Теория и практика многопоточного программирования

Семинар 5

Неганов Алексей

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

Москва 2020

Spinlock

```
class spin_lock_TAS
{
    atomic<unsigned int> m_spin ;
public:
    spin_lock_TAS(): m_spin(0) {}
    ~spin_lock_TAS() { assert( m_spin.load() == 0);}

    void lock() {
        unsigned int expected;
        do { expected = 0; }
        while ( !m_spin.compare_exchange_weak(expected, 1));
    }

    void unlock() {
        m_spin.store(0);
    }
};
```

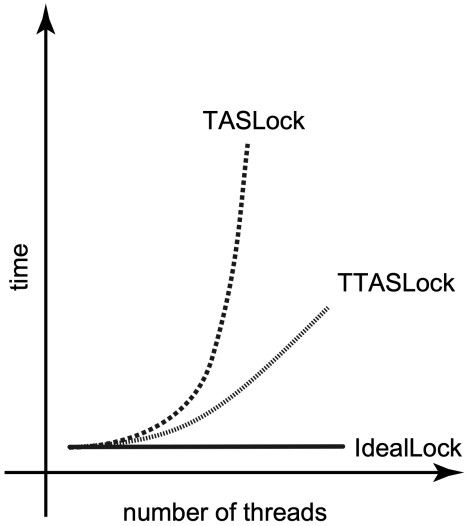
Вопрос

Как в атомарных операциях расставить `memory_order`?

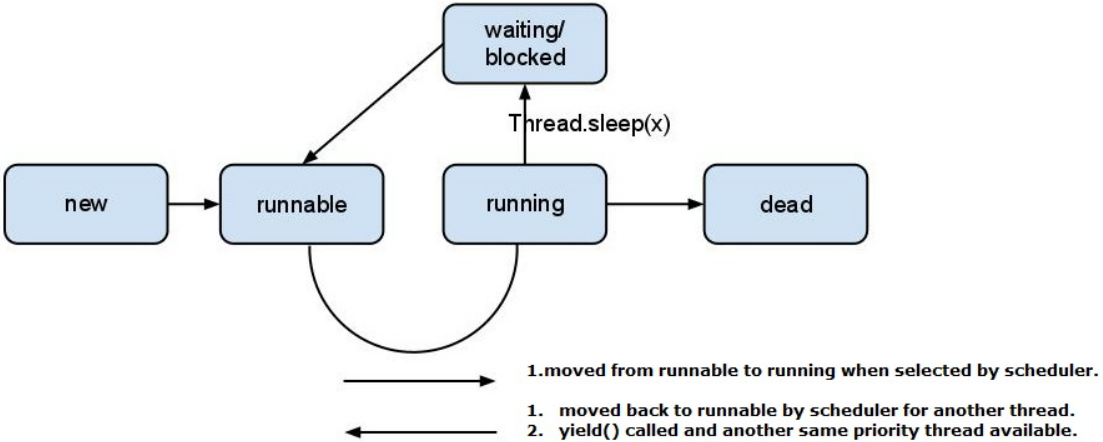
Spinlock

```
class spin_lock_TTAS
{
    atomic<unsigned int> m_spin ;
public:
    spin_lock_TTAS(): m_spin(0) {}
    ~spin_lock_TTAS() { assert( m_spin.load() == 0);}

    void lock() {
        unsigned int expected;
        do {
            while (m_spin.load());
            expected = 0;
        }
        while ( !m_spin.compare_exchange_weak(expected, 1));
    }

    void unlock() {
        m_spin.store(0);
    }
};
```

Spinlock

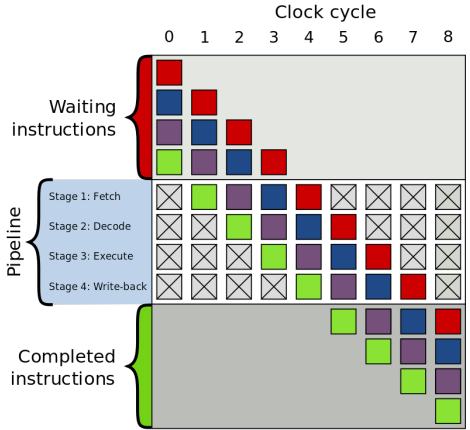


Жизненный цикл потока: yield vs sleep



Как нам обустроить spin loop

```
while(flag.load() == 0) {
  __asm volatile ("pause" ::: "memory");
}
```

