Потоковая обработка данных

Потоки и пакеты

Поток — это бесконечная или очень большая последовательность данных, которую нельзя или нецелесообразно загружать в память целиком.

*Поток данных (**stream**)— это абстракция для последовательной передачи, чтения или записи информации между процессами, файлами или устройствами. Это унифицированный интерфейс, который позволяет программам работать с разными источниками (диск, сеть, клавиатура) одинаково*

Пакетная обработка

Сначала копируем всё в буфер затем обрабатываем.
Минус: высокая латентность и потребление RAM.

Латентность (latency)— это задержка между отправкой запроса и получением ответа.

Потоковая обработка

Не нужно скачивать весь двухчасовой фильм, чтобы увидеть первые 5 секунд. Данные приходят маленькими порциями. Обработали 1 КБ — выкинули его из памяти — взяли следующий 1 КБ.

Программа потребляет мало ОЗУ. Обработка начинается мгновенно.

Пример

File.ReadAllBytes("large_file.zip") — если файл 50 ГБ, компьютер зависнет.

File.OpenRead("large_file.zip") — Память почти не расходуется.

Архитектура ввода-вывода в Windows

Синхронный поток

блокируется, пока диск или сеть не ответят. Поток вызывает функцию ReadFile. Процессор перестает выполнять твой код и ждет, пока головка жесткого диска физически доедет до нужного сектора и прочитает данные.

Асинхронный поток

отправляем запрос системе и продолжаем работу. ОС уведомит нас, когда порция данных будет готова.

Буферизация

ОС использует системный кэш для сглаживания пульсаций потока.

Высокоуровневые абстракции

Базовый класс Stream

Единый интерфейс для файлов, сокетов и памяти
FileStream, NetworkStream, MemoryStream.

Пример

Пайплайны (System.IO.Pipelines)

Чтобы обработать данные, их нужно постоянно копировать: из системы в буфер, из буфера в строку, из строки в объект. Каждое копирование - это трата ресурсов.

Пайплайны позволяют обрабатывать данные без лишнего копирования. Они используют специальный пул памяти.

Пример

```
static async Task FillPipeProperlyAsync(PipeWriter writer){
    string message = "Приветй!";
    // 1. Просим у Pipe память (минимум 128 байт)
    // Мы НЕ создаем массив сами, нам дают доступ к уже готовой памяти в пуле
    Memory<byte> memory = writer.GetMemory(128);
    // 2. Кодируем текст СРАЗУ в эту память
    // Метод GetBytes умеет писать напрямую в Span/Memory
    int bytesWritten = Encoding.UTF8.GetBytes(message, memory.Span);
    // 3. Сообщаем трубе, сколько байт мы реально записали
    writer.Advance(bytesWritten);
    // 4. Отправляем данные "вниз по трубе"
    await writer.FlushAsync();
    await writer.CompleteAsync();
}
```

Async Streams/IAsyncEnumerable<T>

Позволяет использовать await foreach для обработки данных, которые поступают асинхронно порциями.

Что использовать

1. Stream — везде для простоты.

2. IAsyncEnumerable — данные приходят порциями и асинхронно.

3. Pipelines— используем только там, где важна экстремальная производительность