Wykład pierwszy

„Rodowód" Linuksa

Inspiracją dla autora systemu Linux był system Minix napisany przez Andrew Tanenbauma, który bazował z kolei na systemie Unix. Termin Unix w obecnych czasach oznacza całą rodzinę systemów operacyjnych opartych na wspólnej filozofii, której źródłem jest projekt systemu operacyjnego, jaki powstał w roku 1969 w Bell Labs, będących częścią amerykańskiego koncernu AT&T. System ten jest pochodną innego systemu operacyjnego Multics, nad którego powstaniem pracowano w trzech ośrodkach: w MIT, w AT&T i w General Electric. Firma AT&T po pewnym czasie wycofała się z prac, ale osoby zatrudnione przy tym projekcie pozostały jej pracownikami. Do ich grona należeli między innymi Ken Thompson, Dennis Ritchie i Douglas McIlroy. To ta trójka odegrała największą rolę w powstaniu pierwszej wersji systemu Unix. Pracę nad nią rozpoczęli Ken Thompson i Dennis Ritchie, pisząc cały kod w assemblerze komputera PDP-7, który poznali tworząc wcześniej grę „Space Travel" dla tej platformy sprzętowej. Znaczący wkład do projektu systemu wniósł Douglas McIlroy, opracowując np.: łącza nienazwane, które są jednym z środków zapewniających komunikację między procesami, a także wyznaczając reguły tworzenia kodu, które dziś są częścią inżynierii oprogramowania. W roku 1973 kod źródłowy Uniksa został w większości przepisany w języku C, który opracował Dennis Ritchie [1], dzięki temu system ten stał się łatwy w przenoszeniu na inne platformy sprzętowe[2]. Firma AT&T podjęła decyzję o rozpoczęciu sprzedaży systemu Unix. Każda jego wersja była rozprowadzana razem z kodem źródłowym, a więc wiele firm i innych instytucji, które nabyło ten system, mogło go modyfikować wprowadzając nowe funkcje. Pod względem wprowadzonych do tego systemu innowacji wyróżniał się Uniwersytet Berkeley. Wszystkie wersje systemu Unix, które powstały w tej placówce były oznaczane nazwą BSD (Berkeley System Distribution). Największy wkład do prac nad Uniksem BSD wniósł Bill Joy. To w Berkeley powstał słynny edytor vi, powłoka csh, a także tu dodano do jądra podsystemy odpowiedzialne za obsługę protokołu TCP/IP i pamięć wirtualną. Z czasem te funkcje zostały włączone do dystrybucji rozpowszechnianej przez AT&T (wersja System V). W chwili obecnej rozwój gałęzi BSD jest kontynuowany przez takie projekty, jak FreeBSD, OpenBSD, NetBSD i DragonFly BSD. Aby uporządkować rozwój Uniksa opracowano kilka standardów, z których najważniejszymi są POSIX oraz SUS. Większość odmian Uniksa spełnia wymagania tych standardów.
W chwili obecnej niemalże wszystkie odmiany Uniksa to systemy wielodostępne i wielozadaniowe, obsługujące wszystkie rozwiązania w nowoczesnych platformach sprzętowych. Jest jednakże kilka cech, które odróżniają go od innych systemów operacyjnych. Przede wszystkim jest on systemem o stosunkowo prostej budowie (jego twórcy stosowali zasadę Keep It Small and Simple – KISS[3]). Jądro Uniksa implementuje niewielką liczbę wywołań systemowych, podczas gdy w innych systemach ta liczba dochodzi do dziesiątek tysięcy. Wywołania systemowe w Uniksie zostały opracowane w myśl zasady sformułowanej przez Douglasa McIlroy'a: „Rób jedną rzecz, ale rób ją dobrze", co oznacza, że wykonują jedną czynność, ale w sposób uniwersalny. Większość elementów w systemie Unix jest traktowana jako plik, co znacznie ułatwia zarządzanie urządzeniami i danymi. W końcu Unix ma mechanizmy, które pozwalają na tworzenie w krótkim czasie nowych procesów, oraz dostarcza środków prostej komunikacji między nimi. Unix był także jednym z pierwszych systemów operacyjnych, które zostały napisane w języku wysokiego poziomu, co uczyniło go systemem przenośnym.

