Wprowadzenie do systemu Minix

Opis powstał na podstawie pracy dyplomowej Adama Pogorzelskiego *Opracowanie laboratoryjnej wersji systemu Minix 2.0* wykonanej w 1998 roku w IAiIS PW.

1. Wprowadzenie

System operacyjny (SO) to program lub zbiór programów i procedur spełniających dwie podstawowe funkcje :

- tworzenie maszyny wirtualnej,
- zarządzanie zasobami komputera.

Pierwsza z nich polega na ukryciu przed użytkownikiem złożoności sprzętowej komputera i dostarczenie mu wygodnego interfejsu w postaci wywołań systemowych. Druga z funkcji oznacza, że system ma za zadanie udostępnić użytkownikowi wszelkie zasoby komputera i jednocześnie chronić je w przypadku próby jednoczesnego lub nieuprawnionego dostępu do nich.

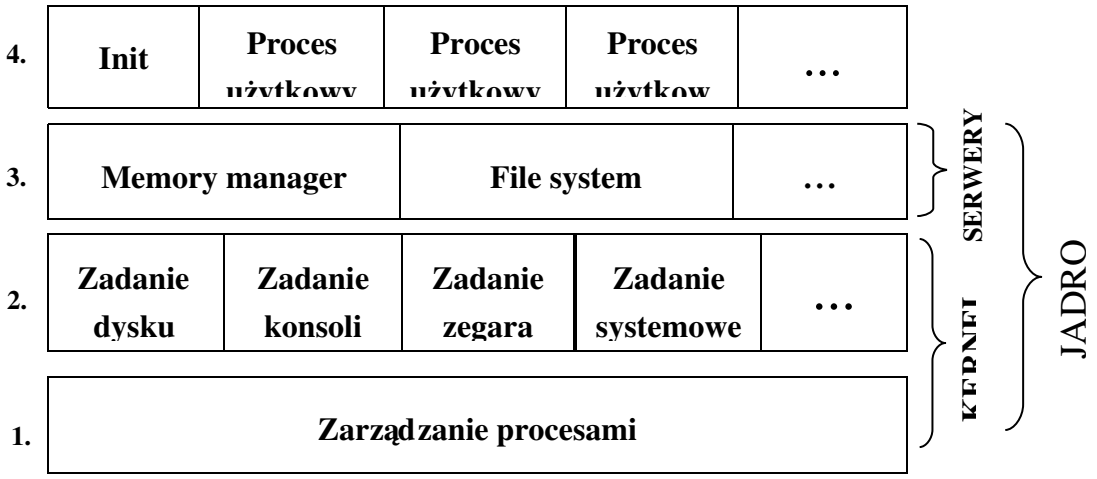
Pierwotnie na kursach systemów operacyjnych wykorzystywano źródła systemu Unix. Gdy firma AT&T wypuściła Version 7 Uniksa był on wartościowym produktem komercyjnym. W celu ochrony tajemnic handlowych firma zastrzegła w umowach licencyjnych zakaz używania źródeł systemu na kursach edukacyjnych. Od tego czasu uczelnie ograniczały się do wykładania teorii, w której było mało o tak istotnych w praktyce częściach systemu jak urządzenia wejścia / wyjścia czy system plików. Aby zmienić tą sytuację Andrew Tanenbaum napisał własny system operacyjny mini-UNIX czyli Minix. Pierwsza wersja była zgodna z Version 7 Umiksa, następne ze standardem POSIX. Aby nie naruszać praw licencyjnych Minix został napisany bez wykorzystania źródeł Uniksa. Jego budowa jest bardziej modularna a kod źródłowy bardziej czytelny, gdyż system od początku był projektowany do celów edukacyjnych. Minix ujrzał światło dzienne w 1980 roku. Początkowo był dostępny na platformie IBM PC. Wkrótce pojawiły się wersje na Atari, Amigę, Macintosh'a i SPARC'a. Warto wspomnieć, że w wyniku dyskusji jaka rozgorzała wokół Miniksa fiński student Linus Torvalds postanowił napisać na bazie Miniksa własny system, który rozwinął się jako dobrze dziś znany Linux.

