Лекция 5. Языки программирования и многопоточность. Время. Конкурентный доступ.

Содержание

- CAS: наиболее мощная операция.
- Многопоточность в языках программирования.
- Время в системах с параллельным исполнением.

Операция CMPXCHG

- Самая мощная и полезная команда.
- Первая изучаемая нами команда для организации многопоточности

```
GCC:
type __sync_val_compare_and_swap(
  type *ptr, type oldval, type newval);

Microsoft:
LONG InterlockedCompareExchange(
  LONG volatile *ptr,  LONG newval,
  LONG  oldval
);
```

Обратите внимание на разный порядок аргументов!

Сементика CMPXCHG

Семантически это исполняется атомарно:

```
type CMPXCHG(type * ptr, type oldval, type newval) {
  type retval = *ptr;
  if (*ptr == oldval)
    *ptr = newval;
  return retval;
}
```

Использование CMPXCHG

Пусть CAS = __sync_val_compare_and_swap

```
#include <pthread.h>
#include <X86intrin.h>
void *func_add(void *argum) {
  arg *a = (arg *)argum;
  const unsigned long ME = 1;
  for (int i = 0; i < 10000000; i++) {
    while (CAS(&a->num, -1, ME) != -1)
      ;
    a->p[0]++;
    a->num = -1;
  }
  return NULL;
}
```

```
pushq   %rbp
movq    %rsp, %rbp
leaq    8(%rdi), %rcx
xorl    %edx, %edx
movl    $1, %esi
LBB0_1:
  movq    $-1, %rax
  lock    cmpxchgq %rsi,(%rcx)
  jne     LBB0_1
  movq    (%rdi), %rax
  incq    (%rax)
  movq    $-1, 8(%rdi)
  incl    %edx
  cmpl    $10000000, %edx
  jne     LBB0_1
xorl    %eax, %eax
popq    %rbp
retq
```

Резюме по архитектуре и работе с памятью

- Операции чтения могут производиться в любом порядке.
- Операции чтения могут быть умозрительными.
- Запись может быть буферизирована.
- Операции чтения могут закончиться позже операций буферизованной записи.
- Операции записи стараются производиться в порядке их появления в программе.
- Для реально исполненных инструкций запись не умозрительна.
- Чтение может производиться из буфера записи.
- Операция `lock` формирует барьер чтения и записи.
- Операции `mfence`, `lfence`, `sfence` формируют барьеры.

Многопоточность в языках программирования.

Модели параллельного исполнения

Кто предоставляет нам возможности параллельного исполнении и гарантии о порядке исполнения операций и взаимодействии?

- Модель, встроенная в язык программирования. `Java`. `C#`. `C++11`, `GO`, `Rust`
- Модель, встроенная в библиотеку времени исполнения.
- Модель, встроенная в окружение языка.

Особенности C/C++ как языка параллельного исполнения

- Использование стека для локальных переменных. Каждый поток может иметь свою песочницу.
- Работа с указателями. Компилятор по умолчанию не предполагает, что указатели ссылаются на объект, принадлежащий другому потоку и вольны использовать их по своему усмотрению.
- Ключевое слово `volatile`, запрещающее оптимизацию операций с памятью и явное требующее обращения к памяти. `volatile` не есть операция синхронизации и не вносит атомарность в операции!

Особенности C/C++ как языка параллельного исполнения

- В окружении языка встраивается модель атрибутов `Thread Local Storage` для глобальных переменных.
- Считаются потокобезопасными:
  - доступ к переменным только для чтения;
  - некоторые встроенные и библиотечные функции;
  - доступ к общим переменным после явной блокировки.

Особенности Java как языка параллельного программирования

- Двухстадийная компиляция:
  - модель: `Java byte code` + `Java Virtual Machine`.
  - реально: `Java byte code` + `Java Virtual Machine` + `Just In Time`.
- Явно определяются понятия атомарности, видимости и упорядоченности.
- Внутри одного потока определено понятие последовательного исполнения.
- Определена частичная упорядоченность `happens before` (если `X` происходит раньше `Y`, то результаты `X` видимы `Y`) между парами
  - `read memory/write memory`
  - `lock/unlock`
  - `start thread/join thread`
- Между потоками такой упорядоченности нет.

