

# 2x DVD

DVD1: Debian 3.0r2 DVD - kaitowa dystrybucja Linuksa, oficjalna stabilna wersja  
DVD2: Gentoo 2004.1 DVD - po raz pierwszy i tylko u nas instalacja z pakietów binarnych lub źródłowych \* NVIDIA 1.0-6106 i ATI 3.9.0 \* SuperKaramba 0.34

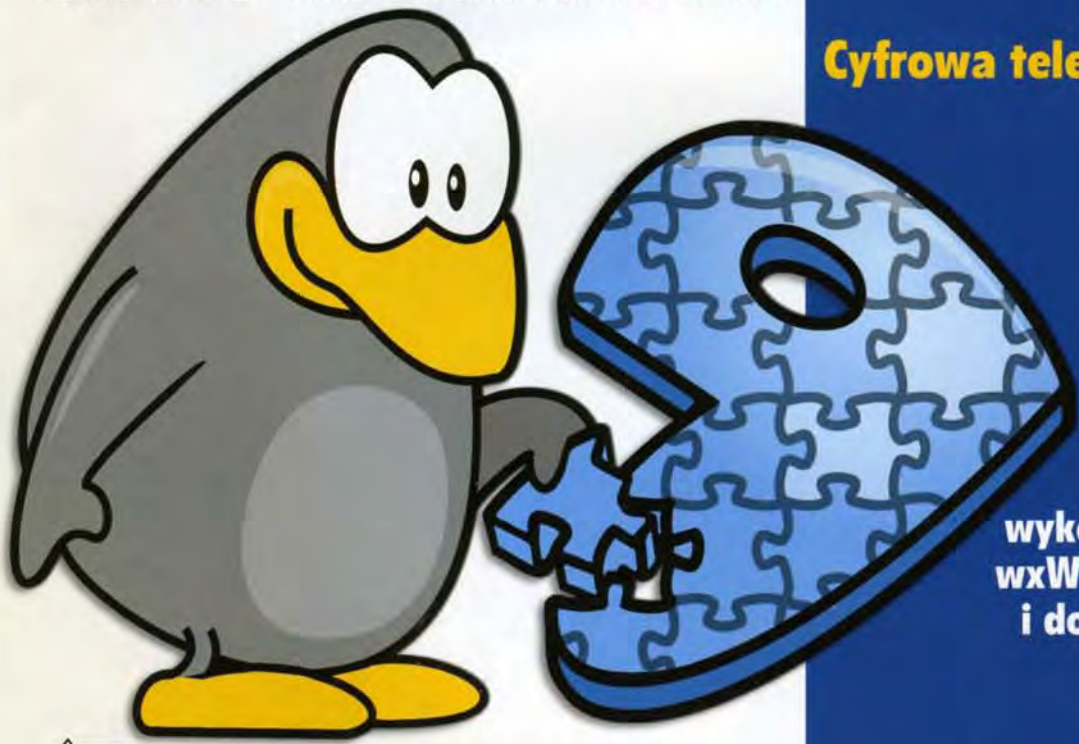
LINUX+ DVD

# LINUX+ DVD

NAJWIĘKSZY W POLSCE MAGAZYN O LINUKSIE

Nr 06 (06) Sierpień 2004 Cena 27,90 zł Stawka VAT 0% INDEX 384267 Nakład 8 000 egz.

# TUNING & KDE & GNOME biurko Twoich marzeń



LINUX+ Live DVD  
prezentacja programów z artykułów na żywo

## Skype for Linux 0.90

narzędzie do rozmów głosowych przez Internet



Tylko u nas

## Cyfrowa telewizja satelitarna pod Linuksem

uruchamianie kart DVB-S

## Piszemy własny katalog filmów

wykorzystanie biblioteki wxWidgets, pakietu Xine i dodatku do eDonkeya

## Wywiad z założycielem Blender Foundation

Ton Roosendaal odpowiada na nasze pytania



### DYSTRYBUCJE LINUKSA

**Linux From Scratch w praktyce**  
Chciałbyś zrobić swoją dystrybucję Linuksa?  
Pokażemy Ci, jak to wykonać!



Grzegorz Prokopski

## Debian GNU/nie Linux? Hurd!

Dzieje Debiana, jak i Linuksa, są mocno powiązane z *Free Software Foundation* i *Richardem Stallmanem*. Kiedy w 1983 roku rozpoczął on *Projekt GNU*, chciał stworzyć system zgodny ze specyfikacją Uniksa, składający się wyłącznie z *Wolnego Oprogramowania*. Dość szybko powstały najbardziej potrzebne narzędzia – oczywiście na licencji GNU GPL, jednak wciąż brakowało jednego, ale niezwykle ważnego elementu – *jądra systemu*. Rozważano wiele możliwości, próbowano namówić kilku właścicieli mikrojąder na ich przekazanie projektowi *GNU* – wszystko po to, aby uniknąć pisania własnego jądra od zera. W końcu, w 1988 roku wybór padł na *mikrojądro Mach*. Niestety, dopiero w 1991 roku kod mikrojądra Mach został formalnie udostępniony na licencji pozwalającej na rozpowszechnianie go jako części *Systemu GNU*.

