Systemy oparte na mikrojądrze

Grzegorz Anielak
Andrzej Kacperski
Remigiusz Mytyk

Dlaczego mikrojądro?

Początkowo systemy operacyjne były stosunkowo małe ze względu na ograniczenie wielkością pamięci. Z czasem możliwości komputerów zaczęły się zwiększać, pojawiało się wiele nowych urządzeń, które systemy musiały obsługiwać.

W 1977 pojawił się BSD (Berkeley Software Distribution), w którym pojawiły się funkcje takie jak obsługa dodatkowych systemów plików i kompletnego protokołu TCP/IP. Jądra rozrastały się, przez co stawały się coraz bardziej podatne na błędy, trudne w utrzymaniu i rozbudowie.

Mikrojądra powstały jako odpowiedź na ten wzrost. W założeniu miały pozwolić na łatwiejsze zarządzanie kodem dzięki podziałowi na usługi działające w przestrzeni użytkownika.

Koncepcja

Mikrojądro, według założeń nie udostępnia żadnych usług, a jedynie mechanizmy niezbędne do zaimplementowania takich usług. W związku z tym część logiki systemu musi być zaszyta w przestrzeni użytkownika.

Jądro zawiera tylko najbardziej niezbędne elementy (tj. komunikację międzyprocesorową, zarządzanie wątkami, obsługę przerwań i wyjątków). Inne zadania takie jak obsługa urządzeń, systemu plików i sieci realizowana jest w przestrzeni użytkownika przez osobne serwery.

Do najbardziej znanych systemów operacyjnych o mikrojądrach należą: MINIX, Hurd, Amoeba, QNX. Mikrojądrami są także Mach oraz L4. Firma Microsoft prowadzi projekt badawczy o nazwie Singularity mający na celu stworzenie systemu opartego na mikrojądrze, który w przyszłości może stać się częścią Microsoft Midori.

MINIX

MINIX jest systemem uniksopodobnym (tj. o zbliżonej do Uniksa budowie, ale o kodzie niewywodzącym się z BSD lub System V) stworzonym przez Andrew Tanenbauma, profesora na Uniwersytecie Vrije w Amsterdamie. MINIX to system tworzony w celach edukacyjnych (co jest wytłumaczeniem dla jego licznych ograniczeń). Pierwsza wersja systemu pojawiła się w roku 1987, czemu towarzyszyło pojawienie się książki Operating Systems Design and Implemetation, w której zawarto elementy kodu źródłowego systemu. W roku 2005 pojawiła się wersja 3.0 systemu wspierająca tylko architekturę x86. Zamierzeniem twórców było, aby był to system używany na słabych maszynach, od którego wymaga się dużej niezawodności.

GNU/Hurd

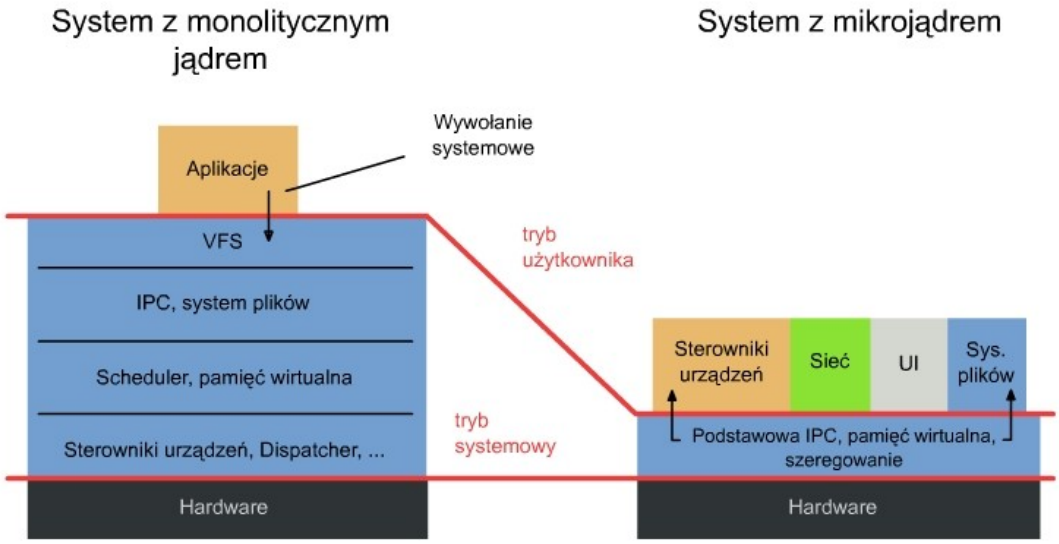
GNU/Hurd to jądro systemu, korzystające aktualnie z mikrojądra Mach 4. GNU/Hurd jest powstał w 1984 roku i od tego czasu jest rozwijany w ramach projektu GNU. Ciekawą cechą tego jądra jest jest mechanizm tzw. translatorów, działających na zasadzie filtrów nakładanych na konkretne miejsca w drzewie katalogowym. GNU/Hurd wykorzystywany jest w dystrybucji Debian GNU/Hurd.

