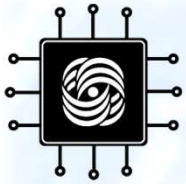
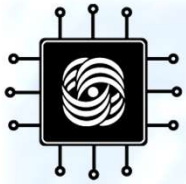


Имитационное моделирование в исследовании и разработке вычислительных систем и сетей

Лекция 1

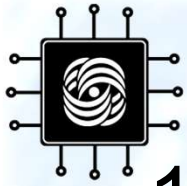
Кафедра АСВК,
лаборатория Вычислительных комплексов
в.н.с. Бахмуров Анатолий Геннадьевич

Москва 2019



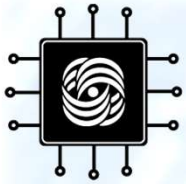
Формальные сведения

- Обязательный спецкурс для 3 курса АСВК
- 72 часа – лекции и самостоятельная работа (приблизительно 12-13 лекций)
- 4 «больших» практических задания
- «малые» домашние задания и (?) контрольные работы
- Предварительный письменный экзамен
- Устный экзамен
- Возможна досрочная сдача и «автомат»



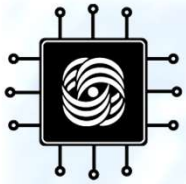
Основные части курса

1. Вычислительные системы (ВС) как объект разработки и исследования
2. Получение характеристик ВС методом наблюдения за их работой
3. Получение характеристик ВС методами моделирования
4. Построение имитационной модели
5. Исследование имитационной модели
6. Особенности имитационного моделирования аппаратных средств



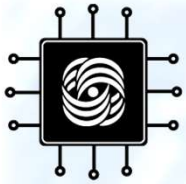
Понятие вычислительной системы

Вычислительная система (ВС) == аппаратные средства + системное ПО + прикладное ПО



Аппаратные средства

- мобильные компьютеры
- персональные компьютеры
- серверы
- системы хранения данных
- сетевые устройства
- суперкомпьютеры
- центры обработки данных (ЦОД)
- специализированные компьютеры

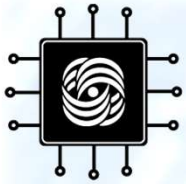


Производительность

- **Пропускная способность** = количество удовлетворённых запросов в единицу времени

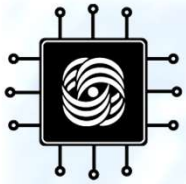
Или: объём вычислительной работы в единицу времени

- **Время отклика** = время между получением запроса и выдачей ответа



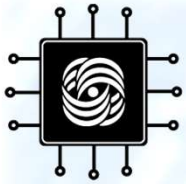
Единицы измерения производительности

- MIPS (GIPS)
- MFLOPS (GFLOPS)
- FPS
- Гб/с
- Пакеты в секунду
- IOPS
- рейтинги тестовых программ (bencmarks)



Программы для измерения производительности (примеры)

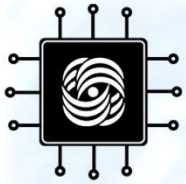
- Standard Performance Evaluation Corporation (SPEC) http://spec.org/
- UL benchmarks: PCMark, 3DMark, VRMark
- LINPACK benchmark
- Iometer – для дисков
- SPC-1, SPC-2 (spcresults.org) – для систем хранения данных
- TPC (www.tpc.org) – для СУБД



Ресурсы и рабочая нагрузка

ВС предоставляет пользователю набор *ресурсов*

ВС исполняет поток запросов (обращений к ресурсам), называемый *рабочей нагрузкой* (workload)

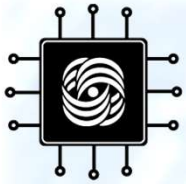


Уровни рассмотрения ВС и примеры рабочей нагрузки

Совокупность вычислительных узлов

Ресурсы: узел, память, канал, приложение …

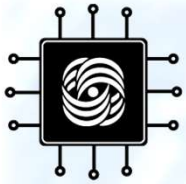
Рабочая нагрузка: выполнение программы, обработка потока запросов.



Уровень общей архитектуры узла

Ресурсы: процессор, память, диск, видео, системный вызов, …

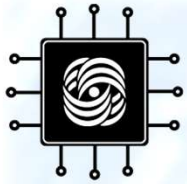
Рабочая нагрузка: выполнение процесса, системные вызовы, чтение-запись файла …



Уровень (микро)архитектуры

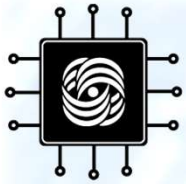
Ресурсы: регистры, ячейки памяти, функциональные устройства, кэш, буфер, …

Рабочая нагрузка: выполнение машинной команды, приём-передача пакета, чтение-запись ячейки памяти …



Для измерения производительности нужно:

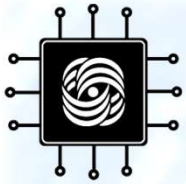
- Наблюдение за работой ВС: отслеживание изменений (также называемых событиями), происходящих в ВС;
- Измерение характеристик потока событий (прежде всего частоты, интервалы времени)
- Расчёт итоговых показателей производительности, интересующих исследователя



Наблюдатели (и измерители)

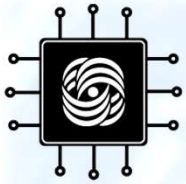
- Аппаратные:
  - встроенные в ВС;
  - внешние по отношению к ВС;
- Программные:
  - на уровне ОС;
  - на уровне типовых программных средств;
  - определяемые пользователем.

Встроенный аппаратный наблюдатель (1)



Отладочные регистры на x86

- 6 регистров: DR0-DR3,DR6,DR7
- Точки останова (4 шт.):
  - по чтению;
  - по записи и чтению;
  - по выполнению;
  - по обращению к порту ввода-вывода.
- Недостаток: Нельзя задать условие по диапазону адресов

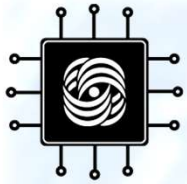


Intel Processor Tracing

Для современных процессоров Intel:

(с 5 поколения, Core i5)

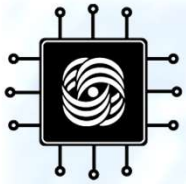
- Сбор трассы выполнения программы (в сжатом виде), до 5% накладных расходов
- Имеется средство для декодирования трассы
- Поддержка в утилите perf для Linux и в Vtune – коммерческом средстве от Intel.