Należy pamiętać, że w tamtym czasie *Linux* dopiero raczkował, był i miał być bardzo ograniczony, był i miał pozostać nieprzenośny między architekturami. Wobec powyższego faktu rozpoczęto prace nad *jądrem Hurd*, opartym o mikrojądro Mach. Cały projekt od początku był technologicznie znacznie lepiej zaplanowany i bardziej obiecujący niż *Linux*, jednak okazało się, że prace nad *Hurdem* przedłużają się, natomiast *Linux* szybko stawał się dostępnym „już”, tak potrzebnym i oczekiwanym jądrem systemu dla oprogramowania z Projektu GNU. Sytuacja ta utrwaliła się i sam *Linux* – ulepszany przez lata – jest dziś najczęściej używanym jądrem dla Systemu GNU.

W 1998 roku wystartował projekt Debian GNU/Hurd. W tej chwili składa się on z około 1/3 pakietów znajdujących się w nadchodzącym wydaniu Debiana Woodiego. Deweloperzy przewidują, że pierwsze oficjalne wydanie nastąpi wraz z kolejnym Debianem – wersja 3.1 będzie dostępny zarówno z jądrem *Linux*, jak i *Hurd*. Nie ma jednak potrzeby czekać z wypróbowaniem Hurda aż do jego pierwszego wydania, gdyż są już dostępne (dość często uaktualniane) nieoficjalne obrazy CD [noCD].

## Siła zamysłu – architektura Hurda

Zanim zajmiemy się *Hurdem*, trzeba jeszcze przypomnieć, że jądro Linuksa jest jądrem *monolitycznym*. Oznacza to, że każdy podsystem jądra może bez żadnych przeszkód zmodyfikować dane należące do innego podsystemu. Powoduje to, że każdy błąd w jądrze powodujący omyłkowe zamazanie fragmentu pamięci, modyfikujący dane w nieprawidłowy sposób lub wywołujący jakąś wewnętrzną funkcję w „niewłaściwym” momencie, prawie natychmiast prowadzi do poważnego błędu całego jądra (*kernel panic*), a nie tylko podsystemu, który jest odpowiedzialny za nieprawidłowe zachowanie. Każdy taki błąd

w kodzie jądra Linuksa ma w większości przypadków katastrofalne skutki dla całego systemu. Z drugiej strony, bezpośredni dostęp do danych wszystkich podsystemów jądra z każdego fragmentu jego kodu daje możliwość szybkiego na nich operowania, a co za tym idzie, zwiększa prędkość całego systemu.

Konstrukcja ta ma podstawy historyczne – *Linus* konstruował swoje jądro jako małe i ograniczone, a także działające na sprzęcie klasy i386, gdzie liczył się każdy takt zegara. *Linux* nie był też nigdy projektowany, by działać na wielu procesorach. O ile taka architektura sprawdzała się przy niewielkiej ilości podsystemów, to przy ich rosnącej ilości, opieka nad kodem stawała się i z każdą nową linią kodu staje się coraz trudniejsza.

Natomiast *Hurd* posiada architekturę z tzw. *mikrojądrem*, ale zanim to wyjaśnię, wypada przybliżyć zamysły twórców. *Hurd*, inaczej niż *Linux*, nigdy nie był projektowany dla jakiejś konkretnej architektury. Z założenia i od samego początku miał on być przenośny, miał działać na wielu procesorach i miał wspierać wielowątkowość. *Projekt GNU* wybrał dla swojego jądra bardziej elegancką i przemyślaną konstrukcję, chociaż może nie od razu bardziej efektywną.

