Systemy operacyjne oparte na mikrojądrze

Krzysztof Kryniecki
Konrad Kurdej
Karol Strzelecki

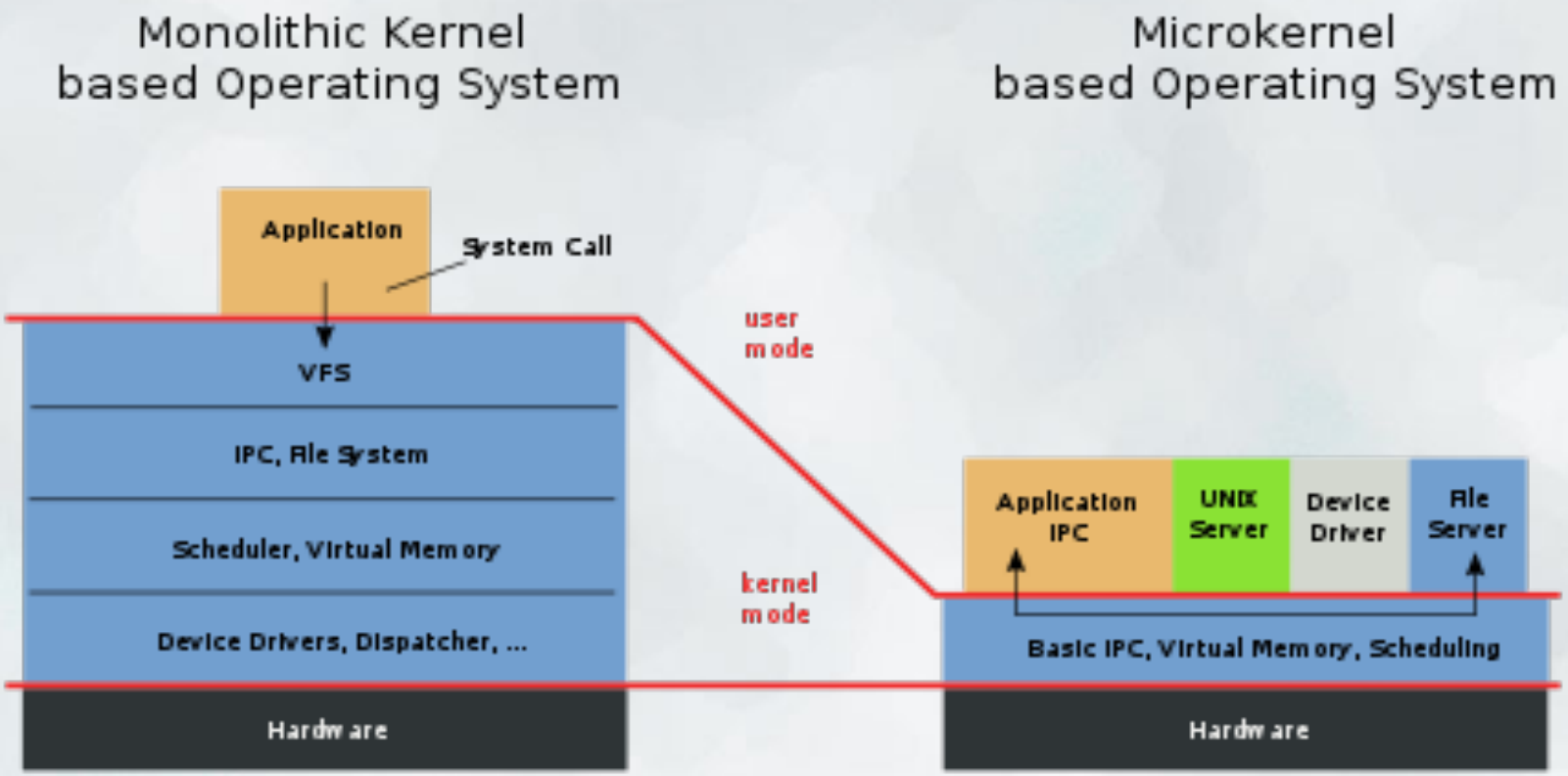
Czemu powstały mikrojądra?

Jak wiadomo jądro jest niezbędną a zarazem najważniejszą częścią systemu operacyjnego.
We wcześniejszych systemach jądra były raczezj małe, ponieważ pamieć komputerów była też ograniczona. Z biegiem czasu gdy możliwości sprzętowe komputerów zaczęły rosnąć liczba urządzeń i czynności jakie jądro musiało obsługiwać też musiała wzrosnąć. Wzrost ten był kontynuowany przez parę dekad, co w rezultacie przyczyniło się do tego, że kod jądra stał się olbrzym,i czyli wykazywał podatność na różne błędy i sprawiał trudności w jego rozbudowywaniu (w wypadku dokonania jakiejś zmiany niezbędna jest kompilacja całego jądra).
Ponadto niektórzy programiści chcieliby mieć większą swobodę podczas pisania programów, móc decydować o hardwerowych abstrakcjach .
Istnieje cała rodzina takich systemów są to exojądra. Ogólnie prefix przed nazwą kernel „pico”, „nano”, „micro” oznacza nie tylko rozmiar jądra ale i poziom abstrakcyjności spraw hardwerowych.

Czemu powstały mikrojądra?

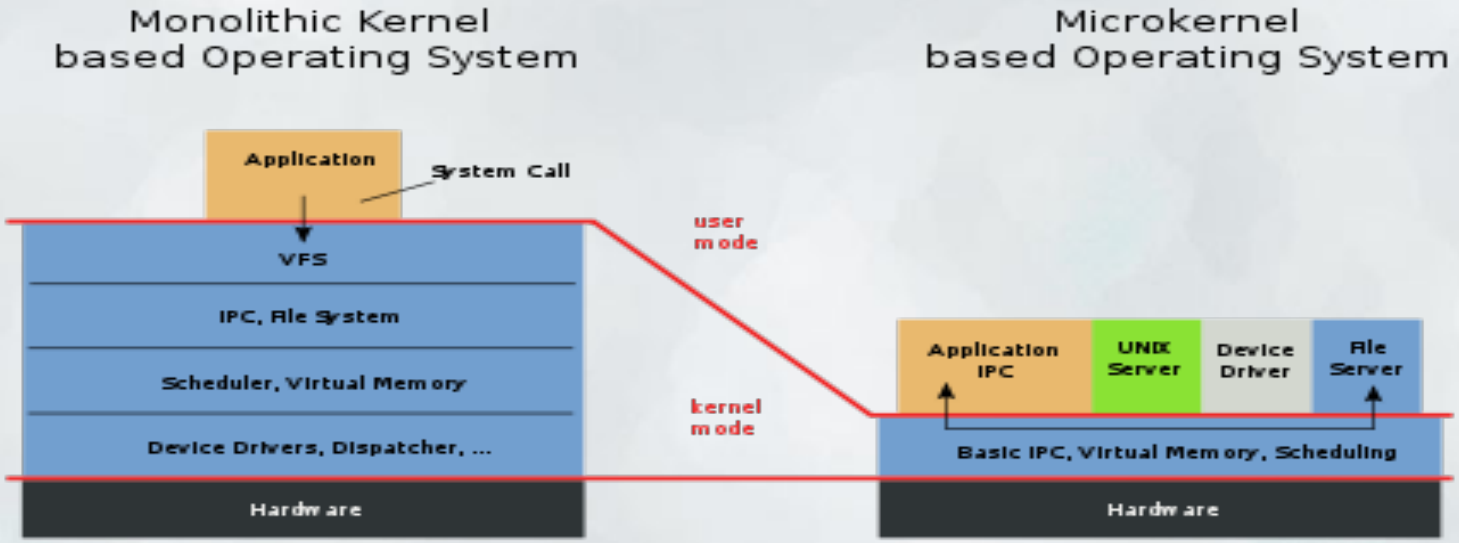
- kod jądra był olbrzymi
- podatność na różne błędy
- trudność w rozbudowywaniu

Koncepcja systemu



zmniejszono jądro do mikrojądra
zmodularyzowano jego strukturę, każda część ma konkretną funkcjonalność.

