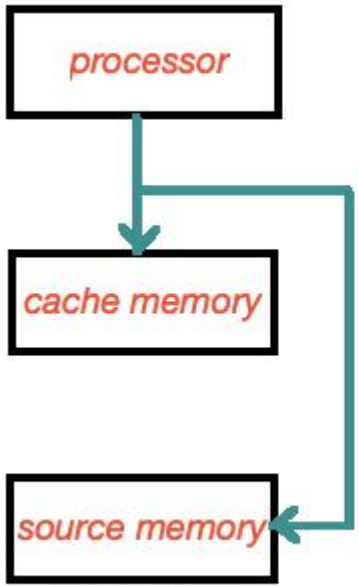
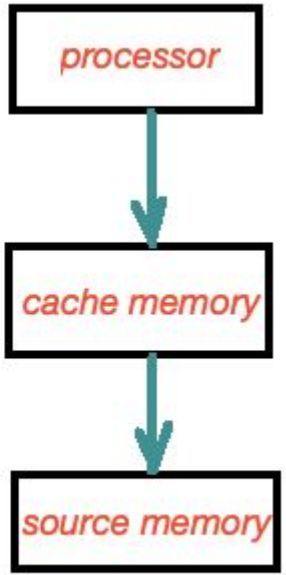
Средства и системы параллельного программирования