Jądro monolityczne a mikrojądro

Podział systemów na mikrojądro i procesy działające w przestrzeni użytkownika powoduje, że struktura takich systemów znacząco różni się od jąder monolitycznych. Systemy z jądrami monolitycznymi mają budowę poprzecznych warstw. Aplikacje komunikują się z jądrem za pomocą wywołań systemowych.

Systemy oparte na mikrojądrach mają horyzontalną budowę, zaś procesy mogą komunikować się z innymi poprzez wywołanie systemowe skierowane do odpowiedniego

serwera poprzez IPC. Serwery te są uruchamiane w trakcie uruchomienia systemu, aby udostępniać usługi zwykłym aplikacjom.



Mikrojądro udostępnia jedynie podstawowe mechanizmy takie jak: zarządzanie pamięcią, zarządzanie wątkami, podstawowa komunikacja międzyprocesowa, obsługa przerwań i wyjątków. Resztę zadań wykonują procesy wykonywane w przestrzeni użytkownika. Zazwyczaj w systemie funkcjonują takie serwery jak: serwer systemu plików, serwery interfejsu użytkownika, serwery sieci oraz sterowników.

Korzystanie z pamięci przez proces nie musi odbywać się za pośrednictwem mikrojądra. Niektóre procesy (np. sterowniki dysków) mogą uzyskać prawa do bezpośredniego zapisu do tych dysków.

Jeśli dojdzie do błędu, serwery mogą być w łatwy sposób wznawiane, co jest chyba największą zaletą systemów opartych o mikrojądra. Ich przewagą nad systemami o jądrach monolitycznych jest także większe bezpieczeństwo. Wyobraźmy sobie, że w sterowniku urządzenia sieciowego wystąpi przepełnienie bufora. W systemie opartym na mikrojądrze reszta systemu pozostanie sprawna, podczas gdy w jądrze monolitycznym może nastąpić ingerencja w przestrzenie adresowe innych procesów, a nawet, co gorsza, samego jądra.

System MINIX posiada dwa serwery umożliwiające tzw. self-repairing, czyli automatyczną naprawę w przypadku wystąpenia błędu. Pierwszym z nich jest serwer reinkarnacji. Serwer ten jest rodzicem wszystkich sterowników i serwerów. Jego zadaniem jest wznawianie sterowników, które ulegną awarii. Wysyła on cyklicznie żądania zwrócenia statusu do wszystkich sterowników w systemie, a następnie w przypadku nieuzyskania odpowiedzi lub uzyskania błędnej odpowiedzi, podmienia dany sterownik, zastępując go jego świeżą kopią. Zamiana ta odbywa się automatycznie i w sposób niewidoczny dla użytkownika. Drugim serwerem jest data store. Jest to minibaza danych, która przechowuje informacje na temat stanu komponentów działających w systemie. Na przykład w przypadku sterownika pamięci RAM, mógłby być to początek i koniec przestrzeni adresowej, aby w przypadku awarii starego sterownika nowa kopia mogła kontynuować jego pracę.

