Лекция 12. Конкуренция потоков. Пулы потоков.

Содержание

- О взаимной конкуренции потоков. `BackOff`.
- Пулы потоков.

О взаимной конкуренции потоков. `BackOff`.

О взаимной конкуренции потоков

Исполняется несколько потоков с высокой конкуренцией:

```
T* pop() {
  do  {
    T* r = top;
    if (r == NULL)
      return NULL;
    T* next = r->next;
  }while(!CAS(top,r,next));
}
```

```
T* pop() {
  do  {
    T* r = top;
    if (r == NULL)
      return NULL;
    T* next = r->next;
  }while(!CAS(top,r,next));
}
```

О взаимной конкуренции потоков

- При высокой конкуренции на нескольких потоках постоянно исполняется операция `CAS`.
- Операция `CAS` вызывает сброс буферов записи в памяти и блокирует шину.
- При постоянном повторении операции все потоки находятся в состоянии живой блокировки.
- Чем больше потоков, тем меньше КПД.
- Ситуация напоминает `Ethernet` на общей шине.

BackOff

- BackOff — соглашение о взаимодействии нескольких параллельных потоков.
- Постоянный CAS-цикл блокирует всех.
- Протокол BackOff уменьшает конкуренцию за счёт «любезностей», оказываемых потоками друг другу.

BackOff

- Каждый поток при попытке конкуренции с другим заводит счётчик попыток L.
- При неудачной попытке CAS счётчик L увеличивается.
- Поток задерживает исполнение на случайное время в диапазоне $[0 \dots 2^L)$ единиц.
- В неблокирующих алгоритмах — задержка есть цикл.
- Сам процесс расходует ресурс процессора.
- Другие процессы имеют шанс продвинуться в исполнении задачи.
- Более точное название — Exponential BackOff

Пулы потоков.

Пулы потоков

- Нами рассматривались сценарии автономных потоков, использующих совместные ресурсы.
- Другой сценарий — поступают ресурсы, для обработки которых требуются потоки.
- На время обработки ресурс монопольно принадлежит потоку.
- Проблема №1: время обработки ресурса должно быть мало.
- Проблема №2: время создания потока довольно большое.

Пулы потоков

- Типичная задача: на вход системы поступают запросы на обработку (очередь запросов).
- Размер каждого запроса различный.
- Запросы позволяют параллельную обработку их фрагментов.
- Требуется обеспечить максимально быструю передачу обработанных запросов в выходную очередь.

Пул потоков

- Классическое использование потоков:
  - Имеется задача, допускающая разбиение на подзадачи.
  - Каждая задача способна решаться отдельно во взаимодействием с другими.
  - Для каждой подзадачи порождается поток.
  - После решения задачи поток завершается.
- Проблемы классического использования:
  - Главная проблема: время порождения потока.
  - Для порождения потока ОС должна:
  - Выделить в адресном пространстве процесса страницы для размещения стека
  - Создать запись о новом потоке в таблицах планировщика.
  - Создать в контексте потока память для сохранения регистров

Пул потоков

- Альтернативное использование потоков:
  - Однократно породить необходимое количество потоков, согласованное с количеством физических и виртуальных процессоров/ядер.
  - Каждый поток находится в состоянии ожидания до тех пор, пока ему не будет назначена работа.
  - Что есть назначение работы?
  - Назначение контекста исполнения.
  - Назначение исполнителя.

Пул потоков

- В системном программировании паттерн использования вычислительных ядер непредсказуем.
- При обработке запросов случайны:
  - время поступления запроса
  - размер запроса
  - вид запроса
  - текущая загрузка существующих ядер

Абстракция пула потоков

Пул вычислительных потоков — абстракция, представляющая данные и методы их использования.