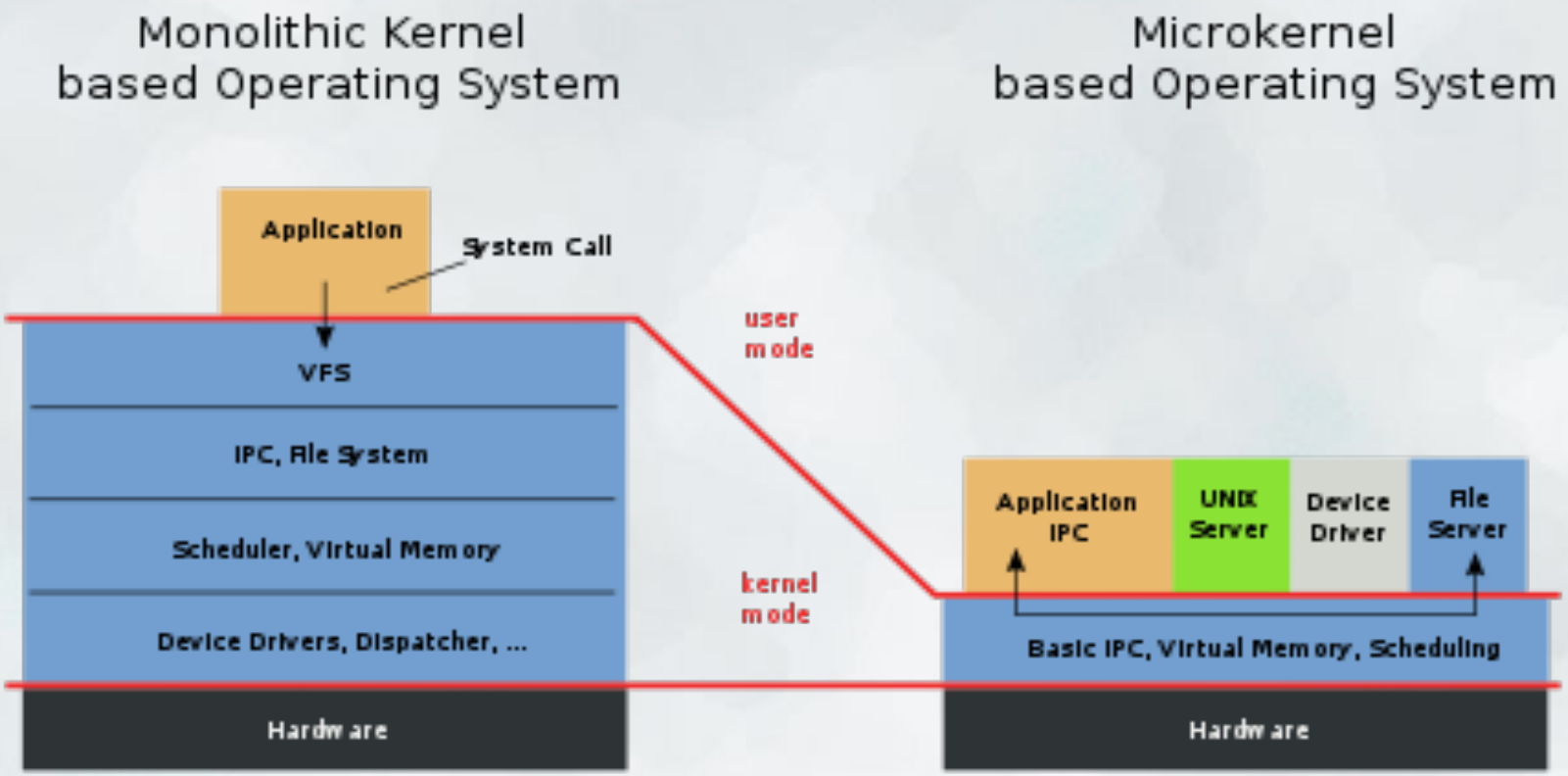
Koncepcja systemu



Mikrojądro zapewnia jedynie niezbędne funkcje takie jak:

- zarządzanie wątkami
- komunikacja między procesowa
- obsługa przerwań
- mechanizm do uruchamiania i kończenia procesów.
- Tylko jego zadania są wykonywane w trybie jądra.

Koncepcja systemu



Standarowe funkcjonalności systemu operacyjnego mają postać osobnych procesów (serwery) działających w trybie użytkownika z wyodrębnioną prywatną przestrzenią adresową chronioną przez MMU

L4Ka

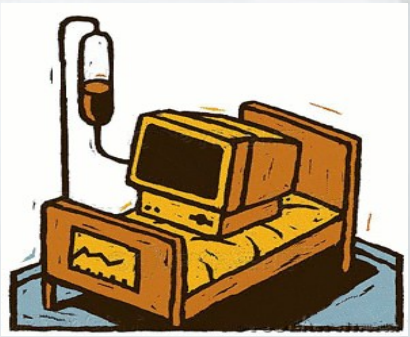
Niektóre mikrojądra są bardziej mikrojądrami niż inne. Dla przykładu, niektóre implementują jednostkę stronicowania w przestrzeni użytkownika ale podstawowe abstrakcje wirtualnej pamięci w jądrze np. Mach. Natomiast inne bardziej ekstremalne implementują większość wirtualnej pamięci w przestrzeni użytkownika. A jeszcze inne te mniej ekstremalne wiele serwerów uruchamia we własnej przestrzeni adresowej ale w trybie jądra np. Chorus.

(IPC)Komunikacja miedzy procesowa (ogólnie)

- może być synchroniczna lub asynchroniczna. Asynchroniczna IPC podobna do komunikacji sieciowej, wymaga od jądra buforowania i kolejkowania wiadomości
- za pośrednictwem wiadomości
- poprzez pamięć dzieloną

Uważa się, że komunikacja między procesowa jest wąskim gardłem całej koncepcji systemów z mikrojądrem pod względem wydajności.

Techniki przyspieszania komunikacji

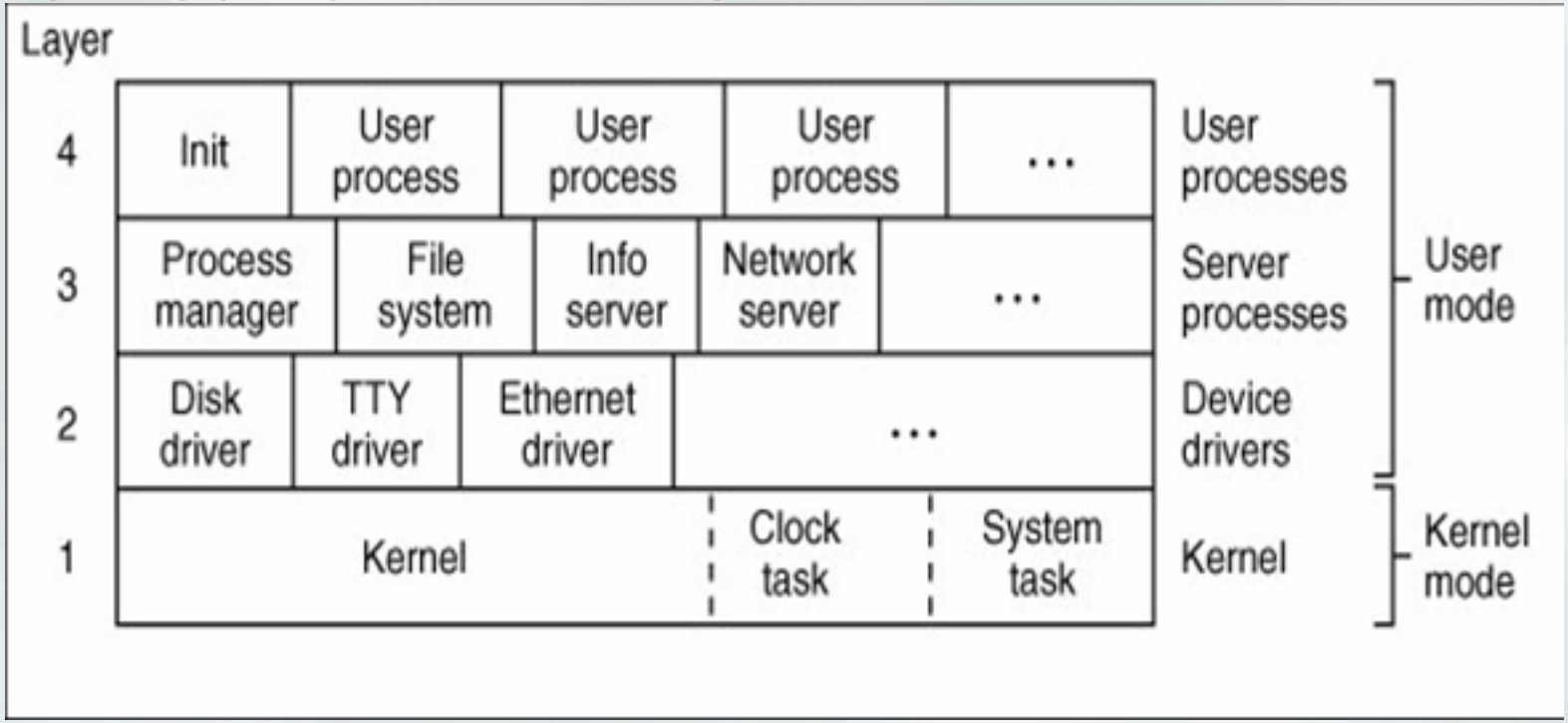


- **Przesyłanie krótkich wiadomości:** Przez przesłanie ich w rejestrach można otrzymać poprawę wydajości.
- **Kopiowanie Dużych danych wiadomości:** Wiele mikrojąder stosuje podwójne kopiowanie, jest to nadmierne gdy buforowanie nie jest wymagane(chybienia w TLB). W L4 nadawca udostępnia chwilowo obszar wiadomości odbiorcom.
- **Leniwe szeregowanie:** konwencjonalnie IPC wymaga uaktualnienia szeregowania wątków w kolejkach. Leniwe szeregowanie polega na ustawieniu flagi w kontrolnych blokach wątku i przeszukaniu kolejek podczas czasu zapytania w celu znalezienia wątków, które powinny być umieszczone w innych kolejkach.W większości zwiększa wydajność IPC o około 25%

