Systemy operacyjne oparte na mikrojądrze

Mateusz Błażewicz Krzysztof Geras Artur Matan

Agenda

* wstęp – czym jest mikrojądro, systemy oparte na mikrojądrze, czym jest Hurd
* mikrojądra w Hurd
    * Mach 4
    * L4, Coyotos
* architektura Hurd
    * Serwery
    * Translatory
* Hurd na żywo
* testy Debian Linux/Debian Hurd
* za i przeciw mikrojądrom

Problem

* kiedyś było mało pamięci – małe jądra systemów operacyjnych
* zwiększenie ilości pamięci i zwiększenie przestrzeni adresowej z 16 na 32 bity
* jądra monolityczne stały się ogromne
* w październiku 2008 długość kodu jądra Linuksa przekroczyła 10 mln linii
* trudno kontrolować coś tak wielkiego
* trudno debugować coś tak wielkiego
* (jak twierdzą zwolennicy mikrojąder) jądro monolityczne może być niestabilne i potencjalnie mało bezpieczne

Mikrojądro
definicja

Mikrojądro (ang. microkernel) to rodzaj jądra systemu operacyjnego, które zawiera tylko najbardziej niezbędne elementy, takie jak:

* funkcje zarządzania wątkami (zarządzanie przydzielaniem CPU)

* komunikacja międzyprocesowa (IPC)

* niskopoziomowe zarządzanie przestrzeniami adresowymi

* (dodatkowo) obsługa przerwań i wyjątków

Liedtke's minimality principle

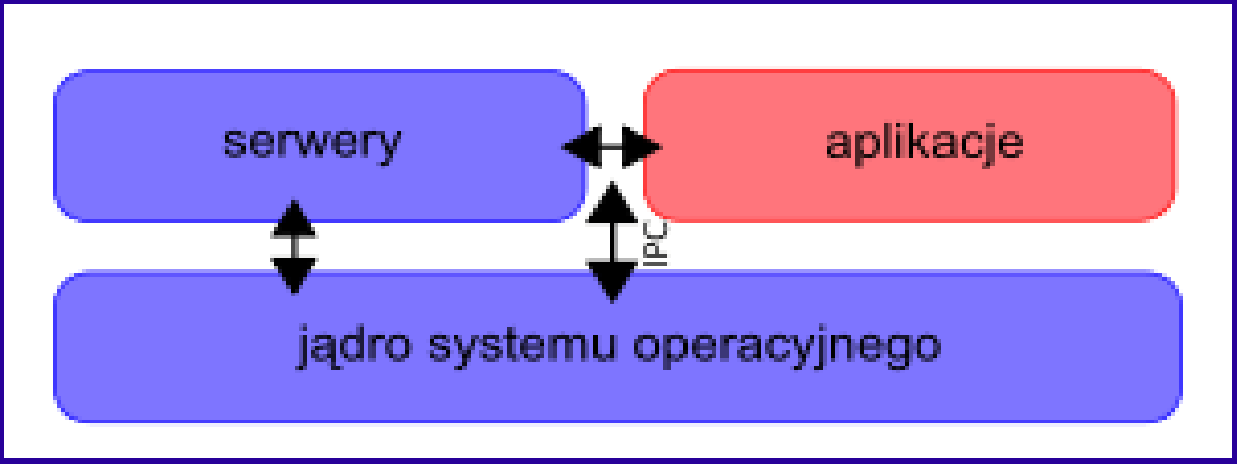
A concept is tolerated inside the microkernel only if moving it outside the kernel, would prevent the implementation of the system's required functionality

Odstępstwa - podstawowe sterowniki, scheduler, menedżery systemu plików wewnątrz mikrojądra

Mikrojądro
definicja

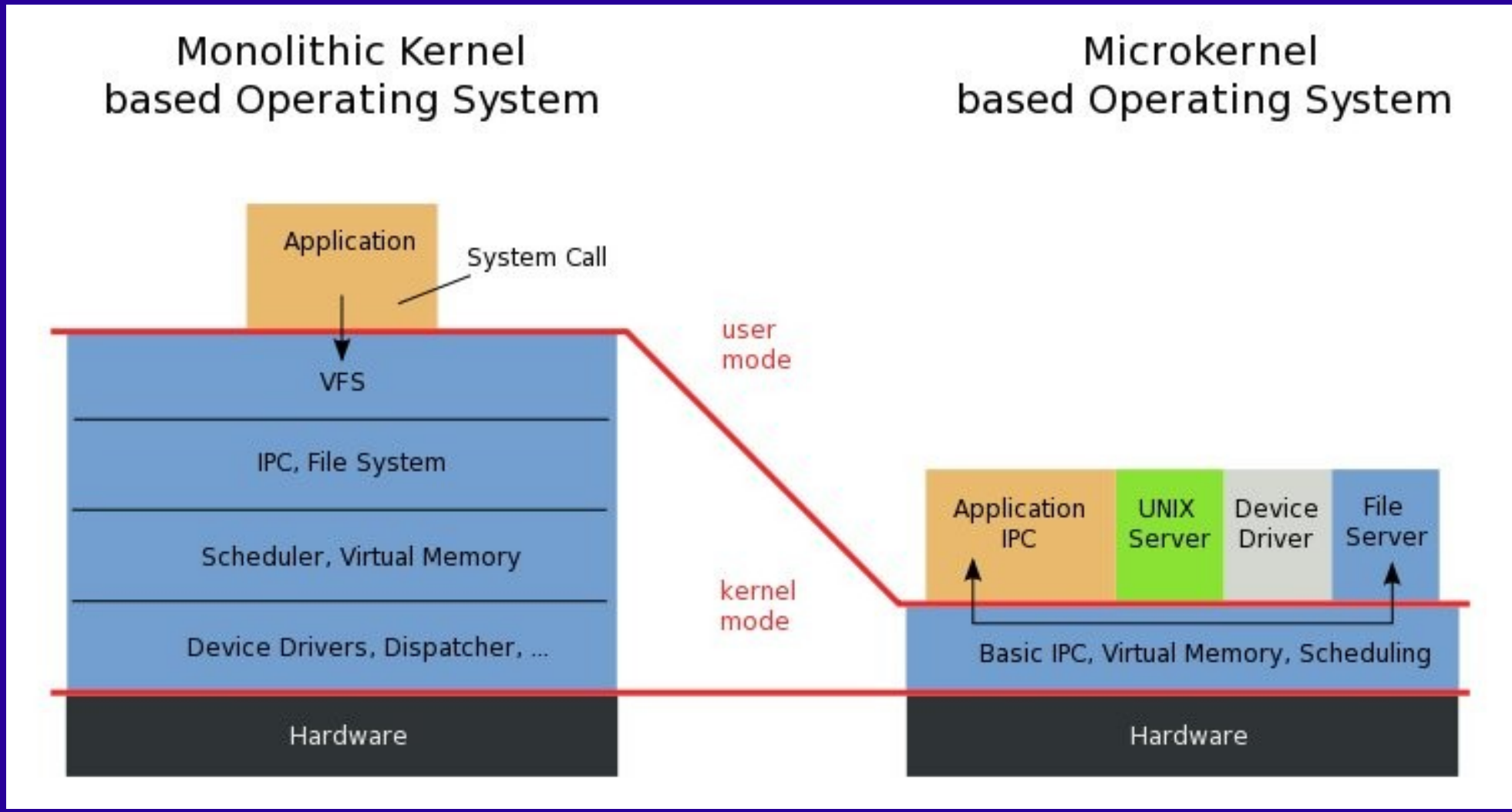
* jeżeli hardware (procesor) ma dwa tryby pracy - użytkownika i systemowy - to jedynie mikrojądro jest wykonywane w trybie systemowym

* wszelkie inne zadania: obsługa systemów plików, urządzeń (sterowniki), interfejsu sieciowego, interfejsu użytkownika realizowane są w przestrzeni użytkownika przez osobne „serwery”



Rozwiązanie - mikrojądro

* prawie wszystko zostało wyrzucone z jądra



Single / Multi server

Single server

* pojedynczy serwer implementuje funkcjonalność systemu operacyjnego

Multi server

* wiele serwerów współpracujących dla stworzenia pełnej funkcjonalności
* pojedynczy serwer implementuje tylko małą, ale dobrze zdefiniowaną część
* odpowiedzialności są logicznie podzielone pomiędzy serwery

Podejście single serwer jest porównywalne z systemem monolitycznym. Ma podobne wady i zalety.