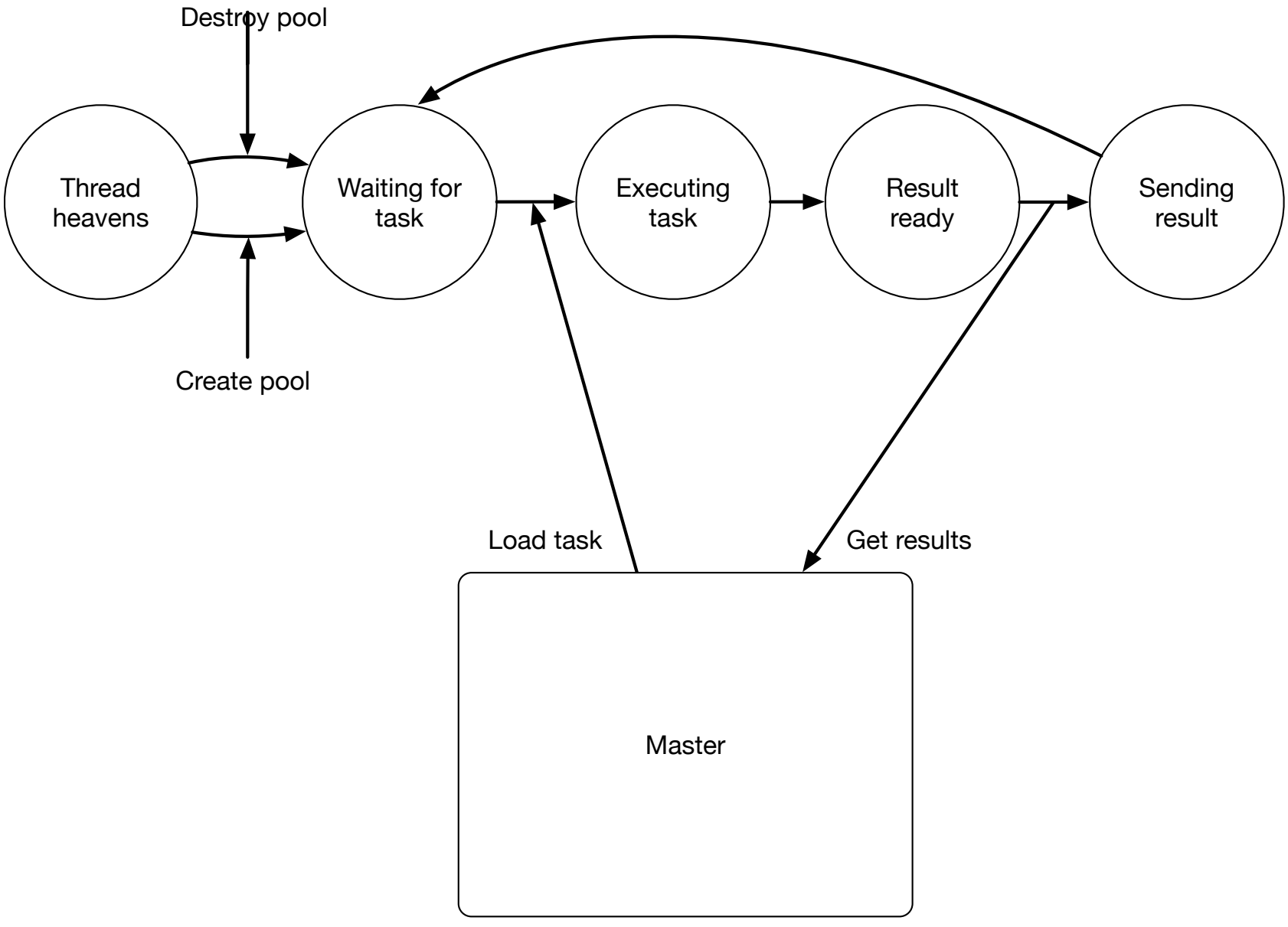
- Положение: планирование потоков недоступно для приложений верхнего уровня.
- Положение: для максимально эффективного использования вычислительных ресурсов потоки нужно создать однократно.
- Следствие: для эффективности потоки требуется планировать статически.
  - Запуск — однократный
  - Возобновление — по мере необходимости
  - Приостановка — по мере необходимости
  - Завершение — однократно

Пул вычислительных потоков

Возможные состояния потоков пула

- Ожидает поступления задачи (`Waiting for task`)
- Исполняет задачу (`Executing`)
- Задача исполнена (`Results ready`)
- Результаты передаются (`Sending results`)