Historia Minixa

- Powstał w 1987 roku na platformę x86. Późniejsze wersje Minixa działały również na platformach opartych o procesor Motorola 6800 (Apple, Macintosh, Amiga, Atari ST).
- 1987 : MINIX 1 (Andrew S. Tanenbaum), późniejsze wersje Minixa działały również na platformach opartych o Motorola 6800 (Apple, Macintosh, Amiga, Atari ST).
- Marzec 1994 : Linux 1.
- 24 października 2005 Minix 3.

Struktura Minixa



Minix ma strukturę warstwową, składa się z 4 warstw, każda ma konkretną funkcjonalność

warstwy 2 i 3: mają przywilej do robienia wywołań
warstwa 4: nie ma specjalnych przywilejów.

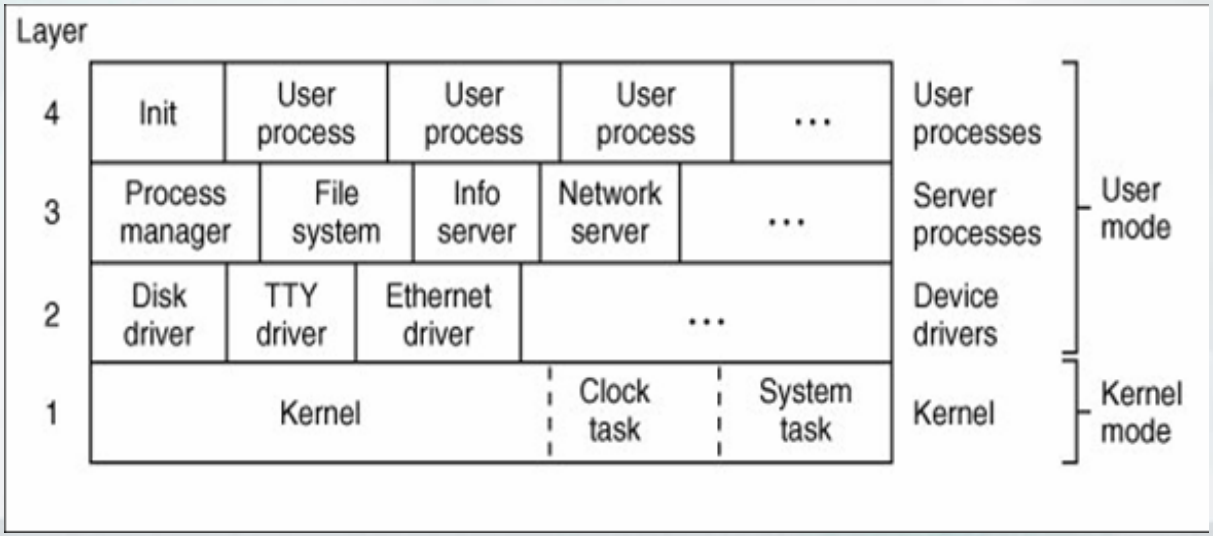
W takim ustawieniu wyższa warstwa ma prawo odwołać się do funkcji udostępnianych przez niższe warstwy.

Warstwa 1: Kernel

Najniższy poziom usług:

- obsługa wątków
- obsługa przerwań
- zapisywanie i przechowywanie rejestrów
- zarządzanie przestrzenią adresową
- szeregowanie procesów
- komunikacja między procesami
-

Znajduje się w niej sterownik (jest to wyjątek bo dla sterowników jest osobna warstwa - 2) zegarowy by ułatwić szeregowanie procesów.



System Task (Sys)

jest interfejsem jądra wobrec wszystkich procesów(serwerów) i sterowników w trybie użytkownika, które wymagają niskopoziomowych operacji w trybie jądra. Wszyskie możliwe przerwania systemowe są zamieniane na wiadomości, które są wysyłane do Sys-a. Obsługuje on je - wysyła odpowiedź jeśli nadawca miał prawo je nadać. Sys nigdy nie wychodzi z inicjatywą tzn. jest zawsze zablokowany czekając na nowe zadanie.
Wywołania systemowe mogą być pogrupowane w różne kategorie : zarządzanie procesami, zarządzanie pamięcią, kopiowanie danych pomiędzy procesami, kopiowanie danych pomiędzy procesami, urządzenia wejścia wyjścia, zarządzanie przerwaniami, dostęp do struktur jądra, usługi zegarowe.

Warstwa 2: sterowniki urządzeń

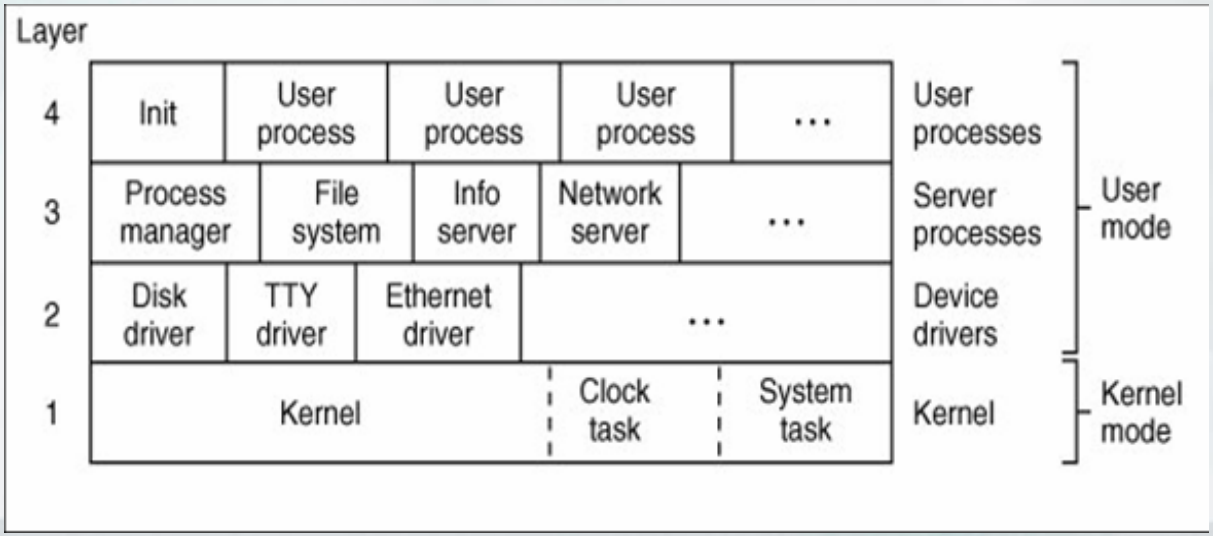
później

Warstwa 3: Serwery procesów

- Każdy serwer udostępnia usługi które mogą zostać użyte przez wszystkie procesy w warstwie 4.