Multi server jest lepszy

* jasno wydzielone odpowiedzialności

* większa stabilność: jeśli jeden serwer ginie, pozostałe zostają

* łatwiejsze rozwijanie: testowanie bez resetowania

* łatwiejsze wprowadzanie zmian i nowych funkcjonalności

Serwery

* programy jak każde inne

* jądro nadaje im pewne przywileje, by mogły działać na fragmentach pamięci fizycznej, które nie są dostępne dla większości programów

* to pozwala serwerom, w szczególności sterownikom urządzeń, oddziaływać bezpośrednio ze sprzętem

Typy serwerów:

* systemów plików
* sterowników urządzeń
* obsługa sieci
* wyświetlania / grafiki
* interfejsu użytkownika

Serwery

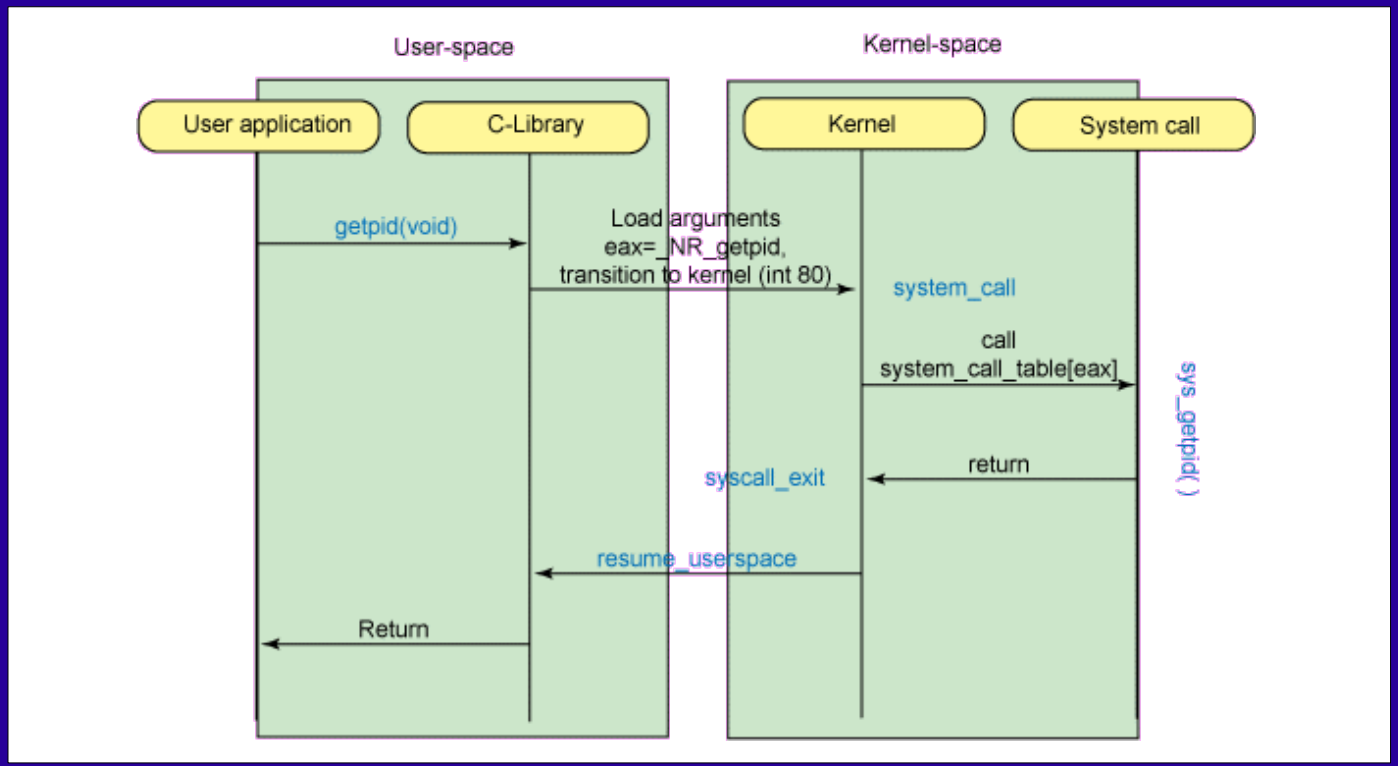
* uruchamiane przy starcie systemu
* zapewniają pozostałym aplikacjom usługi analogiczne do tych, z monolitycznego jądra UNIX
* serwery można tworzyć jak normalne programy
* serwery można uruchamiać jak normalne programy, bez konieczności restartu systemu jak przy jądrze monolitycznym
* w przypadku „zawieszenia” któregoś z serwerów, można go zatrzymać i zrestartować
* techniki bazodanowe dla ułatwienia restartowania: transakcyjność, replikacja, checkpointowanie

Jak wygląda wywołanie systemowe?

W przypadku jądra monolitycznego:

* pojedynczy syscall – 2 context switch'e razem ze zmianą trybu pracy

* jądro może pisać do przestrzeni adresowej danego procesu



Anatomy of the Linux kernel, M.Tim Jones

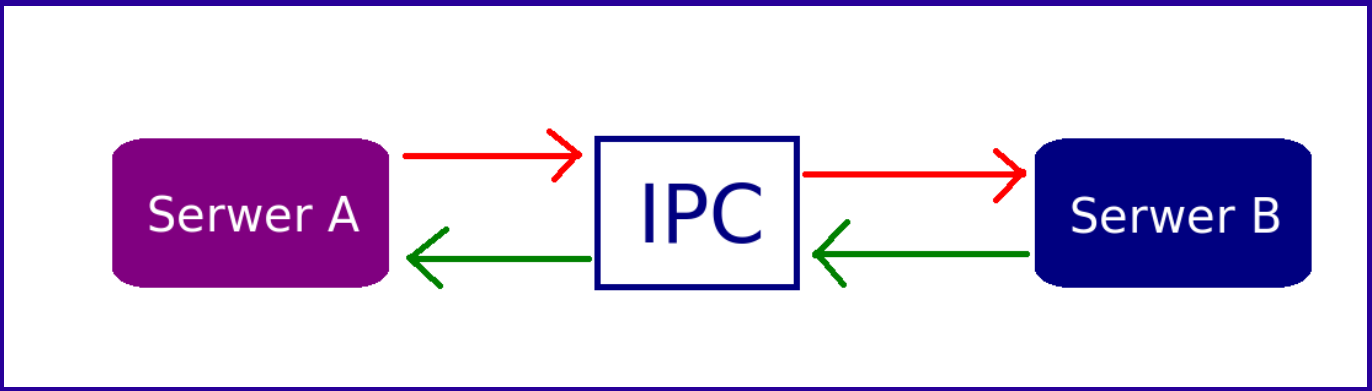
Jak wygląda wywołanie systemowe?

W przypadku mikrojądra

* wysłanie komunikatu do odpowiedniego serwera, odebranie komunikatu (kopiowanie komunikatów możliwe, buforowanie kosztowne etc.)

* w przypadku sterowników w przestrzeni użytkownika, czyli oddzielnych procesów – 2 context switch'e

* w przypadku sterowników w mikrojądrze – tak jak w jądrze monolitycznym (+ IPC)



Wnioski

* syscall'e są droższe w przypadku mikrojąder

* dlatego bardzo ważna jest implementacja IPC

* głównie ona ma wpływ na wydajność systemów opartych o mikrojądra

Bezpieczeństwo

* system oparty na mikrojądrze jest bardziej odporny na błędy (możliwość restartu serwera)

* każdy element systemu ma dostęp tylko do funkcji, do których musi mieć, by zapewnić funkcjonalność

* jedynie mikrojądro wykonuje się w trybie systemowym, to powoduje łatwiejsze zapewnienie bezpieczeństwa

* wniosek: systemy oparte na mikrojądrach są jednymi z najbezpieczniejszych (systemy militarne)

Mikrojądra
lista

* rodzina Mach:

NeXTStep, OSF/1, GNU/Hurd, Apple Rhapsody, Mac OS X, MkLinux, UNICOS Max, ...