KOMUNIKACJA

Wstęp

Wydzielenie sterowników i serwerów z jądra systemu operacyjnego do oddzielnych modułów działających, jako samodzielne procesy w trybie użytkownika niesie za sobą wiele korzyści, ale aby można było z tego skorzystać musimy sobie poradzić z pewnymi problemami:

- Zależności pomiędzy modułami
- Ograniczone pole manewru w trybie użytkownika

Zależności

Pomimo iż sterowniki i serwery działają jako oddzielne procesy, nie mamy możliwości całkowitego ich uniezależnienia od siebie nawzajem. Występuje pomiędzy nimi szereg

zależności takich jak:

- Sterowniki i serwery potrzebują odwoływać się do jądra
- Jedne sterowniki wywołują inne sterowniki

Potrzebujemy więc mechanizmu za pomocą, którego procesy oraz jądro będą mogły sprawnie i bezpiecznie komunikować się ze sobą.

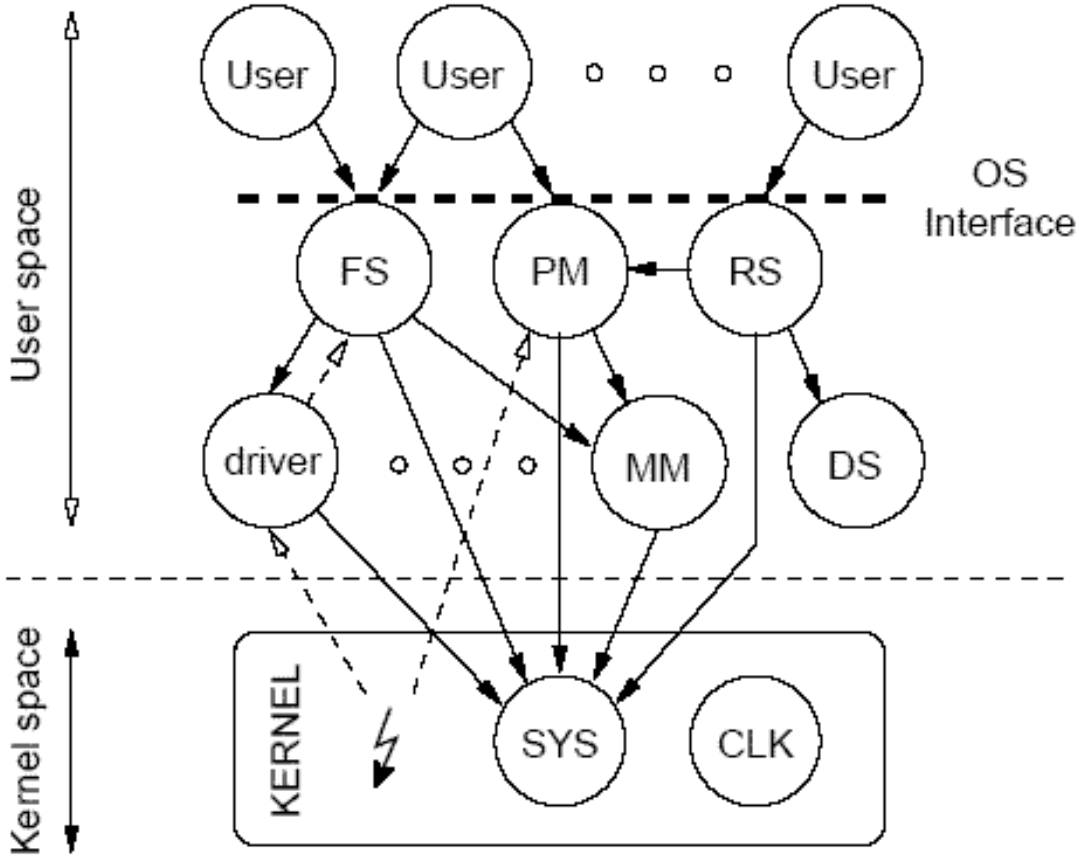
IPC

Procesy do komunikowania się pomiędzy sobą używają mechanizmu IPC. Komunikacja odbywa się synchronicznie za pomocą „wiadomości z żądaniem”, które są zawsze takiego samego typu i rozmiaru.

