Implementacja podsystemów Linuksa

* systemy plików
* planowanie procesów i wątków
* budowa jądra i sterowników

# Wstęp

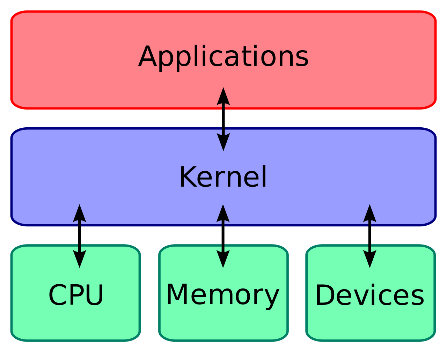
System Linux składa się z wielu współgrających komponentów, niektóre z nich wyróżniają się bardziej od innych tym jak bardzo są niezbędne oraz charakterystyczne, dla danego systemu.

Omówię 3 z nich:

* Kernel
* File system(y)
* Planowanie procesów i wątków

# Kernel

Pierwszym komponentem, który omówię jest Kernel (czyli jądro). Jest on komponentem leżącym w samym sercu systemu.

[](https://www.google.com/url?sa=i&url=https%3A%2F%2Fblog.digilentinc.com%2Fdemystifiying-the-linux-kernel%2F&psig=AOvVaw0CgdeTjfGPaGMiB3aMUW2u&ust=1606613340937000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCPiS2J6LpO0CFQAAAAAdAAAAABAD)

Graficznie Kernel jest zwykle pokazywany w środku jako, że wykonuje bardzo ważną funkcję niejako komunikowania strony hardware’owej komputera ze stroną aplikacji, w której jako użytkownicy czy nawet programiści często poruszamy się nawet bez wiedzy na temat hardware’u, którego używa nasz komputer, lub komputera na, który będzie przeznaczona nasza aplikacja.

# Historia Kernela

Jako datę powstania jądra linuxa przyjmuje się 25 sierpnia 1991, czyli moment, w którym Linus Torvalds wysłał post na grupę dyskusyjną comp.os.minix rozpoczynający się słowami:

„Właśnie piszę (wolny) system operacyjny (dla zabawy, nie będzie wielki i profesjonalny jak gnu) …”

W owym czasie projekt GNU stworzył wiele komponentów niezbędnych dla wolnego systemu operacyjnego, ale jądro GNU Hurd nie było jeszcze gotowe. Z kolei projekt BSD jeszcze nie udostępnił swojego systemu operacyjnego z uwagi na problemy prawne. Dzięki temu pojawiła się luka w świecie Kerneli, którą mogło zapełnić jądro Linux. Mimo ograniczonej funkcjonalności wczesnych wersji, projekt błyskawicznie przyciągnął pierwszych programistów i użytkowników. Początkowo rozwojem i dostarczaniem pomysłów zajmowali się ludzie pracujący nad Minixem; dzisiaj swój kod udostępnia tysiące programistów. W październiku 2008 długość kodu jądra Linux przekroczyła 10 mln wierszy.

Pierwotnie nazwa „Linux” dotyczyła tylko jądra, które stworzył Torvalds. Jądro nie jest kompletnym systemem operacyjnym (przynajmniej w typowym rozumieniu tego pojęcia). Pełny system zbudowany z użyciem jądra Linux najczęściej nazywany jest systemem operacyjnym Linux, chociaż niektórzy wolą nazywać go GNU/Linux, gdyż projekt GNU stworzył bardzo wiele programów, które zwykle są instalowane wraz z jądrem Linux w ramach tzw. dystrybucji Linuksa. System operacyjny jest często mylony z samym jądrem, co prowadzi do nieprawdziwych wniosków, np. że Linus Torvalds pisze lub koordynuje inne części systemu poza jądrem, bądź prowadzi rozwój jakiś konkretnych dystrybucji linuxa. Gdzie w rzeczywistości zajmuje się on tylko jądrem Linux.

