

Systemy operacyjne i architektura komputerów

Arkadiusz Syta, Jacek Zaburko
Politechnika Lubelska
Katedra Informatyki Technicznej

Treści programowe:

Historia i ewolucja systemów operacyjnych, klasyfikacja systemów operacyjnych oraz przykłady popularnych systemów operacyjnych



Historia i ewolucja systemów operacyjnych

System operacyjny (operating system, OS) – oprogramowanie zarządzające systemem komputerowym, tworzące środowisko do uruchamiania i kontroli zadań, najczęściej w formie programów^[1]

1. Podstawy systemów operacyjnych. T. 1 / Abraham Silberschatz, Peter B. Galvin, Greg Gagne. – Wydanie X (I w WN PWN). – Warszawa, 2021

Historia i ewolucja systemów operacyjnych

Pierwsza generacja (1945-1955) – lampy elektronowe:

- John Atanasoff i Clifford Berry (Uniwersytet Stanu Iowa) – pierwszy działający komputer cyfrowy – wykorzystano 300 lamp elektronowych
- Konrad Zasse – komputer Z3 – na bazie przekaźników (Berlin)
- Grupa naukowców (w tym Alan Turing) – komputer Colossus (Bletchley Park Anglia)
- Howard Aiken – komputer Mark I (Uniwersytet Harvarda)
- William Mauchley – komputer ENIAC (Uniwersytet Pensylwania)

Jedna grupa osób projektowała, budowała, programowała i utrzymywała

Programowanie – tylko język maszynowy lub za pomocą łączenia obwodów elektrycznych (tablice programowe)

Historia i ewolucja systemów operacyjnych

Druga generacja (1955-1965) – tranzystory i systemy wsadowe:

Mainframe – duże, centralne maszyny obliczeniowe – instytucje, uczelnie, przedsiębiorstwa

Automatyzacja – pierwszy krok w kierunku systemu operacyjnego – zamiast ręcznego przygotowania do wykonywania zadań – sekwencja zadań do wykonania, np. program sterujący który wczytywał zadanie, uruchamiał i po skończeniu przygotowywał komputer do kolejnego zadania

IBM 1401



IBM 7094



IBM 1401

wczytuje karty
kopiuje taśmy

wykonuje obliczenia

drukuje wyniki

Historia i ewolucja systemów operacyjnych

Druga generacja (1955-1965) – tranzystory i systemy wsadowe:

Mainframe – duże, centralne maszyny obliczeniowe – instytucje, uczelnie, przedsiębiorstwa

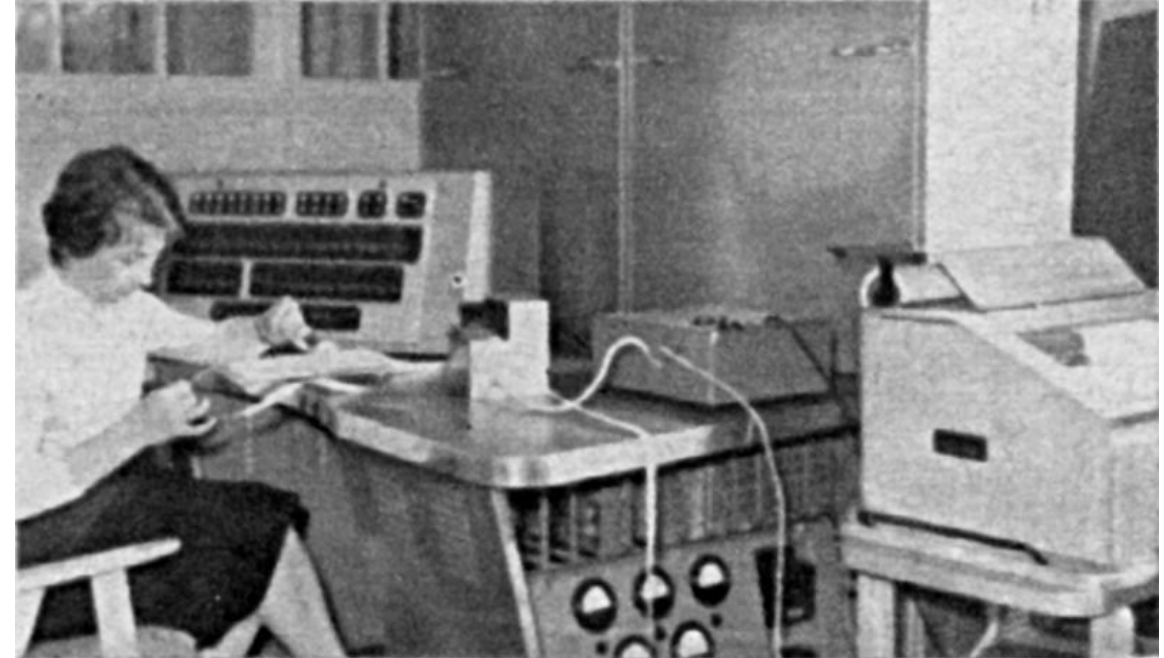
Przetwarzanie wsadowe (karty perforowane) – jednocześnie wielu użytkowników mogło przekazywać swoje zadania do wykonania (kolejki) ale zadania były wykonywane pojedynczo i bez interakcji z użytkownikiem – nieefektywne wykorzystanie zasobów

Urządzenia wejścia/wyjścia pracują wolno → zmiana organizacji – w czasie oczekiwania na nie komputer może wykonywać inne zadania

Historia i ewolucja systemów operacyjnych



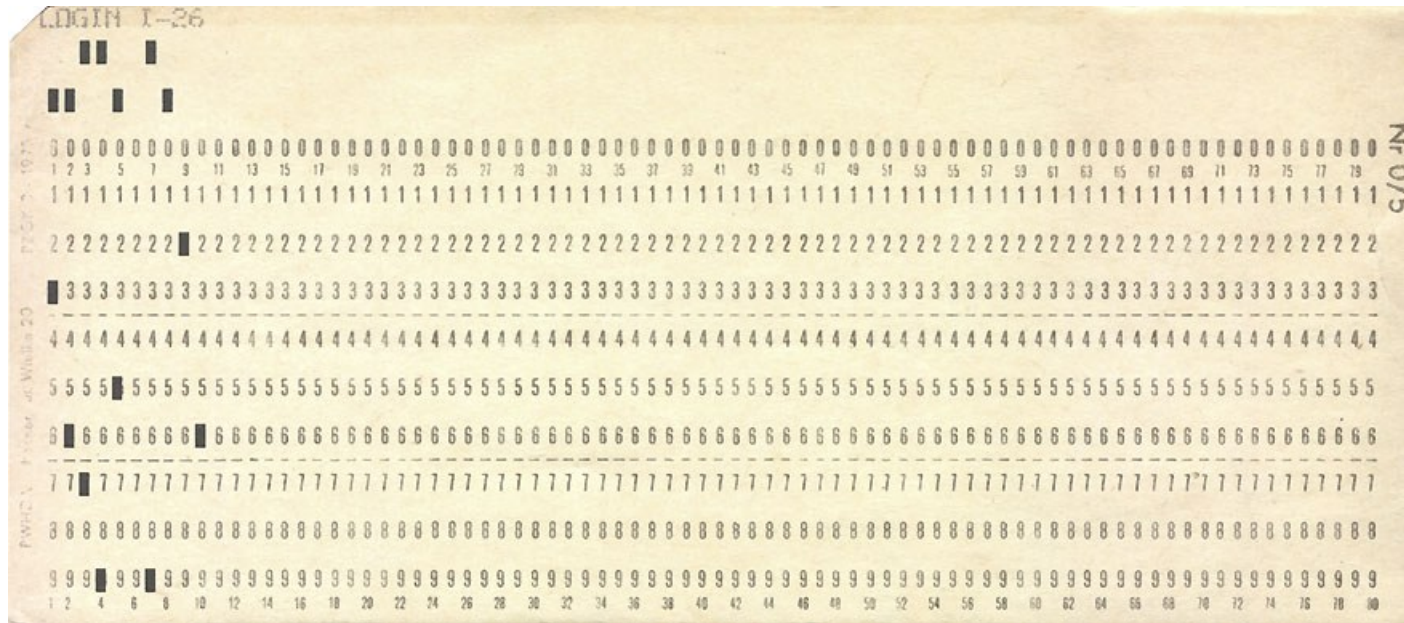
Mainframe IBM 704



Odra 1001

Źródło: Wikipedia

Historia i ewolucja systemów operacyjnych



Karta perforowana – pojemność do 80 bajtów

Źródło: Wikipedia

8

Historia i ewolucja systemów operacyjnych

Trzecia generacja (1965-1980) – układy scalone i wieloprogramowość:

Komputer IBM 360 – pierwszy komputer zbudowany w oparciu o układy scalone – lepszy stosunek cena/wydajność w porównaniu do drugiej generacji (tranzystory)

Pierwsza linia komputerów Systemem/360 zgodnych na poziomie oprogramowania z dostępnymi komputerami

System operacyjny musiał sprawnie działać na komputerach z małą liczbą urządzeń peryferyjnych (IBM 1401) oraz komputerach, które miały ich bardzo wiele (IBM 7094)

Złożony kod źródłowy (assembler) zawierający błędy

OS/360 realizował już wieloprogramowość

9

Historia i ewolucja systemów operacyjnych

Trzecia generacja (1965-1980) – układy scalone i wieloprogramowość:

Wieloprogramowość – umieszczenie w pamięci operacyjnej kilku programów jednocześnie – jeżeli jeden program czekał na operacje wejścia/wyjścia – system przełączał procesor na inny program

Nowe funkcje systemu operacyjnego:

- Przechowywanie informacji: stan programu w trakcie wykonywania
- Przydział pamięci operacyjnej: różnym programom w kontrolowany sposób
- Decyzja: który program ma być wykonany w danej chwili

System operacyjny staje się aktywnym zarządcą zasobów

10

Historia i ewolucja systemów operacyjnych

Trzecia generacja (1965-1980) – układy scalone i wieloprogramowość:

Wielodostęp – system potrafi przełączać wykonywanie programów w szybki sposób → przełącza użytkowników w szybki sposób

Nowe funkcje systemu operacyjnego:

- Wiele użytkowników pracuje w tym samym czasie – identyfikacja użytkowników i kontrola dostępu do zasobów
- Każdy użytkownik uruchamia własne programy – rozdzielenie programów według użytkowników i kontrola uprawnień
- Przełączanie wykonywania pomiędzy użytkownikami – planowanie przydziału czasu procesora

11

Historia i ewolucja systemów operacyjnych

Trzecia generacja (1965-1980) – układy scalone i wieloprogramowość:

W trzeciej generacji systemów operacyjnych następuje wyraźne uporządkowanie architektury systemu - pojawiają się fundamentalne pojęcia systemów:

- **proces** jako program w trakcie wykonywania wraz z jego stanem
- **użytkownik** jako jednostka posiadająca tożsamość i uprawnienia
- **pamięć operacyjna** jako zasób zarządzany centralnie przez system
- **ochrona zasobów** zapobiegająca wzajemnemu zakłócaniu pracy programów

System operacyjny przejmuje kontrolę nad przełączaniem wykonywania programów, obsługą przerwań oraz reakcją na zdarzenia pochodzące od urządzeń

Błędy programu użytkownika nie powinny prowadzić do zatrzymania pracy całego systemu – mechanizmy izolacji i ochrony

Historia i ewolucja systemów operacyjnych

Trzecia generacja (1965-1980) – układy scalone i wieloprogramowość:

UNIX - najbardziej wpływowy system operacyjny trzeciej generacji, opracowany w laboratoriach Bell przez Kena Thompsona i Dennisa Ritchiego

UNIX realizuje koncepcje wieloprogramowości i wielodostępu, a jednocześnie wprowadza prostą, modułarną strukturę systemu

UNIX zapoczątkował standardy:

- wyraźne rozdzielenie systemu operacyjnego i programów użytkownika
- jednolity model dostępu do zasobów poprzez pliki
- kontrolę uprawnień użytkowników
- możliwość przenoszenia systemu pomiędzy różnymi platformami sprzętowymi

13

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

Układy scalone wielkiej skali integracji – obwody zawierające kilka tysięcy na jednym centymetrze kwadratowym krzemu

Początek komputerów osobistych – 1974 Intel wyprodukował pierwszy 8-bitowy procesor ogólnego przeznaczenia – układ 8080

Na początku lat 80 IBM opracował komputer IBM PC – poszukiwał systemu operacyjnego

Bill Gates odkupił od Seattle Computer Products system operacyjny DOS (Disk Operating System) i zaoferował IBM wraz z interpreterem BASIC

IBM chciał poprawek – Gates (Microsoft) zatrudnił Tima Petersona (twórca DOS) –

MS-DOS (Microsoft Disk Operating System)

14

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

1983 – następca komputera IBM PC – IBM PC/AT – mikroprocesor Intel 80286

1973 – Xerox Alto – pierwszy komputer wyposażony w GUI (Graphical User Interface) umożliwiający obsługę okien, myszy oraz ekran bitmapowy

1981 – Xerox Star – pierwszy komercyjny komputer z graficznym interfejsem użytkownika

1984 – premiera Apple Macintosh – komputer przyjazny użytkownikom:

- GUI jako jedyny interfejs (brak trybu tekstowego dla użytkownika)
- mysz jako standardowe urządzenie wejściowe
- aplikacje działające w oknach
- spójny interfejs narzucony przez system operacyjny

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

System Windows – zaczyna jako środowisko graficzne – nakładka na MS-DOS

1985 – Windows 1.0: brak prawdziwej wielozadaniowości, okna nie nachodzą na siebie

1987 – Windows 2.0: okna mogą się nakładać, lepsza obsługa myszy

1990 – Windows 3.0: tryb chroniony procesora, pierwsze aplikacje graficzne

1992 – Windows 3.1: stabilniejszy interfejs, standard w komputerach osobistych

1995 – Windows 95: menu Start, pasek zadań, częściowa wielozadaniowość

1998 – Windows 98: poprawki stabilności lepsza obsługa sprzętu

2000 – Windows Me: ostatni Windows oparty na MS-DOS, problemy ze stabilnością

16

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

Równolegle tworzona linia profesjonalna Windows NT

1993 – Windows NT 3.1: całkowicie nowa architektura, prawdziwa wielozadaniowość, brak zależności od MS-DOS

1996 – Windows NT 4.0: interfejs zbliżony do Windows 95, stabilność i bezpieczeństwo

2001 – Windows XP: połączenie linii 9x i NT: stabilność, popularność, bardzo długi cykl życia

Od tego momentu wszystkie systemy Windows opierają się na architekturze NT

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

2007 – Windows Vista: nowy model sterowników, większe bezpieczeństwo, problemy wydajnościowe

2009 – Windows 7: poprawa Visty, duża popularność

2012 – Windows 8: interfejs dotykowy, kontrowersyjny odbiór

2015 – Windows 10: jeden system dla wielu urządzeń, model ciągłych aktualizacji

2021 – Windows 11: zmiany interfejsu, nacisk na nowy sprzęt

Windows ewoluował od graficznej nakładki na system tekstowy do pełnoprawnego systemu operacyjnego opartego na architekturze NT

18

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

1987 – powstaje MINIX – twórca: Andrew S. Tanenbaum

System zaprojektowany jako system dydaktyczny – główny cel: nauczanie budowy systemów operacyjnych

Architektura MINIX opiera się na jądrze mikrokernelowym – czytelne rozdzielenie komponentów systemu

Cechy: kod źródłowy dostępny do nauki, uproszczona architektura, nacisk na poprawność i przejrzystość, a nie wydajność

MINIX nie był projektowany jako system masowy – odegrał ważną rolę jako referencyjna implementacja idei systemów typu Unix w środowisku akademickim

19

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

1991 – powstaje jądro Linux – twórca: Linus Torvalds

1992 – objęcie systemu licencją GNU General Public License – prawo do:

uruchamiania programu w dowolnym celu, analizowania, dostępu do kodu źródłowego, modyfikowania programu, rozpowszechniania oryginalnej lub zmodyfikowanej wersji

Linux powstał jako jądro systemu operacyjnego, rozwijane publicznie i dostępne bez ograniczeń licencyjnych w otwartym modelu współpracy – udział programistów z całego świata

Linux to nie pojedynczym systemem – podstawa wielu dystrybucji, które łączą jądro z narzędziami systemowymi i oprogramowaniem użytkowym

20

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

Linux nie zawiera środowiska graficznego (jądro systemu operacyjnego) – warstwa nad jądrem działająca jako serwer:

System X11 (X Window System): system okien obsługujący: wyświetlanie grafiki, klawiaturę, mysz, komunikacje ze sprzętem graficznym

X11 nie jest interfejsem użytkownika – dostarcza mechanizm, a nie politykę wyglądu

serwer X: kontroluje ekran, klawiaturę i mysz, działa na komputerze użytkownika

klient X: aplikacja graficzna, może działać lokalnie lub zdalnie

Środowisko graficzne: działa nad systemem X11 – GNOME, KDE Plasma, Xfce

21

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

Wayland – nowoczesny protokół wyświetlania, zaprojektowany jako uproszczenie i następca architektury X11: architektura X11 jest bardzo złożona i trudna w utrzymaniu, duża liczba warstw pośrednich powoduje opóźnienia, brak naturalnego modelu bezpieczeństwa aplikacji graficznych

Najważniejsze zmiany w architekturze: brak osobnego serwera X jako centralnego procesu, środowisko graficzne pełni rolę kompozytora i serwera wyświetlania, aplikacje rysują swoje okna bezpośrednio do buforów, kompozytor składa obraz końcowy na ekranie

Konsekwencje: mniejsze opóźnienia, prostsza architektura, lepsza izolacja aplikacji, naturalna obsługa nowoczesnej grafiki i wielu monitorów

22

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

- 1993 – początek projektu FreeBSD – Uniwersyt Kalifornijski w Berkeley
BSD – Berkeley Software Distribution

FreeBSD wywodzi się bezpośrednio z linii BSD Unix – środowisko akademickie

W przeciwieństwie do Linuxa, FreeBSD jest rozwijany jako jeden spójny system operacyjny, obejmujący jądro, narzędzia systemowe i dokumentację

Cechy charakterystyczne: centralnie koordynowany rozwój, licencja BSD umożliwiająca użycie w systemach zamkniętych, nacisk na stabilność, przewidywalność i jakość kodu

MacOS – zmodyfikowana wersja FreeBSD

iOS i Android – zmodyfikowana wersja FreeBSD

23

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

W latach 80 systemy operacyjne zaczęły traktować sieć jako podstawowy zasób, co doprowadziło do powstania systemów sieciowych oraz początków systemów rozproszonych

Systemy sieciowe (network operating systems): każdy komputer działa autonomicznie, system operacyjny udostępnia zasoby (zdalne logowanie, przesyłanie plików, drukarki sieciowe), użytkownik wie, że pracuje na wielu maszynach

Przykładowe mechanizmy: zdalne powłoki, sieciowe systemy plików, usługi klient-serwer

Historia i ewolucja systemów operacyjnych

Czwarta generacja (1980-obecnie):

W latach 80 systemy operacyjne zaczęły traktować sieć jako podstawowy zasób, co doprowadziło do powstania systemów sieciowych oraz początków systemów rozproszonych

Systemy rozproszone (distributed operating systems): wiele komputerów ma sprawiać wrażenie jednego systemu, zasoby są współdzielone w sposób przeźroczysty, użytkownik nie musi wiedzieć, na której maszynie działa program

Cele systemów rozproszonych: przeźroczystość lokalizacji, równoważenie obciążenia, odporność na awarie, skalowalność

Historia i ewolucja systemów operacyjnych

Piąta generacja (1990-obecnie):

Piąta generacja systemów operacyjnych, rozwijana od lat 90 – systemy mobilne zaprojektowane z myślą o energooszczędności, bezpieczeństwie i obsłudze dotykowej

1998 – Symbian – twórcy: konsorcjum firm (m.in. Nokia, Ericsson, Motorola)
Dominujący w telefonach komórkowych przed smartfonami, niskie zużycie energii

1999 – BlackBerry OS – twórca: Research In Motion
Zoptymalizowany pod komunikację korporacyjną i pocztę elektroniczną, silna integracja z usługami serwerowymi

2000 – Windows Mobile – twórca: Microsoft
Adaptacja systemów komputerowych do urządzeń mobilnych

2010 – następca: Windows Phone

26

Historia i ewolucja systemów operacyjnych

Piąta generacja (1990-obecnie):

Piąta generacja systemów operacyjnych, rozwijana od lat 90 – systemy mobilne zaprojektowane z myślą o energooszczędności, bezpieczeństwie i obsłudze dotykowej

2007 – iOS (Apple), zaprojektowany od podstaw jako system operacyjny dla urządzenia sterowanego wyłącznie dotykem (bez adaptacji systemów komputerowych), architektura systemu zakłada ścisłą izolację aplikacji (każda działa w osobnej piaskownicy), centralny sklep aplikacji jako jedyny oficjalny kanał dystrybucji oraz pełną kontrolę platformy przez producenta, spójny ekosystem (system, sprzęt i oprogramowanie użytkowe) – większe bezpieczeństwo i stabilność, kosztem ograniczone możliwości modyfikacji

2008 – Android (obecnie Google), oparty na jądrze Linux, otwarta platforma, możliwość uruchamiania systemu na sprzęcie wielu producentów, izolacja aplikacji, model uprawnień użytkownika, możliwość modyfikacji interfejsu i funkcji systemowych

27

Klasyfikacja systemów operacyjnych

Systemy operacyjne można klasyfikować według kilku niezależnych kryteriów, z których każde opisuje inny aspekt ich działania i przeznaczenia

Ze względu na liczbę użytkowników

- Jednoużytkownikowe – projektowane z myślą o jednym aktywnym użytkowniku w danej chwili (np. systemy osobiste)
- Wieloużytkownikowe – umożliwiają jednoczesną pracę wielu użytkowników z izolacją uprawnień i zasobów (np. systemy serwerowe)

Ze względu na sposób wykonywania programów

- Jednozadaniowe – wykonują jeden program w danym momencie
- Wielozadaniowe – wykonują wiele programów pozornie jednocześnie dzięki przełączaniu wykonywania (z wywłaszczaniem lub bez)

28

Klasyfikacja systemów operacyjnych

Systemy operacyjne można klasyfikować według kilku niezależnych kryteriów, z których każde opisuje inny aspekt ich działania i przeznaczenia

Ze względu na organizację czasu

- Systemy wsadowe – programy wykonywane kolejno, bez interakcji z użytkownikiem
- Systemy interaktywne – szybka reakcja na działania użytkownika
- Systemy czasu rzeczywistego – gwarantowany czas reakcji na zdarzenia

Ze względu na architekturę sprzętową

- Jednoprocessorowe – jeden procesor wykonujący instrukcje
- Wieloprocessorowe / wielordzeniowe – równoległe wykonywanie na wielu rdzeniach lub procesorach

29

Klasyfikacja systemów operacyjnych

Systemy operacyjne można klasyfikować według kilku niezależnych kryteriów, z których każde opisuje inny aspekt ich działania i przeznaczenia

Ze względu na strukturę systemu

- Monolityczne – większość funkcji systemu w jednym jądrze (np. Linux)
- Warstwowe – system podzielony na warstwy funkcjonalne
- Mikrojądrowe – minimalne jądro, reszta w procesach użytkownika
- Hybrydowe – połączenie podejść (np. Windows NT)

Klasyfikacja systemów operacyjnych

Systemy operacyjne można klasyfikować według kilku niezależnych kryteriów, z których każde opisuje inny aspekt ich działania i przeznaczenia

Ze względu na przeznaczenie

- Ogólnego przeznaczenia – komputery osobiste i serwery (np. Windows, Linux)
- Mobilne – smartfony i tablety (np. Android, iOS)
- Wbudowane – urządzenia specjalizowane (routery, sterowniki, IoT)

Klasyfikacja systemów operacyjnych

Systemy operacyjne można klasyfikować według kilku niezależnych kryteriów, z których każde opisuje inny aspekt ich działania i przeznaczenia

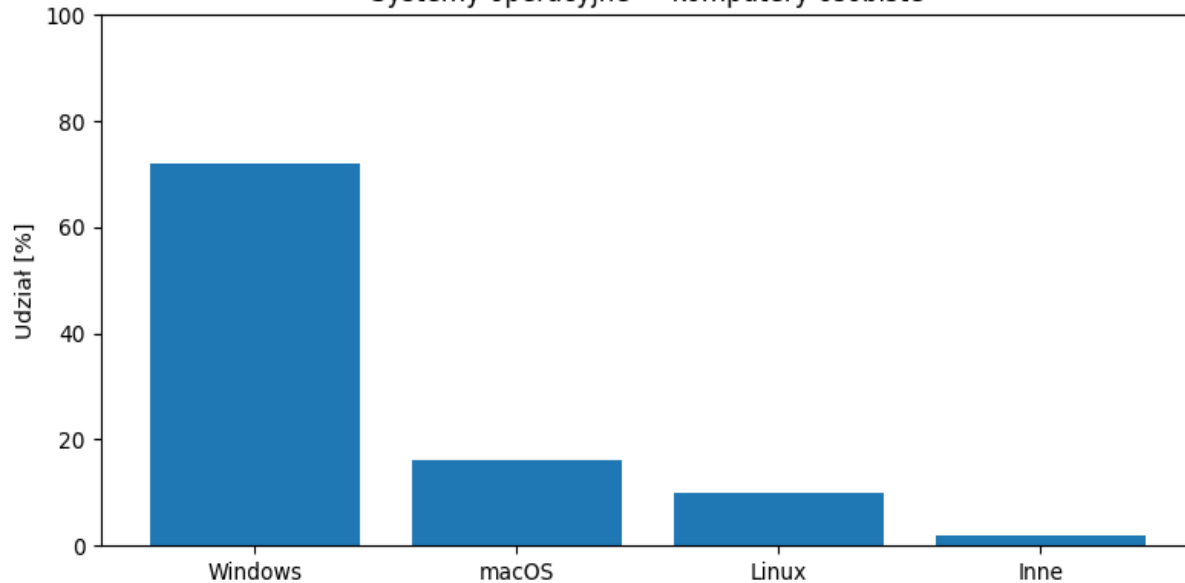
Ze względu na izolację

- Izolacja na poziomie zwykłego procesu – każdy program (proces) ma własną przestrzeń adresową – nie widzi pamięci innych procesów
- Izolacja na poziomie maszyny wirtualnej – każda maszyna wirtualna ma własny kompletny system operacyjny, własne jądro, własne urządzenia wirtualne
- Izolacja na poziomie kontenera – specjalny proces (lub grupa procesów), który dzięki mechanizmom jądra (przestrzenie nazw i grupy kontrolne) widzi tylko swoją część systemu plików, swoje sieciowe adresy, swoje procesy – jądro jest wspólne dla wszystkich kontenerów

32

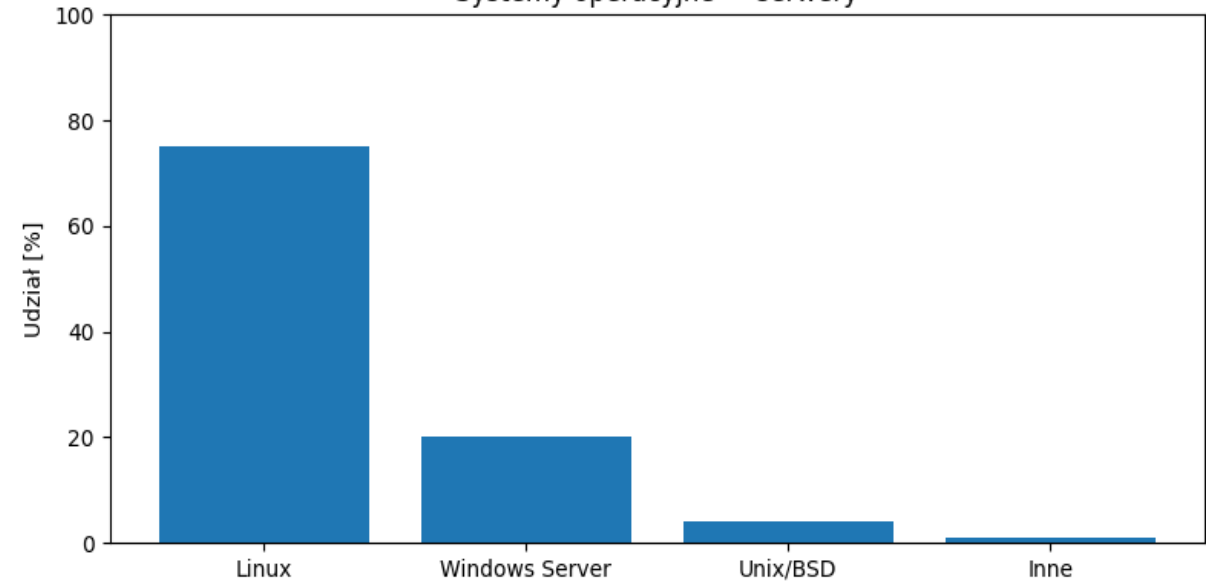
Przykłady popularnych systemów operacyjnych

Systemy operacyjne — komputery osobiste



Źródło:
StatCounter – Desktop Operating System Market Share

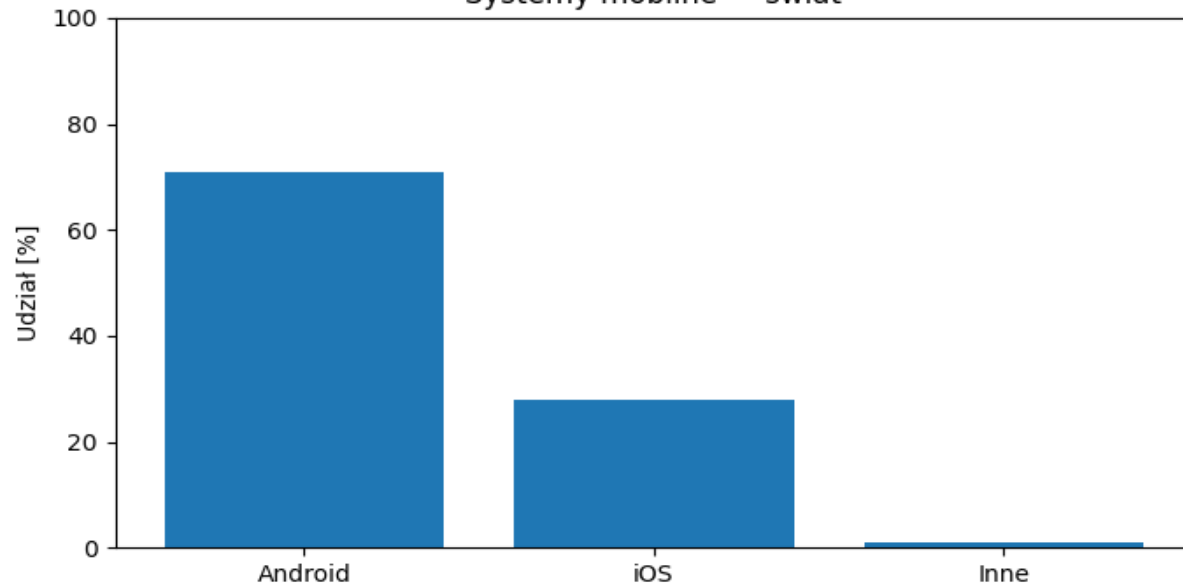
Systemy operacyjne — serwery



Źródło:
W3Techs – Usage of Operating Systems for Websites

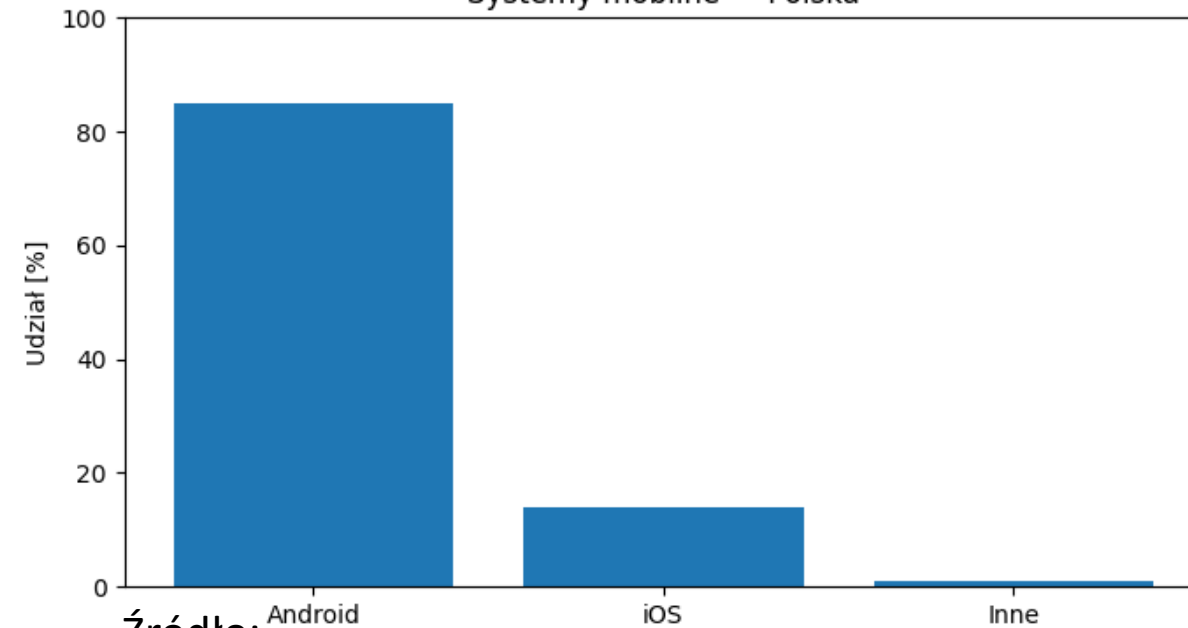
Przykłady popularnych systemów operacyjnych