Najważniejszym kryterium przy tworzeniu Minixa, było zapewnienie jak największego bezpieczeństwa systemu. W związku z tym wszelkie niebezpieczne elementy są upraszczane na tyle ile to możliwe. Ujednolicenie rozmiaru i typu wiadomości miało właśnie na celu zminimalizowanie możliwości powstawania błędów przy przekazywaniu wiadomości.

Sterowniki, serwery i jądro odwołują się do siebie nawzajem, gdy chcą aby jakiś inny komponent wykonał dla nich jakąś czynność, dostarczył im jakieś dane, bądź udzielił pozwolenia na dostęp do przechowywanych przez niego danych. Wysyłanie „wiadomości z żądaniem” polega właśnie na wysyłaniu takiego żądania i czekaniu na odpowiedź, która zawiera informacje o pomyślności przeprowadzonych operacji bądź informacje na temat możliwości dostępu do pewnych danych.

Kolejnym uproszczeniem jest całkowite zrezygnowanie z buforowania. Przekazywane informacje są kopiowane z przestrzeni adresowej jednego procesu bezpośrednio do przestrzeni adresowej drugiego procesu, z którym się on komunikuje.



Przedstawiony powyżej diagram przedstawia schemat komunikacji pomiędzy wszystkimi procesami. Na diagramie przestawionych jest tylko kilka przykładowych kierunków przepływu wiadomości. Możemy sobie tylko wyobrazić jak wyglądałby ten rysunek, gdybyśmy chcieli zaznaczyć wszystkie możliwości przepływu tych wiadomości.

Każde zadanie wykonywane w systemie wiąże się z dużą ilością przesyłanych wiadomości pomiędzy modułami. Jest to swoiste wąskie gardło Minixa. Właśnie dlatego system komunikacji między procesorowej musi być wyjątkowo sprawny i niezawodny.

Blokady

Rozwiązanie przedstawione powyżej nie jest jednak zadawalające, gdyż zawiera jedną poważną wadę, a mianowicie możliwość blokady. Proces, który chce nadać wiadomość musi czekać na odbiorcę, a proces który chce zwrócić odpowiedź musi czekać na nadawcę wiadomości. Oczywiście nie zawsze stanowi to problem, ale nie wszystkie procesy mogą sobie pozwolić na takie czekanie, w szczególności jądro. Natomiast istnieje też zagrożenie przeprowadzania pewnych ataków w postaci nadawania wiadomości i nie odbierania odpowiedzi, co prowadzi do blokad..

Improved IPC

W celu zapobiegania między innymi tym problemom, o których wspomniano powyżej, wprowadzono pewne ulepszenie do mechanizmu IPC:

- Po pierwsze ograniczono swobodę w wysyłaniu wiadomości do wszelkich uruchomionych procesów. Dodano bitmapę, w której zawarte są informacje, z którymi procesami dany proces może się komunikować.
- Po drugie wprowadzono możliwość wymuszania „randki”. To znaczy, że do wymiany informacji pomiędzy dwoma procesami musi dojść jednocześnie (wysłanie żądania i otrzymanie odpowiedzi) – w jednym wywołaniu. Uniemożliwia to blokowanie procesów.
- Po trzecie wprowadzono możliwość asynchronicznego wysyłania wiadomości nieblokujących. W bitmapie(o której była mowa powyżej) przechowywane są również informacje, które procesy mogą takie specjalne wiadomości wysyłać.

Przerwania

Program obsługi przerwań jest jednym z niewielu znajdujących się w mikrojądrze. Zanim omówimy go dokładniej, warto wspomnieć, iż oddzielenie sterowników od mikrojądra, są odpowiedzialne odpowiednie moduły. Dzięki temu mechanizm przerwań został maksymalnie uproszczony i usprawniony. Jednym zadaniem do wykonania przez program obsługi przerwań jest odebranie przerwania i przetworzenia go na wiadomość, która jest wysyłana do odpowiedniego adresata.