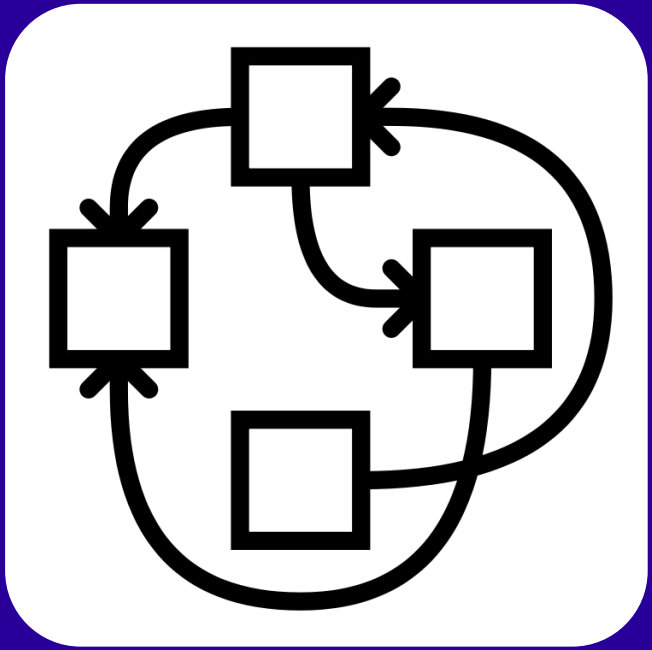
* rodzina L4

* jądro MINIX

* ...

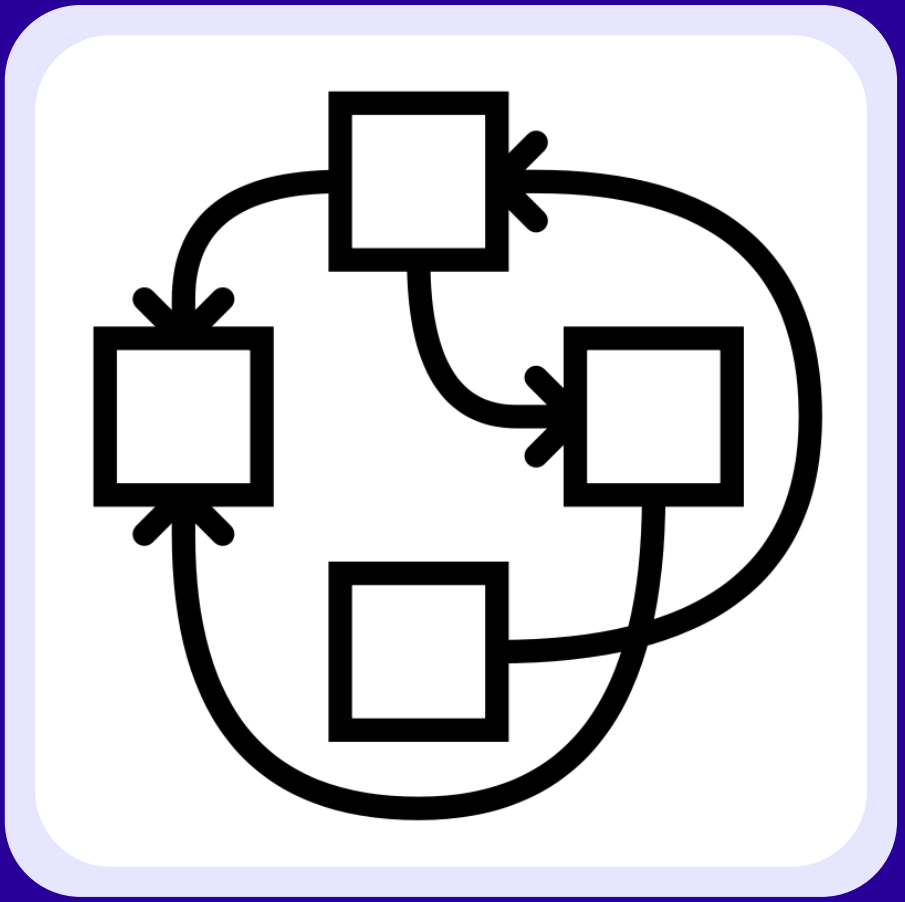
Mikrojądra + serwery = systemy operacyjne

MINIX



GNU Hurd

MINIX



GNU Hurd

GNU Hurd

* projekt został zapoczątkowany przez Free Software Foundation

* miał być darmowym systemem, który zastąpi UNIX

* zestaw serwerów działających pod kontrolą mikrojądra Mach

* wzajemnie rekurencyjny akronim
    * HIRD of Unix-Replacing Daemons
    * HURD of Interfaces Representing Depth

Rozwój GNU Hurd

* 1983: Richard Stallman założył projekt GNU
* 1988: decyzja o użyciu Jądra Mach 3.0
* 1991: wydanie jądra Mach 3.0
* 1991: Thomas Bushnell założył projekt Hurd
* 1994: pierwszy udany boot Hurda
* 1997: wydanie Hurda w wersji 0.2
* 1998: początek Debian hurd-i386
* 2004: decyzja o zmianie jądra Mach na L4
* 2006: decyzja o zmianie jądra L4 na Coyotos

Mikrojądro Mach

* Historia
* Mach jako mikrojądro
* Co zapewnia Mach?
* Nowości
* Problemy
* Wydajność
* Analiza problemów
* Rozwiązania problemów
* Ciekawostki
* GNU/Mach

Mach
Historia

* jedno z pierwszych mikrojąder – I generacja
* opracowany na amerykańskim Uniwersytecie Carnegie-Mellon
* projekt rozwijany w latach 1985-1994 (skończył się na wersji 3.0)
* dalej rozwijany był na Uniwersytecie Utah, gdzie opracowano wersję 4 (do 1996)
* później rozwijany już tylko przy okazji implementacji systemów na nim opartych

Mach
Historia

* początkowo tworzony w systemie BSD jako zamiennik jądra BSD (ewolucja z systemu Carnegie-Mellon Accent – jako zewnętrznego kodu)

* celem były badania nad systemami operacyjnymi – w szczególności równoległymi obliczeniami rozproszonymi

* teraz mnóstwo systemów opartych na nim

Mach

Jako mikrojądro

* zasada minimalizmu – usunięcie z jądra jak największej liczby elementów, które nie są niezbędne do jego działania

* do wersji 2.5 – warstwa wyższego poziomu, z API BSD

* od 3.0 – prawdziwe mikrojądro, warstwa BSD na zewnątrz w przestrzeni użytkownika

Mach
Jako mikrojądro

Co w Machu jest niezgodne z zasadą minimalizmu?

* sterowniki wewnątrz mikrojądra

* zarządzanie prawami plików/portów wewnątrz mikrojądra

* sprawdzanie poprawności komunikatów IPC przez mikrojądro

Mach
Co zapewnia?

* obsługę wieloprocesowości zarówno na jednej maszynie, jak i poprzez sieć
* obsługę wielu aplikacji pracujących w trybie wielozadaniowości z wywłaszczaniem
* wątkowanie działania aplikacji
* bezpośrednią komunikację międzyprocesową (IPC)
* bezpieczną ochronę pamięci

Mach
Nowości

* pojęcie portu – komunikacja IPC, jako odpowiednik pipe'a, tyle że zamiast pliku mamy porty (też z prawami dostępu, widoczne w systemie jak pliki) – proces aby móc skorzystać z kolejki IPC, musi wpierw uzyskać dostęp do danego portu (od jądra)

* IPC

* syscall'e polegają na przesłaniu komunikatów przez IPC, resztę robią odpowiednie serwery

Mach
Nowości

* procesy mogą być wielowątkowe (i to właściwie wątki korzystają z IPC – mogą być wstrzymywane/budzone); wielowątkowość łatwa w implementacji dzięki IPC i portom

* zarządzanie serwerami – możemy w dowolnej chwili uruchamiać nowe, zatrzymywać, kill'ować, restart'ować; czyli możemy dostosowywać dowolnie system „na gorąco”

* wsparcie multiprocessingu; niezależność od architektury

Mach
IPC

* komunikacja międzyprocesowa (IPC) bardzo rozbudowana – komunikaty, porty (kolejki komunikatów)

* [ISTOTNE] IPC wykorzystywane jest nie tylko przez procesy użytkownika, ale także przez samo jądro i jest podstawowym mechanizmem komunikacji