Systemy mobilne — świat



Źródło:
Mobile Operating System Market Share

Systemy mobilne — Polska



Źródło:
Mobile Operating System Market Share

Dziękuję za uwagę

Treści programowe:

Podstawowe pojęcia i definicje, funkcje
i zadania systemu operacyjnego oraz
architektura systemu operacyjnego



Podstawowe pojęcia i definicje

System operacyjny to podstawowe oprogramowanie systemowe, które organizuje wykonywanie programów, zarządza zasobami logicznymi systemu oraz udostępnia jednolite środowisko pracy dla aplikacji i użytkowników

System operacyjny działa ciągle, od momentu uruchomienia komputera aż do jego wyłączenia

System operacyjny znajduje się pomiędzy aplikacjami a sprzętem – warstwa pośrednia

Aplikacje nie komunikują się bezpośrednio z zasobami systemu, lecz korzystają z usług oferowanych przez system operacyjny – uproszczenie tworzenia aplikacji, zwiększenie bezpieczeństwa, uniezależnienie oprogramowania od szczegółów zasobów (urządzenia, lokalizacji, implementacji) – abstrakcyjny interfejs

37

Proces uruchamiania komputera

Komputer klasy PC zawiera płytę główną z pamięcią nieulotną (flash), w której zapisane jest oprogramowanie firmware systemu – pierwszy kod wykonywany po włączeniu zasilania – działa całkowicie niezależnie od systemu operacyjnego

Współcześnie spotykane są dwa standardy firmware:

BIOS (Basic Input/Output System) – klasyczne rozwiązanie, kompatybilne z architekturą IBM PC, inicjuje podstawowe zasoby systemu, wykonuje test startowy (POST) i przygotowuje środowisko do uruchomienia programu rozruchowego (rozwiązanie historyczne, zależne od architektury)

UEFI (Unified Extensible Firmware Interface) – następca BIOS (Intel), udostępnia rozbudowane środowisko startowe, obsługuje sterowniki firmware, zna systemy plików i umożliwia uruchamianie programów rozruchowych jako plików

38

Proces uruchamiania komputera

Po zakończeniu działania programu rozruchowego sterowanie zostaje przekazane do jądra systemu operacyjnego, które zostaje uruchomione w trybie uprzywilejowanym

Od tego momentu system operacyjny przejmuje pełną kontrolę nad komputerem, a firmware oraz bootloader przestają uczestniczyć w dalszym działaniu systemu

Jądro rozpoczyna pracę jako jedyny aktywny byt wykonawczy. Nie istnieją jeszcze procesy użytkownika ani środowisko pracy. Na tym etapie jądro:

- inicjalizuje własne struktury danych
- ustawia mechanizmy ochrony pamięci
- przygotowuje obsługę wyjątków i przerwań
- konfiguruje podstawowe mechanizmy planowania wykonywania

39

Krytyczny etap – błąd w jądrze powoduje natychmiastowe zatrzymanie systemu

Proces uruchamiania komputera

Po wstępnej inicjalizacji jądro uruchamia kluczowe podsystemy systemowe, takie jak:

- zarządzanie pamięcią
- planowanie procesów i wątków
- system plików (na poziomie jądra)
- mechanizmy komunikacji między procesami

Następnie jądro tworzy pierwszy proces użytkownika, który pełni rolę procesu nadrzędnego dla wszystkich pozostałych: nie jest uruchamiany przez użytkownika, posiada specjalny status w systemie, odpowiada za dalszy start systemu

Pierwszy proces uruchamia kolejne usługi systemowe, przygotowuje środowisko logowania i inicjalizuje interfejs użytkownika – system przechodzi w tryb pełnej wielozadaniowości i jest gotowy do normalnej pracy

40

Podstawowe pojęcia i definicje

System operacyjny pełni rolę nadrzędnego kontrolera systemu komputerowego

Wszystkie programy użytkowe działają pod jego kontrolą i nie mają bezpośredniego dostępu do zasobów systemowych

Rola systemu operacyjnego:

- pośredniczenie pomiędzy aplikacjami a zasobami
- organizację współdzielenia zasobów
- ochronę systemu przed błędami aplikacji
- zapewnienie stabilnej pracy

System operacyjny zawsze zachowuje kontrolę nad systemem

41

Podstawowe pojęcia i definicje

Abstrakcja – ukrycie szczegółów realizacji zasobów przed programami użytkowymi

Aplikacje korzystają z zasobów poprzez ujednolicone interfejsy – nie znając ich fizycznej ani technicznej implementacji

Przykłady abstrakcji:

- proces zamiast fizycznego wykonania
- plik zamiast nośnika danych
- pamięć logiczna zamiast rzeczywistego rozmieszczenia danych

Abstrakcja upraszcza programowanie i zwiększa bezpieczeństwo systemu

42

Podstawowe pojęcia i definicje

Zasoby systemowe – elementy systemu, których dostęp jest zarządzany i kontrolowany przez system operacyjny

Zasoby nie są przypisywane programom na stałe, lecz przydzielane dynamicznie

Do zasobów systemowych należą:

- czas wykonania
- pamięć
- pliki i struktury danych
- mechanizmy komunikacji

System operacyjny decyduje o sposobie i czasie wykorzystania zasobów

43

Podstawowe pojęcia i definicje

Program – pasywna postać oprogramowania zapisana w postaci pliku

Program zawiera kod i dane, ale sam w sobie:

- nie jest wykonywany
- nie zużywa zasobów
- nie posiada stanu wykonania

Program staje się aktywny dopiero po uruchomieniu przez system operacyjny

Podstawowe pojęcia i definicje

Proces – instancja programu w trakcie wykonywania, utworzona i zarządzana przez system operacyjny, posiada:

- własną przestrzeń adresową
- własny stan wykonania
- przydzielone zasoby
- jednoznaczność tożsamości

Proces jest podstawową jednostką zarządzania wykonywaniem w systemie operacyjnym

Kontekst procesu – zbiór informacji opisujących aktualny stan procesu, umożliwiający jego zatrzymanie i późniejsze wznowienie, zapisywany i odtwarzany przez system operacyjny

45

Podstawowe pojęcia i definicje

Izolacja procesów – każdy proces działa w odseparowanym środowisku, niewidocznym dla innych procesów

Proces nie może bezpośrednio:

- odczytywać pamięci innych procesów
- modyfikować ich danych
- wpływać na ich wykonanie

Izolacja – podstawowy mechanizm stabilności i bezpieczeństwa systemu

Podstawowe pojęcia i definicje

Wątek – najmniejsza jednostka wykonywania, planowana przez system operacyjny

Wątki działają wewnątrz procesu:

- współdzielą jego zasoby
- posiadają własny stan wykonania

Zastosowanie wątków umożliwia współbieżne wykonywanie zadań w ramach jednego procesu

Podstawowe pojęcia i definicje

Proces zapewnia:

- izolację
- bezpieczeństwo
- przydział zasobów

Wątek zapewnia:

- współbieżność
- efektywne wykonywanie
- mniejszy narzut niż tworzenie wielu procesów

System operacyjny planuje wykonanie wątków, a nie całych procesów

48

Podstawowe pojęcia i definicje

Przestrzeń użytkownika – środowisko wykonywania aplikacji

Kod działający w tej przestrzeni:

- ma ograniczone uprawnienia
- nie może bezpośrednio korzystać z mechanizmów systemowych
- musi korzystać z usług systemu operacyjnego

Przestrzeń użytkownika chroni system przed błędami aplikacji

Podstawowe pojęcia i definicje

Przestrzeń systemowa (jądro) – zarezerwowana dla jądra systemu operacyjnego

Kod w tej przestrzeni:

- ma pełne uprawnienia
- zarządza zasobami
- kontroluje działanie całego systemu

Błędy w przestrzeni systemowej są krytyczne dla działania systemu

Podstawowe pojęcia i definicje

Wielozadaniowość – zdolność systemu operacyjnego do obsługi wielu procesów w tym samym czasie logicznym

System operacyjny przełącza wykonywanie pomiędzy procesami, tworząc iluzję równoczesności

Wielozadaniowość jest realizowana przez:

- planowanie wykonania
- przełączanie kontekstu
- kontrolę dostępu do zasobów

Podstawowe pojęcia i definicje

Wielodostęp – możliwość jednoczesnej pracy wielu użytkowników w tym samym systemie

System operacyjny:

- identyfikuje użytkowników
- rozdziela ich procesy
- egzekwuje uprawnienia

Wielodostęp jest kluczową cechą systemów serwerowych i sieciowych

Architektura systemu operacyjnego

Architektura systemu operacyjnego – wewnętrzna struktura systemu – podział na komponenty oraz sposób ich współpracy:

- określa, które elementy działają w trybie uprzywilejowanym
- określa, które elementy działają w przestrzeni użytkownika
- określa w jaki sposób realizowana jest komunikacja między nimi

Architektura ma bezpośredni wpływ na: wydajność, bezpieczeństwo, stabilność systemu, możliwość rozbudowy

Zmiana architektury systemu operacyjnego jest kosztowna i trudna – fundamenty architektury pozostają niezmiennie przez wiele lat

Architektura systemu operacyjnego

Architektura systemu operacyjnego zależy od jego przeznaczenia:

- systemy desktopowe – responsywność
- systemy serwerowe – stabilność i skalowalność
- systemy wbudowane – minimalizm i deterministyczność

Ten sam system operacyjny może być konfigurowany inaczej, ale architektura wyznacza granice tych konfiguracji

Architektura systemu operacyjnego

Najczęściej system operacyjny opisuje się jako strukturę warstwową – każda warstwa:

- korzysta z usług warstwy niższej
- nie zna jej szczegółów implementacyjnych
- realizuje określony zakres odpowiedzialności

Architektura warstwowa – podziale systemu operacyjnego na logiczne poziomy: każdy realizuje ściśle określony zakres funkcji

Warstwy są uporządkowane hierarchicznie – każda warstwa korzysta wyłącznie z usług warstwy bezpośrednio niższej

Celem architektury warstwowej jest: ograniczenie złożoności systemu, izolacja odpowiedzialności, ułatwienie rozwoju i utrzymania, zwiększenie bezpieczeństwa poprzez separację funkcji

55

Architektura systemu operacyjnego – warstwa sprzętowa

Warstwa sprzętowa obejmuje fizyczne elementy systemu komputerowego:

- procesor wykonujący instrukcje
- pamięć operacyjną
- kontrolery urządzeń wejścia/wyjścia

Sprzęt:

- nie zna pojęcia procesu, pliku ani użytkownika
- realizuje jedynie proste operacje niskiego poziomu
- wymaga pośrednictwa systemu operacyjnego

Ta warstwa stanowi punkt wyjścia dla całej architektury – nie zapewnia środowiska do uruchamiania programów

56

Architektura systemu operacyjnego – warstwa jądra systemu operacyjnego

Jądro – centralna warstwa systemu operacyjnego – działa w trybie uprzywilejowanym procesora – realizuje mechanizmy krytyczne dla działania systemu

Jądro odpowiada za:

- zarządzanie wykonywaniem (planowanie)
- zarządzanie pamięcią i ochronę pamięci
- obsługę przerwań i wyjątków
- realizację wywołań systemowych

Jądro nie udostępnia interfejsu użytkownikowi — jest wyłącznie zapleczem wykonawczym systemu

57

Architektura systemu operacyjnego – warstwa sterowników, wejścia/wyjścia

Warstwa sterowników pośredniczy pomiędzy jądrem a sprzętem:

- tłumaczy ogólne żądania systemowe na operacje sprzętowe
- obsługuje konkretne protokoły urządzeń
- izoluje jądro od szczegółów sprzętowych

Sterowniki:

- silnie zależne od sprzętu
- działają blisko jądra
- mają krytyczny wpływ na stabilność systemu

Bez warstwy sterowników jądro nie byłoby w stanie komunikować się z urządzeniami

58

Architektura systemu operacyjnego – warstwa usług systemowych

Usługi systemowe działają poza jądrem – rozszerzenie funkcjonalne

Realizują zadania, które:

- nie muszą być wykonywane w trybie uprzywilejowanym
- wymagają długotrwałego działania
- obsługują funkcje ogólnosystemowe

Przykładowe zadania usług:

- uruchamianie kolejnych elementów systemu, obsługa logowania
- zarządzanie sesjami użytkowników, usługi sieciowe

Oddzielenie usług od jądra zwiększa stabilność systemu

59

Architektura systemu operacyjnego – warstwa interfejsów systemowych

Warstwa interfejsów systemowych – udostępnia aplikacjom formalny sposób korzystania z usług systemu operacyjnego, obejmuje:

- wywołania systemowe
- biblioteki systemowe
- mechanizmy komunikacji z jądrem

Aplikacje:

- nie komunikują się bezpośrednio z jądrem
- korzystają z ustandaryzowanych interfejsów
- pozostają niezależne od implementacji systemu

Warstwa interfejsów systemowych – granica pomiędzy systemem a oprogramowaniem

60

Architektura systemu operacyjnego – warstwa aplikacji użytkownika

Warstwa aplikacji użytkownika – działają w przestrzeni użytkownika:

- nie mają bezpośredniego dostępu do sprzętu
- korzystają wyłącznie z interfejsów systemowych
- podlegają pełnej kontroli systemu operacyjnego

Aplikacje:

- są izolowane od siebie
- mogą być bezpiecznie uruchamiane i kończone
- nie wpływają bezpośrednio na stabilność systemu

Architektura systemu operacyjnego – przepływ pomiędzy warstwami

Typowe żądanie aplikacji:

- aplikacja zgłasza żądanie przez interfejs systemowy
- następuje przejście do jądra
- jądro deleguje operację do odpowiedniego podsystemu
- sterownik wykonuje operację sprzętową
- wynik wraca tą samą drogą do aplikacji

Każda warstwa realizuje tylko swoją odpowiedzialność

Zalety: przejrzystość struktury, łatwiejsze testowanie, lepsza izolacja błędów

Ograniczenia: narzut wydajnościowy, koszt przechodzenia między warstwami, konieczność starannego projektowania interfejsów

62

Architektura systemu operacyjnego – granica użytkownik – jądro

Granica pomiędzy przestrzenią użytkownika a jądrem systemu operacyjnego – najważniejszym elementem architektury systemu, oddziela kod aplikacji od kodu systemowego i definiuje, które operacje mogą być wykonywane bezpośrednio, a które wymagają pośrednictwa systemu operacyjnego

Przestrzeń użytkownika: służy do wykonywania aplikacji, posiada ograniczone uprawnienia, nie ma bezpośredniego dostępu do sprzętu

Przestrzeń jądra: zawiera mechanizmy zarządzania zasobami, obsługuje przerwania i wyjątki, egzekwuje zasady bezpieczeństwa

Każde przekroczenie tej granicy jest kontrolowane i kosztowne, ale konieczne dla bezpieczeństwa systemu

Architektura systemu operacyjnego

Wywołania systemowe – ściśle zdefiniowany interfejs architektoniczny pomiędzy aplikacjami a jądrem systemu operacyjnego

Wywołania systemowe są jedynym legalnym mechanizmem umożliwiającym aplikacjom wykonywanie operacji systemowych

Architektura systemu operacyjnego określa:

- zestaw dostępnych wywołań systemowych
- sposób przekazywania parametrów
- mechanizm przełączania trybu wykonania
- sposób obsługi błędów

Architektura systemu operacyjnego – planista

Planista wykonywania – element architektury jądra odpowiedzialny za podejmowanie decyzji o przydziale czasu procesora, realizuje wielozadaniowość

Architektura systemu określa:

- gdzie planista jest umiejscowiony
- jakie informacje posiada o procesach i wątkach
- kiedy może przełączyć wykonywanie
- jakie zdarzenia mogą wywołać decyzję planistyczną

Bez planisty system operacyjny nie mógłby obsługiwać wielu programów w tym samym czasie

Architektura systemu operacyjnego – przełączanie kontekstu

Przełączanie kontekstu – bezpośredni skutek architektury wielozadaniowej systemu operacyjnego

Polega na:

- zapisaniu stanu aktualnie wykonywanego wątku
- odtworzeniu stanu innego wątku
- aktualizacji struktur jądra związanych z pamięcią i zasobami

Operacja kosztowna czasowo, dlatego architektura systemu:

- minimalizuje liczbę przełączeń
- stara się utrzymywać wykonywanie w przestrzeni użytkownika
- ogranicza niepotrzebne przejścia do jądra

66

Architektura systemu operacyjnego – bezpieczeństwo

Bezpieczeństwo systemu operacyjnego nie wynika z pojedynczego mechanizmu, lecz z całości architektury, obejmuje:

- izolację procesów
- ochronę pamięci
- separację użytkowników
- kontrolę dostępu do zasobów
- ochronę jądra przed aplikacjami

Jeżeli architektura systemu jest błędna, dodatkowe mechanizmy bezpieczeństwa nie są w stanie zapewnić pełnej ochrony

67

Architektura systemu operacyjnego – stabilność

Stabilność systemu operacyjnego jest bezpośrednią konsekwencją:

- zakresu kodu działającego w jądrze
- stopnia izolacji komponentów
- sposobu obsługi błędów i wyjątków

Właściwa architektura zapewnia, że:

- błąd aplikacji kończy tylko proces
- błąd usługi nie zatrzymuje jądra
- jądro pozostaje nadrzędne wobec wszystkich komponentów

Dziękuję za uwagę

Treści programowe:

Model von Neumanna, podstawowe
komponenty komputera oraz
organizacja i działanie procesora



Architektura komputera – model

Komputer jest systemem złożonym o strukturze hierarchicznej – modele abstrakcyjne pozwalają analizować i projektować komputery

Założenia modeli:

- upraszczają rzeczywistość
- opisują najważniejsze elementy
- pozwalają zrozumieć zasady działania

Wybrane modele komputera (podział ze względu na organizację pamięci):

- model von Neumanna – dane i kod programu przechowywane w tym samym obszarze
- model harwardzki – dane i kod programu przechowywane w różnych obszarach
- model strumieniowy – w odseparowanych obszarach, ale połączone magistralą

71

Architektura komputera – model

Wybrane modele komputera (podział ze względu na połączenie procesor-pamięć i sposób ich wykorzystania) – **taksonomia Flynna**:

- SISD (ang. Single Instruction Single Data) – skalarne
- SIMD (ang. Single Instruction Multiple Data) – wektorowe (macierzowe)
- MISD (ang. Multiple Instruction Single Data) – strumieniowe
- MIMD (ang. Multiple Instruction Multiple Data) – równoległe

Architektura komputera – model

Wybrane modele komputera (podział ze względu na sposób podziału pracy i dostęp procesora do pamięci):

- SMP (ang. Symmetric Multiprocessing) – symetryczne
- ASMP (ang. Asymmetric Multiprocessing) – asymetryczne
- NUMA (ang. Non-Uniform Memory Access) – asymetryczne (rozdzielające pamięć lokalną i zdalną)
- AMP (ang. Asynchronous Multiprocessing) – asynchroniczne
- MPP (ang. Massively Parallel Processors) – równoległe

Architektura komputera – model strumieniowy

Wybrane modele komputera (podział ze względu na organizację pamięci):

Model strumieniowy: architektura zorientowana na ciągły przepływ danych:

dane napływają ciągłym strumieniem (stream), przetwarzane w locie (on-the-fly), wyniki są natychmiast przekazywane dalej (lub na wyjście)

Kluczowe cechy:

- brak (lub minimalny) bufor pamięciowy między etapami
- przetwarzanie potokowe / pipeline'owe
- bardzo niskie opóźnienie (low latency)
- wysoka przepustowość (high throughput)
- brak losowego dostępu do całej pamięci

74

Architektura komputera – model strumieniowy

Typowe źródła strumienia:

- sensory (kamera, mikrofon, radar, akcelerometr)
- transmisje sieciowe (logi, pakiety, giełda, Kafka, MQTT)
- generowane dane (symulacje, rendering, inferencja AI)

Typowe układy realizujące ten model:

- procesory sygnałowe (DSP)
- układy FPGA
- karty graficzne (GPU)
- akceleratory AI (TPU, NPU, systolic arrays)

75

Architektura komputera – model harwardzki

Wybrane modele komputera (podział ze względu na organizację pamięci):

Model Harwardzki (Mark Harvard 1944): oddzielna pamięć programu i danych

Najważniejsza zasada: dwie fizycznie oddzielne pamięci i dwie niezależne magistrale

Kluczowe własności:

- jednoczesny odczyt instrukcji i danych
- możliwość różnych szerokości słów dla kodu i danych
- łatwiejsze projektowanie potoków w procesorze

Historyczne korzenie:

wczesne komputery (pierwsza i druga generacja) i procesory sygnałowe (przetwarzanie)

Architektura komputera – model harwardzki

Modyfikowany model harwardzki (2025):

fizycznie jedna pamięć główna ale oddzielne cache:

- I-cache (instrukcje)
- D-cache (dane)

umożliwia jednoczesny odczyt instrukcji i danych (przy trafieniu w cache)

Przykłady popularnych rodzin:

- ARM Cortex-M (STM32, NXP, etc.)
- ARM Cortex-A (smartfony, tablety – w połączeniu z cache)
- AVR (większość modeli Microchip)
- klasyczne DSP (TI C2000, C5000, C6000)

Architektura komputera – model von Neumanna

Wybrane modele komputera (podział ze względu na organizację pamięci):

Model von Neumanna (John von Neumann 1945): wspólna pamięć programu i danych

Najważniejsza zasada: koncepcja programu zapisanego w pamięci (stored-program computer)

Dominuje w komputerach osobistych, serwerach i większości mikrokontrolerów do dziś

Historia architektury von Neumanna:

- 1943-1945 → projekt ENIAC (pierwszy programowalny komputer elektroniczny)
- 1949 → EDSAC (Cambridge) – pierwszy rzeczywisty komputer wg koncepcji
- 1951-1952 → IAS machine (Princeton) – budowa pod kierunkiem von Neumanna
- Od 1950 → masowe wdrożenie (UNIVAC, IBM 701, IBM 704...)

78

Architektura komputera – model von Neumanna

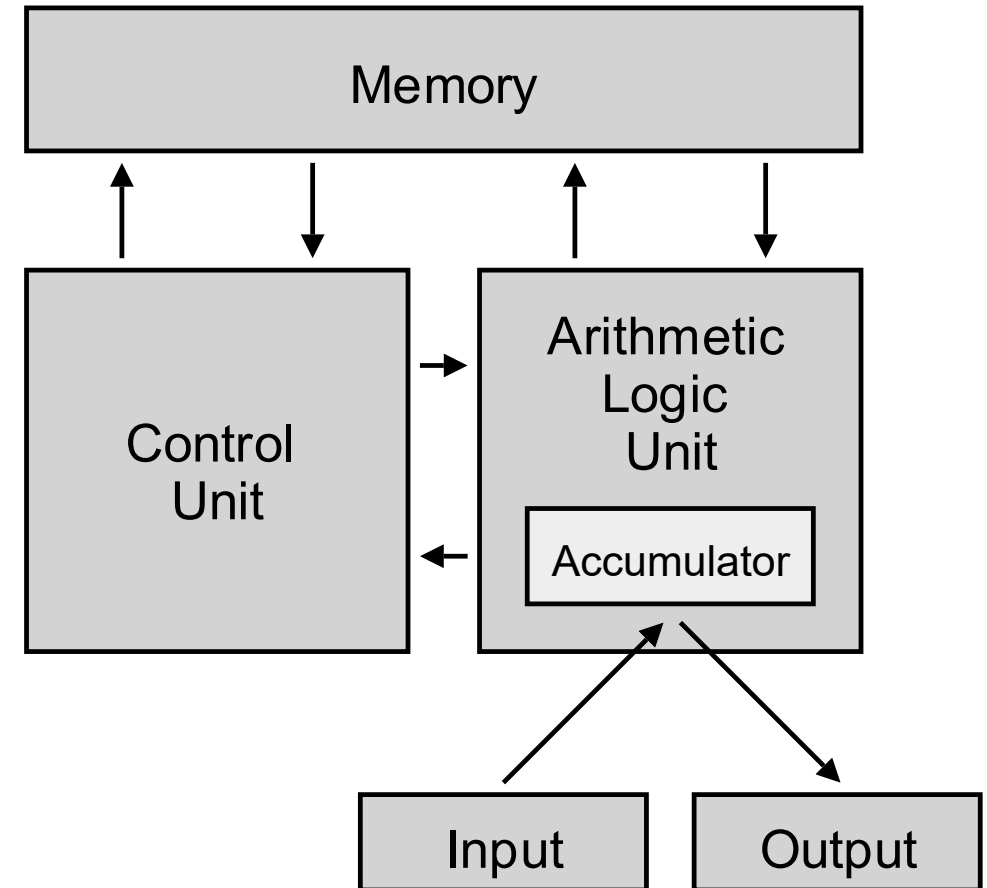
Najważniejsza reguła:

Program (instrukcje) i dane
znajdują się w tej samej
pamięci →

program traktowany jest jak
zwykłe dane →

te same rozkazy mogą
modyfikować program
(samo-modyfikacja kodu)

Źródło: Wikipedia



79

Architektura komputera – model von Neumanna

Główne komponenty architektury:

- Pamięć (jedna wspólna) → kod programu i dane w tej samej przestrzeni adresowej
- Procesor (CPU)
 - Jednostka sterująca (CU) – pobiera i dekoduje instrukcje
 - Jednostka arytmetyczno-logiczna (ALU) – wykonuje obliczenia
 - Rejestry (PC, IR, akumulator, rejestry robocze)
- Magistrala systemowa (jedna główna)
 - Address Bus – adres → Data Bus – dane i instrukcje → Control Bus – sygnały sterujące (czytanie/zapis, przerwania)
- Wejście / Wyjście (I/O) → przez tę samą magistralę lub dedykowane kontrolery

80

Architektura komputera – model von Neumanna

Cykl wykonywania instrukcji (powtarzany w każdej instrukcji):

1. Pobranie instrukcji: licznik programu wskazuje adres → instrukcja jest odczytywana z pamięci i umieszczana w rejestrze rozkazów
2. Dekodowanie instrukcji: jednostka sterująca analizuje kod operacji i określa, co należy zrobić oraz które dane (operandy) będą potrzebne
3. Wykonanie operacji: jednostka arytmetyczno-logiczna wykonuje obliczenia lub operację logiczną albo realizuje transfer danych
4. Dostęp do pamięci lub zapis wyniku: jeśli to konieczne – odczyt operandów z pamięci lub zapis wyniku obliczeń z powrotem do pamięci / rejestru
5. Aktualizacja licznika programu: licznik programu zwiększany jest o 1 (następna instrukcja) lub ustawiany na nowy adres (w przypadku skoku) → cykl zaczyna się od nowa

81

Architektura komputera – model von Neumanna

Wąskie gardło architektury von Neumanna: procesor musi na zmianę pobierać instrukcje i dane przez jedną wspólną magistralę

Skutki:

- procesor często czeka na dostęp do pamięci
- magistrala staje się najwolniejszym elementem systemu
- ogranicza maksymalną wydajność procesora

Współczesne sposoby zmniejszania problemu:

- oddzielne pamięci podręczne dla instrukcji i danych
- przewidywanie i wcześniejsze pobieranie instrukcji
- wykonywanie wielu operacji równoległe w jednym procesorze

82

Architektura komputera – model von Neumanna

Zalety

prosta i tania budowa (jedna pamięć, jedna magistrala)

bardzo duża elastyczność i uniwersalność

program może modyfikować sam siebie

łatwe pisanie programów, kompilatorów i systemów operacyjnych

efektywne wykorzystanie pamięci (brak sztywnego podziału na kod/dane)

ogromny ekosystem narzędzi, bibliotek i procesorów

Wady

wąskie gardło magistrali – główne ograniczenie wydajności

wolniejszy dostęp do danych niż w architekturach z oddzielnymi magistralami

ryzyko przypadkowego nadpisania kodu lub danych innego programu

trudniej zapewnić stały, przewidywalny czas wykonania (real-time)

większe zużycie energii przy ciągłym pobieraniu instrukcji i danych

83

Architektura komputera – model von Neumanna

Architektura von Neumanna obecnie - dominujące procesory oparte na tej koncepcji:

- procesory x86 i x86-64 (Intel Core, AMD Ryzen, serwery)
- wysokowydajne rdzenie ARM (Apple M-seria, Qualcomm Snapdragon, serwery)
- większość procesorów do serwerów i komputerów osobistych
- wiele średniej i wyższej klasy mikrokontrolerów

Najważniejsze modyfikacje:

- oddzielne pamięci podręczne dla instrukcji i danych
- wykonywanie poza kolejnością (out-of-order)
- przewidywanie rozgałęzień
- bardzo szerokie jednostki obliczeniowe wewnątrz procesora

84

Architektura komputera – podstawowe komponenty komputera

Struktura blokowa komputera cyfrowego

Główne bloki:

- Procesor – wykonuje programy i obliczenia
- Pamięć operacyjna – przechowuje aktualnie wykonywany kod i dane
- Pamięć masowa – trwale zapisuje pliki i programy
- Podsystem wejścia-wyjścia – komunikuje się z urządzeniami zewnętrznymi
- Magistrala systemowa – łączy wszystkie komponenty (adres, dane, sterowanie)
- Zasilacz i układ zegara – zasilają i synchronizują cały system

85

Architektura komputera – podstawowe komponenty komputera

Procesor – podstawowe jednostki:

- Jednostka sterująca – dekoduje rozkazy, generuje sygnały sterujące
- Jednostka arytmetyczno-logiczna – wykonuje operacje +, −, ×, logiczne, przesunięcia
- Zespół rejestrów – licznik programu, rejestr rozkazów, rejestry adresowe, rejestry ogólnego przeznaczenia, rejestr stanu (flagi)
- Jednostka zarządzania pamięcią – translacja adresów wirtualnych na fizyczne
- Pamięci podręczne wewnętrzne – poziom 1 dla rozkazów i danych

Architektura komputera – podstawowe komponenty komputera

Płyta główna – elementy łączące

- Gniazdo procesora
- Gniazda pamięci operacyjnej (DIMM / SO-DIMM)
- Złącza rozszerzeń (PCIe x16, x8, x4, x1)
- Złącza pamięci masowej (M.2, SATA)
- Układ zasilania procesora (VRM – moduły regulacji napięcia)
- Chipset lub mostki komunikacyjne
- Kontrolery peryferyjne (USB, audio, sieć)

87

Architektura komputera – podstawowe komponenty komputera

Magistrala systemowa – rodzaje:

- Magistrala adresowa – określa numer komórki pamięci lub urządzenia (40-52 bity)
- Magistrala danych – przenosi rozkazy i dane (64-512 bitów, najczęściej 128-256 wewnątrznie)
- Magistrala sterująca – sygnały: odczyt/zapis, pamięć/We/Wy, zegar, reset, przerwanie, żądanie DMA, gotowość

PCIe (PCI Express) – główna magistrala rozszerzeń:

- Składa się z linii (lanes) – każda linia to para przewodów (TX + RX)
- Typowe konfiguracje slotów: $\times 1$, $\times 4$, $\times 8$, $\times 16$ (liczba linii)
- Współczesne procesory oferują 16-28 linii PCIe bezpośrednio z CPU (np. $\times 16$ na kartę graficzną + $\times 4$ na SSD NVMe)

88

Architektura komputera – podstawowe komponenty komputera

Pamięć operacyjna – cechy techniczne

- Typ: DRAM (Dynamiczna Pamięć Losowego Dostępu)
- Czas dostępu: 50-120 ns
- Przepustowość: 50-400+ GB/s (zależnie od liczby kanałów i taktowania)
- Szerokość słowa: 64 bity na kanał
- Tryby: burst, prefetch, dual-/quad-channel
- Ulotna – dane giną po wyłączeniu zasilania

Architektura komputera – podstawowe komponenty komputera

Pamięć operacyjna – hierarchia pamięci – czasy dostępu

Poziom	Czas dostępu	Pojemność typowa (2026)	Technologia
Rejestry	< 0,5 ns	kilkaset bajtów	flip-flopy
Pamięć podręczna L1	0,5-2 ns	32-256 KB na rdzeń	SRAM
Pamięć podręczna L2/L3	3-40 ns	1-128+ MB	SRAM
Pamięć operacyjna	50-120 ns	16-1024 GB	DDR5/LPDDR5X
Pamięć masowa	10 μ s-10 ms	1 TB-100+ TB	NAND Flash

90

Architektura komputera – podstawowe komponenty komputera

Pamięć masowa – dominujące interfejsy (2026)

- NVMe przez PCIe: 4-16 linii (najczęściej ×4 dla SSD konsumenckich), PCIe 5.0: do ~14 GB/s (jednokierunkowo), PCIe 6.0: do ~28 GB/s (high-end i serwery)
- SATA III 6 Gb/s → realnie 550-600 MB/s (starszy standard)
- UFS 4.0 / UFS 5.0 w smartfonach i tabletach: UFS 4.0: 4-5,8 GB/s, UFS 5.0: 10-12 GB/s
- eMMC 5.1 tańsze systemy wbudowane (IoT), maksymalnie 400-600 MB/s (bardzo wolne w porównaniu do UFS/NVMe)