Mechanizm powiadamiania

Jest to mechanizm umożliwiający wysyłanie asynchronicznych i nieblokujących wiadomości.
Proces wysyła nieblokującą wiadomość i normalnie kontynuuje swoje działanie, natomiast gdy odbiorca nie może w danej chwili odebrać wiadomości, powiadomienie jest przechowywane jako odpowiedni bit w bitmapie. Kiedy odbiorca jest gotowy do odebrania wiadomości jądro tworzy nową wiadomość na podstawie typu powiadomienia i wysyła ją do odbiorcy. Program obsługi przerwań wykorzystuje ten mechanizm aby uniknąć blokowania się gdy, któryś ze sterowników jest zajęty i nie może obsłużyć przerwania.

Praca w trybie użytkownika

Wcześniej omówiliśmy problemy związane z komunikacją dla wydzielonych modułów. Mamy jeszcze jeden problem jaki powstał po wydzieleniu modułów z jądra do oddzielnych procesów działających w trybie użytkownika. Gdy sterowniki i serwery działały w jądrze miały wszelkie prawa przysługujące w trybie jądra, teraz natomiast są ograniczone wyłącznie do swojej przestrzeni adresowej. MMU czuwa by wszystkie procesy mogły zaglądać tylko do swoich przestrzeni adresowych.

Natomiast sterowniki nadal potrzebują czasem odwoływać się do danych zgromadzonych w jądrze, bądź na przykład do I/O urządzeń zewnętrznych. W Minixie sterownik

taki każdorazowo potrzebuje poprosić o zgodę jądro, które pełni standardową rolę nadzorcy.

Wywołania systemowe

Procesy wysyłają wszelkie prośby do jądra w postaci *wywołań systemowych*. W związku z tym Minix zapewnia wsparcie dla tego mechanizmu w postaci zestawu dodatkowych specjalnych wywołań systemowych, które są wykorzystywane przez sterowniki między innymi do kopiowania pewnych informacji z jądra czy czytania z I/O.

Wywołania systemowe odbierane są przez *system tast*, a następnie tworzona jest wiadomość i wysyłana jest do odpowiedniego sterownika bądź serwera.

Fork ()

Prześledźmy schemat przepływu komunikatów na przykładzie funkcji systemowej *fork()*:

1. Wywołujemy funkcję biblioteczną *fork()*
2. *System task* odbiera wywołanie funkcji i tworzy wiadomość z żądaniem do *Process Manager*
3. *Process Manager* odbiera wiadomość i sprawdza czy istnieje wolny slot dla procesu. Następnie wysyła wiadomość z żądaniem do *Memory Manager*
4. *Memory Manager* alokuje pamięć dla nowego procesu i odpowiada do *Process Manager*.
5. Teraz *Process Manager* wysyła kolejną wiadomość z żądaniem, tym razem do jądra.
6. Jądro inicjalizuje nowy proces, aktualizuje tabelkę z procesami a na koniec wysyła odpowiedź do *Process Manager*.
7. *Process Manager* odbiera odpowiedź i sam może już wysłać odpowiedź.
8. Na koniec następuje odebranie wiadomości od *Process Manager* i powrót z wywołania funkcji *fork().*

WYDAJNOŚĆ

To jak szybkie są mikrojądra w ich własnym światku wydaje się być tematem pobocznym. Kogo to obchodzi skoro mają taki piękny, przemyślany design? (Mnie!) Zakłada się, że użytkownicy chętnie oddadzą kilka procent wydajności w zamian za niezawodny, bezpieczny system. Jednocześnie ich twórcy upierają się, że ich mikrojądra rzeczywiście są jedynie kilka-kilkanaście procent za tradycyjnymi jądrami monolitycznymi. Użytkownicy rzeczywiście chętnie wymienią na niezawodność i tak niezauważalną dla nich kilkuprocentową róznicę w osiągach.. tylko, że niestety osiągi mikrojąder obecnie nie są nawet na zbliżonym poziomie.

