Systemy operacyjne oparte na mikrojądrze

Systemy operacyjne 2008/2009

Marcin Hagmajer
Hubert Orlik-Grzesik
Marian Kędzierski

Agenda

Czyli o czym będziemy mówić

- Omówienie mikrojąder
- System MINIX
  - Prezentacja działania systemu
- MINIX czy Linux? (vel mikrojądro vs jądro monolityczne)
  - Spór twórców systemów
  - Testy wydajności
- Zastosowania mikrojąder
  - Gdzie bardziej cenimy stabilność od szybkości?
  - Amoeba, L4, GNU/Hurd, Microsoft Singularity
- Inne podejścia
  - Egzojądra
  - Jądra hybrydowe
- Podsumowanie i perspektywy na przyszłość

Wprowadzenie do mikrojąder

Na podstawie MINIXa

“ A concept is tolerated inside the microkernel only if moving it outside the kernel, i. e., permitting competing implementations, would prevent the implementation of the system's required functionality ”

Jochen Liedtke

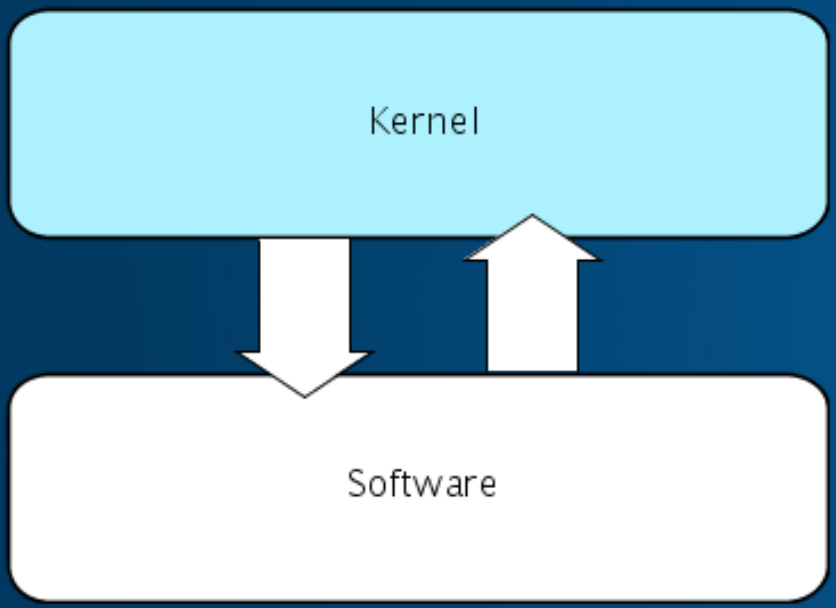
Omówienie mikrojądra

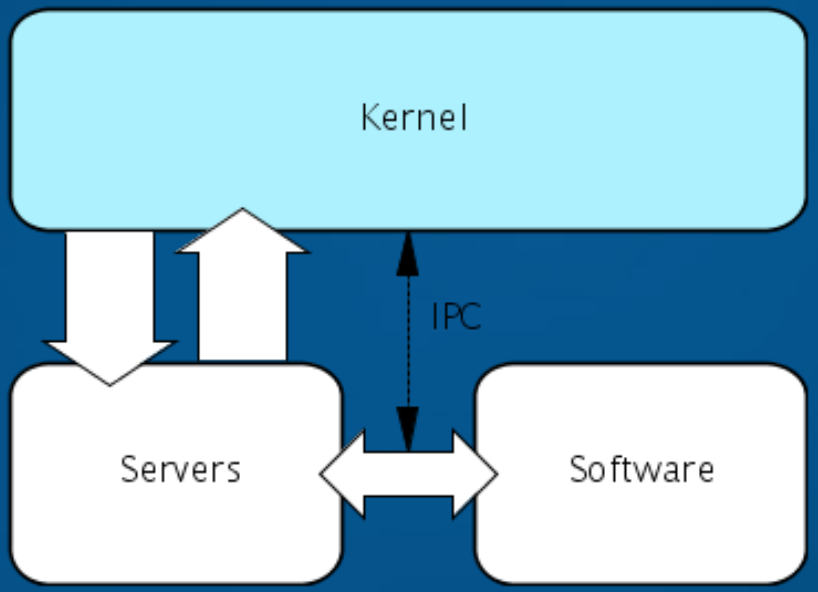
Najistotniejsze cechy

- Jest małe (np. MINIX - ok. 3800 linii kodu)
- Linux - ok. 2,5 mln linii kodu, Windows XP - ok. 5 mln
-
- Udostępnia jedynie podstawowe mechanizmy:
  - Zarządzanie pamięcią
  - Zarządzanie wątkami
  - Komunikacja między procesami (IPC)
- Zawiara cały kod działający w trybie jądra
- Typowe usługi (sterowniki, protokoły sieciowe, systemy plików, interfejsy użytkownika) działają w trybie użytkownika

Omówienie mikrojądra

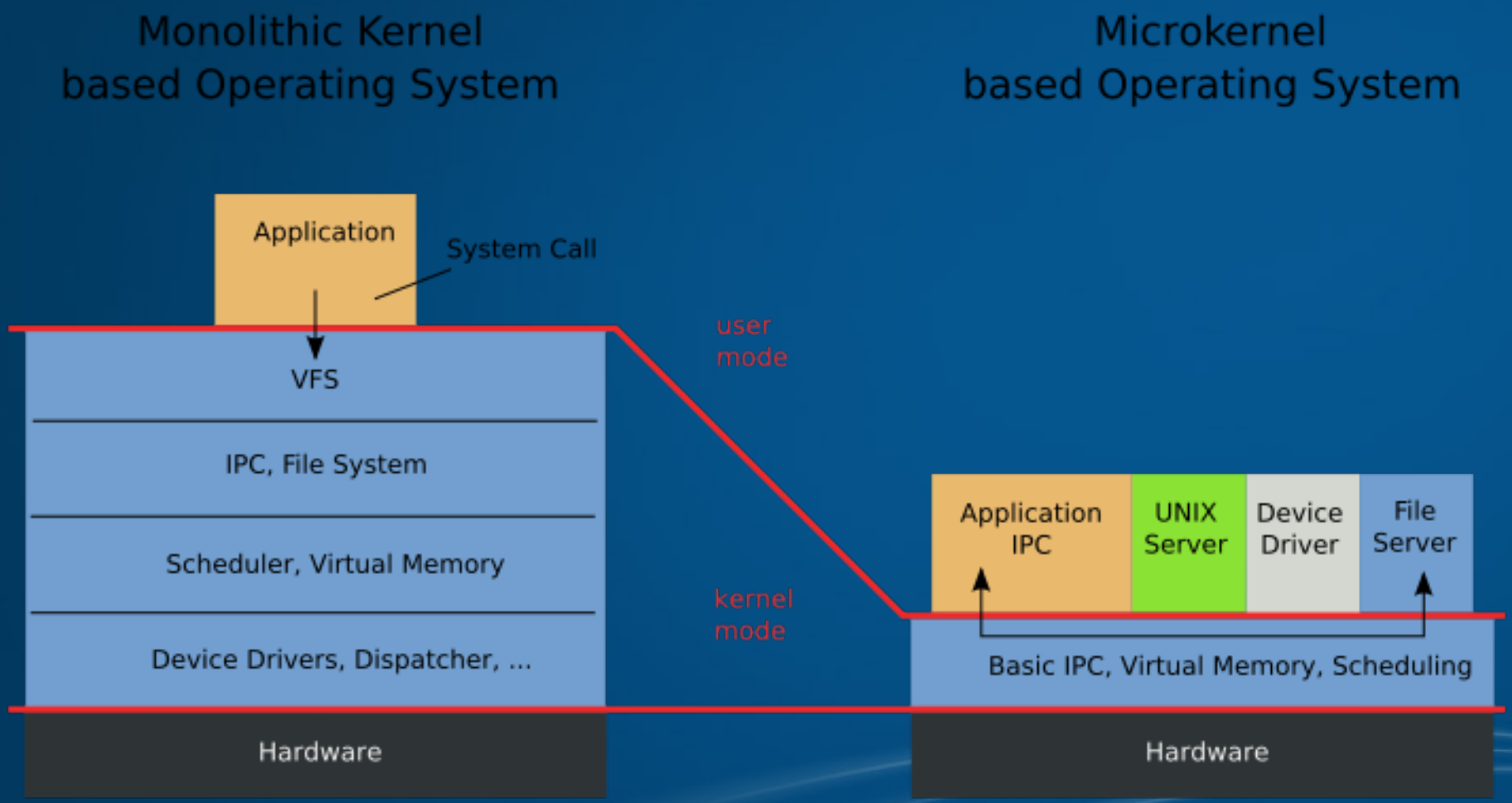
Jądro monolityczne a mikrojądro





Omówienie mikrojądra

Jądro monolityczne a mikrojądro



Omówienie mikrojądra

Serwery (usługi)

