Теория и практика многопоточного программирования

Семинар 6

Неганов Алексей

Московский физико-технический институт (национальный исследовательский университет)
Кафедра теоретической и прикладной информатики

Москва 2020

Array-based queue lock

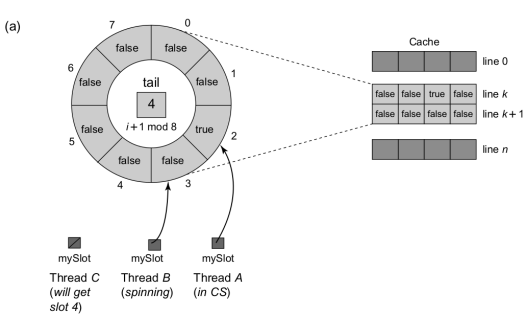
```
class ALock {
    pthread_key_t idx_key;
    atomic<uint64_t> tail;
    atomic<uint8_t> *flag;
    size_t size;

    int get_thread_idx() {
        void *mem = pthread_getspecific(idx_key);
        return mem ? *((int*)mem) : -1;
    }

    void set_thread_idx(int val) {
        void *mem = malloc(sizeof(int));
        *((int*)mem)=val;
        pthread_setspecific(idx_key, mem);
    }
```

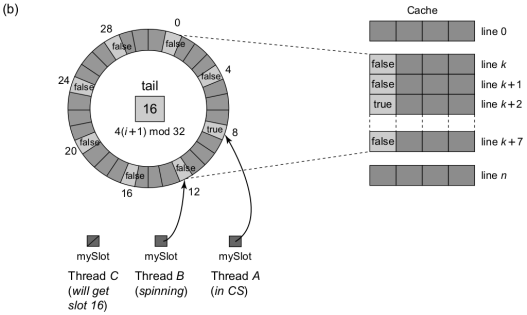
```
public:
    ALock(size_t cap) : tail(0), size(cap) {
        pthread_key_create(&idx_key, NULL);
        flag = new atomic<uint8_t>[cap];
        flag[0].store(1);
        for (size_t i = 1; i < cap; i++)
            flag[i].store(0);
    }

    void lock() {
        const auto idx = tail.fetch_add(1) % size;
        set_thread_idx(idx);
        while (!flag[idx].load());
    }

    void unlock() {
        const auto idx = get_thread_idx();
        flag[idx].store(0);
        flag[(idx + 1) % size].store(1);
    }
};
```

Замечание

Упорядочение памяти (`memory_order`), обработка ошибок, деструктор класса опущены для наглядности

Array-based queue lock





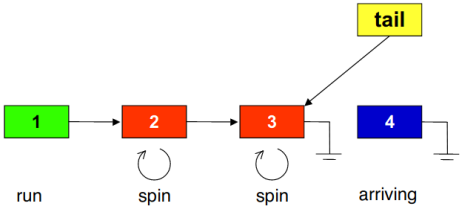
MCS lock

```
struct mcs_lock {
    std::atomic<struct mcs_node *> tail;

    struct mcs_node {
        struct mcs_node *next;
        bool locked;
    };

    thread_local static struct mcs_node qnode;
};

static inline mcs_lock::lock() {
    const auto predecessor = tail.exchange (&this.qnode);

    if (predecessor != nullptr) {
        qnode.locked = true;
        predecessor->next = &this.qnode;
        while (qnode.locked);
    }
}
```

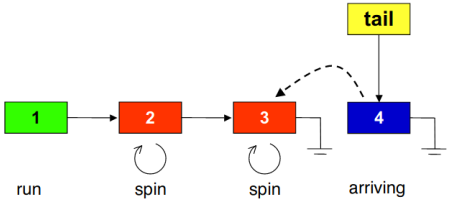
MCS lock

```
static inline mcs_lock::unlock() {
    const auto successor = qnode.next;

    if (successor == nullptr) {
        if (tail.compare_exchange_strong(&this.qnode, nullptr)) {
            // No CPUs were waiting for the lock, set it to nullptr
            return;
        }
    }

    // We could not set our successor to nullptr, therefore qnode.next is out of sync with tail,
    // therefore another CPU is in the middle of `lock()`, prior to linking themselves in the queue.
    // We wait for that to happen:
    while (successor == nullptr) ;

    // The other CPU has linked themselves, all we need to do is wake it up as the next-in-line
    successor->locked = false;
}
```