Mikrojądro Mach odpowiada jedynie za tworzenie i usuwanie procesów, zarządzanie kolejnością wykonywania zadań (*ang. scheduling*), zarządzanie pamięcią i obsługę przerwań. Wszystko inne, tak jak obsługa protokołów, systemy plików czy podstawowe funkcje komunikacji między procesami, jest obsługiwane przez *demony jądra*, ale działające *w przestrzeni użytkownika*, a nie jądra. Komunikacja pomiędzy jądrem systemu, a tymi „dodatkowymi” elementami, odbywa się jedynie poprzez dobrze udokumentowane, jasne i przejrzyste interfejsy, a nie tak, jak w monolitycznym jądrze, gdzie każdy podsystem może robić wszystko z każdym innym podsystemem. I nawet jeśli zdarzy się, że w jednym z podsystemów jest błąd, to w najgorszym wypadku spowoduje on jedynie zakończenie procesu obsługującego ten podsystem, a nie całego jądra.

## Obiektowość, wieloprocusowość i wielowątkowość jądra

Choć mówienie o jądrze napisanym w całości w czystym C, że jest *obiektowo zorientowane* może dziwić, to jednak w przypadku *Hurda* jest dość dobrze uzasadnione. To przecież właśnie demony jądra – tak jak obiekty – udostępniają interfejsy, za pomocą których można się z nimi komunikować i w ten sposób wpływać na ich stan (ich dane). Dzięki temu zmiana wewnętrznej reprezentacji danych czy sposobu obsługi żądań nie wymusza zmian w kodzie innych obiektów.

*Demony jądra*, nazywane w świecie Hurda *serwerami*, udostępniają pewien zestaw swoich usług aplikacjom użytkownika. W ten sposób serwery komunikują się z jądrem, między sobą, a także procesami użytkownika, poprzez dobrze zdefiniowane *interfejsy*.

Jądro Mach zarządza zadaniami w podobny sposób jak wszystkie jądra Uniksa, jednak w odróżnieniu od tradycyjnego rozumienia unixowego procesu, nie zakłada ono tylko jednego

wątku wykonania – wręcz przeciwnie. Mach od początku był projektowany, aby osiągać wysoką wydajność na maszynach z wieloma procesorami. Ponieważ jednak każdy z serwerów np. obsługujących stos TCP/IP jest tylko jednym procesem, więc aby działać efektywnie, musi być bardzo mocno wielowątkowy. W Hurdzie nie dziwią tysiące wątków i setki procesów. To właśnie dzięki nim ta architektura jest (szczególnie w maszynach wieloprocessorowych) niezwykle efektywna.

Wydaje się też, że Hurd znacznie lepiej realizuje w praktyce jedną z idei Uniksów – KISS (*ang. Keep It Simple Stupid!*), czyli w wolnym tłumaczeniu: „Niech to będzie proste – głupcze!”. Tak jak w Uniksie możemy znaleźć masę drobnych, dość prostych, ale wyspecjalizowanych i świetnie dopracowanych programów, tak na jądro Hurda składa się wiele *serwerów*, z których każdy realizuje tylko pewien ograniczony i ściśle zdefiniowany zakres zadań. Dzięki temu Hurd może być prostszy, mniejszy, a także – przez swoją uniwersalność – użyteczniejszy dla innych *serwerów*.

## Systemy plików, autoryzacja i bezpieczeństwo w Hurdzie

Hurd posiada dwie ważne cechy odróżniające go od wielu innych systemów typu Unix: posiada dobrze obmyślane i skonstruowane mechanizmy bezpieczeństwa i uwierzytelniania, a także jest bardzo mocno zorientowany na rozszerzanie jego możliwości przez użytkownika.

Jak to wygląda w praktyce? Łatwo jest zebrać przykłady złych, połowicznych rozwiązań, które powstały, aby ułatwić życie użytkownikowi: *bash* pozwala na dostęp do `/dev/tcp/HOST/PORT`, *GNOME* posiada swoją bibliotekę wirtualnego systemu plików, *KDE* swoją... I tego rodzaju funkcjonalność jest wielokrotnie implementowana przez wiele aplikacji. Dlaczego więc nie zamienić tych wszystkich wirtualnych systemów plików na standardowy, prawdziwy system? Dla Linuksa oznaczałoby to napisanie nowego modułu do jądra (co samo w sobie jest ogromnym zadaniem), wielomiesięcznego testowania go (debuggowanie jądra Linuksa jest dość trudne), aby wreszcie Linus mógł zawyrokować, czy zezwala na włączenie go do jądra.