* uwaga: niektóre syscall'e nie są realizowane przez IPC

Mach
IPC

* komunikacja przez IPC jest wygodna, rozwiązuje wiele problemów IPC Uniksa

* przekazywanie komunikatów przez IPC – przez wspólną pamięć (jeden fragment), używa MMU, żeby przekazać fragment pamięci, natomiast kopiowanie tylko w wypadku modyfikacji

Mach
Problemy

* problemy z odnalezieniem portów – system jest bardzo płynny, porty pojawiają się i znikają

* konieczność sprawdzania poprawności komunikatów przez jądro – ze względów bezpieczeństwa

* konieczność sprawdzania praw do portów przez jądro – też ze względów bezpieczeństwa

Mach
Problemy

Problemy z wydajnością IPC:

* pierwsza generacja mikrojąder – np. Mach
Synchroniczne i asynchroniczne IPC

* potrzebne bufory i kolejki komunikatów, trzeba uważać na przepełnienie, wymaga podwójnego kopiowania komunikatów

* to wszystko powoduje kiepską wydajność

Mach
Wydajność

* problemy z wydajnością – implementacja UNIX'a na Mach 3 – 50% wydajność, w innych przypadkach wydajność spadała o niemal 80% (średnio 66%); powód – nadużycie IPC

* przykład syscall'a – UNIX 21 microsec, port na Mach'u 114, z czego 18 sprawy sprzętowe

Mach
Analiza problemów

Problemy z wydajnością nie wynikały z samego IPC (narzut 77%), ale z implementacji (IPC):

* problemy z mapowaniem pamięci

* 80% czasu komunikacji IPC zajmowały dodatkowe czynności przeprowadzane przez mikrojądro na komunikatach – sprawdzanie praw portów, sprawdzanie poprawności komunikatów

Mach
Analiza problemów

Co jeszcze wpływa na niską wydajność:

* w przypadku Macha syscall robi aż 4 context switch'e, 4 przemapowania pamięci, 2 sprawdzenia komunikatów i praw dostępu do portów

Mach
Analiza problemów

Inne przyczyny niskiej wydajności:

* problemy przy braku fizycznej pamięci i potrzebie stronicowania (nie wiadomo, co wywalić z pamięci, bo mikrojądro nie ma pojęcia, co system ma teraz wykonywać – rozwiązanie – Mach 3.0: przerzucenie wyboru stron do wywalenia na serwery, tyle że komunikacja IPC spowodowała jedynie dalszy spadek wydajności...)

Mach
Analiza problemów

Kiepska obsługa wieloprocesorowości

* był projektowany, gdy pamięć była relatywnie szybka względem procesora, tymczasem potem ta relacja definitywnie się zmieniła – IPC mocno cierpi

Mach
Rozwiązania problemów

* serwery powinny być niezależne między sobą (zmniejszenie liczby komunikatów IPC między nimi)

* można używać pojedynczego serwera - jądra innego systemu, zapewniając funkcjonalność:
    * początkowo rozwijamy je w przestrzeni użytkownika – korzystając z dobrodziejstw Macha
    * zdebugowany serwer przenosimy do przestrzeni jądra, żeby uzyskać lepszą wydajność
    * taki system zwany jest systemem co-located

Mach
Rozwiązania problemów

* mach 4 częściowo rozwiązuje problemy (stosowanie sprytnych remapów pamięci)

* jednakże wydajność nadal jest niska...

Mach
Ciekawostki

* pliki wykonywalne dla Macha – format Mach-O

Mach
GNU/Mach

* zmodyfikowana wersja Mach 4, aktualna wersja GNU/Mach'a -1.3
* implementacja, jak dotąd, tylko na IA-32
* licencja GPL
* brak obsługi SMP (w Mach'u zresztą też)
* mikrojądro używane aktualnie przez GNU Hurd'a
* format wykonywalny - ELF

Mikrojądro L4

* ewolucja mikrojąder do L4
* historia L4
* nowości
* wydajność
* ciekawostki

L4
Ewolucja mikrojąder

* druga generacja mikrojąder – np. L4
* przede wszystkim synchroniczne IPC
* trzeba uważać na blokady; przy zastosowaniu paru tricków, możemy zyskać na wydajności

L4
Ewolucja mikrojąder - tricki

* IPC system call – synchroniczne przesyłanie komunikatów przez rejestry procesora (ile się da)

* bezpośrednie przełączanie procesów – niepełny context switch nadawca>odbiorca w czasie korzystania z IPC (dzięki temu i IPC syscall'owi unikamy kopiowania i odpalania scheduler'a (szczególnie przydatne przy RPC))

L4
Ewolucja mikrojąder – tricki

* procesy, które się blokują na IPC, są umieszczane na początku listy procesów gotowych, w celu przyśpieszenia komunikacji IPC

* trochę problemów z timeout'ami, żeby to jakoś funkcjonowało (deadlock'i)

* (dodatkowo mechanizm asynchronicznego IPC – mechanizm notyfikacji podobny do signal'a – bez danych, bez buforowania) – czasem przydatny

L4
Historia

* druga generacja mikrojąder
* stworzony przez Jochena Liedtke (x86, MIPS, Alpha) – Niemcy, IBM T.J. Watson Research Center, Uniwersytet Karlsruhe
* oryginalny L4 napisany całkowicie w Assemblerze – maksymalna wydajność (rezygnacja z przenośności)
* cel – optymalizacja IPC

L4
Historia

* właściwie L4 stanowi całą rodzinę mikrojąder
* oryginalne mikrojądro zostało przeniesione na wiele platform
* zostały dodane reguły bezpieczeństwa i odporności na błędy (których w L4 standardowo nie ma)
* nadal intensywnie rozwijane

L4
Nowości

* w stosunku do Macha zrezygnowano z wielu zabezpieczeń sprawdzanych przez mikrojądro (przerzucono to na aplikacje)
* absolutna zasada minimalności, wszystko co się da przeniesione do przestrzeni użytkownika
* L4 wewnątrz 7 funkcji i używa 12k pamięci
* Mach wewnątrz 140 funkcji i używa 330k pamięci
* PRAWDZIWE mikrojądro

L4
Nowości

Co właściwie L4 zawiera?

* model wątku
* mechanizm synchronicznego IPC
* mechanizm schedulera
* zarządzanie przestrzeniami adresowymi

L4
Wydajność

* efekty – wzrost wydajności – nawet 20-krotny w stosunku do Macha, i szybszy od monolitycznych jąder (mowa o syscall'ach), ale dodatkowe koszty w aplikacjach
* przy systemie typu co-located (MkLinux) - czyli odpalanym na mikrojądrze jako pojedynczy serwer, straty w wydajności są następujące:
    * L4 (z zabezpieczeniami) – 5%-10%
    * Mach - 15%
* czyli niewielkie zważając na podwójne context switch'e

L4
Ciekawostki

* na L4Ka::Pistachio – rekord w szybkości przekazywania komunikatu – implementacja na architekturę Itanium 36 cykli
* wombat – wysoce przenośna wersja Linux'a, oparta o L4
* na architekturze Xscale – 30 krotnie tańsze context switch'e w stosunku do normalnego Linux'a

L4
Ciekawostki

* na L4 udowodniono, że sterowniki urządzeń w przestrzeni użytkownika, nie muszą powodować spadków wydajności

* jądro Linuksa zostało przeniesione na L4, Hurd może w przyszłości

* zastosowanie w komórkach, innych systemach „embedded”

Mikrojądro Coyotos

* stworzone dla systemu o tej samej nazwie, skupiającym się przede wszystkim na bezpieczeństwie

* pierwszy system, który ma być całkowicie formalnie zweryfikowany

* być może w przyszłości mikrojądro zostanie użyte jako mikrojądro Hurda (prace od połowy 2006 roku...)

Serwery w Hurdzie

Serwery zapewniają funkcjonalność POSIX API. Lista podstawowych (w sumie 24):

* auth (authentication server) – odbiera zgłoszenia i hasła od programów, zwraca im ID, które zmienia prawa danego programu

* crash (crash server)

* exec (execution server) – tłumaczy obraz wykonywalny (ELF lub a.out) do wykonywalnego obrazu w pamięci

Serwery w Hurdzie

* Fifo (FIFO translator)
* New-fifo (new FIFO server)
* Firmlink (the firmlink translator)
* Fwd (forward server)
* Hostmux (host multiplexer server)
* Ifsock (server for sockets interface)

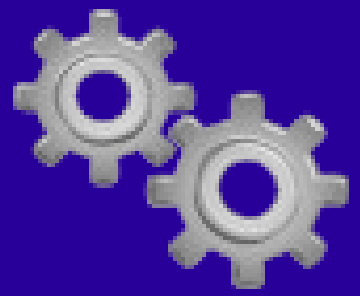
Serwery w Hurdzie

* Init (init server)
* Magic (magic server)
* Proc (process server)
* Pfinet (pfinet server) – odpowiada za sieć
* Pflocal (pflocal server) – odpowiada za sieć
* Term (terminal server)
* Usermux (user multiplexer server)

Translatory
Co to takiego?

