Systemy oparte na mikrojądrze

9 grudnia 2008

Spis treści

1 Mikrojądro, autor: Olga Kowalczuk

1.1 Jaka jest istota mikrojądra?

Mikrojądro vs. jądro monolityczne

Mikrojądro umieszcza kluczową część logiki systemu operacyjnego w jądrze, a pozostałą - w przestrzeni użytkownika. Systemowe procesy w przestrzeni użytkownika to *serwery*. Funkcje pełnione przez jądro monolityczne są rozdzielone między mikrojądro i serwery.

W jądrze monolitycznym aplikacja użytkownika komunikuje się z jądrem za pomocą *wywołania systemowego*. W systemie opartym na mikrojądrze aplikacja użytkownika komunikuje się z serwerem korzystając z *komunikacji międzyprocesowej* (IPC). Stąd w przypadku jądra monolitycznego mówimy o *wertykalnej strukturze* (góra-dół, aplikacja-jądro), a w przypadku mikrojądra - o *horyzontalnej strukturze* (na jednym poziomie, serwer w przestrzeni użytkownika - aplikacja użytkownika). Komunikacja IPC przebiega następująco. Aplikacja wysła żądanie do jądra, jądro przekazuje je do odpowiedniego serwera. Odpowiedź idzie tą samą drogą, tylko w drugim kierunku.

Serwer jest procesem w przestrzeni użytkownika, jednak są między nim, a procesami użytkownika pewne różnice. Serwery mają dostęp do niektórych obszarów pamięci fizycznej, a ponadto niektóre z nich są wywoływane przez jądro w momencie bootowania.

Mikrojądro może samodzielnie pełnić różny zakres funkcji, od bardzo szerokiego, niewiele węższego od - jądra monolitycznego, do bardzo wąskiego. Jednak nawet minimalne mikro jądro zarządza:

- niskopoziomową przestrzenią adresową
- wątkami
- komunikacją międzyprocesową

Serwery obsługują zazwyczaj:

- sterowniki urządzeń

- komunikację sieciową
- system plików
- interfejs użytkownika

Historycznie rzecz ujmując można określić mikrojądro jako powrót do korzeni. Kiedy komputery dysponowały 16 bitowym adresowaniem, jądra musiały być małe. Wprowadzenie 32 bitowych adresów umożliwiło zwiększenie kodu jądra i zakresu pełnionych przez nie funkcji. Rozwój wielkości jądra wiązał się z pewnymi niedogodnościami, stąd idea mikrojądra. Zalety i wady monolitycznego i mikrojądra omówimy w dalszej części tekstu.

1.2 Jak wygląda komunikacja międzyprocesowa?

Komunikacja międzyprocesowa (IPC) jest *synchroniczna* lub *asynchroniczna*.

W komunikacji asynchronicznej nadawca wysyła komunikat i kontynuuje wykonywanie kodu. Odbiorca dowiaduje się o komunikacie albo przez regularne sprawdzanie określonego adresu, albo przez otrzymanie określonego powiadomienia. Komunikacja asynchroniczna wymaga od jądra utrzymywania buforów i kolejek komunikatów oraz rozwiązywania problemu przepełnienia bufora lub kolejki. Komunikacja asynchroniczna oznacza też podwójne kopiowanie komunikatu przez jądro, najpierw od nadawcy do jądra, potem od jądra do odbiorcy, dwie zmiany kontekstu i osiem zmian trybu.

Komunikacja synchroniczna powoduje wstrzymanie pierwszego procesu chcącego się skomunikaować w oczekiwaniu na drugi. Komunikacja synchroniczna nie wymaga od jądra utrzymywania struktur danych służących przechowywaniu danych.

1.3 Quiz: Jądro monolityczne czy mikrojądro? Które zachowa się lepiej w poniższych sytuacjach?

1. Jako system czasu rzeczywistego?
2. Połączenie sieciowe padło z powodu przepełnienia bufora wywołanego przez atak na system poprzez przeglądarkę internetową?
3. Źle współpracujący IPC z CPU (komunikacja międzyprocesowa korzysta z procesora w sposób mało wydajny)?
4. Chcemy rozszerzyć zakres usług jądra?
5. Wiemy, że w jądrze jest błąd i chcemy go znaleźć?
6. Chcemy uzyskać tanią usługę dostępu do sterownika dla aplikacji użytkownika (w sensie liczby operacji wykonanych przez system)?