Dlaczego jest wolno?

Słówko o tym skąd bierze się trudność napisania wydajnego mikrojądra. Przyczyny są prozaiczne, wysoko-poziomowe abstrakcje - to co stanowi o elegancji idei mikrojąder - znacznie utrudniają niskopoziomowe tunningowanie. I tak kolejno, sterowniki działają w osobnych przestrzeniach adresowych dlatego przełączanie kontekstu między nimi jest kosztowne, nie mogą współdzielić informacji, zamiast tego porozumiewają się za pomocą komunikatów, co oznacza niepotrzebne kopiowanie danych. To właśnie komunikacja okazuje się głównym wąskim gardłem mikrojąder. Wreszcie, sterowniki działają w trybie użytkownika, daleko od sprzętu, dlatego trudno korzystać im z specyficznych dla architektury wydajnościowych trików.

Benchmark

Andrew Tanenbaum, twórca systemu Minix, na jednej ze swojej prezentacji w styczniu 2007 roku przytoczył wyniki porównania Minix-a z pewnym jądrem monolitycznym. Uwaga, Minix tracił rzekomo już tylko 5%-10%. Niestety nikomu nie udaje się powtórzyć takich wyników, testy Tanenbauma wydają się mało wiarygodne. Jeden z redaktorów portalu lwn.net podjął się porównania Minix-a 3 z Linux-em 2.6.17. (wykresy można obejrzeć w prezentacji). W ramach testu uruchamiał program IOZone i zestaw skryptów UnixBench.

IOZone

IOZone to program mierzący wydajność operacji dyskowych. O dziwo Minix wypadł na

tym polu nieźle, wcale nie gorzej niż Linux. Nie należy się jednak pochopnie cieszyć gdyż test został wykonany w słabych warunkach – na dysku o małej pojemności 4GB (z powodu ograniczeń Minixa).

UnixBench

UnixBench to obszerny zestaw skryptów testujących przeróżne parametry komputera. Spora uwaga poświęcona jest mierzeniu typowo systemowych operacji, takich jak np. tworzenia procesów. Praktycznie we wszystkich testach Minix przegrywa z kretesem. (szczegóły w prezentacji)

Własne testy

Wykonaliśmy też własne, proste testy. Potwierdzają czarny scenariusz – Minix jest wolny. Test polegał na wygenerowaniu miliona kolejnych liczb, spakowaniu ich programem bzip2, a następnie rozpakowaniu. Wykonaliśmy go na Minix-ie i Linux-ie 2.6.27, w obu przypadkach na maszynie wirtualnej.

Wyniki:

System	seq 1mln > a.txt	bzip2 a.txt	bzip2 -d a.txt.bz2
Minix	1,35 s.	3,56 s.	2,23 s.
Linux	1,27 s.	1,04 s.	0,52 s.

Minix fails.

W obronie Minix-a

Zanim dokonam podsumowania chciałbym wziąć przegrany system w obronę. Minix jest to projekt uczelniany, mało dojrzały, tworzony przez garstkę ludzi, nie ma szans z intensywnie rozwijanym od kilkunastu lat Linux-em. Dodatkowo twórca Minix-a nie ukrywa, że ważny jest dla niego walor dydaktyczny systemu, świadomie rezygnuje z niektórych optymalizacji w imię przejrzystości kodu.

Wnioski

Nie ma wątpliwości, Linux to póki co zupełnie inna liga. Nie należy jednak na tej podstawie przesądzać o wyższości jąder monolitycznych nad mikrojądrami. Należy rozróżnić dyskusję o pomysłach od dyskusji o implementacjach. Obecny status implementacji mikrojąder: słabe. Pomysł jednak pozostaje ciekawy dlatego warto by twórcy mikrojąder przywiązali większą wagę do wydajności. O ile stratą 5%-10% zmartwiony będzie mało kto, o tyle 800% skutecznie zniechęci wszystkich.