Performance Monitoring Counters

- Для процессоров Intel, AMD, ARM, MIPS (количество счётчиков сильно различается для этих архитектур)
- Выполнение команд: число тактов; число команд; число правильно предсказанных переходов;
- Кэш: число обращений в кэш i-го уровня; число промахов -- для кэша команд и данных;
- MMU: число страничных промахов (page faults);
- Память: число тактов ожидания данных
- … (ссылки на полное описание см. в конце презентации)

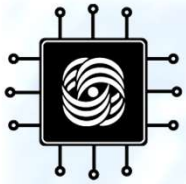
- Поддержка в ядре Linux. Утилита perf.



Встроенные отладочные средства (JTAG – Joint Test Action Group, IEEE 1149)

Набор специальных блоков в составе ИС (существенно зависит от типа ИС). Эти блоки управляются по интерфейсу JTAG.

- Установка заданных значений на выходах ИС и считывание входов (для тестирования соединения ИС с окружающей её схемой).
- Программирование ИС.
- Управление микропроцессором в целях отладки: точки останова, просмотр и запись регистров.



Отладка с помощью JTAG

- Практически все современные процессоры
- Требуется: разъём JTAG на плате, адаптер JTAG-USB, ПО отладчика.
- Для современных процессоров Intel возможно подключение через USB 3.0.
- OpenOCD – агент отладки для gdb с поддержкой JTAG

Универсальные наблюдатели

- Индикаторы (лампы, светодиоды, …)
- Электронный осциллограф
  - Непрерывные и дискретные сигналы
  - Обычно 1-4 канала
  - Непрерывная и ждущая развертка
  - Задержка запуска развертки
- Логический анализатор
  - Дискретный двоичный сигнал
  - Десятки-сотни каналов
  - Задание условий запуска и окончания регистрации

Анализаторы периферийных интерфейсов

VME, PCI,

PCI-Express,

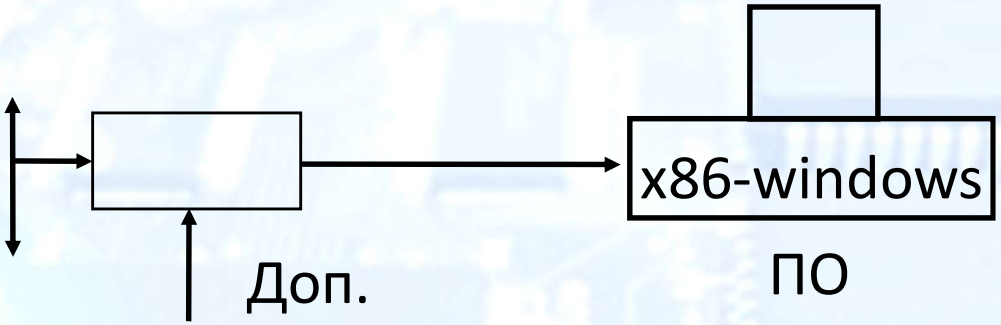
CANbus,

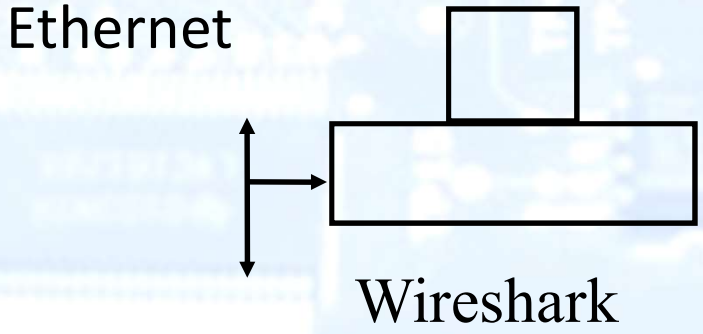
MIL-STD-1553B

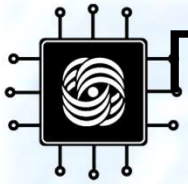
ARINC-429

ARINC-818

и т.д.





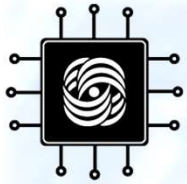


Программные наблюдатели: уровень операционной системы

Трассировка системных вызовов

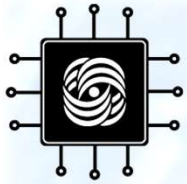
% strace /bin/cat

```
// Динамическая загрузка
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) =
3
read(3,
"\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\226\1\00
04\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1734120, ...}) = 0
mmap2(NULL, 1743580, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb75a0000
mmap2(0xb7744000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a4) = 0xb7744000
mmap2(0xb7747000, 10972, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7747000
close(3)                                    = 0
```



Вывод strace(2)

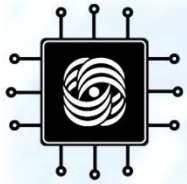
```
read(0, "fkjdfkjfd\n", 32768)    = 10
  write(1, "fkjdfkjfd\n", 10)    = 10
  read(0, "sdcdscdsv\n", 32768) = 10
  write(1, "sdcdscdsv\n", 10)    = 10
  read(0, "fbfb\n", 32768)          = 5
  write(1, "fbfb\n", 5)             = 5
  read(0, "", 32768)                = 0
  close(0)                          = 0
  close(1)                          = 0
  close(2)                          = 0
  exit_group(0)                     = ?
```



