Современные подходы к параллельной разработке

Летняя Школа Parallels, 2012

Коротаев А. Е.

Введение в многозадачность

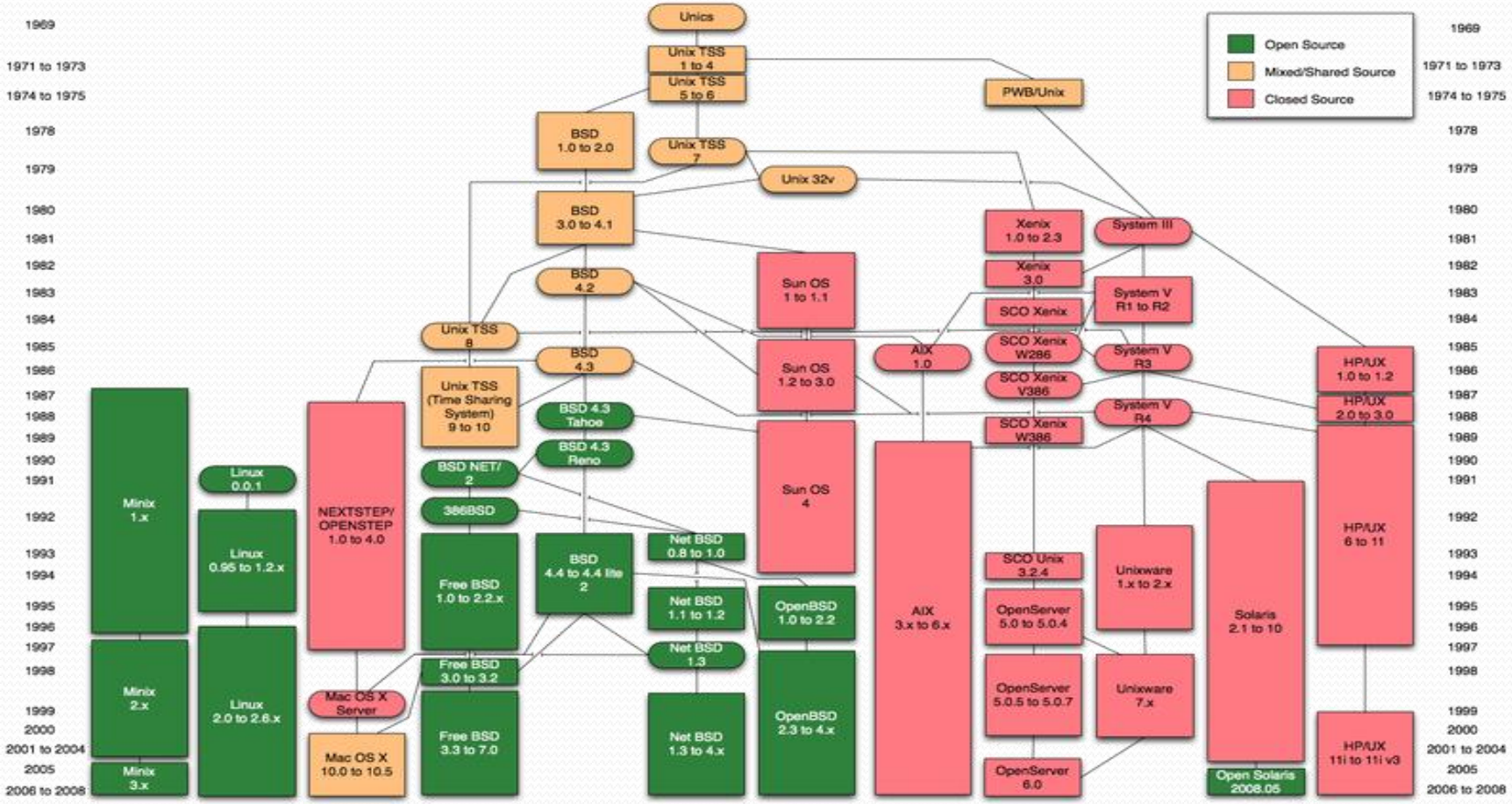
История

- Однозадачные системы
  - Высокая степень надёжности и простота реализации
  - Низкая эффективность
- Многозадачные системы
  - Низкая надёжность и сложность реализации
  - Высокая эффективность
- Первейшие:
  - Multics (1964 год) – первая многозадачная OS
  - Unix (1969 год) – первая удачная многозадачная OS

