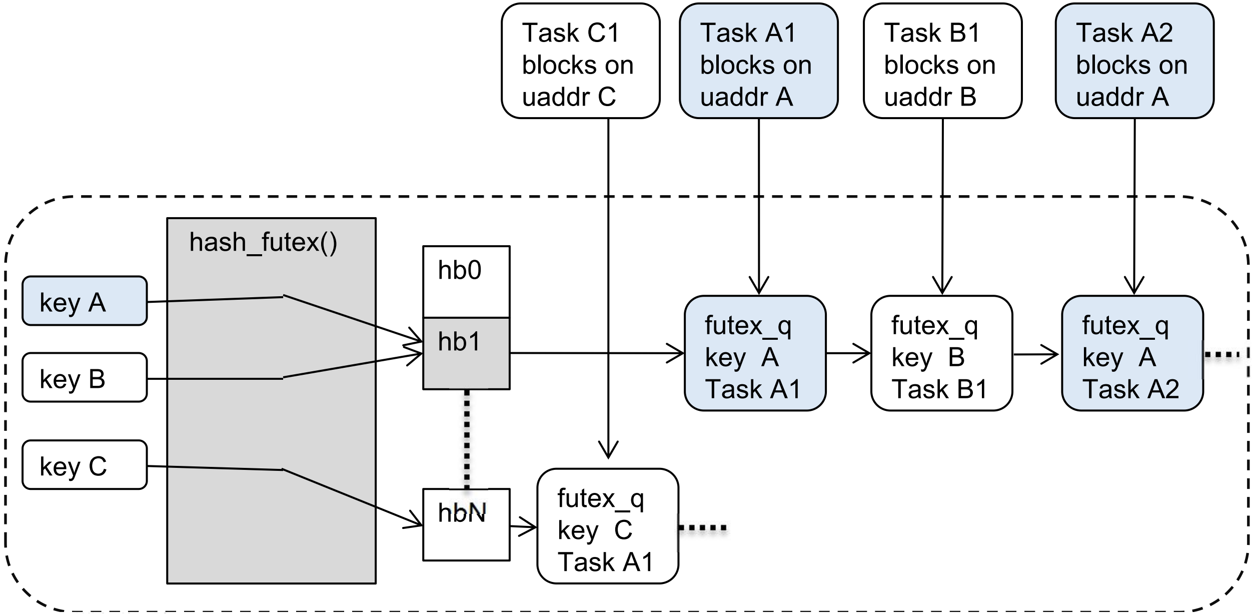
- [定义域 |-> 值域]
  - mutex == [key |-> uval]
  - key为数组SEM_ARRARY的下标
  - uval包括三个字段：tid、died、waited

$$mutex \quad = [n \in 1 \, .. \, Cardinality(SEM\_ARRAY) \mapsto [tid \mapsto \text{"none"}, \, died \mapsto \text{FALSE}, \, waited \mapsto \text{FALSE}]$$

$$make\_user\_val(t, \, d, \, w) \triangleq [tid \mapsto t, \, died \mapsto d, \, waited \mapsto w]$$

$$empty\_uval \triangleq make\_user\_val(\text{"none"}, \, \text{FALSE}, \, \text{FALSE})$$

Futex模型: hash_futex, futex_q




```
struct futex_hash_bucket {
    atomic_t waiters;
    spinlock_t lock;
    struct plist_head chain;
} ____cacheline_aligned_in_smp;

struct futex_q {
    struct plist_node list;

    struct task_struct *task;
    spinlock_t *lock_ptr;
    union futex_key key;
    struct futex_pi_state *pi_state;
    struct rt_mutex_waiter *rt_waiter;
    union futex_key *requeue_pi_key;
    u32 bitset;
} __randomize_layout;
```

variables

$$
\begin{aligned}
&futex\_hash &&= [k \in 0 \,..\, HASHSIZE \mapsto [waiters \mapsto 0,\ locked \mapsto \text{FALSE},\ chain \mapsto \{\}]];\\
&futex\_q &&= [q \in FUTEX\_SET \mapsto [task \mapsto \text{“none”},\ key \mapsto UNKNOWN]];
\end{aligned}
$$

