# Событийно-ориентированные архитектуры

Программирование с использованием POSIX thread library

2006-2007 Иртегов Д.В.

Учебное пособие подготовлено по заказу и при поддержке

ООО «Сан Майкросистемс СПБ»

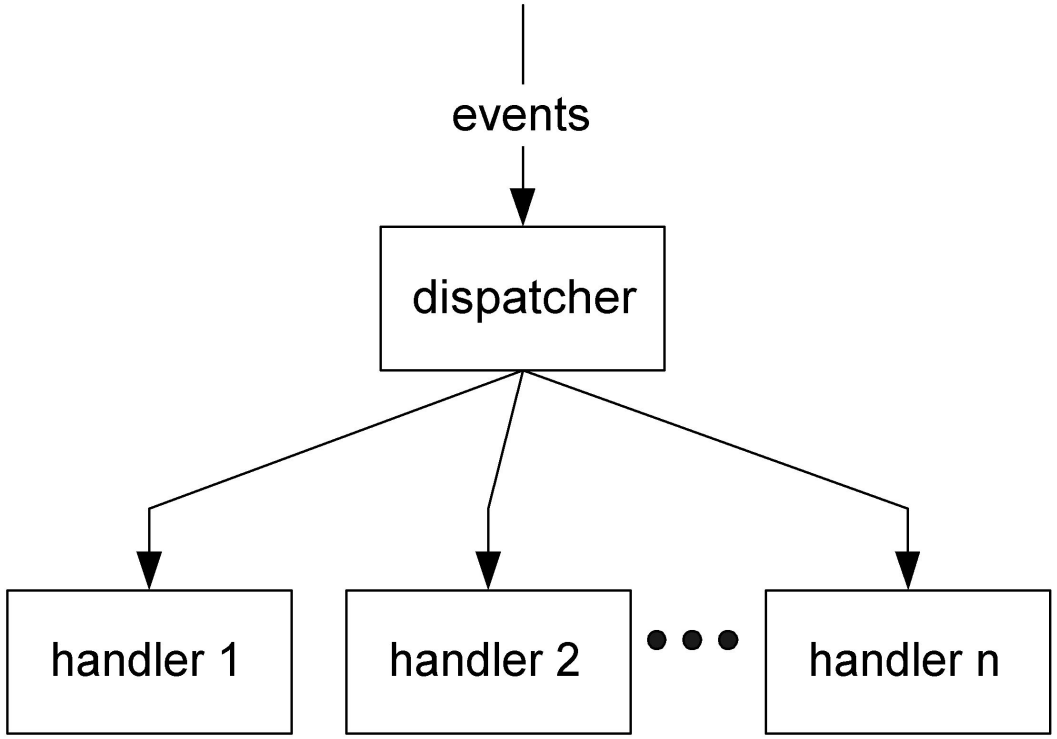
# Событийно-ориентированные архитектуры

- Графические пользовательские интерфейсы
- Сетевые серверные приложения
- Веб-приложения
- Ядра операционных систем
- Приложения жесткого реального времени

