BSD – alternatywa dla Linuksa

Historia

- **1969** – powstaje pierwszy Unix
- **1974** – instalacja Unix'a na maszynie PDP-11 na Uniwersytecie Kalifornijskim w Berkeley
- **1977** – wydanie 1BSD - poprawki do jądra Unix'a oraz kompilator języka Pascal stworzone na ww. uniwersytecie, wydane jako dodatek do Unixa V6. Główny twórca - Bill Joy, późniejszy założyciel Sun Microsystems.

Historia

- **1979** – wydanie 3BSD
  - na maszynę VAX
  - napisany przez studentów z Berkeley
  - kompletny system operacyjny
  - implementacja pamięci wirtualnej
  - sukces! - zainteresowanie Defense Advanced Research Projects Agency
- **1983** – 4.2BSD
  - TCP/IP
  - Berkeley Fast File System (na jego podst. ext2)

Historia

- **1988** – 4.3BSD-Tahoe
  - pierwsza próba podzielenia kodu na część zależną i niezależną od maszyny – podwaliny przenośności systemu operacyjnego
- **1991** – Net/2
  - pierwsze wydanie na licencji BSD (a nie AT&T), a co za tym idzie napisanie do nowa dużej części kodu standardowej dla Unixa
- **1991** – 386BSD
  - pierwszy Unix dla PC

Historia

- **1992-94** – proces sądowy z AT&T
  - okazało się, że z 18000 plików tylko ok.70 musiało zostać zmienionych
  - zahamowanie rozwoju BSD w ważnym momencie – w tym czasie pojawił się Linux, konkurent 386BSD
  - „*Może nie powstałby Linux*” - Linus Torvalds :-)

Historia

- **1995** - 4.4BSD-Lite Release 2
  - ostatni system napisany w Berkeley
- Na 4.4BSD bazują do dziś rozwijające się projekty takie jak:
  - FreeBSD
  - OpenBSD
  - NetBSD

My skupimy się na **FreeBSD**

Historia (FreeBSD)

- **1995** – FreeBSD 2.0
  - FreeBSD Ports
  - Sukces! System użyty w takich przedsięwzięciach jak Yahoo, Hotmail