Odpowiedzi:

1. Jako system czasu rzeczywistego?
   Mikrojądro Mikrojądro składa się z małego jądra, na które łatwiej nałożyć ograniczenia czasowe niż na duże (wielomodułowe) jądro monolityczne. W mikrojądrze w dostarczeniu usługi uczestniczy jeden lub kilka serwerów, których czas odpowiedzi też można z góry oszacować.

2. Połączenie sieciowe padło z powodu przepełnienia bufora?
   System oparty na mikrojądrze będzie miał uszkodzoną pamięć serwera odpowiedzialnego za połączenia sieciowe, a pozostałe serwery i jądro nadal będą funkcjonować. W wielu sytuacjach wystarczy restartować serwer by zaczął poprawnie funkcjonować po upadku.
   W jądrze monolitycznym prawdopodobnie uszkodzona zostałaby też pamięć innych sterowników, a także samo jądro, co doprowadziłoby do awarii całego systemu.

3. Źle współpracujący IPC z CPU (komunikacja międzyprocesowa korzysta z procesora w sposób mało wydajny)?
   Jądro monolityczne Ponieważ usługi systemowe w mikrojądrze oparte są na komunikacji międzyprocesowej (IPC), jej jakość w większym stopniu wpływa na funkcjonowanie całego systemu niż w przypadku jąder monolitycznych.

4. Chcemy rozszerzyć zakres usług jądra?
   Mikrojądro W mikrojądrze wystarczy dodać nowy serwer oferujący daną usługę.
   Jądro monolityczne wymaga, w tej sytuacji, dodania nowego modułu, co jest trudniejsze.

5. Wiemy, że w jądrze jest błąd i chcemy go znaleźć?
   Mikrojądro Jądro mikrojądra ma krótszy kod, dlatego łatwiej go przeczytać i znaleźć błąd niż w jądrze monolitycznym. Stąd też wynika mniejsza liczba błędów występujących w kodzie mikrojądra.

6. Chcemy uzyskać tanią usługę dostępu do sterownika dla aplikacji użytkownika (w sensie liczby operacji wykonanych przez system)?
   Jądro monolityczne Mikrojądro pośredniczy w wymianie komunikatów między aplikacją użytkowanika, a serwerem udostępniającym usługę dostępu do sterownika, co wymaga kopiowania komunikatu i zmiany kontekstu.
   W jądrze monolitycznym, jądro ma bezpośredni dostęp do danych w buforze klienta.

1.3.1 Zalety mikrojądra i jądra monolitycznego

Mikrojądro

1. Prostsza budowa jądra.
2. Prostsze do debugowania.
3. Prostsze do utrzymania.
4. Prostsze do dodawania/modyfikacji usług.
5. Wyższy poziom bezpieczeństwa.

Jądro monolityczne

1. Ma lepsze czasy wykonywania usług.
2. Rzadziej zmienia tryb.

1.4 Literatura

- Roch, B. *Monolitic kernel vs. microkernel*
- $http://wiki.linuxquestions.org/wiki/Microkernel$
- $http://oreilly.com/catalog/opensources$/book/appa.html
- $http://en.wikipedia.org/wiki/Microkernel$
- $http://en.wikipedia.org/wiki/Monolithic_kernel$
- $http://linuxhelp.blogspot.com/2006/05/monolithic-kernel-vs-microkernel-which.html$
- $http://c2.com/cgi/wiki?MicroKernel$

2 GNU/Hurd, autor: Michał Łyczek

GNU/Hurd jest zgodnym z POSIXem wieloserwerowym systemem zbudowanym na bazie mikrojądra GNU/Mach.

Kiedy Richard Stallman założył projekt GNU w 1983 roku chciał stowrzyć system operacyjny składający się tylko i wyłącznie z wolnego oprogramowania. Niedługo po tym większość kluczowych narzędzi została zaimplementowana i wydana pod licencją GPL. Brakowało jednak kluczowego kawałka: jądra. Po rozważeniu kilku możliwości zdecydowano nie pisać nowego jądra od początku lecz wykorzystać jako podstawę mikrojądro Mach. Trzy lata po podjęciu tej decyzji Mach został wydany pod licencją umożliwiającą włączenie go do projektu GNU i dystrybucję jako części systemu. W 1998 rozpoczęto projekt Debian GNU/Hurd, a już w 2001 dostępne dla GNU/Hurd pakiety wypełniały obrazy trzech płyt CD.