HURD – translatory

Translatory to złote rozwiązanie systemu GNU/Hurd. Prawdopodobnie rozwiązuje wszystkie problemy ludzkości. Poważnie mówiąc jest to (wcale nienajnowszy) pomysł zastosowania struktury katalogów i plików jako abstrakcji do przeróżnych rzeczy. I tak kolejno w Hurdzie mogą być to sterowniki urządzeń, zasoby sieciowe, aplikacje użytkownika, dowiązania do plików czy wreszcie same pliki. Efekt ten uzyskuje się dzięki możliwości podłączenia w dowolnym punkcie montowania programu – translatora - odpowiadającego na żądania użytkownika. Do prezentacji przygotowaliśmy demo – prosty translator do gry w kółko i krzyżyk.

Użycie:
$ gcc -o game game.c -ltrivfs -lfshelp (kompilacja)
$ settrans -a -g plik game (podłączenie translatora)

$ cat plik (wyświetlanie planszy)
$ echo 1-9 >> plik (ruch na planszy)
$ echo r >> plik (rozpoczęcie nowej gry)

DYSKUSJA LT Z AT

Na początku 1992 roku na grupie dyskusyjnej `comp.os.minix` odbyła się dyskusja między Andrew Tanenbaumem, Linusem Torvaldsem i innymi osobami związanymi z tematyką systemów operacyjnych.

Argumenty ZA używaniem mikrojądra:

- Jądra monolityczne i mikrojądra mają bardzo podobne osiągi (Tanenbaum powołał się tutaj na pracę Ricka Rashida porównującą wydajność jądra Mach 3.0 i systemów monolitycznych),
- Linux to monolityczny system, co jest powrotem do lat 70. (Tanenbaum porównał to do przepisania działającego programu w C do języka Basic),
- MINIX został zaprojektowany, aby gwarantować przenośność. Linux jest przywiązany do architektury 80x86,
- MINIX swoje ograniczenia „zawdzięcza" temu, że jego twórca jest nauczycielem akademickim i był tworzony, aby studenci mogli na nim pracować na słabszych maszynach,
- MINIX ułatwia badaczom usuwanie i wymianę poszczególnych modułów,
- Gdyby nie determinacja Tanenbauma, aby system pozostał tak prosty jak to tylko możliwe, utraciłby on swoje walory jako pomoc naukowa.

Argumenty PRZECIWKO mikrojądrom:

- Nie ma dobrej implementacji pozwalającej wykorzystać zalety wynikające z zastosowania mikrojądra.
- Przenośność jest zapewniona kosztem, braku sprzętowego wsparcia na jakichkolwiek maszynach.
- Programowanie w systemie opartym na mikrojądrze trudniejsze – częsta potrzeba wykorzystywania algorytmów współbieżnych oraz brak współdzielonych struktur danych.

W związku z tym, że dyskusja ta odbyła się ponad 18 lat temu, część z powyższych argumentów dawno straciła ważność. Gdybyśmy dziś chcieli podsumować zalety systemów opartych na mikrojądrach należałoby wymienić:

- niewielkie rozmiary,
- bezpieczeństwo (lepsza ochrona przed następstwami błędów),
- stabilność,
- modularność,
- niewielkie wymagania sprzętowe,
- self-repairing.

Systemy oparte na mikrojądrze wciąż nie doczekały się implementacji, która mogłaby konkurować z systemami monolitycznymi w standardowych zastosowaniach dla przeciętnych użytkowników. Dopóki Minix będzie rozwijany jako projekt uczelniany bez poważnego wsparcia marketingowego itp., to nie ma on większych szans konkurować chociażby z Linuxem.

Bibliografia:

- Dokumentacja MINIXa (http://www.minix3.org/doc/),
- Dyskusja Linusa Torvaldsa z Andrew Tanenbaumem (http://oreilly.com/catalog/opensources/book/appa.html),
- MINIX 3: A Highly Reliable, Self-Repairing Operating System (Operating Systems Review, lipiec 2006),
- Wikipedia (hasła: Andrew S. Tanenbaum, Microkernel, MINIX, MINX 3).