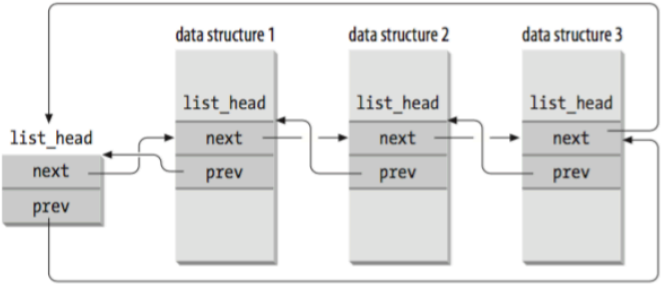


Ticket lock

```
class ticket_lock
{
    std::atomic_size_t now_serving = {0};
    std::atomic_size_t next_ticket = {0};

public:
    void lock() {
        const auto ticket = next_ticket.fetch_add(1);
        while (now_serving.load() != ticket);
    }

    void unlock() {
        const auto successor = now_serving.load() + 1;
        now_serving.store(successor);
    }
};
```

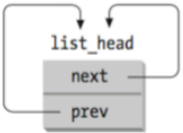
Вопрос

Как в атомарных операциях расставить `memory_order`?



(a) a doubly linked listed with three elements

(b) an empty doubly linked list



Offtop: контейнеры на Си

```
#include <stddef.h>
// #define offsetof(st, m) \
//     ((size_t)((char *)&((st *)0)->m - (char *)0))

#if defined(__GNUC__) || defined(__clang__)

#define container_of(ptr, type, member)                                 \
  ({                                                                    \
    const typeof(((type *)0)->member) *__mptr = (ptr);                  \
    (type *)((char *)__mptr - offsetof(type, member));                  \
  })

#elif defined(_MSC_VER)

#define container_of(ptr, type, member) (type*)((char*)ptr - offsetof(type, member))

#endif
```

Array-based queue lock

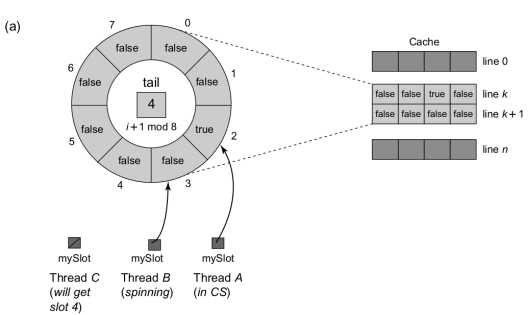
```
class ALock {
    pthread_key_t idx_key;
    atomic<uint64_t> tail;
    atomic<uint8_t> *flag;
    size_t size;

    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);
    }
```

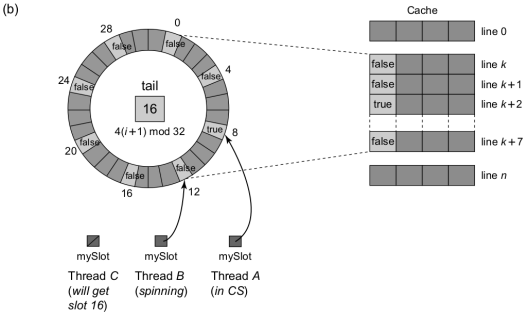
```
public:
    ALock(size_t cap) : tail(0), size(cap) {
        pthread_key_create(&idx_key, NULL);
        flag = new atomic<uint8_t>[cap];
        flag[0].store(1);
        for (size_t i = 1; i < cap; i++)
            flag[i].store(0);
    }

    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





CLH lock

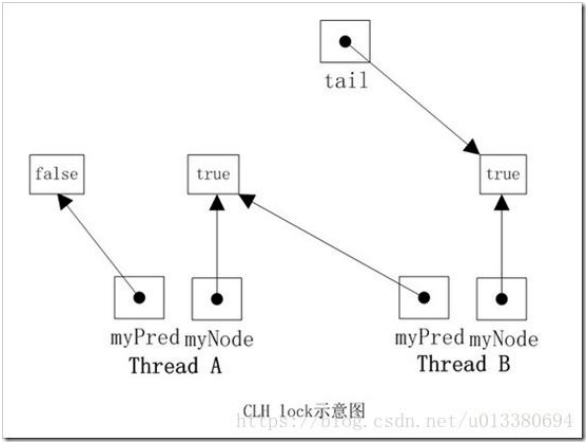
```c
struct clh_mutex_node {
    _Atomic char succ_must_wait;
};

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. Подумайте, как нужно поставить `memory_order` в обращениях к атомарным переменным в примерах с семинара.
2. Можно ли обойтись без `pthreads` для организации thread-local переменных? Предложите свой вариант array-based lock.
3. **(Обязательная)** Напишите свои mutex'ы, использующие yield / exponential backoff. Сравните производительность TAS lock / TTAS lock / ticket lock. Предлагается использовать C++11 (и выше), при желании можно GNU C11 и pthreads.