



ТЕОРИЯ И ПРАКТИКА МНОГОПОТОЧНОГО ПРОГРАММИРОВАНИЯ

Тема 5

Некоторые понятия, используемые при разработке параллельных программ

Д.ф.-м.н., профессор А.Г. Тормасов

Базовая кафедра «Теоретическая и Прикладная Информатика», МФТИ

Тема

- Некоторые понятия, используемые при разработке параллельных программ.
- Уровни абстракции, которыми мыслит программист, и корректность программ.
- Время, простые временные отметки и ограниченные во времени отметки времени.
- О вероятностях и ошибках.

Модель исполнения

- Программист, пишущий программу, имеет «в голове» модель того как она будет исполняться
- Эта модель эволюционирует
 - Когда то были инженеры, которые «знают как течет ток в процессоре»
 - Когда то были инженеры-математики, которые знают как дифурами, переложеными на набор модулей и функций фортран, описывается задача
 - Сейчас есть программисты, оперирующие понятиями объекты, методы, связи и тд в исполняемой программе (забыв о подпрограммах и модулях)
 - Часть программистов манипулирует понятиями «распределенной веб-объектной системы» и тд

Модель исполнения

- Типичная современная модель исполнения параллельных программ:
 1. “мой код исполняется одним процессором в одиночестве без перерывов”
 2. Операторы исполняются последовательно один за другим, «целиком» (псевдоатомарно)
 3. Если несколько процессоров, то поделим код на куски, где опять см пп 1,2!

Но почему?

- А потому, что...
 - Свели задачу к известной (старая история про то «как программист кипятит чайник»?)
 - А как по другому то?
 - А по другому то сверхсложно, недоступно обычному программисту-инженеру!

Как?

- Для этого «изобрели» множество процедур, гарантирующих исполнение кода «в одиночестве» (разновидности критических секций по сути своей)
 - будем называть их «примитивами синхронизации»
 - Примитив потому что обычно неделим
- Их стало очень много (в ядре Windows 7 – 16 типов!)
- Технически они все используют примерно один и тот же набор низкоуровневых операций платформы (x86)

Примитивы

- Согласно теореме об эквивалентности примитивов с одинаковым числом консенсуса (которая будет доказана далее), любой из них может быть реализован через любые другие примитивы одного уровня консенсуса
- Их существование – исключительно удобство уровня абстракции, которым пользуется программист
- Для повышения эффективности, но, в первую очередь, для уменьшения ошибок и приближения «модели апи» к «модели программы»

Корректность программ

- Одно из самых основных понятий остается одинаковым с самых давних времен – корректность
- «Программа должна быть правильной»
- ?!

«корректно работающая программа»

- Что это такое?
- При каких условиях?
 - Если сломался ЦПУ, должна ли программа работать?
 - А на борту космического корабля?
- Будет ли корректна программа, дающие разные ответы в одинаковых условиях?

«корректно работающая программа»

- Если выбранный (и закодированный) алгоритм не всегда работает так как мы ожидали – программа корректна?
 - Мы просто не предусмотрели всех вариантов! – наша ошибка?
 - Предусмотреть все варианты невозможно математически?
- Условно-корректная программа
 - RSA алгоритм базируется на задаче нахождения простых чисел, и на невозможности ее стабильного решения за какое то время
 - Недавно новый результат в фундаментальной математике уменьшил перебор на 2-3 порядка! Заменить ключи 768 бит на 2048 бит.
 - На момент написания – корректна, но потом...
- Один из вариантов задачи корректности программ в математической постановке будет рассмотрен при рассказе об оценке наличия неразрешенных условий гонок

Время как одна из абстракций

- Одновременно сложное и простое понятие, о котором знают все
- Понятие события и отношение
 - Одно событие раньше другого
 - Эти события – одновременны
- Есть «абсолютное знание» с точки зрения которого это правильно или нет
 - А если нет такого?

Отношения и порядок

- А какая инструкция сейчас исполняется?
 - Умозрительное исполнение
- Неатомарность большинства инструкций
- Что есть окончание работы команды? Например,
 - Последний такт процессора
 - Время попадания результата в кэш (который из них)
 - Время попадания результата в память
 - Видимость результата соседним процессором

Отношения и порядок

- Последовательность записей в память может меняться процессором!
- А одновременно ли исполняются две эти инструкции?
 - Если мы не знаем когда она кончается, то...
- А влияет ли исполнение этой инструкции на вот эту?
 - Ключевой вопрос – взаимовлияние!

