Параллельное и асинхронное программирование

Параллельное программирование

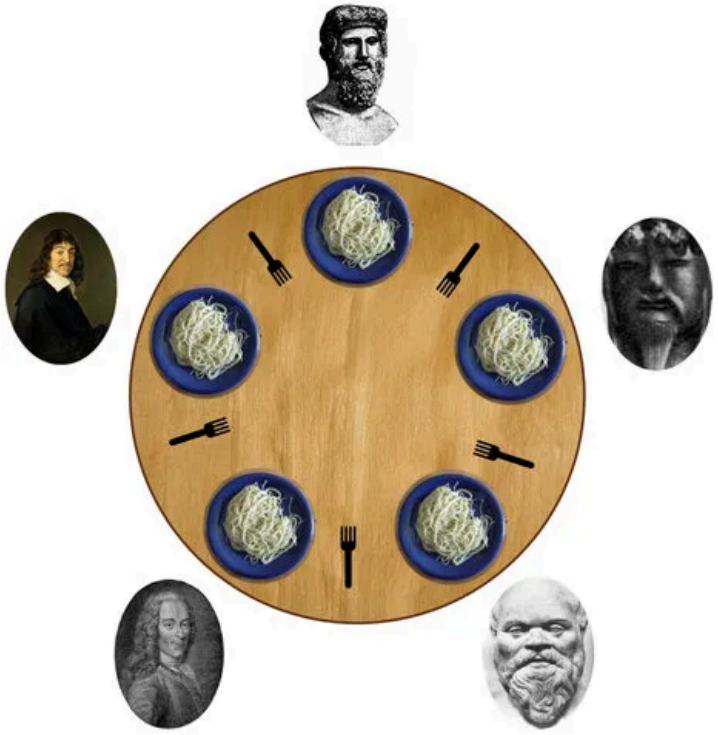
1. Создайте программу, которая считывает список чисел с клавиатуры. Для каждого числа создайте отдельную задачу, которая будет выводить это число на экран. Запустите все задачи параллельно и дождитесь их завершения.
2. Напишите программу, которая считывает список URL-адресов из файла. Для каждого URL-адреса создайте отдельную задачу, которая будет загружать содержимое страницы и сохранять его в отдельный файл. Запустите все задачи параллельно и дождитесь их завершения.
3. Создайте программу, которая считывает список чисел с клавиатуры. Для каждого числа создайте отдельную задачу, которая будет возводить это число в квадрат и возвращать результат. Соберите все результаты в массив и выведите его на экран.
4. Напишите программу, которая считывает список URL-адресов из файла. Для каждого URL-адреса создайте отдельную задачу, которая будет загружать содержимое страницы и возвращать его в виде строки. Соберите все строки в список и выведите его на экран.
5. Создайте программу, которая считывает список файлов из директории. Для каждого файла создайте отдельную задачу, которая будет подсчитывать количество слов в файле. Используйте метод Invoke для запуска всех задач параллельно. После завершения всех задач выведите общее количество слов.
6. Напишите программу, которая считывает список чисел с клавиатуры. Используя метод Parallel.For, вычислите сумму всех чисел и выведите ее на экран.
7. Создайте программу, которая считывает список строк из файла. Используя метод Parallel.ForEach, преобразуйте каждую строку в верхний регистр и сохраните результаты в новый файл.

Асинхронное программирование

1. Создайте асинхронный метод, который будет выполнять простую операцию, например, асинхронную загрузку данных из внешнего источника.
2. Рассмотрите ситуацию, когда вам нужно выполнить несколько асинхронных операций параллельно. Используйте `Task.WhenAll` или `Task.WhenAny`, чтобы дождаться завершения всех асинхронных операций или любой из них соответственно.
3. Используйте `ConfigureAwait`, чтобы контролировать контекст выполнения при использовании `await`. Попробуйте разные значения `ConfigureAwait` и изучите их влияние на поток выполнения кода.
4. Рассмотрите использование асинхронных стримов данных (Async Streams) для эффективной обработки больших объемов данных в асинхронном режиме.

5. Изучите понятие планировщика задач (Task Scheduler) и его роль в асинхронном программировании. Рассмотрите различные способы настройки и настройки планировщика задач для оптимизации производительности.
6. Создайте собственный асинхронный метод, используя ключевое слово `async` вместе с `Task<T>`, чтобы вернуть результат выполнения асинхронной операции.

Задача обедающих философов



Изображение выше иллюстрирует задачу: пять безмолвных философов сидят вокруг круглого стола, перед каждым философом стоит тарелка спагетти. Вилки лежат на столе между каждой парой ближайших философов.

Каждый философ может либо есть, либо размышлять. Приём пищи не ограничен количеством оставшихся спагетти — подразумевается бесконечный запас. Тем не менее, философ может есть только тогда, когда держит две вилки — взятую справа и слева.

Каждый философ может взять ближайшую вилку (если она доступна) или положить — если он уже держит её. Взятие каждой вилки и возвращение её на стол являются раздельными действиями, которые должны выполняться одно за другим.

Вопрос задачи заключается в том, чтобы разработать модель поведения (параллельный алгоритм), при котором ни один из философов не будет голодать, то есть будет вечно чередовать приём пищи и размышления.

Каждая реализация вышеприведенной задачи подвержена следующим проблемам:

Deadlock — состояние системы, в котором каждый философ взял вилку слева и ждёт, когда вилка справа освободится
Starvation (e.g. **livelock**) — состояние системы меняется, но она не совершает никакой полезной работы
Unfairness — состояние системы, в котором некоторые философы получают доступ к своим вилкам существенно чаще, чем остальные философы (такая проблема наблюдается в асимметричных решениях)