- Daemony, którym jądro umożliwia dostęp do pamięci fizycznej i sprzętu (por. sterowniki)
- Mogą być wznawiane, gdy dojdzie do błędu
- Tworzenie nowego serwera (np. systemu plików) przypomina tworzenie "zwykłego programu"
  - nie wymaga kompilacji i bootowania całego jądra
- Podstawowe serwery są uruchamiane wraz ze startem systemu

Omówienie mikrojądra

Komunikacja międzyprocesowa (IPC)

- Używana o wiele częściej niż w systemach z jądrem monolitycznym
- Warianty komunikacji
  - asynchroniczna (wymaga bufora i dwóch kopii wiadomości)
  - synchroniczna (wymaga synchronizacji)
- Postrzegana jako główny powód słabej wydajności mikrojąder (J. Liedtke)
- Rozwój jądra L4 doprowadził do przyśpieszenia komunikacji

Omówienie mikrojądra

Komunikacja międzyprocesowa (IPC) - rozwiązania

- synchroniczna + asynchroniczna (tylko sygnały)
- maksymalnie duża część wiadomości przekazywana w rejestrach
- *natychmiastowe przełączenie wykonania do odbiorcy (direct [incomplete] context switch)*
- wątki procesu wywołującego nie są brane pod uwagę (*lazy scheduling*)
- możliwa iluzja asynchronicznych wiadomości (dodatkowy wątek)

Omówienie mikrojądra

Sterowniki (drivers)

- Wymagają bezpośredniego dostępu do pamięci (DMA)
- Mikrojądro kontroluje ten dostęp
- Obsługa sterowników to większość kodu jąder monolitycznych
- Historycznie sterowników było niewiele i były dobrze przetestowane, dlatego dołaczano je do kodu jąder

Omówienie mikrojądra

Bezpieczeństwo

- Mikrojądro jest małe, zawiera proporcjonalnie mało błędów i jest stabilne
- Wszystkie dodatkowe części systemu uruchamiane zgodnie z zasadą ograniczonego zaufania
- Pad całego systemu bardzo rzadki
- Nie wymaga restartu całego systemu, ani nawet ingerencji użytkownika, w przypadku więszkości problemów

Historia

W dużym skrócie

Historia

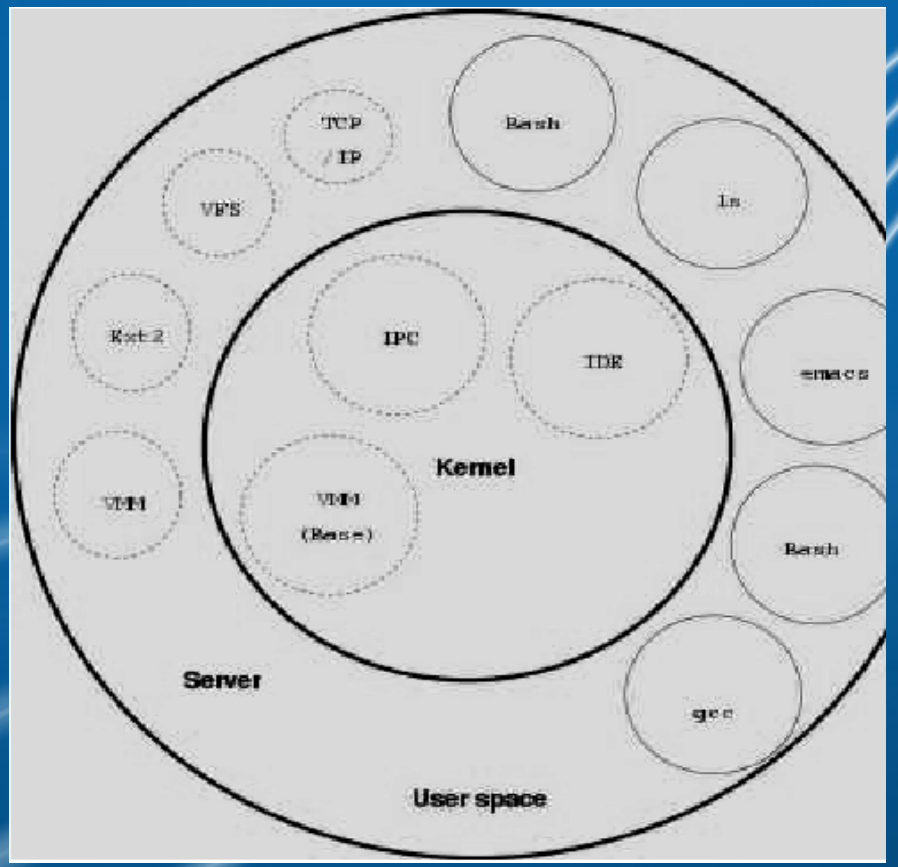
Początki

- Wczesne wersje UNIXa (aż do BSD)
- Idea popularna na przełomie 1980/1990
- Problemy z pełną realizacją, wydajność
- Monoserver system
  - NT
  - MkLinux
  - MachOS

Monoserver system

Kompromis

- Jeden program przejmuje większość funkcji jądra
- Zalety:
  - Większa niezależność
  - Łatwiejszy rozwój
  - Nieco zwiększone bezpieczeństwo
- Wady:
  - W zasadzie nic się nie zmieniło



Mach

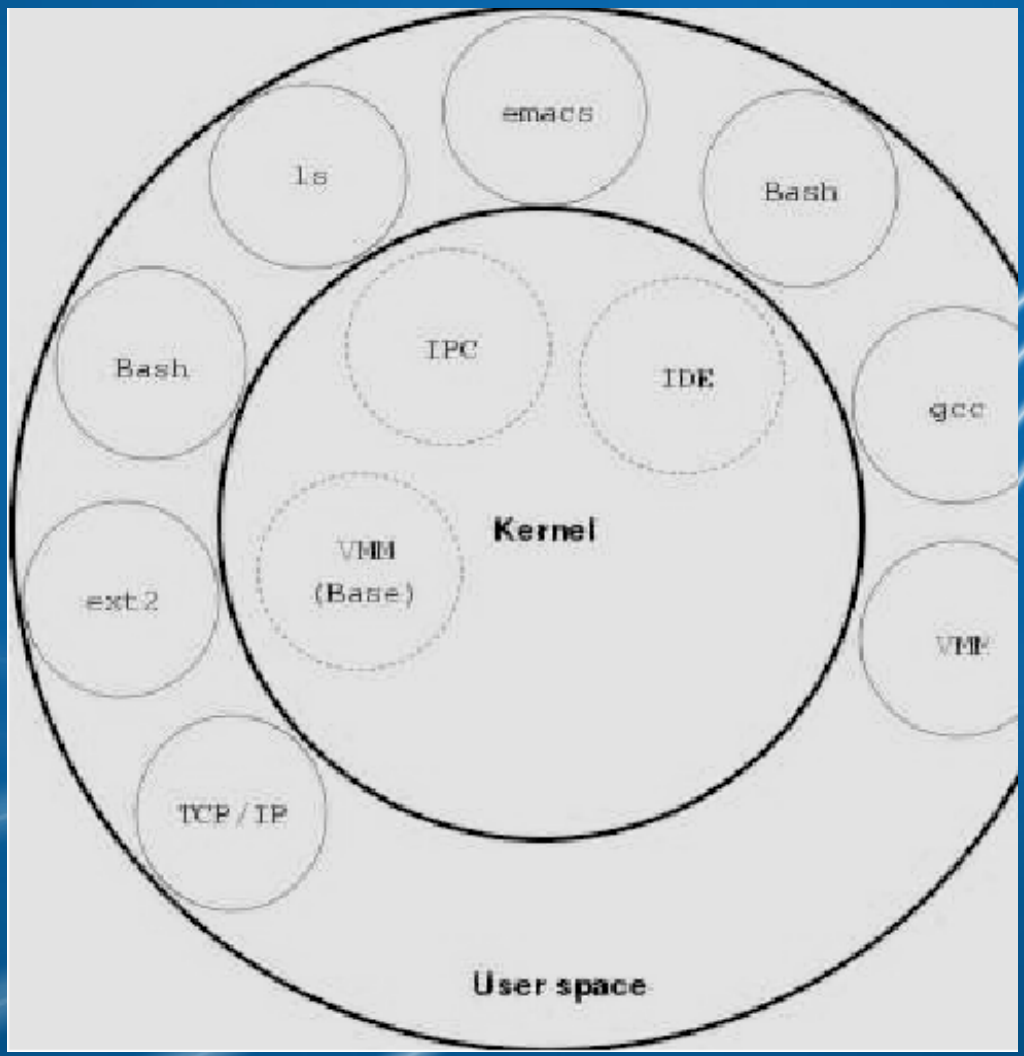
Pierwsze jądro GNU Hurd

- Jedno z pierwszych mikrojąder
- Pierwsze działające mikrojądro
- Carnegie-Mellon, 1985 - 1994
- Nowa teoria: IPC, wielowątkowość...
- Zdefiniowane wątki i zadania

Multiserver system

Czyli to, o czym mówimy

- Wszystko w osobnych procesach
- Zalety:
  - Stabilniejszy
  - Dużo lepszy rozwój
- Problemy:
  - Powolne IPC
  - Potrzebne interfejsy



MINIX

Historia

- Andrew Tanenbaum, Vrije Universiteit, 1987
- Przykład do książki
- 1.5, 1991
- 2.0, 1997 - z 68k na x86
- 3.0, 2005 - dalej jako przykład

System MINIX

Przedstawienie systemu

"We are trying to build a very highly reliable system. It might also be applicable to servers where a mean time to failure of many years is needed."