Kontroler hosta – porównanie protokołów:

- NVMe – nowoczesny, niskie opóźnienia, tysiące kolejek, do 64k komend na kolejkę
- AHCI – starszy protokół, wysokie opóźnienia, tylko 1 kolejka, maksymalnie 32 komendy

91

Architektura komputera – podstawowe komponenty komputera

Wejście-wyjście – metody adresowania i transferu

Adresowanie urządzeń We/Wy

- Mapowanie w przestrzeni pamięci (dominujące) – urządzenia jak komórki pamięci
- Oddzielna przestrzeń We/Wy (port-mapped) – tylko starsze x86 (IN/OUT, 64 KB zakresu)

Sposoby komunikacji / synchronizacji

- Programowe (polling) – procesor sprawdza flagę gotowości → wysokie obciążenie CPU
- Przerwaniowe – urządzenie zgłasza przerwanie po zakończeniu → reakcja CPU
- Bezpośredni dostęp do pamięci (DMA) – kontroler DMA przenosi dane bez procesora → dla dużych transferów (dyski, sieć, grafika)

92

Architektura komputera – podstawowe komponenty komputera

DMA – bezpośredni transfer bez procesora – główne etapy:

1. Konfiguracja kanału: adres źródłowy – adres docelowy – długość bloku – kierunek transferu
2. Żądanie DMA: urządzenie wysyła sygnał żądania (DREQ – DMA Request) do kontrolera
3. Przejęcie magistrali: kontroler DMA wysyła sygnał HOLD → procesor odpowiada HLDA (Hold Acknowledge) → magistrala oddana DMA
4. Transfer blokowy: DMA wystawia adresy i wykonuje odczyty/zapisy tryby: ciągły (burst) / kradzież cykli (cycle stealing) / rozproszony (scatter-gather)
5. Zakończenie: licznik długości osiąga zero → DMA wysyła przerwanie (IRQ lub MSI-X – Message Signaled Interrupt) procesor odbiera przerwanie i kończy operację

93

Architektura komputera – podstawowe komponenty komputera

Przerwania w komputerze – powody istnienia przerwań:

- Natychmiastowa reakcja na zdarzenia zewnętrzne (klawiatura, dysk, timer, sieć, USB – bez ciągłego sprawdzania przez procesor)
- Obsługa błędów procesora w locie (dzielenie przez zero, brak strony, nielegalny rozkaz, przepełnienie)
- Wywoływanie jądra systemu operacyjnego (wywołania systemowe: otwórz plik, przydziel pamięć, utwórz proces)
- Realizacja wielozadaniowości (timer przerywa co 10-20 ms → przełączanie kontekstu między procesami/wątkami)
- Oszczędność energii i wydajności (procesor może spać / czekać → budzi go tylko potrzebne zdarzenie)

94

Architektura komputera – podstawowe komponenty komputera

Działanie przerwania – podstawowy mechanizm:

- Zdarzenie: urządzenie wysyła sygnał sprzętowy / procesor wykrywa błąd / rozkaz syscall
- Zapis stanu bieżącego: procesor automatycznie zapisuje licznik programu i rejestr stanu na stosie
- Przejście do procedury obsługi: według wektora przerw (numer → adres procedury ISR – Interrupt Service Routine)
- Wyłączenie dalszych przerw: aby uniknąć zagnieżdżania w krytycznych miejscach
- Wykonanie procedury obsługi przerw (ISR): odczytuje dane, reaguje, sygnalizuje zakończenie (np. EOI – End Of Interrupt)
- Powrót z przerwania: przywrócenie starego stanu (licznik programu + flagi) → kontynuacja programu

95

Architektura komputera – podstawowe komponenty komputera

Układ zegara – synchronizacja systemu

Źródła zegara:

- Oscylator kwarcowy (najczęściej 14,318 MHz lub 25/48 MHz)
- Generator MEMS (tańszy, mniejszy, coraz popularniejszy w laptopach)

Podwyższanie częstotliwości: Mnożniki fazowe (PLL – Phase-Locked Loop) →
wejście 25-100 MHz → wyjście 1-6 GHz (per rdzeń)

Dystrybucja zegara:

- Globalna sieć zegarowa (board-level)
- Lokalne domeny zegarowe (per rdzeń, per blok IP)
- Clock gating – wyłączanie zegara w nieużywanych częściach (oszczędność energii)

96

Architektura komputera – organizacja i działanie procesora

Procesor składa się z kilku kluczowych bloków:

- Jednostka sterująca (Control Unit – CU) – zarządza kolejnością wykonywania rozkazów i generuje sygnały sterujące dla pozostałych części procesora
- Jednostka arytmetyczno-logiczna (Arithmetic Logic Unit – ALU) – wykonuje operacje matematyczne i logiczne (dodawanie, odejmowanie, porównywanie, AND, OR, itp.)
- Zestaw rejestrów – bardzo szybka pamięć wewnątrz procesora (licznik programu, rejestr rozkazów, rejestry ogólnego przeznaczenia, rejestr stanu)
- Pamięci podręczne (cache) – małe, bardzo szybkie pamięci na poziomie 1, 2 i 3, które przechowują najczęściej używane dane i instrukcje
- Jednostka pobierania i dekodowania rozkazów – odpowiedzialna za odczyt instrukcji z pamięci i ich wstępne przetworzenie 97

Architektura komputera – organizacja i działanie procesora

W najprostszym modelu procesor wykonuje całą instrukcję w jednym cyklu zegara

Kolejność działań:

- Pobranie instrukcji (Instruction Fetch – IF) procesor odczytuje instrukcję z pamięci pod adresem wskazanym przez licznik programu
- Dekodowanie instrukcji (Instruction Decode – ID) jednostka sterująca analizuje kod rozkazu i określa, co należy zrobić oraz jakie dane będą potrzebne
- Wykonanie instrukcji (Execute – EX) jednostka arytmetyczno-logiczna wykonuje obliczenia lub operację logiczną
- Zapis wyniku (Write Back – WB) wynik zapisywany jest do rejestru lub pamięci

98

Architektura komputera – organizacja i działanie procesora

W praktyce większość instrukcji wymaga kilku cykli zegara. Przykładowe czasy:

- Proste operacje na rejestrach (dodaj, odejmij) – 1-2 cykle
- Operacje ze skokiem warunkowym – 2-5 cykli
- Instrukcje dostępu do pamięci (odczyt/zapis) – 3-15 cykli
- Mnożenie i dzielenie – 3-30 cykli (zależnie od implementacji)
- Instrukcje zmiennoprzecinkowe – 4-20 cykli

Potokowanie (pipelining) – podział wykonywania instrukcji na kilka etapów → w każdym cyklu zegara: pobiera się nową instrukcję, jednocześnie kończy się poprzednia → przetwarzanie wielu rozkazów równolegle (w różnych fazach)

Architektura komputera – organizacja i działanie procesora

Potok dzieli wykonanie instrukcji na kilka etapów – każdy etap trwa jeden cykl zegara

Klasyczny 5-etapowy potok:

- Pobranie instrukcji (Instruction Fetch – IF)
- Dekodowanie i odczyt rejestrów (Instruction Decode – ID)
- Wykonanie operacji / obliczenie adresu (Execute – EX)
- Dostęp do pamięci (Memory Access – MEM)
- Zapis wyniku do rejestru (Write Back – WB)

W idealnym przypadku co cykl zegara ukończona jest jedna instrukcja

100

Architektura komputera – organizacja i działanie procesora

Zalety:

- Zwiększenie przepustowości – teoretycznie IPC (Instructions Per Cycle) ≈ 1
- Możliwość wyższego taktowania zegara (każdy etap krótszy)
- Lepsze wykorzystanie zasobów procesora

Wady i problemy:

- Zagrożenia (hazards) powodują puste cykle (bubbles / stalls)
- Zagrożenia strukturalne – brak zasobów
- Zagrożenia danych – zależności między instrukcjami
- Zagrożenia sterowania – skoki warunkowe zaburzają potok

101

Architektura komputera – organizacja i działanie procesora

Zagrożenia w potoku – trzy główne typy:

- Zagrożenia strukturalne – dwie instrukcje chcą jednocześnie korzystać z tego samego zasobu (np. jednej pamięci na pobranie instrukcji i dostęp do danych)
- Zagrożenia danych
 - RAW (Read After Write) – instrukcja czyta wartość, zanim poprzednia ją zapisała
 - WAR (Write After Read) – instrukcja zapisuje do rejestru, zanim poprzednia go odczytała
 - WAW (Write After Write) – dwie instrukcje kolejno zapisują do tego samego rejestru
- Zagrożenia sterowania skok warunkowy – nie wiemy, czy skoczyć, czy nie → potok musi czekać na decyzję

102

Architektura komputera – organizacja i działanie procesora

Rozwiązania zagrożeń danych - najważniejsze techniki:

- Przesyłanie wyniku bezpośrednio (forwarding / bypassing) – wynik z etapu EX lub MEM jest natychmiast przekazywany do następnej instrukcji
- Wstrzymanie potoku (stall / insert bubble) – procesor czeka kilka cykli
- Wykonywanie poza kolejnością (out-of-order execution) – nowoczesne procesory wykonują rozkazy wtedy, gdy dane są gotowe
- Zmiana nazw rejestrów (register renaming) – usuwa sztuczne zależności między instrukcjami

Architektura komputera – organizacja i działanie procesora

I1: ADD R1, R2, R3 : $R1 = R2 + R3$

I2: SUB R4, R5, R6 : $R4 = R5 - R6$

I3: AND R7, R8, R9 : $R7 = R8 \text{ AND } R9$

Cykl	I1: ADD R1, R2, R3	I2: SUB R4, R5, R6	I3: AND R7, R8, R9
1	IF		
2	ID	IF	
3	EX	ID	IF
4	MEM	EX	ID
5	WB	MEM	EX
6		WB	MEM

104

Architektura komputera – organizacja i działanie procesora

I1: ADD R1, R2, R3 : $R1 = R2 + R3$

I2: SUB R4, R1, R5 : $R4 = R1 - R5$

Cykl	ADD R1, R2, R3	SUB R4, R1, R5
1	IF	
2	ID	IF
3	EX	ID
4	MEM	stall
5	WB	stall
6		EX
7		MEM
8		WB

105

Architektura komputera – organizacja i działanie procesora

I1: ADD R1, R2, R3 : $R1 = R2 + R3$

I2: SUB R4, R1, R5 : $R4 = R1 - R5$

Cykl	ADD R1, R2, R3	SUB R4, R1, R5
1	IF	
2	ID	IF
3	EX	ID
4	MEM	EX
5	WB	MEM
6		WB

106

Architektura komputera – organizacja i działanie procesora

Współczesne procesory mają wielopoziomową hierarchię cache:

- L1 (najbliżej rdzenia) • Najszybsza, najmniejsza • Oddzielna dla instrukcji (L1I) i danych (L1D) • Pojemność: 32-128 KB na rdzeń • Czas dostępu: ~0,5-2 ns (1-4 cykle zegara) • Bardzo wysoka częstotliwość trafień (hit rate > 90-95%)
- L2 • Na każdy rdzeń (prywatna) • Pojemność: 512 KB - 3 MB na rdzeń • Czas dostępu: ~3-15 ns (10-40 cykli) • Hit rate: 50-80%
- L3 (ostatni poziom – LLC) • Wspólna dla wszystkich rdzeni • Pojemność: 20-128+ MB (serwerowe 300-500 MB) • Czas dostępu: ~15-60 ns (40-150 cykli) • Hit rate: 20-60%

Architektura komputera – organizacja i działanie procesora

Najczęściej stosowane polityki zapisu:

- write-back – dane najpierw zapisuje się tylko do cache, do pamięci głównej trafiają dopiero przy usunięciu bloku (oszczędność ruchu na magistrali)
- write-allocate – przy zapisie do pamięci, której bloku nie ma w cache → najpierw ładuje się blok z pamięci głównej

Procesor	L1 (na rdzeń)	L2 (na rdzeń / grupa)	L3 / SLC (ostatni poziom)
Intel Core Ultra 9 (Arrow Lake)	48-96 KB dane + 32-64 KB instrukcje	2-3 MB na rdzeń	36-48 MB wspólne dla wszystkich rdzeni
AMD Ryzen 9 9950X (Zen 5)	32 KB instrukcje + 48 KB dane	1 MB na rdzeń	64-128 MB wspólne
Apple M4 / M4 Max	192 KB instrukcje + 128 KB dane (na rdzeń wydajny)	16-32 MB wspólne dla grupy rdzeni	SLC 64-128 MB (system level cache – wspólne dla całego SoC)

108

Architektura komputera – organizacja i działanie procesora

Najważniejsze grupy rejestrów:

- Licznik programu (Program Counter – PC lub Instruction Pointer – IP) przechowuje adres następnej instrukcji do wykonania
- Rejestr rozkazów (Instruction Register – IR) przechowuje pobraną i dekodowaną instrukcję
- Rejestry ogólnego przeznaczenia (General Purpose Registers – GPR) służą do tymczasowego przechowywania danych, wyników obliczeń i adresów
- Rejestr stanu / flag (Flags Register / Program Status Register – PSR) zawiera flagi: zero (Z), przeniesienie (C), znak (N), przepełnienie (V/O)
- Wskaźnik stosu (Stack Pointer – SP) wskazuje na wierzchołek stosu – służy do wywołań funkcji i zapisywania stanu przy przerwaniach

Architektura komputera – organizacja i działanie procesora

Program Status Word (PSW) – specjalny rejestr, który przechowuje najważniejsze informacje o aktualnym stanie procesora

Najważniejsze bity / flagi w PSW:

- Flaga Z (Zero) – wynik ostatniej operacji równy zero
- Flaga C (Carry / Przeniesienie) – wystąpiło przeniesienie poza najstarszy bit
- Flaga N (Negative / Znak) – wynik ujemny (najstarszy bit = 1)
- Flaga V lub O (Overflow / Przepelnienie) – wynik przekroczył zakres reprezentacji
- Flaga I (Interrupt enable) – czy przerwania są dozwolone
- Bity trybu pracy

PSW jest aktualizowany automatycznie po większości operacji arytmetycznych i logicznych

110

Architektura komputera – organizacja i działanie procesora

Procesor może pracować w dwóch podstawowych trybach:

Tryb użytkownika (User mode): zwykłe programy (przeglądarka, gra, edytor), ograniczony dostęp do zasobów, zakazane instrukcje: wyłączanie przerw, zmiana mapowania pamięci, bezpośredni dostęp do portów I/O, próba wykonania zakazanej instrukcji → wyjątek

Tryb jądra (Kernel mode / Privileged mode): pełny dostęp do wszystkich zasobów i instrukcji, pracuje tu jądro systemu operacyjnego, sterowniki, procedury obsługi przerw i wyjątków

Przełączanie trybów: z użytkownika do jądra: wywołanie systemowe (syscall / SVC / INT), przerwanie sprzętowe, wyjątek

Przełączanie trybów: z jądra do użytkownika: specjalna instrukcja powrotu (IRET / ERET / RTI) ¹¹¹

Architektura komputera – organizacja i działanie procesora

- **wywołanie systemowe** (*system call, syscall*)
program celowo żąda usługi systemu operacyjnego, np. otwarcia pliku, zapisu danych lub utworzenia procesu
- **wyjątek** (*exception*), np. błąd strony / wyjątek braku strony (*page fault*)
procesor wykrywa problem podczas wykonania instrukcji, np. brak strony w pamięci RAM albo naruszenie ochrony pamięci
- **przerwanie sprzętowe** (*hardware interrupt*), np. przerwanie od zegara systemowego (*timer interrupt*)
sprzęt zgłasza zdarzenie asynchroniczne, dzięki czemu system operacyjny może odzyskać kontrolę nad CPU, aktualizować czas i planować wykonanie procesów

Wspólny mechanizm: zapis stanu procesora, przejście do trybu jądra, wykonanie procedury obsługi, powrót do bieżącego procesu albo przełączenie kontekstu

112

Dziękuję za uwagę

Treści programowe:

Definicje procesów i wątków, tworzenie,
zarządzanie i zakończenie procesów
oraz synchronizacja i komunikacja
między procesami



114

Architektura komputera – definicje procesów i wątków

Program – plik na dysku (np. exe, elf) – zawiera kod maszynowy oraz dane statyczne potrzebne do uruchomienia

Proces – uruchomiona instancja programu wraz z przydzielonymi zasobami systemu operacyjnego (m.in. własna przestrzeń adresowa pamięci, PID, stan, uchwyty/deskryptory otwartych plików)

Wątek – najmniejsza jednostka wykonywania w procesie: pojedynczy strumień instrukcji planowany przez system, który ma własny licznik rozkazów i stos, ale współdzieli pamięć procesu z innymi wątkami

Architektura komputera – definicje procesów i wątków

System operacyjny opisuje proces w strukturze **PCB** (Process Control Block – blok sterowania procesem), która zawiera:

PID (Process Identifier) oraz PPID (Parent Process Identifier): pozwalają jednoznacznie wskazać proces i jego relacje

Stan procesu: czy proces ma aktualnie dostęp do CPU, czy czeka na zdarzenie

Kontekst CPU: PC (Program Counter) i rejestry procesora zapisują gdzie i jak proces był wykonywany, żeby dało się go wznowić po przełączeniu

Pamięć procesu: informacje o przestrzeni adresowej, mapowaniu stron i limitach zużycia pamięci

Zasoby i I/O: lista otwartych plików (deskryptory/uchwyty), operacje wejścia-wyjścia

Sygnały: które czekają na obsługę i które są zablokowane przez proces

Planowanie: priorytet, nice value oraz maska powinowactwa CPU

Statystyki: ile czasu procesora i pamięci proces już zużył (monitoring, limity, rozliczanie)

116

Architektura komputera – definicje procesów i wątków

Wątek = strumień wykonania procesu – blok sterowania przechowuje tylko dane niezbędne do kontynuacji instrukcji, zawiera:

TID (Thread Identifier): numer jednoznacznie wskazujący wątek w systemie

PC (Program Counter – licznik rozkazów): adres instrukcji, od której wątek ma kontynuować wykonanie

Rejestry procesora: zapis kontekstu CPU dla wątku, potrzebny przy przełączaniu między wątkami

Stos wywołań (stack): prywatna pamięć wątku na zmienne lokalne, parametry wywołań i adresy powrotu z funkcji

Start wątku: wskaźnik na funkcję startową oraz argument przekazany przy uruchomieniu wątku

Stan wątku: gotowy, wykonywany, zablokowany, zakończony — informacja dla planisty, czy wątek może dostać czas CPU

117

Architektura komputera – definicje procesów i wątków

Podstawowe różnice proces vs wątek:

Przestrzeń adresowa: proces ma własną, a wątki współdzielą przestrzeń adresową procesu

Kod, dane globalne i sterta (heap): w procesach są odseparowane, a wątki w tym samym procesie korzystają ze wspólnego kodu i wspólnych danych

Stos wywołań (stack): każdy wątek ma własny stos

Otwarte pliki: proces ma tabelę deskryptorów/uchwytów, a wątki dzielą ją w obrębie procesu

Koszt tworzenia i przełączania: proces zajmuje więcej czasu (typowo ~0,1–2 ms), a wątek zajmuje mniej czasu (typowo ~5-50 μ s)

Komunikacja: między procesami jest trudniejsza (komunikacja międzyprocesowa: potoki, pamięć współdzielona, kolejki), a między wątkami jest łatwiejsza (wspólna pamięć)

118

Architektura komputera – tworzenie procesu

Linux (Unix) – model fork() + exec():

```
pid_t pid = fork();  
if (pid == 0) { // to jest dziecko  
    execl("/usr/bin/ls", "ls", "-la", NULL);  
    // jeśli exec się nie uda  
    _exit(127); }  
// rodzic kontynuuje  
waitpid(pid, &status, 0);
```

119

Architektura komputera – tworzenie procesu

Linux (Unix) – model fork() + exec():

1. Proces rodzic wywołuje funkcję fork(): jądro systemu tworzy nowy proces – dziecko – prawie dokładna kopia rodzica, dziecko dostaje nowy numer PID, rodzic i dziecko kontynuują wykonywanie od miejsca zaraz po fork()
2. Pamięć jest kopiowana dzięki mechanizmowi Copy-on-Write (COW – kopiowanie przy zapisie), na początku strony pamięci są współdzielone między rodzicem a dzieckiem, gdy jeden z procesów coś zmieni w pamięci → system robi prawdziwą kopię strony
3. Jeden z procesów (najczęściej dziecko) wywołuje funkcję exec() / execve() – cały dotychczasowy program zostaje zastąpiony nowym programem, ładowany jest nowy kod, nowe biblioteki, nowy stos, przekazywane są argumenty (argv) i zmienne środowiskowe (env), numer PID pozostaje ten sam, otwarte pliki są dziedziczone
4. Rodzic czeka na zakończenie dziecka – wywołuje wait() lub waitpid(), odbiera kod powrotu dziecka

120

Architektura komputera – tworzenie procesu

Linux (Unix) – funkcja fork():

- Tworzy prawie identyczną kopię procesu rodzica
- Używa mechanizmu copy-on-write (kopia pamięci dopiero przy pierwszej modyfikacji)
- Dziecko dostaje nowy numer PID, a jego rodzicem staje się rodzic
- Zwraca: 0 w dziecku, PID dziecka w rodzicu, -1 w przypadku błędu
- Dziedziczy prawie wszystko: otwarte pliki, zmienne środowiskowe, bieżący katalog, maskę sygnałów
- Nie dziedziczy: nowe numer PID i PPID, oczekujące sygnały (nieprzetworzone), blokady plików (flock), timery alarmowe, pamięć zablokowaną w RAM (mlock)

121

Architektura komputera – tworzenie procesu

Linux (Unix) – funkcje z rodziny `exec()`:

- **`execl`** – argumenty = *lista* parametrów funkcji zakończona wskaźnikiem `NULL`
- **`execvp`** – argumenty = *tablica* (`argv[]`), plik wykonywalny jest wyszukiwany w `PATH`
- **`execve`** – wariant „bazowy”: *pełna ścieżka*, *tablica argumentów* (`argv[]`) oraz *tablica środowiska* (`envp[]`)

Wspólna zasada: wszystkie `exec*` zastępują aktualny obraz procesu (kod, dane, stos) nowym programem – nie powstaje nowy PID

- PID zostaje – ten sam proces
- Otwarte pliki zwykle zostają, ale nie te, które mają ustawioną flagę *close-on-exec*
- Środowisko: większość `exec*` dziedziczy bieżące zmienne środowiskowe

122

Architektura komputera – tworzenie procesu

Windows 11 – model CreateProcess():

```
STARTUPINFO si = { sizeof(si) };  
PROCESS_INFORMATION pi;  
wchar_t polecenie[] = L"notepad.exe";  
CreateProcessW(NULL, polecenie, NULL, NULL, TRUE, 0, NULL, NULL,  
&si, &pi);  
WaitForSingleObject(pi.hProcess, INFINITE);
```

123

Architektura komputera – tworzenie procesu

Windows 11 – model CreateProcess():

1. Proces nadrzędny (Notatnik) wywołuje funkcję CreateProcess() (lub pośrednio ShellExecute – np. kliknięcie pliku .exe)
2. System Windows tworzy: nowy obiekt procesu, główny wątek (primary thread) nadaje nowe numery PID i TID, przygotowuje podstawowe struktury zarządzania
3. Program ładujący systemu (loader) ładuje plik wykonywalny (.exe): mapuje sekcje pliku do pamięci (kod, dane, sterta), ładuje wszystkie potrzebne biblioteki DLL (kernel32.dll, user32.dll itp.), przygotowuje stos i środowisko uruchomieniowe (argumenty, zmienne środowiskowe)
4. Konfigurowane są uchwyt (handles): mogą być dziedziczone, ustawiany jest bieżący katalog roboczy, ustawiane są zmienne środowiskowe, uprawnienia
5. Proces zostaje uruchomiony: system uruchamia główny wątek, wykonanie zaczyna się od punktu wejścia programu (zwykle main() lub WinMain()): gotowy → wykonywany

Architektura komputera – tworzenie procesu

Cecha	Linux (fork + exec)	Windows 11 (CreateProcess)
Wywołanie	fork() → exec()	CreateProcess() lub pośrednio ShellExecute
Mechanizm tworzenia	Najpierw dokładna kopia rodzica (niemal identyczny proces)	Tworzy proces od zera – nowa, czysta przestrzeń adresowa
Kopiowanie pamięci	Copy-on-Write - kopiowane dopiero przy zapisie	Pełna inicjalizacja
Dziedziczenie zasobów	Automatyczne: otwarte pliki, zmienne środowiskowe, katalog roboczy, maska sygnałów	Domyślnie nie dziedziczy – tylko jeśli jawnie ustawiono flagę
Ładowanie nowego programu	exec() zastępuje obraz procesu nowym programem	Loader systemu (Ldrp) ładuje .exe, mapuje sekcje, ładuje wszystkie DLL
Główny wątek	Jeden wątek po wywołaniu fork() – ten który wywołał	Tworzony od zera nowy główny wątek (primary thread)
Koszt pamięci na starcie	Bardzo niski (tylko markery stron + nowa struktura PCB)	Wysoki (pełna alokacja pamięci + ładowanie DLL-i)
Typowe zastosowanie	Powłoka (bash), demony, skrypty, CGI, kontenery	Aplikacje GUI, usługi systemowe, instalatory

125

Architektura komputera – tworzenie wątku

Tworzenie wątków w Linuksie – typowy schemat (POSIX Threads)

Najczęściej używa się biblioteki pthread (POSIX Threads – wątki standardu POSIX):
funkcja `pthread_create()` tworzy nowy wątek w obrębie bieżącego procesu

Podaje się funkcję startową (adres funkcji) i argument, a biblioteka przygotowuje stos wątku (stack – stos wywołań) oraz wpis w strukturach planisty

Wątek startuje współdzieląc kod, dane globalne, stertę (heap) i otwarte pliki procesu, ale ma własny licznik rozkazów (PC), rejestry i stos

Zakończenie i synchronizacja:

- `pthread_join()` – czekanie na zakończenie wątku i odbiór wyników
- `pthread_detach()` – odłączanie wątku – system sprząta zasoby po zakończeniu

126

Architektura komputera – tworzenie wątku

Tworzenie wątków w Linuksie – typowy schemat (POSIX Threads)

```
void* moja_funkcja(void* arg) {  
    long numer = (long)arg;  
    printf("Wątek nr %ld działa\n", numer);  
    return NULL;}  
  
pthread_t moj_watek;  
pthread_create(&moj_watek, NULL, moja_funkcja, (void*)7);  
pthread_join(moj_watek, NULL); // czekamy aż wątek się skończy
```

127

Architektura komputera – tworzenie wątku

Tworzenie wątków w Windows 11 – typowy schemat (WinAPI)

Wątki tworzy się przez API systemowe: funkcja `CreateThread()` – tworzy nowy wątek w bieżącym procesie

Start wątku = wskazana funkcja startowa i parametr; system zakłada stos wątku i rejestruje wątek w strukturach planisty

Wątek współdzieli zasoby procesu (pamięć, pliki/uchwyty), a ma własny kontekst CPU (licznik rozkazów i rejestry) i własny stos

Synchronizacja z zakończeniem:

- `WaitForSingleObject()` / `WaitForMultipleObjects()` – czekanie na zakończenie wątku
- `GetExitCodeThread()` – odczyt kodu zakończenia
- `CloseHandle()` – zwolnienie uchwytu do wątku

128

Architektura komputera – tworzenie wątku

Tworzenie wątków w Windows 11 – typowy schemat (WinAPI)

```
unsigned long __stdcall moja_funkcja(void* arg) {  
    long numer = (long)arg;  
    printf("Wątek nr %ld działa\n", numer);  
    return 0;}
```

```
HANDLE uchwyty = CreateThread(NULL, 0, moja_funkcja, (void*)7, 0,  
NULL);
```

```
WaitForSingleObject(uchwyty, INFINITE); // zakończenie
```

```
CloseHandle(uchwyty); // zamknięcie uchwyty
```

129

Architektura komputera – stany wątku

Stany wątku – najczęstsze:

- Nowy – wątek dopiero został utworzony
- Gotowy – jest przygotowany do pracy, ale czeka na swoją kolej do procesora
- Wykonywany – właśnie działa na procesorze
- Zablokowany / oczekujący – nie może działać, bo czeka na coś (np. dane z dysku/sieci, zwolnienie blokady, sygnał)
- Zakończony – skończył pracę i już nie wykonuje instrukcji
- Zawieszony – ktoś go specjalnie wstrzymał (często w Windows, np. debugger lub system)

130

Architektura komputera – przełączanie kontekstu wątku

1. Zapis rejestrów bieżącego procesu lub wątku
2. Zapis stanu jednostek zmiennoprzecinkowych i wektorowych (jeśli używane)
3. Zmiana wskaźnika na nowy blok sterowania procesem
4. Zmiana rejestru wskazującego tablicę stron (CR3 na procesorach x86)
5. Częściowe wyczyszczenie pamięci podręcznej tłumaczeń adresów (TLB)
6. Załadowanie nowych rejestrów
7. Powrót do trybu użytkownika i kontynuacja wykonania

131

Architektura komputera – przełączanie kontekstu wątku

Koszt przełączania kontekstu (wartości przybliżone, ok. 2026):

- Przełączanie między procesami: typowo $\sim 1\text{-}5\ \mu\text{s}$
 - większy – dochodzi do zmiany przestrzeni adresowej (mapowania pamięci procesu)
 - większe straty na TLB (Translation Lookaside Buffer – bufor translacji adresów), nowy proces = inne dane/kod)
- Przełączanie między wątkami w tym samym procesie: typowo $\sim 0,2\text{--}1,2\ \mu\text{s}$
 - mniejszy – ta sama przestrzeń adresowa $\rightarrow$ bez kosztu zmiany mapowania pamięci
 - często wystarcza zapis/odtworzenie rejestrów + PC (Program Counter – licznik rozkazów) + SP (Stack Pointer – wskaźnik stosu)
- Co najsilniej wpływa na koszt (dlaczego widełki są szerokie):
 - migracja na inny rdzeń, obciążenie systemu i polityka planisty, mechanizmy bezpieczeństwa/izolacji

132

Architektura komputera – proces zombie

Zombie – proces zakończony (exit), którego rodzic nie odebrał jeszcze statusu zakończenia przez wait() / waitpid()

Zombie nie wykonuje kodu i nie trzyma RAM programu, ale zostawia wpis w jądrze: PID, kod powrotu, statystyki → zajmuje miejsce w tabeli procesów

Jak powstaje: fork() → dziecko kończy → jądro trzyma status → brak wait() → stan zombie

Dlaczego pojawia się stan zombie: jądro musi zachować exit status procesu, aż rodzic go odbierze przez wait()/waitpid()

Skutek uboczny: dużo zombie może wyczerpać pulę PID / wpisy procesów

Jak znika: rodzic robi wait() (wpis usunięty), albo rodzic ginie → dziecko przejmuje PID 1, który odbierze status po zakończeniu

Architektura komputera – proces zombie

W Windows nie ma procesów zombie w sensie Unix/Linux:

Po zakończeniu procesu system ustawia go w stan signaled (można na niego czekać), a jego zasoby są zwalniane zgodnie z modelem obiektów jądra

Proces jako obiekt jądra może nadal „istnieć” po zakończeniu, jeśli ktoś trzyma do niego uchwyt (handle) (np. proces nadzorujący, debugger, narzędzie monitorujące)

Taki „zakończony, ale wciąż referencjonowany” proces: nie wykonuje kodu, jego obiekt (i metadane) może być widoczny, dopóki nie znikną odwołania

Kiedy znika całkowicie: gdy wszystkie uchwyty do obiektu procesu zostaną zamknięte (CloseHandle) i nie ma już referencji w jądrze

Windows – „sprzątanie” nie zależy od wait() jak w Unix — zależy od tego, czy ktoś trzyma uchwyty do procesu ¹³⁴

Architektura komputera – proces osierocony

Proces osierocony = proces, którego rodzic zakończył działanie wcześniej, a proces-dziecko nadal pracuje

- W Linuksie/Unix proces nie znika, tylko zmienia się jego relacja rodzic–dziecko: system musi zapewnić, że ktoś odbierze status zakończenia, gdy dziecko kiedyś się skończy
- Linux: osierocone procesy są „adoptowane” przez PID 1 (zwykle systemd, dawniej init), który potem wykona wait() i posprząta zakończone dzieci
- Konsekwencja: osierocony proces może działać dalej jako „daemon” (usługa w tle), o ile nie jest powiązany z terminalem lub sesją, która została zamknięta
- Windows 11: model rodzic-dziecko jest inny niż w fork()/wait(); proces może działać niezależnie od procesu, który go uruchomił, a „sprzątanie” zasobów po zakończeniu realizuje system (nadzór nad procesami i wątkami wspierają komponenty systemowe, m.in. csrss.exe – Client/Server Runtime Subsystem)

135

Architektura komputera – synchronizacja

Race condition (warunek wyścigu): wynik programu zależy od kolejności przeplotu instrukcji wątków/procesów

Przykład: counter++ nie jest atomowe – to zwykle 3 kroki:

1. odczyt counter z pamięci do rejestru
2. inkrementacja w rejestrze
3. zapis wyniku z rejestru do pamięci

Dwa wątki zrobią to jednocześnie → oba mogą odczytać tę samą starą wartość i zapisać ten sam wynik → utracona aktualizacja