# Что такое событийно-ориентированная архитектура

S. Ferg
http://eventdrivenpgm.sourceforge.net/

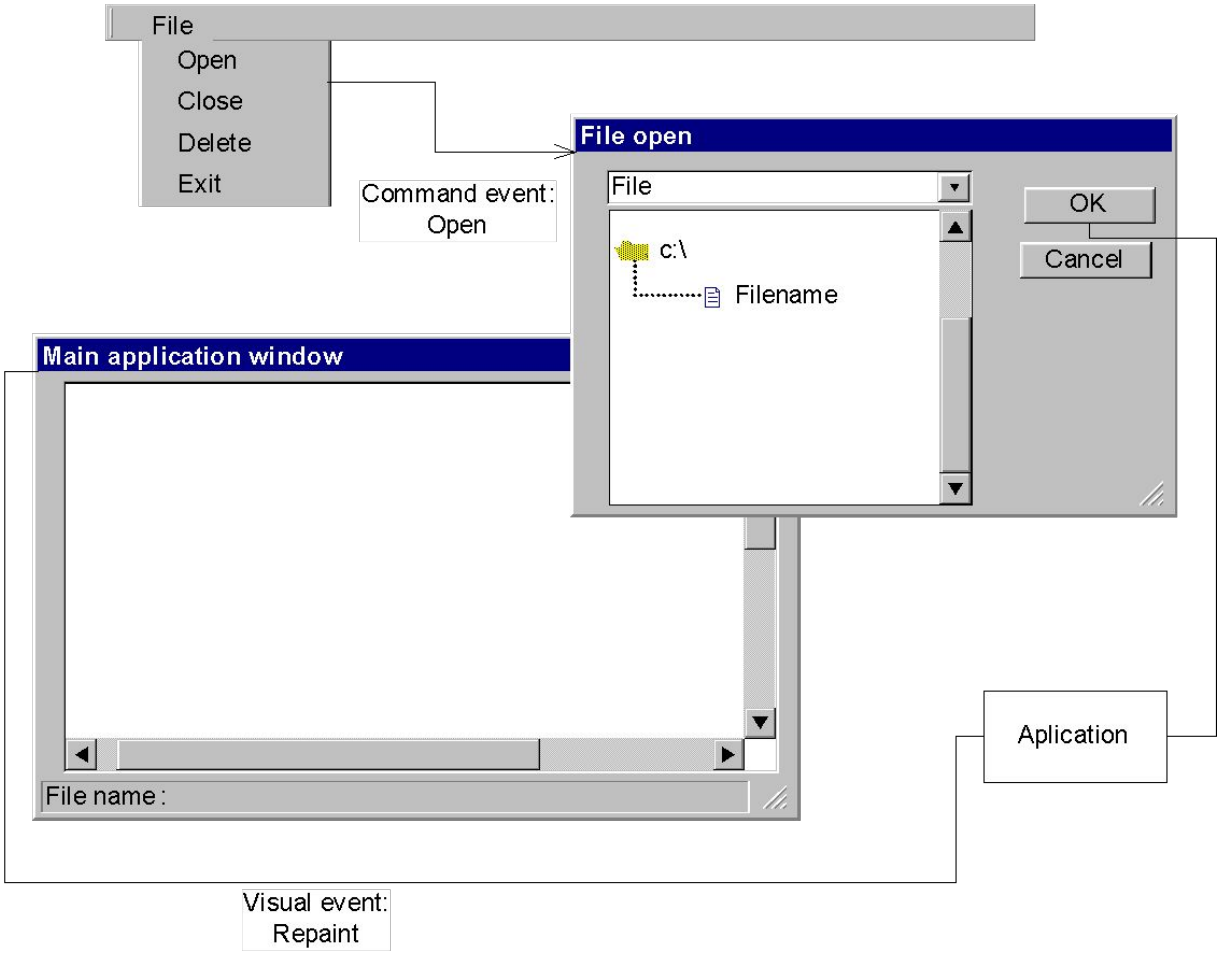
**Event-Driven Programming: Introduction, Tutorial, History**