# Za co odpowiada Kernel?

Jądro systemu - inaczej kernel; jest to ta część oprogramowania, która odpowiada za:

* komunikację systemu z urządzeniami (sterowniki, które w Linuksie nazywane są modułami)
* uruchamianie i komunikację pomiędzy programami
* obsługę systemów plików - ma bezpośredni dostęp do dysków, i plików
* stworzenie środowiska uruchomieniowego dla programów (np. gdy program rezerwuje potrzebną ilość pamięci, to właśnie kernel przyznaje mu taki rejon, który nie jest przez żadne inne programy używany).

# Licencje Kernela Linuxa

Ważnym tematem związanym z Linuxem jest to jak bardzo „wolny” on jest. Gdzie słowo „wolny” (oryginałem będące słowo „free” z angielskiego) oznacza tutaj zarazem darmowość kernela pod względem ceny oraz to, że jego kod jest otwarty i każdy może go zobaczyć, zapożyczyć i zmodyfikować (w zamian muszą jedynie opublikować zmiany jakich dokonali).

Początkowo Torvalds udostępnił Linuksa na licencji zabraniającej użycia komercyjnego. Szybko jednak została ona zmieniona na GPL. Licencja zezwala na rozpowszechnianie i sprzedaż oryginalnych bądź zmodyfikowanych kopii jądra, ale wymaga zachowania tej licencji dla kolejnych kopii oraz udostępnienia kodu źródłowego.

Linus Torvalds określił wybór GPL jako licencji dla jądra Linux jako „najlepszą rzecz jaką kiedykolwiek zrobiłem”.

Jedno z najczęstszych pytań dotyczących licencji GPL dla Linuksa dotyczy kwestii traktowania modułów jądra w prawie autorskim jako „dzieł pochodnych”, które w myśl licencji GPL powinny być objęte tą samą licencją. Torvalds przedstawił swoje zdanie, że programy które używają jedynie ograniczonego, „publicznego” zestawu interfejsów jądra nie muszą być „dziełami pochodnymi”, zezwalając na tworzenie niezgodnych z GPL, dostępnych jedynie w binarnej (wykonywalnej) formie sterowników i innych modułów. Nie wszyscy twórcy jądra zgadzają się z tą interpretacją, niemniej nawet sam Torvalds zaznacza, że wiele modułów należy uznać za dzieła pochodne. Uważa również, że „moduły są domyślnie dziełami pochodnymi”; ostatecznie wyjaśnienie tej kwestii może nastąpić na drodze prawnej.

# Kernel Linuxa vs Windowsa

Windows 7, Windows Vista, Windows Server 2003, Windows XP, itp. są częścią rodziny Windows NT systemów operacyjnych firmy Microsoft. Posiadją one Kernel NT. Umożliwia on między innymi wywłaszczanie oraz wielobieżność i został zaprojektowany do pracy na systemach bazujących na architekturze x86 Intela, zarówno jednoprocesorowych, jak i o architekturze SMP. Do obsługi wejścia/wyjścia architektura NT stosuje przetwarzanie pakietowe, które wykorzystuje IRP (I/O request packet - pakiet zamówienia wejścia/wyjścia), oraz wejście/wyjście asynchroniczne. Poczynając od Windows XP, Microsoft rozpoczął wbudowywanie obsługi trybu 64-bitowego w swoje systemy operacyjne — wcześniejsze wersje korzystały tylko z trybu 32-bitowego. Architektura Windows NT jest wysoce modularna i składa się z dwóch głównych warstw: trybu użytkownika i trybu jądra.

Jądra systemów operacyjnych w technologii NT są określane jako jądra hybrydowe, jednakże warto zauważyć, iż jest to sprawą dyskusyjną, gdyż jądro to jest właściwie jądrem monolitycznym o strukturze zbliżonej do mikrojądra. Linux używa jądra monolitycznego, które zajmuje więcej miejsca, podczas gdy system Windows używa mikro-jądra, które zajmuje mniej miejsca, ale obniża wydajność systemu.

