ТЕОРИЯ И ПРАКТИКА МНОГОПОТОЧНОГО ПРОГРАММИРОВАНИЯ

Тема 7

Синхронизация при помощи примитивов. Общие проблемы средств организации параллельного доступа.

Д.ф.-м.н., профессор А.Г. Тормасов
Базовая кафедра «Теоретическая и Прикладная Информатика», МФТИ

Тема

- Типовые аппаратно реализованные примитивы синхронизации
- “Сборка” более сложных объектов синхронизации из более простых
- Проблемы стандартных средств организации параллельного доступа
  - стандартных блокировок
  - примитивов типа Compare and Set
  - объединения примитивов.

От простого к сложному

- Можно ли из простых синхронизационных объектов получить сложные?
- Может быть, да?
- Например, в ядре Windows можно пользоваться множеством примитивов
- И все они организованы на платформе x86!

Примитивы Windows 7

- Mutex
- FastMutex
- FastMutexUnsafe
- GuardedMutex
- CriticalRegion
- GuardedRegion
- ExecutiveResource
- Event
- Semaphore
- ExecutiveSpinLock
- QueuedSpinLock
- InterruptSpinLock
- Timer
- ConditionVariable
- SlimReaderWriterLock
- InitOnceInitialization

Примитивы

- все они «собраны» из каких то других операций
  - критическая секция (CriticalRegion) реализуется через одну ячейку и атомарную операцию Compare-And-Set.
- из каких низкоуровненвых «аппаратных» примитивов можно собрать более высокоуровневые?
- сколько памяти для этого потребуется?
- Пример нестандартного применения – распределенная файловая система, объединенная одной SCSI шиной.

Что делать?

- проанализировать примитивы и алгоритмы
- ввести классификацию
  - какие алгоритмы можно реализовать какими примитивами
- Об этом – при обсуждении задачи о консенсусе
  - Могут ли много «слабых» примитивов сложным алгоритмом корректно решить задачу, требующую хотя бы один более «сильный» примитив

«Сборка» алгоритмов

- Можно ли собрать один алгоритм из другого? Что такое «собрать»?
  - Пусть существует релизация алгоритма в виде программы.
  - Будем считать, что внутри программы (кода функций и модулей) есть обращения (вызовы, синхронные или асинхронные) к одному или нескольким экземплярам синхронизационных объектов через соответствующий АПИ (без каких либо «прямых доступов к переменной» и других нарушений инкапсуляции).
  - Алгоритм «собран» из синхронизационных объектов
  - Так же, естественно, мы можем утверждать, что алгоритм реализует какой либо синхронизационный примитив, или, в более общем случае, решает какую то задачу, используя синхронизационные объекты.
- Фактически, это композиция объектов

Проблемы стандартных средств

- Как обычно решаются задачи, возникающие при написании параллельных программ?
  - Синхронизационные примитивы, которые поставляет аппаратная платформа
    - Атомарные операции типа Compare and Swap и их родственники
  - Разнообразные блокировки, построенные на их основе

Проблемы стандартных блокировок

- Взаимная блокировка (тупик, deadlock)

Оба процесса хотят взять A и B

Процесс1	Процесс2
Захватывает A	Захватывает B
Пытается захватить B	Пытается захватить A
и ждет его освобождения!	и ждет его освобождения!

Условия Коффмана

- Условия, при которых наступает взаимная блокировка (Коффман, 1971)
  - Условие взаимного исключения, то есть ресурс не может в один момент времени быть использован более чем одним потоком,
  - Условие удержания и ожидания, то есть поток захватывает какие то ресурсы и ожидает, когда сможет захватить еще ресурсы, не освобождая уже захваченные,
  - Условие неперераспределяемости, или невозможности принудительного освобождения уже захваченного ресурса кем либо, кроме захватившего его потока,
  - Условие взаимного кругового ожидания, или существования «круга», в котором каждый участник ждет одного или более ресурсов, захваченных другими участниками «круга».

Проблемы стандартных блокировок

- Инверсия приоритета
  - низкоприоритетный поток захватил блокировку и перестал получать кванты
  - Высокоприоритетный, ожидающий блокировки, эффективно понижает приоритет
- Конвоирование
  - «медленно» исполняющийся поток, периодически захватывающий блокировку
  - Высокоприоритетный поток, который периодически ожидает освобождения блокировки
  - В результате быстрый поток тащится со средней скоростью медленного (конвоирует)

Проблемы CAS

- крайняя сложность в разработке
  - Например, обход проблемы работы только с 1 словом
    - Как ни странно, аппаратный DCAS не спасает
- высокий уровень накладных расходов
  - Блокировки шины
  - Сложные вычислительно алгоритмы

Проблемы композиции

- Крайне сложная и неприятная в решении проблема
  - пример – атомарное переписывание с одного счета на другой
  - Более сложный пример – переполнение хэш таблиц
    - Надо переписать элемент из старой в новую
    - Как быть с выполняемой операцией?
      - Элемент временно невидим
      - Элемент в двух экземплярах
  - Заблокировать обе таблицы на время переписи!
    - Обе таблицы – неблокирующие
    - Зато алгоритм переписи – блокирующий в худшем смысле

Как решать проблемы?

- Никто не знает, как организовать сложную программную структуру с большим количеством блокировок
- Привлекать высококвалифицированных инженеров (дорого!)
- Создается какой то набор соглашений
  - не являются частью программы, а находятся исключительно в голове программиста
  - описываются обычно в документации к программе/комментариях

Выводы

- Для реализации всегда используются только аппаратные примитивные операции
- Практически всегда алгоритмы синхронизации являются композицией из других алгоритмов
- Создание композиционных алгоритмов – крайне сложная задача, «хорошие» составляющие могут дать «плохой» алгоритм
- Все используемые стандартные подходы имеют свои весьма сложные в решении проблемы

(c) А. Тормасов, 2010-11 г.

Базовая кафедра «Теоретическая и Прикладная Информатика» ФУПМ МФТИ
tor@ crec .mipt .ru_