Трассировка библиотечных функций

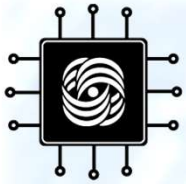
- % ltrace /bin/cat 2> ltrace.out

```
__libc_start_main(0x8049280, 1, 0xbfb66534, 0x80500d0, 0x8050140
<unfinished ...>
getpagesize()                                          = 4096
strrchr("/bin/cat", '/')                               = "/cat"
setlocale(6, "")                                       =
"en_US.UTF-8"
bindtextdomain("coreutils", "/usr/share/locale")       =
"/usr/share/locale"
textdomain("coreutils")                                =
"coreutils"
__cxa_atexit(0x804a2f0, 0, 0, 0xb7769ff4, 0)           = 0
getopt_long(1, 0xbfb66534, "benstuvAET", 0x08050840, NULL) = -1
__fxstat64(3, 1, 0xbfb6641c)                           = 0
__fxstat64(3, 0, 0xbfb6641c)                           = 0
```



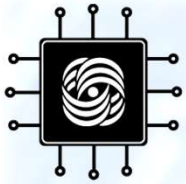
Трассировка библиотечных функций (2)

```
posix_fadvise64(0, 0, 0, 0, 0)                    = 0
malloc(36863)                                     = 0x097e0890
read(0, "qwerty\n", 32768)                        = 7
write(1, "qwerty\n", 7)                           = 7
read(0, "", 32768)                                = 0
free(0x097e0890)                                  = <void>
close(0)                                          = 0
exit(0 <unfinished ...>
```



Измерение времени в ОС (Linux)

- /usr/bin/time *myfile args*
  - real
  - user
  - sys
- Команда rdtsc в x86 – число тактов ЦП с момента запуска
- Системные вызовы time(), times()