Od tej pory FreeBSD jest sukcesywnie rozwijane

- **8 maj 2006 –** FreeBSD 6.1
  - najnowsza stabilna wersja

Dystrybucja

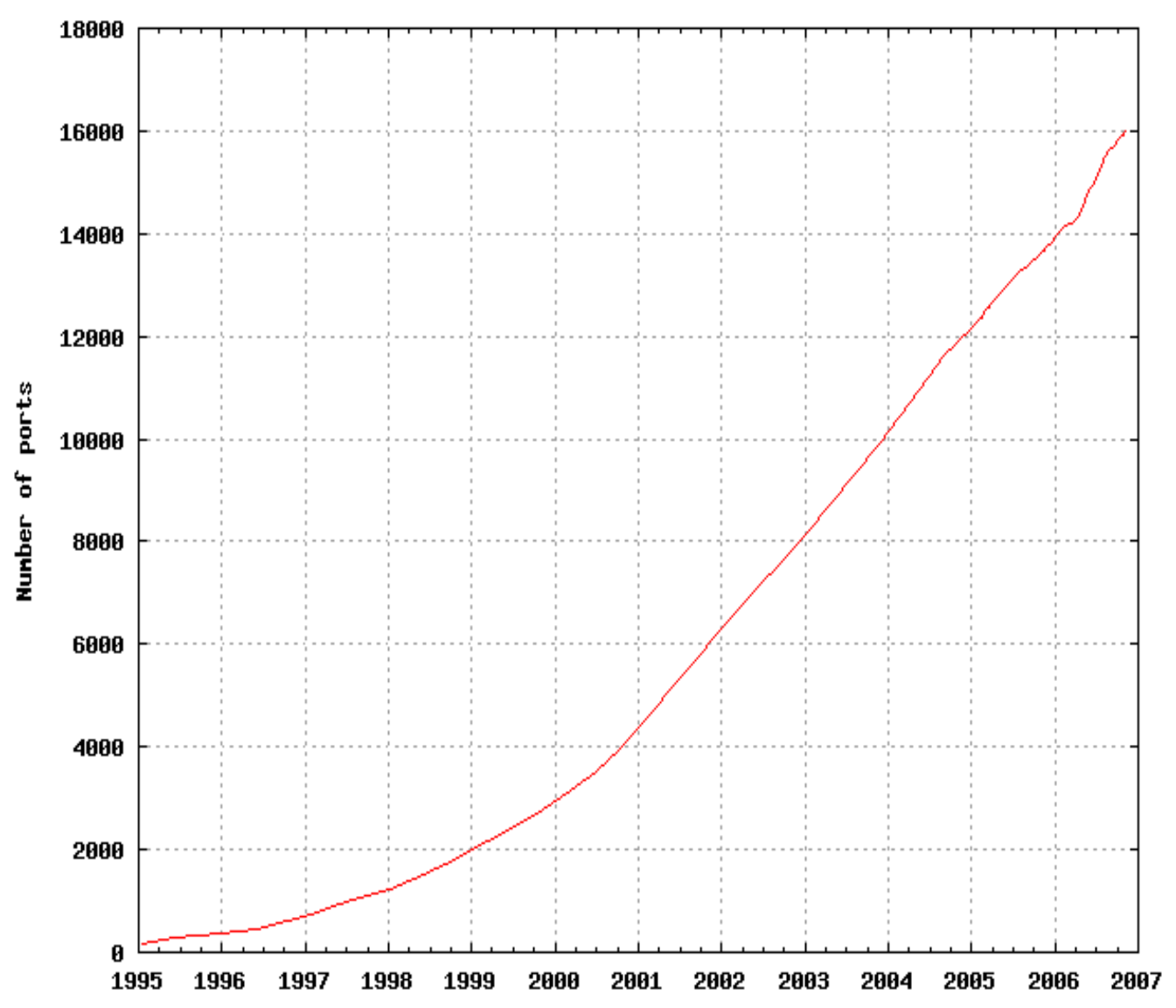
Rozprowadzany jako **kompletny system operacyjny** czyli jądro, sterowniki, programy i narzędzia. W przeciwieństwie do Linuxa gdzie jądro tworzone jest przez jedną grupę ludzi, programy użytkowe przez inną, a to wszystko w paczki zwane dystrybucją Linuxa pakują jeszcze inni.

Porty

- Port – program do instalacji aplikacji z nim związanej
  - Wie jakie aplikacje są potrzebne do poprawnego działania aplikacji i skąd je ściągnąć
- Każdy port ma właściciela – osobę odpowiedzialną za niego
- Makefile - łatwa instalacja i deinstalacja oprogramowania
- Pakiety – tak samo tylko już skompilowane programy

- Praktyczny pokaz działania w dalszej części

Porty


Number of ports
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007

Licencja BSD

- Berkeley Software Distribution License
- free software license
- liberalna, duże prawa użytkownika
- pozwala na włączenie kodu nawet do projektów o zamkniętym kodzie
- na tej licencji min. PostgreSQL.

BSDL vs GNU GPL

- permissive
  - kod po zmianie nie musi juz być na tej licencji
  - wykorzystane np. w Windowsach
  - podoba się dużym firmom
- copyleft
  - kod nawet po zmianie ma pozostać na tej licencji
  - korzystniej wpływa na rozwój open source

Zastosowania

Popularny jako OS na serwer lub firewall

- Dlaczego?
  -wydajny, szczególnie przy dużym obciążeniu sieci nawet do 30% wydajniejszy od Linuxa na tym samym sprzęcie
  -stabilny (wysokie miejsca w rankingach uptime)
- Przykłady:
  - ftp.freesoftware.com (1.2TB/day of downloads)
  - Yahoo!