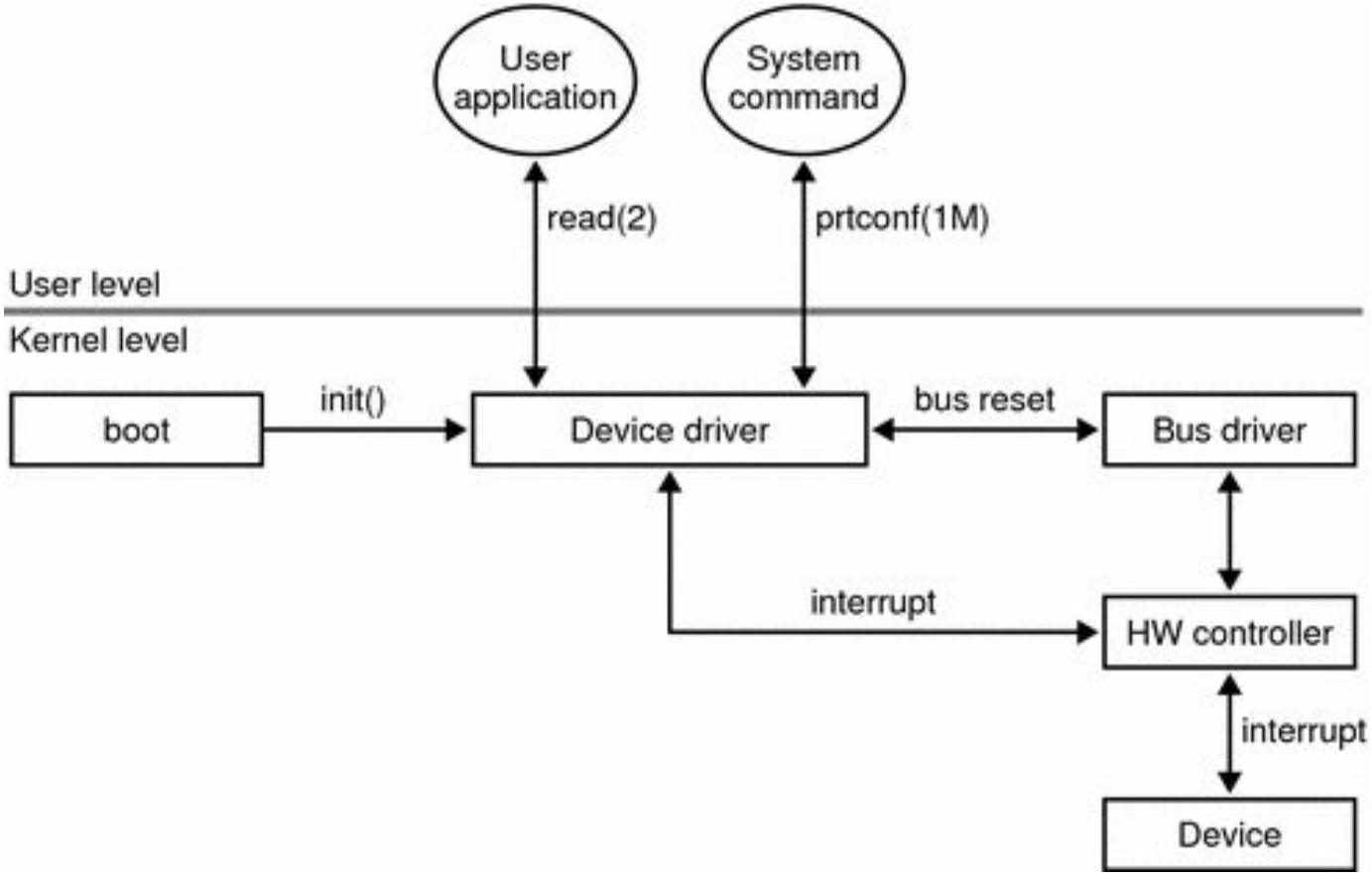
# Голливудский принцип

- Don’t call us, we will call you
  - (не звоните нам, мы сами вам позвоним)
- Среда (framework) вызывает ваши компоненты, а не вы вызываете библиотеку.
- Вы регистрируете обработчики, framework вызывает их.
- Каждый обработчик – конечный автомат (обработка события приводит к изменению переменных состояния)