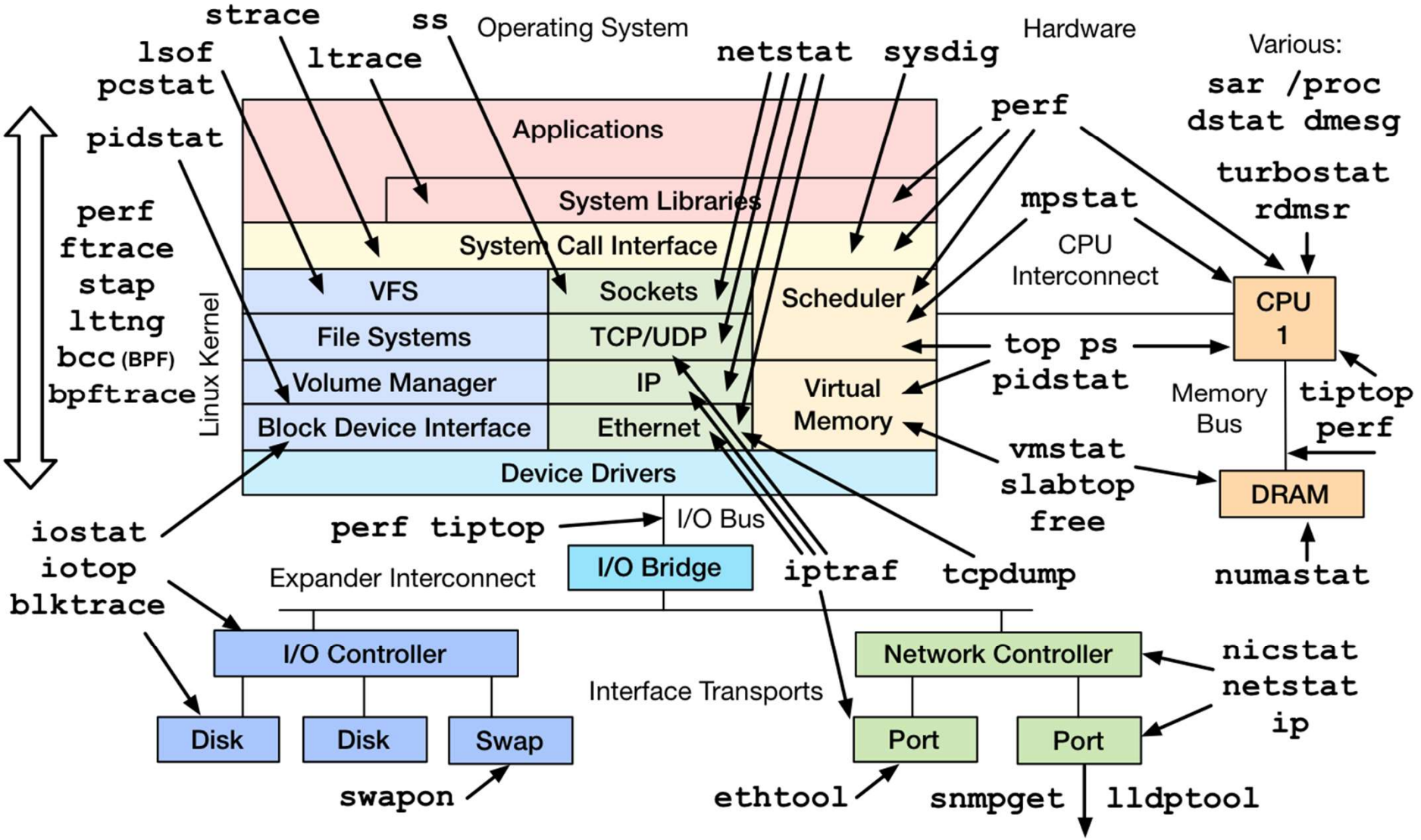
Средства наблюдения в Linux

Из блога Грега Брендана

Linux Performance

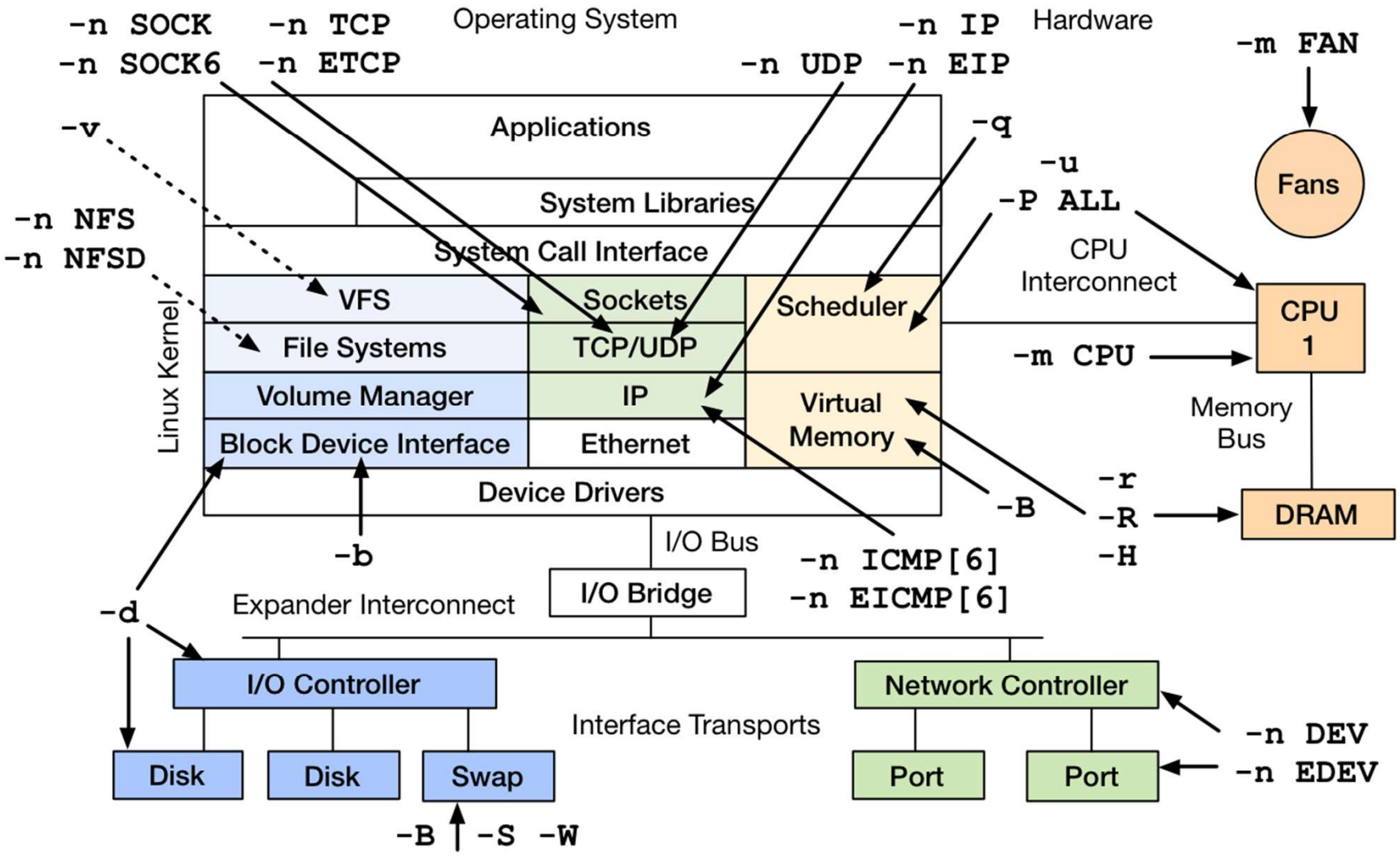
