ТЕОРИЯ И ПРАКТИКА МНОГОПОТОЧНОГО ПРОГРАММИРОВАНИЯ

Тема 4

Языки программирования и многопоточность, конкурентный доступ

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

Тема

- Языки программирования и работа порожденного кода в условиях современных многопоточных систем с разделяемой памятью
- Конкурентный доступ и другие проблемы параллельного программирования - вина оборудования?
- Модель программирования и синхронизации, встроенная в язык (Java, C,C#)
- Потоки, «безопасность для исполнения в параллельных потоках», локальные объекты потока и др

Как мы думаем?

- Программист думает в терминах «операторов», обычно умозрительно выполняя их дискретно и последовательно
- С его точки зрения, каждый оператор «почти атомарен» (кроме: препроцессора, выражений со скобками, логических выражений и тд)
  - Можем ли мы выполнить половину оператора a = 1.0?
  - Будет ли внутри a в этот момент 0.5???
- Надо понимать границы атомарности. В том числе вызванные процессом трансляции операторов в инструкции платформы.
- Четкой грани может и не быть!

Трансляция C ➔ asm

Исходный текст

```
void myfunc(void)
{
   int a,b,c;
   a=1;
   b=f(a);
   c=b;
   b++;
}
```

Код для x86

```
                                        ; пролог
   …
   mov      DWORD PTR _a$[ebp], 1   ; a=1
   mov      eax, DWORD PTR _a$[ebp] ; b=f(a)
   push     eax
   call     ?f@@YAHH@Z
   add      esp, 4
   mov      DWORD PTR _b$[ebp], eax ; c = b
   mov      ecx, DWORD PTR _b$[ebp]
   mov      DWORD PTR _c$[ebp], ecx
   mov      edx, DWORD PTR _b$[ebp] ; b++
   add      edx, 1
   mov      DWORD PTR _b$[ebp], edx
   …                                    ; эпилог
; оптимизированный код для той же функции
   push     1    ; а где же операторы C?
   call     ?f@@YAHH@Z
```

Актуальность значений

- Переменная ассоциирована с ячейкой памяти
- Операции проводятся чащще всего в регистре – считали, изменили, записали
- С переменной ассоциировано несколько значений – ячейка, регистр и тд
- Указатель «умеет» указывать только в память!

Особенности ЯВУ

- каждый оператор языка высокого уровня может быть протранслирован в несколько инструкций аппаратной платформы;
- смысловая последовательность инструкций может не соответствовать последовательности операторов языка (компилятор может переставлять смысловые действия из соображений оптимизации исполнения);
- одна инструкция в коде может не иметь “прямого соответствия” какому либо оператору языка, из которых она была получена;
- часто в последовательности инструкций оптимизированного кода нельзя найти границ операторов языка;
- одно и то же понятие может одновременно иметь разные представления, из которых только одно является “актуальным”, и для его использования необходимы дополнительные действия
- Для managed языков ситуация еще более запутана, нет прямого соответствия, и представление может меняться во времени исполнения программы (один оператор в разные моменты времени будет протранслирован по разному).

Условия гонки

- Они же – конкуретный доступ к памяти из нескольких мест (потоков)
- Разница в семантике языка (единообразие значения переменной) и в его реализации (существование стадий операций, несколько представителей одной переменной с разной актуальностью)

«атомарный» a++

Поток 1

```
reg <- [a]    // загрузить содержание ячейки a
// из памяти (или кэша) в регистр
reg <- reg+1  // инкрементировать регистр
[a] <- reg    // записать значение в память
```

.....

```
reg <- [a]    // загрузить содержание ячейки
reg <- reg+1  // инкрементировать регистр
[a] <- reg    // записать значение в память
```

.....

```
reg <- [a]    // загрузить содержание ячейки
reg <- reg+1  // инкрементировать регистр
[a] <- reg    // записать значение в память
```