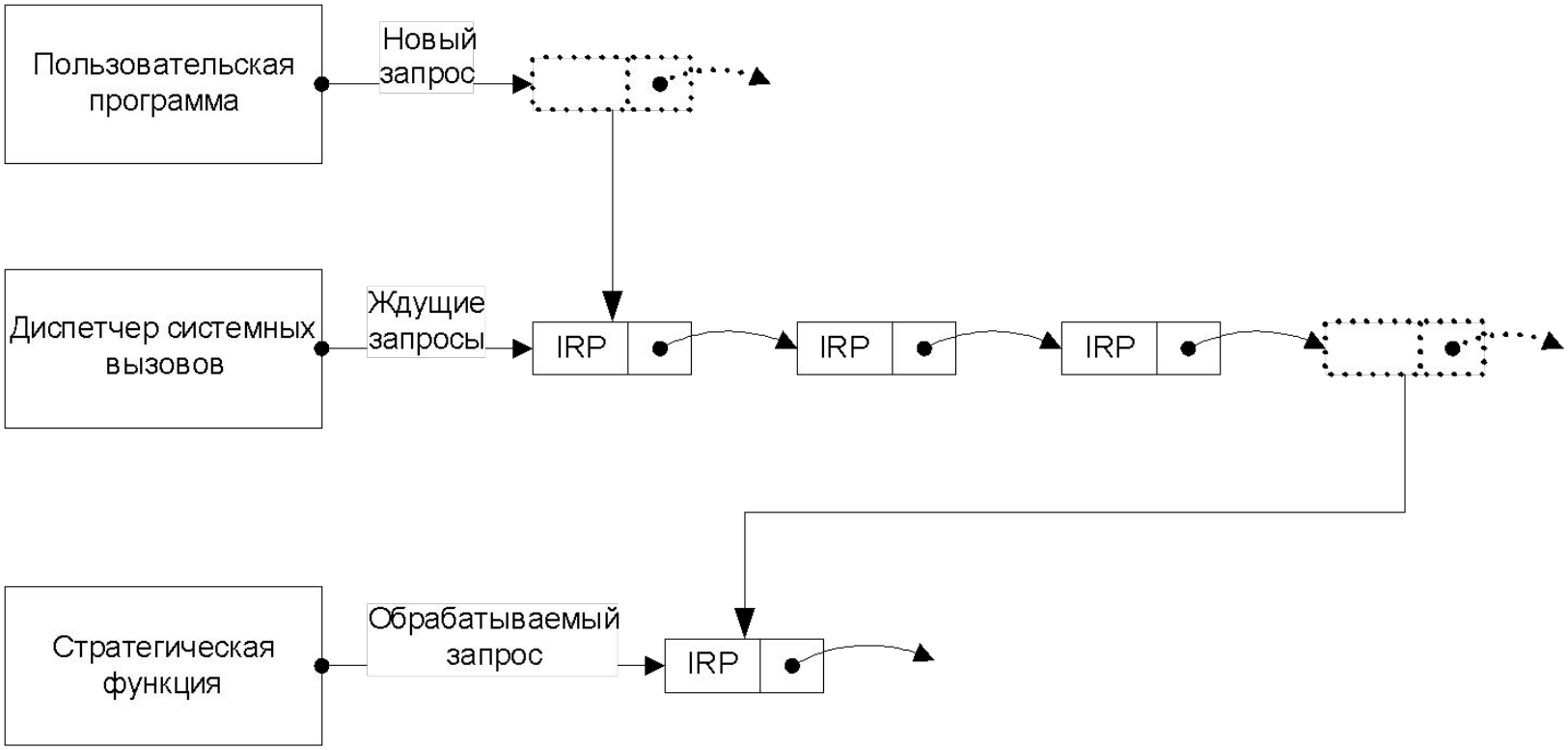
# Графические пользовательские интерфейсы



# Драйверы устройств



# Обработка запросов блочным или STREAMS устройством



# Диаграмма активностей

# Обработка событий и потоки

- Потоки
  - При ожидании события сохраняется контекст потока
  - Поток не знает, что его прерывали
- Обработка событий
  - Обработчик завершается после обработки каждого события
  - Обработчик знает, что он прерывается
  - Нет необходимости сохранять контекст
  - Контекст можно переиспользовать для других обработчиков

# Менеджер событий

- Часть framework
- Это он вызывает обработчики
- Он может блокироваться, ожидая событий
- Или может устанавливать события в очередь
- Может быть однопоточным (синхронная очередь событий)
- Или многопоточным (асинхронная очередь событий)

# Преимущества событийно-ориентированной архитектуры

- Можно обрабатывать много событий в одном потоке
- Потоки создавать дорого (мегабайт стека + LWP в ядре + системные структуры данных …)
- Можно получить значительную часть преимуществ многопоточности в однопоточной программе