Charakterystyka Linuksa

Linux[4] jest systemem uniksopodobnym (ang. Unix-like), ale nie jest Uniksem. Pracę nad jądrem tego systemu rozpoczął Linus Benedict Torvalds w roku 1991, wzorując się na wspomnianym systemie Minix. Nie oznacza to jednak, że kod przez niego napisany zawiera fragmenty kodu Miniksa lub oryginalnego Uniksa. Linux powstał „od zera", ale jest zgodny ze standardami POSIX i SUS. Pierwsza wersja kodu źródłowego Linuksa miała około 10 tysięcy wierszy kodu. Bieżące wersje liczą ponad 20 milionów linii kodu i są tworzone przez obszerną grupę programistów, współpracujących ze sobą przez Internet. Prace tej grupy koordynuje oczywiście Linus Torvalds. Kod źródłowy Linuksa dostępny jest na zasadach licencji GNU GPL v2.0, która gwarantuje, że jest on wolnodostępny. Należy zauważyć, że słowo Linux ma dwa znaczenia. W powyższym tekście oznaczało ono po prostu jądro systemu operacyjnego. W drugim znaczeniu, oznacza podstawową wersję systemu operacyjnego, która oprócz jądra zawiera również zestaw narzędzi do kompilacji programów w językach C i assembler, bibliotekę języka C i powłokę systemową. Wszystkie wymienione tu elementy, oprócz jądra, zostały stworzone przez fundację GNU, założoną przez Richarda Stallmana. Istnieje wiele odmian systemu Linux, nazywanych dystrybucjami, które oprócz wymienionych tu podstawowych narzędzi zawierają również inne oprogramowanie np. system X Window, będący implementacją środowiska graficznego użytkownika, oraz szereg innych aplikacji, niezbędnych użytkownikom. W dalszej części wykładu słowo „Linux" będzie oznaczało wyłącznie jądro systemu operacyjnego. Pierwotnie Linux przeznaczony był tylko na platformę sprzętową i386 (powstał na komputerze wyposażonym w procesor Intel 80386). Obecnie obsługuje on całą gamę mniej lub bardziej popularnych platform sprzętowych, takich jak : ARM, AMD64, PowerPC, Ultra Sparc, MIPS. Wspiera również architektury równoległe (SMP, NUMA).

---

1 Pozostałą część pozostawiono w assemblerze.
2 Dokładniej – łatwiej było go przenieść na inne platformy, niż system napisany w całości w assemblerze.
3 Właściwie ten skrót rozwijano jako "Keep It Simple, Stupid!". Osobiście wolę wersję łagodniejszą :-)
4 W kwestii językowej: piszemy Linux, ale odmieniamy: Linuksa, Linuksowi, itd.

Zasada działania jądra

Jądro (*ang. kernel*) systemu (nazywane czasami rdzeniem (*ang. core*)) jest częścią systemu operacyjnego, która stale rezyduje w pamięci komputera, nadzoruje pracę sprzętu i aplikacji użytkownika, oraz dostarcza tym ostatnim określonych usługi. Najczęściej jądro (monolityczne) zawiera takie elementy, jak: procedury obsługi przerwań, mechanizm szeregowania procesów, mechanizm zarządzania pamięcią i obsługi plików. Jądro pracuje w trybie uprzywilejowanym procesora, co oznacza, że ma nieograniczony dostęp do wszystkich zasobów systemu komputerowego, w tym do pamięci operacyjnej. Obszar pamięci (w sensie zakresu adresów) zajmowany przez jądro nazywany jest *przestrzenią adresową jądra*. Dosyć często o czynnościach wykonywanych przez jądro mówimy, że zachodzą w *przestrzeni jądra*. Procesy użytkownika pracują w nieuprzywilejowanym trybie procesora, w którym dostęp do zasobów systemu jest ograniczony. Obszary pamięci (również w sensie zakresu adresów) przyporządkowane aplikacjom użytkownika nazywamy *przestrzenią adresową użytkownika*. Analogicznie jak ma to miejsce w przypadku jądra, o czynnościach, które są wykonywane przez aplikacje mówimy, że są wykonywane w *przestrzeni użytkownika*. Do komunikacji między procesami użytkownika, a jądrem służą *wywołania systemowe*. Zazwyczaj aplikacje użytkownika nie wywołują ich bezpośrednio, lecz korzystają z funkcji dostępnych w standardowej bibliotece języka C (*libc*). Niektóre z tych funkcji zawierają sam kod wywołania funkcji systemowej, czyli są swego rodzaju „opakowaniami" na wywołania systemowe, inne funkcje wykonują przed zainicjowaniem wywołania systemowego dodatkowe czynności, jeszcze inne mogą korzystać z większej liczby wywołań systemowych. Są również funkcje, które nie korzystają w ogóle z wywołań systemowych. Jeśli jądro wykonuje za pośrednictwem wywołania systemowego jakąś usługę dla aplikacji użytkownika, to działa w *kontekście procesu*, co oznacza, że kod jądra realizujący to wywołanie ma dostęp do informacji o procesie, który korzysta z tego wywołania. W takiej sytuacji można również powiedzieć, że aplikacja wykonuje wywołanie systemowe w przestrzeni jądra, choć nie jest to określenie poprawne. Jądro systemu jest oprogramowaniem sterowanym zdarzeniami. Zdarzenia pochodzące od sprzętu są sygnalizowane za pomocą *przerwań*. Te przerwania pojawiają się zazwyczaj asynchronicznie[5]. Z każdym z nich jest związany pewien numer, na podstawie którego jądro odnajduje i wykonuje odpowiednią procedurę obsługi tego przerwania. Te procedury są wykonywane w *kontekście przerwania*, co oznacza, że nie są związane z żadnym konkretnym procesem. Pozwala to przyspieszyć ich działanie. Jądro ma również możliwość blokowania wszystkich przerwań lub tylko wybranych przerwań, na czas obsługi jednego z nich. Reasumując, procesor w systemie komputerowym, który kontroluje Linux, albo wykonuje kod aplikacji, albo kod jądra w kontekście procesu, albo kod jądra w kontekście przerwania.