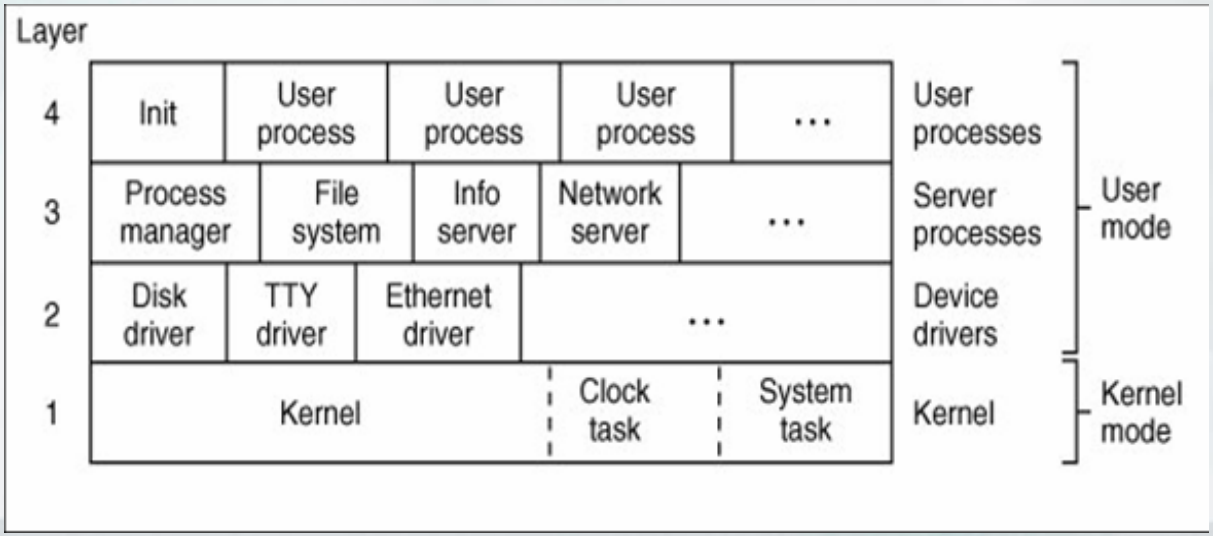
- Serwery Menadżer procesów (PM) oraz System plików(FS)są niezbędne

Przykładowe serwery : System plików, serwer reinkarnacji, Menadżer procesów, Serwer sieci.



Warstwa 4: Procesy użytkownika

Procesy działają w oparciu o usługi udostępniane przez niższe warstwy.
Procesy warstwy 4 mają najmniejsze uprawnienia.



Komunikacja międzyprocesowa.

Trzy metody do wysyłania i odbierania wiadomości:
− send(dest, &message);
− receive(source, &message);
− sendrec(src_dst, &message);

Komunikacja jest raczej synchroniczna rozmiar wiadomości jest ściśle określony.

metoda randek(randezvous principle).

Wiadomości nigdy nie są buforowane w jądrze, ale zawsze kopiowane od wysyłającego do odbiorcy.

Komunikacja międzyprocesowa. c.d

Komunikacja Mikrojądra z procesami użytkownika polega na tym, że Mikrojądro przechwytuje przerwanie i zamienia je na wiadomość, wybiera proces do uruchomienia, ładuje jego odpowiednie rejestry i pola MMU i obsługuje transport określonej wcześniej długości wiadomości do procesu używając metody randek.

Czemu raczej synchroniczna?

Problem polega na tym, że gdy proces w trybie użytkownika wyśle wiadomość a później nie będzie czekał na potwierdzenie to odbiorca jego komunikatu (w pewnych przypadkach może to być też jądro)
zostanie zablokowany gdy zechce wysłać odpowiedź

Roszerzanie się części systemu w trybie użytkownika wymaga ochrony procesów przed niespodziewanymi wiadomościami od innych procesów

By zapobiec takiemu przypadkowi wprowadzono 3 dodatki:

1. Cel komunikacji między procesowej (IPC) jest ograniczony.
2. W większości wypadków wymusza się randkę.
3. Nieblokująca asynchroniczność.

Pierwszy punkt: Cel komunikacji między procesowej (IPC) jest ograniczony

Każdemu procesowi została dana bitmapa mówiąca do którego procesu może on wysyłać wiadomości.
Do dozwolonych odbiorców można zaliczyć sterowniki, serwery, jądro i użytkowników ale wszyscy użytkownicy zostali potraktowani tak samo przez proces – czyli proces może wysyłać do wszystkich użytkowników lub nie może wysyłać do żadnego.

Drugi punkt: W większości wypadków wymusza się randkę.

Połączenie wysyłania i odbierania -w jednym wywołaniu

Odbiorca będzie blokowany do czasu aż cała operacja się nie zakończy. Wprowadzenie takiej metody eliminuje przypadek gdy sterownik wysyła do jądra wiadomość i nie czeka na odbiór potwierdzenia zawiszając jądro.

Trzeci punkt: Nieblokująca asynchroniczność.

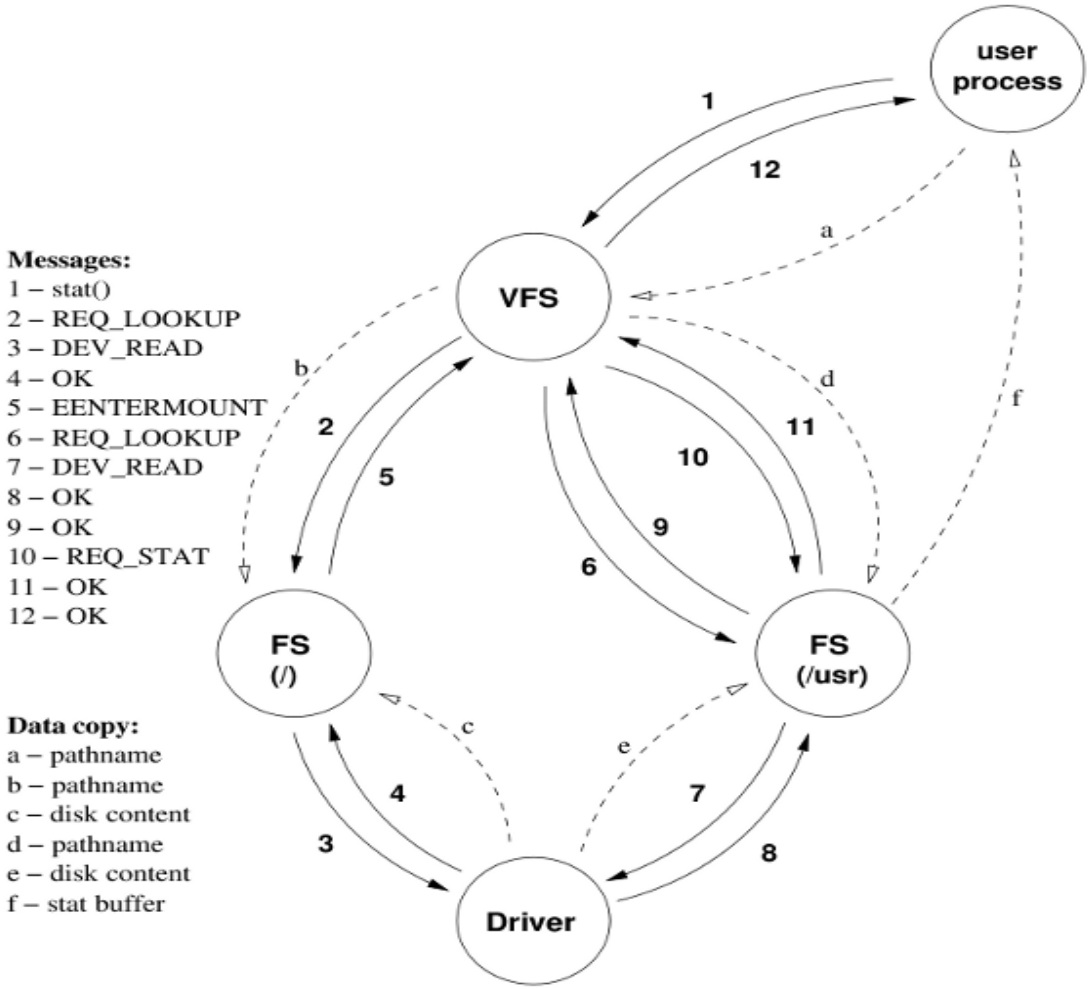
Nieblokująca opcja została dodana dla tych paru przypadków gdzie serwer lub sterownik nie może pozwolić sobie na blokowanie gdy odbiorca jest zajęty.

np. sterownik, który nie może dokładnie dostarczyć ("z rąk do rąk") żądanych danych do serwera plików może wysłać wstępną odpowiedź i dzięki temu później obudzić serwer plików używając nieblokujących powiadomień.

Warstwa 2: sterowniki urządzeń

- podejście obiektowe
- sterownik urządzenia to osobny proces
- błąd programisty piszącego sterownik nie powoduje 'wywalenia' całego systemu
- do gry wkracza serwer reinkarnacji (o którym za moment), który ocala nam system

Komunikacja warstw



Przepływ komunikatów

weźmy read'a:

- program woła POSIXową read()
- do Serwera Plików wysyłana jest wiadomość z odpowiednią prośbą
- SP sprawdza, czy może ma treść w cache'u
- jeśli nie -> wysyła wiadmość do sterownika dysku aby ten zapisał dane do ów cache'a
- po takim przygotowaniu SP prosi jądro o skopiowanie danych do przestrzeni pamięci użytkownika

Bezpieczeństwo

- jądro Minixa ~ 4000 linijek kodu
- jądro Linuxa ~ 2 500 000
- jądro WinXP ~ ponoć 5 000 000

średnio 10 bugów na 1000 linii, więc Minix ma ich mniej niż 50, a WinXP 50 000

Bezpieczeństwo cd.

