Лекция 10. Подходы к синхронизации.

Содержание

- Задача читателей-писателей
- Замки
- Подходы к синхронизации

Задача читателей-писателя

Задача читателей-писателей

- Имеется область памяти, к которой обращаются несколько потоков.
- Пусть потоков на чтение будет произвольное количество, а потоков на запись — один.
- Возможные решения:
  1. Если есть хотя бы один читатель, писатель должен подождать. Резон: если читатель уже обратился за данными, он должен получить их для момента обращения.
  2. Если есть писатель, то читатели должны подождать. Резон: если читатель обратился за данными, он должен получить свежую копию.
- Требование: независимо от порядка вызовов и выбранной стратегии все потоки должны завершить запрос за конечное время.
- **Критическая секция обязательна!**

Приоритет читателя

```
mutex resource, crit;
int readers = 0;
```

```
thread_reader() {
    crit.lock();
    if (++readers == 1)
        resource.lock();
    crit.unlock();
    @@ critical section
    crit.lock();
    if (--readers == 0)
        resource.unlock();
    crit.unlock();
}
```

```
thread_writer() {
    resource.lock();
    @@ critical section
    resource.unlock();
}
```

Приоритет писателя

```
mutex rd, wr, resource, rdentry, rdtry;
int readers, writers = 0;
```

```
thread_reader() {
   rdentry.lock();
    rdtry.lock();
     rd.lock();
       if (++readers == 1)
         resource.lock();
     rd.unlock();
    rdtry.unlock();
   rdentry.unlock();
   @@critical section
   rd.lock();
    if (--readers == 0)
      resource.unlock();
   rd.unlock();
}
```

```
thread_writer() {
   wr.lock();
     if (++writers == 1)
       rdtry.lock();
   wr.unlock();
   resource.lock();
   @@critical section
   resource.release();
   wr.lock();
   if (--writers == 0)
      rdtry.unlock();
   wr.unlock();
}
```

Сбалансированное решение

```
mutex rd, wr, rdcount;
int readers = 0;
```

```
thread_reader() {
   rd.lock();
    rdcount.lock();
      if (++readers == 1)
        wr.lock();
    rdcount.unlock();
   rd.unlock();
   @@critical section
   rdcount.lock();
     if (--readers == 0)
       wr.unlock();
   rdcount.unlock();
}
```

```
thread_writer() {
   rd.lock();
     wr.lock();
       @@critical section
     wr.unlock();
   rd.unlock();
}
```

Проблемы RW-locks

- Приоритет читателя. Пока имеется поток читателей, писатель не может изменить данные (writer starvation). Алгоритм несправедлив для писателя.
- Приоритет писателя. Reader starvation.
- Сбалансированный алгоритм.
  - pthread_rwlock_t
  - ReadWriteLock (java)
  - System.Threading.ReadWriterLockSlim (.NET)

Множественные писатели.

seqlock

```
thread_reader() {
  do {
    while((c=lock->cnt)&1)
      ;
    @@ reading
  } while(lock->cnt!=c);
}
```

```
thread_writer() {
  do {
    while((c=lock->cnt)&1)
      ;
  } while(!CAS(&lock->cnt,c,c+1);
  @@ writing
  atomic_increment(&lock->cnt);
}
```

seqlock: особенности

- Применяется в ядре Linux для операций изменения таймера (много читателей — единичные писатели).
- Сложен в отладке.

Read-Copy-Update

- Хорош для операций с преобладающим количеством операций чтения.
- Основной принцип: если поток захотел изменить данных, он изменяет копию и в удобное для всех время вставляет её.
- *CS* — *Critical Section* R/W
- QS - quiescent state потока — поток не находится в критической секции.
- GP - grace period объекта — все потоки побывали в состоянии QS.
- Каждая CSR вход в которую произошёл до GP должна завершиться прежде завершения GS.
- Каждый GP конечен так как каждая CSR конечна.

Read-Copy-Update: поток писатель

- **Removal phase**: изменяет структуру данных без удаления элемента.
- **Grace period start phase**: объявляет о начале GP (rcu_synchronize)
- **Grace period end waiting phase**: ожидание окончание GP.
- **Reclamation phase**: Коммитит изменения.

Писатель может быть только один. Несколько писателей — использование mutex.

Read-Copy-Update: операции

- `rcu_read_lock`. После этого писатель не имеет права изменять данные.
- `rcu_read_unlock`. Могут быть вложенными и создавать пересекающиеся области.
- `rcu_syncronize`. Аналог барьера. Дожидается завершения всех CS.
- `rcu_assign_pointer`. Присвоить новое значение.
- `rcu_dereference`. Удалить объект.

Read-Copy-Update: проблемы

- Легко блокировать. Трудно понять, когда закончился GP и что после этого делать. Ожидать? Как? mutex?
- Один из выходов — callback вызов в момент завершения GP.
- Прекрасно реализуется в ядре: просто управление передаётся процессору с ожидающим писателем.
- Для RCU user space всё гораздо сложнее.

User space RCU

- Для всех потоков заводится глобальная переменная, содержащая идентификатор `GracePeriod`.
- Имеется связный список переменных с идентификатором `GP` каждого потока и его глубиной вложенности `CS`.
- Перед освобождением памяти писатель проверяет этот список.
- Используются только операции `CAS`.
- Общий объём кода около 400 строк.

Подходы с синхронизации

Примитивы синхронизации и их особенности

- Повторно-входимый замок. `pthreads reenterant mutex`. `C++11 reenterant_mutex`. Содержит счётчик входов.
- Семафор. Ограничитель числа потоков в критической секции. Не более чем: ядер, файлов, сокетов, операций ввода/вывода, лицензий.
- Монитор. Способен либо захватить замок, либо дождаться события. Склонен к проблеме «`lost signal`».

Синхронизация и структуры данных

- Простой подход: имеется структура данных и она реализуется в виде монитора.
- Каждый метод исполняется внутри критической секции (окружается критической секцией).
- Хорошо: лёгкая отладка. Лёгкая разработка. C++ — `auto_releaser` (невозможно забыть разблокировать)
- Плохо: грубая блокировка **(coarse-graining)**. В критическую секцию попадает больше кода, чем требуется. Ожидание разблокировки длиннее, чем могло бы быть. Плохая масштабируемость.

Виды синхронизации в структурах данных

- **Fine-grained**: блокируются независимые подобъекты.
- **Optimistic**: замки не используются. Проверяется, не изменился ли объект. Не годится для нагруженных объектов.
- **Lazy**: сложные операции откладываются на более удобное время. `Deferred Procedure Call` в `Windows`.
- **Lock/wait/obstruction free**: замков нет, есть `CAS` и немного оптимизма.

Спасибо за внимание.

Следующая тема — неблокирующие алгоритмы.