Поток 2

```
reg <- [a]    // загрузить содержание ячейки a
// из памяти (или кэша) в регистр
reg <- reg+1  // инкрементировать регистр
[a] <- reg    // записать значение в память
```

.....

```
reg <- [a]    // загрузить содержание ячейки
reg <- reg+1  // инкрементировать регистр
[a] <- reg    // записать значение в память
```

.....

```
reg <- [a]    // загрузить содержание ячейки
reg <- reg+1  // инкрементировать регистр
[a] <- reg    // записать значение в память
```

Модели языков и систем

- Есть ли в языке предположения или гарантии относительно параллельного исполнения или порядка выполнения операторов?
- Модель, встроенная в язык программирования
- Модель, встроенная в библиотеку времени выполнения
- Модель, встроенная в среду языка

Модели языков типа C

- C, C++, ObjectiveC,
  - новый стандарт C++x – исключение
- Изначально языки этой группы разрабатывались для UP окружения и произвольной платформы
- В стандартах языков нет НИКАКИХ предположений про порядок, место расположения в памяти-регистре и тд, кроме собственно упорядоченности операторов

Особенности C...

- локальные переменные располагаются в стеке, который уникален для каждого потока исполнения
- компилятор имеет право предположить, что никто кроме исполняемого сейчас кода не будет доступаться к данным
- ключ volatile, который означает, что данные надо читать и писать из памяти при каждом к ним обращении
  - Ключ volatile введен для работы с драйверами и «искуственной» памятью, доступ к которой означает IO операции (например, APIC) и необходима была гарантия обращения к памяти.
  - Ключ volatile не имеет отношения к синхронизации

Особенности C...

- Атрибут глобальной переменной TLS (Thread Local Storage) - означает независимую копию для каждого потока (пример: errno)
- Безопасен:
  - доступ к явно блокированным участкам кода
  - доступ к неразделяемым на запись переменным
  - некоторые “встроенные” (intrinsic) примитивы
  - Некоторые RTL функции (thread-safe printf)

Особенности JAVA

- Двустадийная компиляция – байткод + JVM
- Встроенная в JVM модель памяти(эволюционировала, в Java 5.0 2004)
- Обсуждает понятия атомарности, видимости и упорядоченности
- Внутри одного потока: *within-thread as-if-serial*
- Частичный порядок между read/write, lock/unlock, start/join threads - happens-before (происходит раньше)
  - Если X раньше Y, то результат X видим для Y
  - Но между потоками нет гарантии видимости в этом порядке!

Особенности JAVA

- Вводится новое понятие – монитора (для каждого объекта, по умолчанию неявный)
- Используется для взаимного исключения кода
- Synchronized включает его – только один поток может вызвать метод для экземпляра
- Более детально информация – в документации Java

Особенности C#

- Двустадийная компиляция – байткод + CLR
- Встроенная модель памяти, аналогична monitor
- Только явное описание – спец объект, который «можно ожидать»
- Защищенные участки кода lock (в стиле try/catch C++)

Выводы

- Те понятия, которыми манипулирует программист, имеют сложное «переложение» в особенности аппаратной платформы, как минимум, в нетривиальные наборы инструкций
- Конкурентный доступ является следствием организации компьютера и несоответствия понятий, вкладываемых программистом, с их реализацией – как из за последовательности инструкций, так и из за атомарности, видимости и переупорядочения операций работы с памятью
- Некоторые языки имеют понятие о параллельном выполнении и модели памяти, встроенное в язык (типа JAVA), некоторые (типа C) – нет
- Некоторые особенности реализации языков и RTL облегчают создание корректно работающих в параллельном окружении программ (модель памяти, доступ к примитивам синхронизации, локальная память типа стека, потоко-безопасные библиотеки и тд)
- Несмотря на наличие «модели памяти» языка и остальных особенностей, проблемы корректного исполнения кода в параллельном окружении лежат на программисте (не только синхронизация)

(c) А. Тормасов, 2010-11 г.

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

(c) А. Тормасов, 2010-11 г.

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