W systemie Microsoft Windows pliki są przechowywane w katalogach / folderach na różnych dyskach danych, takich jak C: D: E: ale w Linuksie pliki są uporządkowane w strukturze drzewa, począwszy od katalogu głównego, a następnie są rozgałęzione na różne inne podkatalogi.

W Linuksie wszystko jest traktowane jak plik. Katalogi to pliki, pliki to pliki, a urządzenia zewnętrzne (takie jak drukarka, mysz, klawiatura) również są plikami, takie coś nie zachodzi w systemach windows.

# Kernel Linuxa vs Kernel XNU (MacOS)

Podstawowymi różnicami między Kernelem Linux, a kernelem macOS zwanym XNU, gdzie jego akronim rozwija się jako XNUs Not Unix, w bardzo podobny sposób jak GNU, czyli GNUs Not Unix. Jako że macOS jest systemem zgodnym z Unix-like standardami jak na przykład POSIX (podobnie jak Linux). Możemy go nazwać Unix-like systemem operacyjnym. Kernel macOS jest bazowany na dwóch starszych bazach kodu z BSD oraz Mach, obie będące UNIX-like systemami.

Linux i macOS ze względu na swoje UNIXowe podobieństwo są środowiskami gdzie użytkownicy jednej z tych platform będą się czuli w miarę znajomo na drugiej.

MacOS łączy w sobie funkcje microkerneli oraz kerneli monolitycznych. Linux jest kernelem strikte monolitycznym. Kernel monolityczny jest odpowiedzialny za CPU, pamięć, komunikację między procesową, sterowniki, file system i wywołania serwera systemowego. Za to w macOS część z tych rzeczy jest zaimplementowana w user-space systemu.

# Brief czym jest File system

System plików – jest metoda przechowywania plików, zarządzania plikami, informacjami o tych plikach, tak by dostęp do plików i danych w nich zgromadzonych był łatwy dla użytkownika systemu.

Systemy plików stosuje się dla różnych nośników danych, takich jak dyski, dyskietki, a także w strumieniach danych, sieciach komputerowych, pamięciach. We współczesnych systemach operacyjnych bezpośrednie operowanie na danych nośnika pamięci zarezerwowane jest tylko dla systemu operacyjnego, aplikacje mają dostęp tylko do operacji na plikach i mają zabroniony bezpośredni dostęp do nośnika danych.

Z formalnego punktu widzenia system plików to reguły umieszczania na nośniku abstrakcyjnych danych oraz informacji umożliwiających przechowywanie tych danych, łatwy i szybki dostęp do informacji o danych, manipulowania nimi, a także sposobach usuwania ich.

Większość systemów operacyjnych posiada własny (macierzysty) system plików, rozwijany równolegle z nim (np. FAT w DOS-ie, NTFS w Windows NT, APFS w macOS lub ext/ext2/ext3/ext4 i ReiserFS/Reiser4 w Linuksie), ze względu na pewne specyficzne właściwości nadawane plikom (np. atrybut wykonywalności pliku), podobnie jak niektóre nośniki danych, sam system plików jest niezależny od nich. Same systemy operacyjne (w szczególności Unix i jego pochodne) potrafią obsługiwać wiele systemów plików.

# File systemu najpopularniejszych dystrybucji linuxa

Popularne systemy plików linuxa: Minix, xia, Ext, Ext2, Ext3, Ext4, umsdos, msdos, ReiserFS, vfat, XFS, proc, JFS, smb, NFS, ncp, Swap, iso9660, Sysv, hpfs, Affs. Z dużą częścią z nich większość użytkowników się nie spotyka, chyba, że chcą bardziej skcustomizować swoją instalację linuxa jako, że te systemy plików nie występują zwykle na partycjach gdzie użytownik przechowuje swoje pliki, np. swap albo proc.

**Minix** - pierwszy system plików wykorzystywany w linuxie, który został stworzony jeszcze przed powstaniem linuxa dla systemu operacyjnego o nazwie MINIX.