Andy Tanenbaum

Minix 3

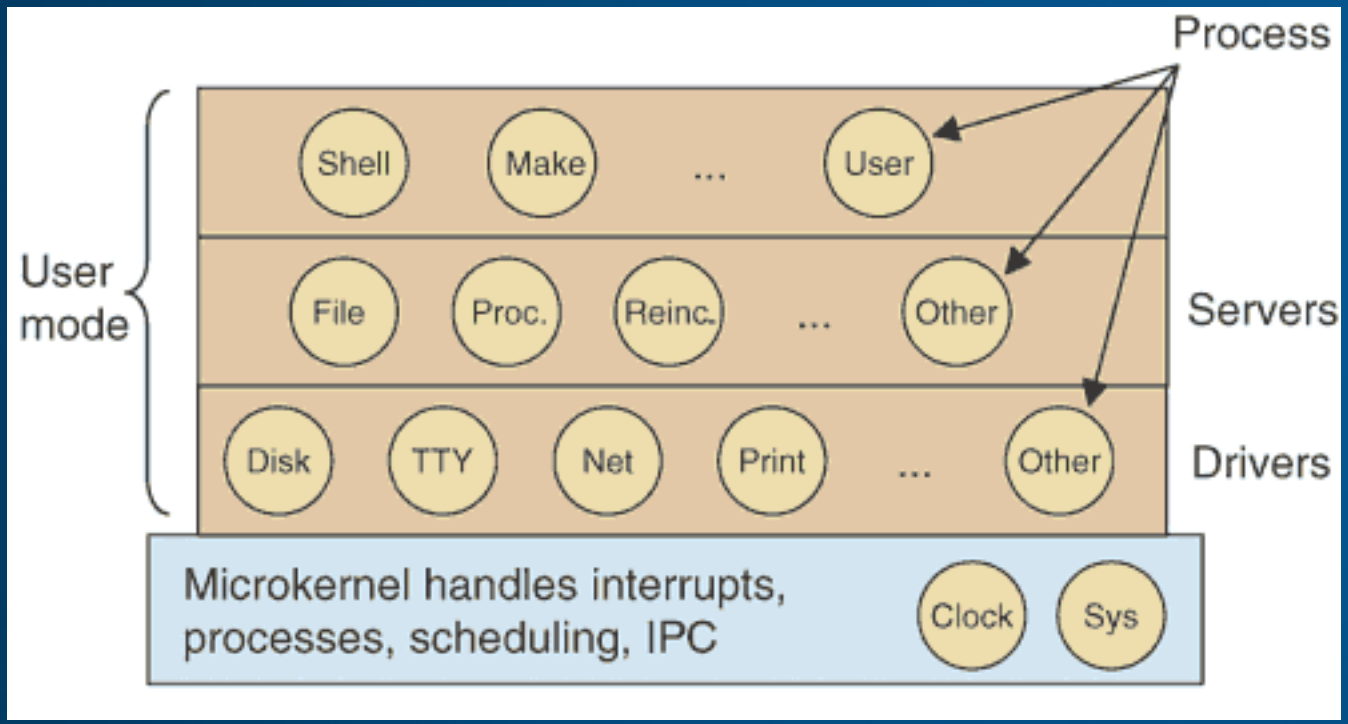
Początki

- Stworzony "od zera" przez A.Tannenbauma w *Vrije Universiteit w Amsterdamie* w 1987r.

- Powstał, by umożliwić każdemu poznanie zasad działania systemu operacyjnego
- Kiedyś płatny, obecnie darmowy
- Oryginalnie (MINIX Trivia) 12,649 linii kodu w C
  - 3000 linii kodu to komentarze

Minix 3

Omówienie architektury



Mikrojądro Minixa obsługuje przerwania, zapewnia podstawowe mechanizmy do zarządzania procesami, implementuje IPC oraz scheduler.

Sterowniki

Czyli dlaczego systemy nie działają?

- 6-16 (lub 2-75) błędów na 1000 linii kodu
- 3-7 razy więcej dla sterowników
- Linux: 2,5 mln linii, 70% to sterowniki
- Pomysły:
  - Uzbrojenie systemu
  - Parawirtualizacja

“Clearly, it is difficult to engineer a system well when nobody really understands it.”

Andy Tanenbaum

Minix 3

Sterowniki

- Działają jako odrębne procesy w prywatnych przestrzeniach adresowych
- Nie korzystają ze sprzętu samodzielnie tylko za pośrednictwem usługi dostarczanej przez jądro
- Wpływają na spadek wydajności czasowej ale zwiększenie stabilności

Minix 3

Serwery

- Serwer reinkarnacji
  - rodzic wszystkich sterowników i serwerów
  - wznawia sterowniki, które przestaną działać
  - wznawia serwery, o ile jest to możliwe
- Serwer systemu plików
  - mały (ok. 4500 linii wykonywalnego kodu)
  - obsługuje wywołania zgodne z POSIX (read, write, etc)
- Serwer procesów
  - zarządza pamięcią procesów
  - POSIX (fork, exec, brk, ...)

