Лекция 31
Модель памяти C/C++
(C11+, C++11+)

Атомарность работы с памятью

- Операции чтения или записи для натурально выравненных значений размером не больше машинного слова — атомарны на практике.
- Но стандарты Си и Си++ этого не гарантируют!
- Операции с данными размера больше машинного слова НЕ АТОМАРНЫ.
- Операции чтение-модификация-запись — НЕ АТОМАРНЫ (++, +=, и аналогичные).

Data race

- Data race — ситуация в многопоточной программе, когда
  - Два потока одновременно работают с одной и той же переменной (ячейкой памяти)
  - Один из потоков пишет в эту ячейку памяти
- Практически всегда data race — это ошибка в программе!
- Data Race — это UNDEFINED behaviour, т. е. компилятор и среда выполнения вольны делать что угодно
- Работа с atomic-переменными не приводит к Data Race

Атомарные переменные C++11

- <atomic>
- std::atomic<T>
- Специализация возможна для любого типа, но не все специализации будут lock-free
- Гарантированно только std::atomic_bool будет lock-free
- std::atomic_bool, std::atomic_char, …
- Определен набор методов и перегружены основные операции

Атомарные переменные C11

- <stdatomic.h>
- _Atomic
  _Atomic int x;
  int * _Atomic y;
- atomic_bool, atomic_int …
- Только atomic_bool гарантируется lock-free
- Поддерживаются основные операции (=, +=, ...)

Атомарные переменные

- Параллельная работа с атомарными переменными
  - Атомарна
  - Свободна от data race
- Если переменная используется из нескольких нитей одновременно (не в критической секции), она должна быть atomic
- Lock-free – для работы с переменной используются специальные инструкции процессора
- Иначе может использоваться мьютекс

Модель памяти

- Компилятор может переставлять операции работы с памятью, при условии, что сохраняется семантика ОДНОПРОЦЕССНОЙ программы.
  - x = 1; y = 2; - переставить можно
  - x = 1; y = x + 1; - переставить нельзя
- При выполнении программы процессор может выполнять спекулятивные загрузки из памяти, самостоятельно переставлять операции
- Процессор реализует протокол поддержки когерентности кешей

Модель памяти

- Минимальные требования к перестановкам операций с памятью между разными нитями задаются *моделью памяти*.
- Модель памяти — это свойство процессорной архитектуры.
- *Сильная модель памяти* (strong memory model) требует, что если одно процессорное ядро выполняет запись в память в определенной последовательности, все другие процессорные ядра видят изменения значений в памяти в той же самой последовательности (x86).
- *В слабой модели памяти* процессор может переставлять порядок и операций чтения, и операций записи при условии, что такие перестановки не изменяют результат вычисления программы состоящей из одной синхронно выполняющейся нити (ARM).

Наблюдаемый порядок операций

- Наблюдаемый порядок изменения значений в памяти в одной нити может отличаться от наблюдаемого порядка изменения значений в памяти в другой нити.

```
y = 0;
x = 0;
// ...
x = 1;
y = 2;
// ...
x = 3;
```

```
if (y == 2) {
  z = x;
}
```

- Во второй нити z может принимать значения 0, 1 или 3!

memory_order

- Операция с _Atomic окружена операциями с обычными (не атомарными) переменными
- memory_order определяет, какие ограничения накладываются на перестановки обычных операций вокруг атомарной операции

memory_order

- memory_order_relaxed — самый слабый
- memory_order_acquire
- memory_order_release
- memory_order_acq_rel
- memory_order_seq_cst — самый сильный

memory_order_relaxed

- memory_order_relaxed - самый слабый порядок работы с памятью. Операция с атомарной переменной в этом режиме предполагает только атомарность и не дает никаких гарантий сохранения порядка записей в память или чтения из памяти для операций вокруг данной, если только они не выполняются с более сильным порядком работы с памятью.

Acquire/release

- В нити A выполняется сохранение в память в режиме memory_order_release
- В нити B выполняется чтение из той же самой переменной в режиме memory_order_acquire.
- Все записи в память (и обычные неатомарные, и в режиме memory_order_relaxed), которые с точки зрения нити A выполняются раньше атомарнй операции сохранения в память гарантированно становятся видимыми в нити B.
- После выполнения атомарной загрузки из памяти в нити B она гарантированно увидит все, что нить A записала в память.

Sequentally consistant

- Последовательно консистентный режим работы — самый строгий.
- Атомарная операция, выполняющаяся в режиме memory_order_seq_cst:
  - является и release, и acquire операцией
  - все нити видят все операции, выполненные в этом режиме, в одном и том же порядке.
- Этот режим действует по умолчанию для атомарных операций.

Операции с atomic

- Загрузка (load)
- Сохранение (store)
- Обмен (exchange)
- Сравнение-обмен (compare_exchange)
- Арифметика (fetch_add, fetch_sub, …)
- Барьер

_Atomic и volatile

- Чтение/запись volatile переменной НЕ ОБЯЗАНЫ быть атомарными
- Влияние операций с volatile на окружающие операции с обычными переменными не определено (компилятор может переставлять как угодно)
- Data Race на volatile — это также UB
- Volatile не предназначен для использования в параллельных программах