Zastosowania

- Wciąż mniej popularny od Linuxa w zastosowaniach „biurkowych”

- Dlaczego?
  - Linux miał „łatwiejszy start” (proces BSD z AT&T)
  - różnorodność dystrybucji Linuxa (ale czy to + ?)

Scheduler we FreeBSD

- Przed FreeBSD 5.0 brak wyodrębnionego scheduler'a (kawałki kodu w różnych miejscach jądra)
- W wersji 5.0 posprzątano – scheduler jako wyodrębniona całość (możliwość łatwiejszej wymiany)

Dwie wersje do wyboru

- 4.4BSD Scheduler – domyślny do wersji 5.1

  `/sys/kern/sched_4bsd.c`

- ULE Scheduler – domyślny od wersji 5.2

  `/sys/kern/sched_ule.c`

4.4BSD Scheduler

- Priorytet statyczny `kg_nice`
  - jak w Linuxie (-20..20, domyślnie 0)
  - ustawia użytkownik
- Kwant czasu 100ms
  - na sztywno! nie zmieniony od 20 lat
  - w Linuxie zależy od `p->static_prio`

    (od 5 do 800ms, domyślnie też 100ms)

Kolejki

- Osobne kolejki FIFO dla każdej wartości priorytetu dynamicznego (jak w Linuxie)
- Jeden zestaw kolejek
  - (w Linuxie `active` i `expired`)
- Wywłaszczony wątek trafia na koniec kolejki
  - (w Linuxie zależnie od *interaktywności* do `active` lub `expired`)

Wywłaszczanie

- Po wykorzystaniu kwantu czasu
  - jak w Linuxie
- Natychmiastowo (ew. na koniec syscall'a)

  jeśli pojawi się wątek o wyższym priorytecie
  - np. doczekał się czegoś (wyszedł ze stanu *idle*)
  - nagle dostał wyższy priorytet dynamiczny

Priorytet dynamiczny

- Obliczany według wzoru:

```
A + B*kg_estcpu + C*kg_nice
```

- `kg_estcpu` – odpowiednik *bonusa* w Linuxie...
- ...ale nie limitowany (w Linuxie -5...+5)

kg_estcpu

- Przeliczany raz na sekundę
- Według wzoru:

$$\texttt{kg\_estcpu} = \frac{2*\texttt{load}}{2*\texttt{load}+1}\ \texttt{kg\_estcpu} + \texttt{ticks}$$

- `ticks` aktualnie działającego wątku zwiększane o jeden co 40ms, zerowane co 1s
- `load` – średnia dł. kolejki oczekujących z ostatniej minuty

kg_estcpu – idea

- duży `load`
  - aktywny proces szybko się *starzeje*
  - `ticks` kumulują się, traci priorytet
  - nie zmonopolizuje procesora (za bardzo :/)
- mały `load`
  - `ticks` szybko się przedawniają

    (po 5s tylko 13% starych zostaje)
  - wybaczamy procesowi, że żużył (i tak wolne) CPU

kg_estcpu...

- Heurystyka
  - śpiącym procesom nie przeliczamy `kg_estcpu`,
  - ...ale od razu jak wstaje przeliczamy:

$$\texttt{kg\_estcpu} = \left[\frac{2*\texttt{load}}{2*\texttt{load}+1}\right]^{\texttt{kg\_spltime}} * \texttt{kg\_estcpu}$$

  - zmniejsza koszt obliczeń
  - zmienia od razu priorytet dynamiczny, być może przerywa aktualnie działający wątek – wyraźnie poprawia interaktywność

różnice w stosunku do Linux 2.6

- Stały kwant czasu
- Pojedyńczy zestaw kolejek
- Brak pojęcia *interaktywności*
- Nielimitowany *bonus*
- Co 1s przeliczamy *bonusa* – koszt O(n)