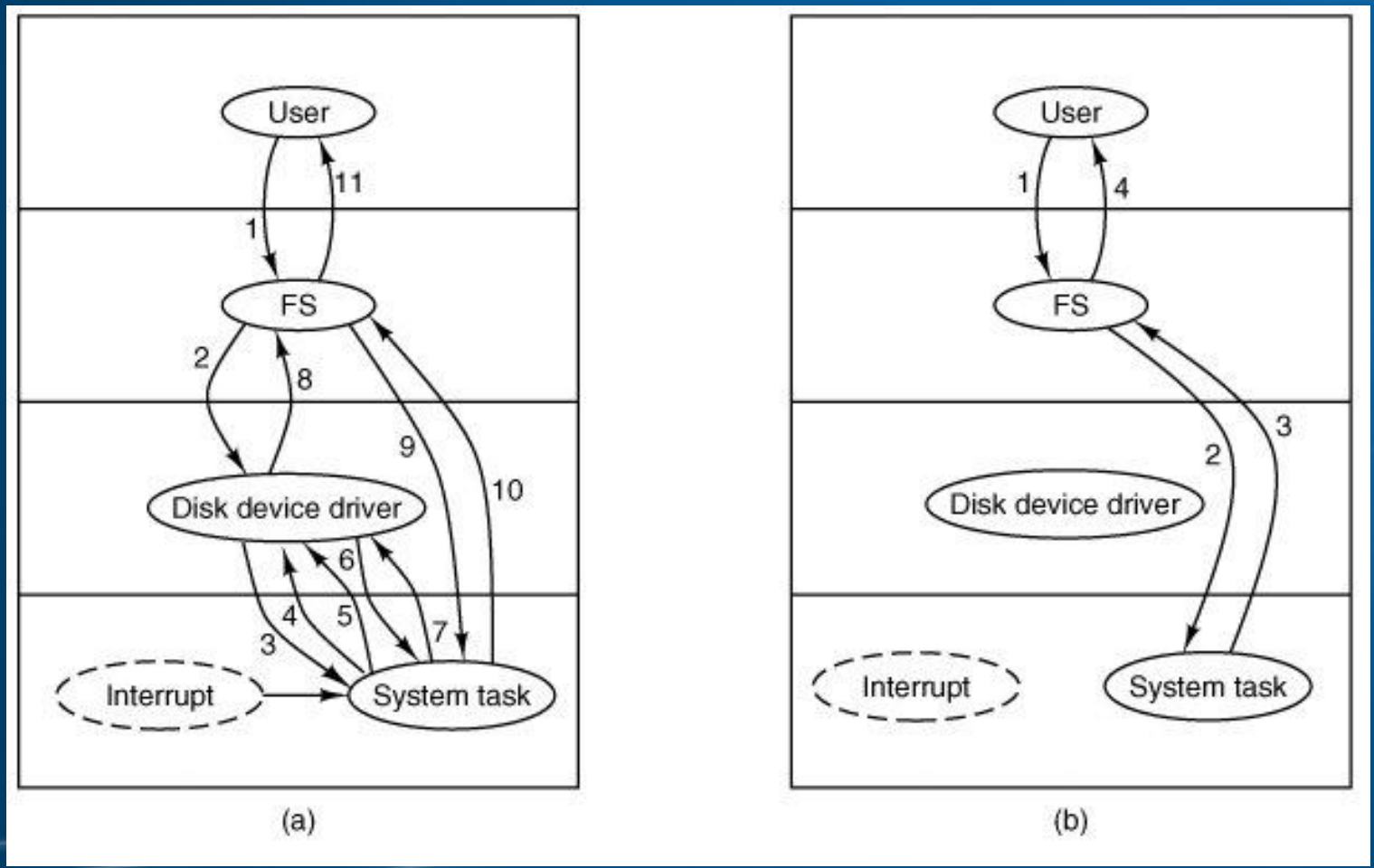
IPC w Minixie

Komunikacja międzyprocesowa

- Zasadniczo dwa poziomy IPC:
  - jądra
  - aplikacji
- IPC obsługiwane przez jądro
  - kluczowe w wydajności systemu
  - dwa rodzaje IPC jądra:
    - mechanizm spotkań
      - komunikaty stałej długości
    - asynchroniczne zdarzenia
      - nie noszą danych (~sygnały)

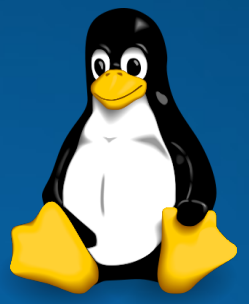
Minix 3

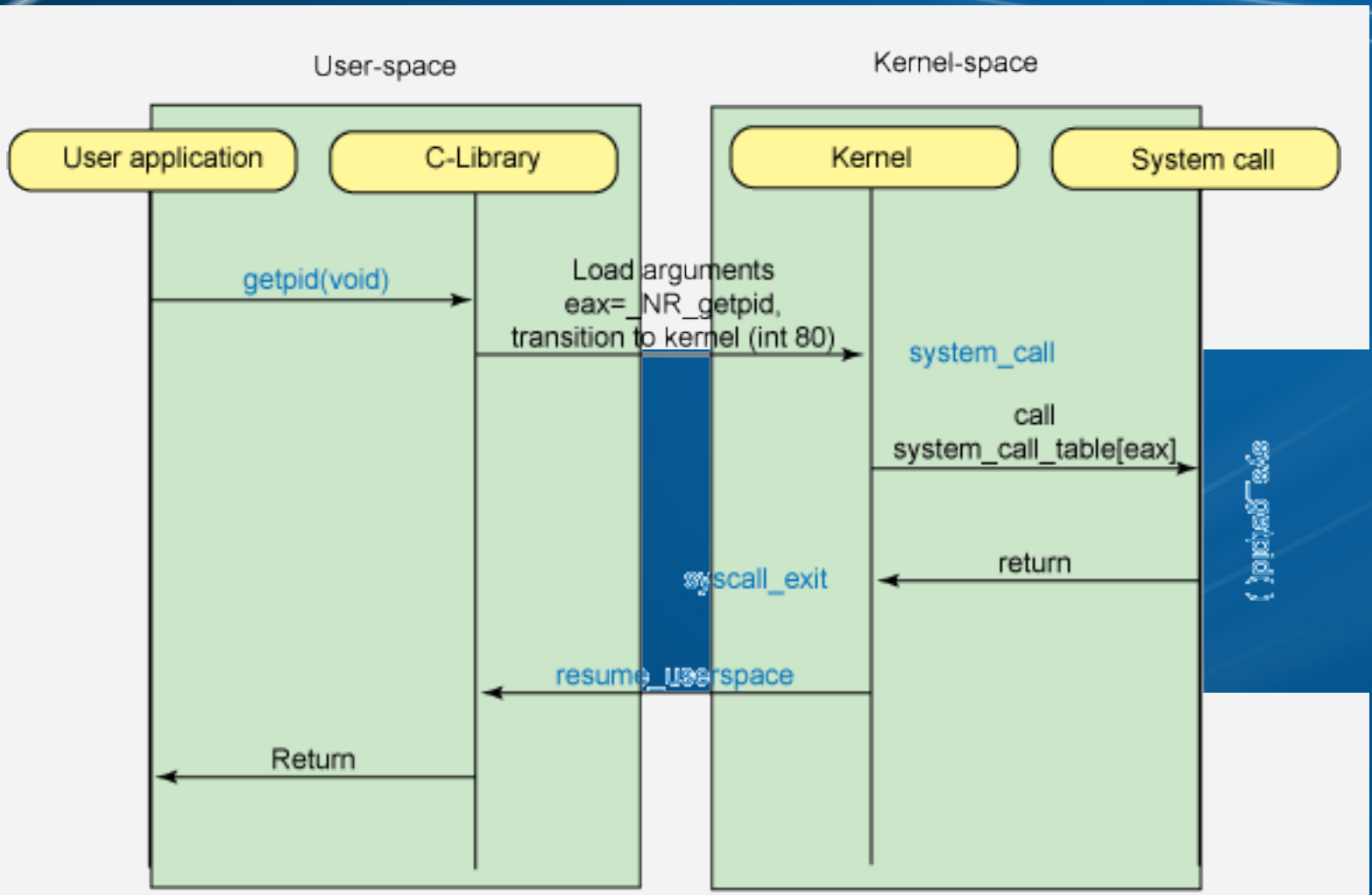
Przepływ komunikatów na przykładzie read()



Linux

Dla porównania





Scheduler w Minixie

Obsługa wieloprocesowości

- Heurystyczny algorytm hybrydowy łączący:
  - taktykę *round-robin* (wszystkie procesy po kolei)
  - priorytety procesów
- W każdym procesie są trzy ustawienia dotyczące priorytetu
- System ustala kwant czasu dla każdego procesu

MINIX 3

Prezentacja działania systemu

"Linux is obsolete" (1992)

Dyskusja A. Tanenbauma z L.Torvaldsem

Za i przeciw mikrojądrom

Dyskusja A. Tanenbauma z L.Torvaldsem

W 1992 A. Tanenbaum, poirytowany przeciągająco się dyskusją nt. Linuxa na comp.os.minix, napisał sławny post.

Andrew Tanenbaum

From: ast@cs.vu.nl
Newsgroups: comp.os.minix
Subject: **LINUX is obsolete**
Date: 29 Jan 92 12:12:50 GMT

Linus Torvalds

...

Za i przeciw mikrojądrom

Dyskusja A. Tanenbauma z L.Torvaldsem

- Mikrojądra zostały już uznane za lepsze, a badania pokazują, że ich jedyną wadę (słabsza wydajność) można pokonać dokonując odpowiednich optymalizacji.
- Koncept jądra monolitycznego pochodzi z lat 70. i jest jest przestarzały.
- *"Moim zdaniem tworzenie takiego systemu w 1991 jest naprawdę słabym pomysłem."*

Za i przeciw mikrojądrom

Dyskusja A. Tanenbauma z L.Torvaldsem

- Brak przenośności
  - Linux pisany tylko na x86
  - Architektury szybko się zmieniają
- MINIXa udało się przenieść z linii Intela na:
  - 680x0 (Atari, Amiga, Macintosh)
  - SPARC
  - NS32016

Za i przeciw mikrojądrom

Dyskusja A. Tanenbauma z L.Torvaldsem

- W przeciwieństwie do Linuxa, Minix jest płatny (169$ za licencję z prawem do dwóch kopii)!
- Idea mikrojąder faktycznie jest dobra, ale nie impementuje jej w dobry sposób, ma problemy z wielowątkowością
- *Portability is for people who cannot write new programs*
- Wystarczu przenośne API

Za i przeciw mikrojądrom

Dyskusja A. Tanenbauma z L.Torvaldsem

- *Minix jest projektowany dla studentów, działa na tanim sprzęcie (nawet 4.77 MHZ i bez HDD!)*
- *Co z tego, że Linux jest darmowy, skoro ma duże wymagania. Mało osób stać na jego używanie.*
- *Gdybyś był moim studentem, to dostałbyś marną ocenę za pisanie jądra monolitycznego*
- *Architektura x86 może zostać wkrótce wyparta przez inną*

Za i przeciw mikrojądrom

Dyskusja A. Tanenbauma z L.Torvaldsem

- *Przywiązanie Linux'a do x86 jest decyzją projektową*
- *Linux miał z założenia wykorzystać dodatki architektury 386*
- *"Różne cele dają różne projekty"*
- *MINIX przywiązany do taniego sprzętu to problemy z jego przenośnością*

Kevin Brown

Za i przeciw mikrojądrom

Dyskusja A. Tanenbauma z L.Torvaldsem

- Minix jest pisany niezależnie od architektury, ale nie wykorzystuje szczególnych możliwości żadnej z nich (np. wsparcia dla stronicowania)!
- Linux umożliwia uruchomienie większej liczby aplikacji, jest już częściowo zgodny z POSIX
- Jest systemem do używania, a nie do nauki jak działa

Za i przeciw mikrojądrom

Dyskusja A. Tanenbauma z L.Torvaldsem

Louie

Michael L. Kaufman

Richard Tobin

Ken Thompson

Charles Hedrick

Randy Burns

Tony Travis

Douglas Graham

Kevin Brown

Julien Maisonneuve

"Can We Make Operating Systems Reliable and Secure?" (2006)

Ciąg dalszy dyskusji...