Futex模型: spin_lock，queue_lock

```
static inline struct futex_hash_bucket *queue_lock(
	__acquires(&hb->lock)
{
	struct futex_hash_bucket *hb;

	hb = hash_futex(&q->key);

	hb_waiters_inc(hb);

	q->lock_ptr = &hb->lock;

	spin_lock(&hb->lock); /* implies MB (A) */
	return hb;
}
```

queue_lock模型：

- queue_inc_waiter行为：查找hash桶并递增等待者计数；
- queue_lock行为：自旋等待hash冲突链锁。

procedure $queue\_lock(\ key\ )$
{
$queue\_inc\_waiters:$
$\quad futex\_hash[key].waiters := futex\_hash[key].waiters + 1\,;$
$queue\_lock:$
$\quad spin\_lock(futex\_hash[key].locked)\,;$
$\quad$ **return** ;
}

macro $spin\_lock(\ lock\ )$ {
$\quad$ **await** $\neg lock\,;$
$\quad lock := \mathrm{TRUE}\,;$
}

macro $spin\_unlock(\ lock\ )$ {
$\quad lock := \mathrm{FALSE}\,;$
}

Futex模型: spin_lock，queue_lock

spinlock模型原理：

- 利用ENABLE条件表达自旋语义；
- BEHAVIOR ≡ ENABLED ∧ ACTION ；
- ENABLED为真才执行ACTION，否则阻塞等待。

$$
\begin{array}{l}
\textbf{macro}\ \mathit{spin\_lock}(\ \mathit{lock}\ )\ \{ \\
\quad \textbf{await}\ \neg \mathit{lock}\,; \\
\quad \mathit{lock} := \mathrm{TRUE}\,; \\
\}
\end{array}
$$

$$
\begin{array}{l}
\textbf{procedure}\ \mathit{queue\_lock}(\ \mathit{key}\ ) \\
\{ \\
\mathit{queue\_inc\_waiters}: \\
\quad \mathit{futex\_hash}[\mathit{key}].\mathit{waiters} := \mathit{futex\_hash}[\mathit{key}].\mathit{waiters} + 1\,; \\
\mathit{queue\_lock}: \\
\quad \mathit{spin\_lock}(\mathit{futex\_hash}[\mathit{key}].\mathit{locked})\,; \\
\quad \textbf{return}\,; \\
\}
\end{array}
$$

$$
\begin{array}{lll}
\mathit{queue\_lock\_}(\mathit{self}) \triangleq & \wedge\ \mathit{pc}[\mathit{self}] = \text{“queue\_lock\_”} & \rbrace\ \text{ENABLED} \\
 & \wedge\ \neg(\mathit{futex\_hash}[\mathit{key\_}[\mathit{self}]].\mathit{locked}) & \\
\text{ACTION}\ \lbrace & \wedge\ \mathit{futex\_hash}' = [\mathit{futex\_hash}\ \mathrm{EXCEPT}\ ![\mathit{key\_}[\mathit{self}]].\mathit{locked} = \mathrm{TRUE}] & \\
 & \wedge\ \mathit{pc}' = [\mathit{pc}\ \mathrm{EXCEPT}\ ![\mathit{self}] = \mathit{Head}(\mathit{stack}[\mathit{self}]).\mathit{pc}] &
\end{array}
$$