W momecie powstawania Hurda istniało kilka systemów operacyjnych opartych o Macha, ale wszystkie dziedziczyły wady systemów z jądrem monolitycznym, ponieważ wykorzystywały pojedynczy serwer wykorzystujący mikrojądro. Serwer wraz z mikrojądrem zastępował zadania monolitycznego jądra. Z punktu widzenia projektanów Hurda było to rozwiązanie pozbawione większego sensu, ponieważ nie wykorzystywało potencjalnych możliwości systemu z mikrojądrem i usługami rozproszonymi pomiędzy wiele serwerów. W przypadku Hurda jest wiele serwerów, z których każdy jest odpowiedzialny za implementację konkretnych funkcjonalności systemu operacyjnego. Serwery są uruchamiane jako zadania (task) Macha i komunikują się za pomocą infrastruktury udostępnianej przez Macha. Każdy z nich udostępnie jedynie małą część funkcjonalności systemu, ale w razem tworzą kompletny zgodny z POSIXem system operacyjny.

Cytując Thomasa Bushnella z artykułu „Towards a New Strategy of OS design” (1996):

The GNU Hurd, by contrast, is designed to make the area of system code as limited as possible. Programs are required to communicate only with a few essential parts of the kernel; the rest of the system is replaceable dynamically. Users can use whatever parts of the remainder of the system they want, and can easily add components themselves for other users to take advantage of. No mutual trust

need exist in advance for users to use each other's services, nor does the system become vulnerable by trusting the services of arbitrary users.

Czyli Hurd jest zbiorem serwerów uruchomionych na bazie mikrojądra Macha dostarczających zgodny z POSIXem, łatwo rozszerzalny system operacyjny.

Dostęp do portów serwera uzyskuje się w Machu dzięki serwerowi nazw. Procesy za pomocą funkcji systemowych otrzymują port serwera nazw i wysyłając komunikaty pod ten port mogą uzyskać dostęp do portów innych serwerów, które wcześniej się zarejestrowały w serwerze nazw.

W Hurdzie rolę serwera nazw przejął system plików. Dzięki temu rozszerzono płaską strukturę listy zarejestrowanych serwerów w serwerze nazw do struktury drzewiastej systemu plików. Dostęp do portu serwera odpowiedzialnego za obsługę danego elementu drzewa katalogowego uzyskuje się dzięki funkcji `hurd_file_name_lookup`. Ze względu na taką orgaznizację serwery osadzone w systemie plików są nazywane translatorami. Translatory tłumaczą odwołania do systemu plików na porty dla danej ścieżki w drzewie katalogów. Ze względu na działanie możemy wyróżnić pasywne i aktywne translatory. Te drugie są zapisami komend do uruchomienia serwerów (translatorów aktywnych) w inode'ach systemu plików.

3 Minix3, autor: Jan Świeżyński

3.1 Po co nowy system operacyjny?

Twórcy systemu *Minix3* chcieli stworzyć system, który byłby niezawodny w o wiele większym stopniu niże te, istniejące dotychczas. Chcieli wyjść naprzeciw „zwykłym" użytkownikom komputera, których zaczęło pojawiać się coraz więcej, a którzy od komputera oczekują sprawnego działania bez zawieszania się, takiego jakie oferują im inne urządzenia.

Żeby osiągnąć zamierzony efekt potrzeba było systemu, którego jądro byłoby naprawdę małe. Błędne działanie systemu nie jest bowiem wynikiem złego działania hardwaru, lecz softwaru. Nie da się ustrzec błędów pisząc 5 mln linii kodu (jądro Windowsa XP), czy nawet 2,5 mln (Linux). Jądro systemu Minix3 miało mieć około 5000 tysięcy linii kodu.

Dodatkowo system miał posiadać właściwość *„self-repairing"*. Miał automatycznie wykrywać defekty i wymieniać źle działające komponenty. To wszystko miało się dziać bez wiedzy użytkownika, który nie zauważałby nawet, że pojawił się jakiś błąd.