- komunikaty mają stałą wielkość
- sterowniki działają w przestrzeni użytkownika, więc nie mogą korzystać np. z I/O czy blokowania przerwań
- można tak skonfigurować system, aby sterowniki korzystały tylko z określonych wywołań kernela

Bezpieczeństwo: Reincarnation server

Czy zabugowany sterownik drukarki powinien powodować krach całego systemu?

```
for(;;);
```


*** STOP: 0x0000001E (0xC0000005, 0xF24A447A, 0x00000001, 0x00000000)
KMODE_EXCEPTION_NOT_HANDLED
*** Address F24A447A base at F24A0000, DateStamp 35825ef8d - wdmaud.sys
If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:
Check to be sure you have adequate disk space. If a driver is identified in the Stop message, disable the driver or check with the manufacturer for driver updates. Try changing video adapters.
Check with your hardware vendor for any BIOS updates. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.
Refer to your Getting Started manual for more information on troubleshooting Stop errors.
Kernel Debugger Using: COM2 (Port 0x2f8, Baud Rate 19200)
Beginning dump of physical memory
Physical memory dump complete. Contact your system administrator or technical support group.

Reincarnation server (RS)

- 'rozwalonym' sterownikiem zajmuje się RS
- odpala procedurę obsługi padniętego sterownika (może przeładować sterownik, zapisać obraz pamięci, czy wysłać wiadomość do administratora)
- co kwant czasu pinguje procesy aby sprawdzić, czy np. nie działają w nieskończonej pętli

Oczywiście przy podejmowaniu tej akcji system działa normalnie!

Nie wyłapuje jednak jeszcze wszystkich błędów...

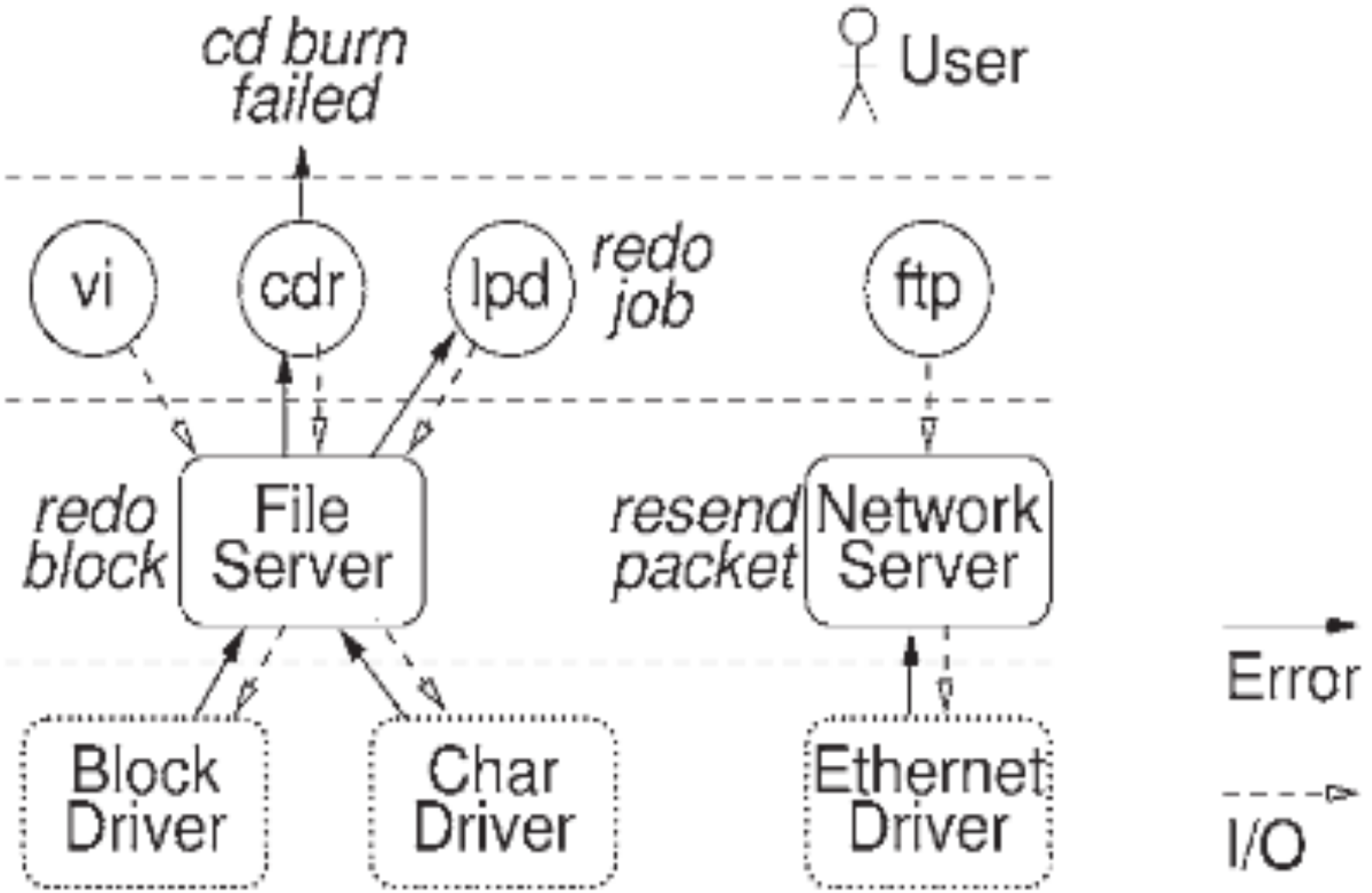
Data Store - pomocnik

mała, prosta baza przechowująca napisy

sterownik RAM może przechować tam informacje o zadeklarowanej pamięci

pomaga przy przywracaniu systemu po niespodziewanym retarcie

Scenariusz ratowania systemu



Ratunek a wydajność

przy symulowanym błędzie krytycznym sterownika z częstotliwością 1 raz / 4 sekundy wydajność procesora spada zaledwie o 8%!

Czy Minix jest samotny?

Inne przykładowe systemy oparte na mikrojądrze

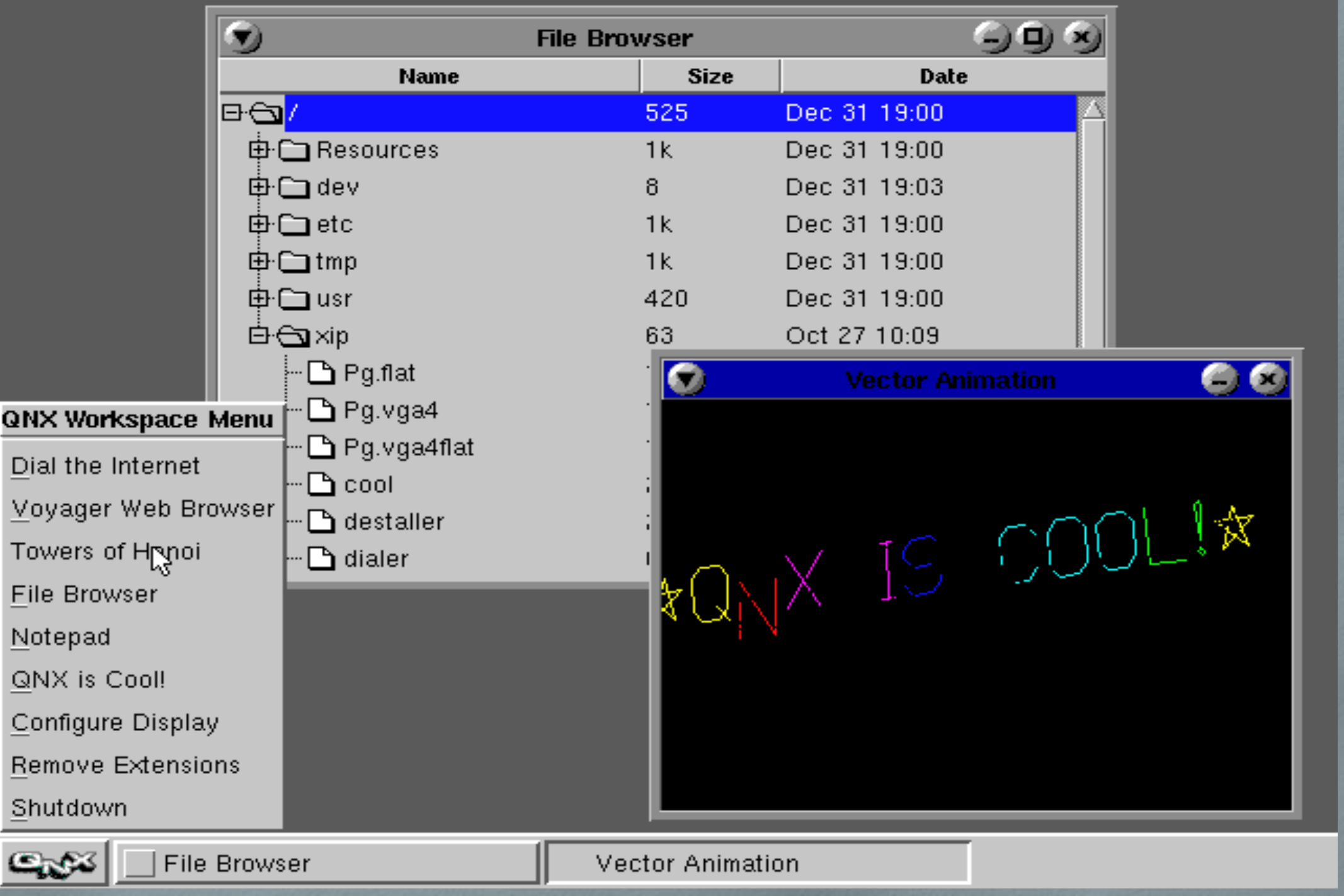
- QNX
- Integrity
- PikeOS
- Symbian
- L4Linux
- Singularity
- K42
- Mac OS X
- HURD
- Coyotos