Dotyczy też: list, map, kolejek, buforów, liczników, stanów maszyn, a nawet logów → niezbędna synchronizacja – poprawność (spójność danych) kosztem narzutu czasowego

136

Architektura komputera – sekcja krytyczna

Sekcja krytyczna – fragment kodu, w którym wątek/proces korzysta ze współdzielonego zasobu (np. zmiennej, listy, pliku, bufora), a który musi mieć na wyłączność → wyścig o dane, warunki:

1. Wzajemne wykluczenie: w danym momencie do sekcji krytycznej wchodzi maksymalnie jeden wątek lub proces
2. Postęp: jeśli sekcja jest pusta, ktoś musi wejść – system nie może blokować bez powodu
3. Ograniczony czas oczekiwania: każdy czekający wątek w końcu dostanie swoją kolej (brak głodzenia)
4. Brak założeń o szybkości procesów: rozwiązanie ma działać poprawnie przy różnych prędkościach i przeplotach wykonywania

137

Architektura komputera – Semafor – licznik dostępu do zasobu

Semafor – najstarszy i najczęściej stosowany mechanizm synchronizacji (Edsger Dijkstra 1965) – kontroluje dostęp do zasobu przez licznik N: jednocześnie może wejść maksymalnie N wątków; składa się z dwóch części:

- Licznika – liczba nieujemna
- Kolejki wątków – czekają na wykonanie

Dwie podstawowe operacje:

- czekaj (wait) / P – zmniejsza licznik o 1;
jeśli licznik ≥ 0 – wątek kontynuuje, jeśli licznik < 0 – wątek czeka w kolejce
- sygnalizuj (signal) / V – zwiększa licznik o 1;
jeśli w kolejce ktoś czeka $\rightarrow$ budzi jeden wątek

138

Architektura komputera – Semafor – licznik dostępu do zasobu

Dwa rodzaje semaforów – binarny i licznikowy

Semafor binarny

- Licznik może być tylko 0 lub 1
- Działa prawie jak blokada wzajemnego wykluczania (mutex)
- Najczęściej używany do wzajemnego wykluczania lub sygnalizacji zdarzenia

Semafor licznikowy

- Licznik może być dowolną liczbą ≥ 0
- Kontroluje ograniczoną liczbę zasobów (np. 3 miejsca parkingowe, 5 wolnych buforów)
- Idealny do problemu producenta-konsumenta z wieloma miejscami

139

Architektura komputera – Semafor – licznik dostępu do zasobu

Semafor binarny – działanie:

1. Początkowa wartość semafora = 1 (wolny)
2. Wątek A wywołuje `czekaj()`
licznik: $1 \rightarrow 0$
A wchodzi do sekcji krytycznej
3. Wątek B wywołuje `czekaj()`
licznik: $0 \rightarrow -1$
B idzie do kolejki i czeka
4. Wątek A wywołuje `sygnalizuj()`
licznik: $-1 \rightarrow 0$
System budzi jeden wątek z kolejki (np. B)
5. Wątek B zostaje obudzony i wchodzi do sekcji krytycznej

140

Architektura komputera – Semafor – licznik dostępu do zasobu

Przykład – semafor licznikowy (ograniczona liczba zasobów)

Cel: tylko 3 wątki mogą jednocześnie korzystać z drukarki

semafor drukarki = 3; // 3 wolne drukarki na start

```
void* drukuj_dokument(void* arg) {  
    czekaj(drukarki); // czekaj na wolną drukarkę  
    // drukowanie dokumentu...  
    sleep(5); // symulacja długiego drukowania  
    sygnalizuj(drukarki); // oddaj drukarkę  
    return NULL;  
}
```

141

Architektura komputera – Semafor – problem producent-konsument

- Mamy bufor o rozmiarze N (np. 10 miejsc)
- Producent wkłada elementy do bufora, konsument je wyjmuje
- Problem:
 - producent nie może włożyć, gdy bufor jest pełny
 - konsument nie może wyjąć, gdy bufor jest pusty
- Dwa semafony licznikowe:
 - $\text{puste_miejsca} = N$ – ile jest wolnych miejsc (gdy 0, producent czeka)
 - $\text{pelne_miejsca} = 0$ – ile jest gotowych elementów (gdy 0, konsument czeka)
- Schemat ($P = \text{wait}$, $V = \text{signal}$):
- Producent: $P(\text{puste_miejsca}) \rightarrow \text{wstaw element} \rightarrow V(\text{pelne_miejsca})$
- Konsument: $P(\text{pelne_miejsca}) \rightarrow \text{wyjmij element} \rightarrow V(\text{puste_miejsca})$

142

Architektura komputera – Muteks – blokada wzajemnego wykluczenia

Muteks (Mutex – mutual exclusion – wzajemne wykluczanie) – mechanizm (blokada), który pozwala, aby tylko jeden wątek naraz wykonywał sekcję krytyczną

Podstawowe operacje:

- lock – przejęcie blokady (gdy zajęta, wątek czeka) – zazwyczaj 1
- unlock – zwolnienie blokady i wpuszczenie kolejnego oczekującego – zazwyczaj 0

Muteks ma właściciela: odblokować powinien ten sam wątek, który go zablokował (inaczej błąd logiczny)

Zastosowanie: ochrona współdzielonych danych (liczniki, listy, kolejki, struktury w pamięci)

Architektura komputera – Muteks – blokada wzajemnego wykluczenia

Muteks – mechanizm działania:

Wątek lub proces potrzebuje dostępu do regionu krytycznego → wywołuje funkcję `mutex_lock`:

- Jeśli muteks jest już odblokowany (region krytyczny dostępny) → wywołanie kończy się sukcesem → wątek może wejść do regionu krytycznego
- Jeśli muteks jest zablokowany (region krytyczny niedostępny) → wątek zablokuje się do czasu odblokowania regionu krytycznego (wywołania funkcji `mutex_unlock`)
- Jeśli na muteksie jest zablokowanych wiele wątków → jeden z nich jest wybierany losowo i zakłada blokadę

144

Architektura komputera – Muteks – blokada wzajemnego wykluczenia – Linux

// Statyczna inicjalizacja

```
pthread_mutex_t moja_blokada = PTHREAD_MUTEX_INITIALIZER;
```

// ... w funkcji wątku:

```
pthread_mutex_lock(&moja_blokada); // ZABLOKUJ
```

// SEKCA KRYTYCZNA – tylko jeden wątek może tu być naraz

```
licznik_globalny++; // bezpieczne zwiększenie zmiennej
```

```
tablica[5] = nowa_wartosc; // bezpieczna modyfikacja
```

```
pthread_mutex_unlock(&moja_blokada); // ODBLOKUJ
```

145

Architektura komputera – Muteks – blokada wzajemnego wykluczenia – Windows

```
CRITICAL_SECTION moja_sekcja;    // Deklaracja zmiennej  
InitializeCriticalSection(&moja_sekcja);    // Inicjalizacja  
// ... w funkcji wątku:  
EnterCriticalSection(&moja_sekcja); // WEJDŹ – czekaj, aż sekcja wolna  
// SEKCA KRYTYCZNA – tylko jeden wątek może tu być naraz  
licznik_globalny++; // bezpieczne zwiększenie  
tablica[5] = nowa_wartosc; // bezpieczna modyfikacja  
LeaveCriticalSection(&moja_sekcja); // WYJDŹ – teraz może wejść inny wątek  
DeleteCriticalSection(&moja_sekcja); // zwolnienie zasobów
```

146

Architektura komputera – Muteks – blokada wzajemnego wykluczenia

Cecha	pthread_mutex (Linux/POSIX)	CRITICAL_SECTION (Windows)
Szybkość przy krótkich sekcjach	dobra	bardzo dobra (spinlock + wait)
Międzyprocesowa	tak – jeśli muteks w pamięci współdzielonej	nie – tylko w obrębie jednego procesu
Rekurencyjność	tak (po ustawieniu atrybutu)	tak (domyślnie)
Wymaga uchwytu (HANDLE)	nie	nie
Inicjalizacja	PTHREAD_MUTEX_INITIALIZER lub init	InitializeCriticalSection
Czyszczenie	pthread_mutex_destroy	DeleteCriticalSection

147

Architektura komputera – Muteks i zmienna warunkowa – Linux

Pthreads – pakiet zawierający funkcje do zarządzania muteksami

Pthreads – zawiera również zmienne warunkowe – przesyłają sygnał z jednego wątku do innych wątków, które oczekują na spełnienie pewnego warunku

Pthreads – zmienne warunkowe są stosowane zawsze razem z muteksem – muteks nie wystarcza do czekania (aktywne czekanie = marnowanie CPU, trzymanie muteksa = blokowanie wątków)

Muteks – chroni stan

Zmienna warunkowa – obsługuje sen/budzenie

Budzenie: wątek zmienia stan → wykonuje:

`pthread_cond_signal` – przesłanie sygnału do innego wątku i obudzenie

`pthread_cond_broadcast` – przesłanie sygnału do wielu wątków i obudzenie

148

Architektura komputera – Muteks – problem producent-konsument

- Mamy bufor o rozmiarze N (np. 10 miejsc), producent wkłada elementy do bufora, konsument je wyjmuję, producent nie może włożyć, gdy bufor jest pełny, konsument nie może wyjąć, gdy bufor jest pusty
- Stosujemy:
 - muteks m – chroni bufor i jego stan (np. count, head, tail)
 - zmienna warunkowa not_full – wątki producenta czekają, gdy bufor jest pełny
 - zmienna warunkowa not_empty – wątki konsumenta czekają, gdy bufor jest pusty

Schemat działań (pthread):

Producent: lock(m) → while (bufor_pełny) wait(not_full, m) → wstaw element
→ signal(not_empty) → unlock(m)

Konsument: lock(m) → while (bufor_pusty) wait(not_empty, m) → wyjmij
element → signal(not_full) → unlock(m)

149

Architektura komputera – Monitor

Monitor – sposób budowy bezpiecznego obiektu współdzielonego w programie współbieżnym, składa się z:

1. danych współdzielonych: np. bufor, licznik, lista
2. funkcji/metod: działających na tych danych
3. reguły dostępu: w danej chwili tylko jeden wątek może wykonywać kod wewnątrz monitora

Nie ma potrzeby pamiętania o blokowaniu w wielu miejscach – pilnują metod monitora

Monitor rozwiązuje dwa problemy jednocześnie:

- chroni dane przed jednoczesną modyfikacją przez kilka wątków
- usypia wątek w oczekiwaniu na spełnienie warunku i budzi po spełnieniu (zmienne warunkowe)¹⁵⁰

Architektura komputera – Monitor – sposób działania

Monitor ma wewnętrzną blokadę wzajemnego wykluczania: gdy wątek wchodzi do metody monitora, blokada jest zajmowana, a inne wątki muszą czekać

Jeśli wątek nie może kontynuować, bo warunek nie jest spełniony, używa zmiennej warunkowej:

- czekanie usypia wątek i jednocześnie zwalnia blokadę monitora, żeby inne wątki mogły zmienić stan
- sygnał budzi uśpiony wątek, gdy stan mógł się poprawić

Po wybudzeniu wątek znowu przejmuje blokadę i dopiero wtedy sprawdza warunek ponownie

Warunek sprawdza się w pętli while → po wybudzeniu stan mógł się zmienić, zanim wątek odzyskał procesor

151

Architektura komputera – Monitor – implementacja Linux

W języku C na Linuksie nie ma gotowego typu „monitor”, ale buduje się go z elementów biblioteki wątków:

- blokada wzajemnego wykluczania z biblioteki wątków (mutex)
- zmienne warunkowe z biblioteki wątków (condition variables)

Typowy monitor to struktura lub „klasa”, która zawiera:

- pola: bufor i jego stan (ile elementów, gdzie wstawiać, skąd pobierać)
- blokadę: chroni te pola
- jedną lub dwie zmienne warunkowe: do czekania

Metody monitora, np. put() i get() → schemat: zablokuj, sprawdź warunek, ewentualnie poczekaj, wykonaj operację, obudź innych, odblokuj

152

Architektura komputera – Monitor – implementacja Windows

W Windows najbliższy odpowiednik monitora buduje się z:

- sekcji krytycznej: szybka blokada wzajemnego wykluczania działająca w obrębie jednego procesu
- zmiennych warunkowych systemu Windows: do usypiania i budzenia wątków

Sekcja krytyczna zapewnia, że tylko jeden wątek naraz modyfikuje dane monitora

Zmienna warunkowa pozwala wątkowi zasnąć, gdy warunek nie jest spełniony, i obudzić się, gdy inny wątek zmieni stan

Mechanizm czekania w Windows również zwalnia blokadę na czas snu i przejmuje ją ponownie po wybudzeniu → inne wątki mogą pracować

153

Architektura komputera – Monitor – problem producent-konsument

- Mamy bufor o rozmiarze N (np. 10 miejsc), producent wkłada elementy do bufora, konsument je wyjmuje, producent nie może włożyć, gdy bufor jest pełny, konsument nie może wyjąć, gdy bufor jest pusty
- Monitor udostępnia tylko dwie metody:
`put()` – wstawia element; jeśli bufor jest pełny, wątek producenta czeka
`get()` – pobiera element; jeśli bufor jest pusty, wątek konsumenta czeka
- Monitor wewnątrz pilnuje dwóch warunków:
 - producent może wstawić tylko wtedy, gdy bufor nie jest pełny
 - konsument może pobrać tylko wtedy, gdy bufor nie jest pusty
- Jeśli warunek nie jest spełniony, wątek zasypia na odpowiedniej zmiennej warunkowej i budzi się dopiero, gdy inny wątek zmieni stan bufora

154

Architektura komputera – Monitor – problem producent-konsument

Monitor – put() i get() w buforze o rozmiarze N

- Monitor zapewnia wyłączny dostęp do bufora i stanu (count, head, tail)

- **put()**:

wejdź → dopóki pełny, czekaj → wstaw i zaktualizuj stan → sygnalizuj niepusty → wyjdź

- **get()**:

wejdź → dopóki pusty, czekaj → pobierz i zaktualizuj stan → sygnalizuj niepełny → wyjdź

Czekanie – wątek zasypia i oddaje monitor – drugi wątek może zmienić stan bufora

Architektura komputera – Monitor – problem producent-konsument

Cecha	Semafor	Muteks	Monitor
Do czego służy	Kontrola liczby równoczesnych wejść do zasobu oraz sygnalizacja zdarzeń	Ochrona sekcji krytycznej – dane współdzielone	Bezpieczny obiekt współdzielony: dane + metody + zasady dostępu
Jak działa	Licznik 0..N: gdy 0, wątki czekają; gdy rośnie, wątki mogą wejść	Blokada 0/1: gdy zajęta, wątek czeka; gdy zwolniona, wchodzi kolejny	Wbudowana blokada: tylko jeden wątek naraz w metodach + czekanie na warunki
Ile wątków naraz	N (licznikowy), 1 (binarny)	1	1 w obrębie monitora
Kto może zwolnić	Dowolny wątek	Ten sam wątek, który zablokował	Reguły ukryte w metodach
Najczęstsze błędy	Zła liczba sygnałów: semafor zostaje „za niski” lub „za wysoki” i logika się psuje	Zakleszczenie (dwa wątki blokują różne zasoby i każdy czeka na drugi), zbyt długie trzymanie blokady	Błędne warunki (brak sprawdzania w pętli), zbyt duży monitor ogranicza równoległość
Implementacja	Prymityw systemowy/biblioteczny: licznik + kolejka czekających	Prymityw systemowy/biblioteczny: atomowe operacje + usypianie/budzenie	Najczęściej muteks + zmienne warunkowe, ukryte wewnątrz metod

156

Architektura komputera – szeregowanie procesora

Szeregowanie – decyzje jądra systemu operacyjnego, który wątek ma wykonać się na danym rdzeniu procesora w danym momencie

Jednostka planowania – wątek → proces może mieć wiele wątków

Planista – kod w jądrze, który operuje na zbiorze wątków (w stanie gotowy) i wybiera następny wątek do uruchomienia (najwyższy priorytet) według algorytmu:

Czas odpowiedzi: jak szybko wątek reaguje na zdarzenie

Czas oczekiwania: ile wątek spędza w kolejce gotowych zanim dostanie procesor

Czas realizacji: od uruchomienia zadania do zakończenia

Przepustowość: ile zadań kończy się w jednostce czasu

Sprawiedliwość: brak sytuacji, w której jeden wątek zabiera CPU kosztem innych

Przewidywalność: podobne zadania mają podobne czasy wykonania

157

Architektura komputera – szeregowanie procesora

Planista podejmuje decyzje o zmianie wykonywanego wątku w typowych sytuacjach:

- koniec kwantu czasu po przerwaniu zegarowym
- wątek blokuje się → czeka na wejście/wyjście, blokadę lub warunek
- wątek zostaje wybudzony po zdarzeniu i trafia do kolejki gotowych
- wątek kończy działanie albo pojawia się wątek o wyższym priorytecie

Kwant czasu – przydzielony wątkowi maksymalny odcinek czasu procesora, po którym jądro może odebrać mu procesor i dać go innemu wątkowi

Planowanie wywłaszczające – jądro może przerwać wątek po upływie kwantu czasu lub gdy pojawi się wątek o wyższym priorytecie

Planowanie niewywłaszczające – jądro nie zabiera procesora w trakcie pracy wątku – wątek sam oddaje procesor (zakończenie lub blokada) ¹⁵⁸

Architektura komputera – szeregowanie procesora

Najprostsze strategie planowania procesora

- **Pierwszy przyszedł, pierwszy obsłużony (FIFO)** - wątki wykonywane w kolejności przyścia do kolejki gotowych
 - zaleta: proste i przewidywalne
 - wada: długie zadanie blokuje wiele krótkich
- **Rotacyjne z kwantem czasu** – każdy wątek dostaje procesor na ustalony kwant czasu, potem trafia na koniec kolejki gotowych
 - zaleta: sprawiedliwość – żaden nie trzyma procesora zbyt długo
 - wada: mały kwant = duży narzut przełączeń kontekstu
duży kwant = pierwszy przyszedł, pierwszy obsłużony

159

Architektura komputera – szeregowanie procesora

Najprostsze strategie planowania procesora

- **Najkrótsze zadanie najpierw** – na początku zadanie o najkrótszym czasie wykonania
 - zaleta: minimalizuje średni czas oczekiwania, szybciej kończy dużo krótkich zadań
 - wada: dobre oszacowanie długości zadania, długie zadania stale odkładane
- **Priorytetowe** – każdy watek ma priorytet i planista wybiera watek o najwyższym priorytecie
 - zaleta: ważne zadania dostają procesor szybciej
 - wada: możliwe zagłodzenie – wątki o niskim priorytecie mogą bardzo długo nie dostać czasu procesora
 - typowa ochrona:
 - zwiększanie priorytetu wątków, które długo czekają
 - ograniczanie czasu pracy wątków o wysokim priorytecie

160

Dziękuję za uwagę

Treści programowe:

Rodzaje pamięci, algorytmy przydziału pamięci oraz zarządzanie pamięcią w systemach operacyjnych



162

Architektura komputera – rodzaje pamięci

Procesor – bardzo szybki (1–2 ns na cykl)

Pamięć RAM – znacznie wolniejsza (50–100 ns)

Bez szybkiej pamięci blisko procesora → procesor marnowałby 90% czasu na czekanie

Kolejność pamięci – od najszybszej do najwolniejszej:

- Rejestry procesora (~0,5–1 ns, kilkadziesiąt bajtów)
- Cache L1 (~1–4 cykle, 32–128 KB na rdzeń)
- Cache L2 (~10–20 cykli, 256 KB–2 MB na rdzeń)
- Cache L3 (~40–80 cykli, 8–128 MB współdzielona)
- RAM (~100–200 cykli, 8–128 GB)
- SSD NVMe (~10–100 μ s)
- HDD (~5–10 ms)

163

Architektura komputera – rodzaje pamięci

Procesor prawie nigdy nie pobiera pojedynczych bajtów – zawsze całe linie pamięci podręcznej (cache) – zwykle 64 bajty

Dwa rodzaje lokalności:

- Czasowa – jeśli odwołaliśmy się do danej komórki, to prawdopodobnie zaraz znowu do niej wrócimy (np. zmienna w pętli)
- Przestrzenna – jeśli odwołaliśmy się do danej komórki, to prawdopodobnie zaraz odwołamy się do sąsiednich (np. tablica, struktura)
- Dzięki lokalności współczynnik trafienia pamięci podręcznej (hit rate cache) często przekracza 95–98% – procesor prawie nie czeka na RAM

164

Architektura komputera – rodzaje pamięci

Rodzaj pamięci	Pojemność (typowa)	Czas dostępu	Gdzie znajduje się
Rejestry	kilkadziesiąt bajtów	~0,5–1 ns	w procesorze
Cache L1	32–128 KB / rdzeń	~1–4 cykle	w procesorze
Cache L2	256 KB–2 MB / rdzeń	~10–20 cykli	w procesorze
Cache L3	8–128 MB (współdzielona)	~40–80 cykli	w procesorze
RAM (DDR4/5)	8–128 GB	~50–100 ns	na płycie głównej
SSD NVMe	256 GB–8 TB	~10–100 μ s	dysk
HDD	1–20 TB	~5–10 ms	dysk

165

Architektura komputera – przestrzeń adresowa

Przestrzeń adresowa – zbiór wszystkich unikalnych adresów, które proces może wykorzystać do zaadresowania pamięci – rozmiar zależy od liczby bitów w rejestrze adresowym: 2^n , n – liczba bitów

Kluczowe aspekty przestrzeni adresowej:

- Logiczna przestrzeń adresowa (wirtualna) – zbiór adresów generowanych przez program, umożliwiający pracę bez wiedzy o fizycznej lokalizacji danych w pamięci
- Fizyczna przestrzeń adresowa – rzeczywiste adresy komórek pamięci RAM, do których odwołuje się szyna adresowa
- Tłumaczenie adresów (MMU – Memory Management Unit) – mechanizm (często sprzętowy), który przekłada adresy logiczne na fizyczne podczas działania programu
- Przestrzeń adresowa I/O – osobny zakres adresów używany do komunikacji z urządzeniami peryferyjnymi (porty wejścia-wyjścia)

166

Architektura komputera – rejestr bazy i limitu

Dynamiczna relokacja – przestrzeń adresowa każdego procesu zmapowana na różne części pamięci fizycznej

Pierwsze rozwiązania – procesor wyposażony w dwa specjalne rejestry sprzętowe:

Rejestr bazy (Base Register) – przechowuje najmniejszy dopuszczalny adres fizyczny w pamięci RAM, od którego zaczyna się obszar zajmowany przez dany proces

Rejestr limitu (Limit Register) – określa rozmiar (zakres) tej przestrzeni adresowej – definiuje jak daleko od adresu bazowego proces może sięgać

Działanie: proces odwołuje się do pamięci (pobranie instrukcji odczyt/zapis słowa danych) → procesor (przed wysłaniem adresu na szynę pamięci) dodaje wartość bazową do adresu wygenerowanego przez proces i sprawdza, czy jest równy lub większy od wartości w rejestrze limitu, jeżeli tak → generuje błąd (przerwana operacja)

167

Architektura komputera – rejestr bazy i limitu

Rejestry bazy i limitu – wady:

Fragmentacja zewnętrzna: przestrzeń adresowa procesu musi być alokowana w pamięci fizycznej w sposób ciągły – ładowanie i usuwanie procesów → w luki w pamięci między procesami (za małe, aby zmieścić nowy proces – marnowanie pamięci)

Brak współdzielenia pamięci: każdy proces działa we własnym, zamkniętym obszarze – brak łatwego współdzielenia pamięci między procesami (np. współdzielonych bibliotek)

Sztywna alokacja: przypisanie całego obszaru pamięci w sposób ciągły – utrudnia to dynamiczne powiększanie pamięci w trakcie działania programu

Wysoki koszt relokacji: konieczność wykonywania porównania (szybkie) i dodawania (wolniejsze) przy każdym odwołaniu do pamięci

Wsparcie sprzętowe: dostęp do pamięci wymaga dodania wartości bazy i porównania z limitem → realizowane sprzętowo, ale dodaje minimalne opóźnienie

168

Architektura komputera – przeładowanie pamięci

Wymiana pamięci (swapping) – mechanizm, w którym cały proces (lub jego duża część) jest przenoszony z pamięci operacyjnej (RAM) na dysk, gdy RAM jest pełny

Potrzeba – lata 60.–80.

- Komputery miały bardzo mało RAM (kilka KB–MB)
- Chciano uruchamiać więcej programów niż mieści się w pamięci
- Swap pozwalał „wypchnąć” nieużywany proces na dysk i wpuścić nowy

Wymiana w praktyce:

- Kopiowanie całego obrazu procesu (kod + dane + stos) na dysk
- Zwalnianie całego miejsca w RAM
- Po wczytaniu z powrotem – wznowienie procesu

Architektura komputera – wymiana pamięci

1. RAM jest pełny, nowy proces chce wejść
2. Jądro wybiera najmniej aktywny proces (np. ten, który najdłużej nie działał)
3. Cały obraz procesu (kod, dane, stos) jest kopiowany na dysk (do pliku swap lub partycji)
4. Miejsce w RAM jest zwalniane
5. Nowy proces jest wczytywany do RAM i zaczyna działać
6. Gdy stary proces jest znowu potrzebny → jest wczytywany z dysku (cały!)

Kompaktowanie – przesuwanie procesów w pamięci fizycznej, aby połączyć wolne bloki w jeden duży: system zatrzymywał wszystkie procesy, przesuwał ich bloki pamięci w górę/dół (wolne miejsca obok siebie), aktualizował adresy bazowe segmentów w tablicach procesów, wznowiał działanie

170

Architektura komputera – wymiana pamięci

Wady klasycznej wymiany pamięci:

- Bardzo wolny – dysk był tysiące razy wolniejszy niż RAM
- Duże opóźnienia – wczytanie procesu trwało sekundy/minuty
- Fragmentacja zewnętrzna – po wielu wymianach powstawały dziury
- Wysokie zużycie procesora i dysku
- Brak izolacji – procesy widziały fizyczne adresy RAM
- Brak ochrony – łatwo o nadpisanie pamięci innego procesu

Obecnie klasyczna wymiana pamięci całych procesów praktycznie nie istnieje – została zastąpiona pamięcią wirtualną i stronicowaniem

Architektura komputera – zarządzanie wolną pamięcią

Zarządzanie wolną pamięcią – funkcja systemu operacyjnego, polegająca na dynamicznym przydzielaniu i zwalnianiu pamięci RAM dla działających procesów/programów

Kluczowe aspekty:

- Zarządca pamięci: śledzi wolne i zajęte bloki – używa różnych algorytmów
- Ochrona pamięci: system zapewnia, że procesy nie uzyskają nieuprawnionego dostępu do pamięci innych procesów
- Zarządzanie automatyczne i ręczne: niektóre języki programowania (Java, Python) automatycznie zwalniają niepotrzebną pamięć, C++ – programista zarządza za pomocą operatorów lub funkcji

172

Architektura komputera – mapa bitowa

Podstawowe mechanizmy:

Mapa bitowa = tablica bitów – każdy bit odpowiada jednemu blokowi pamięci (np. 1KB)

0 = blok wolny

1 = blok zajęty

Przykład: 1MB RAM podzielone na bloki po 1KB = 1024 bloki →

rozmiar mapy = 1024b = 128B

Zalety: bardzo mały rozmiar, łatwe sprawdzanie czy wolny blok, łatwe łączenie sąsiednich wolnych bloków

Wady: długi czas szukania wolnego bloku (liniowe), duży rozmiar przy dużym RAM, nie pamięta rozmiaru wolnych bloków

173

Architektura komputera – lista jednokierunkowa

Podstawowe mechanizmy:

Lista jednokierunkowa = połączone węzły, każdy opisuje jeden wolny blok i zawiera:

- adres początku bloku
- rozmiar bloku
- wskaźnik na następny wolny blok

Przykład:

wolny blok 1: adres 0x1000, rozmiar 200 KB → następny



wolny blok 2: adres 0x2000, rozmiar 80 KB → następny



wolny blok 3: adres 0x3000, rozmiar 150 KB → NULL (koniec listy)

174

Architektura komputera – lista jednokierunkowa

Algorytmy wyszukiwania na liście wolnych bloków:

1. Pierwszy pasujący (first fit) – wybiera pierwszy blok $\geq$ potrzebnego rozmiaru: najszybszy (mała lista do przeszukania), zostawia dużo małych luk
2. Następny pasujący (next fit) – działa jak first fit ale zapamiętuje miejsce, w którym został znaleziony pierwszy blok – kolejne wyszukiwanie od tego miejsca: gorsza wydajność
3. Najlepszy pasujący (best fit) – bierze najmniejszy blok $\geq$ potrzebnego rozmiaru: mniej luk, wolniejszy – przeszukuje całą listę
4. Najgorszy pasujący (worst fit) – bierze największy blok: zostawia duże wolne bloki na przyszłość, tworzy dużo luk

175

Architektura komputera – lista jednokierunkowa

Przykład:

stan pamięci początkowy (wolne bloki w kolejności adresowej): **200 KB | 80 KB | 150 KB**

Proces żąda: **100 KB**

First Fit → przeszukuje listę od lewej: pierwszy blok ≥ 100 KB to 200 KB → wybiera go
100 KB | 80 KB | 150 KB

Best Fit → przeszukuje całą listę: najmniejszy który zmieści 100 KB to ostatni blok 150 KB
→ wybiera go

200 KB | 80 KB | 50 KB

Worst Fit → przeszukuje całą listę: największy blok to 200 KB → wybiera go

100 KB | 80 KB | 150 KB

176

Architektura komputera – alokator bliźniaków

Podstawowe mechanizmy:

Alokator parzystych bloków (buddy system) = pamięć dzielona na bloki będące potęgą dwójki (2^n):

- Gdy proces żąda rozmiaru S , system wybiera najmniejszy blok $2^n \geq S$
Jeśli dostępny jest tylko większy blok, to jest on **dzielony na pół** aż do uzyskania właściwego rozmiaru
- Podczas zwalniania blok może zostać **scalony** ze swoim „kolegą” (*buddy*), ale tylko wtedy, gdy oba bloki mają ten sam rozmiar i oba są wolne

Zaleta: szybkie wyszukiwanie i szybkie scalanie (prosta arytmetyka adresów)

Wada: fragmentacja wewnętrzna (zaokrąglanie do 2^k)

177

Architektura komputera – alokator bliźniaków

Przykład: pula wolnej pamięci: **1024 KB = 2^{10}** (jeden blok)

1. Żądanie: **300 KB**: najmniejsza potęga dwójki ≥ 300 to **512 KB (2^9)**:

- Dzielimy: $1024 \rightarrow 512 + 512$
- Przydzielamy jeden blok **512 KB**, drugi **512 KB** zostaje wolny
- Strata (fragmentacja wewnętrzna): $512 - 300 = 212$ KB

2. Żądanie: **120 KB**: najmniejszy blok ≥ 120 to **128 KB (2^7)**:

- Bierzemy wolne 512 i dzielimy: $512 \rightarrow 256 + 256 \rightarrow 128 + 128$ (+ zostaje 256)
- Przydzielamy **128 KB**, wolne zostają: **128 KB, 256 KB**
- Zwalniamy blok **128 KB**
- Jeśli jego „kolega” **128 KB** jest wolny, scalamy do **256 KB**

178

Architektura komputera – pamięć płytkowa

Podstawowe mechanizmy:

Alokator płytkowy (SLAB allocator) = alokacja pamięci w jądrze systemu zoptymalizowana pod wiele małych obiektów o stałych rozmiarach (np. struktury opisujące procesy, pliki, gniazda sieciowe)

Zamiast wycinać pamięć dla każdego obiektu, system tworzy pamięci podręczne obiektów (object caches): osobne pule dla typu i rozmiaru obiektu

Płytko (slab) – fragment pamięci podzielony na wiele jednakowych slotów, z których każdy mieści jeden obiekt danego typu

Alokacja polega na pobraniu wolnego slotu z odpowiedniego cache, a zwolnienie na oddaniu slotu do puli wolnych

179

Architektura komputera – pamięć płytkowa

Zalety: bardzo szybka alokacja i zwalnianie, mniejsza fragmentacja dla małych obiektów, lepsza lokalność odniesień w pamięci, ponieważ obiekty tego samego typu leżą blisko siebie

Współdziałanie Buddy i SLAB w jądrze:

- **System kolegów (Buddy System):** alokacja niskiego poziomu, zarządza dużymi blokami fizycznej pamięci RAM, szybko dzieli i scala bloki o rozmiarach będących potęgami dwójki
- **Alokator płytkowy (SLAB allocator):** alokacja wysokiego poziomu, wykorzystuje pamięć dostarczoną przez Buddy do budowy płytek i cache obiektów, a następnie dzieli ją na sloty dla małych struktur jądra

180

Architektura komputera – pamięć wirtualna

Pamięć wirtualna – mechanizm zarządzania pamięcią komputera zapewniający procesowi wrażenie pracy w jednym, dużym, ciągłym obszarze pamięci operacyjnej – fizycznie może być ona pofragmentowana, nieciągła i częściowo przechowywana na urządzeniach pamięci masowej

Każdy proces widzi własną, dużą, ciągłą przestrzeń adresową, proces generuje adresy wirtualne (VA), sprzęt (MMU) tłumaczy je na adresy fizyczne (PA) w RAM, Ten sam adres wirtualny w dwóch różnych procesach może wskazywać na zupełnie inne adresy fizyczne

Główne elementy:

- izolacja – procesy nie widzą się nawzajem
- ochrona – sprzęt pilnuje uprawnień (r/w/x)
- iluzja ciągłości – proces widzi ciągłą pamięć, nawet jeśli jest pofragmentowana

181

Architektura komputera – pamięć wirtualna

MMU (Memory Management Unit) – sprzętowa jednostka w procesorze:

- tłumaczy adresy wirtualne → adresy fizyczne
- sprawdza uprawnienia dostępu (read / write / execute / user/supervisor)

Jeśli coś jest nie tak → generuje wyjątek (page fault lub protection fault), tryby pracy:

- tryb użytkownika (user mode) – procesy zwykłe
- tryb jądra (kernel/supervisor mode) – jądro systemu

MMU rozróżnia strony:

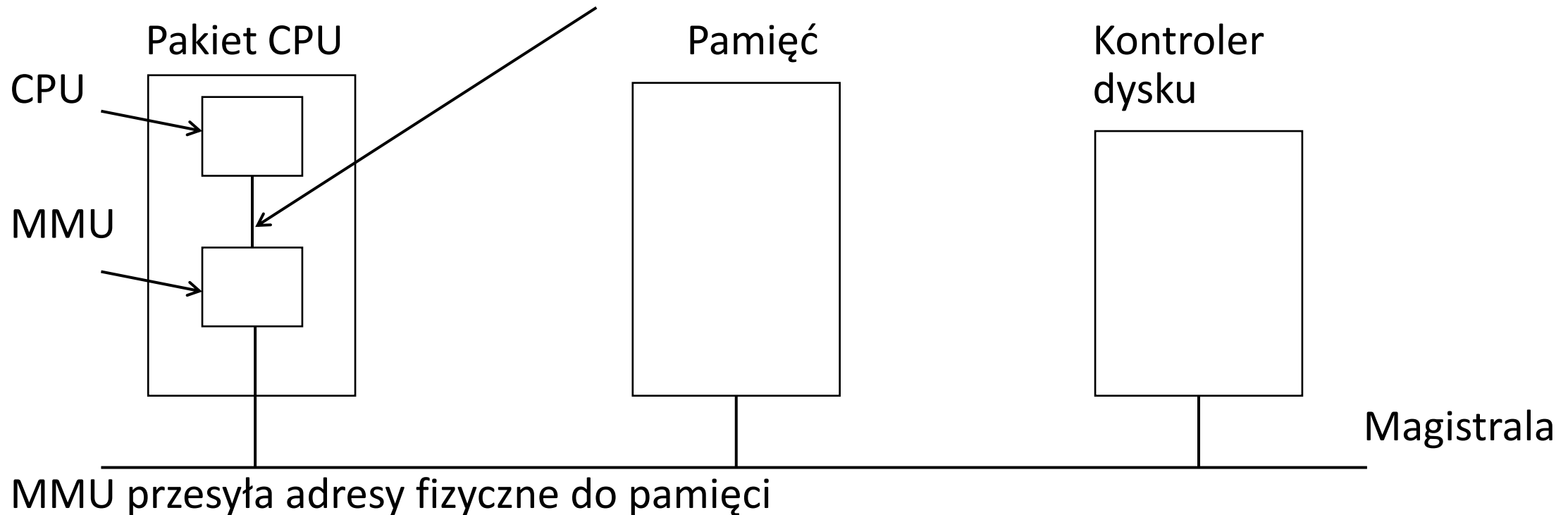
- user – dostępne tylko w trybie użytkownika
- supervisor – dostępne tylko w trybie jądra

Próba dostępu użytkownika do strony kernela → protection fault → SIGSEGV

182

Architektura komputera – pamięć wirtualna MMU

CPU przesyła adresy wirtualne do MMU



183

Architektura komputera – pamięć wirtualna

Stronicowanie – sposób zarządzania pamięcią, w którym komputer zapisuje i pobiera dane z pamięci dodatkowej do wykorzystania w pamięci podstawowej

- Pamięć wirtualna dzielona jest na **strony** (pages) o stałym rozmiarze (najczęściej 4 KB)
- Pamięć fizyczna dzielona jest na **ramki** (frames) o tym samym rozmiarze

Architektura x86-64: strony o rozmiarach 4KB, 2MB, 1GB

Przykład:

strony 4KB – dla użytkownika

strony 1GB – dla jądra

Mapowanie odbywa się na poziomie całych stron:

numer strony wirtualnej → numer ramki fizycznej + bity atrybutów

184

Architektura komputera – pamięć wirtualna

Format adresu wirtualnego – strona + przesunięcie (offset)

Dla strony 4 KiB przesunięcie (offset) ma 12 bitów, bo $2^{12} = 4096$

Adres wirtualny dzieli się na: numer strony (page number) oraz offset – nie zmienia się podczas translacji

Adres fizyczny – powstaje przez podstawienie numeru ramki (frame number) zamiast numeru strony i dołączenie offsetu

Adres wirtualny = [numer strony | offset]

Adres fizyczny = [numer ramki | offset] – offset pozostaje bez zmian!