Minix 2.0 został ukończony w 1996 roku, jego autorami są Andrew Tanenbaum i Albert S.Woodhull. W odróżnieniu od swego poprzednika nowy Minix jest systemem 32-bitowym, pracującym w trybie chronionym procesorów 80x386 i wyższych. Praca w tym trybie zapewnia przestrzeń adresową 4GB i sprzętową ochronę pamięci. Nie stosuje się stronicowania ani wymiatania procesów na dysk. Przesyłanie wiadomości oparte jest na tym samym mechanizmie randez-vous. System został przystosowany do pracy na twardym dysku i w środowisku sieciowym. Aby móc kompilować wszystkie źródła system potrzebuje 30 MB miejsca na twardym dysku. Poprzednia wersja wymaga zaledwie 8MB, lecz nie zawiera źródeł do wszystkich komend, ma mniej obszerne strony dokumentacji i posiada mniejszy zasób programów użytkowych. Wywołania systemowe są zgodne ze standardem

POSIX(prawie). Inne zasoby : kompilator ANSI C, shell zgodny z shellem Bourne'a, obsługa sieci TCP/IP, edytory emacs, vi, ex, ed, mined, ponad 200 programów (cat, cp, ps, grep, kermit, ls, make, sort, idt.), ponad 300 procedur bibliotecznych (atoi, fork, malloc, read, stdio, itd.)

2. Warstwy systemu Minix, procesy

W odróżnieniu od systemu Unix jądro Miniksa nie jest monolitycznym programem lecz zbiorem co najmniej trzech modułów – niezależnie kompilowanych programów łączonych w jądro. Niezbędne moduły to KERNEL – zarządzanie procesami, sterowniki **I/O,** MM (od Memory Manager) – zarządzanie pamięcią i FS (od File System) – zarządzanie system plików. Dodatkowo procedury modułu KERNEL obsługujące urządzenia wejścia / wyjścia uruchamiane są jako niezależne procesy nazwane zadaniami. Rysunek 1.1 pokazuje cztery warstwy jakie tworzą wszystkie procesy w działającym systemie. Pierwszą stanowi kod modułu KERNEL obsługujący zarządzanie procesami tzn. szeregowanie procesów, przełączanie procesów (zapamiętywanie i odtwarzanie kontekstu procesów), przesyłanie wiadomości między procesami (zawieszanie, odwieszanie, kopiowaniem danych między buforem nadawcy a buforem odbiorcy), dodatkowo procedury obsługi przerwań i kod realizujący przełączanie między trybem rzeczywistym a wirtualnym procesora.



Rys 1.1 Warstwy systemu Minix

Drugą warstwę tworzą procesy sterowników urządzeń wejścia/wyjścia a trzecią serwery. Wszystkie te procesy działają w typowej dla architektury klient-serwer pętli programowej :

```
while ( TRUE )
{
        receive(ANY,&msg);  /* odbierz wiadomość / zlecenie */
```

```
        caller = msg.source;
        switch(msg->type){
                ...                 /* wykonaj pracę */
        }
        send(caller,&msg)       /* wyślij odpowiedź */
}
```

System operacyjny ma z definicji dwa zadania : zarządzanie zasobami i tworzenie maszyny wirtualnej poprzez implementację wywołań systemowych. W SO Minix można przyjąć, że to pierwsze zadanie wykonuje moduł KERNEL, natomiast wywołania systemowe obsługują serwery.

Podział systemu na wiele modułów wymaga co prawda rozproszenia systemowej tablicy procesów (każdy moduł ma swoją, gdzie przechowuje potrzebne mu dane) ale ułatwia modyfikacje (każdy moduł może być modyfikowany i kompilowany oddzielnie), analizę i testowanie.