# Недостатки событийно-ориентированной архитектуры

- Код обработчика событий не должен
  - делать длинных вычислений
  - вызывать блокирующиеся системные вызовы
- Код, рассчитанный на другие архитектуры, нуждается в переделке
  - либо его надо запускать в отдельных нитях

# Событийно-ориентированная архитектура сетевого сервера

- Приложение ориентированное на ввод/вывод
- Менеджер событий на select/poll
  - Событие – готовность дескриптора к чтению или записи
- Менеджер событий на aio
  - Событие – завершение предыдущей операции чтеня или записи
- Solaris ports можно использовать для диспетчеризации событий обоих типов

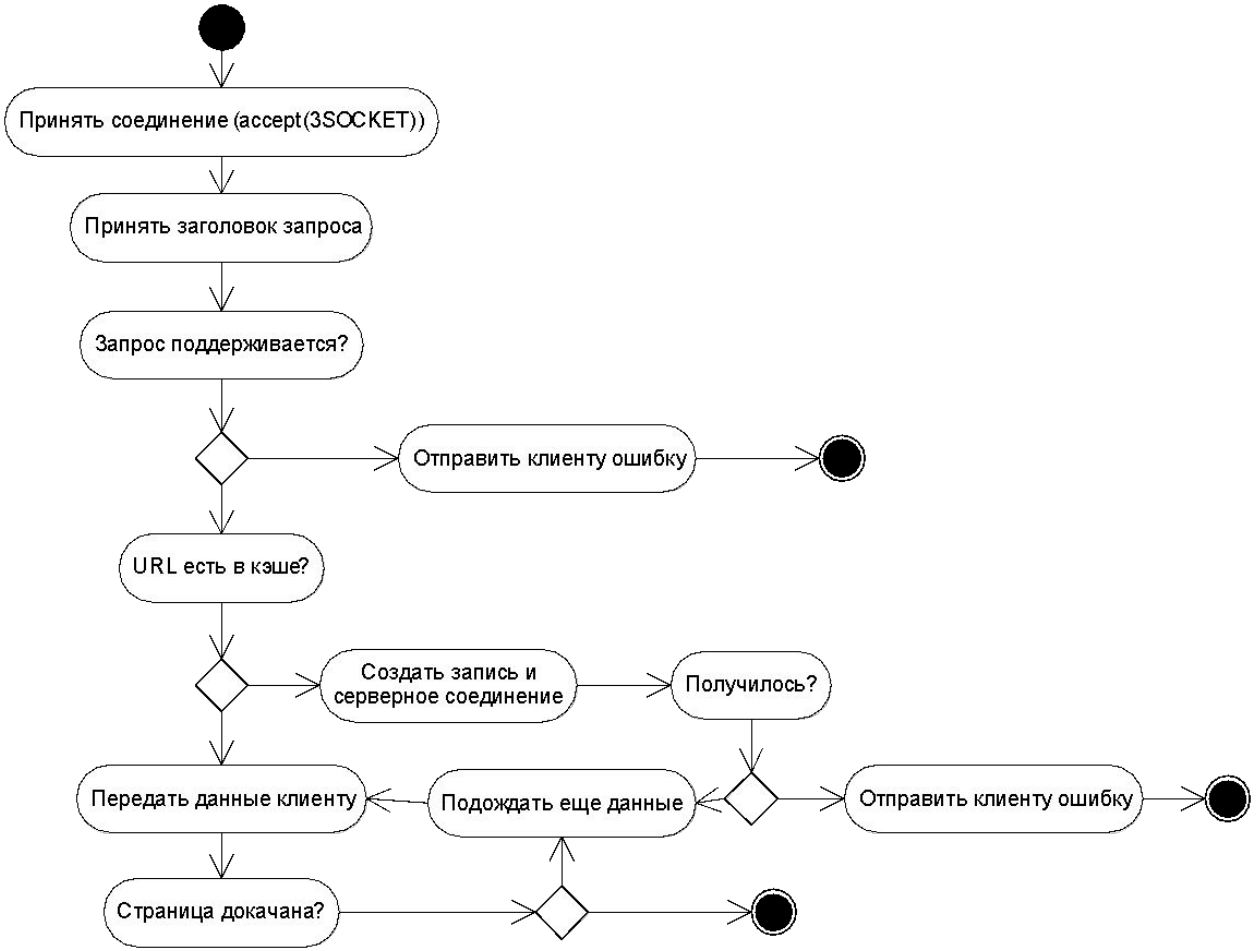
# Реализация кэширующего прокси

- Очень простой прокси
  - Кэш в ОЗУ (теряется при рестарте)
  - Поддерживает только GET и HEAD
  - Кэширует все подряд
- Предполагается, что вы знаете HTTP
- Рассмотрим диаграмму активностей по обработке одного запроса
- Будем реализовать обработку запроса двумя обработчиками – клиентского соединения и серверного соединения

# Клиентское соединение (последовательность событий)

- Ждем входящего соединения
- Получаем его
- Получаем строку запроса
  GET http://www.lenta.ru HTTP/1.0
- Поддерживается ли тип запроса?
- Есть ли страница в кэше? (если нет – надо создать)
- Докачана ли страница в кэше или нет?
- Передать страницу из кэша клиенту
- Если страница не докачана, надо подождать, пока не появятся еще данные.

# Картинка


Принять соединение (accept(3SOCKET))
Принять заголовок запроса
Запрос поддерживается?
Отправить клиенту ошибку
URL есть в кэше?
Создать запись и серверное соединение
Получилось?
Передать данные клиенту
Подождать еще данные
Отправить клиенту ошибку
Страница докачана?

# Серверное соединение (последовательность событий)

- Создается по запросу клиентского соединения
- Найти IP-адрес веб-сервера
- Установить соединение
- Вернуть ошибку, если предыдущие два шага не получились
- Переслать запрос
- Принимать данные
- Оповещать клиентов, когда приходят новые данные
- По закрытию соединения, пометить страницу в кэше как докачанную