http://www.brendangregg.com/linuxperf.html

Linux Performance Observability Tools



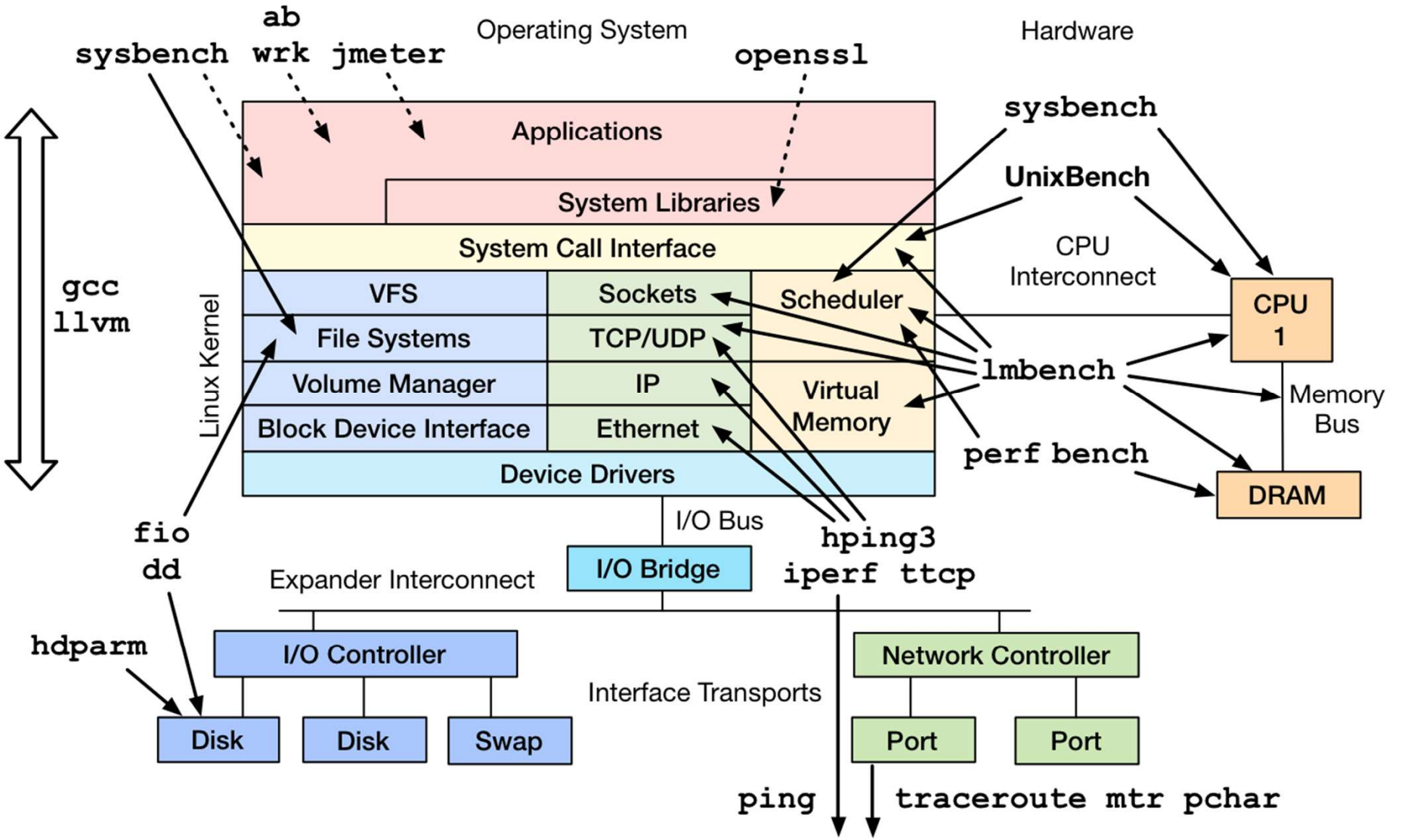
http://www.brendangregg.com/linuxperf.html 2018

