Концепции потока

Определение потока

Поток

Опр. 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

Опр. Многопоточность (multithreading) – технология, позволяющая включать в состав процесса несколько работающих потоков для выполнения параллельных операций, возможно даже одновременно (для многопроцессорных / многоядерных систем).

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

Вопрос для самопроверки

- Верно ли, что создание потока требует меньшего числа тактов, чем создание процесса? (Да/Нет)

Вопрос для самопроверки

- Верно ли, что создание потока требует меньшего числа тактов, чем создание процесса? (Да/Нет)
- Да. Многие элементы процесса совместно используются всеми его потоками. По этому при создании нового потока они уже существуют.

Вопрос для самопроверки

- Верно ли, что взаимодействие процессов более эффективно, чем взаимодействие потоков одного процесса? (Да/Нет)

Вопрос для самопроверки

- Верно ли, что взаимодействие процессов более эффективно, чем взаимодействие потоков одного процесса? (Да/Нет)
- Нет. Потоки, принадлежащие одному процессу, могут обмениваться друг с другом данными с помощью общего адресного пространства, обходясь без механизма взаимодействия процессов, подразумевающего обращение к ядру.

Вопрос для самопроверки

- Могут ли многопоточные приложения выполняться быстрее однопоточных? (Да/Нет)

Вопрос для самопроверки

- Могут ли многопоточные приложения выполняться быстрее однопоточных? (Да/Нет)
- Да. Большинство приложений содержат фрагменты кода, которые могут выполняться независимо от остальной части приложения. Если выделить эти фрагменты в отдельные потоки, можно будет выполнять их отдельно на разных процессорах / ядрах.

Концепции потока

Асинхронное параллельное выполнение

Асинхронные параллельные потоки

Опр. 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

Опр. 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

Опр. 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.

Пример: необходимость взаимоисключения

Пусть потоки A и B – счетчики, использующие общую глобальную переменную count

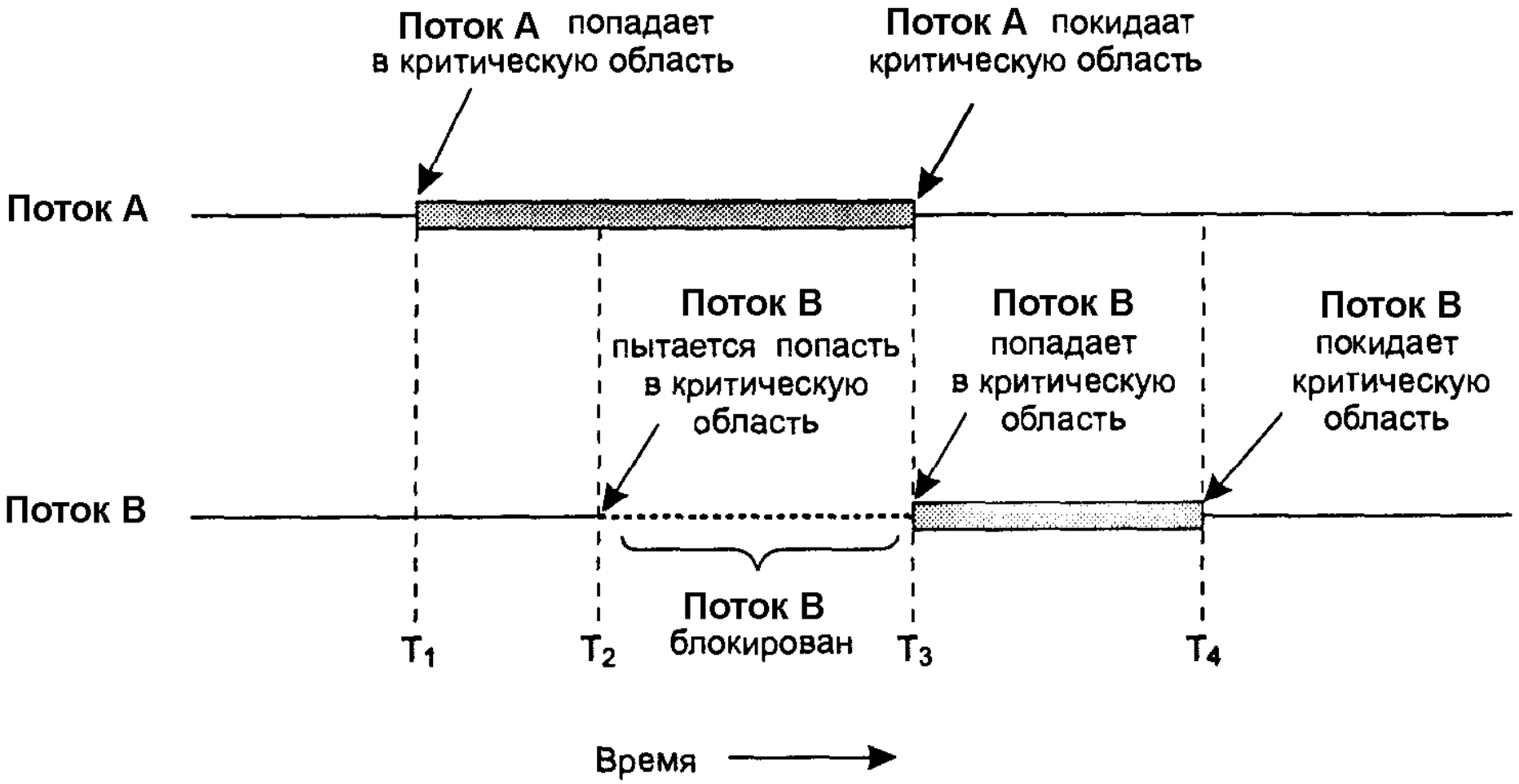
1. Поток A считывает в регистр R значение переменной count=n: R=n
2. Поток A увеличивает в регистре R значение на единицу: R=n+1
3. По истечении кванта времени процессор передается потоку B