- system operacyjny czasu rzeczywistego*

- *Stworzony przez QNX Software Systems Ltd*
- *Komercyjny*
- Od 12 września 2007 dostępny jest kod źródłowy jądra systemu (for non-commercial users)
- Przykładowi użytkownicy (wersji dostosowanych do swoich potrzeb): Cisco, Delphi, General Electric, Siemens, Thales
- **IOS XR** - bazuje na mikrojądrze QNX. System wykorzystywany w najnowszych i najbardziej zaawansowanych urządzeniach Cisco

* **Real-Time Operating System** (**RTOS)** - nastawione na prace aplikacji real-time. Np precyzyjne w czasie sterowanie rakietami

QNX Floppy Demo



Podstawowe cechy:

- Zaliczany do klasy Unix
- Dostępny na wiele platform (x86 family, MIPS, PowerPC, SH-4 and the closely related family of ARM, StrongARM and XScale CPUs)
- Zwykle działa jako system wbudowany (w urządzeniach)
- W jądrze znajduje się tylko: zarządzanie CPU,komunikacja pomiędzy procesami, przekierowanie przerwań, zegary
- reszta działa jako procesy użytkownika (proces **proc** tworzący procesy, zarządzanie pamięcią)
- komunikacja w stylu: komunikat do innego procesu i oczekiwanie na odpowiedź
- specjalny bootloader - ładuje nie tylko jądro, ale także programy użytkownika

symbian

- Chyba najpopularniejszy system operacyjny zbudowany na mikrojądrze (65% rynku urządzeń mobilnych w lipcu 2008, po zaprezentowaniu iPhona zmalało)...
- ale nie jest to do końca mikrojądro, gdyż zawiera w sobie sterowniki urządze
- Wywłaszcza procesy i stosuje ochronę pamięci
- Projektuje się (SO i aplikacje) według wzorca *Model-View-Controller*

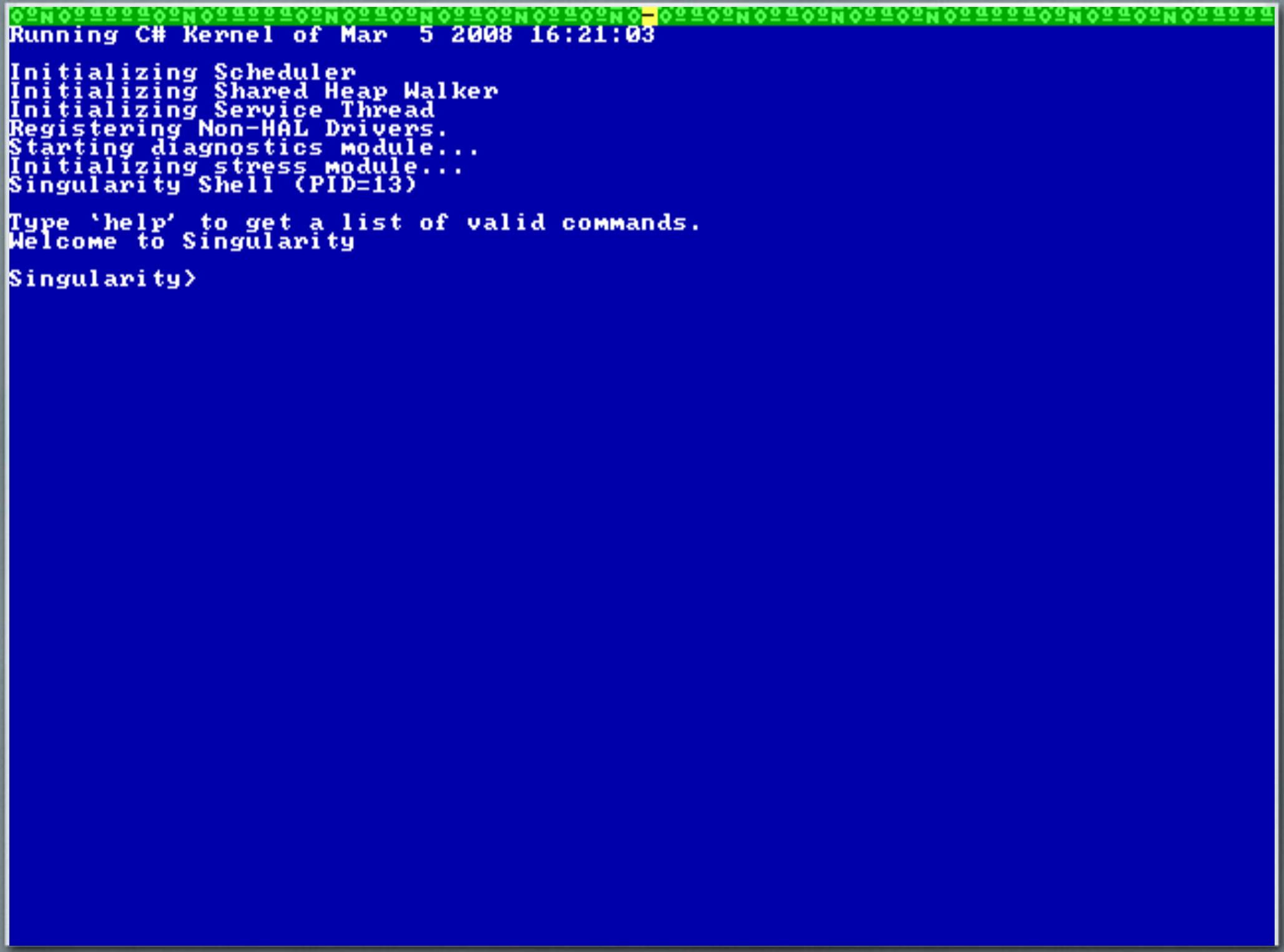
Więcej opowie inna grupa przy okazji tematu **"Systemy operacyjne na urządzenia mobilne"**

symbian


E-Plus
Bluetooth
Galerie
System
Optionen
Zurück
orange
Messaging
Contacts
Menu
Gallery
Camera
Internet
Games
Office
E-Plus
Sa 13.10.2007
Bluetooth
Galerie
Menü
Optionen
Schließen
orange
Menu
Contacts
Gallery
Tools
Internet
Games
Office
Multimedia
Applications
Dictionary

