Лекция 8: Семафоры и мониторы

Алексей Линёв
Александр Мощук
Кирилл Погорельский

somes slides are adapted from the OS course at the University of Washington

Семафоры

- Семафор – примитив синхронизации
  - более высокого уровня абстракции, чем признаки блокировки
  - предложены Дийкстрой (Dijkstra) в 1968 г. в качестве компонента операционной системы THE
- Семафор – это
  - переменная, для которой определены 2 операции: P и V (сокращения для “wait” и “signal” в голландском языке)
    - **P(sem)** (wait/down)
      - ожидает до выполнения условия sem > 0, затем уменьшает sem на 1 и возвращает управление
    - **V(sem)** (signal/up)
      - увеличивает sem на 1
- Операции P и V выполняются *атомарно*

Как работают семафоры?

- С каждым семафором ассоциирована очередь потоков
  - при вызове потоком **P(sem)**
    - если семафор «свободен» (>0), его значение уменьшается на 1, и выполнение потока продолжается
    - если семафор «занят» (<=0), поток помещается в очередь, соответствующую данному семафору; запускается какой-либо другой поток
  - при вызове потоком **V(sem)**
    - если очередь потоков, ассоциированная с данным семафором, не пуста – один из них разблокируется
      - и помещается в очередь готовых к выполнению
      - поток, вызвавший V (sem) продолжает свое выполнение
    - в противном случае (нет потоков, ожидающих освобождения семафора), значение семафора увеличивается
      - то есть информация о прошедшем вызове будет запомнена и востребована при следующем вызове P(sem)
- Таким образом, у семафоров есть история

Реализация семафоров

```
type semaphore = record
    value: integer:
    L: list of threads;
end

P(S):
    S.value = S.value - 1;
    if S.value < 0
    then begin
            add this thread to S.L;
            block;
    end;

V(S):
    S.value = S.value + 1;
    if S.value <= 0
    then begin
            remove a thread T from S.L;
            wakeup T
    end;
```

P ()/V() – критические секции, должны выполняться атомарно

Два вида семафоров

- Двоичный семафор (мьютекс, mutex)
  - S.value инициализируется значением 1
  - обеспечивает эксклюзивный доступ к ресурсу (например, при работе в критической секции)
  - одновременно может выполняться только один поток/процесс
- Счетный семафор
  - S.value инициализируется значением N
    - N = число доступных единиц ресурса
  - представляет ресурсы, состоящие из нескольких однородных элементов
  - позволяет потокам исполняться пока есть неиспользуемые элементы

Использование

- С точки зрения программиста, операции P и V над двоичным семафором семантически эквивалентны операциям Acquire и Release над признаками блокировки

```
P(sem)
  :
  :
выполнить все действия, требующие взаимного исключения; здесь
может быть достаточно большой кусок кода
  :
  :
V(sem)
```

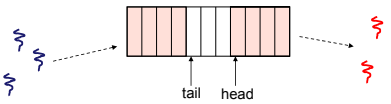
  - как и в случае признаков блокировки, отсутствует поддержка со стороны языков программирования (что не способствует безошибочному использованию)
- Однако, реализации данных механизмов существенно отличаются

Вопросы

- Есть ли какие-либо преимущества в использовании семафоров вместо активного ожидания (проверка-и-установка) или запрета прерываний (если вы выполняетесь в режиме ядра)?
- Какие нежелательные последствия возникнут, если выполнить "лишний" вызов V(sem)?
- Какие нежелательные последствия возникнут, если "забыть" выполнить вызов P(sem)?

Пример: Использование кольцевого буфера

- *Задача "поставщик-потребитель« (producer/consumer problem)*
  - имеется циклический буфер из N элементов
  - потоки-поставщики добавляют в него записи (по одной за одну операцию)
  - потоки-потребители извлекают записи (также по одной за раз)
- Потоки выполняются параллельно
  - таким образом, мы должны использовать синхронизирующие конструкции для управления доступом к разделяемым переменным, описывающим состояние буфера



Реализация с использованием семафоров (как двоичных, так и счетных)

```
var mutex: semaphore = 1    ;мьютекс, управляющий доступом к разделяемым данным
    empty: semaphore = n    ;количество пустых записей (после запуске все записи пусты)
    full: semaphore = 0     ;количество заполненных записей (после запуска нет заполненных записей)
```

```
producer:
    P(empty)        ; нужна свободная запись, если их нет - ждем
    P(mutex)        ; получаем доступ к указателям
      <записываем значение в запись>
    V(mutex)        ; завершили работу с указателями
    V(full)         ; теперь есть еще одна заполненная запись
```

```
consumer:
    P(full)         ;ждем появления заполненной записи
    P(mutex)        ;получаем доступ к указателям
      <извлекаем данные из записи>
    V(mutex)        ; завершили работу с указателями
    V(empty)        ; теперь есть еще одна свободная запись
      <используем данные из записи>
```

Замечание 1:
Мы не приводим код, относящийся к указанию первой и последней заполненным записям и т.д.

