

Лекция 5. Концепции потока

Преподаватели:

Арчакова С.А., Шубин А.А.

Вопросы, рассматриваемые на данной лекции

- Определение потока;
- Асинхронное параллельное выполнение;
- Семафоры;
- Мониторы



Определение потока

- Поток (thread) — логический объект, описывающий последовательность независимо выполняемых программных инструкций внутри процесса. Потоки позволяют воспользоваться преимуществами параллельного выполнения операций в рамках процесса. Каждый процесс имеет как минимум один поток выполнения.
- Многопоточность (multithreading) — технология, позволяющая включать в состав процесса несколько работающих потоков для выполнения параллельных операций, возможно даже одновременно (для многопроцессорных / многоядерных систем).

Определение потока

Элементы процесса

Совместно используемые всеми потоками процесса	Индивидуальные для каждого потока процесса
<p>Адресное пространство; Родительский процесс; Дочерние процессы; Открытые файлы; Необработанные аварийные сигналы; Сигналы и их обработчики; Информация об использовании ресурсов</p>	<p>Состояние (готов, выполняется, блокирован); Программный счетчик; Контекст выполнения; Стек процедур потока</p>

Определение потока

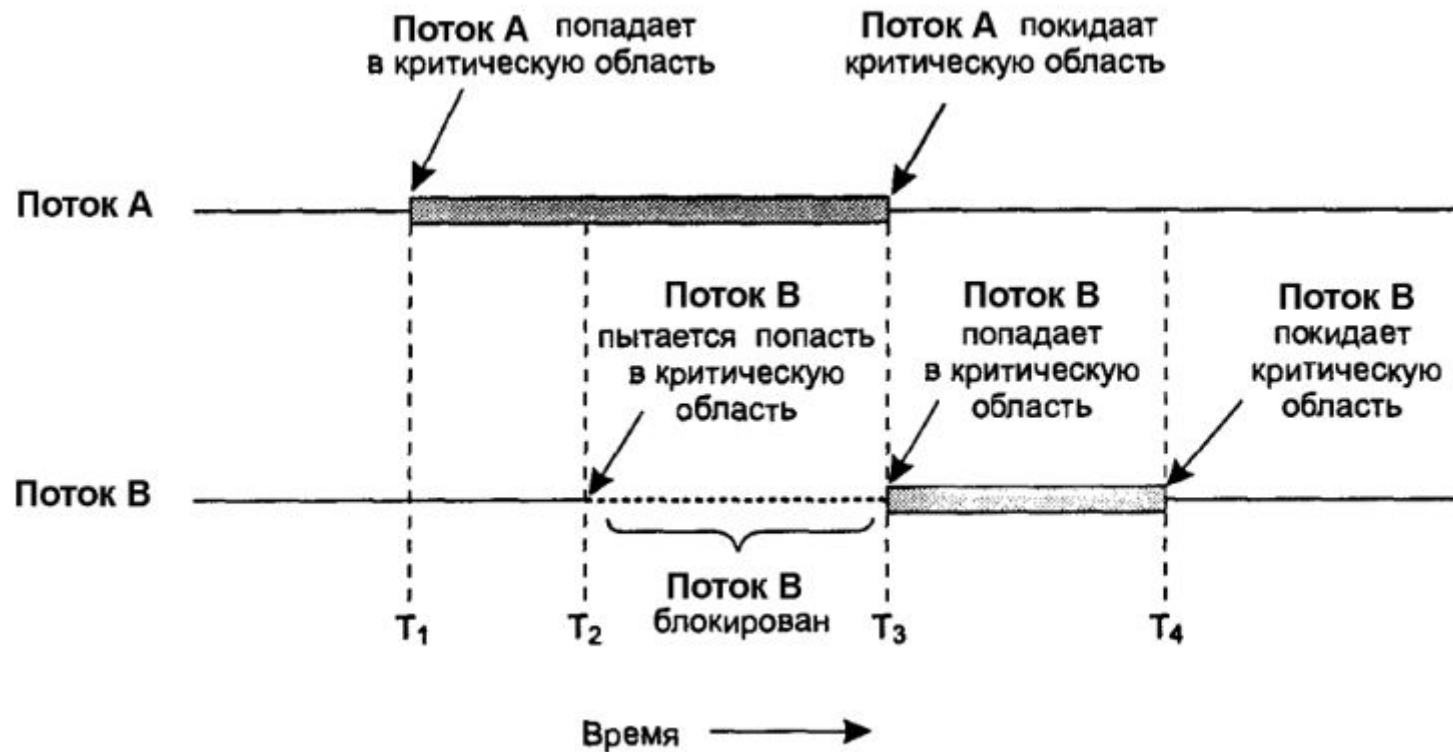
Мотивы использования потоков

- Архитектура системы программирования обеспечивает написание фрагментов кода, которые должны выполняться параллельно;
- Производительность многопроцессорных / многоядерных систем;
- Взаимодействие потоков через общее адресное пространство

Асинхронное параллельное выполнение

- Асинхронные параллельные потоки (asynchronous concurrent threads) - потоки, которые существуют в системе одновременно и выполняются независимо друг от друга, но периодически должны синхронизироваться и взаимодействовать.
- Критический участок (critical section, критическая область) - фрагмент кода, выполняющий операции над разделяемым ресурсом (например, запись значения в разделяемую переменную). Чтобы добиться корректной работы программы, в своем критическом участке в любой момент времени должен находиться только один поток.
- Взаимоисключение (mutual exclusion) - ограничение, в соответствии с которым выполнение одного потока внутри своего критического участка исключает выполнение других потоков внутри своих критических участков. Взаимоисключение жизненно важно для обеспечения корректной работы нескольких потоков, обращающихся к одним и тем же разделяемым данным для их модификации.