Architektura komputera – pamięć wirtualna

Tablica stron (page table) – przechowuje mapowania dla procesu:

page number → frame numer

Wpis tablicy stron (page table entry, PTE) – zawiera również bity ochrony i stanu, aby MMU mogło wykonać kontrolę dostępu bez udziału jądra

Tablice są przełączane przy zmianie procesu (context switch) – każdy proces ma własne mapowania

Rozmiar pojedynczego wpisu we współczesnych komputerach – najczęściej 64 bity

Najważniejsze pole = fizyczny numer ramki strony

Architektura komputera – pamięć wirtualna

Typowe pola wpisu tablicy stron (64-bitowy wpis):

- Present (bit 0) – 1 = strona w RAM, 0 = page fault
- Read/Write (bit 1) – 1 = można zapisywać
- User/Supervisor (bit 2) – 1 = dostęp w trybie user
- Execute Disable (XD/NX, bit 63) – 1 = nie można wykonywać kodu
- Accessed (bit 5) – strona była odczytywana
- Dirty (bit 6) – strona była modyfikowana
- Page Frame Number – fizyczny numer ramki (przesunięty o 12 bitów)

187

Architektura komputera – pamięć wirtualna

Typowe pola wpisu tablicy stron (64-bitowy wpis):

- **Numer ramki strony** (Page Frame Number, PFN) – fizyczny numer ramki (przesunięty o 12 bitów w lewo) – wskazuje, gdzie w pamięci fizycznej znajduje się strona
- **Obecna / nieobecna** (Present / Valid, bit 0) 1 – strona jest aktualnie zmapowana w RAM 0 = strona nieobecna → dostęp generuje page fault
- **Zabezpieczenia** (Read / Write / Execute Disable, bity 1 + 63 + inne)
 - bit 1 (R/W): 1 = można zapisywać, 0 = tylko odczyt
 - bit 63 (XD / NX): 1 = zabronione wykonywanie kodu (ochrona DEP/NX)
- **Nadzorca** (User / Supervisor, bit 2) 0 – strona tylko dla trybu jądra (supervisor / kernel) 1 = strona dostępna także w trybie użytkownika

188

Architektura komputera – pamięć wirtualna

Typowe pola wpisu tablicy stron (64-bitowy wpis):

- **Zmodyfikowano** (Dirty, bit 6) 1 – strona była zapisywana od ostatniego wyczyszczenia bitu (używane do algorytmów wymiany – czy trzeba zapisać na dysk)
- **W użyciu** (Accessed / Referenced, bit 5) 1 = strona była odczytywana lub zapisywana od ostatniego resetu (kluczowe dla algorytmów LRU / Clock – druga szansa)
- **Buforowanie włączone** (PCD – Page Cache Disable / PWT – Page Write-Through, bity 4 i 3)
 - PCD = 1 → wyłączone buforowanie (uncacheable)
 - PWT = 1 → write-through (zamiast write-back)

189

Architektura komputera – pamięć wirtualna

Problemy stronicowania:

1. Odwzorowanie adresów wirtualnych na adresy fizyczne musi być szybkie
2. Duża przestrzeń adresów wirtualnych → tablica stron nie może być zbyt duża

Bufor translacji adresów (Translation Lookaside Buffer TLB) – szybka, sprzętowa pamięć podręczna umieszczona wewnątrz procesora

Przechowuje on najnowsze lub najczęściej używane tłumaczenia:

- wirtualny adres strony → fizyczny adres ramki w pamięci operacyjnej
- informacje o uprawnieniach (czy można czytać, zapisywać, wykonywać kod)

Bez tego bufora każde tłumaczenie adresu wirtualnego na fizyczny wymagałoby wielokrotnego odczytu tablic stron z pamięci operacyjnej – spowolnienie systemu

190

Architektura komputera – pamięć wirtualna

Bufor translacji adresów jest wbudowany bezpośrednio w procesor – znajduje się bardzo blisko jednostki zarządzania pamięcią (MMU)

Każdy rdzeń procesora ma własny, niezależny bufor (tzw. per-core TLB)

Współczesne procesory dzielą go na poziomy (podobnie jak zwykły cache danych):

- Poziom L1 – najszybszy, ale najmniejszy (zwykle 32–128 wpisów dla stron 4 KiB)
- Poziom L2 – większy i wolniejszy (512–4096 wpisów)
- Poziom L3 – współdzielony między rdzeniami (największy, ale wolniejszy)

Architektura komputera – pamięć wirtualna

Bufor translacji adresów – schemat działania:

Proces generuje adres wirtualny (VA)



Procesor najpierw sprawdza, czy ten adres jest już zapisany w buforze TLB?

- TAK → trafienie (TLB hit) → natychmiastowy wynik: fizyczny adres + uprawnienia (1–2 cykle zegara)
- NIE → nietrafienie (TLB miss) → procesor musi wykonać pełne przeszukanie tablic stron w pamięci operacyjnej (spacer po tablicy stron, page table walk) → to trwa kilkanaście-kilkadziesiąt cykli

Po znalezieniu poprawnego tłumaczenia wpis jest dodawany do bufora TLB

192

Architektura komputera – pamięć wirtualna

Sytuacja	Czas w cyklach zegara	Czas w nanosekundach	Jak często występuje
Trafienie w buforze TLB (hit)	1-4	~0,2-0,8 ns	95-99 %
Nietrafienie + trafienie w L2 TLB	10-40	~2-8 ns	1-4 %
Nietrafienie + pełne przeszukanie tablic stron	100-500+	~20-100 ns	<1 %
Nietrafienie + błąd strony (page fault)	miliony cykli	mikrosekundy– milisekundy	bardzo rzadko

193

Architektura komputera – pamięć wirtualna

Typowy wpis w buforze TLB zawiera:

- Tag (fragment wirtualnego adresu strony) – służy do sprawdzenia, czy wpis pasuje do bieżącego adresu
- Fizyczny numer ramki (numer strony fizycznej w pamięci operacyjnej)
- Bity uprawnień (czy można czytać, zapisywać, wykonywać kod; czy strona jest dostępna w trybie użytkownika czy tylko jądra)
- Czasami dodatkowe bity stanu (czy strona była używana, czy była modyfikowana)
- Identyfikator przestrzeni adresowej (Process Context Identifier / Address Space Identifier) – pozwala współdzielić wpisy między procesami bez konieczności czyszczenia bufora przy zmianie procesu

194

Architektura komputera – pamięć wirtualna

Przykładowe rozmiary bufora TLB w aktualnych procesorach (Intel Core 14./15. generacja, AMD Zen 4/5):

- Bufor L1 dla danych (Data TLB): 64-128 wpisów (strony 4 KiB)
- Bufor L1 dla kodu (Instruction TLB): 32-96 wpisów (strony 4 KiB)
- Bufor L2 (zunifikowany): 512-2048 wpisów (strony 4 KiB)
- Bufor dla stron wielkich (2 MiB / 1 GiB): 8-64 wpisy
- Liczba możliwych identyfikatorów kontekstu: 4096-65536

Programy z dobrą lokalnością odwołań (mało różnych stron) działają znacznie szybciej

Im więcej różnych stron programu – tym więcej nietrafień w buforze

195

Architektura komputera – pamięć wirtualna

Architektura x86-64 definiuje adresy wirtualne jako 64-bitowe → teoretycznie → 2^{64} bajtów = 16 eksabajtów (EB)

W praktyce procesory implementują tylko 48 bitów efektywnego adresu → pozostałe bity (63-48) muszą być rozszerzeniem znaku bitu 47 → tzw. adresowanie kanoniczne

- 48 bitów → 2^{48} bajtów = 256 TB (tebibajtów) przestrzeni wirtualnej na jeden proces
- 128 TB dla przestrzeni użytkownika (bit 47 = 0)
- 128 TB dla przestrzeni jądra (bit 47 = 1)
- Dlaczego akurat 48 bitów?
 - Kompromis między wielkością przestrzeni a zużyciem pamięci na tablice stron
 - 48 bitów + 4 poziomy po 9 bitów idealnie pasuje do stron 4 KB
 - Każda tablica na poziomie ma 512 wpisów $\times$ 8 B = dokładnie 4096 B = 1 strona

196

Architektura komputera – pamięć wirtualna

W systemie 64-bitowym przestrzeń wirtualna ma ogromny rozmiar (np. 256 TB). Gdyby tablica stron była jednopoziomowa:

- $256 \text{ TiB} / 4 \text{ KiB} = 64$ miliardy wpisów
- Każdy wpis = 8 bajtów $\rightarrow$ tablica zajmowałaby 512 GB RAM

Żaden proces nie potrzebuje aż tyle pamięci jednocześnie

Rozwiązanie:

Wielopoziomowe tablice stron – numer strony z adresu wirtualnego jest dzielony na kilka pól indeksujących kolejne poziomy tablic; każdy poziom zawiera wskaźniki do następnego poziomu, a ostatni poziom zawiera wpis PTE (page table entry) z numerem ramki i bitami ochrony

Jądro tworzy tylko te fragmenty struktury, które są potrzebne – brak kosztu dla nieużywanych zakresów przestrzeni adresowej

197

Architektura komputera – pamięć wirtualna

Standard w procesorach Intel i AMD od ok. 2008 r. – adres wirtualny (48 bitów):

[9 bitów] [9 bitów] [9 bitów] [9 bitów] [12 bitów offset]



PML4

PDP

PD

PT

→ strona 4 KB

Każdy poziom to tablica 512 wpisów ($2^9 = 512$) – każda tablica zajmuje dokładnie 4 KB

PML4 (Page Map Level 4) → najwyższy poziom → 512 wskaźników do tablic PDP → 256 TB

PDP (Page Directory Pointer) → drugi poziom → 512 wskaźników do tablic PD → 512 GB

PD (Page Directory) → trzeci poziom → 512 wskaźników do tablic PT → 1 GB

PT (Page Table) → najniższy poziom → 512 wpisów wskazujących na ramki 4 KiB → 2 MB

198

Architektura komputera – pamięć wirtualna

Drzewo tablic w praktyce:

Proces startuje → jądro tworzy tylko najwyższy poziom (PML4) – 512 wpisów

Gdy proces pierwszy raz dotknie danej strony wirtualnej → jądro alokuje kolejno niższe poziomy (PDP → PD → PT)

W efekcie:

- większość procesów ma tylko kilka-kilkanaście tablic na niższych poziomach
- całkowity rozmiar tablic stron dla typowego procesu to kilka-kilkanaście megabajtów - zamiast 512 GB

199

Architektura komputera – pamięć wirtualna

Tablice wielowymiarowe zalety:

Mniejsze zużycie pamięci na struktury mapowania: tworzone są tylko te fragmenty tablic, które odpowiadają faktycznie używanym zakresom przestrzeni adresowej procesu

Naturalne dopasowanie do modelu procesu: każdy proces ma własny zestaw tablic, co upraszcza izolację i kontrolę uprawnień dostępu

Łatwe zwalnianie metadanych: gdy proces kończy działanie lub zwalnia duży obszar pamięci, można usuwać całe podtablice, a nie pojedyncze wpisy

Architektura komputera – pamięć wirtualna

Tablice wielowymiarowe wady:

Większy koszt tłumaczenia adresu bez wpisu w buforze translacji: gdy procesor nie ma gotowego mapowania, musi odczytać kolejne poziomy tablic z pamięci operacyjnej = kilka dodatkowychostępów do RAM zanim nastąpi właściwy odczyt danych

Narzut pamięci i presja na pamięć podręczną procesora: same tablice stron zajmują RAM i konkurują o miejsce w pamięciach podręcznych z danymi i kodem, co może pogarszać wydajność przy dużej liczbie aktywnych mapowań

Zależność wydajności od sprzętowego przyspieszania: w systemach 64-bit mechanizm buforowania translacji oraz sprzętowe wsparcie przechodzenia po tablicach są konieczne, aby narzut stronicowania nie dominował czasu wykonania

201

Architektura komputera – pamięć wirtualna

Odwrócona tablica stron – struktura mapowań pamięci, w której jeden wpis odpowiada jednej ramce pamięci fizycznej RAM, a nie jednej stronie pamięci wirtualnej procesu

Zamiast tablic osobnych dla każdego procesu, system utrzymuje jedną wspólną tablicę o rozmiarze proporcjonalnym do liczby ramek w RAM

Typowa zawartość wpisu: identyfikator procesu, numer strony wirtualnej, która aktualnie znajduje się w tej ramce, oraz bity ochrony/stanu

Ogranicza ilość metadanych tablic stron w systemach, gdzie przestrzeń adresowa procesów jest bardzo duża, ale liczba ramek RAM jest skończona

Architektura komputera – pamięć wirtualna

W odwróconej tablicy stron nie da się bezpośrednio indeksować numerem strony wirtualnej → tablica jest indeksowana ramkami fizycznymi

Translacja adresu wirtualnego działa jako wyszukiwanie: system oblicza funkcję skrótu z pary: identyfikator procesu + numer strony wirtualnej = hashowanie i szuka pasującego wpisu

Większa złożoność i koszt tłumaczenia bez buforowania – kluczowa jest sprzętowa pamięć podręczna mapowań adresów, która przechowuje ostatnie translacje

Zaleta: mniejszy koszt pamięciowy metadanych

Wada: trudniejsza i potencjalnie wolniejsza ścieżka tłumaczenia, gdy brakuje gotowego mapowania w pamięci podręcznej procesora

203

Architektura komputera – typowe błędy pamięci

Nielegalny dostęp do pamięci (segmentation fault / access violation)

Proces próbuje odczytać, zapisać lub wykonać kod spod adresu, który:

nie jest zmapowany w przestrzeni adresowej procesu (brak mapowania)

jest zmapowany, ale bez wymaganych uprawnień (np. zapis do obszaru tylko do odczytu, wykonanie z obszaru niewykonywalnego)

należy do jądra lub do obszaru niedostępnego dla trybu użytkownika

Mechanizm powstawania:

jednostka zarządzania pamięcią wykrywa naruszenie → powstaje wyjątek → jądro analizuje przyczynę → jeśli dostęp jest nielegalny, proces dostaje SIGSEGV i kończy się; w Windows analogicznie pojawia się Access Violation

204

Architektura komputera – typowe błędy pamięci

Nieprawidłowe wyrównanie dostępu (alignment fault / bus error)

Niektóre architektury wymagają, aby dane były pod adresami zgodnymi z wyrównaniem

Dostęp niewyrównany może spowodować błąd sprzętowy (często sygnalizowany jako SIGBUS) albo zostać zablokowany przez CPU

Błąd strony (page fault) – legalny vs nielegalny

Legalny: brak danych w RAM, ale adres jest poprawny (ładowanie na żądanie, odczyt z obszaru wymiany, kopiowanie przy zapisie) → jądro przygotowuje dane/mapowanie → proces kontynuuje

Nielegalny: adres nie należy do procesu lub brak uprawnień → proces kończy się (najczęściej SIGSEGV; w niektórych przypadkach SIGBUS, np. błędy mapowania pliku)

205

Architektura komputera – typowe błędy pamięci

Błędy w jądrze i sterownikach (kernel oops / kernel panic)

Sterownik lub kod jądra wykonuje niepoprawny dostęp, np. zapis do nieistniejącego adresu, użycie wskaźnika NULL, użycie pamięci po zwolnieniu

Skutek: zapis w logach i próba kontynuacji (kernel oops), albo zatrzymanie systemu, gdy nie da się bezpiecznie działać dalej (Linux - kernel panic; Windows - BSOD)

Thrashing (intensywna wymiana danych między RAM a dyskiem)

Gdy jest za mało RAM, system stale przenosi dane między RAM a obszarem wymiany

Skutek: bardzo dużo obsługi błędów strony i operacji dyskowych, duże opóźnienia

Wyciek pamięci w jądrze

Jądro lub moduł nie zwalnia zaalokowanej pamięci, więc zużycie RAM rośnie w czasie

Skutek: spadek wydajności, a w skrajnym przypadku brak możliwości przydziału pamięci

206

Architektura komputera – algorytmy zastępowania stron

Gdy pamięć operacyjna jest pełna, a proces chce załadować nową stronę, jądro systemu operacyjnego musi wyrzucić jedną z już obecnych stron

Cel algorytmów zastępowania stron: wybrać stronę do wyrzucenia w taki sposób, aby spowodować jak najmniejszą liczbę przyszłych błędów strony (czyli sytuacji, gdy strona jest potrzebna, ale nie ma jej w pamięci operacyjnej)

Główne algorytmy:

- First In, First Out (pierwszy weszło, pierwszy wyszło)
- Least Recently Used (najmniej niedawno używana)
- Clock (zegar oraz druga szansa)
- inne, takie jak Optimal (optymalny, teoretyczny), Not Recently Used (nie niedawno używana), Least Frequently Used (najmniej często używana), Random (losowy)

Architektura komputera – algorytmy zastępowania stron

First In, First Out – pierwsze weszło, pierwsze wyszło

Zasada działania algorytmu: wyrzucić stronę, która najdłużej jest w pamięci operacyjnej

Przykład sekwencji odwołań do stron (używane strony: 1,2,3,4,1,2,5,1,2,3,4,5; liczba ramek w pamięci: 3): Liczba błędów strony: 9

Schemat krok po kroku:

- Początek: pamięć pusta
- Odwołanie do 1 → błąd strony → pamięć: 1
- Odwołanie do 2 → błąd strony → pamięć: 1,2
- Odwołanie do 3 → błąd strony → pamięć: 1,2,3
- Odwołanie do 4 → błąd strony → wyrzucić 1 (najstarsza) → pamięć: 2,3,4

208

Architektura komputera – algorytmy zastępowania stron

First In, First Out – pierwsze weszło, pierwsze wyszło

Anomalia Belady’ego: sytuacja, w której zwiększenie liczby ramek w pamięci operacyjnej powoduje paradoksalnie większą liczbę błędów strony

Przykład:

Ta sama sekwencja odwołań do stron, liczba błędów strony przy 3 ramkach = 9;

przy 4 ramkach = 10

Sposób powstawania:

Algorytm nie bierze pod uwagę, czy strona będzie używana w przyszłości – opiera się tylko na kolejności wejścia, co czasem pogarsza sytuację przy zmianie liczby ramek

209

Architektura komputera – algorytmy zastępowania stron

Algorytm drugiej szansy – modyfikacja FIFO

Algorytm drugiej szansy jest praktycznym przybliżeniem LRU, w którym strona używana niedawno nie jest od razu usuwana

Ramki są uporządkowane cyklicznie, a wskaźnik przeglądu kandydatów do usunięcia w stałej kolejności

Jeśli strona była używana, system zeruje znacznik użycia i przesuwają wskaźnik dalej, dając tej stronie drugą szansę

Jeśli strona nie była używana od ostatniego przeglądu, jest wybierana do usunięcia

Koszt działania jest mały, a skuteczność w praktyce dobra, dlatego to podejście jest bardzo popularne

210

Architektura komputera – algorytmy zastępowania stron

Least Recently Used – najmniej niedawno używana

Zasada działania algorytmu:

Wyrzucić stronę, której najdłużej nie używano (opiera się na historii ostatnich odwołań)

Przykład (ta sama sekwencja odwołań do stron, 3 ramki w pamięci operacyjnej):

Liczba błędów strony = 8

Implementacja w systemie operacyjnym:

Utrzymuj listę stron uporządkowaną według czasu ostatniego użycia (każde odwołanie przenosi stronę na początek listy)

Architektura komputera – algorytmy zastępowania stron

Least Recently Used – najmniej niedawno używana

Wady:

Wysoki koszt obliczeniowy: przy każdym odwołaniu do strony jądro musi aktualizować całą listę stron (przeniesienie na początek listy)

Potrzebna złożona struktura danych:

Lista powiązana z dodatkową tablicą haszującą do szybkiego wyszukiwania

W praktyce – zbyt wolny dla milionów stron w pamięci operacyjnej

Systemy operacyjne stosują przybliżenia LRU, które bazują na znacznikach użycia, listach aktywności i okresowej aktualizacji stanu

212

Architektura komputera – algorytmy zastępowania stron

Clock – Algorytm zegarowy z preferencją stron niemodyfikowanych

Zasada działania algorytmu:

Strony ułożone w kole (jak tarcza zegara), każda strona ma bit referencyjny (bit wskazujący, czy strona była niedawno używana)

Wskaźnik (wskazówka zegara) krąży po kole: jeśli bit = 1, resetujemy go na 0 i dajemy drugiej szansie; jeśli bit = 0, wyrzucamy stronę

Strony modyfikowane są kosztowne do usunięcia, bo wymagają zapisu na dysk, więc algorytm powinien je usuwać możliwie rzadko

Strony nieużywane i niemodyfikowane mogą być usuwane natychmiast bez operacji dyskowej

213

Architektura komputera – algorytmy zastępowania stron

Optimal – algorytm optymalny: punkt odniesienia

Zasada działania algorytmu: wyrzucić stronę, która będzie używana najpóźniej w przyszłości (algorytm zakłada, że jądro systemu operacyjnego zna całą przyszłą sekwencję odwołań do stron)

Przykład (ta sama sekwencja odwołań do stron, 3 ramki w pamięci operacyjnej):

Liczba błędów strony: 5 (najmniej ze wszystkich algorytmów)

Nie jest możliwy do implementacji w działającym systemie, bo wymaga znajomości przyszłych odwołań do pamięci

Jest wykorzystywany do analiz i porównań, ponieważ daje dolną granicę liczby błędów strony dla danej sekwencji odwołań

Jeśli inny algorytm praktyczny zbliża się do optymalnego w typowych obciążeniach, uznaje się go za skuteczny

Architektura komputera – algorytmy zastępowania stron

Inne algorytmy zastępowania stron:

Not Recently Used (nie niedawno używana) – prostsza wersja algorytmu zegara, używa dwóch bitów: bit referencyjny (czy używana) + bit modyfikacji (czy zmieniona)

Least Frequently Used (najmniej często używana) – wyrzuca stronę używaną najrzadziej (liczy liczbę odwołań)

Random (losowy) – wybiera stronę do wyrzucenia całkowicie losowo – prosty ale nieprzewidywalny

Architektura komputera – zarządzanie pamięcią – podsumowanie

Pamięć wirtualna z ochroną: każdy proces ma własną przestrzeń adresową, a sprzętowa translacja adresów wymusza uprawnienia odczytu/zapisu/wykonania

Stronicowanie, błędy strony i wymiana: dane są ładowane na żądanie, a przy presji pamięci system zwalnia RAM przez usuwanie danych odtwarzalnych i ewentualnie przenoszenie danych prywatnych do obszaru wymiany na dysku

Optymalizacja oparta o lokalność: mechanizmy zastępowania stron i pamięci podręczne są projektowane tak, aby utrzymać w RAM to, co jest najczęściej używane w danej fazie pracy

Architektura komputera – zarządzanie pamięcią – podsumowanie

Linux:

- Agresywnie wykorzystuje wolny RAM jako pamięć podręczną plików (buforowanie odczytów/zapisów), dlatego „mało wolnej pamięci” zwykle oznacza duży cache, a nie problem
- Pod presją pamięci najpierw odzyskuje RAM z cache plików, a dopiero potem intensywniej sięga po obszar wymiany; w skrajności uruchamia mechanizmy kończenia procesów, gdy pamięci nie da się już odzyskać
- Izolacja i limity pamięci są często realizowane mechanizmami kontroli zasobów (np. dla usług i kontenerów), co stabilizuje system przy wielu procesach

217

Architektura komputera – zarządzanie pamięcią – podsumowanie

Windows:

- Zarządza pamięcią przez model „zobowiązania” (ile pamięci system obiecał zapewnić procesom) oraz przez aktywne sterowanie zestawem stron rezydentnych procesów w RAM
- Plik stronicowania jest elementem utrzymania tych zobowiązań i buforem bezpieczeństwa przy chwilowych pikach zapotrzebowania na pamięć
- Diagnostycznie istotne jest rozróżnienie pomiędzy ogólną liczbą błędów strony a sytuacjami, gdy system musi realnie czytać z dysku, bo to wtedy pojawiają się opóźnienia

218

Dziękuję za uwagę

Treści programowe:

Struktura i organizacja systemów plików, typy systemów plików oraz operacje na plikach i katalogach



220

Architektura komputera – system plików

Plik – logiczna jednostka przechowywania danych udostępniana przez system operacyjny

Dla użytkownika plik ma nazwę, lokalizację i określoną zawartość

Dla systemu operacyjnego plik jest obiektem posiadającym:

- dane użytkownika
- metadane opisujące ten obiekt
- informację o położeniu danych na nośniku lub o sposobie ich odnalezienia

Plik stanowi abstrakcję, która ukrywa przed użytkownikiem fizyczną organizację sektorów i bloków na dysku

221

Architektura komputera – system plików

Bez systemu plików dane byłyby przechowywane jedynie jako ciągi bloków na nośniku. System operacyjny wprowadza pojęcie pliku, aby:

- umożliwić zapis i odczyt danych pod nazwą
- oddzielić aplikację od szczegółów fizycznej organizacji dysku
- zapewnić ochronę danych poprzez prawa dostępu
- umożliwić współdzielenie danych między procesami i użytkownikami
- uprościć organizację dużej liczby dokumentów, programów i danych systemowych

Dzięki temu aplikacja operuje na obiektach logicznych, a nie na surowych adresach bloków

222

Architektura komputera – system plików

Każdy plik posiada nazwę, pod którą jest widoczny w katalogu

Nazwa pliku służy do identyfikacji obiektu w przestrzeni nazw systemu plików

Nazwa często składa się z:

- części głównej
- rozszerzenia, które zwyczajowo sugeruje typ lub przeznaczenie pliku

Przykłady:

- raport.txt – plik tekstowy
- dane.csv – dane tabelaryczne
- program.exe – plik wykonywalny w Windows

223

Architektura komputera – system plików

Systemy plików nakładają ograniczenia na:

- maksymalną długość nazwy
- zestaw dozwolonych znaków
- sposób rozróżniania małych i wielkich liter
- występowanie nazw zastrzeżonych

Przykładowo:

- w Linuxie nazwy plik.txt i Plik.txt są zwykle traktowane jako różne
- w Windows w praktyce użytkowej najczęściej nie rozróżnia się wielkości liter
- niektóre znaki są niedozwolone w nazwach plików w Windows, np. \ / : * ? " < > |

224

Architektura komputera – system plików

Długość nazw plików i rozszerzeń - Linux (ext4 / VFS):

- maksymalna długość pojedynczej nazwy pliku lub katalogu wynosi 255 bajtów
- limit w Linuxie podawany jest w bajtach a nie w znakach – liczba widocznych znaków może być mniejsza niż 255

Długość nazw plików i rozszerzeń (NTFS / exFAT / FAT32 z długimi nazwami):

- maksymalna długość pojedynczego elementu ścieżki wynosi 255 znaków Unicode
- historycznie system Windows zachowuje zgodność z formatem 8.3 (8 znaków nazwy + 3 znaki rozszerzenia)

225

Architektura komputera – system plików

Plik może być rozumiany na kilka sposobów:

- jako nieustrukturyzowany ciąg bajtów
- jako ciąg rekordów
- jako zbiór logicznych bloków lub stron

W nowoczesnych systemach operacyjnych plik jest traktowany jako ciąg bajtów, a interpretacja jego wewnętrznej struktury należy do aplikacji

Przykład:

- edytor tekstu interpretuje plik jako tekst
- odtwarzacz multimedialny interpretuje plik jako strumień audio lub wideo
- baza danych może interpretować plik jako strony rekordów

226

Architektura komputera – system plików

Fizycznie plik nie musi być zapisany jako jeden ciągły fragment dysku, składa się z:

- rekordu metadanych
- jednego lub wielu bloków danych
- wpisu katalogowego, który wiąże nazwę z obiektem plikowym

Dane pliku mogą być:

- zapisane w blokach sąsiadujących
- rozproszone w różnych miejscach nośnika
- opisywane przez listy bloków, tabelę alokacji lub i-węzły

Nazwa pliku nie mówi nic o fizycznym położeniu jego danych

227

Architektura komputera – system plików

System operacyjny rozróżnia różne typy obiektów plikowych

Najważniejsze z nich to:

- **zwykłe pliki** – zawierają dane użytkownika lub programów
- **katalogi** – organizują przestrzeń nazw
- **dowiązania symboliczne** – wskazują ścieżkę do innego obiektu
- **pliki specjalne urządzeń** – reprezentują urządzenia
- **potoki i gniazda** – służą do komunikacji między procesami

Wspólna abstrakcja pliku → system operacyjny może udostępniać różne zasoby podobnym interfejsem: operacje wejścia-wyjścia