Dla procesorów 286 i wyższych każdy proces może działać na jednym z czterech poziomów uprzywilejowania. W SO Minix najwyższy poziom uprzywilejowania ma warstwa pierwsza czyli kod jądra i procedury obsługi przerwań i sytuacji wyjątkowych. Tak więc „prawdziwy" kod jądra może odwoływać się do całego obszaru pamięci i wszystkich rejestrów procesora. Procesy z warstwy zadań mimo że ich kod jest częścią tego samego modułu KERNEL mają niższy poziom uprzywilejowania – nie mają dostępu do wszystkich rejestrów procesora i całej pamięci, nie mogą również wykonywać pewnych instrukcji procesora, mają natomiast prawo dostępu do pamięci procesów mniej uprzywilejowanych oraz do portów. Procesy w warstwach 3 i 4 pracują w najmniej uprzywilejowanym trybie procesora. Mają dostęp tylko do przydzielonych im segmentów pamięci. Procesy warstwy 3 czyli serwery pracują co prawda jak zwykłe procesy użytkowe ale istotnie się od nich różnią. Po pierwsze są uruchamiane przy starcie systemu i działają nieprzerwanie aż do zamknięcia. Po drugie są traktowane specjalnie, mają ustalone, uprzywilejowane miejsce w tablicy procesów i mogą komunikować się z zadaniami wykorzystując przy tym wszystkie trzy dostępne w systemie funkcje do przesyłanie wiadomości: send, receive, sendrec. Zwykłe procesy mogą komunikować się tylko z serwerami i tylko przy pomocy funkcji sendrec. Do wzajemnej komunikacji procesów użytkowych autorzy przewidzieli jedynie potoki. Odblokowanie funkcji send i receive dla procesów użytkowych (do komunikacji między sobą, do zaimplementowania mechanizmu producent-konsument) jest jednym z zadań które wykonałem w ramach tej pracy, szczegóły znajdują się w rozdziale 4.6.

Warstwę 4 tworzą procesy użytkowe w tym pierwszy z nich **init**. Przegląda on plik /etc/ttytab i dla każdej znalezionej tam konsoli uruchamia program **getty**, który oczekuje na wpisanie nazwy i hasła użytkownika. Jeśli dane te są poprawne uruchamiany jest program powłoki **sh**. Typowo w systemie działa w tle jeszcze kilka procesów użytkowych np.: cron, update.

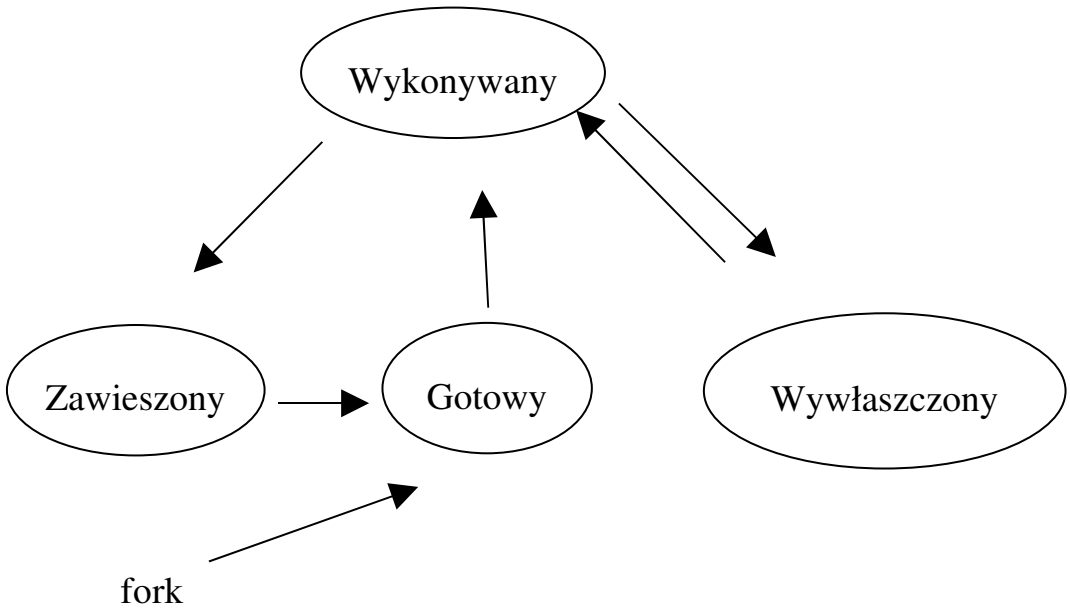
Do utworzenia nowego procesu w SO Minix służy funkcja **fork**. Gdy program jest uruchamiany przydzielana jest mu porcja pamięci, której rozmiar podany jest w nagłówku programu. Gdy proces się tworzy lub kończy najpierw moduł MM obsługuje wywołanie systemowe (FORK lub EXIT), uaktualnia swoją tablicę procesów, po czym wysyła wiadomość do modułów FS i KERNEL, które uaktualniają swoje tablice procesów.

3. Stany i szeregowanie procesów

Proces to program, który jest wykonywany. Z procesem związana jest przestrzeń adresowa - fragment lub zbiór fragmentów pamięci do których proces może pisać/czytać. Przestrzeń adresowa zawiera kod programu, dane programu oraz jego stos. Ponadto z procesem związane są wartości rejestrów procesora (np. licznik instrukcji który wskazuje ostatnio wykonywaną instrukcję).