Пример: необходимость взаимоисключения

4. Поток B увеличивает значение переменной count на единицу: count=n+1
5. Процессор возвращается потоку A
6. Поток A 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 A и B, должно быть: count=n+2

Взаимоисключение с использованием критических участков



Вопрос для самопроверки

- Верно ли, что синхронизироваться и взаимодействовать должны только асинхронные параллельные потоки? (Да/Нет)

Вопрос для самопроверки

- Верно ли, что синхронизироваться и взаимодействовать должны только асинхронные параллельные потоки? (Да/Нет)
- Нет. Синхронизироваться и взаимодействовать должны также асинхронные параллельные процессы.

Вопрос для самопроверки

- Необходимо ли реализовывать взаимоисключение, если потоки считывают значение разделяемой переменной? (Да/Нет)

Вопрос для самопроверки

- Необходимо ли реализовывать взаимоисключение, если потоки считывают значение разделяемой переменной? (Да/Нет)
- Нет. Взаимоисключение необходимо реализовывать только в случае, если потоки модифицируют разделяемую переменную.

Концепции потока

Семафоры

Двоичный семафор

Опр. Двоичный семафор (binary semaphore) – абстракция, используемая при реализации взаимоисключения, 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).

Защищенная переменная S

Опр. 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

Опр. Атомарная операция (atomic operation) – операция непрерывно выполняемая от начала до конца.

Операция P

Опр. 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

Опр. 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.

Фрагмент кода потока

```
Некритический участок
P(S)
Критический участок
V(S)
Некритический участок
```

Считающий семафор

Опр. 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

Вопрос для самопроверки

- Можно ли реализовать бинарный семафор на основе считающего семафора? (Да/Нет)

Вопрос для самопроверки

- Можно ли реализовать бинарный семафор на основе считающего семафора? (Да/Нет)
- Да. Для этого достаточно задать в качестве начального значения переменной S считающего семафора единицу.

Вопрос для самопроверки

- Верно ли, что в любой момент времени поток может находиться в очереди ожидания только одного семафора? (Да/Нет)

Вопрос для самопроверки

- Верно ли, что в любой момент времени поток может находиться в очереди ожидания только одного семафора? (Да/Нет)
- Да. При помещении в очередь ожидания семафора поток блокируется, будучи не в состоянии выполнить программный код, из-за которого он мог бы оказаться в очереди ожидания другого семафора.

Вопрос для самопроверки

- Верно ли, что операция V считающего семафора всегда увеличивает значение этого семафора на единицу? (Да/Нет)

Вопрос для самопроверки

- Верно ли, что операция V считающего семафора всегда увеличивает значение этого семафора на единицу? (Да/Нет)
- Нет. Если один и более потоков находятся в состоянии ожидания, операция V разбудит один из них, не увеличивая значения счетчика, так как разбуженному потоку будет выделен один освободившийся ресурс.

Концепции потока

Мониторы

Монитор

Опр. Монитор (monitor) – конструкция параллельного программирования, которая содержит, как данные, так и процедуры, необходимые для управления взаимоисключением 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

Переменная-условие

Опр. 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

wait

Опр. **`wait(conditionVariable)`** – процедура монитора, которую поток использует в случае, если ресурс занят; выдав команду ожидания поток выходит из монитора и попадает в очередь.

signal

Опр. **`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.

Простейший монитор на псевдокоде

```
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
```

```
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
```

getResource – аналог операции ожидания P

returnResource – аналог операции оповещения V

Вопрос для самопроверки

- Можно ли реализовать двоичный семафор с помощью монитора? (Да/Нет)

Вопрос для самопроверки

- Можно ли реализовать двоичный семафор с помощью монитора? (Да/Нет)
- Да. Пример простейшего монитора показывает, как это можно сделать.

Вопрос для самопроверки

- Верно ли, что каждый монитор имеет ровно одну переменную-условие? (Да/Нет)

Вопрос для самопроверки

- Верно ли, что каждый монитор имеет ровно одну переменную-условие? (Да/Нет)
- Нет. Монитор может иметь отдельные переменные-условия для каждой отдельной ситуации, которая может привести к вызову команды ожидания в мониторе.

Вопрос для самопроверки

- Верно ли, что для переменных-условий мониторов используются очереди с приоритетами? (Да/Нет)

Вопрос для самопроверки

- Верно ли, что для переменных-условий мониторов используются очереди с приоритетами? (Да/Нет)
- Нет. Поток с низким приоритетом может оказаться в ситуации бесконечного откладывания из-за множества высокоприоритетных потоков, вызывающих функцию ожидания монитора при входе в приоритетную очередь.