Наблюдаемая упорядоченность

- Результат действия одних инструкций по отношению к состоянию других инструкций
- Даже если мы можем «измерить» все оборудование, не поможет «знать точно состояние ячейки»!
- Скорее, не просто «может быть так», а «так является»
- Команды «наблюдения» влияют на наблюдаемые
 - Совсем как в квантовой механике
 - Процесс измерения меняет измеряемую величину?
 - Измеряемое явление не существует до момента измерения!

Наше время

- Последовательность операций внутри одного потока как последовательность переданных на исполнение CPU команд
- Общее время
 - все потоки разделяют одно и тоже понятие времени
- В дальнейшем понятие времени будет формализовано

Временные отметки (TS)

- Один из важных приемов – используется во множестве алгоритмов
- «в какой то степени уникальное» значение счетчика
- «монотонно» растёт
 - Пример: время, прошедшее с какого то момента

Временные отметки

- `TS_instance CreateTS();`
Создает экземпляр временной метки
- `TS_value GetTSvalue(TS_instance);`
получает новое значение временной метки (и, обычно, инкрементирует ее экземпляр)
- `int CompareTSvalues(TS_value, TS_value);`
сравнивает два значения
- Изменение внутреннего счетчика
 - По запросу
 - Постоянно меняющееся значение (например, RDTSC)
 - Обычно целое слово (4 байта)

Временные отметки

- Пример – счетчик в алгоритме булочной
- Проблемы
 - Переполнение (Y2K – проблема 2000 года)
 - Точность
 - Калибровка (RDTSC)
 - Гарантии уникальности

Ограниченные отметки

- Как избежать проблем переполнения?
 - получить такие значения счетчиков, которые можно сравнивать всегда
- Есть сложные реализации, с гарантией
- Пример – реализация с ограничением схемы использования
 - Сканирование – чтение отметки другого потока
 - Самопометка – выставка себе большей метки
 - Только последовательно (упорядоченно)

Ограниченные отметки

- ассиметричный упорядоченный граф (предшествования) для представления
- Ребро от a к b означает что a позднее
- Нет транзитивности ($a \rightarrow b + b \rightarrow c \neq a \rightarrow c$)
- Метка потока «переползает» по ребрам

Ограниченные отметки

G есть граф предшествования, и A и B есть его подграфы

Определение: A доминирует над B в графе G , если каждая вершина графа A имеет ребра, идущие к каждой вершине графа B .

Определение: Операция размножения графа G через H ($G \circ H$) есть некоммутативный оператор следующего вида:

Заменим каждую вершину v графа G копией графа H (обозначаемой далее Hv), причем Hv доминирует над Hu в $G \circ H$, если v доминирует над u в G .

Определение: граф предшествования для k потоков T^k есть:

T^1 – единственная вершина;

T^2 – определен выше;

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

Используя предложенный граф T^k , можно всегда найти новое место для знака потока, который будет доминировать надо всеми остальными потоками

Вероятности и ошибки

- Можно ли пренебречь маловероятным событием?
 - неатомарная инкрементация $i++$ в 2 потоках
 - Мала вероятность совпадения?
 - На 1М ошибка ~ 10 раз, несколько раз в секунду!

Вероятности и ошибки

- Другой пример – сравнение страниц (4кб)
 - Можно сравнивать побайтово (а если их много?)
 - Можно сгенерировать хеш и сравнить его
 - Вторым случаем имеет «встроенную» ошибку
 - Для 64 бит хеша с учетом парадокса дня рождения надо сделать 6 сравнений (10^5 байт) для достижения вероятности совпадения 10^{-18} , т.е. 1 отказ на 10^{23} байт
 - Первый – тоже! (ЕСС память)
 - 1 отказ на 10^{20} байт, хотя, возможно, это разные ошибки

Выводы

- Программист использует понятные ему абстракции (понятия) в рамках модели исполнения
- Типовые понятия (кроме обычных для языка) включают
 - Корректность программы
 - Время и процесс выполнения кода
 - Наблюдаемый порядок выполнения инструкций, отношения между ними
 - Примитивы синхронизации
- Для «прямой» работы со временем используются временные отметки
 - Они часто используются для организации параллельных алгоритмов и имеют специфику реализации
- Типовая модель исполнения для параллельного программирования является продолжением последовательной
 - Чаще всего используется модель «автономного исполнения участков кода»
- При выборе алгоритмов, в том числе параллельных, необходимо учитывать вероятности и ошибки

(с) А. Тормасов, 2010-11 г.

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

Для коммерческого использования курса просьба связаться с автором.