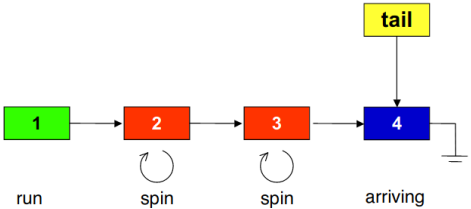
MCS lock



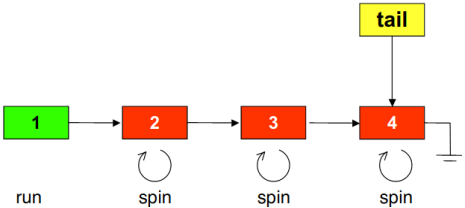
MCS lock



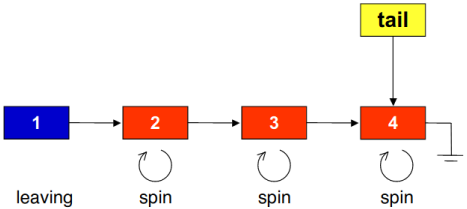
MCS lock



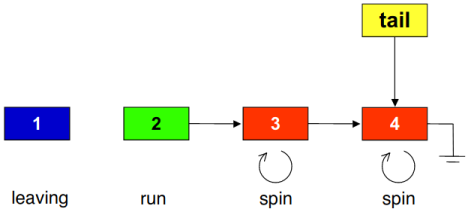
MCS lock



MCS lock



MCS lock



CLH lock

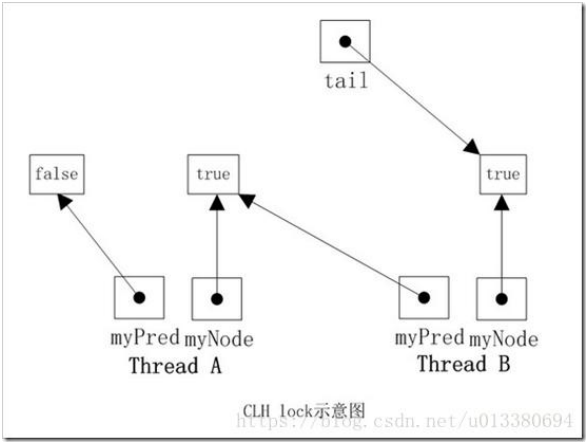
```c
struct clh_mutex_node {
    _Atomic char succ_must_wait;
};

typedef struct {
    clh_mutex_node_t * mynode;
    char padding[64];  // To avoid false sharing with the tail
    _Atomic (clh_mutex_node_t *) tail;
} clh_mutex_t;

static clh_mutex_node_t * clh_mutex_create_node(char islocked) {
    clh_mutex_node_t * new_node = (clh_mutex_node_t *)malloc(sizeof(clh_mutex_node_t));
    atomic_store_explicit(&new_node->succ_must_wait, islocked, memory_order_relaxed);
    return new_node;
}

void clh_mutex_init(clh_mutex_t * self) {
    // We create the first sentinel node unlocked, with islocked=0
    clh_mutex_node_t * node = clh_mutex_create_node(0);
    self->mynode = node;
    atomic_store(&self->tail, node);
}
```

https://github.com/pramalhe/ConcurrencyFreaks/blob/master/C11/locks

CLH lock

```c
// simplified version
void clh_mutex_lock(clh_mutex_t * self) {
    // Create the new node locked by default, setting islocked=1
    clh_mutex_node_t *mynode = clh_mutex_create_node(1);
    clh_mutex_node_t *prev = atomic_exchange(&self->tail, mynode);

    // This thread's node is now in the queue, so wait until it is its turn
    while (atomic_load(&prev->succ_must_wait));

    // This thread has acquired the lock on the mutex and it is now safe to
    // cleanup the memory of the previous node.
    free(prev);

    // Store mynode for clh_mutex_unlock() to use. We could replace
    // this with a thread-local, not sure which is faster.
    self->mynode = mynode;
}

void clh_mutex_unlock(clh_mutex_t * self) {
    if (self->mynode == NULL) {
        // ERROR: This will occur if unlock() is called without a lock()
        return;
    }
    atomic_store(&self->mynode->succ_must_wait, 0);
}
```

CLH lock


tail
false
true
true
myPred
myNode
Thread A
myPred
myNode
Thread B
CLH lock示意图
https://blog.csdn.net/u013380694

Задачи

1. Можно ли обойтись без `pthreads` для организации thread-local переменных? Предложите свой вариант array-based lock.
2. Попробуйте написать свой MCS / CLH lock.
3. Как можно использовать то, что в замках, основанных на списке, треды, находящиеся далеко от «выхода» могут пореже проверять условие блокировки?