Posiadał kilka ograniczeń:

* długość nazwy pliku maksymalnie 14 znaków, (po pewnym czasie zwiększono do 30 znaków)
* wielkość partycji nie większa niż 64 MB,

**Ext (EXTENDED FILE SYSTEM)** - następca Minix File System, dopuszczał pliki i partycje do rozmiaru 2 GB oraz długości nazw plików do 255 znaków. Miał jedną dużą wadę, która spowodowała, że nie stał się popularny: sposób zapamiętywania wolnych bloków i węzłów doprowadzał do znacznej fragmentacji dysku co wpływało negatywnie na wydajność systemu.

**Ext2** - zastąpił ext i w krótkim czasie stał się podstawowym systemem plików dla linuxa. W porównaniu do poprzednika ma wiele zalet:

* obsługuje partycje o wielkości do 4 TB, pliki o wielkości do 2 GB,
* rozpoznaje uszkodzenie systemu plików,
* automatycznie naprawia uszkodzone sektory,
* automatycznie sprawdza system po awarii i co określony czas, posiada wysoką wydajność dzięki przeciwdziałanie fragmentacji

Wady:

* długotrwałe sprawdzanie systemu plików po niepoprawnym zamknięciu,
* niska wydajność dla bardzo małych plików,
* mało efektywna obsługa katalogów.

**Ext3** - nowoczesny system plików oparty na ext2. Domyślny w większości dystrybucji systemu GNU/Linux opartych na jądrze 2.4.

**Właściwości:**

* Journaling - mechanizm księgowania zwiększający bezpieczeństwo systemu,
* Indeksowane katalogi - znacznie zwiększają wydajność systemu przy dużej ilości plików,
* Zapis synchroniczny - w najnowszych wersjach systemu Ext3 (jądro 2.4.19) działa ponad 10 razy szybciej od wersji z Ext2,
* Kompatybilność z Ext2.

**Ext4** - Rozwój systemu ext4 rozpoczął się w 2006 roku wraz z opracowaniem dwóch zmian dla ext3. Pierwsza z nich rozszerzyła wielkość numeru bloku do 48 bitów, druga zaś zastąpiła mechanizm pośredniego adresowania bloków przez tzw. extenty, a więc obszary bloków danych. Ponieważ po zastosowaniu wspomnianych nowości zmianie ulega struktura danych na dysku, programiści zadecydowali, że aktualizacje te nie zostaną wprowadzone do starego systemu ext3, lecz na bazie jego kodu zostanie opracowana nowa wersja systemu plików o nazwie ext4. System plików ext4 pracuje z 48-bitowymi numerami bloków przy standardowej wielkości bloku nadal wynoszącej 4 kilobajty. Dzięki temu wielkość systemu plików może wynosić do jednego eksabajta (1024 petabajty), zamiast jak w przypadku ext3 – 16 terabajtów.

# Planowanie procesów i wątków, czym jest i po co jest

Za planowanie procesów w linuxie odpowiada Scheduler.

Scheduler jest wykonywany w celu przydzielenia czasu procesora do różnych zadań w systemie operacyjnym. Ogólnie myślimy o planowaniu jako uruchamianiu i przerywaniu procesu, w przypadku Linuksa istnieje inny ważny aspekt planowania: taki jak uruchamianie jądra z wieloma zadaniami. Zadanie jądra obejmuje zadanie wymagane przez trwający proces, a zadania wykonywane wewnętrznie dotyczą odpowiedniego sterownika urządzenia.

Kiedy jądro osiągnie punkt zmiany scheduler, albo z powodu zmiany scheduler, albo przerwania, ponieważ uruchomiony proces jądra został zablokowany, aby czekać na obudzenie się sygnału, musi zdecydować, jaki następny proces ma zostać uruchomiony. Linux ma już dwa oddzielne algorytmy planowania procesów. Pierwszy algorytm to algorytm współdzielenia czasu dla sprawiedliwego planowania wyprzedzającego między wieloma procesami. Natomiast drugi algorytm jest przeznaczony do zadań w czasie rzeczywistym, w których absolutny priorytet jest ważniejszy niż uczciwość w uzyskaniu usługi.