3.2 Architektura systemu

3.2.1 Zarys ogólny

W celu zapewnienia większej niezawodności systemu twórcy postanowili uruchamiać wszystkie serwery i sterowniki w przestrzeni użytkownika. Zarządać nimi miało bardzo niewielkie, zaufane mikrojądro. Dodatkowo nad poprawnym działaniem systemu miał czuwać specjalny serwer zwany *Reincarnation Server*, który zapewniał właściwość *„self-repairing"*.

3.2.2 Procesy i komunikacja międzyprocesorowa

Procesy są w zasadzie takie same jak te w systemach UNIX-owych, mają prywatną przestrzeń adresową, rejestry i mogą przechowywać wiele zasobów, jednak część zasobów jest zarządzana przez serwery przestrzeni użytkownika. Mimo że serwery i sterowniki wymagają większych przywilejów, to jednak wszystkie procesy są traktowane przez jądro jednakowo.

Komunikacja międzyprocesorowa jest synchroniczna i działa na zasadzie spotkań. Procesy komunikują się ze sobą i z jądrem poprzez wiadomości o stałej długości. Dodatkowo istnieje mechanizm powiadomień, który pozwala wysyłającemu na przekazanie informacji bez blokowania się, jeśli odbiorca nie jest gotowy.

3.2.3 Przestrzeń jądra

Mikrojądro obsługuje podstawowe mechanizmy, spośród tych, za które odpowiada normalnie monolityczne jądro(obsługa przerwań, zarządzanie procesorem i pamięcią, kolejkowanie procesów, komunikacja międzyprocesorowa i wywołaniam jądra). W przestrzeni jądra znajdują się dodatkowo dwa procesy. Pierwszym z nich jest *sterownik zegara*, który jest niezbędny do kolejkowania procesów. Drugim jest *system task*, który odpowiada za obsługę wywołań kierowanych do jądra.

3.2.4 Przestrzeń użytkownika

Warstwa sterowników

W warstwie sterowników leżą wszystkie sterowniki urządzeń, poza sterownikiem zegara. Każdy sterownik jest odrębnym procesem i każdy dostaję swój kwant czasu procesora, kiedy przychodzi jego kolej. Poza kilkoma wyjątkami, sterowniki mają takie same prawa jak inne procesy.

Warstwa serwerów

Serwery również są zwykłymi procesami. *file server* jest programem, który

zajmuje się obsługą POSIX-owych wywołań dotyczących plików. Innymi serwerami obsługującymi POSIX-owe wywołania są *process manager* i *network serwer*. Istnieją jeszcze dwa specjalne serwery odpowiedzialne za właściwość „self-repairing".

3.3 Własność „self-repairing"

Automatyczne naprawianie błędów wspomagane jest przez dwa specjalne serwery. *Reincarnation server* zarządza wszystkimi pozostałymi serwerami i sterownikami, a także monitoruje stan systemu. *Data store* zapewnia miejsce do zbackupowania stanu systemu, a także pozwala serwerowi reincarnation na publikowanie informacji o konfiguracji systemu.

Reincarnation serwer jest procesem macierzystym wszystkich pozostałych procesów warstwy użytkownika. Dzięki temu wie, które procesy przestały działać i może na to odpowiedno zareagować. Poza tym wysyła co jakiś czas do pozostałych procesów żadanie zwrócenia statusu i jeśli proces nie odpowie w przeciągu określonego czasu, to podejmuje dalsze działanie.

Data store to serwer mini bazy danych, która umożliwia odzyskiwanie niektórych utraconych stanów po restarcie komponentu, a także informowanie zależnych procesów.

3.4 Wydajność

Twórcy Minixa3 nie skupiali się na tym, żeby ich system był tak samo szybki jak linux oparty o jądro monolityczne. Woleli raczej używać prostszych, nieco bardziej kosztownych algorytmów, które są łatwiejsze do zaimplementowania, przy których kodowaniu ryzyko popełnienia błędu jest mniejsze. Nie znaczy to jednak, że Minix3 jest systemem bardzo wolnym. Przy dzisiejszych wydajnościach sprzętu różnice nie są aż tak bardzo odczuwalne.