Linux Performance Observability: sar



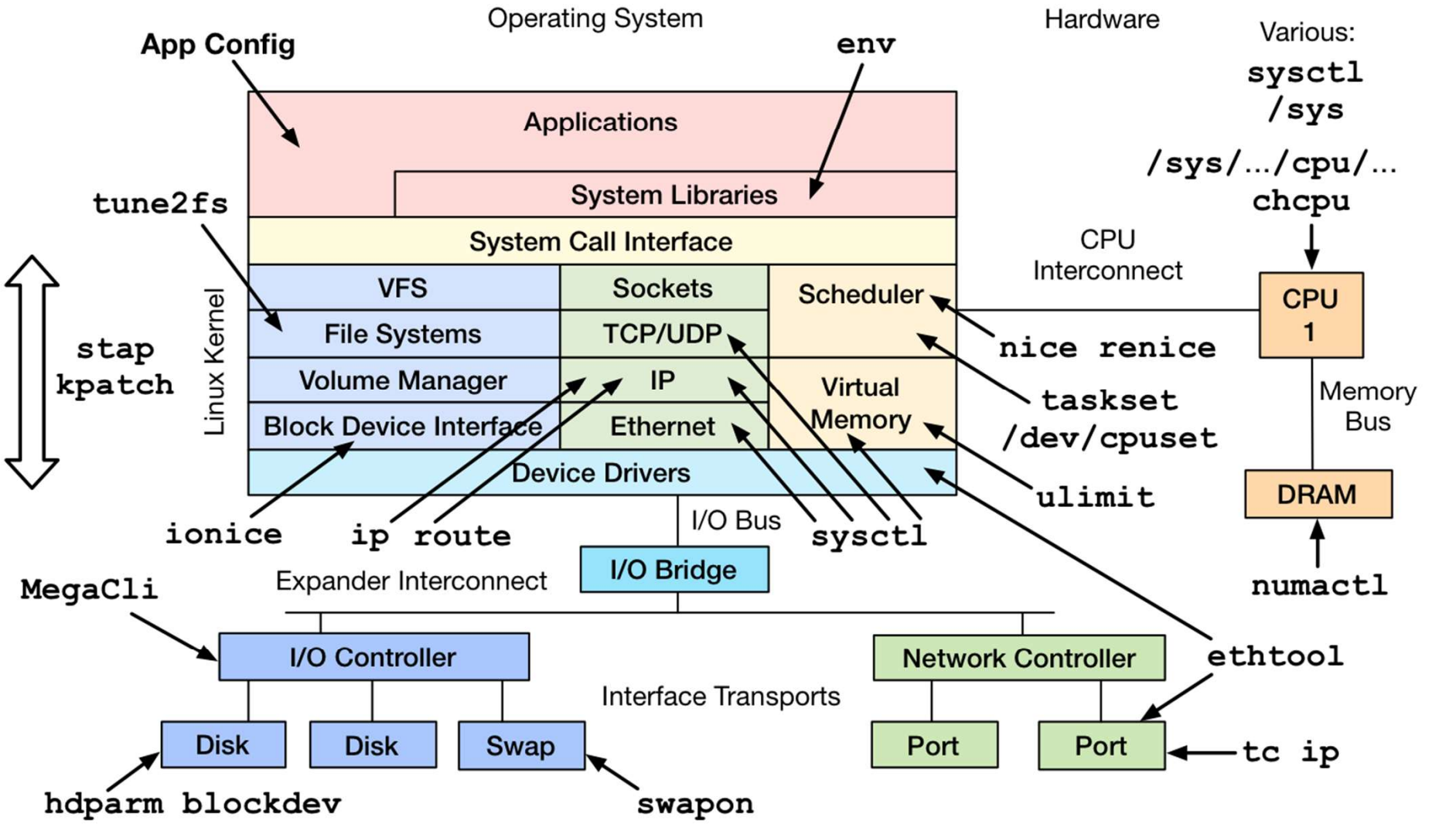
http://www.brendangregg.com/linuxperf.html 2016

Linux Performance Benchmark Tools

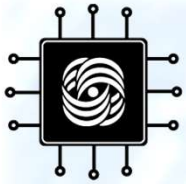


http://www.brendangregg.com/linuxperf.html 2016

Linux Performance Tuning Tools



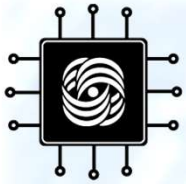
http://www.brendangregg.com/linuxperf.html 2016



Анализ производительности

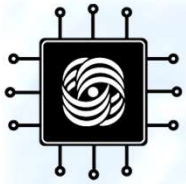
Выяснение факторов, наиболее сильно влияющих на производительность:

- Узкие места (bottlenecks) – компоненты, ограничивающие производительность системы в целом;
- «Горячие точки» – компоненты, потребляющие б**о**льшую часть ресурсов (например, время ЦП)



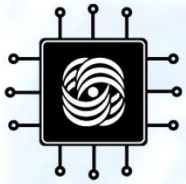
Синтез производительности

- Принятие решений, обеспечивающих требуемую производительность ВС, в том числе:
- настройка параметров;
- обновление компонентов:
  - замена на более производительные;
  - оптимизация (программ);
- изменение структуры ВС.



