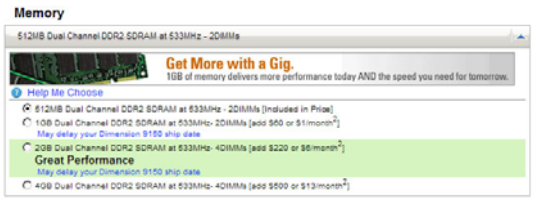
## Лекция 2: Поддержка операционных систем со стороны аппаратного обеспечения

Алексей Линёв
Александр Мощук
Кирилл Погорельский

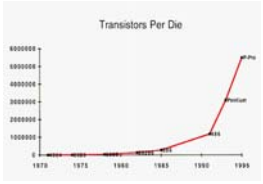
# Компьютерная Архитектура и ОС

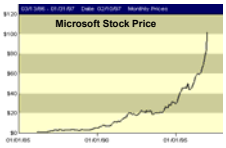
- Даже незначительные тенденции в развитии архитектуры критично влияют на структуру ОС
- Вычислительная мощность
  - Удваивается каждые 18 месяцев
  - увеличивается на 60% в год
  - Увеличивается в 100 раз за 10 лет
  - 1980:
    - 1 MHz Apple II+ == $2,000
    - 1 MIPS VAX-11/780 == $120,000
  - 2006:
    - 3.0GHz Pentium 4 == $800


- Объем оперативной памяти
  - та же история, по той же причине (закон Мура)
    - 1972 (основная память): 1 Мб = $1,000,000
  - сейчас:




- Отступление: куда все это уходит?
  - Афоризм: “Гордон дал – Билл забрал”
  - Практически: наши ожидания о возможностях систем непрерывно увеличиваются
    - например, возможности графических интерфейсов (GUI)
  - "Программное обеспечение как газ – оно расширяется до полного заполнения доступного пространства" – Натан Мирвольд (1960-…)




- Ёмкость жесткого диска, 1975-1989
  - удваивается каждые 3+ года
  - 25% увеличение в год
  - 10-кратное увеличение за 10 лет
  - экспоненциальный рост, но гораздо более медленный, чем рост производительности ЦП
- Ёмкость жесткого диска с 1990 г.
  - удвоение каждые 12 месяцев
  - 100% увеличение в год
  - 1000-кратное увеличение за 10 лет
  - растет в 10 раз быстрее, чем производительность ЦП
  - *[но скорость доступа увеличивается только на 10% в год]*

- Буквально несколько лет назад стоимость диска рассчитывалась исходя из его объема в мегабайтах (и это была наша боль!)
- Сейчас **1 Гб** (миллиард байтов) стоит **$0.50** => 1 Тб стоит $500, 1 Пб стоит $500К
- Через 2 года 1 Гб будет стоить $0.10
  - 1 Тб будет стоить $100, 1 Пб – $100К

- Пропускная способность оптоволоконных каналов
  - *удваивается каждые 9 месяцев*
  - ежегодное увеличение – 150%
  - увеличение в течение 10 лет – в 10,000 раз
  - растет в 10 раз быстрее, чем объем жестких дисков!
  - растет в 100 раз быстрее, чем производительность ЦП!!
- Какие выводы можно сделать?
  - Всего лишь один пример:
    Мы привыкли проектировать системы таким образом, чтобы они расходовали дополнительную вычислительную мощность, чтобы минимизировать “дефицитное” хранение на жестких дисках и “медленные” сетевые соединения.

Archive

The New York Times

October 22, 2003, Wednesday

BUSINESS/FINANCIAL DESK

TECHNOLOGY; Low-Cost Supercomputer Put Together From 1,100 PC's

By JOHN MARKOFF (NYT) 649 words

SAN FRANCISCO, Oct. 21 -- A home-brew supercomputer, assembled from off-the-shelf personal computers in just one month at a cost of slightly more than $5 million, is about to be ranked as one of the fastest machines in the world.

Word of the low-cost supercomputer, put together by faculty, technicians and students at Virginia Polytechnic Institute, is shaking up the esoteric world of high-performance computing, where the fastest machines have traditionally cost from $100 million to $250 million and taken several years to build.