228

Architektura komputera – system plików

W Linuxie pojęcie pliku jest bardzo szerokie i obejmuje również wiele obiektów systemowych: zwykłe pliki, katalogi, dowiązania, urządzenia znakowe i blokowe, potoki nazwane, gniazda

W Windows użytkownik najczęściej spotyka: zwykłe pliki, katalogi, skróty, specjalne pliki systemowe i wykonywalne

Systemy Unixowe – tekstowy, zunifikowany interfejs zasobów

Systemy windowsowe – model obiektów systemowych realizowany przez warstwy API i GUI

Architektura komputera – dostęp do plików – dostęp sekwencyjny

W dostępie sekwencyjnym dane są czytane lub zapisywane kolejno od początku do końca pliku – najprostszy model dostępu

Cechy:

- odpowiada naturalnemu przetwarzaniu tekstów, logów i strumieni
- prostota implementacji
- efektywny odczyt i zapis dużych, liniowych danych

Wady:

przejsię od razu do środka pliku bywa niewygodne lub kosztowne, jeśli system lub aplikacja nie utrzymują odpowiednich mechanizmów pozycjonowania

230

Architektura komputera – dostęp do plików – dostęp bezpośredni i indeksowany

W dostępie bezpośrednim można odwołać się do wybranego fragmentu pliku bez czytania wszystkiego od początku, wymaga to:

- znajomości pozycji w pliku
- możliwości wykonania operacji przesunięcia wskaźnika
- efektywnego mapowania danych pliku na bloki

Kluczowy dla:

baz danych, systemów multimedialnych, plików z dużą liczbą rekordów, aplikacji edytujących tylko fragment danych

W bardziej złożonych przypadkach stosuje się również mechanizmy indeksowania, które przyspieszają odwołania do określonych rekordów lub obszarów danych

231

Architektura komputera – atrybuty plików

Każdy plik oprócz treści posiada zestaw atrybutów, czyli metadanych:

nazwa, identyfikator obiektu, typ pliku, rozmiar, właściciel i grupa, prawa dostępu, liczba dowiązań, znaczniki czasu utworzenia, modyfikacji i dostępu

Atrybuty są niezbędne → system musi wiedzieć:

- kto może korzystać z pliku
- jaki jest jego charakter
- jak duży jest plik
- gdzie i w jaki sposób przechowywane są jego dane

Architektura komputera – operacje na plikach

System operacyjny udostępnia zestaw podstawowych operacji na plikach:

- **create** – utworzenie nowego pliku
- **open** – otwarcie pliku
- **read** – odczyt danych
- **write** – zapis danych
- **append** – dopisywanie na końcu
- **seek** – zmiana bieżącej pozycji w pliku
- **close** – zamknięcie pliku
- **rename** – zmiana nazwy
- **delete / unlink** – usunięcie wpisu pliku z przestrzeni nazw

233

Architektura komputera – operacje na plikach – otwarcie pliku

Otwarcie pliku nie jest tylko prostym odczytem nazwy – system operacyjny musi wykonać kilka kroków:

- rozwiązać ścieżkę do pliku
- przejść przez katalogi pośrednie
- sprawdzić prawa dostępu użytkownika lub procesu
- odnaleźć rekord metadanych pliku
- utworzyć wpis w tablicy otwartych plików
- zwrócić procesowi deskryptor lub uchwyt

Wykonanie tych czynności → proces może wykonać odczyt lub zapis

234

Architektura komputera – pusty plik

Pusty plik ma rozmiar logiczny 0 bajtów – nie zawiera danych użytkownika

System plików musi zapisać:

- nazwę pliku w katalogu
- metadane obiektu
- identyfikator i informacje o prawach dostępu
- znaczniki czasu

W Linuxie będzie to związane z utworzeniem i-węzła (inode)

W Windowsie będzie to związane z utworzeniem rekordu (MFT)

pusty plik może nie zajmować bloków danych → zajmuje miejsce w strukturach systemu plików 235

Architektura komputera – katalogi

Katalog – specjalny obiekt systemu plików służący do organizacji przestrzeni nazw

Jego podstawową rolą jest powiązanie:

- nazwy pliku lub podkatalogu z
- odpowiednim obiektem systemu plików

Katalog jest rzeczywistą strukturą danych, dzięki której system potrafi odnaleźć plik po nazwie

Z punktu widzenia systemu katalog jest mapą nazw

Gdy użytkownik wpisuje ścieżkę lub klika nazwę pliku w eksploratorze, system przeszukuje odpowiednie katalogi, aby znaleźć obiekt końcowy

236

Architektura komputera – katalogi

Katalog – specjalny obiekt systemu plików służący do organizacji przestrzeni nazw

Katalogi są potrzebne, aby:

- porządkować dużą liczbę plików
- grupować dane według użytkownika, projektu lub typu
- umożliwiać tworzenie hierarchii nazw
- ułatwiać zarządzanie i nawigację
- wspierać ochronę i separację zasobów

Bez katalogów wszystkie pliki musiałyby znajdować się w jednej wspólnej przestrzeni nazw, co w praktyce byłoby niewygodne i mało skalowalne

237

Architektura komputera – katalogi

Najprostszą organizacją katalogów jest system jednopoziomowy, w którym wszystkie pliki znajdują się w jednym katalogu

Zalety:

- bardzo prosta implementacja, łatwe wyszukiwanie przy małej liczbie plików, mały narzut organizacyjny

Wady:

- trudność w zachowaniu unikalnych nazw, brak naturalnego podziału danych, bardzo słaba skalowalność, problemy w systemach wieloużytkownikowych

Architektura komputera – katalogi

Rozwinięciem prostego modelu jest system dwupoziomowy lub częściowo hierarchiczny, w którym każdy użytkownik (lub program) otrzymuje własny katalog główny

Zalety:

- łatwiejsza separacja danych użytkowników, zmniejszenie konfliktów nazw, lepsza organizacja niż w modelu jednopoziomowym

Wady:

- nadal ograniczona elastyczność, mniejsza wygoda niż w pełnym drzewie katalogów, słabsze wsparcie dla złożonych projektów z wieloma poziomami podziału

Architektura komputera – katalogi

Najbardziej rozpowszechnioną organizacją jest hierarchiczny system katalogów

Cechy:

- istnieje katalog główny
- katalogi mogą zawierać podkatalogi
- całość tworzy drzewo katalogów
- każdy obiekt ma określone miejsce w hierarchii

Zalety:

- bardzo dobra skalowalność,
- naturalny podział na grupy tematyczne
- łatwiejsza administracja
- większa czytelność i porządek

Architektura komputera – ścieżki

Położenie pliku lub katalogu opisuje się za pomocą ścieżki

Wyróżniamy:

- **ścieżkę bezwzględną** – rozpoczynającą się od katalogu głównego
- **ścieżkę względną** – interpretowaną względem katalogu bieżącego

Przykłady:

- Linux: /home/student/projekt/plik.txt
- Windows: C:\Users\Student\Documents\plik.txt

Ścieżka stanowi uporządkowany opis przejścia przez kolejne katalogi prowadzące do danego obiektu

241

Architektura komputera – ścieżki

Aby odnaleźć plik, system:

- ustala punkt startowy – katalog główny lub bieżący
- odczytuje pierwszy element ścieżki
- wyszukuje go w odpowiednim katalogu
- przechodzi do kolejnego poziomu
- na każdym etapie sprawdza prawa dostępu
- ostatecznie dociera do obiektu docelowego

Odnalezienie pliku nie jest jedną prostą operacją → sekwencją odwołań do katalogów i struktur metadanych

242

Architektura komputera – katalogi

Podstawowe operacje katalogowe obejmują:

- tworzenie katalogu
- usuwanie pustego katalogu
- listowanie zawartości
- zmianę katalogu bieżącego
- zmianę nazwy katalogu
- przenoszenie katalogu lub jego zawartości
- wyszukiwanie obiektów po nazwie.

Operacje te modyfikują przestrzeń nazw systemu plików, a niekoniecznie same dane użytkownika

243

Architektura komputera – katalogi

Katalog może być zaimplementowany na różne sposoby:

- **lista liniowa wpisów:** zawiera kolejne wpisy z nazwami plików i odwołaniami do ich metadanych – prostota implementacji, ale wyszukiwanie konkretnej nazwy wymaga często przeszukania wielu wpisów po kolei
- **tablica mieszająca (hash table):** nazwa pliku jest przekształcana przez funkcję mieszającą, co pozwala szybciej odnaleźć odpowiedni wpis – średni czas wyszukiwania może być znacznie krótszy niż przy liście liniowej
- **drzewo uporządkowane według nazw:** wpisy katalogowe są organizowane w strukturę drzewa, np. B-drzewa lub podobne struktury indeksujące – efektywne wyszukiwanie, dodawanie i usuwanie wpisów

244

Architektura komputera – implementacja systemu plików – model warstwowy

Aplikacja nie komunikuje się bezpośrednio z dyskiem, między programem użytkownika a fizycznym nośnikiem działa kilka warstw pośrednich, z których każda realizuje określone zadanie

Typowy przepływ:

- aplikacja wywołuje funkcję biblioteczną
- biblioteka wykonuje wywołanie systemowe
- jądro korzysta z **wirtualnego systemu plików (Virtual File System – VFS)**
- VFS przekazuje operację do konkretnego systemu plików
- sterownik urządzenia wykonuje operację na nośniku

VFS – warstwa pośrednia w jądrze systemu operacyjnego, która udostępnia jednolity interfejs dostępu do plików niezależnie od rodzaju systemu plików

245

Architektura komputera – dysk, partycja, wolumin

Przed utworzeniem systemu plików nośnik jest zwykle dzielony logicznie

Wyróżniamy:

- **dysk fizyczny** – rzeczywiste urządzenie magazynujące dane
- **partycję** – logicznie wydzielony obszar dysku
- **wolumin** – logiczny obszar udostępniany systemowi operacyjnemu

System plików jest tworzony na partycji lub woluminie = łatwiejsza organizacja danych, wielosystemowość i zarządzanie przestrzenią

Pojęcie woluminu może być szersze niż pojęcie partycji: obszary budowane programowo, np. przez RAID lub mechanizmy wirtualizacji pamięci masowej

246

Architektura komputera – partycjonowanie

Partycjonowanie – sposoby zapisywania informacji o partycjach

- **MBR – Master Boot Record** – zapis w pierwszym sektorze dysku (LBA 0), zawiera:
 - kod rozruchowy
 - tablicę partycji
 - sygnaturę sektora rozruchowego

Obsługuje zwykle maksymalnie 4 partycje podstawowe, więcej partycji – partycja rozszerzona i partycje logiczne

klasyczne sektory 512 B → praktyczny limit adresowanej przestrzeni wynosi około 2 TB

Brak rozbudowanych mechanizmów kontroli poprawności i redundancji metadanych

247

Architektura komputera – partycjonowanie

Partycjonowanie – sposoby zapisywania informacji o partycjach

GPT – GUID Partition Table – zapis na początku i na końcu dysku, zawiera:

- GUID partycji: Unikalny identyfikator (UUID) dla każdej partycji
- Typ partycji: Unikalny identyfikator GUID określający, do czego służy partycja (np. partycja systemowa EFI, partycja danych podstawowych Microsoft)
- Początkowy i końcowy LBA: Adresy, gdzie partycja się zaczyna i kończy
- Atrybuty: Informacje o atrybutach partycji (np. partycja tylko do odczytu, partycja ukryta)
- Nazwa partycji: Tekstowa nazwa partycji w formacie UTF-16.

Obsługuje dyski – do 18 eksabajtów (milionów TB); pozwala na zapis dużej liczby partycji; posiada kopię zapasową nagłówka i tablicy partycji na końcu dysku

248

Architektura komputera – implementacja pliku

Implementacja pliku wymaga rozwiązania trzech problemów:

- gdzie przechowywać metadane pliku
- jak wskazać bloki zawierające dane
- jak powiązać nazwę pliku z jego reprezentacją wewnętrzną

Realizowana za pomocą:

- rekordów opisujących plik
- wpisów katalogowych zawierających nazwę
- mechanizmów wskazywania bloków danych

Różne systemy plików rozwiązują te problemy w odmienny sposób, np. przez FAT, i-węzły lub rekordy MFT 249

Architektura komputera – metody alokacji plików

Najważniejsze metody alokacji:

- alokacja ciągła
- alokacja listowa
- alokacja z tabelą w pamięci
- alokacja indeksowa
- alokacja oparta na i-węzłach

Każda metoda jest kompromisem między: prostotą, szybkością dostępu, łatwością rozszerzania pliku, odpornością na fragmentację

250

Architektura komputera – alokacja ciągła

Alokacja ciągła – cały plik zajmuje jeden ciągły zakres bloków na nośniku

Plik zaczyna się w bloku b_start i ma długość n bloków, to zajmuje bloki:

$$b_start, b_start + 1, \dots, b_start + n - 1$$

Do opisanie pliku wystarczą dwa parametry:

- **numer pierwszego bloku**
- **liczba bloków albo całkowity rozmiar pliku**

Jeżeli rozmiar bloku wynosi B , odczytanie bajtu o przesunięciu $offset$, to: numer logicznego bloku pliku wynosi $offset / B$

System może łatwo przeliczyć pozycję logiczną w pliku na pozycję fizyczną na nośniku

251

Architektura komputera – alokacja ciągła

Alokacja ciągła – zalety: bardzo prosty opis pliku: początek + długość; bardzo szybki odczyt sekwencyjny; łatwy i tani dostęp losowy; niewielki narzut metadanych, dobra lokalność danych na dysku

Dyski HDD – zapis ciągły = mniejszą liczbę przeskoków głowicy i krótszy czas dostępu

Dyski SSD – zapis ciągły = uproszczone zarządzanie mapowaniem logicznym

Alokacja ciągła – wady: trudność w rozszerzaniu pliku; konieczność znajdowania odpowiednio dużego ciągłego obszaru wolnego miejsca; podatność na fragmentację zewnętrzną; potrzeba przenoszenia pliku podczas jego wzrostu

Rozmiar pliku rośnie → system musi sprawdzić, czy za przydzielonym zakresem znajdują się kolejne wolne bloki, jeżeli nie: trzeba przenieść plik w inne miejsce albo zarezerwować więcej miejsca albo porzucić czystą alokację ciągłą

252

Architektura komputera – alokacja na bazie listy jednokierunkowej

Alokacja listowa – plik jest reprezentowany jako łańcuch bloków rozproszonych na nośniku, blok danych zawiera:

- **fragment danych użytkownika**
- **wskaźnik do następnego bloku pliku**

System przechowuje zwykle adres pierwszego bloku i ewentualnie ostatniego bloku pliku

Kolejne bloki są odnajdywane przez podążanie za wskaźnikami

Jeżeli plik składa się z bloków $b_1 \rightarrow b_7 \rightarrow b_{19} \rightarrow b_4$, to logicznie tworzy jeden plik, mimo że fizycznie bloki są rozrzucone: plik może rosnać dynamicznie, nie trzeba rezerwować dużego ciągłego obszaru, wolne miejsce może być wykorzystywane bardziej elastycznie:

- aby dojść do k-tego bloku pliku, trzeba przejść przez wcześniejsze elementy łańcucha
- część miejsca w każdym bloku jest zajęta przez wskaźnik

253

Architektura komputera – alokacja na bazie listy jednokierunkowej

W alokacji listowej koszt dostępu zależy od typu operacji

Dostęp sekwencyjny

- po odczytaniu jednego bloku system zna adres następnego
- nie trzeba przechowywać dużych struktur indeksowych
- plik może być łatwo rozszerzany

Dostęp bezpośredni

- nie można łatwo obliczyć fizycznego adresu k-tego bloku
- trzeba przejść kolejne wskaźniki od początku łańcucha
- czas dostępu rośnie wraz z pozycją w pliku

254

Architektura komputera – alokacja na bazie listy jednokierunkowej

Alokacja listowa – zalety:

- brak konieczności przydzielania jednego dużego, ciągłego obszaru dysku
- łatwe rozszerzanie pliku przez dołączanie kolejnych bloków
- dobre wykorzystanie rozproszonej wolnej przestrzeni
- prosty model wzrostu pliku bez potrzeby jego natychmiastowego przenoszenia
- naturalne dopasowanie do dostępu sekwencyjnego

Alokacja listowa – wady:

- bardzo słaby dostęp losowy, słaba lokalność danych na dysku
- aby odczytać blok ze środka pliku, trzeba przejść przez wcześniejsze elementy łańcucha 255
- część miejsca w każdym bloku zajmuje wskaźnik do następnego bloku

Architektura komputera – alokacja z wykorzystaniem tabeli pamięci

Alokacja z tabelą pamięci – informacje o łańcuchu bloków do osobnej tablicy alokacji:

- każdy blok lub klaster ma odpowiadający mu wpis w tablicy
- wpis zawiera informację o następnym elemencie łańcucha
- specjalna wartość oznacza koniec pliku
- osobne oznaczenia mogą informować o bloku wolnym, uszkodzonym lub zarezerwowanym

Przykład: **FAT (File Allocation Table)**

- plik zaczyna się w klastrze 5, $FAT[5] = 9$, $FAT[9] = 17$, $FAT[17] = EOF$

Plik zajmuje łańcuch klastrów: $5 \rightarrow 9 \rightarrow 17$

256

Architektura komputera – alokacja z wykorzystaniem tabeli pamięci

Alokacja z tabelą pamięci – zalety:

- dane użytkownika nie są mieszane ze wskaźnikami
- łatwiej analizować strukturę pliku
- rozszerzanie pliku pozostaje proste
- system może śledzić stan całego woluminu przez jedną centralną strukturę

Alokacja z tabelą pamięci – wady:

- tablica alokacji może być bardzo duża dla dużych woluminów
- centralna struktura staje się krytycznym elementem systemu
- uszkodzenie FAT może utrudnić odtworzenie wielu plików
- dostęp losowy bez cache wymaga przechodzenia przez kolejne wpisy łańcucha

257

Architektura komputera – alokacja indeksowa

Alokacja indeksowa – adresy bloków należących do pliku są zapisywane w osobnym bloku indeksowym przypisanym do konkretnego pliku:

- każdy plik ma swój blok indeksowy
- blok indeksowy zawiera listę adresów wszystkich bloków danych pliku
- dane użytkownika są przechowywane oddzielnie od informacji sterujących
- system, znając adres bloku indeksowego, może odnaleźć wszystkie bloki pliku

Przykład: plik ma blok indeksowy I, w bloku I zapisano adresy bloków: 7, 21, 4, 33,

Plik zajmuje bloki danych: 7, 21, 4, 33

- blok indeksowy przechowuje mapę pliku
- bloki danych przechowują zawartość użytkownika

258

Architektura komputera – alokacja indeksowa

Alokacja indeksowa – zalety:

- dane użytkownika nie są mieszane ze wskaźnikami
- łatwiejszy dostęp bezpośredni niż w alokacji listowej
- nie ma potrzeby przechodzenia przez cały łańcuch bloków
- plik może być przechowywany w blokach rozproszonych po całym nośniku
- rozszerzanie pliku jest łatwiejsze niż w alokacji ciągłej
- blok indeksowy zawiera pełną mapę bloków należących do pliku

Alokacja indeksowa – wady:

- dla bardzo małych plików blok indeksowy może być częściowo niewykorzystany
- dodatkowy blok indeksowy stanowi narzut metadanych

259

Architektura komputera – I-węzły

W systemach z rodziny Unix i Linux plik jest opisywany przez **i-węzeł (inode)** – rekord metadanych zawierający najważniejsze informacje o obiekcie plikowym:

- typ pliku
- prawa dostępu
- identyfikator właściciela i grupy
- rozmiar pliku
- znaczniki czasu
- liczbę dowiązań
- wskaźniki do bloków danych

Nazwa pliku nie jest przechowywana w i-węźle

260

Architektura komputera – I-węzły

Nazwa pliku znajduje się w **katalogu**, który zawiera wpis: **nazwa pliku** oraz **numer i-węzła**

Katalog mapuje nazwę na numer i-węzła, i-węzeł opisuje sam plik, dane pliku są odnajdywane przez wskaźniki zapisane w i-węźle

Przykład:

- katalog zawiera wpis: raport.txt → inode 1532
- i-węzeł 1532 zawiera metadane oraz adresy bloków danych
- system najpierw znajduje i-węzeł, a potem odnajduje zawartość pliku

Model inode rozdziela:

- warstwę nazewniczą – katalogi i nazwy
- warstwę metadanych i adresowania – i-węzły
- warstwę danych – właściwe bloki pliku

Architektura komputera – I-węzły

Typowy i-węzeł może zawierać:

- kilka **wskaźników bezpośrednich** do bloków danych
- jeden wskaźnik **pojedynczo pośredni**
- jeden wskaźnik **podwójnie pośredni**
- jeden wskaźnik **potrójnie pośredni**

Zasada działania:

- dla małych plików wystarczają wskaźniki bezpośrednie
- dla większych plików system korzysta z bloków pośrednich zawierających kolejne adresy bloków danych

262

Architektura komputera – I-węzły

I-węzły – zalety:

- wyraźne rozdzielenie nazwy pliku od samego obiektu
- dobra organizacja metadanych
- łatwy dostęp do atrybutów pliku
- efektywna obsługa małych plików przez wskaźniki bezpośrednie
- możliwość obsługi bardzo dużych plików dzięki wskaźnikom pośrednim
- naturalne wsparcie dla **dowiązań twardych**
- katalogi przechowują tylko nazwy i numery i-węzłów, co upraszcza strukturę przestrzeni nazw

263

Architektura komputera – I-węzły

I-węzły – wady:

- struktura jest bardziej złożona niż prosta alokacja ciągła lub listowa
- dostęp do bardzo dużych plików może wymagać przechodzenia przez dodatkowe poziomy pośrednie
- liczba i-węzłów może być ograniczona przez parametry systemu plików
- można wyczerpać pulę i-węzłów nawet wtedy, gdy na dysku pozostaje jeszcze wolne miejsce
- uszkodzenie metadanych i-węzła może utrudnić dostęp do pliku

264

Architektura komputera – współdzielenie plików

Pliki w systemie operacyjnym mogą być współdzielone:

- przez wielu użytkowników
- przez wiele procesów
- przez różne aplikacje jednocześnie
- lokalnie lub przez sieć

Współdzielenie oznacza, że ten sam obiekt plikowy może być:

- odczytywany przez wiele procesów równocześnie
- modyfikowany przez jeden lub wiele procesów
- dostępny pod więcej niż jedną nazwą
- udostępniany różnym użytkownikom z różnymi prawami dostępu

Architektura komputera – współdzielenie plików

System operacyjny – mechanizmy współdzielenia plików:

- **prawa dostępu** – określają, kto może plik odczytać, zmodyfikować lub wykonać
- **blokady plików** – pozwalają ograniczyć jednoczesny zapis przez wiele procesów
- **licznik odwołań / liczba dowiązań** – informuje, ile nazw wskazuje na dany obiekt
- **deskryptory i uchwyty** – pozwalają procesom korzystać z otwartych plików
- **buforowanie i synchronizacja zapisu** – wpływają na widoczność zmian

Sytuacje problemowe:

- dwa procesy dopisują dane do tego samego pliku, jeden czyta plik, który drugi usuwa
- plik ma kilka nazw i użytkownik usuwa tylko jedną z nich
- plik jest używany lokalnie i jednocześnie przez system sieciowy

266

Architektura komputera – dowiązania

Dowiązania – jeden obiekt jest osiągalny pod więcej niż jedną nazwą:

1. Dowiązanie twarde (hard link) – kolejna nazwa tego samego obiektu plikowego:

- Dowiązanie twarde i plik „oryginalny” są równoważne
- nie ma jednej „głównej” nazwy pliku
- usunięcie jednej nazwy nie usuwa danych, jeśli istnieją inne odwołania
- liczba dowiązań jest przechowywana w metadanych pliku

W systemach Unix: różne wpisy katalogowe wskazują ten sam **i-węzeł**

Architektura komputera – dowiązania

Dowiązanie twarde:

Katalog zawiera wpis: raport.txt → inode 1532

Tworzymy dowiązanie twarde: raport_kopia.txt → inode 1532

W efekcie:

- obie nazwy wskazują **ten sam i-węzeł**
- obie nazwy odnoszą się do **tych samych danych**
- modyfikacja pliku przez jedną nazwę jest widoczna przez drugą
- usunięcie raport.txt nie usuwa danych, jeśli raport_kopia.txt nadal istnieje

268

Architektura komputera – dowiązania

Dowiązania – jeden obiekt jest osiągalny pod więcej niż jedną nazwą:

2. Dowiązanie symboliczne (symbolic link, symlink) – osobny obiekt, który przechowuje ścieżkę do innego pliku lub katalogu:

- dowiązanie symboliczne nie jest tym samym obiektem co plik docelowy
- działa jak specjalne odwołanie po ścieżce
- może wskazywać pliki i katalogi
- po usunięciu obiektu docelowego może pozostać „wiszącym” odwołaniem

Architektura komputera – dowiązania Linux a Windows

W systemach Linux i Unix - inode:

- dowiązania twarde wynikają bezpośrednio z architektury opartej na i-węzłach
- wpis katalogowy wiąże nazwę pliku z numerem i-węzła
- wiele wpisów katalogowych może wskazywać ten sam i-węzeł
- dowiązanie symboliczne jest osobnym obiektem przechowującym ścieżkę do pliku lub katalogu docelowego

W systemach Windows (NTFS) – uchwyt i listy ACL:

- NTFS obsługuje dowiązania twarde dla plików oraz dowiązania symboliczne
- dodatkowo stosowane są punkty połączeń, np. **junction points**
- skrót .lnk nie jest dowiązaniem systemu plików – powłoka Windows

270

Architektura komputera – systemy plików o strukturze dziennika

System plików o strukturze dziennika (*log-structured file system*) – nowe dane i nowe metadane są dopisywane sekwencyjnie do dziennika – główna struktura organizująca zapis

Zasada działania:

- system grupuje zmodyfikowane bloki danych i metadanych w segmenty
- segmenty są zapisywane sekwencyjnie na końcu dziennika
- po zapisaniu nowej wersji danych aktualizowane są struktury wskazujące, gdzie znajduje się najnowsza wersja bloku lub i-węzła
- starsze wersje bloków pozostają fizycznie na nośniku do czasu ich późniejszego usunięcia podczas czyszczenia segmentów

271

Architektura komputera – systemy plików o strukturze dziennika

System plików o strukturze dziennika (*log-structured file system*) – nowe dane i nowe metadane są dopisywane sekwencyjnie do dziennika – główna struktura organizująca zapis

Cechy techniczne:

- dominującą operacją jest dopisywanie, a nie nadpisywanie
- zapis ma charakter zbliżony do dużego sekwencyjnego transferu
- zmniejsza się liczba losowych zapisów małych bloków
- system musi utrzymywać informację, które segmenty zawierają dane aktualne, a które już nieaktualne

272

Architektura komputera – systemy plików o strukturze dziennika

System plików o strukturze dziennika – zalety:

- bardzo dobra wydajność zapisu sekwencyjnego
- dobre grupowanie wielu małych zapisów w większe operacje
- ograniczenie kosztu losowych zapisów

System plików o strukturze dziennika – wady:

- konieczność czyszczenia segmentów
- wzrost narzutu metadanych opisujących aktualność bloków
- potrzeba kopiowania nadal aktualnych danych z segmentów częściowo przestarzałych
- możliwe pogorszenie wydajności – system poświęca czas na reorganizację przestrzeni

Architektura komputera – systemy plików z księgowaniem

System plików z księgowaniem (*journaling file systems*) – dane i metadane są przechowywane w strukturze systemu plików, **dziennik (*journal*)** służy do zapisu informacji o operacjach modyfikujących stan systemu

Zasada działania:

- przed wykonaniem zmiany system zapisuje do dziennika informację o planowanej operacji
- po bezpiecznym zapisaniu wpisu w dzienniku może wykonać właściwą modyfikację danych lub metadanych
- po zakończeniu operacji wpis w dzienniku może zostać oznaczony jako zakończony
- po awarii system analizuje dziennik i odtwarza albo kończy operacje rozpoczęte przed przerwaniem pracy

274

Architektura komputera – systemy plików z księgowaniem

System plików z księgowaniem (*journaling file systems*) – dane i metadane są przechowywane w strukturze systemu plików, dziennik (*journal*) służy do zapisu informacji o operacjach modyfikujących stan systemu

Informacje księgowane:

- tylko metadane
- metadane wraz z uporządkowaniem zapisu danych
- rzadziej pełne dane użytkownika i metadane

Tryby pracy:

- księgowanie metadanych – zapis tylko zmian w metadanych
- tryb uporządkowany – zapis danych przed zatwierdzeniem metadanych
- pełne księgowanie danych – zapis metadanych i danych

275

Architektura komputera – systemy plików z księgowaniem

System plików z księgowaniem – zalety:

- szybsze odzyskiwanie spójności po awarii
- mniejsze ryzyko uszkodzenia katalogów, tablic alokacji i metadanych
- brak konieczności pełnego skanowania całego woluminu po każdym niepoprawnym wyłączeniu

System plików z księgowaniem – wady:

- dodatkowy narzut związany z zapisem do dziennika
- możliwe podwójne zapisanie części informacji
- księgowanie poprawia spójność, ale nie zastępuje kopii zapasowych

276

Architektura komputera – systemy plików z księgowaniem a dziennikiem

System plików z księgowaniem (journaling):

- dziennik – pomocniczym mechanizmem ochronnym (awarie, spójność danych)
- główna struktura systemu plików pozostaje klasyczna
- dane i metadane są finalnie zapisywane w swoich docelowych miejscach
- dziennik służy głównie do zachowania spójności po awarii

System plików o strukturze dziennika (log-structured):

- dziennik jest podstawowym mechanizmem organizacji zapisu
- nowe wersje danych i metadanych są dopisywane do dziennika jako głównej struktury systemu
- dziennik nie jest dodatkiem – centralny model przechowywania

277

Architektura komputera – wirtualny system plików

Wirtualny system plików (*Virtual File System, VFS*) – warstwa abstrakcji w jądrze systemu operacyjnego, która udostępnia jednolity interfejs operacji na plikach niezależnie od konkretnego typu systemu plików

Główne zadania VFS:

- udostępnienie wspólnych operacji: open, read, write, close, mkdir, unlink, rename
- ukrycie różnic implementacyjnych między ext4, XFS, FAT, NTFS, NFS, tmpfs, procfs itd.
- kierowanie wywołań do właściwego sterownika systemu plików
- utrzymywanie wspólnych struktur jądra opisujących otwarte pliki, i-węzły, wpisy katalogowe i punkty montowania

278

Architektura komputera – wirtualny system plików

Aplikacja:

- wywołuje funkcję biblioteczną
- trafia przez wywołanie systemowe do jądra
- jądro przekazuje operację do VFS
- VFS rozwiązuje ścieżkę, sprawdza strukturę montowania i wybiera implementację
- konkretna implementacja systemu plików wykonuje operację na własnych strukturach

Typowe obiekty VFS (Unix/Linux):

superblok – opis zamontowanego systemu plików; i-węzeł / węzeł wirtualny – opis obiektu plikowego; wpis katalogowy – wynik rozwiązywania nazwy; obiekt otwartego pliku – opis aktualnie otwartego pliku

279

Architektura komputera – montowanie systemu plików

Aby system plików był dostępny dla użytkownika i aplikacji, musi zostać **zamontowany**

Montowanie – logiczne włączeniu danego systemu plików do przestrzeni nazw systemu operacyjnego, tak aby jego zawartość była widoczna pod określoną ścieżką

System operacyjny podczas montowania:

- rozpoznaje typ systemu plików
- wiąże urządzenie lub wolumin z odpowiednim sterownikiem
- przypisuje mu miejsce widoczne w przestrzeni nazw
- udostępnia jego katalog główny procesom i użytkownikom

Bez zamontowania poprawnie sformatowany i technicznie sprawny wolumin nie jest dostępny w zwykły sposób

280

Architektura komputera – montowanie systemu plików Linux i Windows

W systemach Linux i Unix:

- różne systemy plików są dołączane do jednego wspólnego drzewa katalogów
- używa się punktów montowania, czyli katalogów, w których pojawia się zawartość danego woluminu
- typowe przykłady to: /, /home, /mnt, /media, /boot

W systemach Windows:

- woluminy najczęściej otrzymują litery dysków, np. C:, D:, E:
- użytkownik widzi je zwykle jako osobne logiczne jednostki pamięci
- możliwe są także punkty montowania w katalogach NTFS, choć nie są one domyślnym sposobem organizacji

281

Architektura komputera – zarządzanie systemem plików i optymalizacja

System plików musi stale utrzymywać informację o tym:

- które bloki są **zajęte**
- które bloki są **wolne**
- które bloki należą do konkretnych plików

Podstawowe zadania:

- przydzielanie bloków nowym plikom
- rozszerzanie istniejących plików
- zwalnianie miejsca po usunięciu danych
- aktualizacja struktur opisujących wolną przestrzeń

282

Architektura komputera – zarządzanie systemem plików i optymalizacja

Blok – podstawowa jednostka alokacji danych w systemie plików
Rozmiar bloku wpływa na wydajność i wykorzystanie miejsca:

Mały blok

- mniejsza strata miejsca przy małych plikach
- większa liczba bloków do zarządzania
- większy narzut metadanych i wskaźników

Duży blok

- lepsza wydajność dla dużych plików sekwencyjnych
- mniejsza liczba operacji wejścia-wyjścia
- większa strata miejsca przy małych plikach