ULE Scheduler

- znacznie bardziej jak Linux 2.6
- O(1) zamiast O(n)
  - kolejki `current` i `next` – jak w Linuxie
  - *interactivity score* – podobnie jak w Linuxie,

    ale *czy_interaktywny* nie zależy od `kg_nice`

ULE Scheduler i SMP/SMT

- własny zestaw kolejek dla każdego CPU
- migracja wątków
  - *push migration* – zajęty procesor przed dodaniem wątku do swojej kolejki `active` sprawdza czy nie ma wolnego procesora
  - co 0.5s `sched_balance()` - wyrównanie kolejek najmniej i najbardziej obciążonego

Bibliografia

- *The Design and Implementation of the FreeBSD Operating System*
  - *By Marshall Kirk McKusick, Marshall Kirk McKusick, George V. Neville-Neil.*
  - *Published by Addison Wesley Professional.*

- Rozdział 4.4 Thread Scheduling

API – różnice

- POSIX
  - standard dla Linuxa i FreeBSD
  - nie precyzuje wszystkiego
- Linux Binary Compatibility Layer
  - zwykle bardzo proste mapowanie
  - szybkość!

Szukamy różnic... `open()`

- Podobnie:
  - robią z grubsza to samo!
  - składnia ta sama
  - ten sam *systemcall* (wartości `eax` itp.)

Szukamy różnic... `open()`

- Ale np.:
  - creat() niezalecana w FreeBSD
  - brak flag `O_SHLOCK` i `O_EXLOCK` w Linuxie
  - `O_SYNC` w Linuxie (z licznymi uwagami),

    `O_FSYNC` w FreeBSD (opisany jako niezawodny :/)
  - `O_DIRECT` w FreeBSD, w Linuxie brak
  - itd...

`lseek ()`

- Bardzo podobnie
  - np. oba obdługują *dziurawe* pliki
- Ale np. błąd `EOVERFLOW` tylko we FreeBSD

`write ()`

- Drobne różnice
- Syscall `pwrite()` we FreeBSD,

  w Linuxie nie ma nic podobnego
- Więcej błędów we FreeBSD
  - np. `ENOBUFS`
  - głównie związane z obsługą socketów

FreeBSD, część III - pokaz

- Porty/pakiety
- Linux compatibility layer
- Testy wydajności

Porty/pakiety

- Pakiety
- Porty
- Pokaz

Porty/pakiety

- Pakiety
  - Standardowe podejście
  - Administracja poprzez pkg_add, pkg_info, pkg_delete
  - Obsługa zależności
  - Jedno oficjalne repozytorium, brak możliwości definiowania innych, ale możliwość instalowania pakietów spoza niego
  - Możliwość budowania pakietów z portów

Porty/pakiety

- Porty
  - Protoplasta ebuild-ów w Gentoo, czyli opis, skąd zdobyć i co zrobić ze źródłami
  - Makefile z kilkoma pomocniczymi plikami
  - Obsługa zależności w połączeniu z pakietami
  - Stworzony framework do pisania portów
  - /usr/ports

Porty/pakiety

- Pokaz
  - Wyszukanie port-a
  - Instalacja (kompilacja) porta
  - Pokazanie wszystkich zainstalowanych pakietów
  - Usunięcie port-a
  - Wyszukanie pakietu
  - Instalacja pakietu
  - Pokazanie wszystkich zainstalowanych pakietów
  - Usunięcie pakietu

Linux compatibility layer

- Co to jest?
- Czy to faktycznie jest potrzebne?
- Na jakiej zasadzie to działa?
- Jak dobrze to działa?
- Instalacja
- Pokaz

Linux compatibility layer

- Czy to jest faktycznie potrzebne?
  - Różnice w API
  - Istnienie programów dostępnych tylko w postaci binarnej
  - Ułatwia testowanie programów na dwóch systemach
  - Doom ;)

Linux compatibility layer

- Na jakiej zasadzie to działa
  - Moduł z linuksowym API
  - Binary branding
  - Podmiana wektora sysent[] w strukturze
  - Podmiana najwyższego katalogu w wyszukiwaniach bibliotek