The Virginia Tech supercomputer, put together from 1,100 Apple Macintosh computers, has been successfully tested in recent days, according to Jack Dongarra, a University of Tennessee computer scientist who maintains a listing of the world's 500 fastest machines.

The official results for the ranking will not be reported until next month at a supercomputer industry event. But the Apple-based supercomputer, which is powered by 2,200 I.B.M. microprocessors, was able to compute at 7.41 trillion operations a second, a speed surpassed by only three other ultra-fast computers.

Archive

The New York Times

May 26, 2003, Monday

BUSINESS/FINANCIAL DESK

TECHNOLOGY; From PlayStation to Supercomputer for $50,000

By JOHN MARKOFF (NYT) 913 words

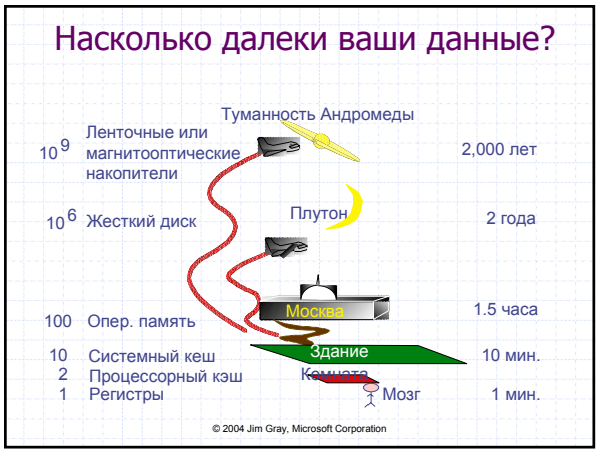
As perhaps the clearest evidence yet of the computing power of sophisticated but inexpensive video-game consoles, the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign has assembled a supercomputer from an array of Sony PlayStation 2's.

The resulting system, with components purchased at retail prices, cost a little more than $50,000. The center's researchers believe the system may be capable of a half trillion operations a second, well within the definition of supercomputer, although it may not rank among the world's 500 fastest supercomputers.

Perhaps the most striking aspect of the project, which uses the open source Linux operating system, is that the only hardware engineering involved was placing 70 of the individual game machines in a rack and plugging them together with a high-speed Hewlett-Packard network switch. The center's scientists bought 100 machines, but are holding 30 in reserve, possibly for high-resolution display application.

"It took a lot of time because you have to cut all of these things out of the plastic packaging," said Craig Steffen, a senior research scientist at the center, who is one of four scientists working part time on the project.

The scientists are taking advantage of a standard component of the Sony video-game console that was originally intended to move and transform pixels rapidly on a television screen to produce lifelike graphics. The chip is not the PlayStation 2's MIPS microprocessor, but rather a graphics co-processor known as the Emotion Engine. That custom designed silicon chip is capable of producing up to 6.5 billion mathematical operations a second.




## Низкоуровневая архитектура влияет на ОС в еще большей степени

- Функционирование операционной системы зависит, по крайней мере частично, от нижележащей аппаратной архитектуры
  - набор инструкций (синхронизация, ввод/вывод, …)
  - и аппаратные компоненты, такие как MMU или контроллеры DMA
- Поддержка со стороны архитектуры может чрезвычайно упростить (или усложнить!) задачи ОС
  - например, в ранних ОС для ПК (DOS, MacOS) отсутствовала поддержка виртуальной памяти по причине отсутствия необходимой поддержки со стороны аппаратного обеспечения
  - В большинстве современных Intel-совместимых ПК все еще отсутствует поддержка 64-битной адресации
    - что уже в течении десятилетия доступно для других платформ: MIPS, Alpha, IBM,…
    - быстро изменяется, отчасти благодаря 64-битной архитектуре от AMD