Futex模型: CAS

Compare And Swap模型：

- CAS语义：采用原子方式进行比较和交换；
- 实现方式：使用宏(无标号）实现，表示过程是原子的，不会并发抢占。

$$
\begin{array}{l}
\textbf{macro}\ \mathit{cmpxchg\_futex\_value\_locked}(\ \mathit{key},\ \mathit{uval},\ \mathit{newval}\ ) \\
\{ \\
\quad \textbf{if}\ (\ \mathit{mutex}[\mathit{key}] = \mathit{uval}\ )\ \{ \\
\qquad \mathit{mutex}[\mathit{key}] := \mathit{newval}\,; \\
\qquad \mathit{ret}[\mathit{self}] := \text{“OK”}\ ; \\
\quad \}\ \textbf{else}\ \{ \\
\qquad \mathit{ret}[\mathit{self}] := \text{“EAGAIN”}\ ; \\
\quad \}\ ; \\
\}
\end{array}
$$

$$
\begin{array}{l}
\mathit{mutex\_take\_on\_empty}: \\
\qquad \mathit{cmpxchg\_futex\_value\_locked}(\mathit{id},\ \mathit{empty\_uval},\ \mathit{make\_user\_val}(\mathit{self},\ \text{FALSE},\ \text{FALSE}))\,; \\
\qquad \textbf{if}\ (\ \mathit{ret}[\mathit{self}] = \text{“OK”}\ )\ \{ \\
\qquad\qquad \text{got lock} \\
\qquad\qquad \textbf{goto}\ \mathit{mutex\_take\_out}\,; \\
\qquad \}\ ;
\end{array}
$$

self：代表当前线程的内部变量

ret：记录每个线程的返回值

Futex模型: CAS

$mutex\_take\_on\_empty:$

$\quad cmpxchg\_futex\_value\_locked(id,\ empty\_uval,\ make\_user\_val(self,\ \text{FALSE},\ \text{FALSE}))\ ;$

$\quad \textbf{if}\ (\ ret[self] = \text{"OK"}\ )\ \{$

$\qquad \text{got lock}$

$\qquad \textbf{goto}\ mutex\_take\_out\ ;$

$\quad \}\ ;$

Compare And Swap原子性：

- mutex_take_on_empty行为：在一个行为中实现CAS语义。不被抢占，具备原子性。

$$
\begin{aligned}
mutex\_take\_on\_empty(self) \triangleq\ & \land\ pc[self] = \text{"mutex\_take\_on\_empty"} \\
& \land\ \text{IF}\ \ mutex[id\_m[self]] = empty\_uval \\
& \qquad \text{THEN}\ \ \land\ mutex' = [mutex\ \text{EXCEPT}\ ![id\_m[self]] = make\_user\_val(self, \\
& \qquad\qquad\quad \land\ ret' = [ret\ \text{EXCEPT}\ ![self] = \text{"OK"}] \\
& \qquad \text{ELSE}\ \ \land\ ret' = [ret\ \text{EXCEPT}\ ![self] = \text{"EAGAIN"}] \\
& \qquad\qquad\quad \land\ mutex' = mutex \\
& \land\ \text{IF}\ \ ret'[self] = \text{"OK"} \\
& \qquad \text{THEN}\ \ \land\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{"mutex\_take\_out"}] \\
& \qquad \text{ELSE}\ \ \land\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{"mutex\_take\_check\_died"}]
\end{aligned}
$$

Futex模型: futex_wait、futex_wake

futex_wait/futex_wake模型：

- 函数由若干个行为组合：

  Next ≡ BEHAIOR1 ∨ BEHAIOR2 ∨ ...

- ENABLE条件、PC变量控制执行顺序：

  BEHAVIOR ≡ ENABLED ∧ ACTION

futex_behavior = /\ pc[self] = “futex_behavior”