Za i przeciw mikrojądrom

Dyskusja A. Tanenbauma z L.Torvaldsem

- Kodowanie mikrojądra jest znacznie trudniejsze, przez co są wolniej rozwijane
  - Sterowniki i serwery w mikrojądrze pracują w oddzielnych przestrzeniach adresowych
  - Nie można dzielić struktur
  - Wszystkie algorytmy stają się współbieżne - trudniejsze
- Serwery nie działają do końca niezależnie od jądra
  - nie zawsze da się wznowić popsuty proces

Za i przeciw mikrojądrom

Dyskusja A. Tanenbauma z L.Torvaldsem

- Pisanie algorytmów dla jądra Minixa ułatwia mała ilość interakcji pomiędzy częściami systemu
  - Trudniejsza jest obsługa dostępu do dzielonych struktur danych (nawet z muteksami, semaforami i monitorami)
- Mikrojądro stosuje podejście obiektowe
  - Dekopomozycja
  - Nieduże, dobrze określone interfejsy
  - Ukrywanie informacji, zamiast ich współdzielenie

Porównanie MINIXa i Linuxa

Szop pracz czy pingwin?

MINIX vs Linux

Niezawodność

- Linux (i inne jądra monolityczne)
  - Szacowana liczba błędów w jądrze: 15,000
  - Problemem są głównie sterowniki
    - stanowią większość kodu systemu
    - duże potencjalne źródło błędów
    - muszą być zaufane
  - dużo dynamicznych struktur danych
- Mikrojądra
  - szacowana liczba błędów w jądrze: 24
  - sterowniki w trybie użytkownika
  - brak obcego kodu w trybie jądra
  - wszystkie struktury danych statyczne

MINIX vs Linux

Porównanie wydajności

Na początku 2007 roku Tanenbaum wygłosił wykład na którym przedstawił porównanie wydajności MINIXa z jądrem monolitycznym. Z testów (nieznanego pochodzenia) wynikało, że MINIX traci tylko ok 5-10%.

Niedługo później zaprezentowano inne testy...

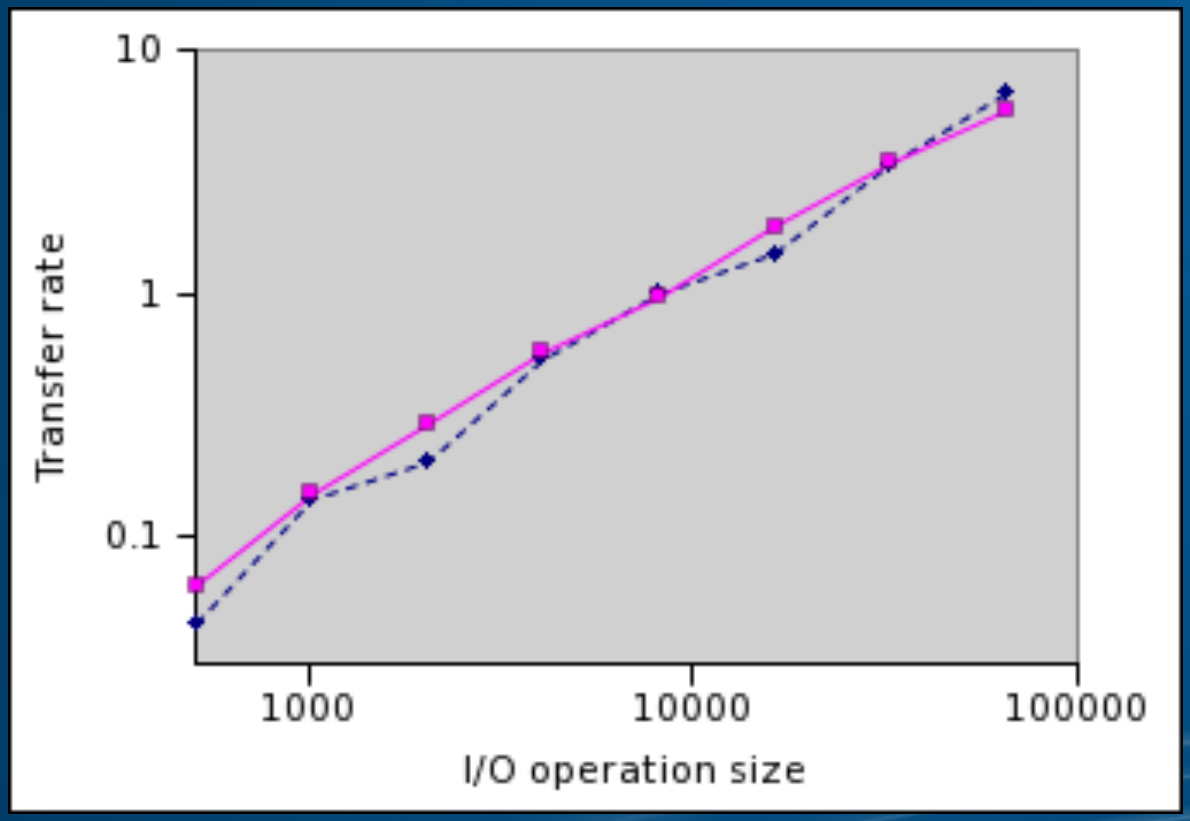
Procesor: AMD Athlon 1700
Dysk: 4GB, ATA

Źródło: http://lwn.net/Articles/220255/

MINIX vs Linux

Porównanie wydajności - test IO

4 procesy niezależnie piszą i odczytują dane z losowych miejsc na dysku

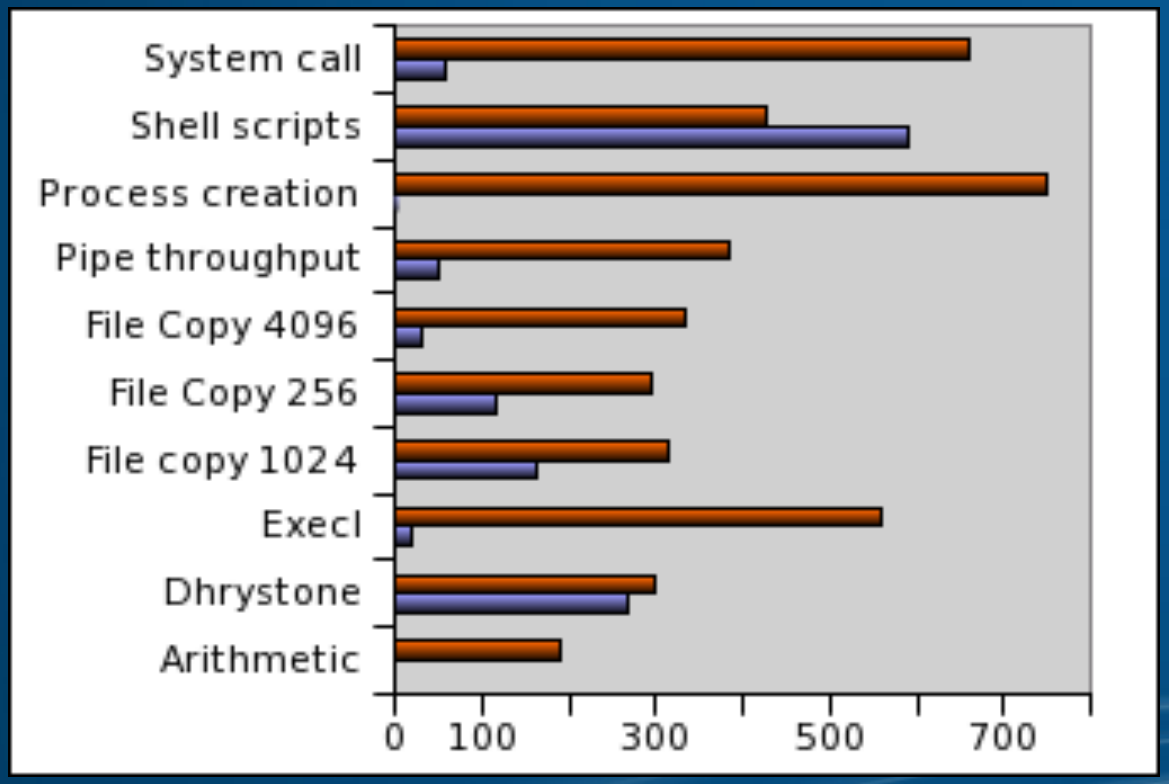


Legenda: Linux MINIX

MINIX vs Linux

Porównanie wydajności - testy UnixBench

UnixBench - narzędzie do testowania
Wyniki punktowe - czym więcej, tym lepiej



Legenda: Linux MINIX

MINIX vs Linux

Porównanie wydajności - podsumowanie

- MINIX nie był tak "tuningowany" jak Linux
- MINIX nie ma kodu dedykowanego na konkretne platformy (nie jest optymalizowany)
- MINIX jest projektem tworzonym przez małe grono osób

Zastosowania mikrojąder

Inne systemy oparte na mikrojądrze

Zastosowania mikrojąder

Gdzie liczy się mały rozmiar i stabilność?