Natomiast w Hurdzie każdy zwykły użytkownik może napisać, a co najmniej uruchomić program, który będzie np. *serverem* systemu plików. Upraszczając nieco sprawę, aby stworzyć swój system plików dla protokołu XYZ (*http, ftp, smb,...*), wystarczy wykorzystać zwykłą bibliotekę implementującą ten protokół i zaimplementować podstawowe funkcje systemu plików (jak wyświetlanie zawartości katalogu, odczyt i zapis pliku). Dostępne są nawet biblioteki (*libnetfs, libdiskfs, librivfs*), które bardzo ułatwiają stworzenie nowej implementacji dowolnego systemu plików.

Użytkownik może uruchomić swój własny serwer powodując zamontowanie systemu plików pod wskazanym katalogiem, do którego ma prawa zapisu. Nic nie stoi na przeszkodzie, aby udostępnić ten system plików innym, zaufanym użytkownikom. I to wcale niekoniecznie przez nadanie odpowiednich praw „*rwX*”.

Ciekawostką, z którą warto się oswoić, jest fakt, iż proces w Hurdzie może nie posiadać żadnego UID czy GID. Jest tak, dopóki nie przeprowadzi on autoryzacji, dzięki której zyskuje przywileje należne odpowiedniemu użytkownikowi. Serwer *ssh* nie będzie miał powodu działać z prawami administratora (choć póki co działa jeszcze tak samo jak w Linuksie). Zamiast działać z prawami superużytkownika, aż do momentu autoryzacji, po której następuje *pozbycie się* nadmiaru uprawnień, w Hurdzie *zyskałby* on UID odpowiedniego użytkownika. A więc niezależnie od błędów w serwerze *ssh* czy innym, nie byłoby możliwe uzyskanie zdalnego roota czy nawet praw innego użytkownika systemowego.

## Kilka ciekawostek...

- W Hurdzie, zamiast standardowych trzech zestawów praw dostępu „*rwX*”, każdy plik ma cztery zestawy. Ten dodatkowy obowiązuje dla użytkowników bez żadnego UID.

### [ W Sieci: ]

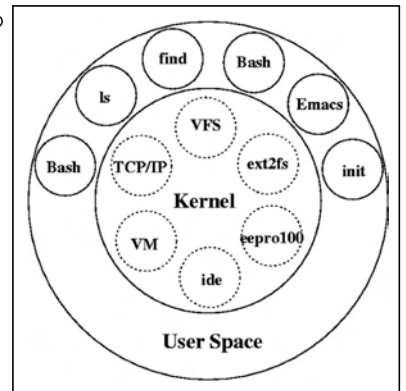
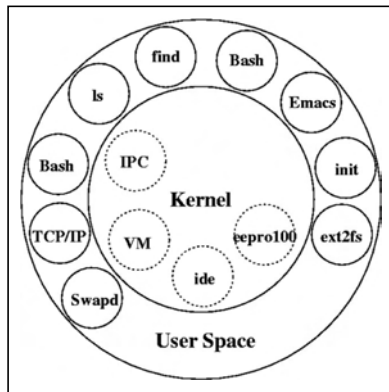
[**hurd**] – <http://www.gnu.org/software/hurd/hurd.html>  
[**HurdFAQ**] – <http://www.gnu.org/software/hurd/faq.en.html>  
[**noCD**] – <http://www.planetmirror.com/pub/debian-cd/unofficial/hurd/>  
[**trans**] – <http://www.debian.org/ports/hurd/hurd-doc-Translator>  
[**manual**] – <http://www.debian.org/ports/hurd/reference-manual/hurd.html>  
[**geneza**] – <http://www.cs.pdx.edu/~trent/gnu/hurd/>  
[**Neal**] – <http://web.walfield.org/~neal/papers/>  
[**Debian GNU/BSD**] – <http://sourceforge.net/projects/debian-bsd/>  
[**Debian GNU/w32**] – <http://debian-cygwin.sourceforge.net/>  
[**Hurd@IRC**] – #hurd irc.openprojects.net



Jeżeli dla danego pliku ten zestaw nie został zdefiniowany, używa się praw dla użytkownika „other”.

- Autoryzacja w Hurdzie, np. podczas logowania, jest realizowana za pomocą dwóch serwerów. Najpierw serwer *passwd* sprawdza, czy przekazana para *id użytkownika i hasła* znajduje się w */etc/passwd* i */etc/shadow*. Jeśli tak, zwraca *token*, który proces użytkownika przekazuje serwerowi *auth*, nadającemu procesowi odpowiedni UID.
- Każdy system plików w Hurdzie jest tzw. tłumaczem [trans]. Jest to serwer, który odpowiada za zwrócenie odpowiedzi do programu wywołującego funkcję dostępu do ścieżki do pliku. Może on także po prostu wskazać inne miejsce w systemie plików, inną ścieżkę i do niej przekierować wywołanie. Tłumaczami mogą być także pojedyncze pliki, np. dowiązanie symboliczne. Planuje się stworzenie tzw. *shadowfs*, którego jedyną funkcją będzie przekierowywanie odwołań do plików do innych systemów plików. Pozwoli on bardzo elastycznie zarządzać całą strukturą systemu plików Hurda.