Замечание 2:
Попробуйте догадаться, как решить данную задачу без использования счетных семафоров!

Пример: Писатели-Читатели

- Постановка задачи:
  - Некоторый объект разделяется несколькими потоками/процессами
  - Время от времени поток читает содержимое объекта
  - Время от времени поток изменяет (записывает) содержимое объекта
  - **Можно ли допустить одновременную работу нескольких читателей?**
    - почему?
  - **Можно ли допустить одновременную работу нескольких писателей?**
    - почему?

Решение задачи " Писатели-Читатели "

```
var mutex: semaphore = 1    ; управляет доступом к readcount
    wrt: semaphore = 1      ; управляет допуском к данным писателя или 1-го читателя
    readcount: integer = 0  ; количество "активных" (читающих) читателей
```

```
writer:
    P(wrt)                  ; кто-то читает или пишет?
        <выполняем операцию записи>
    V(wrt)                  ; разблокируем доступ к ресурсу
```

```
reader:
    P(mutex)                        ; получаем эксклюзивный доступ к readcount
      readcount++                   ; увеличиваем число активных читателей
      if readcount == 1 then P(wrt) ; если мы первые – блокируем доступ к ресурсу
    V(mutex)
        <выполняем операцию чтения>
    P(mutex)                        ; получаем эксклюзивный доступ к readcount
      readcount--                   ; уменьшаем число активных читателей
      if readcount == 0 then V(wrt) ; мы были последними – разблокируем доступ
    V(mutex)
```

Замечания к решению задачи "Читатели-Писатели"

- Первый появившийся читатель переходит в состояние ожидания при выполнении P(wrt) в присутствии писателя
  - все остальные читатели переходят в состояние ожидания при выполнении вызова P(mutex)
- Если имеется ожидающий писатель, последний завершивший чтение читатель подаст ему сигнал об освобождении ресурса
  - могут ли новый читатели начинать чтение, если имеется ожидающий писатель?
- Если при завершении работы писателем уже присутствуют и писатели, и читатели, ожидающие освобождения ресурса – кто продолжит свое выполнение?

Семафоры vs. признаки блокировки

- Потоки, перешедшие в состояние ожидания согласно логике построения программы, помещаются в очередь, а не выполняют активное ожидание
- Активное ожидание может использоваться для "настоящего" взаимного исключения, необходимого для реализации операций P и V
  - но это критические секции очень малого объема и абсолютно не зависящие от логики работы программы

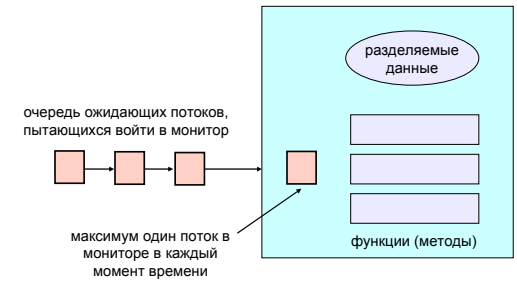
Проблемы при использовании семафоров (и признаков блокировки)

- С их помощью можно решить любую классическую задачу синхронизации, но
  - по сути, семафоры – это разделяемые глобальные переменные
    - и к ним можно получить доступ отовсюду (что является признаком плохой структуры программы – bad software engineering)
  - нет связи между семафором и данными, доступом к которым он управляет
  - они используются как для реализации критических секций (решения задачи взаимного исключения), так и для координации действий (планирования)
  - нет контроля за их использованием, соответственно – нет гарантий, что они будут использованы правильно
- Таким образом, их использование чревато возникновением ошибок
  - еще один подход (лучший ли?): использование поддержки со стороны языка программирования

Еще один подход: Мониторы

- *Монитор* – это конструкция языка программирования, поддерживающая управляемый доступ к разделяемым данным
  - код синхронизации добавляется компилятором
    - чем это лучше?
- Монитор инкапсулирует
  - ***разделяемые структуры данных***
  - ***функции***, использующие разделяемые данные
  - ***синхронизацию*** выполнения параллельных потоков, вызывающих указанные функции
- Доступ к данным, расположенным в мониторе, реализуется только посредством вызова предоставленных функций
- Учтены ключевые замечания по использованию, отмеченные нами для семафоров