Семинар #2
Синхронизация потоков в pthreads

Cache policy

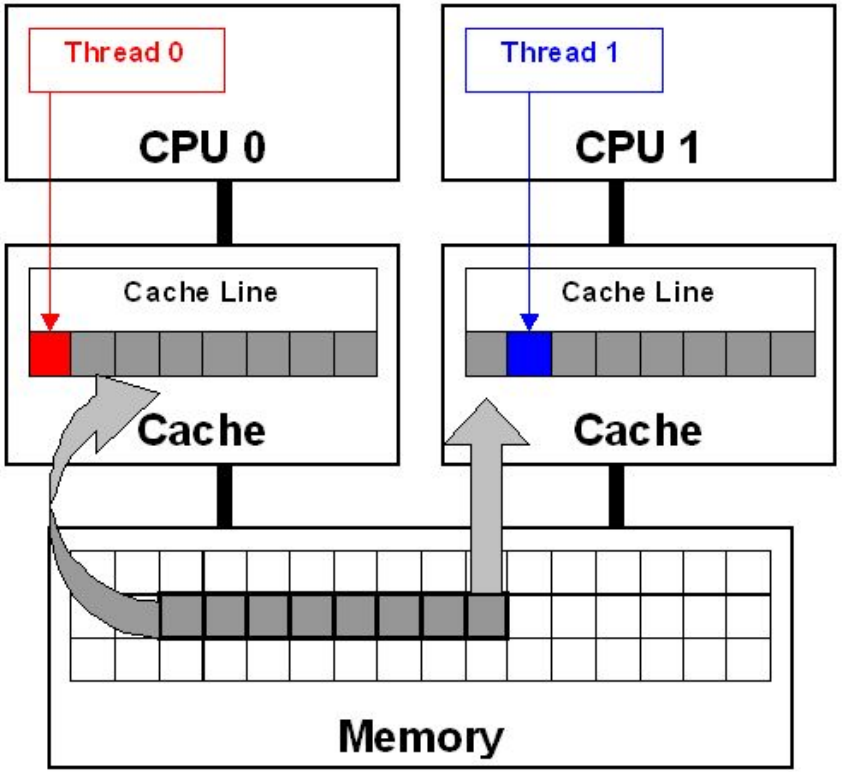


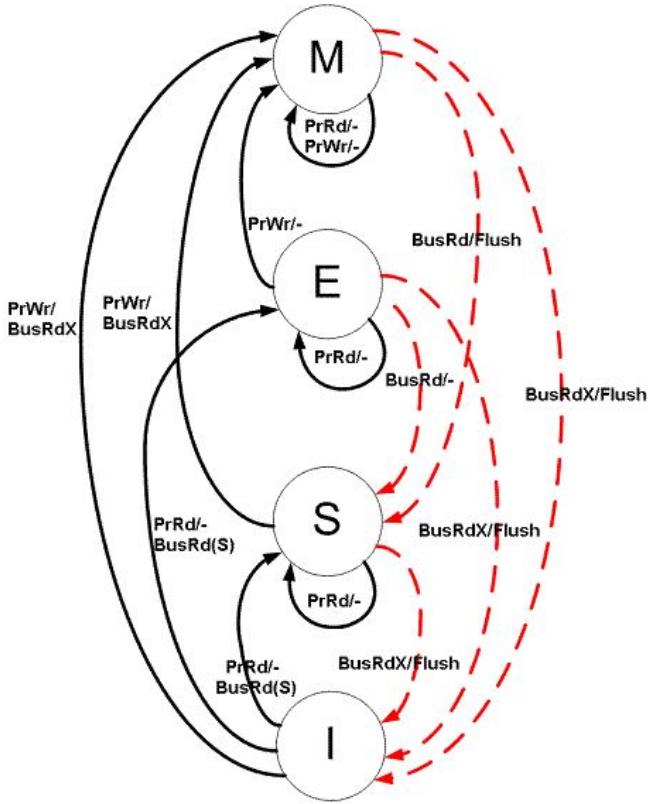
write-through



write-back

Cache coherence





Cache coherence vs race condition

Write Propagation

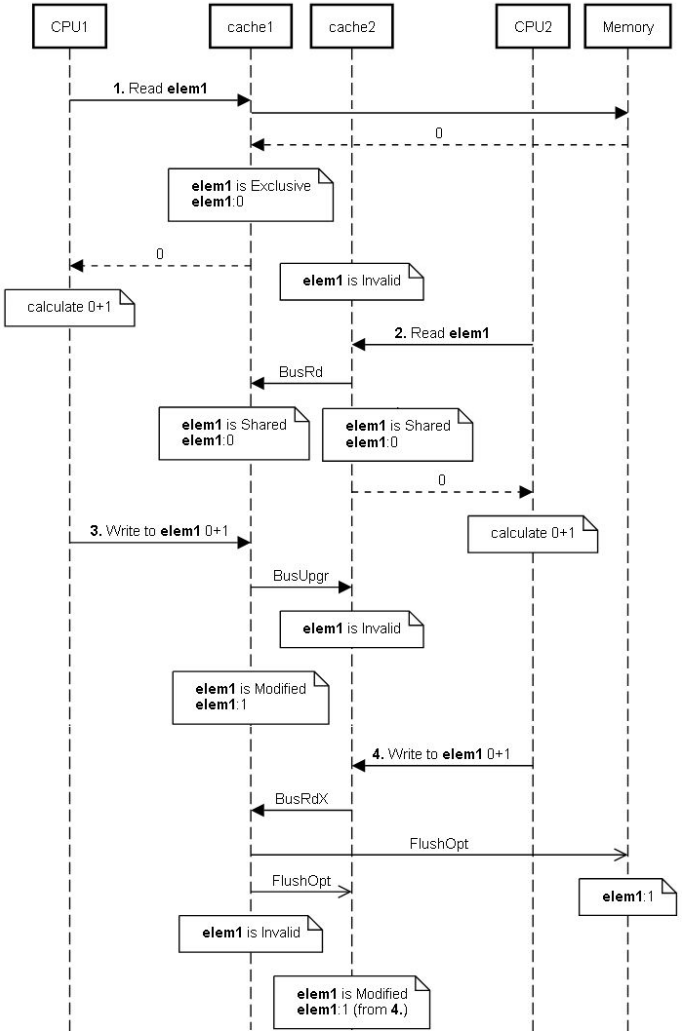
Changes to the data in any cache must be propagated to other copies (of that cache line) in the peer caches.

Transaction Serialization

Reads/Writes to a single memory location must be seen by all processors in the same order.

Что нас не устраивает?

1. Нам часто нужен определённый порядок выполнения инструкций (доступы к ресурсам, и.т.п)
2. Многие операции не атомарны, например a += val;



Peterson algorithm

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
    int other;                          /* number of the other process */

    other = 1 – process;                /* the opposite of process */
    interested[process] = TRUE;         /* show that you are interested */
    turn = process;                     /* set flag */
    while (turn == process && interested[other] == TRUE)  /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
    interested[process] = FALSE;        /* indicate departure from critical region */
}
```

в чём недостаток?

Test and set lock (TSL)

TSL RX, LOCK - атомарная запись на регистр значения слова LOCK, присваивание LOCK ненулевого значения.

Основной момент - процессор, выполняющий TSL, блокирует шину памяти ВСЕМ процессорам!

```
int test_and_set(int* lockPtr) {
    int oldValue;
    // -- Start of atomic segment --
    oldValue = *lockPtr;
    *lockPtr = LOCKED;
    // -- End of atomic segment --
    return oldValue;
}
```

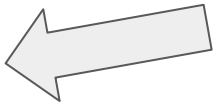
Test and set lock (TSL) (2)

```
enter region:
    TSL REGISTER,LOCK
    CMP REGISTER,#0
    JNE enter_region

    RET


leave region:
    MOVE LOCK,#0
    RET
```

Особого выигрыша от использования TSL не получаем

Mutex

```
mutex_lock:
    TSL REGISTER,MUTEX

    CMP REGISTER,#0
    JZE ok


    CALL thread_yield

    JMP mutex lock
ok:    RET



mutex_unlock:
    MOVE MUTEX,#0
    RET
```



Mutex (2)

```
int pthread_mutex_init(pthread_mutex_t *mutex, pthread_mutexattr_t *attr);

int pthread_mutex_lock(pthread_mutex_t *mutex);

int pthread_mutex_unlock(pthread_mutex_t *mutex);

int pthread_mutex_trylock(pthread_mutex_t *mutex);

int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

pthread_mutex_t должен быть разделяемым(глобальным)!