Пул вычислительных потоков

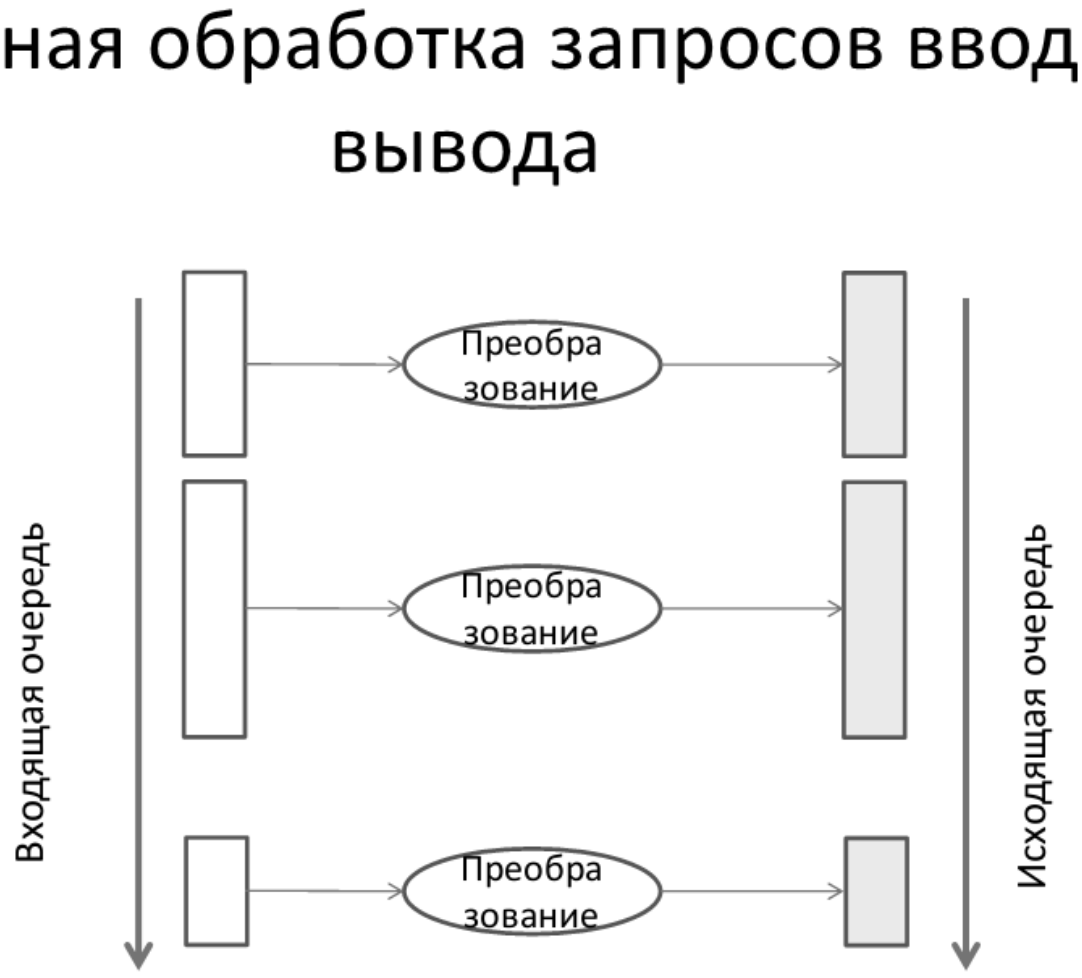


Пул вычислительных потоков

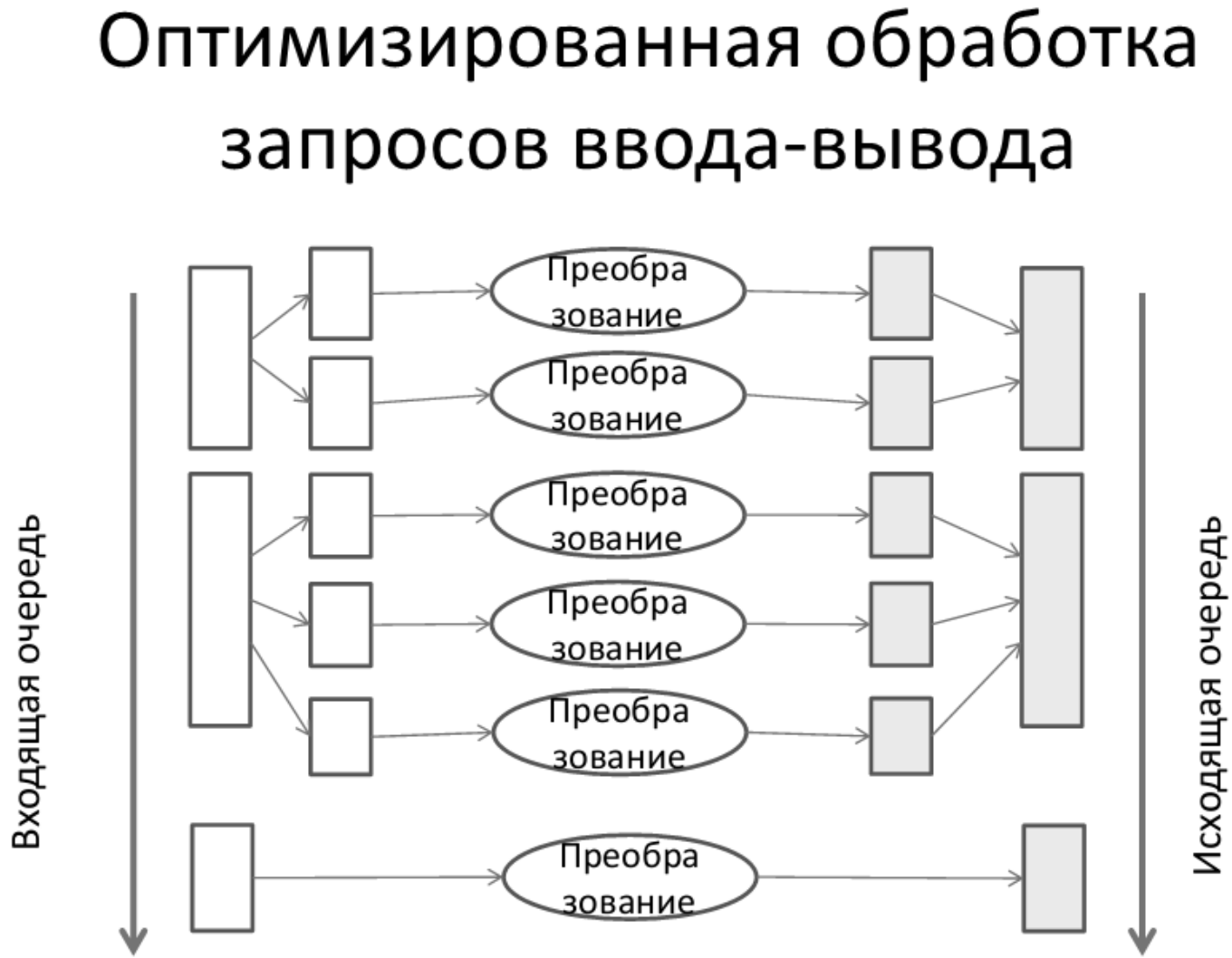
После исполнения задачи возможны несколько сценариев:

- Задача самодостаточна и обнаружение её завершения ложится на вызывающую сторону.
- Задача остаётся в пуле до того, как вызывающая сторона явным образом не запросит результаты.
  - Обратным вызовом процедуры завершения (`callback`).
  - Установкой события о завершении.

Пул вычислительных потоков: оптимизация ввода/вывода



Пул вычислительных потоков: оптимизация ввода/вывода



Пул вычислительных потоков: оптимизация ввода/вывода

- Время обработки запроса определяется:
  - латентностью механизма, применяемого при обработке ввода/вывода (константная для конкретной вычислительной системы величина)
  - пропускной способностью, то есть, количеством обрабатываемой информации в единицу времени.
- При константной латентности требуется увеличить пропускную способность, что может быть достигнуто в том числе и параллельной обработкой фрагментов запроса.

ПВП для обработки ввода/вывода

- Появление запроса на исполнение задачи непредсказуемо, следовательно, будут использоваться «длинные» примитивы синхронизации, связанные с контекстными переключениями потоков.
- Операция переключения контекста непродуктивна с точки зрения основного алгоритма и должна быть отнесена к накладным расходам.
- Для реализации пула вычислительных потоков требуется алгоритм, минимизирующий количество операций синхронизации, особенно на критических маршрутах.

ПВП для обработки ввода/вывода

Пример: фрагменты из технического задания на разработку ПВП.

1. Должны быть поддержаны языки программирования C и C++
2. Время, затрачиваемое на накладные расходы должно быть минимально возможным.
3. Реализация должна быть максимально переносимой и поэтому все синхронизирующие примитивы должны быть стандартными и реализуемыми на разных операционных системах.
4. Не должно быть значительных периодов активного ожидания.
5. Должна быть возможность использования механизма в ядре операционной системы.
6. Как следствие из второго пункта выводится, что количество операций синхронизации, исполняемых в одном потоке, должно быть минимальным.

Организация ПВП и методы работы с пулом

Организация ПВП

Два принципиально разных методов организации ПВП:

- Динамическое планирование: вычислительная работа организуется в очередь, которая обрабатывается пулом. Вычислительная работа может исполняться любым свободным потоком из пула.
- Динамическое планирование очень удобно в том случае, если у нас заранее неизвестно ни количество вычислительной работы, ни моменты начала отдельных её этапов.
- Статическое планирование: потоки для фрагментов задания распределяются заранее.
- Если количество и точный момент начала всех этапов работы нам известны - имеет смысл воспользоваться статическим планированием.