Rysunek 1 Podsystemy jądra monolitycznego



Rysunek 2 Podsystemy współpracujące z mikrojądrem – nie wszystkie są jeszcze wydzielone do przestrzeni użytkownika

## Co z tego wynika dziś?

Hurd (w sensie jądra systemu) jest mniej więcej w tym stadium, co Linux 4-5 lat temu, jednak dzięki możliwościom znacznie łatwiejszego rozwijania kolejnych komponentów systemu poza samym jego jądrem, bardzo szybko jest ulepszany. W tej chwili znajduje się w stadium, gdy deweloperzy mogą go w miarę normalnie używać. Nie posiada własnego instalatora, a przy instalacji korzysta się z Linuksa. Praktycznie nie działa na systemach SMP (choć, gdy tylko zostaną zakończone ważniejsze prace, możliwość ta zostanie zaimplementowana – znacznie łatwiej i szybciej niż to się działo z Linuksem, który nigdy nie był projektowany, by działać na wielu procesorach). Na dziś podstawowym systemem plików Hurda jest *ext2*, jednak na partycji nie większej niż 1GB. W tym momencie Hurd jest dostępny jedynie na architekturze IA32 (i386) (choćby było, że starsze wersje przenoszono na inne procesory).

Do obsługi urządzeń zewnętrznych Hurd używa biblioteki OSKit, która jest uniwersalną biblioteką sterowników urządzeń stworzoną na Uniwersytecie Utah. Dzięki niej Hurd może używać sterowników IDE, SCSI i Ethernetu z Linuksa 2.2. Od jakiegoś czasu działają także serwery XFree86 i prostsze menedżery okien, jak Window Maker czy Black Box.

Wydawać by się mogło, że wiele jest tych ograniczeń i niedoróbek Hurda, jednak nie wynika to z niedbałości, bałaganu czy błędnego zamysłu, ale z faktu, że niewielka liczba deweloperów Hurda do tej pory zajmowała się przede

wszystkim doprowadzeniem systemu do stanu, w którym może go używać każdy w miarę doświadczony użytkownik. I cel ten właśnie jest osiągnięty! W chwili, gdy piszę te słowa, dostępny jest zestaw obrazów CD Hurda oznaczony H2 – myślę że warto po niego sięgnąć, choćby z ciekawości. A nawet jeśli nie, to warto dowiedzieć się czym jest Hurd, bo może za kilka lat...

## Co przyszłość pokaże ?

Przyszłość dla Hurda zaczyna się już dziś. Biorąc pod uwagę łatwość rozwijania tego systemu, a także istniejącą bazę kodu jądra Linuksa, który może być przeniesiony do użycia z jądrem Hurda, no a przede wszystkim całą masę programów, które dziś działają z Linuksem, powinniśmy spodziewać się dość sporego wzrostu zainteresowania tym systemem.

W styczniu 1992 roku podczas publicznej dyskusji z Andym Tanenbaumem Linus w liście pod tytułem „Re: LINUX is obsolete” (tłum. „LINUX jest przestarzały”) napisał: „Jeśli dziś piszesz programy dla Linuksa, nie powinieneś być zbyt zaskoczony, gdy w XXI. wieku po prostu zrekompilujesz je dla Hurda.”

## Inne Debiany

Przeglądając archiwa list dyskusyjnych Debiana można znaleźć wzmianki o jeszcze dwóch portach: w32 – współpracującym z jądrem Windows (z użyciem biblioteki *cygwin*) i BSD – działającym z jądrem BSD. Pierwszy z tych projektów powstał dość niedawno, ale już można go używać i jest on intensywnie rozwijany. Drugi natomiast pozostaje jak dotąd w sferze zamysłów – na pocieszenie wyjątkowo prezentujemy „diabełka” BSD. :-)

Będzie mi bardzo miło Czytelniku, jeśli zechcesz podzielić się ze mną swoimi uwagami, komentarzami na temat artykułu czy też po prostu będziesz miał jakieś pytania i wyślesz je do mnie na adres: [debian@linux.com.pl](mailto:debian@linux.com.pl). Do zobaczenia za miesiąc! 🐧