Нынешнее время

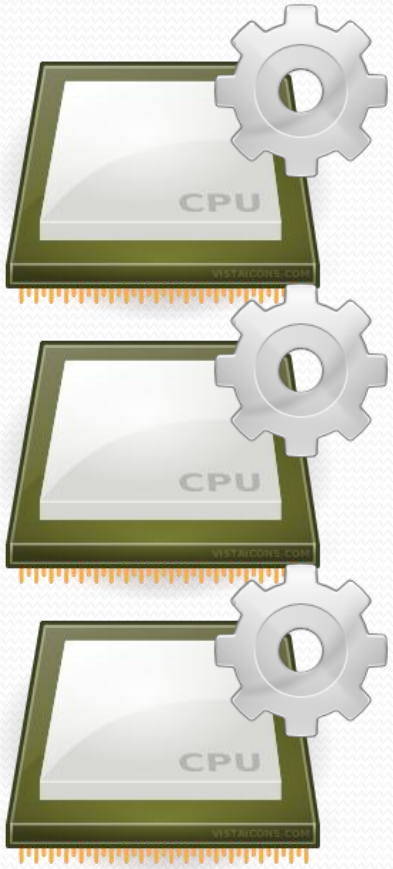
- Семейство OS Windows на ядрах:
  - Win 9X
  - Win NT
- Mac OS
- Linux
- Семейство классической ОС UNIX:
  - Solaris
  - Free BSD
  - OS X
  - …

Семейство ОС UNIX

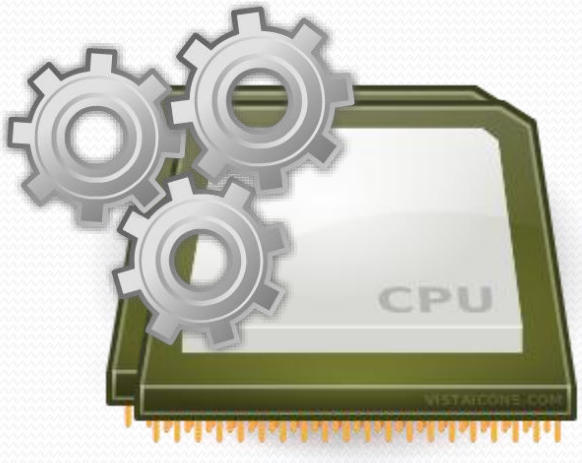


Виды многозадачности

Истинная



Псевдопараллельная



Поток и процесс

- Поток
  - Единица планирования и исполнения
  - Периодически подключается к процессору
  - Содержит процессорные инструкции
- Процесс
  - Единица изоляции ОС
  - Контейнер для потоков

Согласующая многозадачность

- Подвид псевдопараллельной многозадачности
- Поток сам принимает решение, когда ему безопасно переключиться
  - Неявно: вызов функции обработчика сообщений потока
  - Явно: вызов специальной функции ОС
- Примеры:
  - Mac OS
  - Windows 3.x

Вытесняющая многозадачность

- Подвид псевдопараллельной многозадачности
- Решение о передачи CPU другому потоку принимает ОС
  - Обработчик прерывания системного времени
  - Процессорный квант (размер зависит от типа и класса ОС)
- Примеры:
  - Win 9X и Win NT
  - UNIX
  - Linux

“Гибридная” многозадачность

- Все современные ОС реализуют гибридную схему
  - Поддержка множества процессоров – истинная многозадачность
  - Каждый процессор разделяется между потоками – псевдопараллельная многозадачность

Сделайте мне параллельно

Многозадачность в разное время

1964 – середина 2000-ых

- Серверные операционные системы
- Кластеры в научных институтах
- Редкие задачи с большой вычислительной нагрузкой

Наше время

- Рядовые приложения
- Мобильные платформы

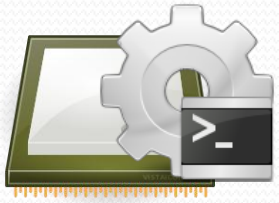
Мир изменился!