Организация ПВП

Накладные расходы при различной организации ПВП:

- При динамическом планировании запросы должны помещаться в очереди.
- Между потоками потребуются действия по синхронизации входной и выходных очередей.
- Эти накладные расходы можно свести к минимуму, тщательно используя неблокирующие механизмы активного ожидания но, тем не менее, они существуют.
- При статическом планировании расходов на поддержание функционирования очереди не требуется.
- Все широко известные реализации пула вычислительных потоков используют динамическое планирование.

Организация ПВП

Ещё особенности использования пулов потоков.

- Возможно создать несколько пулов. Это полезно на NUMA-архитектурах, в которых имеются группы процессорных ядер и связанная с группой память, доступ к которой из данной группы занимает меньше времени, чем доступ к памяти другой группы.
- Существуют также процессорные архитектуры, строго говоря, не являющиеся представителями архитектуры NUMA, использующие общую кэш-память какого-либо уровня на группу процессоров (AMD Bulldozer/Steamroller). В этом случае распределение ПВП по группам может привести к увеличению общей производительности вычислительной системы за счёт исключения лишних операций синхронизации кэша.

Функционирование ПВП

- Создание ПВП может происходить однократно. После создания ПВП в операционной системе появляются вычислительные потоки, управляемые ПВП, которые ожидают появления исполнителя и данных для обработки.
- В случае появления запроса, допускающего параллельное исполнение на нескольких вычислительных ядрах, к ПВП выдаётся обращение для формирования начала пакета заданий. К пакету заданий добавляются исполнитель кода, и данные для его работы.
- Операция запуска пакета приводит к назначению на свободные процессоры заданий из пакета.
- После завершения всех исполнителей из пакета (что может быть обнаружено операцией группового ожидания) пакет считается завершённым становятся доступны результаты исполнения.
- При данном подходе не используются операции активного ожидания и он позволяет минимизировать как количество переключений контекста вычислительных потоков, так и совокупное время исполнения запроса.

Функционирование ПВП

Частое и неизбежное использование примитивов синхронизации при реализации ПВП приходит к тому, что верификация ПВП на `race conditions` и `deadlocks` становится нетривиальной.

Пример ПВП: операции

- `PoolId ThreadPoolCreate(numThreads, flags)` - создание ПВП с numThreads ожидающими потоками. Возвращает идентификатор пула.
- `JobId ThreadPoolPrepareJob(PoolId)` — начало формирования задания. Возвращается идентификатор задания, который затем используется в дальнейших операциях с ПВП.
- `ThreadPoolAddToJob(JobId, Executor)` — добавить к заданию JobId нового исполнителя.
- `ThreadPoolStartJob(JobId)` - запустить задание.
- `ThreadPoolWaitForJobDone(JobId)` — дожидается исполнения всех запущенных заданий. Похож на операцию `join`, но потоки переходят в свободное состояние (состояние ожидания).
- `ThreadPoolDestroy(PoolId)` — дожидается завершения всех активных вычислительных потоков и затем останавливает их, уменьшая количество вычислительных потоков в системе.

Под исполнителем в данном случае понимается пара, состоящая из исполняемой внутри потока функции, и данных для конкретного экземпляра потока.

Использование ПВП для обработки запросов ввода-вывода

Всё множество активных вычислительных потоков в данном алгоритме может быть разбито на два типа

1. первичные, которые занимаются получением запросов ввода-вывода и их предобработкой;
2. вторичные, работающие в составе пула потоков.

- При поступлении новых входных данных для обработки ввода-вывода поток первого типа анализирует наличие целесообразности и возможности для распараллеливания обработки.
- Запросы ввода-вывода небольшого размера обрабатываются в один вычислительный поток, для запросов больших размеров запрос на преобразование разбивается на несколько подзапросов меньшего размера, количество подзапросов не должно превосходить количество вычислительных потоков из пула.

Использование ПВП для обработки запросов ввода-вывода

- Резервируются необходимые ресурсы и подготавливаются структуры данных для дальнейших запросов.
- Для подготовленных подзапросов исполняются операции добавления исполнителя в пул.
- Запускается задание.
- В это же время работает и основной поток, обрабатывающий часть запроса.
- При неуниморфной архитектуре памяти имеет смысл выбирать тот пул, к группе процессоров которого принадлежит основной поток.

Использование ПВП для обработки запросов ввода-вывода

- После завершения основного потока вызывается операция, синхронизирующая основной и дополнительные вычислительные потоки.
- Результаты параллельной работы остаются в структурах данных, подготовленных ранее и принадлежащих исполнителю.
- Основной вычислительный поток должен проанализировать коды возвратов обрабатывающих модулей и консолидировать их в единый код возврата.

Единственными накладными расходами остаются расходы на операции синхронизации.

Спасибо за внимание.

Следующая тема —
Параллельные коллекции.