Częścią tożsamości każdego procesu jest klasa planowania, która definiuje algorytm używany dla każdego procesu. Klasa planowania używana przez Linuksa jest oparta na standardzie rozszerzeń POSIX dla systemów komputerowych czasu rzeczywistego.

W procesie podziału czasu Linux używa technik priorytetowych, algorytmu opartego na kuponach. Każdy proces ma kilka kuponów planowania; gdzie kiedy jest możliwość wykonania zadania, wtedy kolej na proces z największą liczbą kuponów. Za każdym razem, gdy nastąpi przerwa, trwający proces spowoduje utratę jednego kuponu; a kiedy posiadany kupon się skończy, proces zostanie odroczony, a inne procesy będą miały możliwość wejścia.

Jeśli działający proces w ogóle nie ma kuponu, linux wykona operację kuponu, dając kupon każdemu procesowi w systemie, z regułami: kupon = kupon / 2 + priorytet. Algorytm ma tendencję do łączenia dwóch istniejących czynników: historia procesu i priorytet samego procesu. Połowa kuponów posiadanych od ostatniej operacji dystrybucji kuponów zostanie zachowana po uruchomieniu algorytmu, zachowując część historii nastawienia procesu. Proces, który trwa przez cały czas, będzie miał tendencję do szybkiego wydawania kuponów, ale więcej procesów oczekiwania może gromadzić kupony z. Ten system dystrybucji kuponów automatycznie nada wysoki priorytet procesom związanym z We / Wy lub procesom interaktywnym, w przypadku których wymagana jest szybka odpowiedź.

Przydatność procesu nadawania priorytetu w liczeniu nowych kuponów sprawia, że ​​priorytet procesu może ulec poprawie. Praca wsadowa w tle może mieć niski priorytet; proces automatycznie otrzyma mniej kuponów w porównaniu do pracy interaktywnej, a także otrzyma mniejszy procent czasu procesora w porównaniu do tego samego zadania o wyższym priorytecie. Linux używa tego systemu priorytetów do implementacji standardowego mechanizmu lepszego współdzielenia priorytetów procesów.

Planowanie w czasie rzeczywistym w Linuksie jest jeszcze prostsze. Linux, implementuje dwie klasy planowania czasu rzeczywistego wymagane przez POSIX 1.b: First In First Out i round-robin. W obu przypadkach każdy proces ma priorytet oprócz klasy planowania. Jednak w scheduler z podziałem czasu procesy o różnych priorytetach mogą konkurować z kilkoma dylematami; w planowaniu w czasie rzeczywistym program planujący zawsze uruchamia proces o wysokim priorytecie. Spośród procesów o tym samym priorytecie zostanie przeprowadzony długo oczekiwany proces. Jedyną różnicą między planowaniem FIFO a działaniem okrężnym jest to, że proces FIFO będzie kontynuowany, dopóki nie zostanie zamknięty lub zablokowany, podczas gdy proces okrężny będzie działał wywłaszczająco po chwili i zostanie przeniesiony na koniec kolejki,

Należy pamiętać, że planowanie w czasie rzeczywistym w systemie Linux ma miękkie właściwości. Kreator scheduler Linuksa oferuje ścisłe gwarancje dotyczące względnego priorytetu procesów w czasie rzeczywistym, ale jądro nie gwarantuje, jak szybkie będzie planowanie procesów w czasie rzeczywistym, gdy proces będzie gotowy do uruchomienia. Pamiętaj, że kod jądra Linuksa nigdy nie może być wywłaszczany przez kod trybu użytkownika. Jeśli wystąpi przerwa, która budzi proces czasu rzeczywistego, podczas gdy jądro jest gotowe do wykonania wywołania systemowego w ramach innego procesu, proces czasu rzeczywistego musi czekać, aż bieżące wywołanie systemowe zostanie zakończone lub zablokowane.

# Planowanie procesów i wątków hisotira

# Planowanie procesów i wątków na Linuxie