Асинхронное параллельное выполнение



На рисунке показано взаимное исключение с использованием критических участков

Асинхронное параллельное выполнение

Пример: необходимость взаимного исключения.

Пусть потоки А и В — счетчики, использующие общую глобальную переменную *count*.

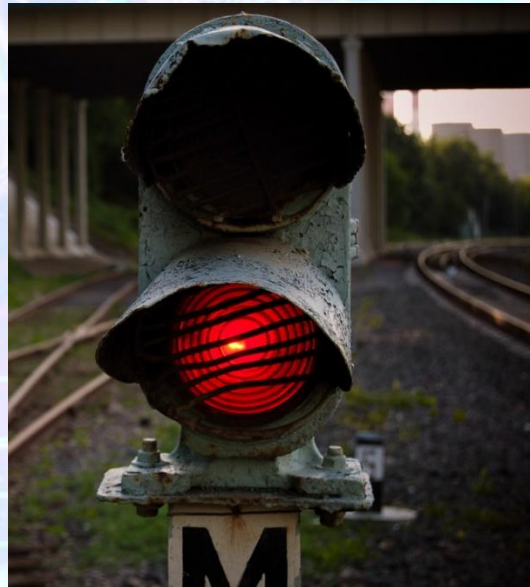
Рассмотрим следующую последовательность событий:

- 1) поток А считывает в регистр R значение переменной $count=n$: $R=n$;
- 2) поток А увеличивает в регистре R значение на единицу: $R=n+1$;
- 3) по истечении кванта времени процессор передается потоку В;
- 4) поток В увеличивает значение переменной *count* на единицу:
 $count=n+1$;
- 5) процессор возвращается потоку А;
- 6) поток А сохраняет значение $n+1$ из регистра R в переменную *count*:
 $count=n+1$.

В результате, переменная *count* имеет значение $n+1$, а при корректной работе потоков А и В, должно быть: $count=n+2$.

Семафоры

- Двоичный семафор (binary semaphore) абстракция, используемая при реализации взаимного исключения, в которой применяются две атомарные операции (P и V) для доступа к защищенной переменной S, определяющей, могут ли потоки входить в свои критические участки (S=1 могут, S=0 нет).



Семафоры

- Защищенная переменная S (protected variable S) - бинарное значение S , в котором хранится состояние семафора. Опросить либо изменить это значение можно только с помощью атомарных операций P и V . При создании семафора S присваивается значение 1.
- Атомарная операция (atomic operation) - операция непрерывно выполняемая от начала до конца.

Семафоры

- Операция P (операция ожидания) - одна из двух операций, позволяющих менять значение семафора S . Если $S=0$, то операция P блокирует вызывающий поток. Если $S>0$, то операция P уменьшит значение S на единицу и позволит вызывающему потоку продолжить работу;
- Операция V (операция оповещения) - одна из двух операций, позволяющих менять значение семафора S . Если у данного семафора есть заблокированные потоки, операция V будит один из них и увеличивает значение S на единицу, если заблокированных потоков нет, то просто увеличивает значение S на единицу.

Семафоры

- Считающий семафор (counting semaphore) - семафор, в котором переменная S целочисленная и может принимать значения больше 1. Считающие семафоры применяются, когда необходимо выделить ресурс из пула идентичных ресурсов.

Считающий семафор

- При создании считающего семафора, переменной S присваивается значение числа n ресурсов в пуле;
- Каждая операция P уменьшает значение S на единицу, показывая, что некоторому потоку выделен один ресурс из пула;
- Каждая операция V увеличивает значение S на единицу, показывая, что поток возвратил один ресурс в пул.

Мониторы

- ❖ **Монитор (monitor)** - конструкция параллельного программирования, которая содержит как данные, так и процедуры, необходимые для управления взаимным исключением при распределении общего ресурса или пула идентичных ресурсов.

Монитор

- Потоки, обращающиеся к монитору, не знают какие данные находятся внутри монитора и не имеют к ним доступа;
- В каждый момент времени в мониторе может находиться только один поток.

Мониторы

- ❖ Переменная–условие (condition–variable) - переменная, которой соответствует очередь потоков, ожидающих входа в монитор, в случае, если распределяемый ресурс занят.

Переменная–условие

- Если потоку необходимо дождаться переменной–условия в тот момент, когда он находится внутри монитора, он выходит из монитора и попадает в очередь ожидания переменной–условия
- Потоки пребывают в этой очереди до тех пор, пока не получат оповещения от других потоков

Мониторы

```
1 // Resource allocator monitor
2
3 // monitor initialization (performed only once)
4 boolean inUse = false; // simple state variable
5 Condition available; // condition variable
6
7 // request resource
8 monitorEntry void getResource()
9 {
10     if ( inUse ) // is resource in use?
11     {
12         wait( available ); // wait until available is signaled
13     } // end if
14
15     inUse = true; // indicate resource is now in use
16
17 } // end getResource
18
19 // return resource
20 monitorEntry void returnResource()
21 {
22     inUse = false; // indicate resource is not in use
23     signal( available ); // signal a waiting thread to proceed
24
25 } // end returnResource
```

Простейший монитор на псевдокоде. Здесь *getResource()* – аналог операции ожидания P, а *returnResource()* – аналог операции оповещения V

Мониторы

- *wait(conditionVariable)* - процедура монитора, которую поток использует в случае, если ресурс занят; выдав команду ожидания поток выходит из монитора и попадает в очередь.
- *signal(conditionVariable)* - процедура монитора, используя которую поток оповещает другие потоки о том, что ресурс свободен и выходит из монитора; первый поток, ожидающий в очереди, получив сигнал, может выйти из очереди и войти в монитор.