Porównanie z innymi systemami uniksowymi

Jądro sytemu Linux jest podobne do rdzeni innych systemów wzorowanych na Uniksie, ale istnieje również kilka znaczących różnic. Podobnie jak w większości Uniksów, jądro Linuksa jest monolityczne, ale udostępnia możliwość korzystania z ładowanych dynamicznie modułów, które zazwyczaj są sterownikami urządzeń. Linux obsługuje również symetryczne przetwarzanie wieloprocesorowe (SMP) oraz architekturę NUMA, co nie jest cechą wszystkich Uniksów. Podobnie jak Solaris i IRIX, Linux może obsługiwać od wersji 2.6 wywłaszczanie zadań jądra. Jądro Linuksa nie rozróżnia wątków i procesów, traktuje je niemalże w ten sam sposób. Niektóre mechanizmy, które przez twórców Uniksa zostały uznane za mało wydajne lub wręcz za zbędne (jak np. obsługa strumieni) nie zostały w ogóle zaimplementowane w jądrze Linuksa. Ze względów wydajnościowych strony pamięci zajmowanej przez jądro Linuksa nie podlegają wymianie. Ta ostatnia cecha jest konsekwencją faktu, że Linux jest tworzony przez ogromną społeczność programistów – w wyniku dyskusji i rozważań członkowie tej społeczność doszli do wniosku, że taki mechanizm spowodowałby pogorszenie wydajności systemu. Kolejne wydania jądra są oznaczane symbolami składającymi się z trzech[6] liczb rozdzielonych kropkami. Pierwsza liczba to główny numer wersji, druga to numer główny rewizji, trzecia to numer poboczny rewizji i oznacza wydanie, w którym dokonano poprawek znalezionych defektów. Numer główny rewizji oznacza wydanie, w którym dokonano znaczących zmian lub rozbudowano funkcjonalność jądra. Przed wydaniem jądra 2.6.0 nieparzysty główny numer rewizji oznaczał wydanie rozwojowe, które mogło być niestabilne, a główny numer parzysty rewizji, wydanie stabilne. Od wersji 3.0 jądra numer główny rewizji oznacza kolejne wydanie, a trzecia liczba służy do oznaczania wydań, w których usunięto wykryte defekty. Wydania rozwojowe oznaczane są za pomocą liter *rc*, za którymi występuje numer wersji rozwojowej. Niektóre dystrybucje Linuksa mogą dodawać do tych oznaczeń dodatkowe liczby.

Krótka charakterystyka metodologii programowania jądra

Ponieważ jądro systemu z definicji ma być wydajne pod względem wykorzystania pamięci i szybkości działania, to przeglądając kod źródłowy Linuksa możemy znaleźć wiele fragmentów, które łamią niektóre przyjęte kanony dobrze napisanego oprogramowania (jak choćby nieużywanie instrukcji *goto*). Tak, jak w przypadku innych systemów uniksopodobnych, większość kodu Linuksa jest napisana w języku C, a tylko niewielka część, zależna od konkretnej architektu-

5 Istnieją również przerwania synchroniczne.

6 Pojawiła się również czwarta liczba (jądro 2.6.8.1). Jej wprowadzenie związane było z naprawieniem poważnego błędu, który wykryto tuż po wypuszczeniu wersji 2.6.8. Tę numerację stosowano czasem w późniejszych wersjach.

