Теория и практика многопоточного программирования

ЛЕКЦИЯ 5. ПРОБЛЕМЫ МНОГОПОТОЧНОСТИ

В прошлый раз …

Модель исполнения кода

Корректность программы

«А что если…» - вероятность ошибки

Время как абстракция – bounded timestamps

Темы лекции

Общие проблемы многопоточности

Проблемы работы с разделяемой памятью

Разделяемые объекты, синхронизация, примитивы синхронизации

Базовая проблема: race condition (гонка данных)

Ситуация, в которой работа программы *зависит от порядка выполнения инструкций* в параллельных потоках и (**важно!**) *нарушает* заложенную *логику* исполнения.

Порождаемые «баги» являются «плавающими».

```
volatile int x;
```

```
// Поток 1:
while (!stop)
{
  x++;
  …
}
```

```
// Поток 2:
while (!stop)
{
  if (x%2 == 0)
    System.out.println("x=" + x);
  …
}
```

Базовая проблема: race condition (гонка данных)

Пути решения:

- Копирование данных (для атомарного копирования)
- Примитивы синхронизации
- Комбинация (неатомарное копирование)

```
volatile int x1, x2;
```

```
// Поток 1:
while (!stop)
{
  synchronized(SomeObject)
  {
    x1++;
    x2++;
  }
  …
}
```

```
// Поток 2:
while (!stop)
{
  long cached_x1, cached_x2;
  synchronized (SomeObject)
  {
    cached_x1 = x1;
    cached_x2 = x2;
  }
  if ((cached_x1 + cached_x2)%100 == 0)
    DoSomethingComplicated(cached_x1, cached_x2);
  …
}
```

Проблема ABA

Сравнение указателей как способ сравнить объекты приводит к ложным срабатываниям.

Управляемые (managed) языки следят за ссылками, но можно их обмануть.

Примитивы синхронизации

Mutex

FastMutex

FastMutexUnsafe

GuardedMutex

CriticalRegion

GuardedRegion

ExecutiveResource

Event

Semaphore

ExecutiveSpinLock

QueuedSpinLock

InterruptSpinLock

Timer

ConditionVariable

SlimReaderWriterLock

InitOnceInitialization

Примитивы синхронизации

Все примитивы типа Mutex / CritialRegion / Lock построены на базе compare-and-set инструкции.

Два основых синхронизационных подхода:

- Критическая секция (эксклюзивный доступ), реализующая принцип **mutual exclusion** (взаимное исключение).
- Ограниченный пул (ограниченный доступ), опирающийся на принцип семафора (**semaphore**).

Lock contention (спор за блокировку)

Невозможность увеличить скорость программы при масштабировании.

Lock contention не является «ошибкой», так как не влияет на логику приложения, но является примером **плохого дизайна** или **безысходности**.

Dead lock (взаимная блокировка)

Ситуация, когда 2+ потоков пытаются эксклюзивно* завладеть 2+ объектами в неодинаковом порядке, может привести к **взаимному ожиданию**.

Условия Коффмана возникновения ВБ:

- Условие взаимной блокировки
- Удержание ресурса + ожидание другого
- Невозможность принудительного освобождения
- Условие взимного кругового ожидания

Live lock (имитация деятельности)

Осуществление множества операций из-за конфликта интересов (ресурса).

Основная причина возникновения – «вежливые алгоритмы» или возможность прекращения ожидания.

Потерянный сигнал (lost signal)

Попадание потока в состояние ожидания сигнала после того, как сигнал был отправлен.

- Ожидание изменения состояния объекта
  - `while (curr == prev) { prev = curr; }`
- Подписка на события после наступления события

Заброшенные замки (abandoned locks)

Поток покидает критическу секцию, не освободив ресурса. Ожидающие потоки никогда не смогут заполучить блокировку.