

Теория и практика многопоточного программирования

Семинар 4

Неганов Алексей

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

Москва 2020

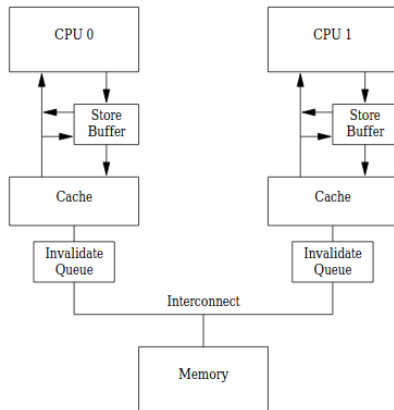
LL/SC:

```
word LL(word *ptr) {  
    return *ptr; // and set cache line status bit simultaneously  
}  
  
bool SC(word *ptr, word newval) {  
    if ( /* cache line bit is set */ ) {  
        *ptr = newval;  
        return true ;  
    }  
    else  
        return false ;  
}  
  
bool CAS(word *ptr, word oldval, word newval) {  
    if (LL(ptr) == oldval)  
        return SC(ptr, newval);  
    return false ;  
}
```

```
int a = 0;
int b = 0;

void foo(void) {
    a = 1;
    b = 1;
}

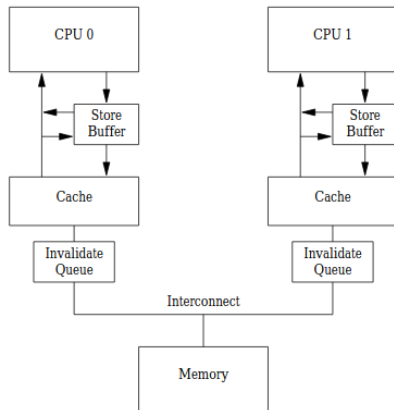
void bar(void) {
    while(b == 0);
    assert(a == 1);
}
```



```
int a = 0;
int b = 0;

void foo(void) {
    a = 1;
    smp_wmb();
    b = 1;
}

void bar(void) {
    while(b == 0);
    smp_rmb();
    assert(a == 1);
}
```



```
// 'memory_order_relaxed' argument is omitted for brevity
atomic_t x, y;
x.store(0);
y.store(0);

y.store(20);      if (x.load() == 10) {      if (y.load() == 10)
x.store(10);      assert (y.load() == 20)      assert (x.load() == 10);
                  y.store (10);
                  }
```

Load/store операции могут переупорядочиваться, может сработать и тот, и другой assert.

```
x.store(1);      y = x.load();
x.store(2);      z = x.load();
                  assert(y <= z);
```

Гарантия – assert не может сработать.

```
atomic_t x, y;  
x.store(0);  
y.store(0);  
  
y.store(20);    if (x.load() == 10) {    if (y.load() == 10)  
x.store(10);    assert (y.load() == 20)    assert (x.load() == 10);  
                y.store (10);  
            }  
}
```

Load/store операции могут выполняться лишь в том порядке, что и в коде, ни один assert выполниться не может.

```
atomic_t x;  
x.load(0);  
int y = 0;  
  
y = 1            if (x.load() == 2)  
x.store(2);      assert (y == 1)
```

Запись в y в коде происходит до записи в x, следовательно, assert выполниться не может.

```
load memory, register ;  
membar #LoadLoad | #LoadStore ; // acquire barrier  
  
...  
  
membar #LoadStore | #StoreStore ; // release barrier  
store regiser, memory
```

Модели памяти: acquire/release

```
#define A memory_order_acquire
#define R memory_order_release

// thread 1                                // thread 2
y.store(20, R);                             x.store(10, R);

// thread 3                                // thread 4
assert(y.load(A) == 20 && x.load(A) == 0);  assert(y.load(A) == 0 && x.load(A) == 10);
```

В sequential consistency один из assert сработает, в acquire/release — оба могут пройти.

```
// 'memory_order_XXX' argument is omitted for brevity
atomic_t x, y;
x.store(0);
y.store(0);

y.store(20);    if (x.load() == 10) {          if (y.load() == 10)
x.store(10);    assert (y.load() == 20)        assert (x.load() == 10);
                y.store (10);
            }
```

В acquire/release assert второго потока пройдёт, а третьего — сработает.