Теория и практика многопоточного программирования

ЛЕКЦИЯ 4. ФОРМАЛЬНОЕ ПРЕДСТАВЛЕНИЕ МНОГОПОТОЧНОЙ СИСТЕМЫ

В прошлый раз...

Процессы и потоки

Инструкции x86

Видимость результатов

Модель упорядоченности доступа к памяти

Атомарность и атомарные примитивы

Темы лекции

Уровни абстракции программы

Корректность программы

Время как абстракция

«А что если…» - вероятность ошибки

Уровни «понимания как работает программа»

Проблема стохастичности

«Типичный» подход к написанию многопоточной программы:

1. Мой код последовательный и уникально исполняется на процессоре
2. У меня несколько процессоров?!
3. Поделим код на куски, пусть они исполняются на разных процессорах!
4. См. п. 1.

Если есть спрос, должен быть и инструмент

Мой код последовательный и уникально исполняется на процессоре

«Изобретены» **примитивы синхронизации**

- «Синхронизации» – позволяют гарантировать исполнения правил межпоточного взаимодействия (для обеспечения уникальности и последовательности)
- «примитивы» - потому что рассмативаются как неделимые кирпичики, атомарные инструменты при построении многопоточной задачи

Примитивов много, но они эквивалентны и имеют в основе одни и те же инструкции
(Monitor, Mutex, SpinLock, Semaphore)

Примитивы

Согласно теореме об **эквивалентности примитивов с одинаковым числом консенсуса** (*далее в курсе*), любой из них может быть реализован через любые другие примитивы одного уровня консенсуса.

Их существование – исключительно удобство уровня абстракции, который приближает использование низкоуровневых сценариев к объектно-ориентированному мышлению

Проблема корректности

Вопрос, старый как мир: что такое корректная программа?

- Насколько широк диапазон входных данных?
- Где проходит граница системы? (поток, сессия ОС, компьютер, домен, планета Земля?)
- Корректна ли программа, генерирующая разные данные в «одинаковых» условиях?
- Если число влияющих факторов велико?
- Если программа **пока** работает корректно?

Проблема корректности

Частая проблема – недооценка «несущественной ошибки»

- Вероятность коллизии для неатомарной операции невелика
- Вероятность коллизий для hash-функции (обход полного сравнения)
  - Парадокс дней рождения

Время как абстракция

Базовые понятия времени

- Предшествование событий (событие A случилось раньше B)
  - Всегда ли это важно, предсказуемо?
  - Не раньше, не позже, не одновременно, ... А как тогда?
- Одновременность событий. Моментальность
  - Многие инструкции неатомарны. Что такое окончание её работы?
    - Последний такт процессора
    - Попадание в кэш (какой?)
    - Попадание в память
    - **Видимость результата соседним процессором**

Порядок во времени

Последовательность может меняться компилятором – беда «умозрительного» исполнения

Ключевое значение имеет взаимовлияние инструкций.

- Если результат действия зависит от исполнения другой инструкции
- «Квантовый» эффект наблюдения: команды наблюдения влияют на результат

Временные отметки

Не то же самое, что и время (дискретно, ограничено)

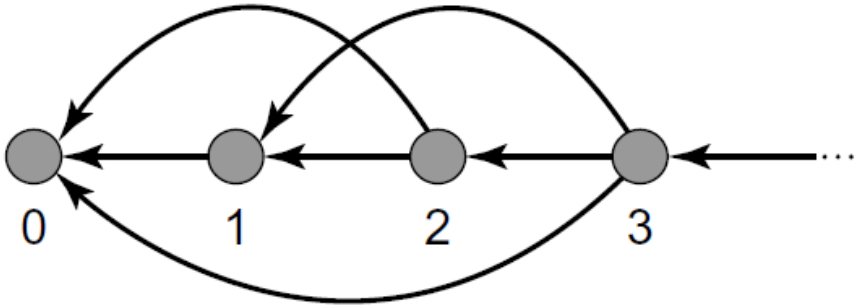
Не то же самое, что часы (не всегда подходят)

Свойства:

- Монотонно возрастает
- В некоторой степени уникальны
- Может меняться постоянно (RDTSC) или по запросу значения

Проблемы:

- Переполнение
- Точность (атом – такт)
- Уникальность



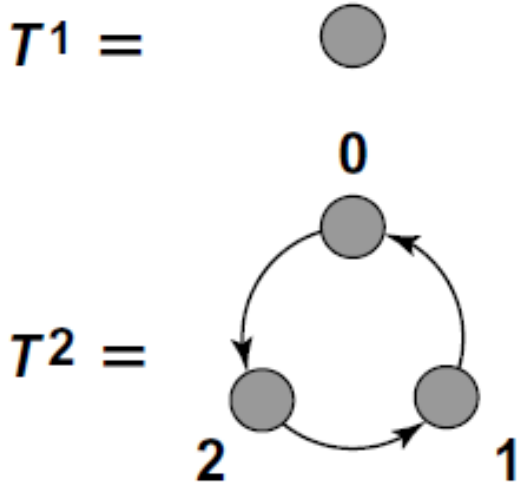
Решаем проблему переполнения

Монотонность может быть ограниченной и неограниченной.