## Архитектурные особенности, влияющие на ОС

- Следующие возможности появились в первую очередь для поддержки функционирования ОС:
  - таймеры
  - инструкции поддержки синхронизации (например, test-and-set)
  - защита памяти
  - управление/контроль ввода/вывода
  - прерывания (interrupts) и исключения (exceptions)
  - уровни привилегий (ядро / пользователь)
  - защищенные инструкции
  - системные вызовы (и программные прерывания)
- 2006: архитектуры виртуализации
  - Intel VT, AMD Pacifica

# Защищенные инструкции

- часть инструкций может выполнять только ОС
  - они называются *защищенными* или *привилегированными инструкциями*
- например, только ОС может:
  - непосредственно управлять устройствами ввода-вывода (диски, сетевые адаптеры,…)
    - причина?
  - манипулировать параметрами управления памятью
    - таблицы страниц, TLB и т.д.
    - причина?
  - изменение "битов режима/состояния"
    - уровень приоритета прерывания
    - причина?
  - инструкция halt
    - причина?

# Защита ОС

- Как центральный процессор определяет, нужно ли выполнять привилегированную инструкцию?
  - архитектура должна поддерживать по крайней мере два режима выполнения (уровня привилегий): режим/уровень *ядра* и режим/уровень *пользователя* (*kernel* mode and *user* mode)
    - VAX, x86 поддерживают 4 уровня привилегий
  - режим устанавливается изменением бита в защищенном регистре процессора
    - пользовательские (прикладные) программы выполняются в пользовательском режиме
    - ОС выполняется в режиме ядра (ОС == ядро)
- Защищенные инструкции могут быть выполнены только в режиме ядра
  - что произойдет, если в пользовательском режиме выполнить защищенную инструкцию?

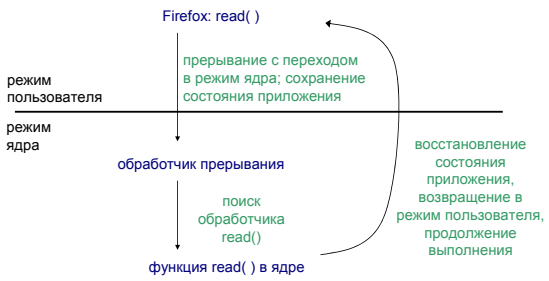
# Переключение между режимами выполнения

- Каким же образом программы пользовательского уровня выполняют привилегированные операции?
  - Как записать что-либо на жесткий диск, если запрещено выполнять инструкции ввода-вывода?
- Прикладные программы должны вызывать функции ОС
  - в ОС определено множество *системных вызовов* (*system calls*)

# Переключение между режимами выполнения

- Как происходит переход из пользовательского режима в режим ядра?
  - Должна быть инструкция, выполняющая системный вызов, которая:
    - вызывает исключение (инициирует программное прерывание), обработчик которого расположен в ядре
    - передает параметр, указывающий, какой именно системный вызов выполнить
    - сохраняет состояние вызывающего кода (значения регистров, флаги, биты режима) для последующего восстановления
    - ОС должна проверить корректность переданных аргументов (например, указателей)
    - должна иметься возможность возврата в пользовательский режим при завершении обработки

# Иллюстрация переключения между режимами

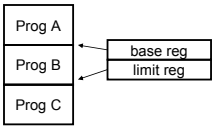



# Замечания на тему системных вызовов

- Что произойдет, если ядро не сохранит состояние вызывающего кода?
- Почему ядро должно проверять корректность переданных аргументов?
- Как ссылаться на объекты ядра при их использовании в качестве аргументов системного вызова или возвращаемых значений?

# Защита памяти

- ОС должна защищать пользовательские приложения друг от друга
  - приложения-злоумышленники и приложения с ошибками
- ОС также должна защищать себя от прикладных программ
  - целостность и безопасность
  - а как насчет защиты прикладных программ от ОС?
- Простейшая схема: регистры *начала* (*base*) и *размера* (*limit*)
  - все защищено?