- Systemy wbudowane, np.:
  - Telefony komórkowe
- Część systemów funkcjonujących w przemyśle:
  - serwery

Amoeba

System rozproszony

vrije Universiteit amsterdam

department of computer science
faculty of sciences

- Zapoczątkowany w 1980 roku, przez:
  - Andrew Tanenbauma
  - Zespół Uniwersytetu Vrije w Amsterdamie
- W założeniu: system podziału czas, który sprawia, że cała sieć komputerów wydaje się użytkownikowi być pojedynczą maszyną
- Język programowania Python został oryginalnie zaprojektowany dla tego systemu
- Wykorzystuje specjalny język do tworzenia oprogramowanie na systemy rozproszone: Orca

Amoeba

Cele projektowe

vrije Universiteit amsterdam

department of computer science
faculty of sciences

- Rozproszenie:
  - Podział zasobów i procesów na wielu komputerach
  - Wykorzystuje model puli procesów i protokół FLIP
- Przezroczystość:
  - Wrażenie pracy na jednym komputerze dzielącym czas na procesy
- Współbieżność:
  - W obrębie całej puli oraz pojedynczych procesów
- Wysoka wydajność

BeOS / Haiku

- System do zastosowań multimedialnych
- Efektywne wykorzystanie sprzętu
- W 2001 sprzedany firmie Palm
- Haiku - pisany od nowa, rok - dwa do wydania (2006)

CapROS

Capability-based reliable OS

- Automatyczne zachowywanie stanu
- Żetony uprawnień
- Porozumiewanie wyłącznie z pośrednictwem jądra
- GNOSIS -> KeyKOS -> EROS -> CapROS
- Najstabilniejszy

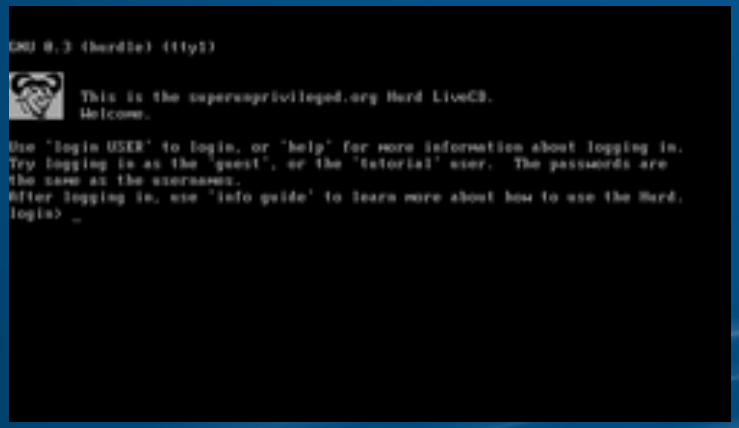
Rodzina mikrojąder L4

- Zapoczątkowana przez Jochane Liedtke na i386, cel: szybsza alternatywa dla jąder Mach
- Początkowo napisane całkowicie w assemblerze
- Synchroniczna komunikacja wątków
- Ich rozwój doprowadził do zwiększenia efektywności mikrojąder
- Wbudowane wsparcie dla parawirtualizacji (stabilność)

GNU/Hurd

Przedstawienie

- HURD - wzajemnie rekurencyjny skrót z HIRD:
  - *HURD = HIRD of Unix-Replacing Daemons*
  - *HIRD = HURD of Interfaces Representing Depth*
- Powolny rozwój, wciąż brak stabilnych wersji



GNU/Hurd Live CD

GNU/Hurd

Translatory

- Przezroczyste translatory
  - przekierowanie wywołań funkcji read() i write()
  - każdy i-węzeł związany z własnym translatorem:
    - kompresji
    - szyfrowania
    - montowania
    - zdalnego dostępu (ftpfs)
    - łączenia katalogów (unionfs)

GNU/Hurd

Translatory

- Cały system działa na translatorach
  - wszystkie operacje na systemie wykonywane przez nie
  - wszystkie serwery są widoczne jako translatory
    - np. StoreIO, ext2fs
- wszystkie abstrakcje sprzętu to translatory
- Translatory można łączyć
  - tar.gz
  - stos protokołów sieciowych

Singularity

Czyli młodszy brat Windows

Microsoft

- Pisany w dedykowany języku - Sing#:
  - Sing# został zaprojektowany do pisania jądra systemu
  - Sing# jest rozszerzeniem C#, powstałym ze Spec#
  - type safe
- IPC zdefiniowane przez formalne *kontrakty*
  - brak potrzeby dodatkowej ochrony pamięci
  - wspólna przestrzeń adresowa aplikacji i jądra
    - uproszczenie przełączeń kontekstu

Singularity

Czyli młodszy brat Windows

Microsoft

- Brak możliwości dynamicznego ładowania kodu (sterowników, wtyczek, itp.):
  - rozszerzenie mogłoby popsuć główny kod
  - każde rozszerzenie musi być osobnym procesem komunikującym się z procesem macierzystym

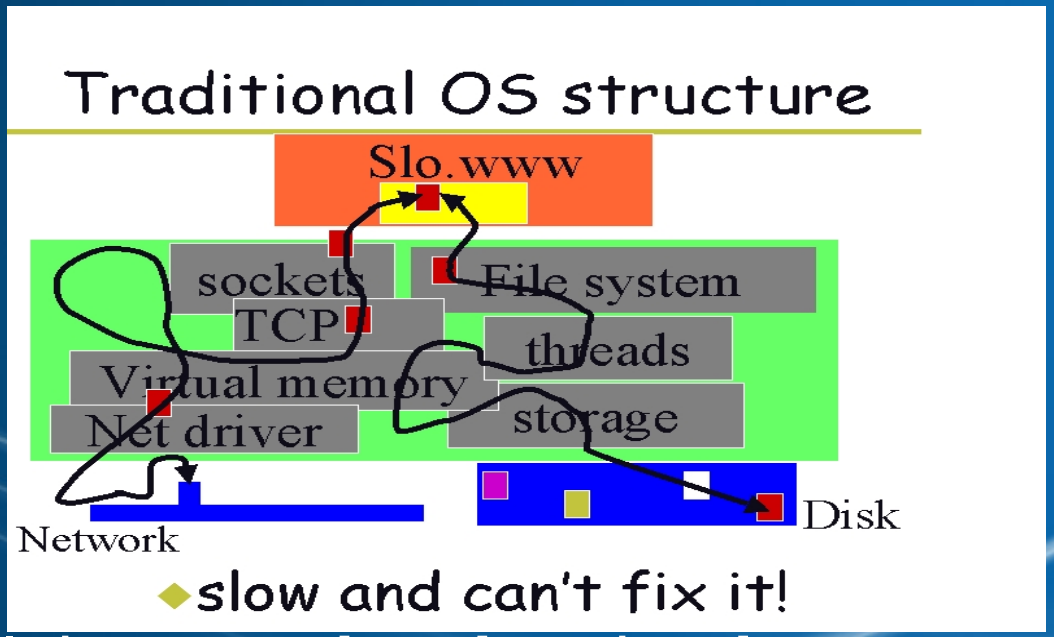
Inne podejścia

Egzojądra

Egzojądra

Exokernels

- motywacja: potrzeba SZYBKIEGO serwera



- oddzielają **zarządzanie** od **ochrony**
  - system w zasadzie zawarty w bibliotekach
- małe jądro z niskopoziomowym interfejsem