- Достигнут технологический предел размера транзистора
  - Нельзя сделать проводник тоньше атома
- Нельзя более наращивать тактовую частоту процессора
  - Производители включились в гонку ядер

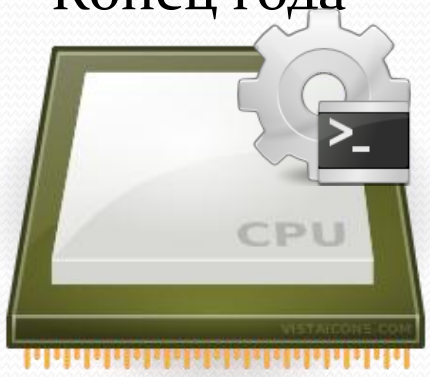
Зачем писать параллельно?

Раньше

- Начало года

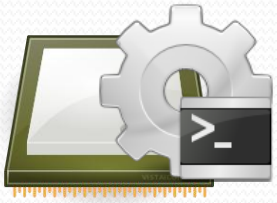


- Конец года

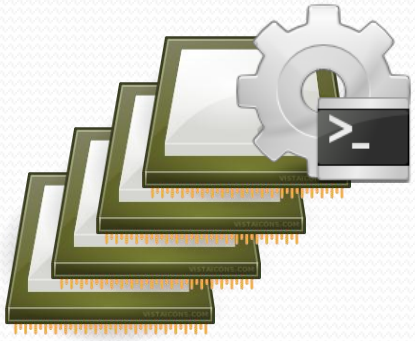


Теперь

- Начало года



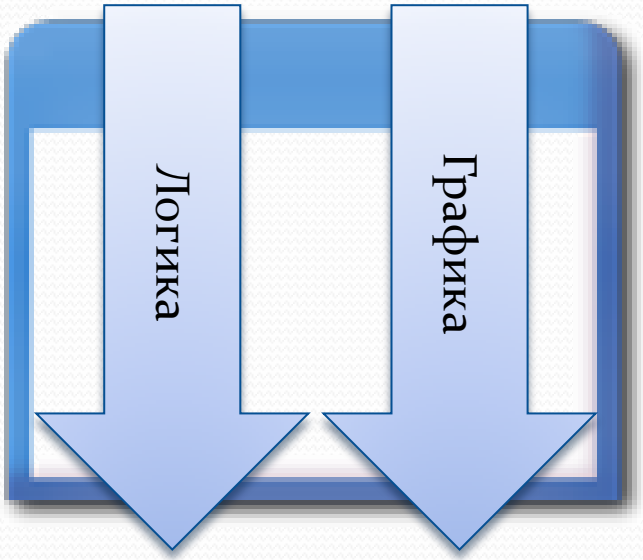
- Конец года



Алгоритмический параллелизм

Как делает большинство

Мой первый реальный опыт в написании многопоточного приложения был связан с разработкой какой-то казуальной игрушки. Она немного подтормаживала, поэтому пришла идея – разбить её на 2 потока: один поток должен был обсчитывать логику, второй отрисовывать графику. Тогда это действительно помогло. Этот метод называется алгоритмическим распараллеливанием. Большинство программистов в то время делало точно также, многие продолжают делать и сейчас.



Масштабируемость

- Главный критерий хорошего многозадачного приложения
- Способность исполняться на том количестве ядер, которые есть в наличии
- Приложение с фиксированным количеством потоков не удовлетворяет этому критерию

Сарказм

Готовые алгоритмы

- Большинство библиотек для параллельной разработки содержат множество готовых алгоритмов
  - QSort
  - Перемножение матриц
  - Параллельный цикл
  - …

Библиотеки

- MS Concurrency Runtime
  - Visual Studio 2010
  - Windows XP SP3
- .NET Task Parallel Library
  - .NET 4.0
- Intel Threading Building Blocks
  - Windows, Linux, OS X
  - Intel, последние модели AMD
- Open MP