/\ ACTION

/\ pc ' [self] = “xxxx”

$$
\begin{aligned}
futex\_wait(self) \triangleq\ & wait\_queue\_lock(self) \lor wait\_get\_uval(self) \\
& \lor wait\_check\_uval(self) \\
& \lor wait\_queue\_unlock\_on\_uval\_changed(\\
& \lor wait\_again\_on\_uval\_changed(self) \\
& \lor wait\_get\_init\_futex\_q(self) \\
& \lor wait\_futex\_enqueue(self) \lor wait\_unlo \\
& \lor wait\_wake(self) \lor wait\_waked(self) \\
& \lor wait\_put\_release\_futex\_q(self) \\
& \lor wait\_queue\_unlock\_ret(self) \\
& \lor wait\_return\_ok(self)
\end{aligned}
$$

$$
\begin{aligned}
futex\_wake(self) \triangleq\ & wake\_check\_waiters(self) \lor wake\_hash\_lock \\
& \lor wake\_check\_futex\_queue(self) \\
& \lor wake\_get\_all\_waiters(self) \\
& \lor wake\_hash\_unlock(self) \lor wake\_all\_wa \\
& \lor wake\_get\_waiter(self) \lor wake\_waiter(\\
& \lor wake\_out(self)
\end{aligned}
$$

Futex模型: 测试框架

```
fair process ( mutexProc ∈ TASK )
variable id, cycle ; {
mutex_test:
    cycle := MAX_TEST_CYCLE ;
mutex_begin:
    while ( cycle > 0 ) {
        id := CHOOSE n ∈ 1 .. Cardinality(SEM_ARRAY) : TRUE ;
mutex_take:
        mutex_required[self] := id ;
        call mutex_take(id) ;
cs:
        skip ;
mutex_give:
        call mutex_give(id) ;
dec_cycle:
        cycle := cycle − 1 ;
    }
}
```

测试框架模型：

- 创建多个测试线程，并发互斥量PV操作：
- cycle：循环测试次数
- id：代表互斥量地址（key）

Futex模型: Spec

- Spec ≡ Init ⋀ □[Next]$_{var}$ ⋀ □◇<Liveness>
    - Init ：状态变量的初始值
    - Next： 线程可执行所有的系统行为
- 状态空间穷举：穷举多个线程并发的状态空间

$$
\begin{aligned}
Init \triangleq\ & \text{Global variables} \\
& \land\ futex\_hash = [k \in 0\,..\,HASHSIZE \mapsto [waiters \mapsto 0,\ locked \mapsto \text{FAL} \\
& \land\ futex\_q = [q \in FUTEX\_SET \mapsto [task \mapsto \text{"none"},\ key \mapsto UNKNC \\
& \land\ mutex = [n \in 1\,..\,Cardinality(SEM\_ARRAY) \mapsto [task \mapsto \text{"none"} \\
& \qquad died \mapsto \text{FALSE}, \\
& \qquad waited \mapsto \text{FALS}
\end{aligned}
$$

$$
\begin{aligned}
mutexProc(self) \triangleq\ & mutex\_test(self) \lor mutex\_begin(self) \\
& \lor mutex\_take\_m(self) \lor cs(self) \\
& \lor mutex\_give\_(self) \lor dec\_cycle(self)
\end{aligned}
$$

$$
\begin{aligned}
Next \triangleq\ & (\exists\, self \in ProcSet : \lor queue\_lock(self) \lor queue\_unlock(self) \\
& \qquad \lor futex\_wait(self) \lor futex\_wake(self) \\
& \qquad \lor mutex\_take(self) \lor mutex\_give(self)) \\
& \lor (\exists\, self \in TASK : mutexProc(self)) \\
& \lor\ \text{Disjunct to prevent deadlock on termination} \\
& \quad ((\forall\, self \in ProcSet : pc[self] = \text{"Done"}) \land \text{UNCHANGED } vars)
\end{aligned}
$$