Egzojądra

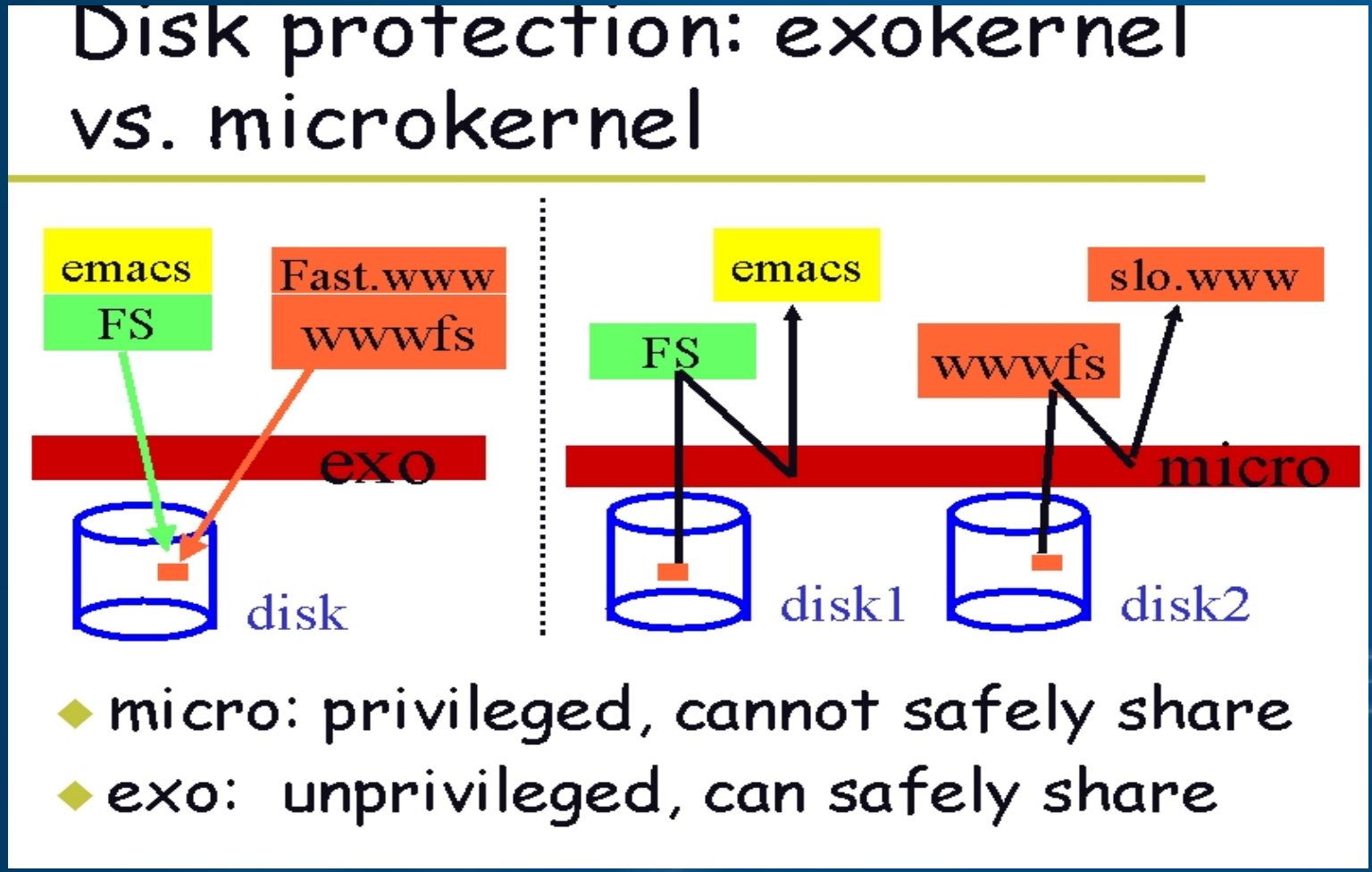
Exokernels

- Jądro skupione na podziale pomiędzy procesy:
  - czasu procesora
  - stron pamięci
  - bloków dyskowych
- Aplikacje użytkowe mogą:
  - same tworzyć abstrakcje sprzętu
  - korzystać z bibliotecznych (wielu różnych)
- Lepsza wydajność (10x) i uniwersalność

Egzojądra

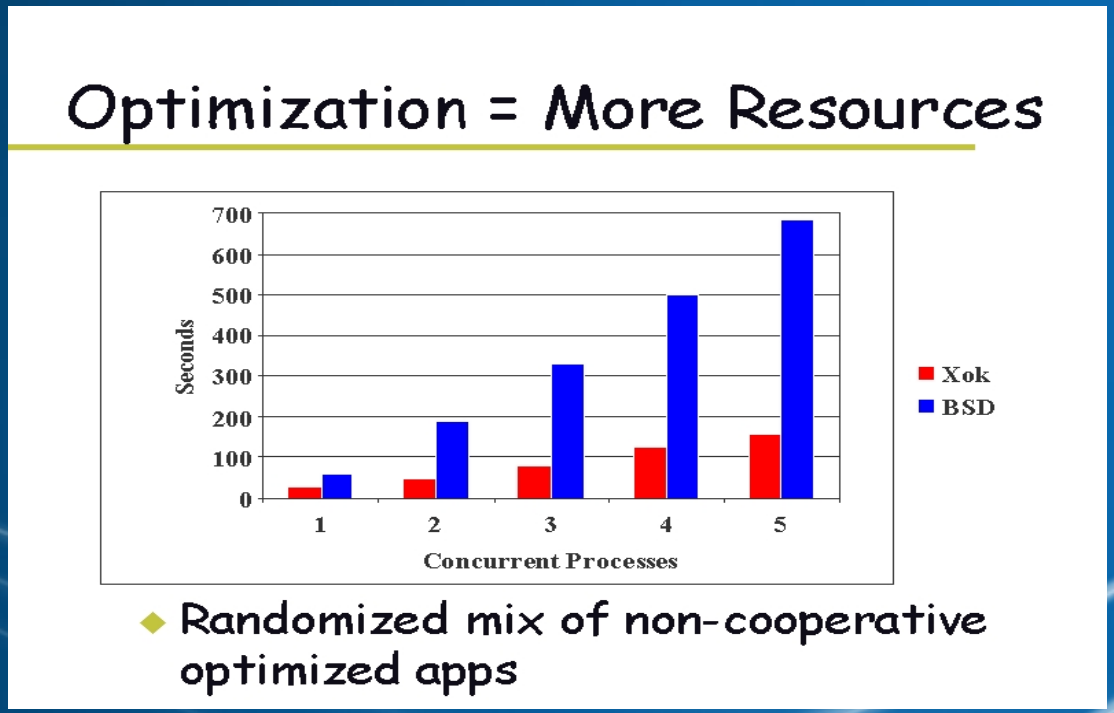
Dostęp do dysku

- Ochrona na poziomie pojedynczych bloków
  - możliwość ich współdzielenia



Egzojądra

Szybkość egzojąder



- Niezmienione aplikacje: **4x szybciej**
- Ogólna szybkość systemu: **4x szybciej**
- Dostosowane aplikacje: **8x szybciej**

Egzojądra

Przykłady systemów

- Stworzone przez *MIT Parallel and Distributed Operating Systems group:*
  - Aegis (prototyp)
  - XOK
    - "Unix jako biblioteka"
- ExOS
- Cheetah web server

Egzojądra

Zestawienie wad i zalet

Zalety

- Mało kodu działa w trybie jądra
- Możliwość współistnienia różnych abstrakcji
- możliwość agresywnych optymalizacji bez utraty ochrony

Wady

- Trudniejsze programowanie
- Biblioteki mogą zawierać błędy
- Nie można odzyskać utraconych danych