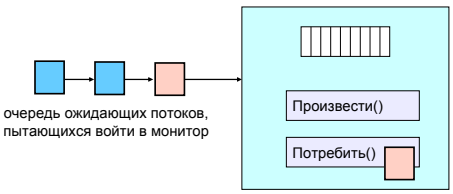
Монитор



Возможности мониторов

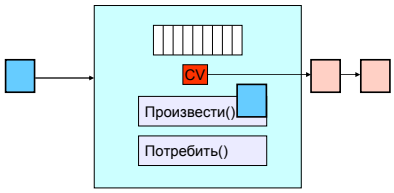
- "Автоматическое" взаимное исключение
  - только один поток может находиться в мониторе в любой момент времени
    - таким образом, монитор по определению реализует синхронизацию – бери и пользуйся
  - если второй поток пытается выполнить функцию монитора, он переходит в состояние ожидания до выхода первого потока из монитора
    - более жесткие условия, чем при использовании семафоров
    - но проще в использовании (в большинстве случаев)
- Но,… есть проблема

Пример: Циклический Буфер



- Заполненных записей нет
- Что получилось?
- Решение?

Пример: Циклический Буфер



CV = condition variable / условная переменная

Условные переменные (Condition variables)

- Место ожидания, иногда называемое местом встречи (rendezvous point)
- "Требуются" для мониторов
  - Но настолько полезны, что часто реализуются даже в их отсутствие
- Над условными переменными определены три операции
  - **wait(c)**
    - снимает блокировку монитора, после чего другой поток может войти в него
    - ожидает, пока какой-либо другой поток не подаст условный сигнал
    - таким образом, условным переменным ставятся в соответствие очереди ожидания
  - **signal(c)**
    - пробуждает максимум один ожидающий поток
    - если нет ожидающих потоков, информация о приходе сигнала теряется
      - в этом отличие от семафоров – никакой истории!
  - **broadcast(c)**
    - пробуждает все ожидающие потоки

Реализация Циклического Буфера с использованием мониторов

```
Monitor bounded_buffer {
   buffer resources[N];
   condition not_full, not_empty;

   produce(resource x) {
      if (array "resources" is full)          // determined by a counter
         wait(not_full);
      insert "x" in array "resources"
      signal(not_empty);
   }

   consume(resource *x) {
      if (array "resources" is empty)         // determined by a counter
         wait(not_empty);
      *x = get resource from array "resources"
      signal(not_full);
   }
}
```

Два вида мониторов

- Кто запускается по окончании обработки вызова signal(), и где расположены потоки, ожидающие выполнения условия?
- **Мониторы Хоара**: signal(c) означает:
  - запустить ожидающий поток немедленно
  - поток, пославший сигнал, немедленно блокируется
    - и остается блокированным все время, пока выполняется поток, которого он вывел из состояния ожидания
    - однако, пославший сигнал поток должен восстановить инварианты монитора перед посылкой сигнала!
      - он не должен оставлять неразбериху ожидающему потоку, который запустится сразу после его сигнала!
- **Мониторы Меса**: signal(c) означает:
  - ожидающий поток переводится в состояние "готов к выполнению", а поток, пославший сигнал, продолжает исполнение
    - ожидавший поток запускается при выходе потока, пославшего сигнал, из монитора или его перехода в состояние ожидания
  - но поток, пославший сигнал, может не восстанавливать состояние монитора вплоть до выхода из него
  - если вас разбудили – это всего лишь намек на какие-то произошедшие изменения
    - условие ожидания может не быть выполнено
    - нужно опять проверять выполнение ожидаемых условий

Мониторы Хоара vs. Мониторы Меса

- Мониторы Хоара: `if (notReady) wait(c)`
- Мониторы Меса: `while (notReady) wait(c)`
- Мониторы Меса проще использовать
  - более эффективны
  - меньшее количество переключений
  - непосредственно поддерживает broadcast (широковещательную рассылку сигналов)
- С мониторами Хоара меньше случайностей
  - если поток разбужен, гарантируется выполнение условия, которого он ожидал

Мониторы: Выводы

- Мониторы поддерживаются языками программирования
- Они обрабатываются компилятором
  - компилятор при построении кода добавляет в него вызовы функций библиотеки времени исполнения (runtime routines):
    - monitor entry
    - monitor exit
    - signal
    - wait
- Реализация перечисленных процедур находится в библиотеке времени исполнения (runtime system)
  - перевод потоков из очереди в очередь
  - взаимное исключение!

Условные переменные и мьютексы

- Условные переменные могут быть использованы с мьютексами (см. проект №2):

```
pthread_mutex_t mu;
pthread_cond_t co;
boolean ready;
void foo( ) {
 pthread_mutex_lock(&mu);
 if (!ready)
   pthread_cond_wait(&co, &mu);
 …
 ready = TRUE;
 pthread_cond_signal(&co);  // unlock and signal atomically
 pthread_mutex_unlock(&mu);
}
```

- Реализация мониторов:
  - Компилятор использует мьютекс для контролирования доступа потоков к функциям монитора