第 14 讲: Concurrency in OS Kernel

第二节：Scalable Concurrency – Lock

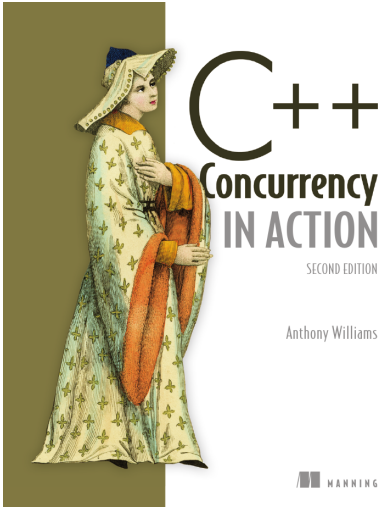
陈渝

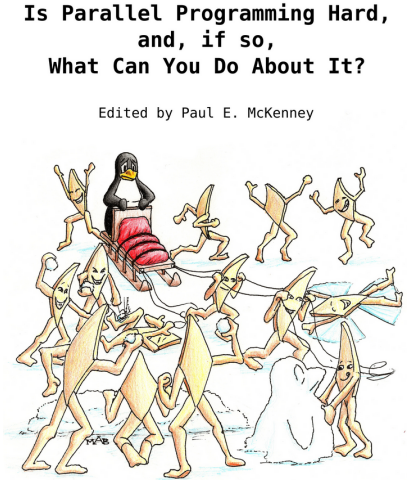
清华大学计算机系

yuchen@tsinghua.edu.cn

2020 年 5 月 18 日

Resource





Reference:
”Is Parallel Programming Hard, And, If So, What Can You Do About It?”,Paul McKenney;
”C++Concurrency in Action”, ANTHONY WILLIAMS;
”CS510 - Advanced Topics in Concurrency”, Jonathan Walpole; 很 Adam Belay from MIT PDOS

Scalable Locking

- Why do we need locking in the kernel?
- Which problems are we trying to solve?
- What implementation choices do we have?
- Is there a one-size-fits-all solution?

ticket spinlock

Goal

- Correctness: Mutual exclusion, Progress, Bounded wait
- Fairness
- Performance

ticket spinlock

Idea:

- reserve each thread's turn to use a lock
- each thread spins until their turn
- Use new atomic primitive: fetch-and-add (FAA)
- Spin while not thread's ticket != turn
- Release: Advance to next turn

ticket spinlock

```
typedef  struct {
    int ticket;
    int turn;
} lock_t;

void lock_init(lock_t *lock) {
    lock->ticket = 0;
    lock->turn = 0;
}
void acquire(lock_t *lock) {
    int myturn = FAA(&lock->ticket);
    while (lock->turn != myturn); // spin
}
void release(lock_t *lock) { lock->turn += 1; }
```

ticket spinlock

Ticket lock time analysis

- Atomic increment –O(1) broadcast message
- Then read-only spin, no cost until next release
- release OP invalidates message sent to all cores, and O(N) find messages, as they re-read
- But fairness and less bus traffic while spinning

Ticket are “non-scalable” locks, cost of handoff scales with number of waiters

MCS lock

- Goal: O(1) message release time
- Can we wake just one core at a time?
- Idea: Have each core spin on a different cache-line

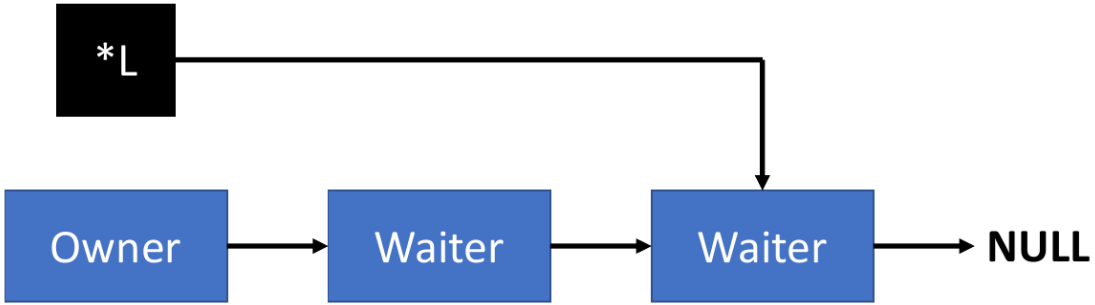
MCS lock

- Each CPU has a qnode structure in its local memory

```
typedef struct qnode {
    struct qnode *next;
    bool locked;
} qnode;
```