Атрибуты мьютекса

PTHREAD_MUTEX_NORMAL - при попытке повторного захвата мьютекса (без разблокировки) будет дэдлок

PTHREAD_MUTEX_ERRORCHECK - при попытке повторного захвата мьютекса (без разблокировки) выдаст ошибку

PTHREAD_MUTEX_RECURSIVE - можно повторно захватывать несколько раз (освободить нужно столько же раз)

PTHREAD_PROCESS_SHARED или PTHREAD_PROCESS_PRIVATE - ставим возможность другим ПРОЦЕССАМ брать мьютекс (по умолчанию PTHREAD_PROCESS_PRIVATE)

Spin-блокировка

int pthread_spin_init(pthread_spinlock_t *_lock_, int _pshared_); (pshared = PTHREAD_PROCESS_SHARED или PTHREAD_PROCESS_PRIVATE)

int pthread_spin_destroy(pthread_spinlock_t *_lock_);

int pthread_spin_lock(pthread_spinlock_t *_lock_);

int pthread_spin_trylock(pthread_spinlock_t *_lock_);

int pthread_spin_unlock(pthread_spinlock_t *_lock_);

pthread_spinlock_t должен быть глобальным!

Barrier

```
int pthread_barrier_init(pthread_barrier_t  *barrier,  const pthread_barrierattr_t *restrict attr,
unsigned count);


int pthread_barrier_wait(pthread_barrier_t  *barrier);


int pthread_barrier_destroy(pthread_barrier_t *barrier);
```

Типичная ситуация

```
lock(mutex);
while(!flag) {
   sleep(100);
}
unlock(mutex);
```

Conditional variables (1)

pthread_cond_t

pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t *attr);

pthread_cond_wait(pthread_cond_t *c, pthread_mutex_t *m);

pthread_cond_signal(pthread_cond_t *c);

pthread_cond_broadcast (pthread_cond_t *c);

pthread_cond_destroy(pthread_cond_t *cond);

Conditional variables (2) - а можно без мьютексов?

```
void child_func() {
    done = 1;
    pthread_cond_signal(&c);
}
void parent_func() {
    if (done == 0) {
        pthread_cond_wait(&c);
    }
}
```

Conditional variables (3) - а можно без state?

```
void child_func() {
     pthread_mutex_lock(&m);
     pthread_cond_signal(&c);
     pthread_mutex_unlock(&m);
}

void parent_func() {
     pthread_mutex_lock(&m);
     pthread_cond_wait(&c, &m);
     pthread_mutex_unlock(&m);
}
```

Conditional variable (4)

Всегда (99%) следует работать с conditional_variable под мьютексом и с проверкой какого-то предиката (состояния, размера очереди, флага готовности и.т.п) !

Для проверки предиката всегда (100%) используем while-цикл, а не if! (Spurious wakeups & Mesa semantics)

Что остается за пределами курса

- C/C++ atomic variables
- Модель памяти C++
- Корутины
- Приоритет потоков и их планирование

Что почитать?

- *C++ Concurrency in Action*, Second Edition, Anthony Williams
- *Operating System Concepts (Chapter 4 & 7)*, Abraham Silberschatz
- *Concurrency with Modern C++*, Rainer Grimm
- *A Primer on Memory Consistency and Cache Coherence,* Vijay Nagarajan

Задание

Реализовать класс/структуру MyConcurrentQueue со следующими свойствами:

- атрибут MyConcurrentQueue::queue - произвольная (“*однопоточная*”) реализация очереди фиксированного размера. Тип данных - любой POD-тип.
- метод MyConcurrentQueue::**put**() - положить одно значение в очередь
- метод MyConcurrentQueue::**get**() - взять одно (первое) значение из очереди
- методы put/get могут вызываться одновременно различным (произвольным) числом потоков
- **put**() и **get**() должны корректно работать при любом состоянии очереди - при пустой очереди **get**()-поток ждёт появления элемента, при полной очереди **put**()-поток ждёт освобождения места в очереди
- Если поток A сделал put() или get() раньше потока B, не обязательно, что A выйдет из метода раньше B

Разрешается и рекомендуется добавлять дополнительные атрибуты в класс

Снабдить решение множеством тестов ({1 пишущий поток, N читающих}, {N, 1}, {M, N}, {1, 1})

Дедлайн: через 2 недели, 30.09, Дедлайн: 7.10