Задание №6. Модель памяти

Рубаненко Евгений

Апрель 2017

1 Test-And-Set spinlock

Ниже приведена корректная реализация $Test - And - Set\ spinlock$.

```
#pragma once

#include <atomic>
#include <mutex>

// Test-And-Set spinlock
class TASSpinLock {
public:
    void Lock() {
        while (locked_.exchange(true,                                    (1)
                                std::memory_order_acquire)) {
            std::this_thread::yield();
        }
    }

    void Unlock() {
        locked_.store(false, std::memory_order_release);                 (2)
    }

private:
    std::atomic<bool> locked_{false};
};
```

Неатомарные чтения и записи

Необходимо добиться того, чтобы чтение и запись в критической секции были возможны только между вызовами $lock()$ и $unlock()$ в данном треде. Таким образом, необходимо сделать так, чтобы после успешного вызова $lock()$ другие треды знали, что сейчас происходят изменения, а после вызова $unlock()$ - что они могут начать работать с памятью.
Следовательно, становится логичным использование
$std :: memory\_order\_acquire$ в (1) и
$std :: memory\_order\_release$ в (2).

Happens-before

С помощью $std :: memory\_order\_acquire$ и $std :: memory\_order\_release$ построится связь $Synchronized - with$. Далее по транизитивности
$ProgramOrder \implies Synchronized - with \implies ProgramOrder$
получим отношение $Happens - before$.
Так как в функциях происходит только одна атомарная операция, то такой гарнтии достаточно ($Synchronized - order$ появляется в данном

случае автоматически).

Гарантии упорядочивания

Нельзя ослабить до $std :: memory\_order\_relaxed$, так как в данном случае необходима не только атомарность записи, но и синхронизация с другими тредами (если синхронизация будет отсутствовать, то тогда несколько тредов смогут войти в критическую секцию).