$$
Spec \triangleq\ \land Init \land \Box[Next]_{vars}
$$

Futex属性: Type Invariant

$$
\begin{aligned}
TypeInv \triangleq\ & \land\ mutex\_required \in [TASK \rightarrow 1 \,..\, Cardinality(SEM\_ARRAY) \cup \{UNKNOWN\}] \\
& \land\ futex\_q \in [FUTEX\_SET \rightarrow [task : TASK \cup \{\text{"none"}\}, \\
& \qquad\qquad\qquad\qquad\qquad key : 0 \,..\, HASHSIZE \cup \{UNKNOWN\}]] \\
& \land\ mutex \in [1 \,..\, Cardinality(SEM\_ARRAY) \rightarrow [task : TASK \cup \{\text{"none"}\}, \\
& \qquad\qquad\qquad\qquad\qquad died : \text{BOOLEAN}, \\
& \qquad\qquad\qquad\qquad\qquad waited : \text{BOOLEAN}]] \\
& \land\ futex\_hash \in [0 \,..\, HASHSIZE \rightarrow [waiters : 0 \,..\, Cardinality(TASK), \\
& \qquad\qquad\qquad\qquad\qquad locked : \text{BOOLEAN},
\end{aligned}
$$

模型运行过程中所有状态变量取值必须在预期的范围以内。

Futex属性: 互斥属性

$$
\begin{aligned}
MutualExclusion \triangleq\ & \forall\, p,\, q \in TASK : \\
& (\land\, p \neq q \\
& \quad \land\, mutex\_required[p] = mutex\_required[q] \\
& \quad \land\, mutex\_required[p] \neq UNKNOWN) \Rightarrow \neg \land\, pc[p] = \text{“cs”} \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \land\, pc[q] = \text{“cs”}
\end{aligned}
$$

任意两个不同的任务如持有相同的信号量，则两个任务不能同时处于临界区

标号“cs”为临界区

Futex属性: 无死锁属性

$$
\begin{aligned}
DeadlockFree \triangleq\ & \forall\, p \in TASK : \\
& (pc[p] = \text{“cs”}) \leadsto (\exists\, q \in TASK : pc[q] = \text{“cs”})
\end{aligned}
$$

当任意任务P进入临界区之后，一定存在另一个任务Q可以进入临界区

Futex属性: 无饿死属性、Liveness属性

$$StarvationFree \triangleq \forall p \in TASK : (pc[p] = \text{"wait\_wake"}) \leadsto (task[p].lock = \text{FALSE})$$

$$LivenessProperty \triangleq \Diamond\Box(\forall p \in TASK : pc[p] = \text{"Done"})$$

无饿死属性：当任意任务P进入等待状态之后，任务P一定会被唤醒

标号“wait_wake”阻塞等待futex

Liveness属性：最终所有的任务都将退出完成

Futex: BUG

故障现象

- 大量线程阻塞在互斥量
- 阻塞key值为0x80000000（FUTEX_WAITERS）
- 普通futex+robust属性

```
oldval = mutex->__data.__lock;
while (1){
  /* Try to acquire the lock through a CAS from 0 (not acquired) to
     our TID | assume_other_futex_waiters.  */
   if (__glibc_likely ((oldval == 0)
                && (atomic_compare_and_exchange_bool_acq
                  (&mutex->__data.__lock,
                   id | assume_other_futex_waiters, 0) == 0)))
      break;
    if ((oldval & FUTEX_OWNER_DIED) != 0){
      ...
      return EOWNERDEAD;
    }
    /* Check whether we already hold the mutex.  */
    if (__glibc_unlikely ((oldval & FUTEX_TID_MASK) == id)){
      ...
    }
    if ((oldval & FUTEX_WAITERS) == 0){
       if (atomic_compare_and_exchange_bool_acq (&mutex->__data.__lock,
                                 oldval | FUTEX_WAITERS, oldval) != 0){
         oldval = mutex->__data.__lock;
         continue;
       }
       oldval |= FUTEX_WAITERS;
    }
    assume_other_futex_waiters |= FUTEX_WAITERS;
    lll_futex_wait (&mutex->__data.__lock, oldval, PTHREAD_ROBUST_MUTEX
    oldval = mutex->__data.__lock;
```

Futex: BUG

Specify the values of declared constants.

defaultInitValue <- [model value]
SEM_ARRAY <- [model value] {sem1, sem2}
HASHSIZE <- 10
MAX_TEST_CYCLE <- 3
UNKNOWN <- 255
TASK <- [model value] {t1, t2}
FUTEX_SET <- [model value] {futex1, futex2}

Formulas true in every reachable state.

