ТЕОРИЯ И ПРАКТИКА МНОГОПОТОЧНОГО ПРОГРАММИРОВАНИЯ

Тема 6

Проблемы и специфика параллельного программирования

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

Тема

- Общие проблемы параллельного программирования.
- Проблемы, возникающие при работе с разделяемой памятью.
- Разделяемые объекты и синхронизация.
  - Синхронизация выполнения кода и синхронизация обращения к данным.
  - Явная и неявная синхронизация.
  - Синхронизация путем обеспечения условий неизменности.
- Ожидание

Общие проблемы

- Синхронизация
- Коммуникация
- Балансировка нагрузки
- Масштабируемость

Разделяемая память

- Разные проблемы для тесно (Tightly-coupled) и слабо (Loosely-coupled) связанных систем
- Будем рассматривать только тесносвязанные системы, почти всегда SMP
- Типовые проблемы
  - Состояние «гонки» (race condition)
  - Взаимная блокировка (deadlock)
  - Плохая масштабируемость изза конфликтах на блокировках (lock contention)
  - Активная блокировка (live lock) – «имитация бурной деятельности»

Разделяемые объекты и синхронизация

- метод сериализации выполнения операций
  - гарантирование того, что в один момент времени с разделяемыми данными будет работать только один поток
  - Как сведение к уже известной задаче

Организация синхронизации

- Явная синхронизация
  - Через вызовы АПИ примитивов (их много разных типов!)
    - Синхронизация выполнения кода (плохо)
    - Синхронизация обращения к данным (стандарт)
- Классическая проблема: взаимоблокировка при попытке захвата «по очереди» 2 ресурсов A и B
  - Поток 1: успешно захватил ресурс A и ждет освобождения B
  - Поток 2: успешно захватил ресурс B и ждет освобождения A
  - Они будут ждать вечно!

Неявная синхронизация

- Можно создавать работающие программы без явных обращений к синхронизационному API
- Синхронизация путем обеспечения условий неизменности
  - Сделаем так, чтобы ошибка была невозможна!
    - Если проблема в прерываниях, запретим их через вызовы CLI/STI или через APIC TPR
    - В результате код не будет прерываться и критическая секция будет исполняться строго в одиночестве
      - Тем не менее, “`a = 0; a = 1/a;` ” в коде и…

Неявная синхронизация

- Можно использовать чтото «гарантировано безопасное», например, interlocked операции
  - Увы, они не гарантируют правильности алгоритма!
- Пример: «атомарный перевод» денег с a на b

```
a = a-5;     // даже использвание атомарного сложения
b = b+5;     // не гарантирует общую корректность!
```

Проблемы синхронизации

- ABA - проблема алгоритмов с неявной блокировкой (вернее, неблокирующих алгоритмов)
  - Использование адреса полученной для хранения содержания объекта области памяти для их «различения» - типичный пример
  - Жизненный цикл экземпляра структуры:
    - malloc(); <заполнили>; <используем>; free();
    - Следующий malloc() может вернуть ТУ ЖЕ память

Пример с ошибкой ABA

```
// Реализация неблокирующего стека с ошибкой ABA
class Stack
{
   volatile Obj* top_ptr;
   // достаем и возвращаем верхний объект
   Obj* Pop()
   {
      while(1)
      {
         Obj* ret_ptr = top_ptr;
         if (!ret_ptr) return NULL;
         Obj* next_ptr = ret_ptr->next;
         // если верхушка стека равна ret_ptr,
         // то никто не изменил стек
         // ! Это не всегда правда изза ABA !
         // Атомарно заменяем верхушку стека
         // на следующий элемент
         if (CompareAndSwap(
            top_ptr, ret_ptr, next_ptr))
               return ret_ptr;
         // стек изменился, начнем сначала
      }
   }

   // добавляем объект obj_ptr в стек
   void Push(Obj* obj_ptr)
   {
      while(1)
      {
         Obj* next_ptr = top_ptr;
         obj->next = next_ptr;
         // если верхушка стека равна ret_ptr,
         // то никто не изменил стек
         // ! Это не всегда правда изза ABA !
         // Атомарно заменяем верхушку стека
         // на входной элемент
         if (CompareAndSwap(
               top_ptr, next_ptr, obj_ptr))
               return;
         // стек изменился, начнем сначала
      }
   }
};
```

Пример с ошибкой ABA

Пусть в стеке хранится последовательность A B C

Поток 1

```
POP - ret_ptr == A, next_ptr == B
Дошли до CAS
«заморозились»
…
…
«разморозились» и ret_ptr == top_ptr
CAS выполнился, но top_ptr = B (!)
```

Поток 2

```
Pop (получили A)
Pop (получили B)
Удалили B
Push A top_ptr снова равен A
```

Решение: «склейка» номера версии с указателем (TS)

Ожидание

- Что делает программа, когда ей ничего нельзя делать?
  - Пользовательская программа: sleep()
    - ОС есть чем заняться – переключится на другую задачу (ядра или пользователя)
    - Когда ресурс освободится, «нас» пробудят wakeup()
  - Ядро ОС: spin (примитив spinlock)
    - ОС вынуждена ждать освобождения ресурса

Производительность ожидания

- Даже в программе пользователя вызовется как минимум пара sleep() – wakeup()
  - Два переключения контекста, тысячи тактов
- Если ожидаем, что время ожидания мало, то, возможно, стоит подождать в цикле какое то время, и только потом уже делать sleep()
  - Обязательно оценить вероятность коллизий

Выводы

- Для обеспечения работы традиционных последовательных алгоритмов в условиях соисполнения требуется принимать какие то меры – обеспечивать синхронизацию
- Синхронизация бывает явной и неявной
- Все алгоритмы синхронизации сложны и подвержены проблемам (из-за взаимовлияния команд друг на друга)
  - Просто ошибкам алгоритма изза последовательности доступа к данным
  - Ошибкам типа взаимоблокировок
  - Нетривиальным ошибкам типа ABA
  - Проблемам производительности, связанным с масштабированием задач
- В процессе синхронизации часто приходится ожидать освобождения ресурса
  - Это может приводить к сущетсвенным потерям производительности программ

(c) А. Тормасов, 2010-11 г.

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