* szczególny typ serwerów, dostarczających podstawowe operacje na plikach
* „filtry” przypisywane do konkretnych węzłów w systemie plików
* zwykłe procesy w przestrzeni użytkownika, bez dodatkowych uprawnień

Translatory
Jak to działa?



* każdy dostęp do węzła przetwarzany przez translator
* wykonanie operacji na węźle = wywołanie funkcji translatora
* duża dowolność implementowanej funkcjonalności:
  * odczyt nie musi oznaczać fizycznego dostępu do zawartości pliku na dysku
  * plik może zachowywać się dla jednych operacji jak katalog, dla innych jak zwykły plik
* Działanie translatorów ograniczone tylko fantazją programistów

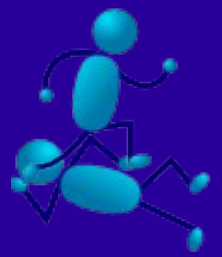
Translatory
Aktywne czy pasywne?

Translatory aktywne:

* uruchamiane w chwili przypisania do węzła
* gdy giną – przypisanie znika

Translatory pasywne:

* przypisywane do węzła na stałe
* uruchamiane przy odwołaniu do węzła
* niewrażliwe na śmierć procesu translatora (np. reboot)
* mogą być przykrywane przez aktywne



Translatory
Podstawowe zastosowania

* translatory różnych systemów plików: ext2fs, fatfs, isofs
* translator sieciowego systemu plików (nfs)
* translator FTP (ftpfs) – pliki zdalne dostępne jak lokalne
* translatory urządzeń
* translator symlinków
* pfinet – translator do protokołu tcp/ip

Translatory
Ciekawe zastosowania

* Xmlfs – translator plików XML
* Magic – dostęp do deskryptorów aktualnego procesu jak do plików
* Mboxfs – dostęp do poczty przez system plików, można sortować maile (tylko odczyt)

Translatory
Obecnie rozwijane

* emailfs – translator do pisania i wysyłania maili
* cvsfs – translator do obsługi cvs'a
* procfs – symulacja unixowego katalogu proc
* tmpfs

Translatory
Pomysły na przyszłość

* tłumacz – zapis do pliku w jednym języku, odczyt tekstu przetłumaczonego na drugi język
* kompilator
* wyszukiwarka plików
* filtr antyspamowy
* zarządca pakietów – instalowanie pakietów poprzez kopiowanie do odpowiedniego katalogu

Debian Hurd
Pierwsze kroki

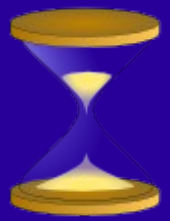
* skąd pobrać?
  * http://debian.org/hurd
  * apt-get install crosshurd
* proces instalacji
  * z płyty instalacyjnej
  * spod innego systemu
* pierwsze kroki
  * stworzenie urządzeń
  * ustawienie translatorów
* konfiguracja

Debian Hurd
Live

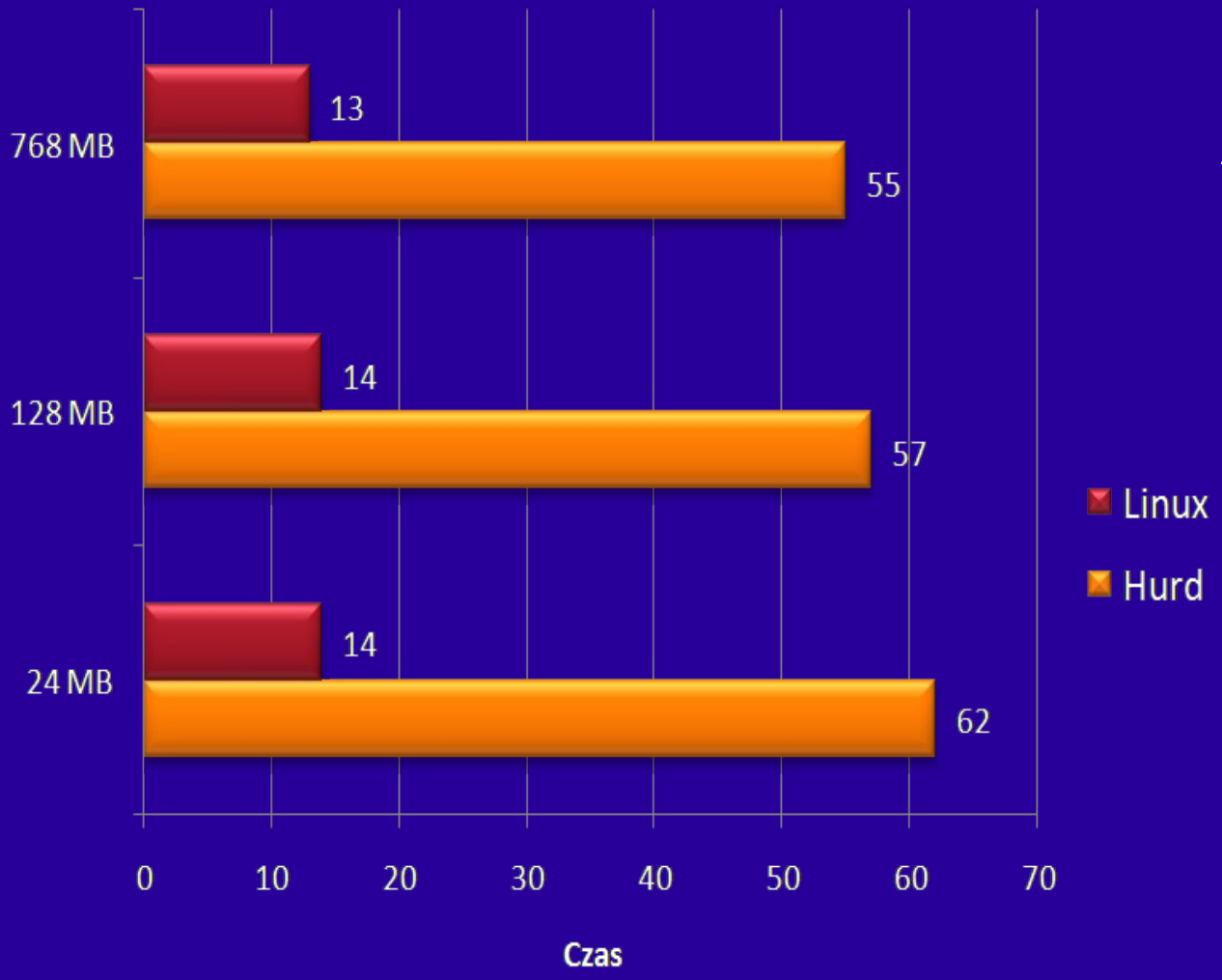
* popularne programy
  * Midnight Commander
  * Lynx
* translator /dev/hurd-quotes
* środowisko graficzne (WindowMaker)
* niespodzianka

Testy
Debian HURD vs Debian LINUX

Użyta konfiguracja:

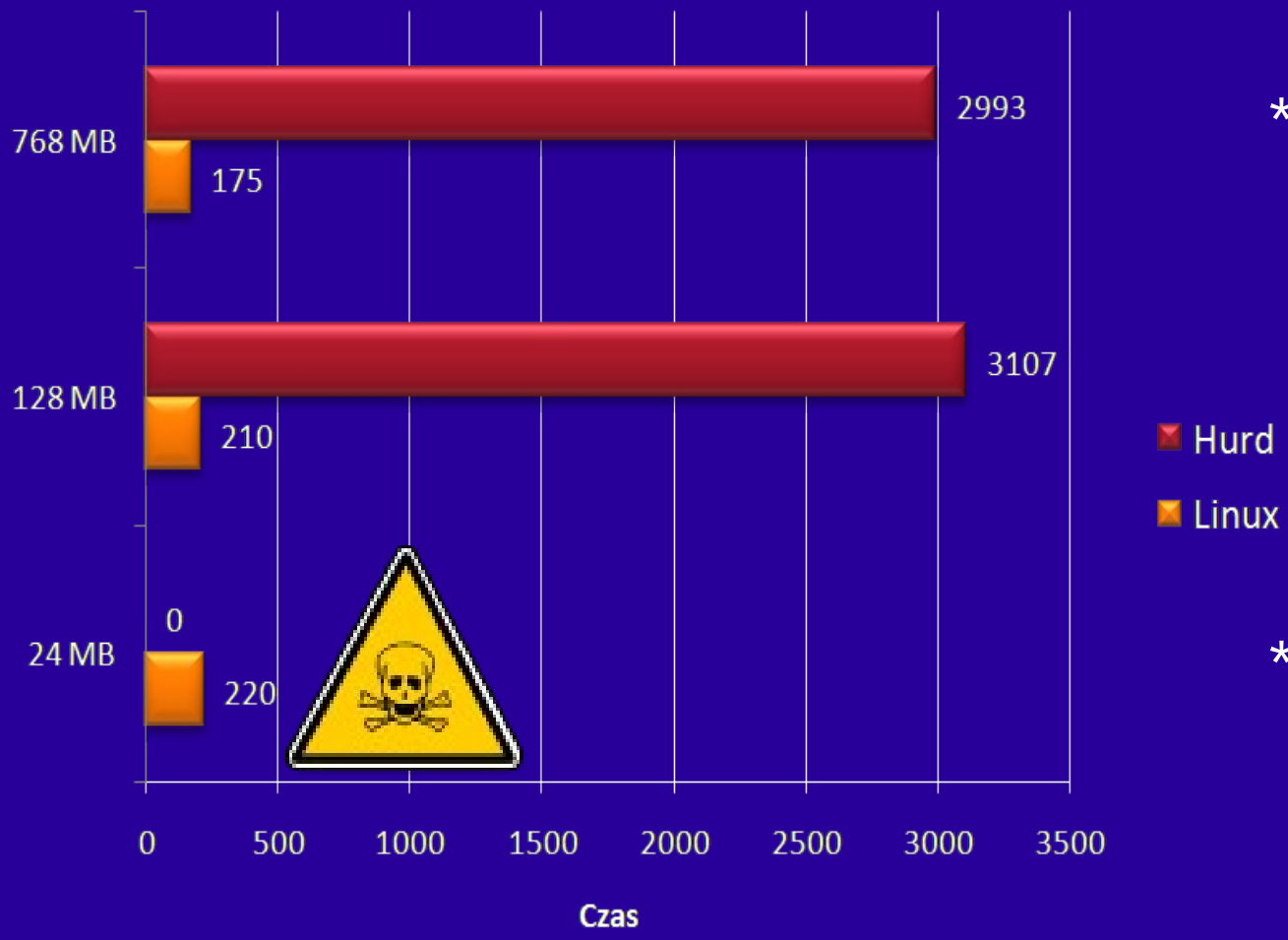
* system host: Debian (Etch)
* wirtualizacja: qemu + kqemu
* pamięć: 24MB, 128MB i 512MB
* podstawowy zestaw pakietów Debiana
* 

Test 1
Pakowanie plików



* spakowanie plików do archiwum (tar + gzip) (40 MB)
* Hurd ponad 4 razy wolniejszy

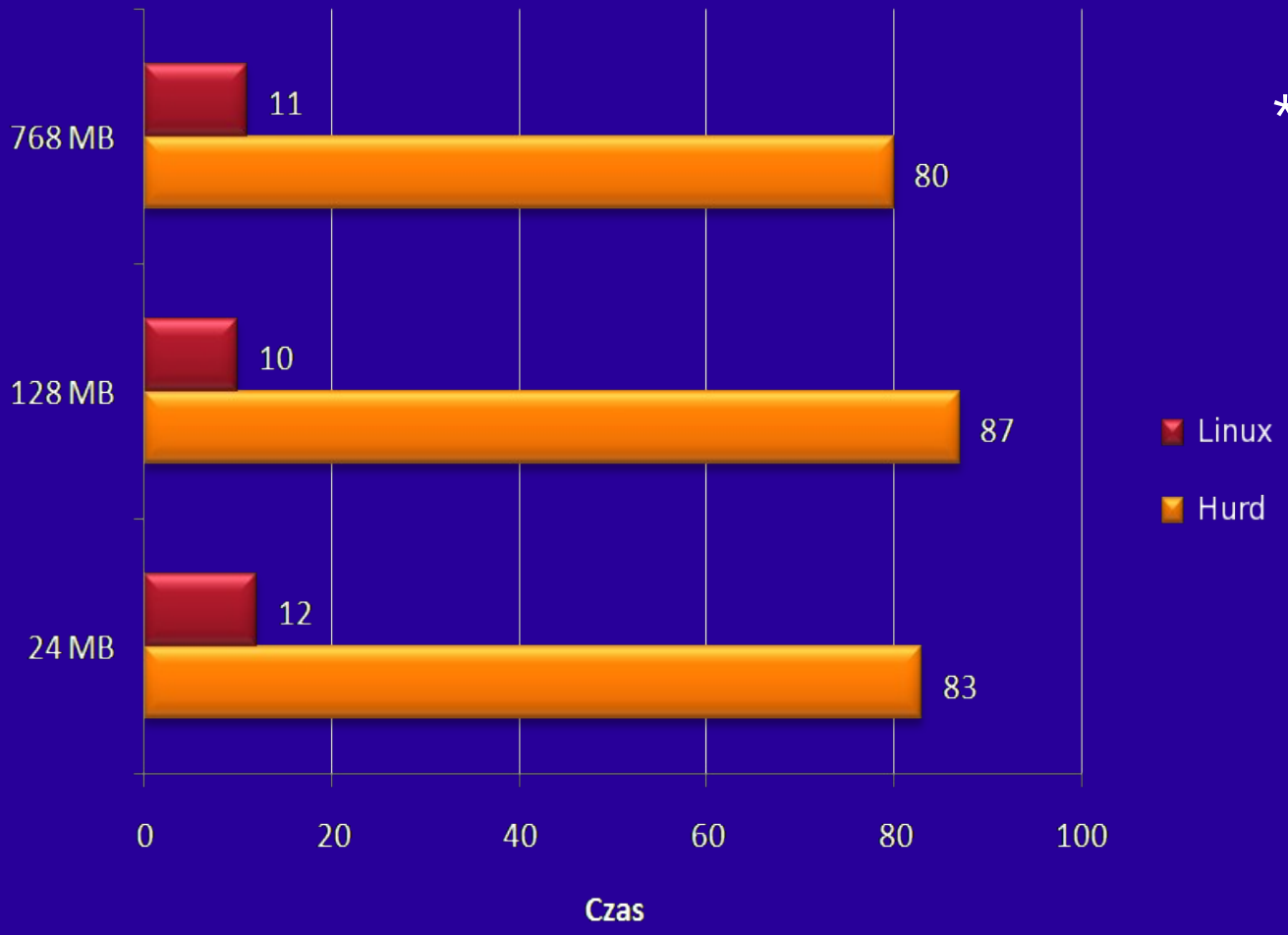
Test 2
Rozpakowanie plików



* rozpakowanie dużego archiwum .tgz (~1500 MB)
* Hurd około 15 razy wolniejszy...