- [x] TypeInv
- [x] MutualExclusion

Add | Edit | Remove

Properties

Temporal formulas true for every possible behavior.

- [x] Termination
- [x] DeadlockFree
- [x] StarvationFree
- [x] LivenessProperty

Add | Edit | Remove

```
Deadlock reached.
```

Error-Trace Exploration

Error-Trace

Name	Value
<wait_unlock line 609, col 22 to line 616, col 63 of module fu	State (num = 56)
<wait_futex_enqueue line 599, col 29 to line 607, col 70 of m	State (num = 55)
<wait_get_init_futex_q line 586, col 32 to line 597, col 67 of m	State (num = 54)
<wait_check_uval line 539, col 26 to line 548, col 67 of modu	State (num = 53)
<wait_get_uval line 529, col 24 to line 537, col 52 of module	State (num = 52)
<queue_lock_ line 479, col 22 to line 489, col 45 of module f	State (num = 51)
<wait_unlock line 609, col 22 to line 616, col 63 of module fu	State (num = 50)
<queue_inc_waiters line 469, col 28 to line 477, col 69 of mo	State (num = 49)
<wait_queue_lock line 515, col 26 to line 527, col 54 of modu	State (num = 48)
<mutex_take_futex line 928, col 27 to line 945, col 78 of mod	State (num = 47)
<mutex_take_tag_waited line 897, col 32 to l	Click on a row to see in viewer below
<mutex_take_on_none_empty_uval line 883, col 40 to line 895	State (num = 45)
<mutex_take_check_died line 855, col 32 to line 865, col 67 c	State (num = 44)
<mutex_take_get_oldval line 823, col 32 to line 837, col 60 of	State (num = 43)
<mutex_take_retry line 812, col 27 to line 821, col 74 of mod	State (num = 42)
<mutex take line 1257, col 22 to line 1270, col 63 of modul	State (num = 41)

Futex: BUG

```
t1                              | t2                                         |key value
pthread_mutex_lock              |                                            |0
                                | pthread_mutex_lock                         |
 oldval = mutex->__data.__lock; |   oldval = mutex->__data.__lock;           |
 while(1){                      |   while(1){                                |
   if((oldval == 0) &&          |     if((oldval == 0) &&                    |
     cas(key, t1, 0) == 0)      |       cas(key, t2, 0) == 0)                |t1
     break;                     |                                            |
cs code                         |                                            |
pthred_mutex_unlock             |                                            |0
                                |       break;                               |
                                |     ... /*oldval = 0 */                    |
                                |     if(oldval & FUTEX_WAITERS == 0)        |
                                |       cas(key, oldval| FUTEX_WAITERS, oldval) |0|FUTEX_WAITERS
                                |     lll_futex_wait->futex_wait             |
                                |         futex_wait_setup                   |
                                |         futex_wait_queue_me                |
                                |         /*BLOCK HERE*/                     |
pthread_mutex_lock              |                                            |0|FUTEX_WAITERS
  while(1){                     |                                            |
    if (oldval == 0){}          |                                            |
    ...                         |                                            |
    lll_futex_wait->futex_wait  |                                            |0|FUTEX_WAITERS
        futex_wait_queue_me     |                                            |
        /*BLOCK HERE*/          |                                            |
                                |    oldval = mutex->__data.__lock;          |
                                |  }                                         |
```

小结

- 覆盖内核语义：
  模型检查语言能够表达Linux内核常用语义：list、hlist、struct、spinlock、mutex、CAS、函数调用等。
  命题逻辑、时态逻辑、谓词逻辑可准确表达设计约束：需求、不变式、Safety属性、Liveness属性等。

- 正确性证明：
  对于内核等复杂的并发系统，通过穷举状态空间可以发现隐晦故障或从理论上证明系统逻辑的正确性。

谢谢！

未 来，不 等 待