Лекция 20
Классические задачи синхронизации

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

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

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!

std::memory_order

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

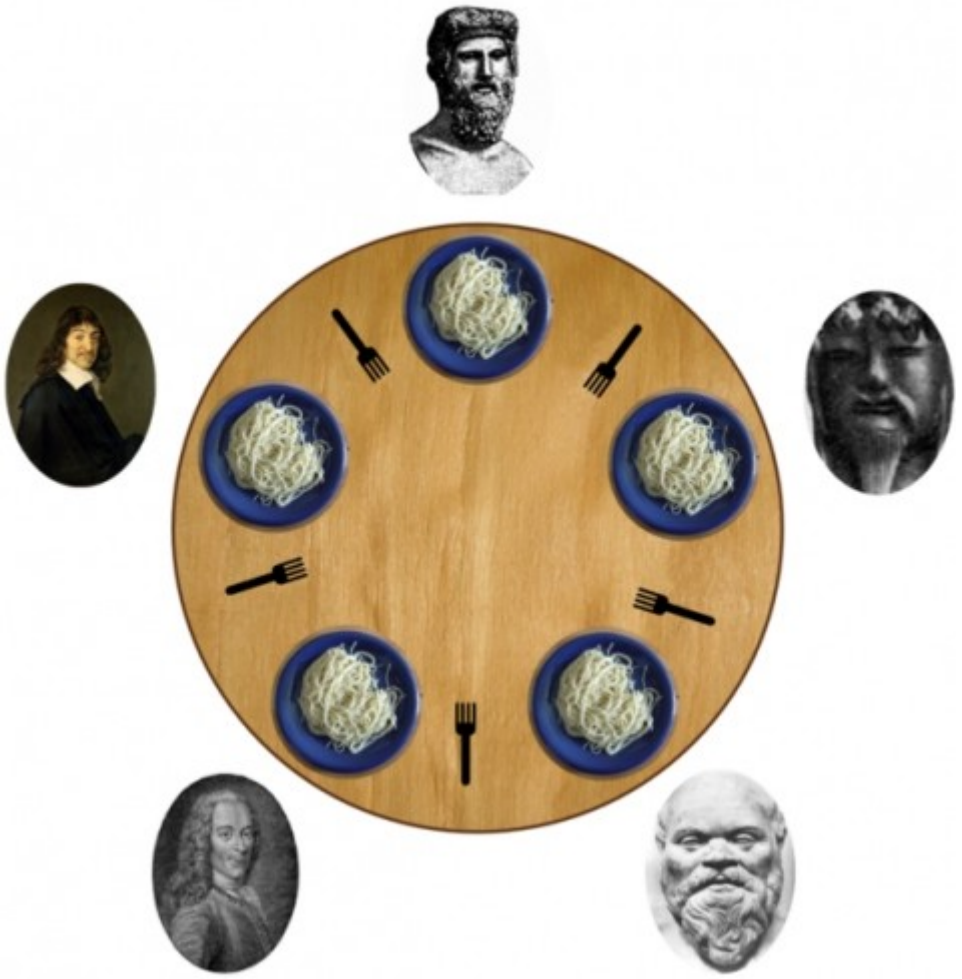
std::memory_order

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

Atomic<T> 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
- Семантика volatile слишком различается на разных платформах!

Обедающие философы



Наивное решение

```
void philosopher ( int i ) {
  while (TRUE) {
    think () ;
    take_fork ( i ) ;
    take_fork ( ( i + 1 ) % N ) ;
    eat () ;
    put_fork ( i ) ;
    put_fork ( ( i + 1 )  % N ) ;
  }
  return;
}
```

Deadlock

- Возможна ситуация, когда все философы одновременно захотят есть и возьмут левую от себя вилку — никто не сможет начать есть
-
- «Обедающие философы» показывает важность корректного порядка занятия ресурсов при входе в критическую секцию

Избежание блокировки

- При N = 2 задача сводится к:

- Процесс 1:

```
take_fork(0);
take_fork(1);
```

- Процесс 2:

```
take_fork(1);
take_fork(0);
```

- Изменение порядка взятия вилок в процессе 2 решает проблему!

Избежание блокировки

- Каждый процесс может сначала взять вилку с меньшим номером, потом взять вилку с большим номером
- Недостатки:
  - Операция по взятию вилок неатомарна
  - Могут появляться цепочки философов, которые взяли вилку с меньшим номером, но ждут вилку с большим номером — такая цепочка разрушится только когда философ с максимальным номером закончит есть
  - Требуется отношение порядка на множестве вилок

Избежание блокировки

- Проверяем состояние соседей философа под мьютексом
- Если философ не может начать есть, он засыпает на условной переменной
- Когда состояние изменится, его разбудят
- Недостатки:
  - Мьютекс на весь стол, только один философ может проверить состояние
  - Немасштабируемо

Читатели и писатели

- Дана некоторая разделяемая область память
- К этой структуре данных может обращаться произвольное количество «читателей» и произвольное количество «писателей»
- Несколько читателей могут получить доступ одновременно, писатели в этот момент не допускаются
- Только один писатель может получить доступ, другие писатели и читатели должны ждать

Решение 1

- Первое решение: читатель может войти в критическую секцию, если нет писателей
- Это решение несправедливо, так как отдает предпочтение читателям
- Плотный поток запросов от читателей может привести к тому, что писатель никогда не получит доступа к критической секции: ситуация «голодания» (starvation)

Решение 2

- Отдадим предпочтение писателям, то есть читатель не входит в критическую секцию, если есть хотя бы один ожидающий писатель
- Данное решение отдает приоритет писателям, и тоже несправедливо
- Возможно «голодание» (starvation) читателей

Решение 3

- Третье решение: не отдавать никому приоритета, просто использовать мьютекс
- Не используется возможность одновременного чтения

Решение 4

- Формируем очередь запросов
- Несколько идущих подряд в очереди запросов на чтение могут выполняться параллельно
- Запросы на запись выполняются в эксклюзивном режиме

Производители-потребители (producer-consumer problem)

- Дан буфер фиксированного размера (N), в котором размещается очередь.
- Производители добавляют элементы в конец очереди, если буфер заполнился, производители засыпают
- Потребители забирают элементы из начала очереди, если буфер пуст, потребители засыпают

Спящий парикмахер (sleeping barber)

- В парикмахерской имеется одно кресло для стрижки и N кресел для ожидающих посетителей
- Если нет посетителей, парикмахер спит
- Если приходит посетитель и кресло для стрижки свободно, посетитель садится в него и парикмахер начинает его стричь
- В противном случае посетитель садится в кресло для ожидающих
- Если все кресла заняты, посетитель уходит