Linux compatibility layer

- Jak dobrze to działa?
  - Netscape®
  - Adobe®
  - Acrobat®
  - RealPlayer®
  - VMware™
  - Oracle®
  - WordPerfect®
  - Doom
  - Quake,

Linux compatibility layer

- Instalacja
  - Moduł linux w jądrze
  - Port emulators/linux_base*
  - brandelf / sysctl kern.fallback_elf_brand=3
  - resolv.conf
  - devfs
  - procfs

Testy wydajnościowe

- Czas działania wywołań funkcji systemowych (program libMicro) dla Linuksa, FreeBSD i FreeBSD przez linuksowe ABI.
- Średnia przepustowość systemu plików na symulacji obciążenia serwera SMB (dbench)
- Efektywność systemów plików (reiserfs pod Linuksem, reiserfs i ufs pod FreeBSD, bonnie++)
- glxgears

Testy wydajnościowe

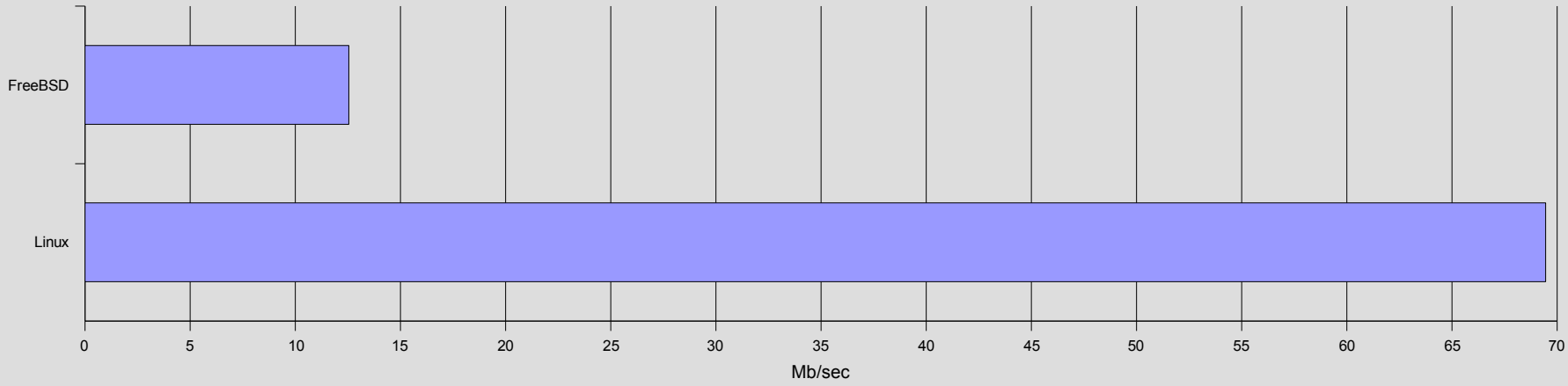
- Środowisko testowe
  - Pentium 4, 2,66 GHz, 512Kb cache
  - 1024 Mb RAM, DDR 400
  - Chipset Intel ICH4
  - Dysk 80GB WDC-WD800VE
  - ATI Radeon 9000M
  - Linux: Gentoo, gcc 3.4.6, kernel 2.6.16, i686,reiserfs
  - FreeBSD: 6.1 RELEASE, gcc 3.4.4, i686,ufs
  - Oba jądra kompilowane dokładnie pod sprzęt testowy

Testy wydajnościowe

- Czas działania wywołań funkcji systemowych
  - LibMicro 0.3.0 (http://www.opensolaris.org/os/community/performance/libmicro/)
  - Wyniki na http://students.mimuw.edu.pl/~md219418/syscalls.html

Testy wydajnościowe

- Średnia przepustowość systemu plików na symulacji obciążenia serwera SMB
  - Program dbench 3.04
  - 100 użytkowników
  - Wyniki:
    - Linux: 69,4554Mb/sec
    - FreeBSD: 12,5435Mb/sec