Профилирование программ

Определение использования ресурсов для всех элементов структуры программы (функций, иногда операторов языка программирования или машинных команд)



Профилирование программ: gprof

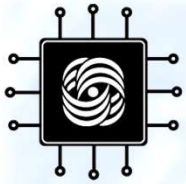
% gcc –pg a.c

% a.out *(создаётся gmon.out)*

% gprof

Выборка 100 раз в секунду

- Число вызовов функций
- Время, проведённое в каждой функции
- Граф вызовов



Профилирование программ: perf

Запись профиля

% perf record *команда аргументы*

% perf record –e pages *команда аргументы*

Отображение результатов

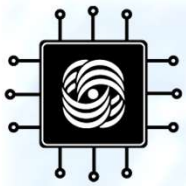
% perf report

Список событий

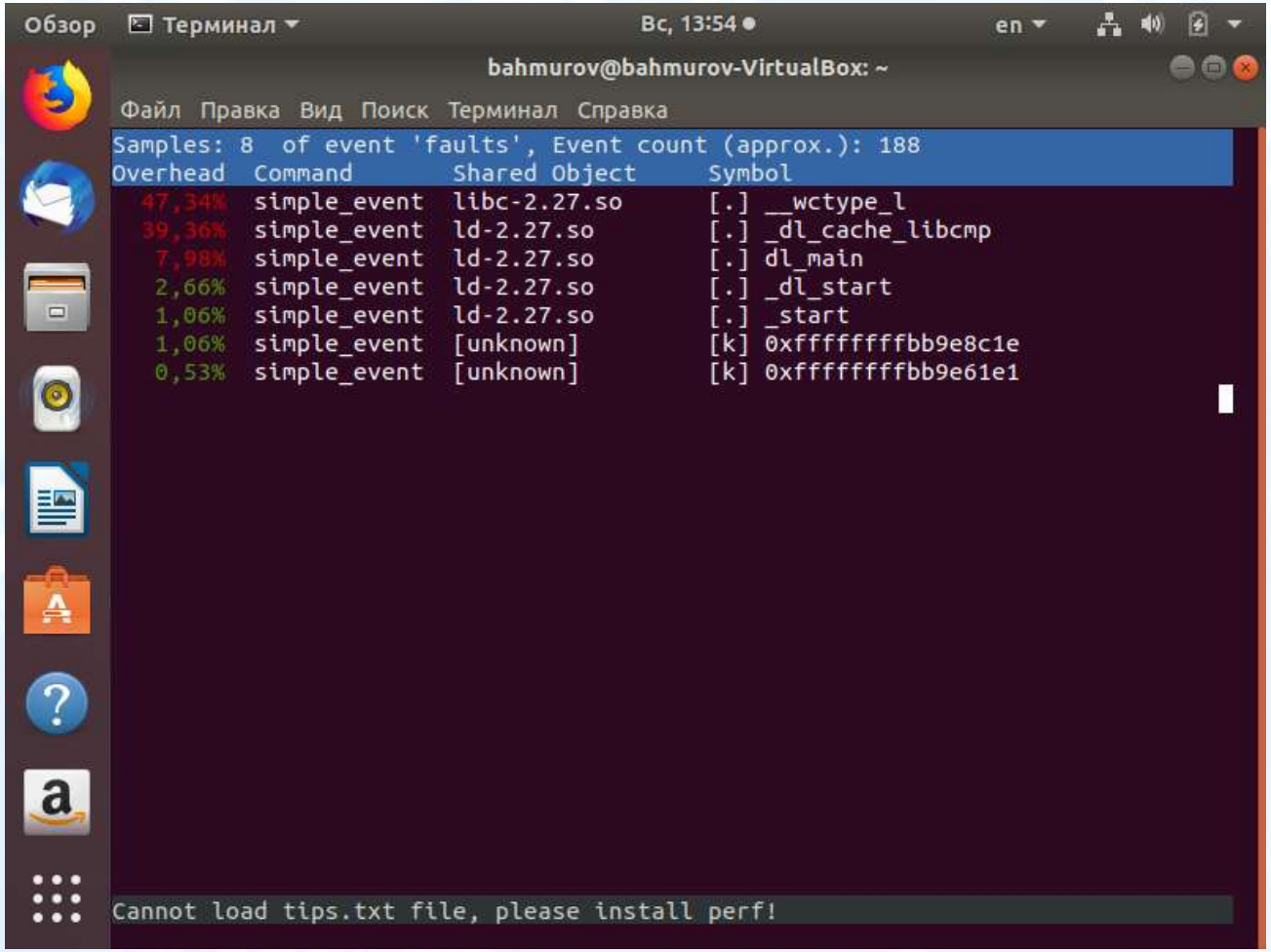
% perf list

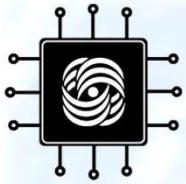
Трассировка на уровне системных вызовов

% perf trace *команда аргументы*



Профилирование программ: perf

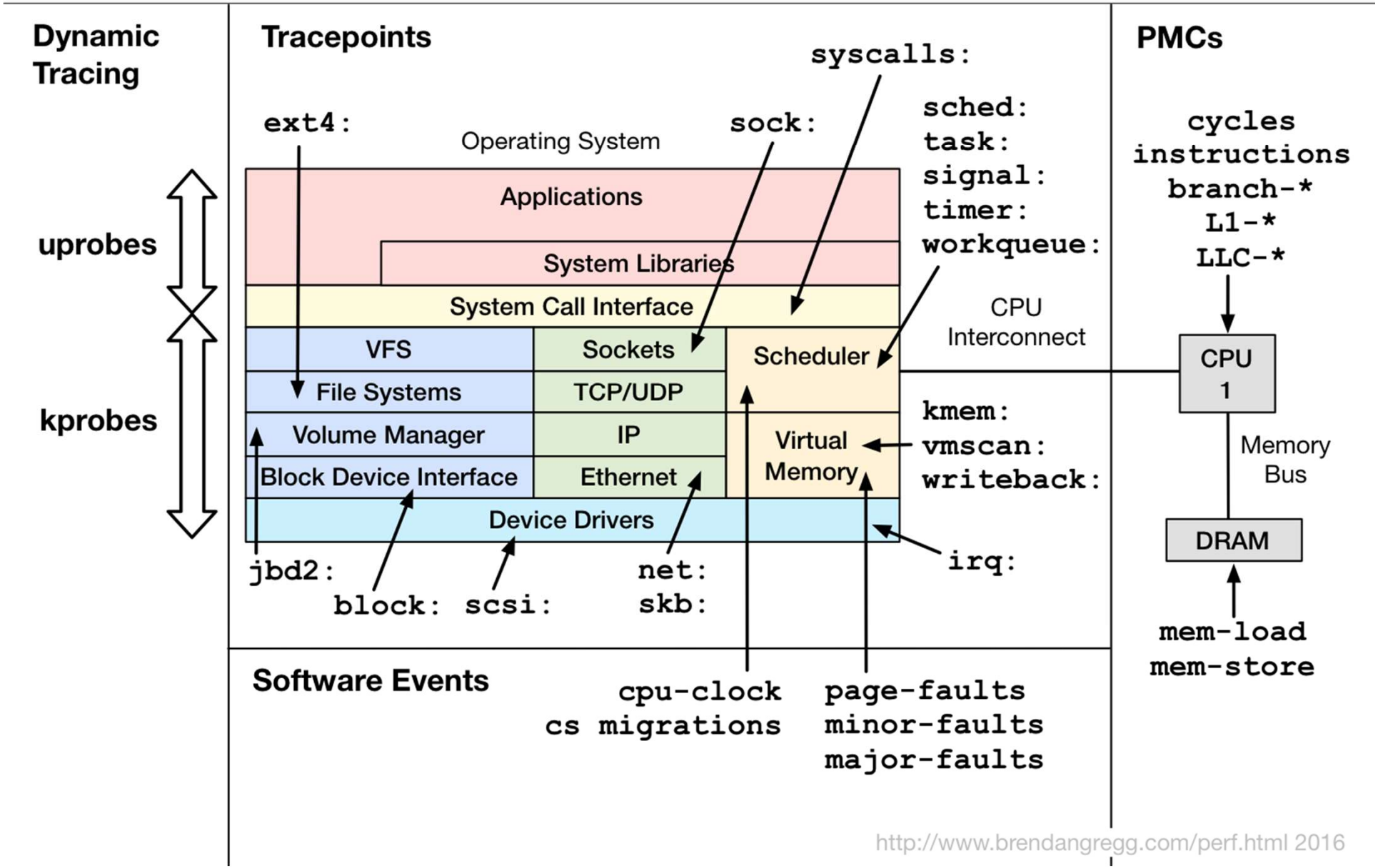




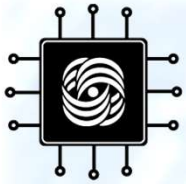
Профилирование программ: perf

- Регулируемая частота выборки
- Большое число типов событий, как с аппаратных, так и программных, в т.ч. в ядре Linux
- Возможность вводить свои события
- По сути, обобщённый интерфейс к средствам наблюдения в Linux

Linux perf_events Event Sources



http://www.brendangregg.com/perf.html 2016



Valgrind (**www.valgrind.org**)

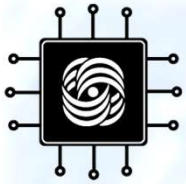
% valgrind *исполняемый_файл аргументы*

% valgrind -- tool=cachegrind *исполняемый_файл аргументы*

Динамическая трансляция исполняемого кода с добавлением проверяющего кода

Инструменты (некоторые):

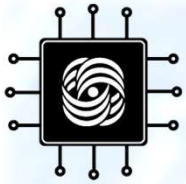
- Memcheck - Проверка корректности работы с указателями и malloc/free
  - Использование неинициализированной переменной
  - Запись и чтение за пределами выделенного блока памяти
  - Повторное освобождение блока и освобождение по некорректному адресу