регистры начала (base) и размера (limit) устанавливаются ОС перед запуском приложения

# Более сложные методы защиты памяти

- Рассматриваются в курсе немного позже
- Страничное преобразование, сегментная адресация, виртуальная память
  - таблицы страниц
  - translation look-aside buffers (TLBs) – буферы быстрого преобразования адреса
  - обработка страничных сбоев (page fault handling)

# Принцип функционирования ОС

- По окончании загрузки все переключения в режим ядра происходят в результате возникновения *событий (events)*
  - событие немедленно прерывает исполняющийся код
  - происходит переключение в режим ядра и вызов обработчика события (event handler)

# Принцип функционирования ОС

- Ядро устанавливает обработчики для всех типов событий
  - множество событий определяется архитектурой ЭВМ
    - например, таймер, прерывания ввода-вывода, системные вызовы
  - когда ЦП получает событие определенного типа, он
    - передает управление обработчику ядра
    - обработчик сохраняет состояние прерванной программы (программный счетчик, регистры и пр.)
    - обработчик выполняет функциональную обработку события
    - обработчик восстанавливает состояние прерванной программы и возвращает ей управление

# Прерывания и исключения

- Существуют два основных типа событий: *прерывания (interrupts)* и *исключения (exceptions)*
  - исключения происходят при выполнении программами инструкций
    - инструкция 'int' в x86
    - при страничном сбое или попытке записи в страницу, для которой разрешено только чтение
    - ожидаемое исключение называется "ловушка" (trap), непредвиденное – сбой/ошибка (fault)
  - прерывания генерируются аппаратными устройствами
    - устройствами, завершившими ввод-вывод
    - таймером

# Управление вводом-выводом

- Вопросы:
  - как ядро инициирует ввод-вывод?
    - специальные инструкции ввода-вывода
    - ввод-вывод с отображением в оперативную память
  - как ядро узнает о завершении ввода-вывода?
    - опрос состояния (polling)
    - прерывания
- Прерывания являются основой асинхронного ввода-вывода
  - устройство выполняет свою работу параллельно с ЦП
  - устройство по завершении операции отправляет по шине сигнал прерывания
  - *вектор прерываний* (*vector table*) в оперативной памяти содержит адреса функций ядра – обработчиков прерываний различных типов
    - кто заполняет вектор прерываний? когда?
  - по номеру возникшего прерывания из вектора прерываний берется адрес обработчика, и ЦП переключается на его выполнение

# Таймеры

- Как ОС может не допустить безудержного роста потребления ЦП вышедшими из под контроля прикладными программами (например, с бесконечным циклом)?
  - использовать аппаратный таймер, периодически генерирующий прерывания
  - перед передачей ЦП прикладной программе, ОС запускает таймер с заданным временем прерывания
    - "квант" ("quantum") – насколько он должен быть велик?
  - когда срабатывает таймер, прерывание возвращает управление ОС
    - в этот момент ОС должна решить, какой программе теперь предоставить центральный процессор
    - крайне интересен способ выбора программы – мы посвятим этому вопросу одну из лекций
- Должен ли таймер быть привилегированным объектом?
  - для чтения или для записи?

# Синхронизация

- Прерывания являются причиной некоторого затруднения:
  - они возникают в любой момент, что может привести к выполнению кода обработчика, влияющего на исполнение прерванного кода
  - ОС должна иметь механизмы *синхронизации* параллельно выполняющихся процессов
- Синхронизация
  - гарантирует атомарное выполнение коротких последовательностей инструкций (например, считать-изменить-записать)
  - один из способов: запретить прерывания перед выполнением последовательности инструкций, по окончании – разрешить
    - архитектура должна поддерживать запрет прерываний
  - другой способ: использование специальных атомарных инструкций
    - read-modify-write (чтение-изменение-запись)
    - test-and-set (проверка-и-установка)
    - load-linked store-conditional (загрузка с сохранением при выполнении условия)

# Параллельное программирование

- Управление параллельным выполнением и асинхронными событиями – наибольшее различие между "системным программированием" и "традиционной разработкой приложений"
  - современная разработка "событийно-ориентированнных" приложений занимает промежуточное положение
- Является результатом развития архитектуры ЭВМ
- Невозможно игнорировать, но возможно облегчить
- Серьезная и сложная задача
  - в отличие от уязвимостей, связанных с переполнением буфера, являющихся следствием неаккуратного программирования