Пулы и задачи

Задачи или рабочие элементы

- Самая распространённая модель
- Приложение – множество многократно повторяемых, слабосвязанных процедур
  - Процедура реализует сложный блок функциональности
  - Выполняется в отдельном потоке
- Процедура = задача
  - Число задач не зависит от числа ядер
  - Идеально если разбиение обусловлено обработкой входных данных

Пул потоков

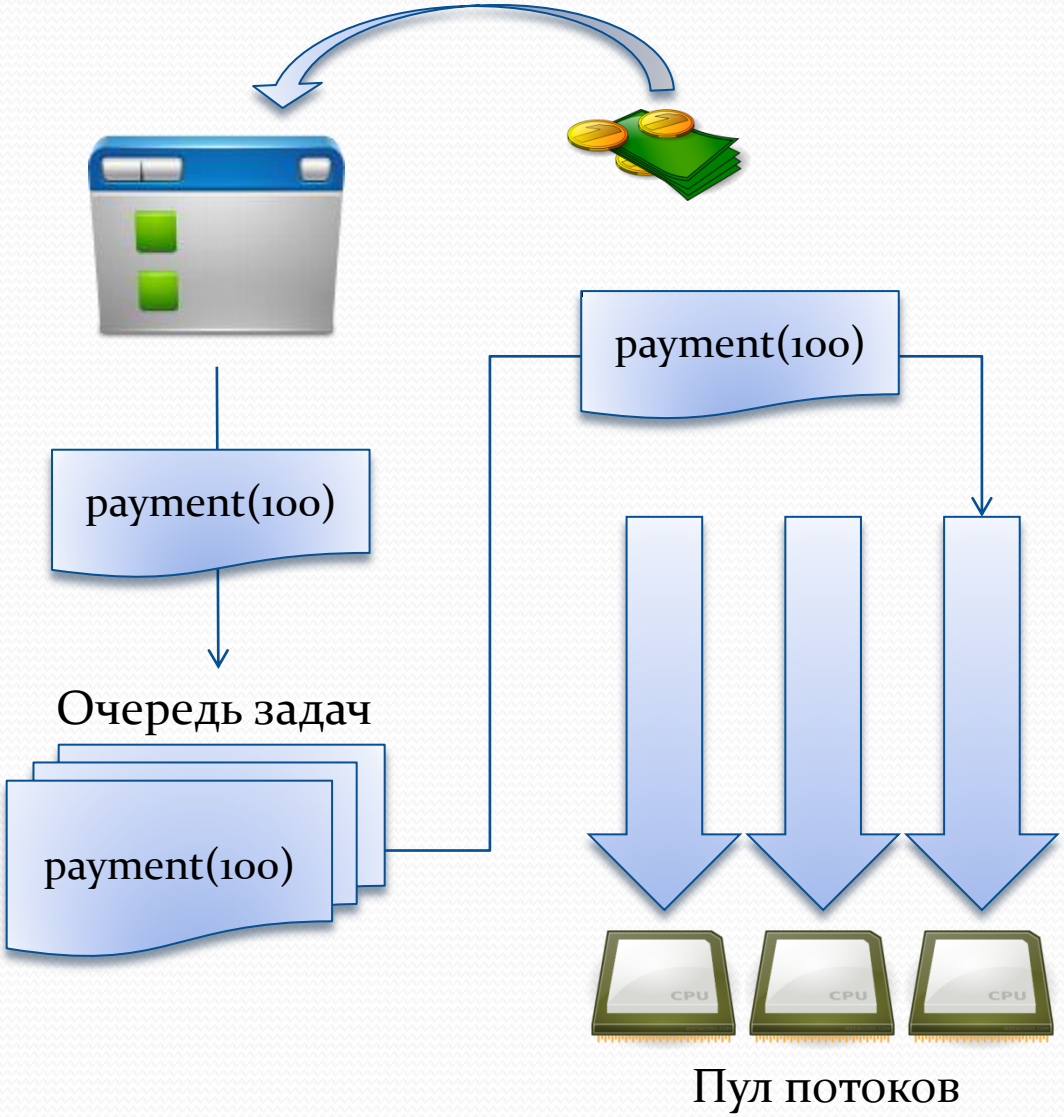
- Поток – дорогой объект
  - Дорого создавать
  - Дорого держать
- Количество одновременно живущих потоков должно быть сбалансированным
- Пул потоков – набор потоков (как правило фиксированный), созданных заранее и существующих в течении жизни приложения, из которого потоки на время передаются на нужды других компонентов приложения.