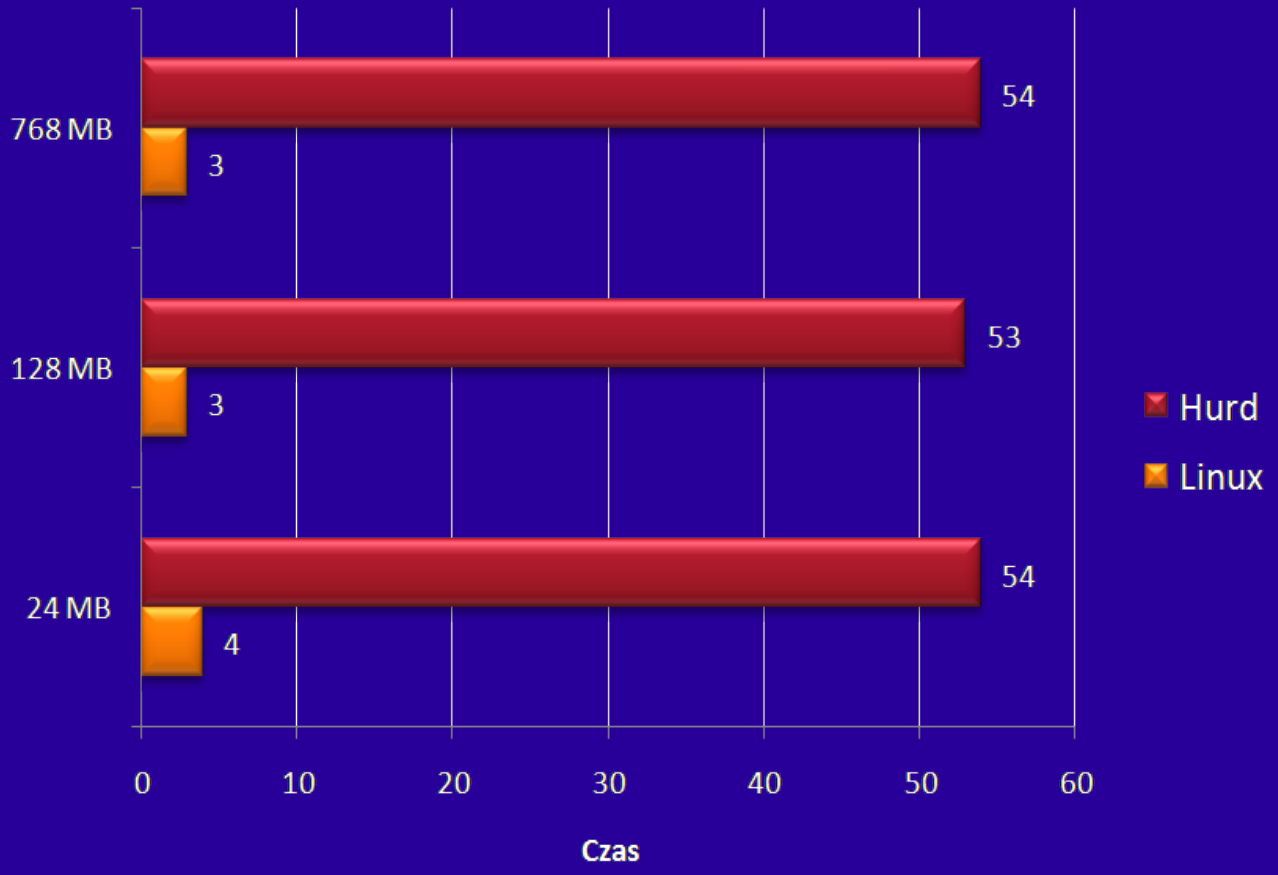
* ... o ile w ogóle ukończy test

Test 3
Wyszukiwanie plików



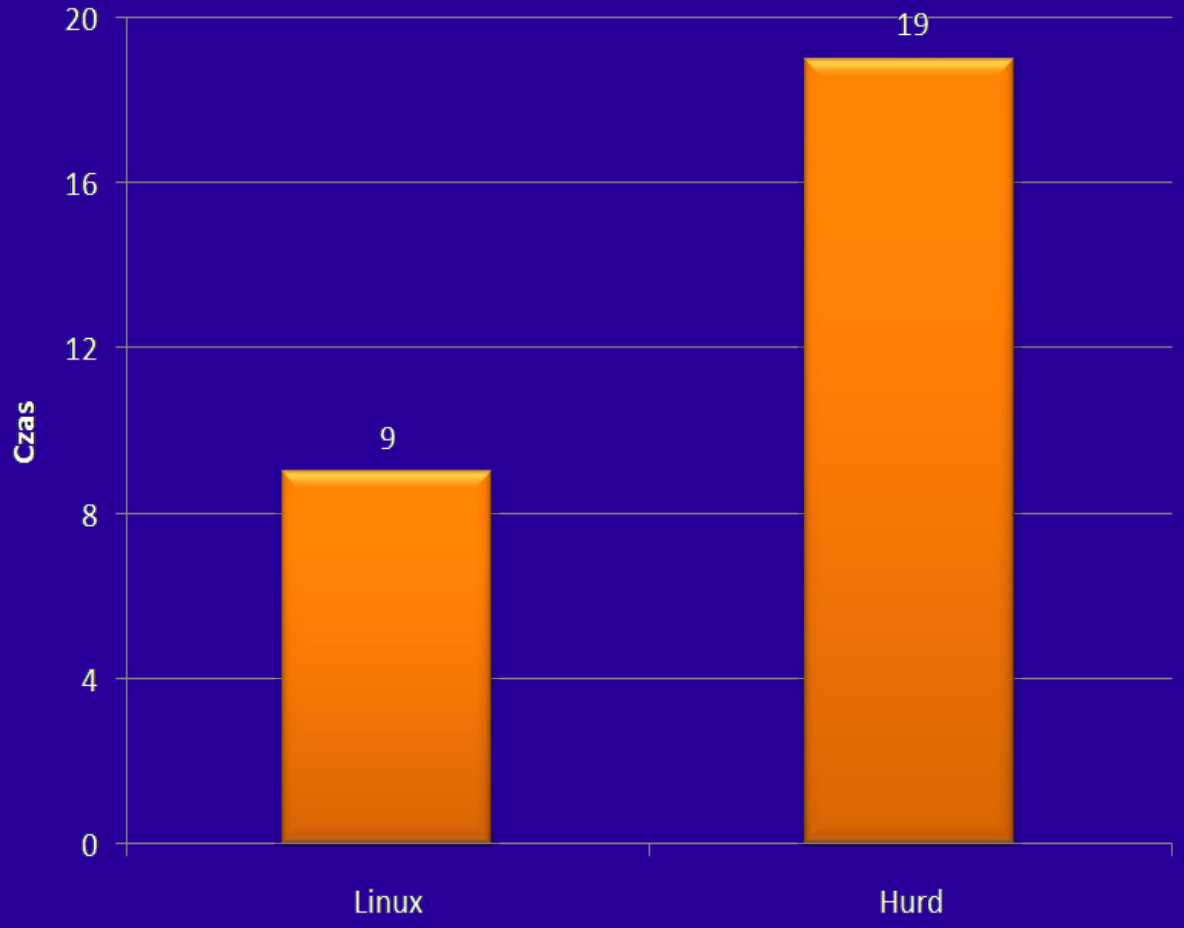
* wyszukanie pliku w rozbudowanym drzewie plików
* Hurd 8 razy wolniejszy

Test 4
Kopiowanie plików



* kopiowanie pliku (45 MB) w obrębie jednej partycji
* Hurd ponad 15 razy wolniejszy

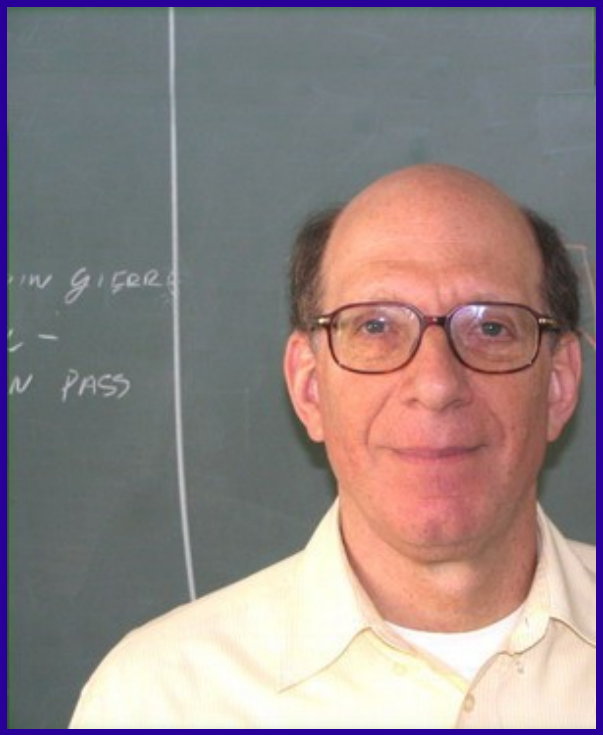
Test 5
startx



* uruchomienie środowiska graficznego i managera okien
* Hurd tylko dwa razy wolniejszy