283

Architektura komputera – zarządzanie systemem plików i optymalizacja

Fragmentacja wewnętrzna – plik nie wypełnia całego ostatniego przydzielonego bloku, strata miejsca wewnątrz już przydzielonej jednostki alokacji

Fragmentacja zewnętrzna – wolne miejsce istnieje, ale jest rozproszone w wielu małych, nieciągłych obszarach, utrudnia przydzielanie dużych ciągłych zakresów bloków

Skutki fragmentacji: trudniejsza alokacja nowych plików, spadek lokalności danych, więcej operacji wejścia-wyjścia, niższa wydajność - HDD

HDD a SSD:

- HDD – fragmentacja zwiększa liczbę ruchów głowicy
- SSD – fragmentacja wpływa na organizację danych, metadanych i zapisów – nie ma kosztu mechanicznego

284

Architektura komputera – zarządzanie systemem plików i optymalizacja

System plików musi wiedzieć, które bloki mogą zostać przydzielone nowym danym – potrzeba struktury opisującej wolną przestrzeń:

- **mapa bitowa** – 1 bit opisuje stan jednego bloku
- **lista wolnych bloków** – przechowuje adresy wolnych jednostek
- **grupowanie** – wolne bloki są opisywane w grupach
- **zliczanie zakresów** – zapisywane są ciągłe przedziały wolnej przestrzeni

Kryteria wyboru metody:

- narzut pamięciowy
- szybkość wyszukiwania wolnego miejsca
- łatwość znajdowania dużych ciągłych obszarów
- rozmiar woluminu i charakter obciążenia

Architektura komputera – zarządzanie systemem plików i optymalizacja

W systemach wieloużytkownikowych stosuje się **limity dyskowe** (*quota*), aby kontrolować zużycie zasobów przez użytkowników i grupy:

- łączny rozmiar danych użytkownika
- liczbę tworzonych plików
- wykorzystanie miejsca przez grupę użytkowników

Typy limitów:

- **limit miękki** – może być chwilowo przekroczony
- **limit twardy** – granica nieprzekraczalna

Architektura komputera – zarządzanie systemem plików i optymalizacja

System plików sam w sobie nie gwarantuje pełnego bezpieczeństwa danych
Do ochrony danych potrzebne są **kopie zapasowe**

Podstawowe rodzaje kopii:

- **pełna** – zapis całego zbioru danych
- **przyrostowa** – zapis zmian od ostatniej kopii
- **różnicowa** – zapis zmian od ostatniej kopii pełnej
- **migawka** (*snapshot*) – zapis stanu systemu plików w danym momencie

Dlaczego kopie zapasowe są konieczne:

- uszkodzenie fizyczne nośnika może uniemożliwić odzyskanie danych
- przypadkowe usunięcie pliku nie jest awarią systemu plików
- **księgowanie nie zastępuje backupu**

Architektura komputera – zarządzanie systemem plików i optymalizacja

Spójność systemu plików – zgodność między: strukturą katalogów, metadanymi plików, informacją o alokacji bloków, rzeczywistym stanem danych na nośniku

Przyczyny niespójności: zanik zasilania, awaria systemu operacyjnego, błąd sterownika, przerwanie zapisu w połowie operacji

Skutki niespójności: utracone bloki, błędne wpisy katalogowe, sierocie pliki, rozbieżność między metadanymi a danymi

Mechanizmy ochrony:

- księgowanie
- mechanizmy kopiowania przy zapisie (copy-on-write)
- narzędzia sprawdzania i naprawy

288

Architektura komputera – zarządzanie systemem plików i optymalizacja

- Do diagnozy i naprawy systemów plików używa się narzędzi analizujących metadane systemu plików:
- **fsck** (*file system check*) w Linuxie
- **chkdsk** (*check disk*) w Windows

Głównym celem jest przywrócenie spójności logicznej między: wpisami katalogowymi, rekordami metadanych plików, informacją o alokacji bloków, stanem wolnej i zajętej przestrzeni

Zadania narzędzi naprawczych: wykrywanie niespójności między katalogami a metadanymi, sprawdzanie poprawności tablic alokacji, i-węzłów, rekordów MFT lub bitmap wolnych bloków, naprawa uszkodzonych odwołań do bloków danych, wykrywanie osierocych plików i utraconych bloków, odtwarzanie części logicznej struktury katalogów, oznaczanie uszkodzonych obszarów nośnika

Dziękuję za uwagę

Treści programowe:

Sterowniki urządzeń i ich rola, mechanizmy wejścia/wyjścia oraz zarządzanie urządzeniami wejścia/wyjścia w systemie operacyjnym



291

Architektura komputera – podsystem wejścia/wyjścia

System operacyjny odpowiada za zarządzanie urządzeniami wejścia/wyjścia oraz za ukrycie ich sprzętowej różnorodności przed aplikacjami

Program użytkownika nie musi znać budowy kontrolera dysku, czy karty sieciowej

Powinien korzystać z ujednoliconego interfejsu udostępnianego przez jądro

Podsystem I/O obejmuje:

- abstrakcje urządzeń
- sterowniki
- obsługę przerwań
- buforowanie
- planowanie operacji wejścia/wyjścia
- mechanizmy oszczędzania energii

292

Architektura komputera – podsystem wejścia/wyjścia vs CPU

Typowy przepływ żądania:

aplikacja → biblioteka systemowa → wywołanie systemowe → jądro → warstwa I/O → sterownik → kontroler → urządzenie

Aplikacja zgłasza operację, np. odczyt pliku, zapis danych, pobranie znaku z klawiatury albo wyświetlenie obrazu

Jądro sprawdza poprawność parametrów, uprawnienia oraz stan urządzenia, a następnie przekazuje żądanie do odpowiedniej warstwy sterownika

Sterownik zna już szczegóły konkretnego sprzętu: rejestry, komendy, tryby pracy i sposób zgłaszania zakończenia operacji

Aplikacje są odseparowane od warstwy sprzętowej – system zachowuje kontrolę nad bezpieczeństwem i współdzieleniem urządzeń

293

Architektura komputera – podsystem wejścia/wyjścia vs CPU i RAM

CPU i pamięć operacyjna są z punktu widzenia systemu względnie jednorodne i przewidywalne

Urządzenia wejścia/wyjścia są bardzo różne:

- mają różne czasy odpowiedzi
- różne przepustowości
- różne jednostki transferu
- różne sposoby sygnalizacji gotowości i błędów
- różne tryby pracy: synchroniczne, asynchroniczne, blokowe, strumieniowe

Wiele urządzeń jest znacznie wolniejszych od CPU – dopasowanie szybkości pomiędzy procesorem a urządzeniem

294

Architektura komputera – podsystem wejścia/wyjścia – podstawowe cele

Podsystem I/O powinien realizować kilka celów jednocześnie:

1. Powinien zapewniać **niezależność od urządzenia**
Aplikacja nie powinna wiedzieć, czy dane są pobierane z HDD, SSD, terminala czy klawiatury USB
2. Powinien zapewniać **jednolity interfejs dostępu**
Zamiast programować rejestry kontrolera, proces korzysta z operacji typu open, read, write, close lub z odpowiedników systemowych
3. System musi zapewniać: ochronę dostępu, współdzielenie urządzeń, kolejkovanie operacji, buforowanie danych, obsługę błędów, dobrą wydajność

Dobrze zaprojektowany podsystem I/O powinien jednocześnie upraszczać programowanie i nie ograniczać zbytnio efektywności

295

Architektura komputera – klasy urządzeń wejścia/wyjścia

Z punktu widzenia systemu operacyjnego urządzenia można klasyfikować według modelu pracy:

- **Urządzenia blokowe** – przesyłają dane w blokach o określonym rozmiarze: dyski i pamięci masowe; istotne pojęcia bloku logicznego, kolejki żądań i planowania dostępu
- **Urządzenia znakowe** – przesyłają dane jako strumień bajtów lub znaków: terminale, porty szeregowy i część urządzeń wejściowych
- **Urządzenia zdarzeniowe** – generują zdarzenia, a nie klasyczny strumień bloków: klawiatura i mysz
- **Urządzenia sieciowe** – pracują na pakietach i ramach, a nie na modelu plikowym
- **Urządzenia czasowe** – dostarczają impulsów zegarowych, pomiaru czasu lub zdarzeń czasowych

296

Architektura komputera – warunki urządzeń wejścia/wyjścia

System operacyjny może obsługiwać urządzenie = sprzęt musi udostępniać przynajmniej podstawowy, kontrolowalny interfejs

Urządzenie lub jego kontroler powinno umożliwiać:

- identyfikację urządzenia
- inicjowanie operacji
- odczyt aktualnego stanu
- przesyłanie danych
- zgłaszanie zakończenia operacji
- raportowanie błędów.

Urządzenie powinno mieć zestaw rejestrów albo inny zdefiniowany interfejs programowy, przez który sterownik może kontrolować jego pracę

Architektura komputera – urządzenie a kontroler urządzenia

W wielu przypadkach CPU nie komunikuje się bezpośrednio z samym urządzeniem fizycznym – **kontroler urządzenia**

Kontroler jest elementem pośredniczącym:

- odbiera polecenia od procesora lub sterownika
- tłumaczy je na sygnały sprzętowe
- zarządza transferem danych
- raportuje status i błędy

Przykład: system operacyjny nie steruje bezpośrednio komórkami pamięci flash w SSD ani głowicą dysku HDD – steruje kontrolerem, który realizuje właściwe operacje wewnętrzne

298

Architektura komputera – rejestry urządzenia i ich funkcje

Klasyczny interfejs sprzętowy urządzenia opiera się na rejestrach:

- **Rejestr sterujący** – służy do wydawania poleceń, np. start transferu, reset urządzenia, wybór trybu pracy
- **Rejestr stanu** – zawiera informacje o gotowości, zajętości, zakończeniu operacji i błędach
- **Rejestr danych** – służy do przesyłania danych albo fragmentów danych
- **Rejestr błędów** – zawiera bardziej szczegółowy kod niepowodzenia
- **Rejestry pomocnicze** – mogą przechowywać adres bufora, długość transferu albo numer komendy

Sterownik zapisuje odpowiednie wartości do rejestrów, a następnie obserwuje zmianę stanu urządzenia – podstawowy model sterowania sprzętem na niskim poziomie

299

Architektura komputera – adresowanie interfejsu urządzeń

Klasyczne sposoby adresowania interfejsu urządzeń:

- **Port-mapped I/O** oznacza, że urządzenia mają osobną przestrzeń adresową wejścia/wyjścia – procesor korzysta ze specjalnych instrukcji do odczytu i zapisu portów
- **Memory-mapped I/O (MMIO)** oznacza, że rejestry urządzeń są mapowane do przestrzeni adresowej pamięci – CPU używa zwykłych instrukcji odczytu i zapisu, ale obszar taki nie jest traktowany jak zwykła pamięć RAM

MMIO upraszcza programowanie, ale wymaga ostrożności:

- nie wszystkie optymalizacje pamięci podręcznej są tu dopuszczalne
- czasem konieczne są bariery pamięci
- kolejność operacji musi być zachowana zgodnie z wymaganiami sprzętu

300

Architektura komputera – magistrala systemowa i transmisja I/O

Urządzenia wejścia/wyjścia muszą komunikować się z resztą systemu przez odpowiednią magistralę lub interfejs transmisyjny

Magistrala łączy CPU, pamięć operacyjną i kontrolery urządzeń – wspólną infrastrukturę wymiany danych

Z punktu widzenia systemu ważne są:

- przepustowość magistrali
- opóźnienie transmisji
- możliwość arbitrażu dostępu
- kompatybilność ze standardem urządzenia

Opóźnienie operacji I/O może wynikać z zajętości kontrolera, magistrali i pamięci

301

Architektura komputera – przerwania sprzętowe

Procesor nie sprawdza stale stanu każdego urządzenia – marnowanie zasobów

Urządzenia wejścia/wyjścia wspierają **przerwania sprzętowe**

Przerwanie pozwala urządzeniu zgłosić:

- zakończenie operacji
- wystąpienie błędu
- gotowość do dalszej transmisji
- nadejście danych wejściowych

CPU może wykonywać inne zadania – po zdarzeniu przechodzi do obsługi urządzenia

Przerwania = mechanizmasynchronicznej współpracy procesora z urządzeniami

302

Architektura komputera – szybki transfer danych

Duże transfery danych przez CPU są kosztowne – **DMA (Direct Memory Access)**

W tym modelu kontroler DMA lub sam kontroler urządzenia przenosi dane bezpośrednio między urządzeniem a pamięcią operacyjną

CPU nie wykonuje transferu, ale:

- przygotowuje parametry operacji
- wskazuje adres bufora
- określa kierunek i długość transferu
- uruchamia operację
- po zakończeniu odbiera przerwanie

DMA znacząco narzut CPU i jest niezbędne przy obsłudze szybkich interfejsów

303

Architektura komputera – wymagania jakościowe wobec I/O

Istnienie rejestrów i przerwań nie wystarcza – sprzęt I/O powinien być przewidywalny i poprawnie współpracować z systemem

Pożądane cechy sprzętu:

- jednoznaczne raportowanie błędów
- możliwość resetu i ponownej inicjalizacji
- stabilna konfiguracja zasobów
- zgodność ze standardem magistrali
- poprawna współpraca z mechanizmami ochrony pamięci
- możliwość wykrycia obecności lub braku urządzenia

Spójny i standardowy interfejs sprzętu = prostsze i bardziej niezawodne oprogramowanie sterownika

304

Architektura komputera – warunki oprogramowania I/O

Oprogramowanie wejścia/wyjścia powinno realizować jednocześnie cele funkcjonalne i systemowe: ma umożliwiać poprawną komunikację ze sprzętem oraz ma ukrywać złożoność tej komunikacji przed aplikacją

Najważniejsze wymagania:

- niezależność od konkretnego urządzenia
- jednolity interfejs dostępu
- bezpieczeństwo i kontrola uprawnień
- obsługa błędów
- buforowanie i kolejkovanie
- możliwość pracy synchronicznej i asynchronicznej
- dobra skalowalność przy wielu procesach i wielu urządzeniach

305

Architektura komputera – I/O – niezależność od urządzenia

Podstawowa zasada projektowania: niezależność od konkretnego urządzenia

Aplikacja nie powinna znać:

- numerów portów
- układu rejestrów
- sposobu sygnalizacji przerwań
- topologii magistrali
- wewnętrznej organizacji danych na urządzeniu

Program powinien operować na abstrakcjach udostępnianych przez system, np. pliku, deskryptorze, strumieniu wejścia albo gnieździe komunikacyjnym – działać na różnych konfiguracjach sprzętowych bez zmian w kodzie aplikacji

306

Architektura komputera – I/O – jednolity model dostępu

System operacyjny powinien dążyć do tego, aby wiele urządzeń było obsługiwanych podobnym zestawem operacji:

- otwarcie urządzenia
- zamknięcie urządzenia
- odczyt danych
- zapis danych
- operacje sterujące
- oczekiwanie na gotowość

W systemach Unix/Linux – wszystko jest plikiem – urządzenia są reprezentowane jako obiekty podobne do plików

307

Architektura komputera – I/O – dopasowanie prędkości

Procesor, pamięć i urządzenia I/O pracują z różnymi prędkościami – bezpośrednie przekazywanie każdego bajtu między procesem a urządzeniem prowadzi do dużego narzutu

System stosuje **buforowanie**: tymczasowe przechowywanie danych w pamięci operacyjnej, które pozwala:

- grupować małe operacje w większe transfery
- zmniejszać liczbę wywołań sprzętowych
- dopasowywać szybkość producenta i odbiorcy danych
- poprawiać przepustowość

Bufor = mechanizm organizacyjny i wydajnościowy != miejsce przechowania danych

308

Architektura komputera – I/O – ochrona i kontrola dostępu

Urządzenia wejścia/wyjścia są zasobami systemowymi – często współdzielone i potencjalnie wrażliwe

Nie każdy proces powinien mieć możliwość bezpośredniego sterowania dyskiem, kartą sieciową czy układem zasilania

System operacyjny musi egzekwować:

- prawa dostępu
- izolację między procesami
- politykę współdzielenia
- ograniczenie operacji uprzywilejowanych do trybu jądra

Sterowanie sprzętem jest realizowane przez jądro i sterowniki, a nie bezpośrednio przez zwykłe procesy użytkownika

309

Architektura komputera – warstwowy model I/O

Podsystem wejścia/wyjścia jest zorganizowany warstwowo – oddzielenie funkcji ogólnych od zależnych od sprzętu

Model uwzględnia:

- przestrzeń użytkownika
- warstwę niezależną od urządzenia w jądrze
- sterowniki urządzeń
- obsługę przerwań
- kontroler i sprzęt

Warstwowość: upraszcza utrzymanie systemu, oddziela politykę od mechanizmu

310

Architektura komputera – oprogramowanie I/O warstwa użytkownika

Nie wszystkie funkcje związane z I/O są wykonywane w jądrze – część działa w przestrzeni użytkownika w postaci bibliotek i usług systemowych: biblioteki standardowe I/O, kolejki wydruku, serwery dźwięku, systemy wyświetlania i narzędzia konfiguracyjne

Warstwa użytkownika:

- przygotowuje żądania w postaci wygodnej dla programu
- może realizować dodatkowe buforowanie
- ukrywa szczegóły wywołań systemowych
- organizuje część współdzielenia urządzeń na poziomie usług

Aplikacja korzysta z usług wyższej warstwy, a ta przekazuje odpowiednie operacje do jądra

311

Architektura komputera – oprogramowanie I/O warstwa niezależna

Warstwa niezależna od urządzenia realizuje funkcje wspólne dla wielu klas sprzętu – nie zna szczegółów konkretnego kontrolera – zna model pracy urządzenia i sposób reprezentacji obiektu I/O w systemie

Warstwa niezależna od urządzenia:

- ujednolica nazewnictwo urządzeń
- obsługuje deskryptory i uchwyt
- kontroluje uprawnienia
- obsługuje buforowanie i pamięć podręczną
- przekazuje żądania do odpowiedniego sterownika
- standaryzuje wyniki i kody błędów

312

Architektura komputera – sterownik urządzenia

Sterownik urządzenia – elementem programowym zależny od sprzętu w podsystemie I/O

Sterownik = tłumacz pomiędzy abstrakcyjnym modelem systemowym a konkretnym sprzętem

Sterownik zna:

układ rejestrów kontrolera, format komend, sposób inicjalizacji urządzenia, tryby pracy, kody błędów i stanów, sposób obsługi przerwań i DMA

Sterownik odpowiada za:

inicjalizację urządzenia, przygotowanie operacji odczytu, zapisu i sterowania, programowanie kontrolera, współpracę z obsługą przerwań, przekazywanie wyników do wyższych warstw jądra

313

Architektura komputera – typy sterowników urządzeń

Sterowniki można klasyfikować według rodzaju obsługiwanego sprzętu i sposobu integracji z systemem:

- **sterowniki blokowe** — obsługują urządzenia o dostępie blokowym, np. dyski
- **sterowniki znakowe** — obsługują strumienie bajtów, np. terminale
- **sterowniki sieciowe** — pracują na pakietach i kolejkach transmisyjnych
- **sterowniki magistral** — odpowiadają za wykrywanie i konfigurację urządzeń na danym interfejsie
- **sterowniki wirtualne** — udostępniają urządzenia logiczne, a nie fizyczne
- **sterowniki filtrujące** — przechwytyją i modyfikują operacje innych sterowników

Pojęcie sterownika jest szersze niż obsługa konkretnego urządzenia peryferyjnego

314

Architektura komputera – architektura I/O – obsługa przerwania

Urządzenie zgłasza zdarzenie → kontrola przechodzi do procedury obsługi przerwania

Procedura przerwania działa w uprzywilejowanym kontekście i musi wykonać działania konieczne do zachowania poprawności systemu

Zadania obsługi przerwania:

- identyfikacja źródła zgłoszenia
- potwierdzenie przyjęcia przerwania
- odczyt statusu urządzenia
- zamknięcie lub aktualizacja stanu operacji
- wybudzenie procesu albo wątku oczekującego na wynik

Obsługa przerwania = warstwa pośrednia między sprzętem a resztą jądra

315

Architektura komputera – architektura I/O – obsługa przerwania

Obsługa przerwania nie powinna być zbyt długa – zwiększa opóźnienia systemu i blokuje dalsze zdarzenia: systemy dzielą ją na część pilną i odroczoną

Górna połówka – działania najpilniejsze:

- potwierdza przerwanie
- pobiera krytyczny status
- zabezpiecza minimalny stan urządzenia

Dolna połówka – wykonuje pracę odroczoną:

- dalsze przetwarzanie danych
- aktualizację kolejek, budzenie procesów
- przekazywanie informacji do innych części jądra

316

Architektura komputera – architektura I/O – ścieżka odczytu

Operacja odczytu realizowana jest wieloetapowo:

1. Proces zgłasza żądanie do jądra, które sprawdza poprawność parametrów, deskryptor i uprawnienia
2. Warstwa I/O wybiera właściwy sterownik i przekazuje mu żądanie, sterownik:
 - przygotowuje parametry operacji
 - programuje kontroler
 - uruchamia transfer
 - ewentualnie konfiguruje DMA
3. Po zakończeniu urządzenie zgłasza przerwanie
4. System odczytuje status, umieszcza dane w odpowiednim buforze i zwraca wynik do procesu 317

Architektura komputera – architektura I/O – ścieżka zapisu

Operacja zapisu bywa myląca z punktu widzenia semantyki – zakończenie wywołania funkcji zapisu nie zawsze oznacza, że dane zostały już fizycznie zapisane na nośniku: system przyjął dane do bufora

Istotne różnice:

- dane mogą najpierw trafić do bufora jądra
- fizyczny zapis może nastąpić później
- trwałość może wymagać dodatkowego wymuszenia opróżnienia buforów

System rozdziela więc pojęcie **zakończenia logicznego** operacji od **zakończenia fizycznego**

318

Architektura komputera – mechanizm realizacji wejścia/wyjścia

1. **Sondowanie (polling)** – cykliczne sprawdzanie stanu urządzenia przez procesor: CPU odczytuje rejestr stanu i sprawdza, czy urządzenie jest gotowe, czy operacja została zakończona, albo czy pojawił się błąd

Zalety sondowania:

- prostota implementacji
- przewidywalny przebieg sterowania
- użyteczność w bardzo prostych systemach i krótkich odcinkach kodu startowego

Wady sondowania:

- marnowanie czasu CPU na oczekiwanie
- słaba skalowalność przy wielu urządzeniach
- nieefektywność dla operacji o dużych opóźnieniach

319

Architektura komputera – mechanizm realizacji wejścia/wyjścia

2. **Przerwania** – CPU inicjuje operację wejścia/wyjścia, a następnie może wykonywać inne zadania; gdy urządzenie zakończy operację, będzie gotowe do dalszej transmisji albo wykryje błąd, zgłasza przerwanie; procesor przechodzi wtedy do procedury obsługi przerwania, odczytuje stan urządzenia i kończy obsługę operacji

Zalety przerwania:

- efektywniejsze wykorzystanie czasu CPU, poprawa współbieżności
- brak konieczności ciągłego sprawdzania stanu urządzenia
- dobra obsługa zdarzeń pojawiających się nieregularnie

Wady przerwania:

- większa złożoność jądra i sterowników, zbyt duża liczba przerwania obciąża system
- konieczność obsługi procedur przerwania i synchronizacji

320

Architektura komputera – mechanizm realizacji wejścia/wyjścia

- 3. Bezpośredni dostęp do pamięci (DMA)** – brak konieczności kopiowania każdego fragmentu przez CPU; procesor przygotowuje operację, przekazuje kontrolerowi adres bufora, długość transferu i kierunek przesyłu; sprzęt wykonuje transfer samodzielnie; po zakończeniu operacji urządzenie lub kontroler zgłasza przerwanie

Zalety bezpośredniego dostępu do pamięci:

- zmniejszone obciążenie CPU, może wykonywać inne instrukcje w czasie transmisji
- poprawa przepustowości transferu danych

Wady bezpośredniego dostępu do pamięci:

- większa złożoność sprzętu i sterownika, spójność danymi między RAM i cache
- konieczność poprawnej konfiguracji adresu, długości i kierunku transferu

321

Architektura komputera – operacje blokujące i nieblokujące

System operacyjny może udostępniać operacje wejścia/wyjścia w trybie **blokującym** albo **nieblokującym** – różnica: co dzieje się z procesem po wywołaniu operacji, jeśli urządzenie lub dane nie są jeszcze gotowe

Operacja blokująca – wstrzymuje wykonanie procesu do chwili spełnienia warunku potrzebnego do kontynuacji: pojawienie się danych do odczytu, zwolnienie miejsca w buforze wyjściowym, zakończenie operacji zapisu, zakończenie pracy urządzenia

Proces nie wykonuje dalej własnego kodu użytkownika, dopóki operacja nie zakończy się sukcesem albo błędem

Przykłady:

odczyt z klawiatury w terminalu: proces czeka, aż użytkownik wpisze dane,

odczyt z potoku lub gniazda sieciowego: proces czeka, aż nadejdą dane

zapis do wolnego urządzenia lub pełnego bufora: proces czeka na możliwość wykonania zapisu

322

Architektura komputera – operacje blokujące i nieblokujące

System operacyjny może udostępniać operacje wejścia/wyjścia w trybie **blokującym** albo **nieblokującym** – różnica: co dzieje się z procesem po wywołaniu operacji, jeśli urządzenie lub dane nie są jeszcze gotowe

Operacja nieblokująca – zwraca sterowanie natychmiast, nawet jeśli w danym momencie nie da się wykonać żądania w całości; proces może wykonywać dalszy kod, ale sam sprawdza, czy operacja się powiodła, czy dane są gotowe, czy spróbować później:

- odczyt może zwrócić informację, że danych jeszcze nie ma
- zapis może zwrócić informację, że urządzenie lub bufor nie są gotowe
- aplikacja musi użyć mechanizmu sprawdzania gotowości albo zdarzeń

Przykład:

serwer obsługujący wielu klientów: nie może czekać na jednego klienta

323

Architektura komputera – operacje synchroniczne i asynchroniczne

Operacje wejścia/wyjścia można klasyfikować także jako **synchroniczne** i **asynchroniczne**

Podział: przebieg programu jest bezpośrednio związany z zakończeniem operacji I/O, czy też operacja może zostać zlecona i dokończona niezależnie od dalszego wykonywania kodu

Operacja synchroniczna – program otrzymuje wynik w bezpośrednim przebiegu wywołania i dalsze przetwarzanie logicznie zależy od zakończenia tej operacji:

- program zleca odczyt albo zapis
- czeka na wynik tej konkretnej operacji
- po jej zakończeniu kontynuuje wykonanie
- wynik lub błąd odbierany jest od razu w kontekście tego wywołania

Przykład: program odczytuje plik i po otrzymaniu danych wykonuje dalsze obliczenia

324

Architektura komputera – operacje synchroniczne i asynchroniczne

Operacje wejścia/wyjścia można klasyfikować także jako **synchroniczne** i **asynchroniczne**

Podział: przebieg programu jest bezpośrednio związany z zakończeniem operacji I/O, czy też operacja może zostać zlecona i dokończona niezależnie od dalszego wykonywania kodu

Operacja asynchroniczna – program zleca wykonanie operacji I/O, ale nie musi czekać na jej natychmiastowe zakończenie:

- program inicjuje operację
- system lub urządzenie realizuje ją niezależnie
- program może w tym czasie wykonywać inne zadania
- wynik jest odbierany później, po zgłoszeniu zakończenia

Przykład: serwer zleca kilka operacji sieciowych jednocześnie i odbiera wyniki później

325

Architektura komputera – kolejkovanie żądań wejścia/wyjścia

Wiele procesów może jednocześnie zgłaszać żądania do tego samego urządzenia – system operacyjny musi utrzymywać kolejkę operacji I/O i decydować, w jakiej kolejności będą one realizowane – kolejkovanie:

- zapewnienia sprawiedliwy dostęp
- poprawia przepustowość
- redukuje średni czas oczekiwania
- dopasowuje strategię obsługi do rodzaju urządzenia

Ustalenie kolejności żądań nie rozwiązuje problemu I/O – system musi zdecydować, gdzie tymczasowo przechowywać dane i zadania związane z tymi operacjami

326

Architektura komputera – buforowanie, pamięć podręczna, buforowanie zadań

Strategie przechowywania danych lub zadań:

- **Buforowanie (buffering)** – tymczasowe przechowywanie danych w trakcie transferu – dopasowanie szybkości producenta i odbiorcy danych, zmniejszenie liczby drobnych operacji, lepsza organizacja przesyłania danych między procesem a urządzeniem
- **Pamięć podręczna (cache)** – przechowywanie danych z nadzieją na ich ponowne użycie – ograniczenie liczby kosztownych odwołań do wolniejszego urządzenia i przyspieszenie dostępu do często używanych informacji
- **Buforowanie zadań (spooling)** – mechanizm kolejkowania zadań dla urządzenia współdzielonego, zwykle wolnego i używanego przez wielu użytkowników – zadania nie trafiają od razu bezpośrednio do urządzenia, lecz są najpierw gromadzone w obszarze pośrednim i realizowane kolejno

327

Architektura komputera – urządzenia I/O – pamięć masowa

Pamięć masowa jest z punktu widzenia systemu operacyjnego urządzeniem blokowym – dane są adresowane i przesyłane w blokach logicznych

Ta warstwa stanowi podstawę dla systemu plików – jądro i sterownik operują na blokach lub sektorach logicznych, a dopiero wyższe warstwy interpretują je jako pliki, katalogi i metadane

Wydajność pamięci masowej ma duży wpływ na cały system: zależą od niej uruchamianie programów, ładowanie bibliotek, zapisywanie plików, działanie baz danych i mechanizmy pamięci wirtualnej

328

Architektura komputera – urządzenia I/O – pamięć masowa

Dysk magnetyczny HDD – budowa i zasada działania

Dysk HDD przechowuje dane na wirujących talerzach pokrytych warstwą magnetyczną. Odczyt i zapis odbywają się za pomocą głowic przemieszczających się nad powierzchnią talerzy, dane organizowane są w ścieżki i sektory.

Z punktu widzenia systemu ważne są dwa mechaniczne składniki opóźnienia:

- czas pozycjonowania głowicy nad właściwą ścieżką
- opóźnienie obrotowe, czyli oczekiwanie, aż właściwy sektor znajdzie się pod głowicą

HDD oferuje dużą pojemność i niski koszt jednostkowy, ale jego charakterystyka czasowa jest silnie zależna od mechaniki.