Пример

Обработка денежной транзакции в платёжной системе:

- ✓поступает информация о совершении платежа
- ✓в очередь задач ставится задача по обработке платежа
- ✓очередная задача передаётся в пул потоков

Число потоков в пуле зависит от числа ядер: увеличение числа ядер приведёт к увеличению одновременно обрабатываемых транзакций, что означает увеличение общей производительности системы.



Модель асинхронных агентов

Критика рабочих элементов

- Независимость задач – главная составляющая эффективности подхода
- Любое взаимодействие между задачами порождает проблемы
- Процедурный подход в обычном программировании – аналог концепции рабочих элементов в параллельном

Агент – характеристики

- Асинхронный агент (актёр, actor) – объект
- Реализует ровно одну задачу
- Все вычисления в рамках задачи выполняются в отдельном потоке
  - Поток из пула
- Может менять внутреннее состояние в процессе вычислений

Агент – характеристики

- Внешние наблюдатели через вызов публичных методов могут:
  - Запрашивать состояние
  - Отправлять сообщения влияющие на вычисления
- Любой публичный метод может
  - Вернуть состояние
  - Передать данные выполняющейся задаче асинхронным образом
- Ни один публичный метод не выполняет никаких вычислений и не изменяет состояние агента!

Агент – характеристики

- Конечная задача
  - Агент может перейти из состояния “запущен” в состояние “занят”
- Непрерывная задача
  - Завершается принудительно, например при уничтожении агента