W systemie wielozadaniowym jakim jest Minix co pewien czas zdarza się, że obecnie wykonywany proces jest czasowo zawieszany. Wymaga to zapamiętania kontekstu procesu w systemowej tablicy procesów, tak by można było wznowić proces w stanie z przed zawieszenia. Ze względu na podział jądra SO Minix na moduły, tablica procesów jest rozproszona. Każdy moduł posiada swoją tablicę procesów a w niej dane, które są mu potrzebne.

Rys 1.1 pokazuje ogólny graf stanów procesów. Nie wszystkie procesy w systemie mogą znaleźć się w każdym ze stanów. W szczególności zadania systemowe nie mogą przejść w stanie wywłaszczony (nie licząc przerwań) ponieważ wykonywane są dotąd, aż same się zawieszą.



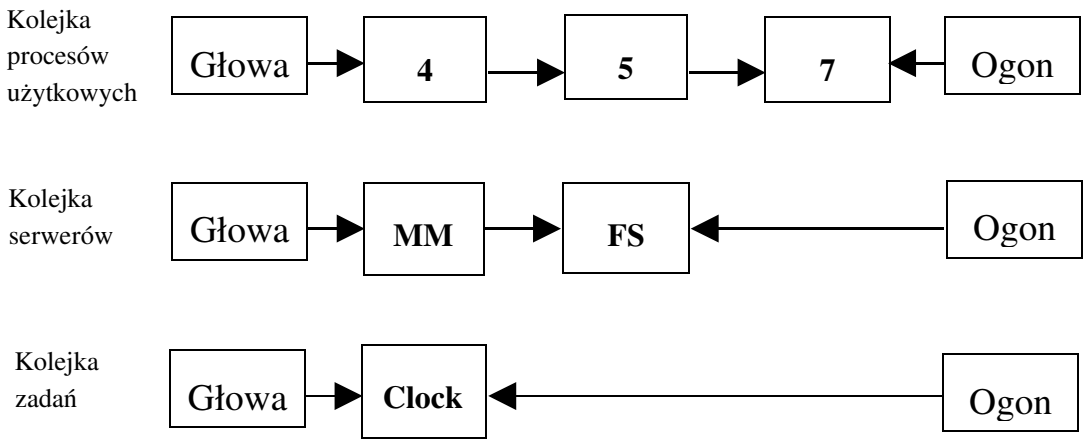
Rys 1.2 Graf stanów procesu w systemie operacyjnym Minix.

Ze stanu gotowy został wyróżniony stan wywłaszczony dla odróżnienia sytuacji, w której procesowi odbierany jest czas procesora od takiej, w której proces przechodzi w stan gotowości po uprzednim zawieszeniu w oczekiwaniu na zdarzenie zewnętrzne np.: dotarcie wiadomości.

Do szeregowania procesów wykorzystuje się trzy kolejki priorytetowe (rys 1.3). W pierwszej szeregowane są zadania systemowe. Spośród zadań gotowych do wykonania wybierane jest pierwsze w kolejce. Zadanie nie może być wywłaszczone przez żaden inny proces nie licząc przerwań sprzętowych. W szczególności zadanie nie może być wywłaszczone przez inne zadanie. Takie rozwiązanie jest całkowicie uzasadnione, gdyż zadania wykonują swoją pracę szybko. Trzeba założyć, że jako część jądra nie zawierają błędów i nie zawładną procesorem na wieczność.

W drugiej kolejce, o niższym priorytecie, na wykonanie oczekują serwery. Nie mogą one wywłaszczać siebie na wzajem ale mogą być wywłaszczone przez zadania.

W trzeciej kolejce na wykonanie oczekują procesy o najniższym priorytecie czyli procesy użytkowe. Mogą być wywłaszczane przez procesy o wyższym priorytecie: zadania i serwery. Każdemu procesowi użytkowemu przydzielany jest kwant czasu, standardowo 100 ms. Po upływie tego czasu wybierany jest następny proces według algorytmu karuzelowego (Round Robin). Podany kwant czasu nie zawsze jest przestrzegany. Każdy proces użytkowy po wywołaniu funkcji systemowej zawsze wędruje na koniec kolejki, niezależnie ile czasu wykorzystał z przydzielonego mu kwantu.



Rys 1.3 Kolejki procesów o różnych priorytetach.