ry sprzętowej, w assemblerze[7]. Pisząc własny fragment jądra, czy to w postaci modułu, czy ingerując w istniejący kod należy mieć świadomość pewnych ograniczeń. Przede wszystkim, w kodzie jądra nie mamy dostępu do funkcji standardowej biblioteki C (libc, a konkretniej glibc). Jest to naturalną konsekwencją tego, że to jądro stanowi wsparcie dla tej biblioteki, a nie odwrotnie. Na szczęście zostały zaimplementowane w jądrze funkcje będące substytutami tych z biblioteki glibc. Zamiast funkcji *printf()* jest funkcja *printk()*, zamiast funkcji do manipulowania łańcuchami znaków, które są dostępne po włączeniu pliku *string.h* są podobne funkcje dostępne po włączeniu pliku *linux/string.h*. Uogólniając – jądro może korzystać wyłącznie ze swoich plików nagłówkowych, niedozwolone jest korzystanie z zewnętrznych plików nagłówkowych. Kod jądra jest pisany z zgodnie ze standardem ISO C99 języka C oraz z wykorzystaniem rozszerzeń GNU C. Oznacza to, że może być kompilowany prawie wyłącznie przez kompilator C pochodzący z GCC (GNU Compilers Collection). Do wspomnianych rozszerzeń należy wykorzystywanie funkcji rozwijanych w miejscu wywołania (ang. *inline functions*), obecnie wprowadzone już do standardu języka C. Takie funkcje są bardziej bezpieczne i eleganckie niż makra, a przy tym równie wydajne. Funkcje rozwijane w miejscu wywołania są definiowane z użyciem słów kluczowych *static* i *inline*. Należy jednak pamiętać, aby ich nie nadużywać, bo prowadzi to do niepotrzebnego zwiększenia objętości kodu wynikowego. Innym z wykorzystywanych rozszerzeń jest możliwość wprowadzania specjalnych wstawek assemblerowych do kodu napisanego w języku C. Te wstawki są stosowane wszędzie tam, gdzie wymagana jest optymalizacja szybkości działania jądra. Kompilator gcc udostępnia dwie dyrektywy wspomagające optymalizację kodu. Są to *likely()* i *unlikely()*. Służą one do oznaczania prawdopodobieństwa wykonania kodu w instrukcjach warunkowych, np.: jako *unlikely()* możemy zaznaczyć warunki związane z obsługą większości wyjątków. Pisząc kod działający w przestrzeni jądra musimy pamiętać, że nie mamy „siatki zabezpieczającej" w postaci ochrony pamięci. Jeśli będziemy źle zarządzać pamięcią jądra, to możemy naruszyć stabilność systemu operacyjnego. Jeśli chcemy użyć liczb i arytmetyki zmiennoprzecinkowej, to musimy sami oprogramować FPU, jednakże nigdy dotąd praktyczna potrzeba używania takiej arytmetyki w przestrzeni jądra nie zaistniała. Rozmiar stosu jądra[8] jest ograniczony i wynosi dwie strony, czyli w przypadku 32-bitowych procesorów o architekturze x86, 8 KiB[9], a w przypadku 64 bitowych procesorów Alpha 16 KiB. Procesy użytkownika dysponują stosem znajdującym się w przestrzeni użytkownika, który nie jest ograniczony[10]. Programując w jądrze musimy pamiętać o synchronizacji dostępu do zasobów współdzielonych. Mamy do dyspozycji np. takie środki synchronizacji jak semafory i rygle pętlowe (*ang. spin-lock)*, czyli semafory z aktywnym oczekiwaniem. Ostatnią ważną rzeczą, o której należy pamiętać pisząc lub zmieniając kod jądra, to przenośność. Nawet pisząc kod w języku C możemy doświadczyć różnych problemów, jak choćby porządek bitów i bajtów w słowie, czy rozmiary poszczególnych typów danych.

7 Dodajmy: w notacji AT&T.

8 Określenia co najmniej mylące. Chodzi o stos jądra, który jest przydzielany każdemu procesowi użytkownika i używany jeśli ten proces zainicjuje wywołanie systemowe.

9 Istnieje możliwość zmniejszenia tego rozmiaru do 4 KiB.

10 Dokładniej: większość dystrybucji Linuksa ograniczają rozmiar stosu do 8MiB, ale użytkownik uprzywilejowany może znieść te ograniczenia.