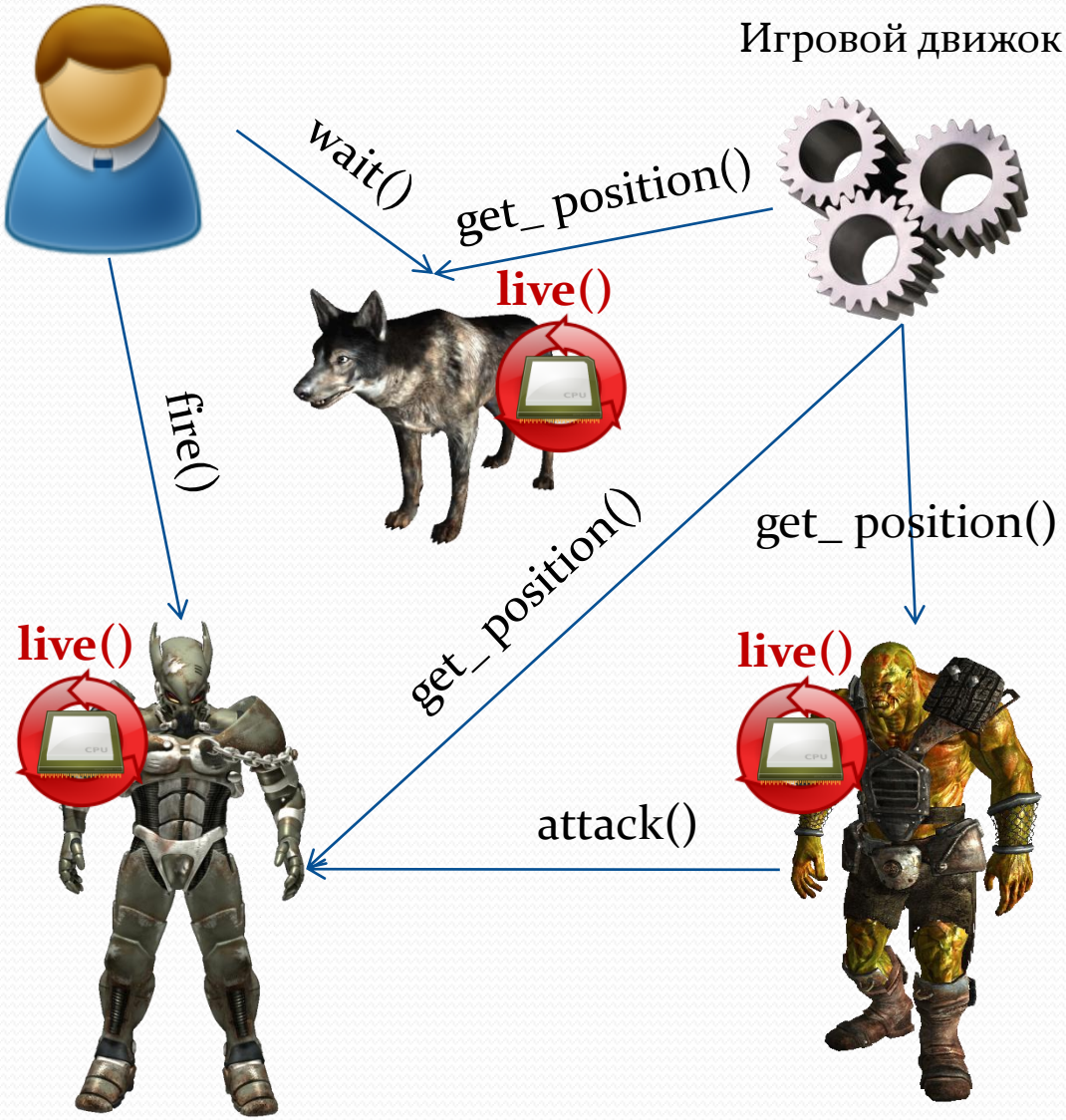
Пример

Хорошей иллюстрацией этой модели является любая игра со множеством персонажей.

Каждый персонаж – это агент., реализующий задачу вычисления логики поведения данного персонажа. Другие персонажи-агенты, движок игры или сам игрок могут:

- ✓запрашивать его состояние, чтобы узнать местоположение или настроение;
- ✓отправлять сообщения чтобы изменить его поведение.

Например хороший пример сообщения: attack() ☺.



Data Flow

Командный интерпретатор

- Вход: последовательный поток команд
  - help, get, dir, cd, …
- Функции интерпретатора:
  - распознать команду
  - выполнить
  - отправить результат на выход
- Выход: последовательный поток ответов
- Задача: распараллелить обработку

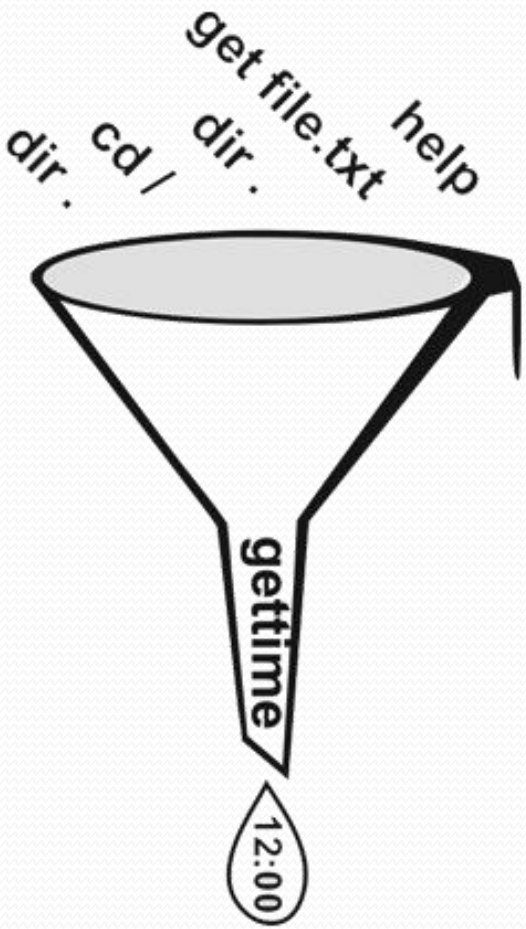
Традиционная реализация

Решение

Агент или рабочий элемент принимает очередную строку, распознаёт команду, выполняет и отправляет результат на выход.