Особенности Java как языка параллельного программирования

- Для каждого объекта последовательно проводится концепция монитора (`synchronized`)
- Над одним объектом можно провести ровно одну операцию из разных потоков в один момент времени.
- События операций над объектами управляются самими объектами.

Особенности C# как языка параллельного программирования

- Двухстадийная компиляция:
  - модель: `byte code` + `Common Language Runtime`
  - реально: `byte code` + `Common Language Runtime` + `Just In Time`.
- Встроенная модель памяти.
- Синхронизация через специальные объекты.
- Модель `lock`/`unlock` для защищаемых участков кода.

Особенности GO как языка параллельного программирования

- Компиляция в естественный код.
- Модель памяти отображается на аппаратную.
- Сопрограммы `goroutines` есть средства языка. Запуск любой функции параллельно: `go func(args)`.
- Синхронизация через специальные объекты.
- Синхронизация через каналы.
- Канал — разделяемая структура данных, имеющая методы посылки сообщения и приёма сообщения.
- Один канал может использоваться несколькими сопрограммами.

Особенности `Rust` как языка параллельного программирования

- Компиляция в естественный код.
- Модель памяти отображается на аппаратную.
- Принцип владения `borrowing` защищает общие объекты.
- Использование `move`-замыканий.
- `spawn/join`: запуск потоков без разделяемых ресурсов.
- Каждый поток имеет свой стек и отображатся на потоки операционной системы.
- Втроенные примитивы счётчиков ссылок.
- Параллельные итераторы.
- Каналы для общения потоков в режиме `MPSC` используют очереди без блокировок.
- Как запасные варианты: `Mutex`, `RWLock`, `CondVar`, `atomic`....

Общие принципы создания потоков

- Поток — единица планирования современных ОС.
- Создание потока — результат соответствующего системного вызова.
- При создании потока:
  1. Создаётся запись в ядре — системный контекст потока.
  2. В текущем процессе выделяется стек необходимого размера (изменяются таблицы страниц).
  3. Создаётся запись в таблице существующих потоков.
  4. Инициализируется пользовательский контекст потока, заполняются регистры стека, фрейма и указатель на переданные аргументы.
  5. Поток помечается готовым к исполнению и передаётся планировщику.

Варианты создания потоков

- С точки зрения языка программирования — поток есть функция, исполняющаяся параллельно с другими функциями.
- Для создания потока требуется предъявить функцию, которая будет исполняться как *функция потока*.
- Завершение *функции потока* приводит к завершению самого потока.
- Адресное пространство потока становится недоступным (в `pthread` поведение немного другое).

Создание потоков в различных ОС

- Windows:

```
DWORD WINAPI ThreadFunc(void *arg);
th = CreateThread(NULL, 0, thread, &args, flags, &tid);
```

- pthreads:

```
void *thread_func(void *arg);
code = pthread_create(&ptid, NULL, thread_func, &args);
```

- C++11:

```
AnyType func(type1 arg1, type2 arg2...);
thread t = thread(func, param1, param2);
```

Синхронизация с завершением потока

- Windows:

```
err = WaitForSingleObject(th, INFINITE);
```

- pthreads:

```
code = pthread_join(ptid, &retaddr);
```

- C++11:

```
t.join();
```

Особенности реализации потоков на C++11

- Функции потока не выделяются явным образом, в их качестве может использоваться любая функция.
- Компилятор создаёт контекст функции потока в виде невидимой структуры, в которую копирует аргументы.
- Компилятор создаёт невидимую функцию (*proxy*, которая и является реальной функцией потока с точки зрения ОС).

Время в системах с параллельным исполнением

Время в системах с параллельным исполнением

- Что такое время в параллельных системах?
- Время — это показания часов?
- Время — это количество секунд от 1 января 1970 года?
- Время — это количество тиков процессора от его запуска?
- Что есть высказывание: время события `A` равно времени события `B`?

Время в системах с параллельным исполнением

- Что такое время в параллельных системах?
- Время — это показания часов?
- Время — это количество секунд от 1 января 1970 года?
- Время — это количество тиков процессора от его запуска?
- Что есть высказывание: время события `A` равно времени события `B`?
- Ответ: для параллельных процессов понятия одновременности не существует.
- В некоторых случаях существует понятие «событие `B` произошло *после* события `A`».