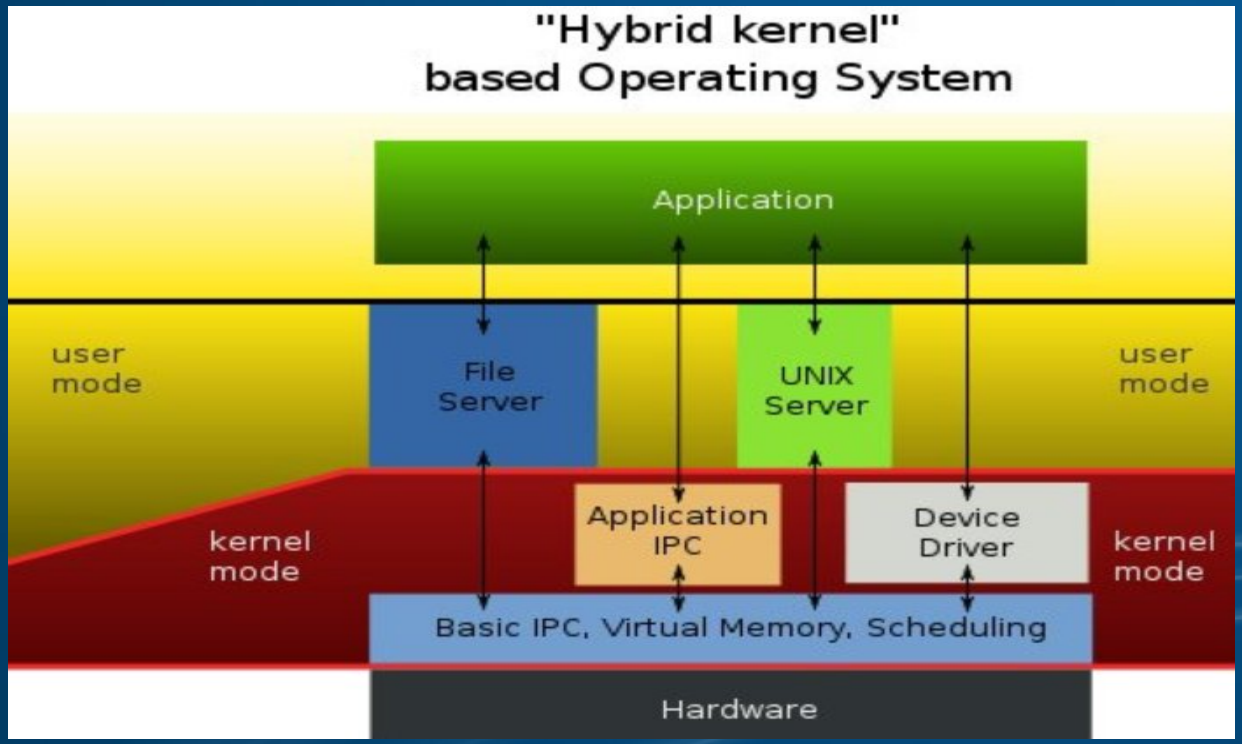
Inne podejścia

Jądra hybrydowe

Jądra hybrydowe

Czyli konsensus (???)

- Quasi-kategoria, często nieuznawana
  - podobne do jąder monolitycznych
- *"Architektura podobna do mikrojądra, ale zaimplementowana jako jądro monolityczne"*



Jądra hybrydowe

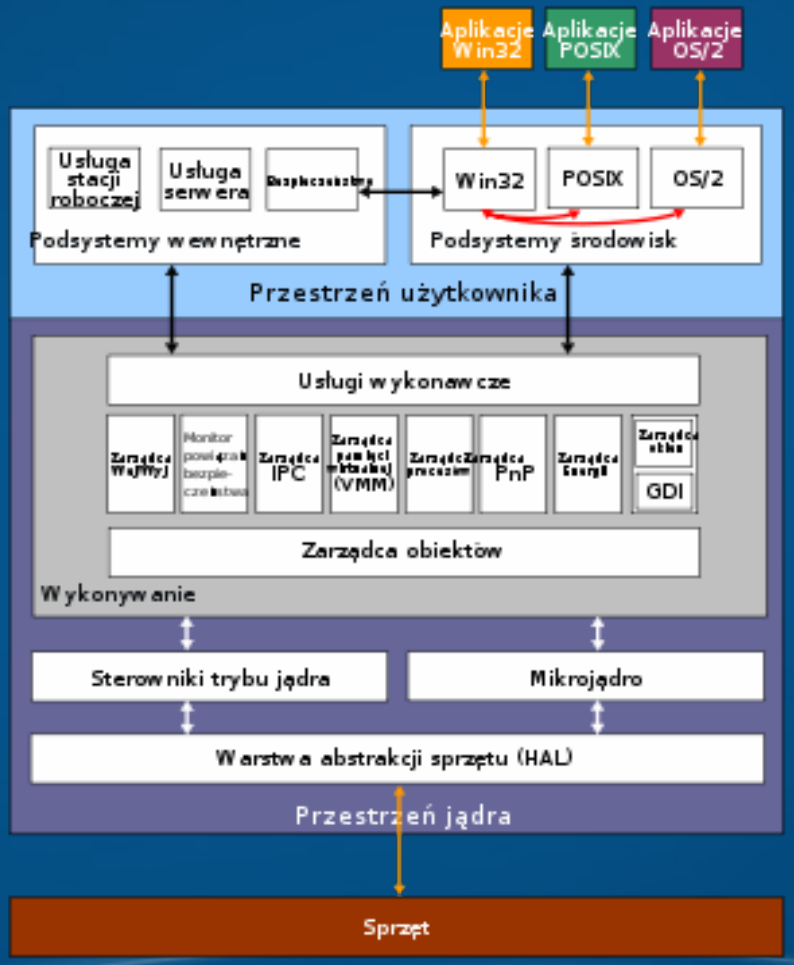
Czyli konsensus (???)

- Lansowane przez... Microsoft
  - inaczej nazywane przez nich "makrojądro"
- Cechy
  - podsystemy vs serwery
    - część usług systemowych (podsystemów) wypchniętych do trybu użytkownika
    - część (większa) w trybie jądra
  - Mikrojądro wewnątrz makrojądra
- Idea krytykowana przez wielu
  - *"it's just marketing"*, Linus Torvalds

Jądra hybrydowe

Przykład - Windows NT

- Tak naprawdę:
  - Windows NT
  - Windows 2000
  - Windows XP
  - Windows Server 2003
  - Windows Vista
  - Windows Server 2008



Architektura Windows NT

Jądra hybrydowe

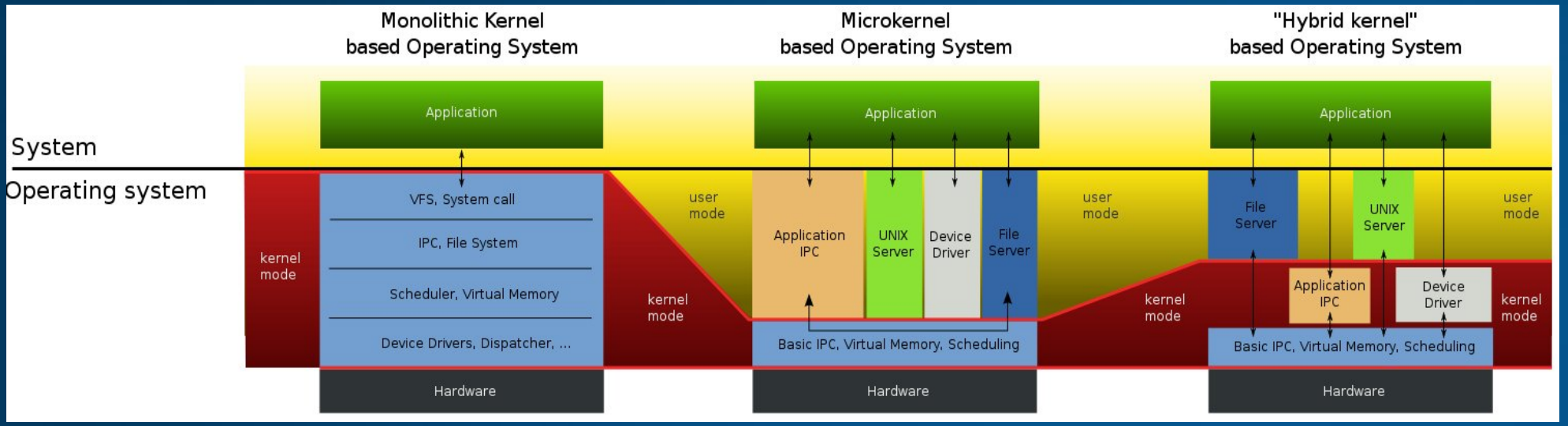
Inne przykłady

- BeOS
  - Haiku
- Syllable
- bazujące na BSD
  - DragonFly BSD
  - XNU
- NetWare
- Plan 9
  - Inferno
  - pokazuje wszystkie zasoby (łącznie z sieciowymi) jako pliki

Podsumowanie

I perspektywy na przyszłość

Podsumowanie różnych podejść



Ogólne idee:

- Monolityczne jądro - abstrakcje sprzętu
- Mikrojądro - komunikacja przez "mini sieć"
- Egzojądro - multipleksowanie zasobów

Perspektywy

Co dalej?

- Coraz więcej zastosowań
- Nacisk nie na wydajność, ale na stabilność
- CapROS - w pełni sprawny system operacyjny
- Zerokernel (Flux)

“As I did 20 years ago, I still fervently believe that the only way to make software secure, reliable, and fast is to make it small. Fight Features.”

Andy Tanenbaum

In short: just say NO TO DRUGS
and maybe you won't end up
like the Hurd people.

Linus Torvalds

Dziękujemy!

Pytania?

Owacje?

ŹRÓDŁA

Najważniejsze

- en.wikipedia.org
- www.minix3.org
- Introduction to Minix (http://osnews.com/story/15960/Introduction-to-MINIX-3/)
- minix3.blogspot.com
- www.minix3.ru
- http://www.microsoft-watch.com/content/operating_systems/microsofts_other_os.html
- http://netbsd-soc.sourceforge.net/projects/hurdt/
- Can We Make Operating Systems Reliable and Secure?; A. Tanenbaum, J. Herder, H. Bos
- Operating Systems: Design and Implementation; A. Tanenbaum