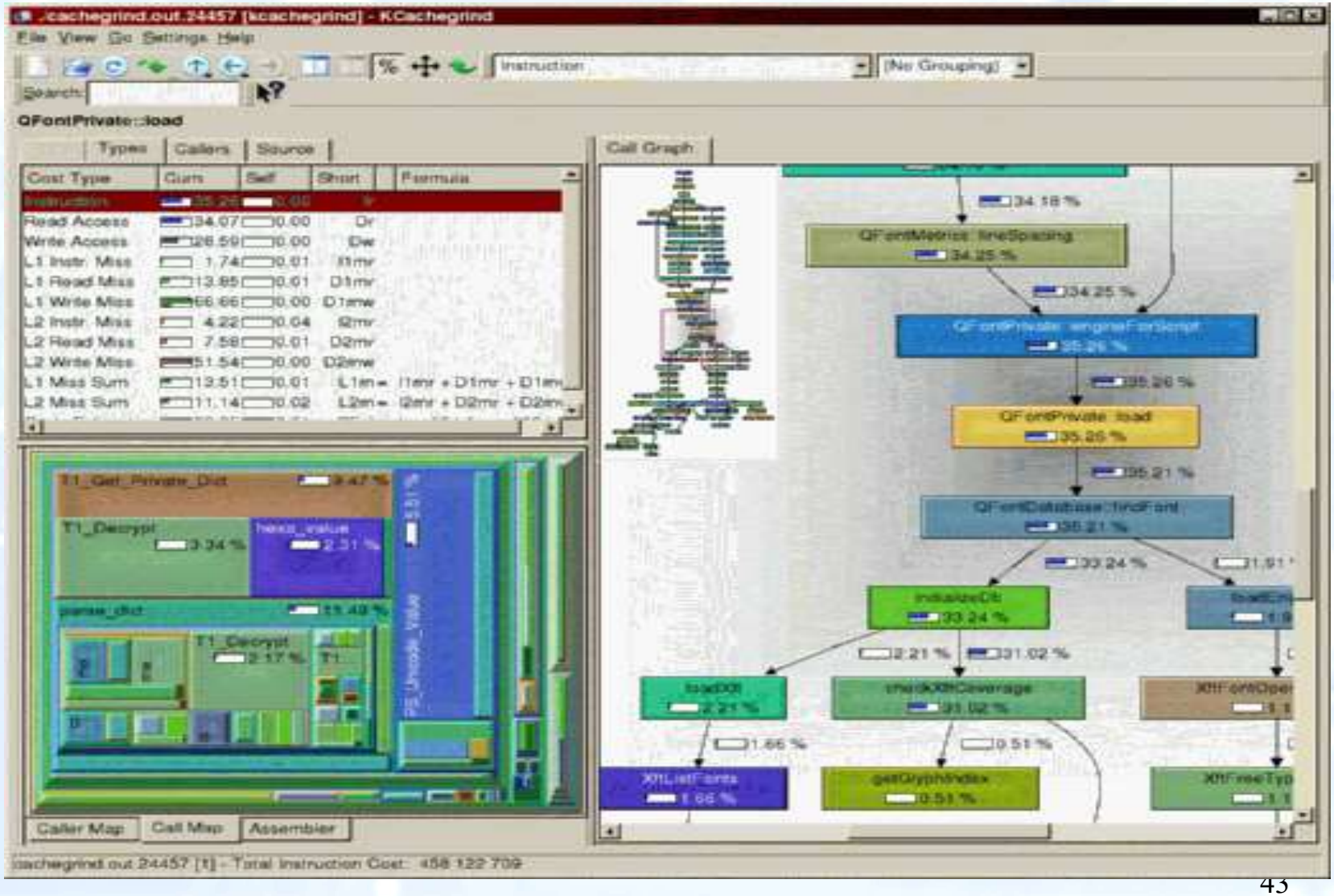
Инструменты valgrind (2)

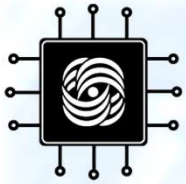
Callgrind – профилировщик в части использования кэш-памяти и построитель графа вызовов

При помощи визуализатора kcachegrind можно посмотреть граф вызовов и количество промахов в I1, L1 и L2 по функциям программы



kcachegrind





Инструменты valgrind (3)

Helgrind, DRD

Обнаружение «гонок» обращений в память из нескольких нитей, обнаружение некорректного использования интерфейса pthreads

http://www.valgrind.org/docs/manual/hg-manual.html

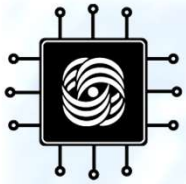
http://www.valgrind.org/docs/manual/drd-manual.html

Helgrind – имеет более наглядные сообщения, DRD – быстрее работает.

Есть ещё профилирование работы с кучей (см. http://valgrind.org/info/tools.html)

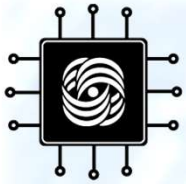
Лабораторная работа 1

- Измерение времени выполнения программы (файл p.c на странице курса)
- Используйте
  - команду time
  - rdtsc (как вставлять asm в Си?)
- Повторите замер несколько раз, возьмите среднее
- Уменьшите время выполнения как можно больше, изменяя её текст (сохраняя семантику!) и настройки сборки.
- Попробуйте применить другие подходящие средства наблюдения



Работа №1: отчёт

- Титульный лист: факультет, кафедра, название работы, кто выполнил, где и когда (Москва 2019)
- Постановка задачи
- Конфигурация вычислительной системы
- Результаты измерения времени
- Способы ускорения выполнения программы и результаты их применения
- Объяснение, почему получилось ускорить выполнение (к каждому применённому способу).



Литература (1)

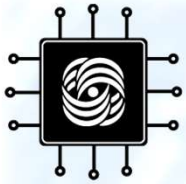
О тестах для измерения производительности (см. раздел 3 документа)
http://citforum.ru/hardware/app_kis/contents.shtml

Трассировка в процессорах Intel
https://software.intel.com/en-us/blogs/2013/09/18/processor-tracing

Учебные материалы по perf
https://perf.wiki.kernel.org/index.php/Tutorial

Брендан Грегг о производительности в Linux
http://www.brendangregg.com/linuxperf.html

О профилировании в Linux
https://habr.com/ru/company/metrotek/blog/261003/



Литература (2)

Среднее число команд за такт для разных процессоров

https://www.agner.org/optimize/instruction_tables.pdf

Рекомендации Агнера Фога по оптимизации программ на C++

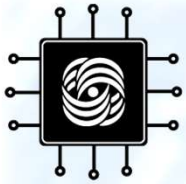
https://www.agner.org/optimize/optimizing_cpp.pdf

JTAG через USB на процессорах Intel

https://itnan.ru/post.php?c=1&p=341946

Средства отладки микропроцессорных систем

https://www.intuit.ru/studies/courses/604/460/lecture/10353?page=4



Литература (3)

Использование Helgrind и DRD для отладки мультизадачных программ

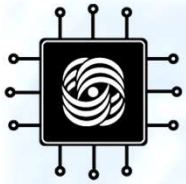
https://accu.org/index.php/articles/1867

Ещё об измерении производительности

https://www.intuit.ru/studies/courses/3457/699/lecture/14133?page=2

Оценка производительности вычислительных систем

https://www.osp.ru/os/1996/02/178845



Спасибо за внимание!