Время в системах с параллельным исполнением

- Что для нас событие «окончание инструкции процессора»?
- Связано ли оно с окончанием какой-либо фазы?
- Имеются следующие фазы:
  - Фаза исполнения команды.
  - Фаза записи результата в кэш.
  - Фаза записи из кэша в память.

Время в системах с параллельным исполнением

- Что для нас событие «окончание инструкции процессора»?
- Связано ли оно с окончанием какой-либо фазы?
- Имеются следующие фазы:
  - Фаза исполнения команды.
  - Фаза записи результата в кэш.
  - Фаза записи из кэша в память.
- Событие интересно только с момента видимости результата другим процессором.

Временны́е отметки

- Два процессора исполняют каждый свой поток команд.
- Для каждого процессора происходит ряд событий.
- Что можно сказать об их взаимосвязи?
- Физическое время непрерывно.
- Компьютерное время дискретно.
- Вводится понятие временны́х отметок.
- Они монотонно возрастают на каждом из процессоров.
- Желательно, чтобы каждое событие имело уникальную метку.
- Изменение меток может происходить независимо от процессора либо по требованию процессора.

Временны́е отметки (идеальные)

- Определим отношение упорядоченности (операции $>$ и $<$)
- Для каждого процессора k временна́я метка результата исполнения каждой последующей инструкции больше, чем предыдущей (монотонность внутри процессорного ядра)

$$T(k, A_i) < T(k, A_j), \text{если } i < j,$$

где i, j — порядковые номера команд процессора k.

- Для разных процессоров временная метка инструкции, получающей результат исполнения инструкции другого процессора, больше временно́й метки команды, вызвавшей появление результата.

$$T(k_1, A) < T(k_2, B),$$

если команда B процессора k_2 использует результат команды A процессора k_1.

Свойства временны́х отметок (идеальные)

- Два процессора исполняют каждый свой поток команд.
- Для каждого процессора происходит ряд событий.
- Что можно сказать об их взаимосвязи?
- Физическое время непрерывно.
- Компьютерное время дискретно.
- Временны́е метки монотонно возрастают на каждом из процессоров.
- Желательно, чтобы каждое событие имело уникальную метку.
- Изменение меток может происходить независимо от процессора либо по требованию процессора.
- Имеется команда процессора `RDTSC`

Отношение и порядок

Последовательность записей в память может меняться процессором.

- Какой ответ на вопрос: одновременно ли исполняются эти две инструкции?
- Попутное замечание: исполнение команды умозрительно.
- Ещё замечание: как определить, завершилась ли команда?
- Ещё замечание: как влияет исполнение этой команды на исполнение той?
- Эврика! Взаимовлияние определяет всё.

Наблюдаемая упорядоченность

- Упорядоченность - результат действия одних инструкций по отношению к состоянию других инструкций
- Команды наблюдения влияют на наблюдателя.
- Измеряемое явление не существует до момента измерения.

Вероятность и ошибки

Какова вероятность, что следующий код будет прерван между инструкциями?

```
mov %rax, 4(%rbp)
inc %rax
mov 4(%rbp), %rax
```

Вероятность и ошибки

Какова вероятность, что следующий код будет прерван между инструкциями?

```
mov %rax, 4(%rbp)
inc %rax
mov 4(%rbp), %rax
```

- Процессор прерывает исполнение потока по таймеру пусть 100 раз в секунду.
- Пусть за секунду исполняется 10^{10} инструкций. Тогда вероятность прерывания программы между первыми двумя инструкциями равна примерно $\frac{100}{10^{10}} = 10^{-8}$
- При темпе исполнения 10^{10} инструкций в секунду за секунду в среднем код будет прерван 100 раз!

Вероятность и ошибки

- При обращении к оперативной памяти могут возникать ошибки.
- Какова вероятность ошибки при сравнении двух страниц памяти в 4KB?
- Хорошая память имеет частоту ошибок 1 отказ на 10^{20} байт.
- Сравнение двух блоков даст вероятность ошибки в $\frac{4096 \times 2}{10^{20}} \approx 10^{-15}$
- Сравнение значений двух хеш-функций длиной 64 бита даст вероятность $\approx 10^{-18}$.

Спасибо за внимание.

Следующая тема — Проблемы многопоточного программирования.