Проблема

Результаты работы команд должны выводиться последовательно в порядке очереди поступления самих команд. Чтобы гарантировать сохранения порядка, необходимо синхронизировать вывод результатов. Это приводит к проблеме Lock Convoys - все потоки выстраиваются в очередь на одной взаимоисключающей блокировке.

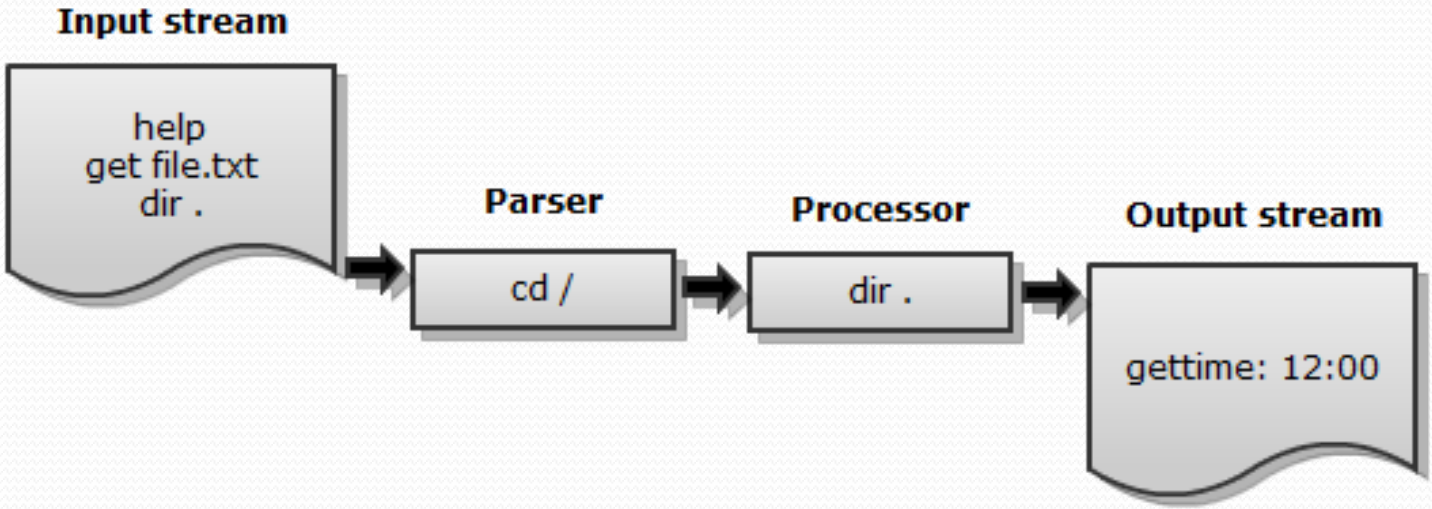


Data Flow

- Другое название: конвейер обработки
- Крупная задача со сложными зависимостями разбивается на мелкие
  - подзадачи зависят друг от друга только входными данными
  - подзадачи выстраиваются в конвейер
  - каждая подзадача запускается только при поступлении на вход очередной порции данных
  - результат обработки передаётся следующему узлу

Реализация в виде конвейера

- *узел-читатель* – считывает очередную строку;
- *узел-парсер* – выполняет разбор строки и формирует объект-команду;
- *узел-исполнитель* – исполняет команду, поступившую ему на вход в виде объекта, и отправляет результат дальше по конвейеру;
- *узел-писатель* – записывает в выходной поток результат, поступивший ему от предыдущего узла.



Подведём итог

- Без параллельного программирования в современном ИТ делать нечего
- Параллельное программирование не так сложно как кажется, если понимать что и как делать
- Сети Петри и MPI используются только в институтах ☺, реальность чуть-чуть иная

Вопросы

Материалы к лекции

- http://mrnone.blogspot.com/search/label/Multithreading

Спасибо

- Алексей Коротаев
  - Twitter: @mister_none
  - Blog: http://mrnone.blogspot.com