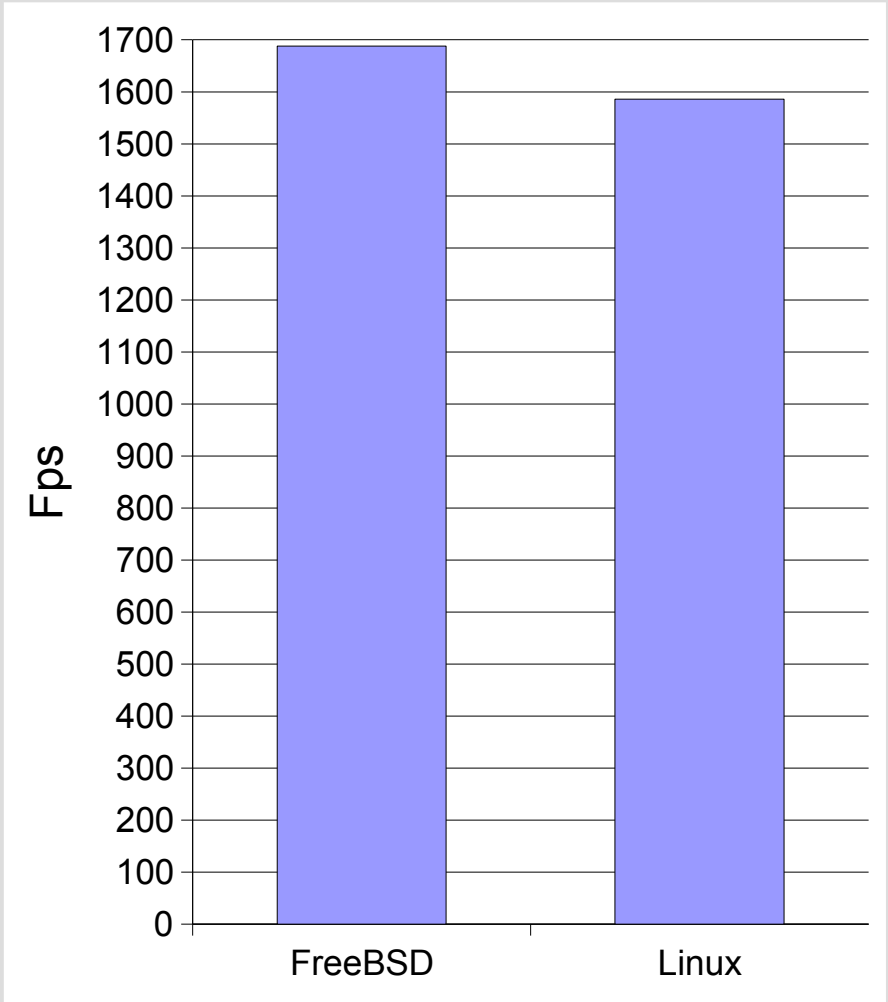


Testy wydajnościowe

- Bonnie++ 1.93c
  - bonnie++ -s 1800 -n 100:1024:0:100 -m siersciu -u 0
  - Dokładne wyniki na http://students.mimuw.edu.pl/~md219418/bonnie.html

Testy wydajnościowe

- glxgears



Bibliografia

- http://www.networkworld.com/supp/2005/opensource/070405-face-off-no.html
- http://www.networkworld.com/supp/2005/opensource/070405-face-off-yes.html
- http://en.wikipedia.org/wiki/BSD_and_GPL_licensing
- http://en.wikipedia.org/wiki/FreeBSD_Ports
- http://www.freebsd.org/
- http://en.wikipedia.org/wiki/Copyleft
- http://people.freebsd.org/~murray/bsd_flier.html
- http://uptime.netcraft.com/up/today/top.avg.html
- http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/
- http://www.coker.com.au/bonnie++/
- http://www.opensolaris.org/os/community/performance/libmicro/
- http://www.dbench.org/home/index.php