Singularity

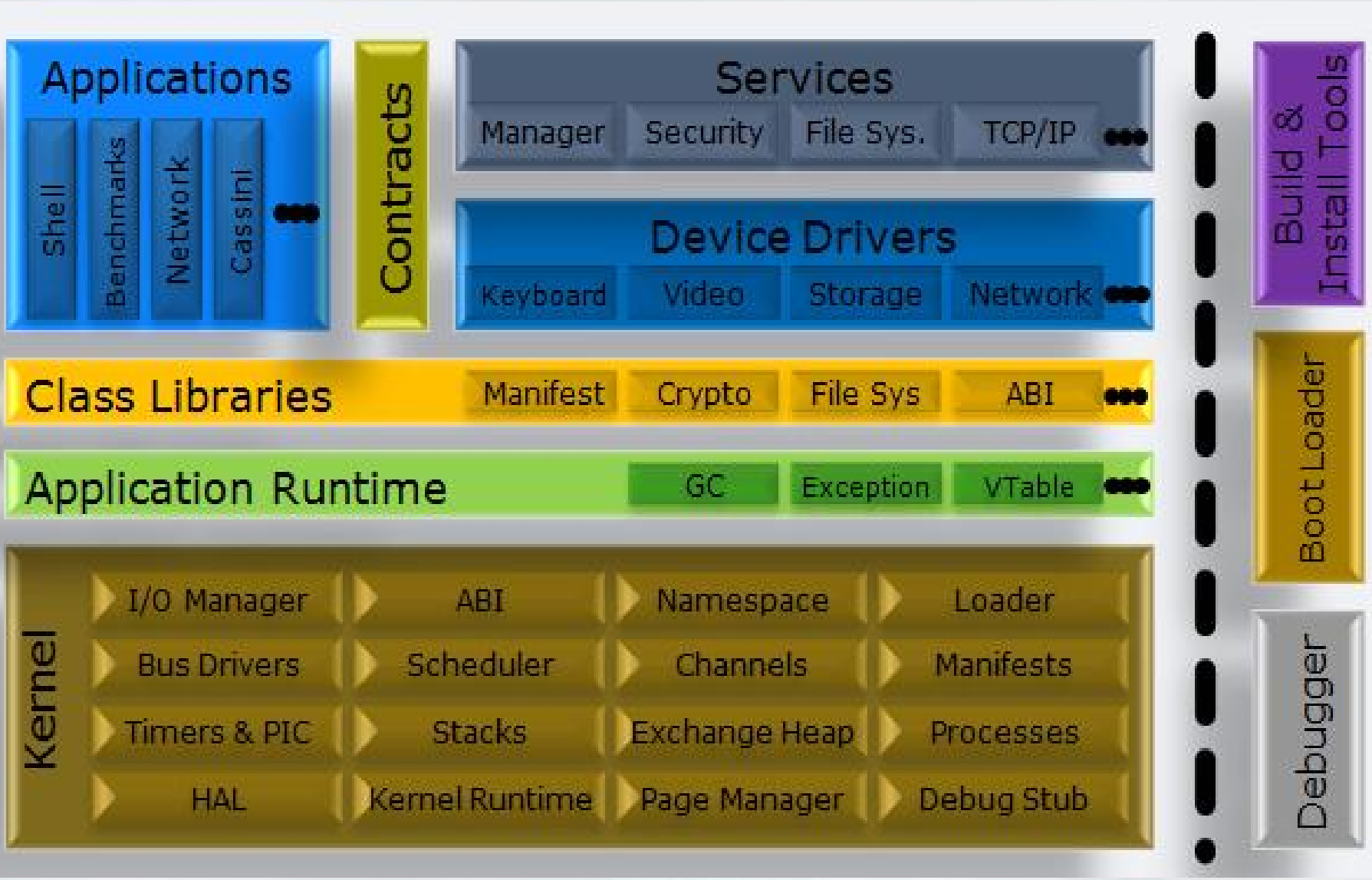
- Eksperymentalny system operacyjny stworzony w Microsoft Research
- Początek prac w 2003. Wersja 1.0 w marcu 2007, zaś 2.0 14 listpada 2008
- W większości napisany w C#
- Dostępne Research Development Kit wraz z źródłami systemu (niekomercyjne użycie w celach akademickich)
- Programy kompilowane są do bezpiecznego kodu zarządzanego (analogia do wirtualnej maszyny Javy)
- Prawdopodobnie jest to poligon doświadczalny dla przyszłych systemów (komercyjny następca: Midori)


Running C# Kernel of Mar 5 2008 16:21:03
Initializing Scheduler
Initializing Shared Heap Walker
Initializing Service Thread
Registering Non-HAL Drivers.
Starting diagnostics module...
Initializing stress module...
Singularity Shell (PID=13)
Type 'help' to get a list of valid commands.
Welcome to Singularity
Singularity>

Singularity

- Brak sprzętowej ochrony zasobów jak np. odzielne przestrzenie adresowe, poziomy ochrony procesora
- ... zastąpione są one programowymi rozwiązaniami np. "Software-Isolated Processes" (SIP). Programy mogą dostawać różne części pamięci. Możliwe to jest dzięki kompilacji programów do bezpiecznego kodu zarządzanego (safecode)
- Operacje takie jak przełączanie zadań, czy wywołanie systemowe, dzięki brakowi konieczności zmiany przestrzeni adresowej i trybu ochrony procesora wykonywane są znacznie szybciej
- wspólny garbage collector dla systemu i aplikacji

Singularity - Architektura



Porównanie wydajności z jądrem monolitycznym

Na początku 2007 roku odbył się wykład zatytułowany "Andrew Tanenbaum on creating reliable systems". Jednym z głównych punktów było udowodnienie, że Minix z racji swojego mikrojądrza jest wolniejszy tylko o ok 5-10% od systemu opartego o jądro monolityczne. Niestety nie było wiadomo jak zostały przeprowadzone testy co rzuciło cień podejrzeń, że porównywanym systemem był ... Minix tyle że z monolitycznym jądrem.
Nie trwało długo nim ktoś postanowił to sprawdzić w praktyce ...

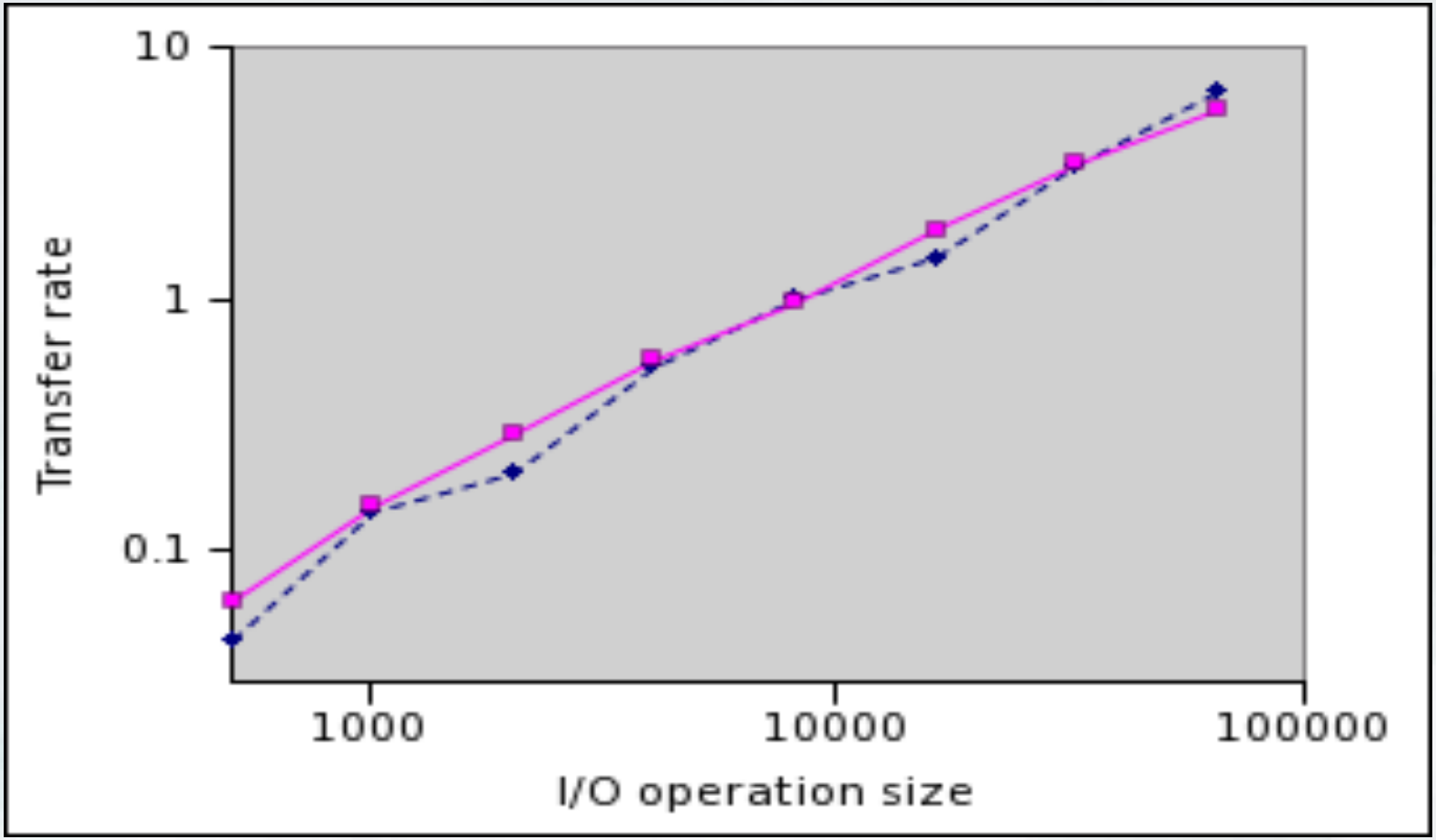
Wyniki opublikowano w serwisie LWN.net

Platforma testowa

Porównano: Minix 3.1.2a i Debian Etch z jądrem 2.6.17
Sprzęt: AMD Athlon 1700 z dyskiem ATA
Dysk podzielono na 3 partycje: systemową, swap (wykożystywaną tylko przez linuxa) i testową dla destrukcyjnych testów. Były mniejsze niż 4GB (ograniczenie Minixa).