Формализуем требования:

- a→ b, однако, a → b + b → c ≠ a → c

Рассмотрим граф, где временная метка «ползает по рёбрам».



Определения и принцип построения Bounded Timstamps

Есть граф G

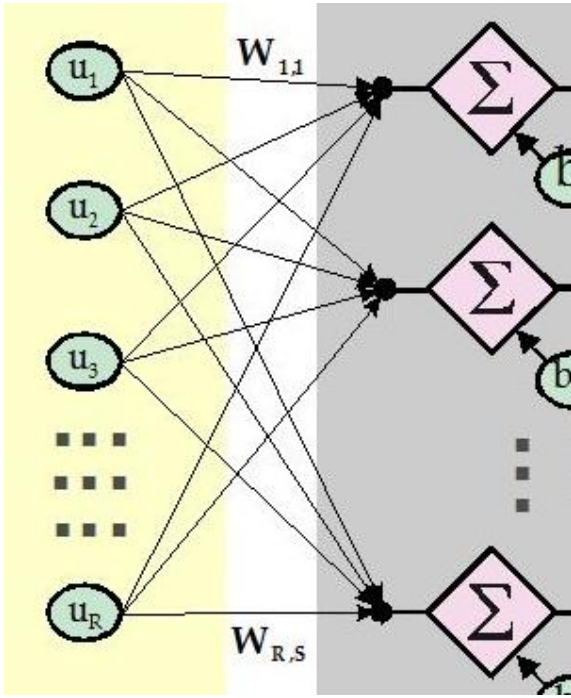
Подграф A доминирует над подграфом B, если для каждой вершины A есть стрелка в каждую вершину B

Произведение графов A ○ B – замена всех вершин ***x*** из A на подграф B, с условием, что если
x>y, то и $B_x > B_y$

T^1 – один узел

T^2 – циклическая тройка

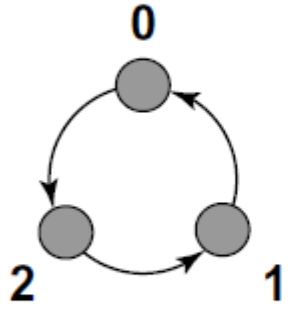
$T^k = T^2 \circ T^{k-1}$



Bounded TS

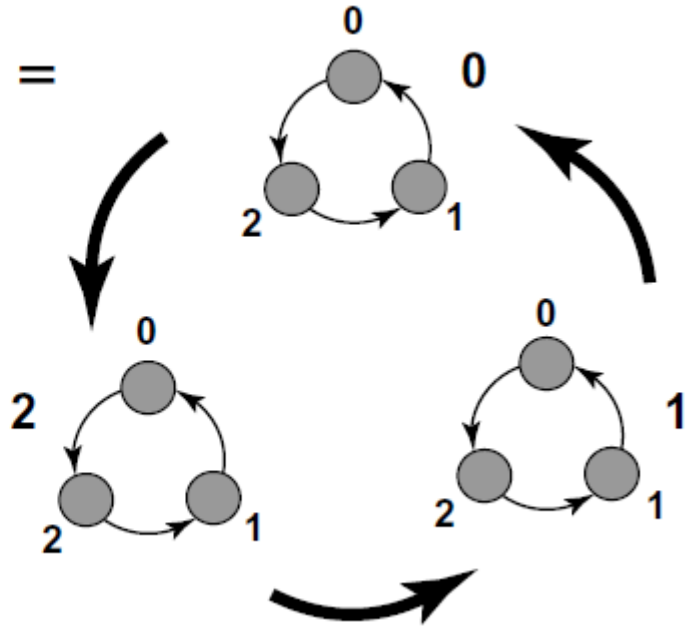
$T^1 =$

$T^2 =$



$T^k = T^2 * T^{k-1}$

$T^3 =$



Выводы

Существуют абстракции разной «детализации»

Корректность программы - условная характеристика, нужно понимать эти условия

«Статистически несущественную ошибку» часто недооценивают

Время можно сделать удобным