# Надо ли анализировать заголовок?

- Настоящий прокси должен анализировать запрос
  - Cache-control: no-cache
- И ответ
  - Pragma: No-cache
  - Expires
- Наш игрушечный прокси будет кэшировать все подряд

# Что делать, если клиент закроет соединение?

- Оповещать серверное соединение
- Но если есть еще клиенты, работающие с этой страницей, серверное соединение завершать не следует
- Варианты:
  1. Серверное соединение должно считать, сколько клиентов работает с этой страницей
  2. Серверное соединение в любом случае должно докачивать страницу до конца
- Второй вариант явно проще

# Это еще не все

- http://debian.nsu.ru/debian-cd/current/i386/iso-dvd/

**Index of /debian-cd/current/i386/iso-dvd/**

| Name | Last Modified | Size | Type |
|---|---|---|---|
| Parent Directory/ | | - | Directory |
| debian-40r0-i386-DVD-1.iso | 2007-Apr-23 10:07:09 | 4.3G | application/octet-stream |
| debian-40r0-i386-DVD-2.iso | 2007-Apr-23 10:10:47 | 4.3G | application/octet-stream |
| debian-40r0-i386-DVD-3.iso | 2007-Apr-23 10:14:35 | 4.2G | application/octet-stream |

- Под .iso на 32-разрядной платформе вам никогда не дадут памяти
- На 64-разрядной платформе может не хватить swap-пространства или квоты памяти

# Что делать, если не хватает памяти под страницу?

- Надо прекращать кэширование и переходить в «сквозной» режим
- Если сервер хороший, он передаст вам размер страницы в поле заголовка Content-Length
- Но сервера HTTP/1.0 имеют право этого не делать
- Надо уметь переключаться в «сквозной» режим на ходу
- Но что делать, если с этой страницей работает несколько клиентов?

# Что делать, если не хватает памяти под страницу?

- Первые три студента в группе, которые будут сдавать прокси, *не будут* тестироваться на сайтах с большими объектами

# Еще рекомендации

- Стандартного API для асинхронных обращений к DNS нет
  - Допускается блокировка на gethostbyname
- Существует стандартный API асинхронного connect(3SOCKET)
  - fcntl(socket, F_SETFL, O_NONBLOCK);
- Для стресс-тестирования можно использовать wget(1)