2 testy:
IOtest - 4 (ograniczone ze względu na słabe odzyskiwanie zasobów przez Minixa) wątki w losowych miejscach zapisują lub odczytują dane
UnixBench - program z dostępnym źródłem przeprowadzający różnego rodzaju testy

IOtest

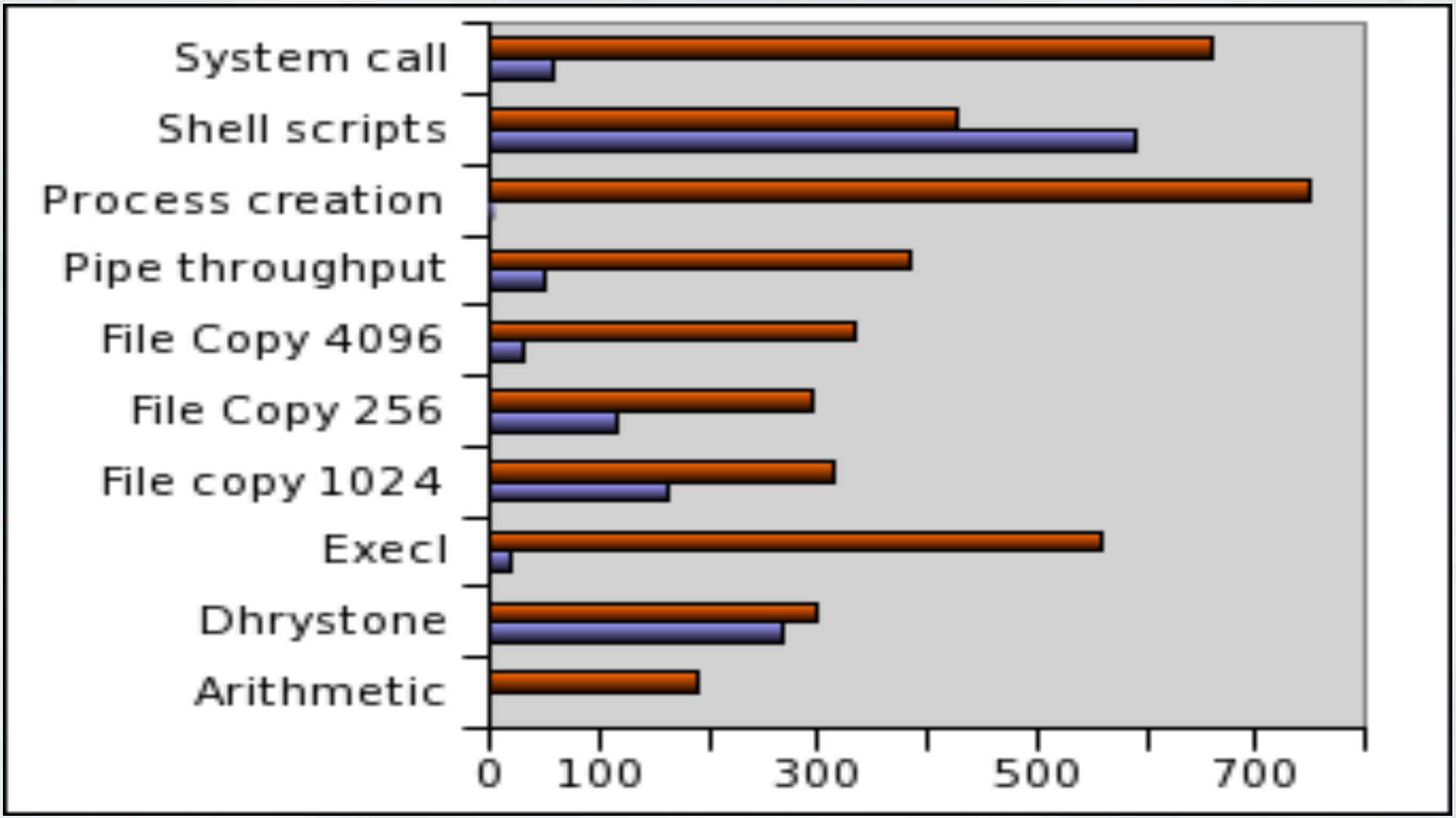


różowa linia - Linux
niebieska, przerywana - Minix

IOtest

Wyniki bardzo zbliżone. Zdominowane głównie przez wydajnośc operacji seek sterownika dysku. Scheduler I/O nie miał zbyt szerokiego pola do popisu ze względu na bardzo mały rozmiar dysku (4 GB).

UnixBench



czerwony - Linux
niebieski - Minix

więcej = lepiej

UnixBench

Przewaga Linuxa jest ogromna. Uzyskał on ogólnie 389,4 punktów zaś Minix 48,1 (8 krotna różnica!). Jedyną dziedziną w której Minix był lepszy jest szybkość działania skryptów konsoli. Patrząc jednak na możliwości konsoli Minixa i basha to przypominało to wyścig przyczepy campingowej i malucha w parkowaniu.
Narzut wywoływania funkcji systemowych różni się 10 krotnie. Tworzenie nowego procesu na Minixie odbywało się 140 razy wolniej!

Wydajność - podsumowanie

Można twierdzić, że Minix jest nowym systemem, nie dopracowanym co pozostawia wiele okazji do zwiększenia wydajności. Zapewne jest to po części prawdziwe. Z drugiej strony można było śmiało twierdzić że Linux ma prawo być wolniejszy ze względu na ilość architektur na jakich działa oraz mnogość sprzętu jaki obsługuje.

Dr. Tanenbaum uważa ("Tanenbaum-Torvalds Debate: Part II"), że użytkownicy są w stanie poświęcić część wydajności w zamian za niezawodność. Myślę, że w 80% przypadów jest to prawda. Niestety, na dzień dzisiejszy ta część wydajności nie jest rzędu 5-10%, a większej niezawodności nie udowodniono...

Zalety i wady mikrojądra

Za:

- niezawodność
- bezpieczeństwo
- programowanie SO przypomina OOP (mniejsze problemy przy wykorzystywaniu współdzielonych struktur danych => mniejsze problemy synchronizacyjne; łatwiej testować; można zrobić diagram UML-o podobny:))
- skalowalność

Przeciw:

- mniejsza wydajność
- przy programowaniu może być potrzebna znajomość algorytmów równoległych

Podsumowanie: Gdzie to się naprawde sprawdza

- Tam gdzie krytyczną wage ma niezawodność i bezpieczeństwo, a rzeczą drugorzędną jest wydajność (np oprogramowanie do obsługi samolotu, sterowanie rakietami). Systemy oparte o mikrojądro wykorzystuje np wojsko, takie firmy jak Cisco.
- Wszystkie systemy wbudowane w których zmiana kontekstu procesora jest bardzo szybka
- Większość urządzeń mobilnych bazuje na systemie operacyjnym opartym o mikrojądro

Bibliografia

http://en.wikipedia.org/wiki/Microkernel
http://www.cs.vu.nl/~ast/reliable-os/ "Tanenbaum-Torvalds Debate: Part II"
http://en.wikipedia.org/wiki/MINIX_3
http://en.wikipedia.org/wiki/Qnx
http://en.wikibooks.org/wiki/Minix_3
http://en.wikipedia.org/wiki/Singularity_(operating_system)
http://en.wikipedia.org/wiki/GNU_Hurd
http://lwn.net/Articles/220255/ "Comparing Linux and Minix"
http://pl.wikipedia.org/wiki/System_operacyjny_czasu_rzeczywistego
http://research.microsoft.com/os/singularity/
http://en.wikipedia.org/wiki/Managed_code
http://pl.wikipedia.org/wiki/Microsoft_Midori
http://www.cs.cornell.edu/home/ulfar/ukernel/ukernel.html
http://www.minix3.org/doc/EDCC-2006.pdf
http://www.minix3.org/doc/gerofi_thesis.pdf
http://www.minix3.org/doc/OSR-2006.pd