- A lock is a qnode pointer to the tail of the list
- While waiting, spin on local locked flag

MCS lock



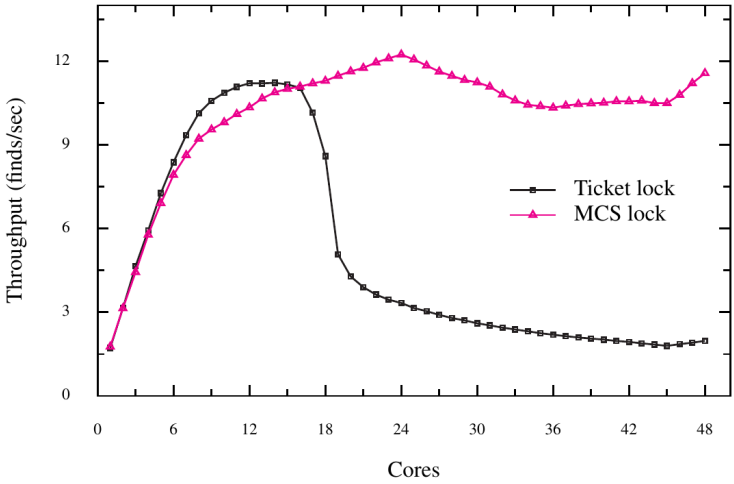
Acquiring MCS locks

```
acquire (qnode *L, qnode *I) {
    I->next = NULL;
    qnode *predecessor = I;
    XCHG (*L, predecessor);
    if (predecessor != NULL) {
        I->locked = true;
        predecessor->next = I;
        while (I->locked) ;
    }
}
```

Releasing MCS locks

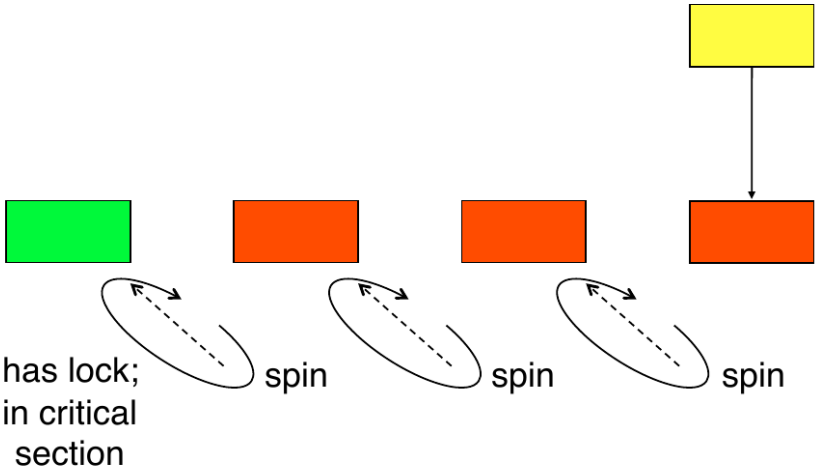
```
release (lock *L, qnode *I) {
    if (!I->next)
        if (CAS (*L, I, NULL))
            return;
    while (!I->next) ;
    I->next->locked = false;
}
```

MCS lock



But not a panacea

CLH lock



CLH lock

```
type qnode = record
    prev : ^qnode
    succ_must_wait : Boolean

type lock = ^qnode   // initialized to point to an unowned qnode

procedure acquire_lock (L : ^lock, I : ^qnode)
    I->succ_must_wait := true
    pred : ^qnode := I->prev := fetch_and_store(L, I)
    repeat while pred->succ_must_wait

procedure release_lock (ref I : ^qnode)
    pred : ^qnode := I->prev
    I->succ_must_wait := false
    I := pred    // take pred's qnode
```

lock comparison