* ale potrzebuje dużo więcej RAMu

Torvalds vs. Tannenbaum

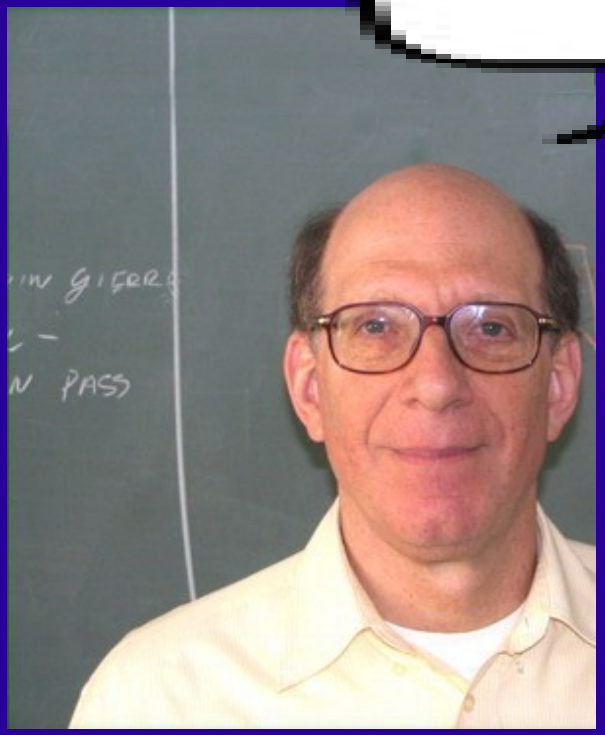


Andrew Tannenbaum

Linus Torvalds

Torvalds vs. Tannenbaum

* Linux jest przestarzały



Andrew Tannenbaum

* Linux jest bardzo zależny od architektury
* jedynym argumentem za monolitycznością jest wydajność a testy wykazały, że mikrojądro może być równie szybkie

Torvalds vs. Tannenbaum

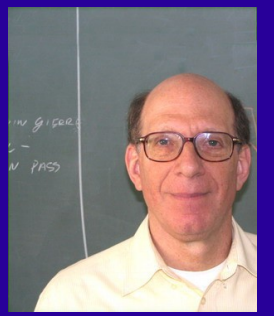
* MINIX jest nastawiony na bycie nieużywalnym dziełem sztuki

Linus Torvalds

* koncepcja mikrojądra jest dobra, ale tylko do nauki SO
* Linux się szybciej rozwija
* nie liczy się wnętrze systemu, tylko to, co widzi użytkownik
* Linux działa lepiej

Torvalds vs. Tannenbaum

While I could go into a long story here about the relative merits of the two designs, suffice it to say that among the people who actually design operating systems, the debate is essentially over. Microkernels have won.



Torvalds vs. Tannenbaum

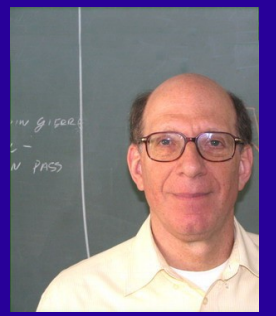
If the GNU kernel had been ready last spring, I'd not have bothered to even start my project: the fact is that it wasn't and still isn't. Linux wins heavily on points of being available now.

Torvalds vs. Tannenbaum

Portability is for people who cannot write new programs.

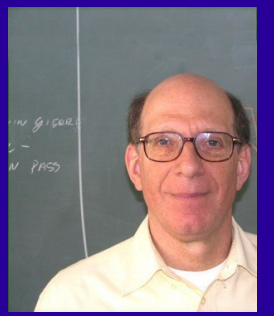
Torvalds vs. Tannenbaum

Of course 5 years from now that will be different, but 5 years from now everyone will be running free GNU on their 200 MIPS, 64M SPARCstation-5.



Torvalds vs. Tannenbaum

A multithreaded file system is only a performance hack. When there is only one job active, the normal case on a small PC, it buys you nothing and adds complexity to the code.



Torvalds vs. Tannenbaum

If you write programs for linux today, you shouldn't have too many surprises when you just recompile them for Hurd in the 21st century.

Hurd - za i przeciw

* ciekawa idea, brak dobrej realizacji
* Hurd powstaje już 20 lat...
* wciąż brak wersji stabilnej
* brak spójnej wizji rozwoju: wiele gałęzi
* mało prężna społeczność użytkowników i developerów

Hurd - za i przeciw

Aktywni developerzy

There are a couple of different Hurd FAQs. There are a number of IRC channels and several different mailing lists with searchable archives.

Before asking a question on a mailing list or on IRC, first, please try to answer your own question using a search engine and reading the introductory information. If you have done this and you cannot find the answer to your question, feel free to ask on a mailing list or on IRC.

Running the Hurd

The most functional distribution of the Hurd is the one provided by Debian. Find more information about it at the Debian GNU/Hurd website.

There are various possibilities of running a GNU/Hurd system.

And these web pages are a living proof of the usability of the Hurd, as they are rendered on a Debian GNU/Hurd system. More people using GNU Hurd in production can be found on ?WhoRunsGNU.

Current Status

There has not yet been an official 1.0 release. The Hurd is developed by a few volunteers in their spare time. The project welcomes any assistance you can provide. Porting and development expertise is still badly needed in many key areas.

Functional systems are installable in a dual-boot configuration. Development systems are currently mostly based on the Debian GNU/Hurd port sponsored by the Debian project.

Community resources for related projects focus around these pages, http://hurd.gnu.org/, the mailing lists and the IRC channels.

If you want to see the current discussions in the Hurd project, please have a look at the bug-hurd mailinglist archives.

For more details, please read our writeup on the current state of the GNU Hurd.

How is this site arranged?

The menu on the upper right corner provides a rough structuring about the available content. Just follow those topics and explore these pages.

Further information about this site and how it was created can be found in the colophon.

Next: Requirements, Previous: Top, Up: Top

1 About this document

Next: Topic, Up: About this document

1.1 Conventions

The Version of this document follows the convention <hurd version>_<document release>. This means that 0.2_7 is the seventh release since Hurd 0.2 and a version like 0.4_1 would be the first release for Hurd 0.4 (which, of course, is not yet available :)).

`$(HURD)` means the top directory of the `Hurd' source tree. `$(GNUMACH)` means the top directory of the `GNU Mach' source tree. `$(MIG)` means the top directory of the `MIG' source tree. `$(GLIBC)` means the top directory of the `GNU Libc' source tree.

Hurd - za i przeciw

* nie może być substytutem dla pełnego systemu operacyjnego
* mnogość serwerów i mikrojądro Mach powoduje spowolnienie wykluczające używanie GNU Hurd w praktyce
* bugi
* niekompletna dokumentacja
* ciekawa zabawka dla hackerów (Google Summer of Code), ale nie zwykłego użytkownika komputera

Mikrojądra - za i przeciw

Zalety:

* stabilne
* skalowalne
* (powinno być) szybkie, gdy nie ma wielu serwerów
* łatwe dopasowanie do konkretnego sprzętu

Mikrojądra - za i przeciw

Zastosowania:

* akademickie (MINIX)
* wojskowe
* systemy wbudowane
* parawirtualizacja w maszynach wirtualnych
* Mac OS X
* Singularity

Bibliografia

* http://www.gnu.org/software/hurd/hacking-guide/hhg.txt
* http://www.gnu.org/software/hurd/hurd/translator/writing/example.html
* http://www.gnu.org/software/hurd/hurd/translator/examples.html
* http://www.gnu.org/software/hurd/hurd/documentation/translators.html
* http://www.gnu.org/software/hurd/hurd/debugging/translator.html
* http://www.debian.org/ports/hurd/hurd-install
* http://www.debian.org/ports/hurd/hurd-cd
* http://web.walfield.org/pub/people/neal/papers/hurd-installation-guide/english/hurd-install-guide.html
* http://uwhug.org.uk/index.pl?Hurd_Installation_Guide
* http://www.l4hq.org/
* Anatomy of the Linux kernel, M.Tim Jones
* http://en.wikipedia.org/wiki/Microkernel
* http://en.wikipedia.org/wiki/GNU_Hurd
* http://en.wikipedia.org/wiki/L4_microkernel_family
* http://en.wikipedia.org/wiki/Mach_(kernel)
* http://oreilly.com/catalog/opensources/book/appa.html