329

Architektura komputera – urządzenia I/O – pamięć masowa

Dysk magnetyczny HDD – budowa i zasada działania

Wydajność dysku magnetycznego opisują przede wszystkim:

- **seek time** – czas przemieszczenia głowicy
- **rotational latency** – czas oczekiwania na właściwy obrót talerza
- **transfer time** – czas samego przesłania danych
- **access time** – łączny czas dotarcia do danych i ich odczytu

Dla HDD dostęp sekwencyjny jest zwykle dużo szybszy niż losowy

Przy dostępie losowym dominują opóźnienia mechaniczne, a nie sama transmisja danych – kolejność żądań ma duże znaczenie dla wydajności

330

Architektura komputera – urządzenia I/O – pamięć masowa

Dysk magnetyczny HDD – budowa i zasada działania

Ruch głowicy jest kosztowny – system operacyjny może stosować algorytmy planowania żądań dyskowych:

- **FCFS (First Come, First Served)** realizuje żądania w kolejności nadejścia: prosty i sprawiedliwy, ale nie zawsze efektywny
- **SSTF (Shortest Seek Time First)** wybiera żądanie najbliższe aktualnemu położeniu głowicy: poprawia średni czas obsługi, ale może prowadzić do głodzenia odległych żądań
- **SCAN** działa jak winda: głowica przesuwa się w jednym kierunku, obsługując kolejne żądania
- **C-SCAN** obsługuje żądania w jednym kierunku, a potem szybko wraca na początek: bardziej równomierne czasy oczekiwania

331

Architektura komputera – urządzenia I/O – pamięć masowa

Dysk SSD – zasada działania

SSD nie ma elementów mechanicznych, dane przechowywane są w pamięci flash NAND, a odczyt i zapis realizowane są elektronicznie: opóźnienia dostępu losowego są znacznie mniejsze niż w HDD

Ograniczenia SSD:

- zapis odbywa się zwykle stronami
- kasowanie odbywa się blokami
- liczba cykli zapisu komórek jest ograniczona
- kontroler musi zarządzać mapowaniem logicznych adresów na fizyczne komórki

Z punktu widzenia systemu SSD wygląda jak urządzenie blokowe – fizyczna organizacja jest inna niż w przypadku HDD

332

Architektura komputera – urządzenia I/O – pamięć masowa

Warstwa translacji pamięci flash (Flash Translation Layer, FTL) tłumaczy logiczne adresy bloków widziane przez system operacyjny na fizyczne komórki pamięci flash – dysk SSD może emulować klasyczny model urządzenia blokowego

Równoważenie zużycia (wear leveling) polega na równomiernym rozkładaniu zapisów pomiędzy różne obszary pamięci – pamięć flash ma ograniczoną liczbę cykli zapisu i kasowania

Oczyszczanie pamięci (garbage collection) polega na reorganizacji danych i odzyskiwaniu bloków, które mogą zostać ponownie użyte do zapisu – pamięć flash nie pozwala na dowolne nadpisywanie danych bajt po bajcie bez wcześniejszego przygotowania odpowiednich bloków

333

Architektura komputera – urządzenia I/O – pamięć masowa

System plików usuwa plik → wie, które bloki logiczne nie są już potrzebne

Dysk SSD nie otrzyma tej informacji → traktuje te dane jako ważne – utrudnia późniejsze porządkowanie pamięci i przygotowanie miejsca do kolejnych zapisów

Oznaczanie nieużywanych bloków (TRIM) pozwala systemowi operacyjnemu poinformować dysk SSD, że określone bloki logiczne mogą zostać uznane za zwolnione: kontroler może wcześniej przygotować wolne miejsce do przyszłych operacji zapisu

Zalety **TRIM**:

- lepsza długookresowa wydajność dysku
- sprawniejsze działanie mechanizmu oczyszczania pamięci (**garbage collection**)
- mniej niepotrzebnego kopiowania danych wewnątrz SSD
- mniejsze pogorszenie wydajności po dłuższym okresie użytkowania

Architektura komputera – urządzenia I/O – zegary i liczniki

System operacyjny potrzebuje zegarów i liczników czasu (timers) do sterowania przebiegiem wykonywania zadań oraz do realizacji usług zależnych od czasu:

- odmierzanie czasu rzeczywistego (kalendarzowego)
- odmierzanie kwantu czasu procesora (time slice, time quantum)
- realizacja funkcji usypiania procesu (sleep)
- budzenie procesów po zadany czasie
- obsługa limitów czasu oczekiwania (timeouts)
- realizacja retransmisji w systemach sieciowych
- pomiar czasu wykonania programu i profilowania

Czas jest zasobem sterującym działaniem planisty, synchronizacji i usług systemowych

335

Architektura komputera – urządzenia I/O – typy zegarów i liczników

- **Zegar czasu rzeczywistego (Real-Time Clock, RTC)** przechowuje czas kalendarzowy (data i godzina); działa niezależnie od bieżącego planowania procesów i jest wykorzystywany m.in. podczas uruchamiania systemu do odczytu bieżącego czasu
- **Zegar systemowy (system clock)** dostarcza impulsów czasowych używanych przez jądro do odmierzenia upływu czasu działania systemu; system aktualizuje liczniki czasu i obsługuje mechanizmy planowania
- **Licznik czasu okresowego (periodic timer)** generuje zdarzenia cykliczne, np. co określoną liczbę milisekund
- **Licznik czasu jednorazowego (one-shot timer)** generuje zdarzenie tylko raz, po upływie zadanego czasu
- **Liczniki czasu wysokiej rozdzielczości (high-resolution timers)** stosowane przy potrzebie większej precyzji np. w systemach multimedialnych lub pomiarowych

336

Architektura komputera – urządzenia I/O – przerwania zegarowe

Przerwanie zegarowe (timer interrupt, clock interrupt) – sygnał generowany przez układ czasowy, który powoduje przejście procesora do kodu jądra obsługującego zdarzenia czasowe: pozwala jądru okresowo odzyskiwać kontrolę nad wykonywaniem systemu

Przerwania zegarowe służą do:

- aktualizacji czasu systemowego
- odmierzania kwantu czasu procesora
- wywłaszczania procesów w systemach z planowaniem wywłaszczającym
- sprawdzania, czy upłynęły zadane limity czasu oczekiwania
- budzenia procesów po zakończeniu uśpienia

337

Architektura komputera – urządzenia I/O – dokładność czasu

Częste przerwania zegarowe – dokładniejsze reakcje systemu na zdarzenia czasowe –
większy narzut na procesor i spadek efektywności energetycznej

Zbyt rzadka obsługa licznika czasu:

- pogarsza dokładność funkcji usypiania
- zmniejsza precyzję limitów czasów oczekiwania
- pogarsza responsywność systemu

Zbyt częsta obsługa licznika czasu:

- zwiększa liczbę przerwania
- podnosi koszt przejścia do jądra i powrotu do procesu
- zwiększa obciążenie CPU pracą administracyjną

338

Architektura komputera – urządzenia I/O – interfejs użytkownika

Klawiatura – **urządzenie zdarzeniowe**: nie udostępnia danych w postaci sektorów ani bloków – generuje zdarzenia odpowiadające naciśnięciu i zwolnieniu klawisza

Ścieżka obsługi wejścia z klawiatury:

- kontroler klawiatury wykrywa zmianę stanu klawisza
- generowany jest kod skanowania (scan code): fizyczne naciśnięcie lub zwolnienie klawisza
- sprzęt zgłasza przerwanie
- sterownik odbiera zdarzenie i interpretuje je
- wyższa warstwa systemu mapuje je na znak, skrót klawiaturowy albo akcję interfejsu

kod fizyczny klawisza – konkretny przycisk

kod logiczny – zależny od układu klawiatury, języka oraz stanu klawiszy modyfikujących, takich jak Shift, Ctrl i Alt

Architektura komputera – urządzenia I/O – interfejs użytkownika

Mysz – **urządzenie zdarzeniowe**: Nie dostarcza obrazu położenia – generuje serię raportów opisujących aktywność użytkownika – sterownik przekazuje raporty do warstwy obsługi interfejsu użytkownika

Mysz zgłasza:

- ruch wskaźnika
- naciśnięcia i zwolnienia przycisków
- ruch rolki przewijania
- dodatkowe przyciski, gesty lub dane z czujnika dotykowego

Ruch może być raportowany:

- względnie – zmiana położenia względem poprzedniej pozycji
- bezwzględnie – konkretna pozycja na powierzchni roboczej

340

Architektura komputera – urządzenia I/O – interfejs użytkownika

Monitor – urządzenie wyjściowe – obraz trafia do struktury pośredniej: bufor ramki albo pamięć karty graficznej

Bufor ramki – obszar pamięci zawierający reprezentację obrazu, który ma zostać wyświetlony na ekranie

Model uproszczony = tablica pikseli przechowująca kolor każdego punktu obrazu

Typowy tor przetwarzania:

- aplikacja generuje zawartość obrazu
- biblioteka lub API graficzny przygotowuje operacje rysowania
- system okien lub kompozytor zarządza układem treści na ekranie
- sterownik grafiki komunikuje się z GPU i monitorem
- monitor odświeża wynikowy obraz

341

Architektura komputera – urządzenia I/O – interfejs użytkownika

Sterownik grafiki: tryb pracy urządzenia, rozdzielczość, częstotliwość odświeżania, współpraca z akceleracją sprzętową, przekazywanie gotowego obrazu do warstwy wyświetlania

Obraz nie jest rysowany bezpośrednio w pamięci aktualnie wyświetlanej na ekranie: podwójne buforowanie – dwa buforów obrazu:

- bufor przedni (front buffer) – aktualnie wyświetlany
- bufor tylny (back buffer) – używany do przygotowania nowej wersji obrazu

Po zakończeniu renderowania następuje zamiana ról buforów

Podwójne buforowanie: zmniejsza migotanie, poprawia płynność animacji, zapewnia spójniejszy obraz, ułatwia współpracę z synchronizacją odświeżania ekranu

Połączenie wymagań sprzętowych i wymagań interfejsu użytkownika

342

Architektura komputera – urządzenia I/O – zarządzanie energią

Cele zarządzania energią:

- wydłużenie pracy na baterii
- zmniejszenie poboru mocy
- ograniczenie temperatury pracy
- zmniejszenie hałasu generowanego przez chłodzenie
- poprawę trwałości elementów sprzętu
- obniżenie kosztów eksploatacji

System: zbyt agresywne oszczędzanie energii może pogarszać responsywność, zwiększać opóźnienia wybudzenia urządzeń i obniżać wydajność

343

Architektura komputera – urządzenia I/O – zarządzanie energią

System operacyjny może oszczędzać energię na wielu poziomach: procesora, pamięci, urządzeń peryferyjnych i całej platformy sprzętowej – część decyzji podejmuje jądro, a część realizowana jest przez sprzęt zgodnie z udostępnionym modelem zasilania

Mechanizmy:

- usypianie nieaktywnych urządzeń
- wygaszanie ekranu
- ograniczanie aktywności interfejsów komunikacyjnych
- obniżanie częstotliwości i napięcia CPU
- przełączanie procesora i urządzeń do stanów niższego poboru mocy
- dobór polityki energetycznej zależnie od bieżącego obciążenia

344

Architektura komputera – urządzenia I/O – zarządzanie energią

Podstawowe pojęcia: **stany wydajności (performance states, P-states)** oraz **stany bezczynności (idle states, C-states)**

- **Stany wydajności (P-states)** opisują poziomy wydajności procesora, związane z częstotliwością taktowania i napięciem: niższa częstotliwość i napięcie oznaczają mniejszy pobór energii, ale także niższą wydajność obliczeniową
- **Stany bezczynności (C-states)** opisują poziomy „uśpienia” procesora wtedy, gdy CPU chwilowo nie wykonuje użytecznej pracy: głębszy stan bezczynności – mniejsze zużycie energii, dłuższy czas powrotu do pełnej aktywności

Analogicznie dla urządzeń peryferyjnych: dysk, karta sieciowa, interfejs USB czy ekran mogą przechodzić do stanów ograniczonego poboru mocy, jeśli system uzna, że chwilowo nie są potrzebne

345

Architektura komputera – urządzenia I/O – zarządzanie energią

System operacyjny może stosować **globalne stany zasilania**, które obejmują większą część platformy sprzętowej:

- **Uśpienie do pamięci RAM (suspend to RAM)** – większość urządzeń zostaje wyłączona lub przechodzi do stanu niskiego poboru mocy, ale zawartość pamięci RAM jest nadal podtrzymywana: powrót do pracy jest szybki, lecz zużycie energii nie spada do zera
- **Hibernacja, czyli uśpienie do dysku (suspend to disk)** – zapisanie stanu pamięci operacyjnej na nośniku i możliwość niemal całkowitego wyłączenia zasilania: dużo mniejsze zużycie energii, ale czas wznowienia pracy zwykle dłuższy

Zarządzanie energią jest kompromisem:

- zwiększenie oszczędności = dłuższy czas powrotu do pełnej gotowości
- większa responsywność = wyższy pobór mocy

346

Dziękuję za uwagę

Treści programowe:

Podstawy konfiguracji sieci, narzędzia do konfiguracji sieci w różnych systemach operacyjnych oraz zapory sieciowe i ich konfiguracja



348

Architektura komputera – podstawy sieci

Sieć komputerowa:

Tanenbaum (1996):

Zbiór zlokalizowanych oddzielnie komputerów połączonych w celu wykonania określonego zadania

Wikipedia:

Grupa komputerów i innych urządzeń połączonych z sobą kanałami komunikacyjnymi oraz oprogramowanie wykorzystywane w tej sieci. Umożliwia ona wymianę danych lub współdzielenia różnych zasobów

349

Architektura komputera – podstawy sieci – modele

Komunikacja urządzeń w sieci komputerowej składa się z kilku etapów

Każdy z nich realizuje inne zadania niezbędne do poprawnej komunikacji

Etapy określone są przez **modele warstwowe**

Istnieją dwa modele warstwowe:

- Model **TCP/IP** – model protokołów, każda z warstw wykonuje konkretne zadania za pomocą odpowiednich protokołów
- Model **ISO/OSI** – model odniesienia, stosowany do analizy w celu zrozumienia procesów komunikacyjnych

350

Architektura komputera – podstawy sieci – modele

Model TCP/IP – cztery warstwy

1. **Warstwa aplikacji** – protokoły działające najbliżej użytkownika i aplikacji:
HTTP/HTTPS, DNS, DHCP, SSH, FTP/SFTP
2. **Warstwa transportowa** – komunikacja między procesami: **TCP** – połączeniowy, uporządkowany, retransmisje, kontrola przepływu, **UDP** – bezpołączeniowy, szybszy, bez gwarancji dostarczenia
3. **Warstwa internetowa** – **IP** – adresacja i routing pakietów, **ICMP** – komunikaty kontrolne i diagnostyczne, **ARP** (dla IPv4) / **NDP** (dla IPv6) – mapowanie adresów logicznych na adresy warstwy łącza
4. **Warstwa dostępu do sieci** – Ethernet, Wi-Fi, PPP, sterowniki kart sieciowych i ramki

351

Architektura komputera – podstawy sieci – modele

Model OSI (siedem warstw) ma znaczenie dydaktyczne i analityczne

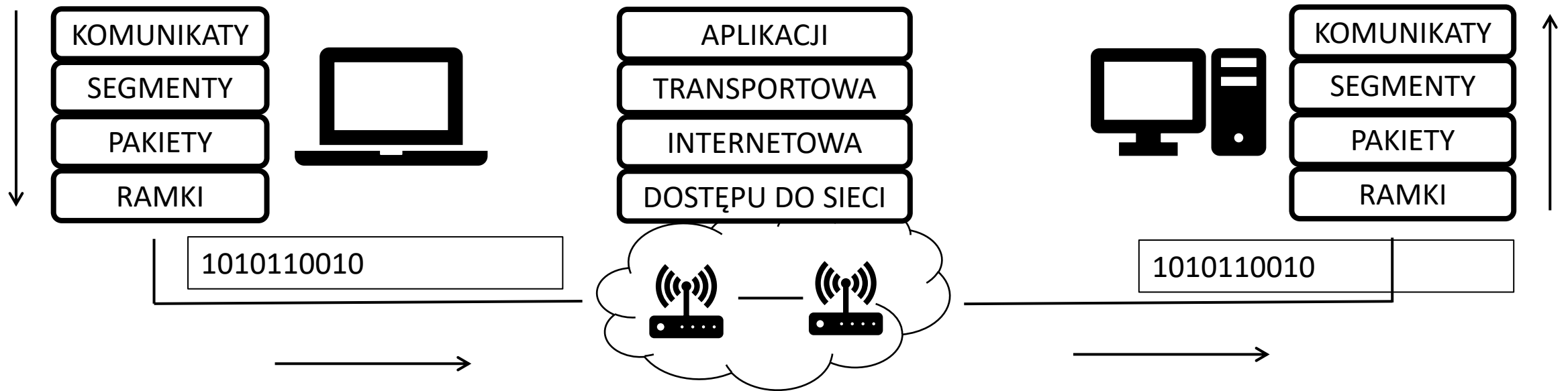
Przybliżone mapowanie

- **TCP/IP Aplikacja** $\approx$ **OSI Aplikacja + Prezentacja + Sesja**
- **TCP/IP Transport** $\approx$ **OSI Transport**
- **TCP/IP Internet** $\approx$ **OSI Sieć**
- **TCP/IP Dostęp do sieci** $\approx$ **OSI Łączy danych + Fizyczna**

Posługujemy się bezpośrednio pojęciami: adres IP, port, gniazdo, trasa, brama, DNS, interfejs, firewall

352

Architektura komputera – podstawy sieci – enkapsulacja i dekapstylacja



353

Architektura komputera – podstawy sieci – adresacja

IP (Internet Protocol) – podstawowy protokół warstwy sieciowej w modelu TCP/IP

Adres IP identyfikuje jednoznacznie urządzenie w sieci:

- identyfikacja hosta (komputera, routera, telefonu itp.)
- określenie lokalizacji w sieci (do jakiej sieci należy host)

Adresacja IPv4:

- 32 bity → zapis dziesiętny, 4 oktety: 192.168.1.10
- Zakres: ok. 4,3 mld adresów

Adresacja IPv6:

- 128 bitów → zapis szesnastkowy: 2001:0db8:85a3::8a2e:0370:7334
- Zakres: praktycznie niewyczerpywalny

354

Architektura komputera – podstawy sieci – adresacja

- Adres IPv4 ma **32 bity**, zapisane jako **4 oktety dziesiętne**

Przykład adres IPv4: 192.168.1.10

Część sieciowa (Network ID)

- Wskazuje, do której sieci należy host
- Wszystkie hosty w tej samej sieci mają identyczny Network ID

Część hosta (Host ID)

- Identyfikuje konkretne urządzenie w obrębie sieci
- Musi być unikalny w danej sieci (nie mogą istnieć dwa takie same Host ID)

Przykład prefiks i maska: /24 = 255.255.255.0

Maska określa: część sieciową adresu, część hosta, liczbę adresów w podsieci

355

Architektura komputera – podstawy sieci – adresacja

Adres IPv4:	192.168.1.10/24
Prefiks i maska:	/24 = 255.255.255.0
Część sieciowa (Network ID):	192.168.1.0
Część hosta (Host ID):	10
Adres sieci:	192.168.1.0
Adres rozgłoszeniowy:	192.168.1.255
Liczba hostów:	$254 = 2^{32-24} - 2 = 2^8 - 2 = 256 - 2$
Zakres hostów:	192.168.1.1 – 192.168.1.254

Architektura komputera – podstawy sieci – adresacja

Początek lat 80 – podział wszystkich adresów IPv4: 0.0.0.0 – 255.255.255.255 na klasy:

Klasa A: zakres: 0.0.0.0 – 127.255.255.255, maska: 255.0.0.0 (/8),
liczba hostów: ~16 milionów w jednej sieci

Klasa B: zakres: 128.0.0.0 – 191.255.255.255, maska: 255.255.0.0 (/16),
liczba hostów: ~65 tys. w jednej sieci

Klasa C: zakres: 192.0.0.0 – 223.255.255.255, maska: 255.255.255.0 (/24),
liczba hostów: 254 w jednej sieci

Klasa D: zakres: 224.0.0.0 – 239.255.255.255 (multicast)

Klasa E: zakres: 240.0.0.0 – 255.255.255.255 (eksperymentalne, zarezerwowane)

Lata 90 – wprowadzono CIDR (bezklasowe trasowanie międzydomenowe): brak klas –
sieć można podzielić dowolnie za pomocą maski, zapis: **adres_sieci/maska**

357

Architektura komputera – podstawy sieci – adresacja

Sieci prywatne (RFC 1918) – adresy zarezerwowane do użytku wewnętrznego – nie są routowane bezpośrednio w internecie (NAT):

- 10.0.0.0 – 10.255.255.255 (/8)
- 172.16.0.0 – 172.31.255.255 (/12)
- 192.168.0.0 – 192.168.255.255 (/16)

Adresy specjalne:

- 127.0.0.0/8 – loopback, zwykle 127.0.0.1
- 169.254.0.0/16 – APIPA, automatyczna adresacja przy braku DHCP
- 0.0.0.0 – adres nieokreślony / używany m.in. w trasie domyślnej,

Typowy domowy router dostaje adres z zakresu **192.168.x.x**

358

Architektura komputera – podstawy sieci – adresacja

Adres IPv6 ma 128 bitów, zapisany w 8 grupach po 16 bitów w systemie szesnastkowym

Przykład adres IPv6: 2001:0db8:0000:0000:0000:ff00:0042:8329

Skracanie zapisu, można usuwać zera wiodące, jeden ciąg zer można zastąpić ::.

Przykład adres IPv6: 2001:db8::ff00:42:8329

Typowe klasy adresów

- Global Unicast – routowalne globalnie
- Link-Local – lokalne dla łącza, automatycznie na każdym interfejsie
- Unique Local – odpowiednik prywatnych zakresów IPv4
- Typowy prefiks LAN: /64

359

Architektura komputera – podstawy sieci – adresacja

Adresacja IPv6 vs adresacja IPv4

- brak klasycznego broadcast – zamiast tego multicast i anycast
- większa przestrzeń adresowa
- uproszczony nagłówek podstawowy
- autokonfiguracja SLAAC
- szerokie użycie NDP zamiast ARP
- w praktyce host ma zwykle jednocześnie adres IPv4 i IPv6

Aktualnie w systemie operacyjnym analizuje się równolegle obydwie konfiguracje:
osobno trasę IPv4, osobno trasę IPv6, osobno reguły zapory dla obu rodzin protokołów

360

Architektura komputera – podstawy sieci – konfiguracja

Konfiguracja statyczna vs dynamiczna (DHCP):

Konfiguracja statyczna:

- administrator ręcznie ustawia IP, maskę, bramkę, DNS
- zalety: przewidywalność, serwery, drukarki
- wady: dużo pracy, błędy konfiguracji

Konfiguracja dynamiczna (DHCP):

- klient prosi serwer DHCP o adres
- serwer przydziela IP + maskę + bramkę + DNS + lease time
- zalety: łatwe zarządzanie dużą liczbą urządzeń
- wady: zależność od serwera DHCP

Architektura komputera – podstawy sieci – konfiguracja

Konfiguracja dynamiczna (DHCP):

- Działa w warstwie aplikacyjnej – używa UDP (67 - serwer, 68 - klient)
- RFC 2131 (1997) – definicja DHCP IPv4, RFC 8415 – definicja DHCPv6 IPv6
- Rozszerzenie protokołu BOOTP (Bootstrap Protocol)

Automatyczne przydzielanie:

- Adresów IP
- Maski podsieci
- Bramy domyślnej - router przekazujący ruch z sieci lokalnej do innych sieci
- Adresu DNS (Domain Name System) - serwer tłumaczący nazwy domen na adresy IP
- Adresu NTP (Network Time Protocol) - serwer synchronizacji czasu

362

Architektura komputera – podstawy sieci – konfiguracja

Proces DORA (Discover–Offer–Request–Acknowledge):

- *Wykrywanie serwera DHCP* (Discover) – komunikat wykrywania serwera DHCP: klient pyta „kto może mi dać IP?” – rozgłaszanie do wszystkich hostów w sieci – adres rozgłaszania 255.255.255.255, adres źródłowy to 0.0.0.0
- *Oferta DHCP* (Offer) – serwer przesyła klientowi komunikat oferty DHCP (adres IP + konfiguracja) oraz okres dzierżawy adresu IP (czas, przez który adres IP jest prawidłowy) – przekazywany do wszystkich węzłów w sieci
- *Żądanie DHCP* (Request) – klient wybiera ofertę (może być jedna z wielu) i prosi o przydzielenie – komunikat żądania DHCP z powtórzonymi parametrami konfiguracji
- *Potwierdzenie DHCP* (Acknowledge) – serwer odpowiada wysłaniem komunikatu potwierdzenia DHCP – potwierdza żądane parametry

363

Architektura komputera – sieci – podstawowe narzędzia diagnostyczne

ping – testuje połączenie sieciowe za pomocą komunikatów ICMP Echo Request i Echo Reply (wszystkie systemy operacyjne)

tracert / traceroute – śledzi trasę pakietów poprzez zmniejszanie wartości TTL i odbieranie komunikatów ICMP Time Exceeded (Linux/macOS: `tracert`, Windows: `tracert`)

ip addr / ip link – wyświetla konfigurację interfejsów sieciowych wraz z adresami IP i stanem (Linux)

ip route – wyświetla i modyfikuje tabelę routingu w systemie (Linux, Windows: `route`)

ss -tuln / netstat -tuln – pokazuje otwarte porty nasłuchujące wraz z protokołami (Linux)

Get-NetAdapter – wyświetla listę kart sieciowych i ich stan (PowerShell – Windows)

Test-NetConnection – testuje połączenie, porty i trasę (PowerShell – Windows)

364

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

ifconfig i **route** – stare, przestarzałe narzędzia, usunięte w wielu nowoczesnych dystrybucjach

iproute2 (polecenie **ip**) – aktualne i zalecane narzędzie od jądra Linux 2.2

NetworkManager (nmcli, nmtui, GUI) – najpopularniejsze w środowiskach desktopowych

systemd-networkd – lekka, wbudowana w systemd usługa zarządzania siecią, stosowana głównie na serwerach, w kontenerach (Docker, Podman, LXC) oraz systemach wbudowanych, gdzie nie jest potrzebny pełny NetworkManager

netplan – system konfiguracji oparty na plikach YAML (domyślny w Ubuntu od 18.04)

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

iproute2 – zestaw nowoczesnych narzędzi do: konfiguracji interfejsów, adresacji, routingu, tuneli, reguł polityki routingu

Najważniejsze obiekty:

- link – interfejs warstwy łącza
- addr – adresy IP na interfejsach
- route – trasy w tablicy routingu
- neigh – sąsiedzi ARP/NDP
- rule – reguły routingu

366

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

- ip link show
 - wyświetla listę interfejsów sieciowych warstwy łącza danych; pokazuje nazwę interfejsu, jego stan administracyjny (UP / DOWN), stan nośnika oraz adres MAC
- ip addr show
 - wyświetla adresy przypisane do interfejsów; pokazuje konfigurację IPv4 i IPv6
- ip -br addr
 - pokazuje w jednym wierszu nazwę interfejsu, jego stan oraz przypisane adresy IP
- ip route show
 - wyświetla tablicę routingu systemu; pokazuje trasy bezpośrednie, trasę domyślną oraz interfejsy i bramy używane do wysyłania pakietów
- ip neigh show
 - wyświetla tablicę sąsiadów warstwy lokalnej; pokazuje skojarzenia IP z MAC

367

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

- `ip addr add 192.168.1.10/24 dev enp0s3`
– dodaje adres IPv4 192.168.1.10/24 do interfejsu enp0s3; od tego momentu host może komunikować się w tej podsieci z użyciem tego adresu
- `ip addr del 192.168.1.10/24 dev enp0s3`
– usuwa wskazany adres z interfejsu enp0s3; po usunięciu system przestaje używać tego adresu do komunikacji
- `ip link set enp0s3 up`
– aktywuje interfejs logicznie; system traktuje kartę jako włączoną i gotową do pracy
- `ip link set enp0s3 down`
– wyłącza interfejs logicznie; interfejs przestaje obsługiwać ruch, nawet jeśli fizycznie karta jest obecna

368

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

- `ip route show`
 - wyświetla aktualną tablicę routingu; pozwala sprawdzić, którędy system wysyła ruch do poszczególnych sieci i czy ma ustawioną trasę domyślną
- `ip route add default via 192.168.1.1`
 - dodaje trasę domyślną przez bramę 192.168.1.1; wszystkie pakiety do sieci, dla których nie ma bardziej szczegółowej trasy, będą wysyłane do tej bramy
- `ip route add 10.0.0.0/24 via 192.168.1.254`
 - dodaje statyczną trasę do sieci 10.0.0.0/24 przez router 192.168.1.254; system będzie kierował ruch do tej konkretnej sieci przez wskazaną bramę
- `ip route del default`
 - usuwa trasę domyślną; po jej usunięciu host zwykle traci możliwość komunikacji poza lokalnymi sieciami, chyba że istnieją inne odpowiednie trasy

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

Netplan – warstwa wysokiego poziomu: administrator zapisuje konfigurację w pliku YAML, a Netplan przekazuje ją do odpowiedniego renderu:
systemd-networkd, NetworkManager

Pliki zapisuje się w /etc/netplan/

netplan apply – zastosuj konfigurację – pozostaje po restarcie

Netplan ustawia:

- DHCP lub adres statyczny
- bramę
- DNS
- interfejsy Ethernet i Wi-Fi
- bridge, VLAN i inne elementy topologii

370

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

Przykładowy plik konfiguracyjny yaml:

```
network:
```

```
  version: 2
```

```
  renderer: networkd lub NetworkManager
```

```
  ethernets:
```

```
    enp0s3:
```

```
      dhcp4: true
```

```
sudo netplan apply
```

- stosuje konfigurację zapisaną w plikach Netplan

```
sudo netplan status --a
```

- pokazuje stan wszystkich interfejsów i konfigurację sieci

371

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

Każdy host IP podejmuje decyzję, przez który interfejs i do którego następnego węzła wysłać pakiet - tablica routingu:

- sieć docelowa lub pojedynczy host
- długość prefiksu
- interfejs wyjściowy
- opcjonalnie bramę pośrednią (*next hop*)
- metrykę trasy
- źródło trasy, np. konfiguracja statyczna, DHCP, jądro

372

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

Podstawowe rodzaje tras:

- Trasa bezpośrednia (connected route)
Powstaje automatycznie po przypisaniu adresu do interfejsu
Jeżeli interfejs ma adres 192.168.1.10/24, system wie, że sieć 192.168.1.0/24 jest osiągalna bezpośrednio przez ten interfejs
- Trasa statyczna
Dodana ręcznie przez administratora
Jest używana do kierowania ruchu do konkretnej sieci przez określoną bramę
- Trasa domyślna
Używana wtedy, gdy w tablicy nie ma bardziej szczegółowego wpisu dla adresu docelowego
W IPv4 oznaczana jako default albo 0.0.0.0/0

373

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

- `ip route show`
 - wyświetla bieżącą tablicę routingu IPv4; pokazuje trasę domyślną, trasy bezpośrednie, trasy statyczne, bramy i interfejsy wyjściowe
- `ip -4 route`
 - wyświetla wyłącznie trasy IPv4
- `ip -6 route`
 - wyświetla wyłącznie trasy IPv6
- `ip route get 8.8.8.8`
 - pokazuje, jaką decyzję routingu podejmie system dla konkretnego celu; wskazuje interfejs wyjściowy, bramę oraz często także adres źródłowy, który zostanie użyty

374

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

- ping 192.168.1.1
 - sprawdza osiągalność bramy lokalnej; pozwala potwierdzić, że działa komunikacja w obrębie lokalnej podsięci
- ping 8.8.8.8
 - sprawdza, czy host potrafi wyjść poza własną sieć lokalną; jeżeli to polecenie nie działa, problem zwykle dotyczy adresacji, bramy, routingu albo zapory
- traceroute 8.8.8.8
 - pokazuje kolejne routery po drodze do celu; wykorzystuje rosnący TTL, aby wykrywać kolejne przeskoki (*hop*)
- tracepath 8.8.8.8
 - podobne do traceroute, ale zwykle prostsze w użyciu; przy okazji może pokazać ograniczenie maksymalnego rozmiaru pakietu (MTU) na trasie

375

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

DNS (Domain Name System) – translacja nazw logicznych na adresy IP

Aplikacje zwykle odwołują się do nazw, np. google.com – system musi przekształcić je na rekordy typu A (IPv4) lub AAAA (IPv6) zanim rozpocznie komunikację sieciową

W Ubuntu z Netplan:

- DNS można dostać z DHCP
- ustawić ręcznie w pliku YAML
- bieżącą obsługę resolvera realizuje zwykle systemd-resolved.

Wniosek: DNS jest warstwą wyższą niż sama łączność IP

Jeżeli nie działa routing albo brama, to DNS też nie będzie działał poprawnie

376

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

Przykładowy plik konfiguracyjny yaml:

```
network:  
version: 2  
renderer: NetworkManager  
ethernets:  
  enp0s3:  
    dhcp4: true  
    nameservers:  
      addresses:  
        - 8.8.8.8  
        - 1.1.1.1
```

377

Architektura komputera – sieci – narzędzia konfiguracyjne Linux

- `resolvectl status`
 - pokazuje aktywną konfigurację resolvera: serwery DNS przypisane do interfejsów, domeny wyszukiwania i stan `systemd-resolved`
- `resolvectl query google.com`
 - wykonuje zapytanie DNS przez systemowy resolver i pokazuje wynik dla nazwy
- `cat /etc/resolv.conf`
 - pokazuje, jaką konfigurację resolvera widzą aplikacje
- `dig google.com`
 - pokazuje odpowiedź DNS bardziej szczegółowo: rekordy, serwer odpowiadający i parametry odpowiedzi

378

Architektura komputera – sieci – zaporą Linux

W Ubuntu filtrowanie ruchu realizuje **Netfilter** w jądrze Linux, podejmuje decyzje, czy pakiet ma zostać:

- zaakceptowany
- odrzucony
- zmodyfikowany
- przekazany dalej
- **Netfilter** – mechanizm w jądrze
- **nftables / nft** – nowoczesny natywny interfejs do definiowania reguł
- **iptables** – starszy interfejs do definiowania reguł
- **ufw** – uproszczony frontend, zalecany w typowych scenariuszach hostowych Ubuntu

379

Architektura komputera – sieci – zaporą Linux

Rodzaje zapór sieciowych

- **Stateless firewall (zapora bezstanowa):**

Analizuje **każdy pakiet niezależnie**, bez pamiętania wcześniejszej komunikacji
Decyzja o przepuszczeniu lub odrzuceniu pakietu jest podejmowana na podstawie pól nagłówka, takich jak: adres źródłowy i docelowy, numer portu, protokół, interfejs wejściowy lub wyjściowy

- **Stateful firewall (zapora stanowa)**

Analizuje pakiety **z uwzględnieniem stanu połączenia**, system pamięta, czy dany pakiet: rozpoczyna nowe połączenie, należy do połączenia już zestawionego, jest powiązany z istniejącą sesją, jest pakietem niepoprawnym.

380

Architektura komputera – sieci – zaporą Linux

ufw – podstawowe zarządzanie zaporą w Ubuntu – początkowo wyłączone,
przeznaczone do prostych zapór hostowych IPv4 i IPv6

Podstawowe polecenia

- `sudo ufw status`
– pokazuje, czy zaporą jest aktywna oraz podstawowe reguły
- `sudo ufw status verbose`
– pokazuje bardziej szczegółowy stan, w tym domyślne polityki
- `sudo ufw enable`
– włącza zaporę i ładuje zdefiniowane reguły
- `sudo ufw disable`
– wyłącza zaporę

381

Architektura komputera – sieci – zaporą Linux

ufw – reguły precyzyjne, profile aplikacji i logowanie

- `sudo ufw insert 1 allow 80`
– wstawia regułę na początek listy, co ma znaczenie przy kolejności przetwarzania reguł
- `sudo ufw status numbered`
– pokazuje reguły z numerami, co ułatwia zarządzanie kolejnością i usuwanie
- `sudo ufw delete deny 22`
– usuwa wskazaną regułę
- `sudo ufw allow proto tcp from 192.168.0.2 to any port 22`
– dopuszcza SSH tylko z wybranego hosta; analogicznie można dopuścić całą podsieć, np. 192.168.0.0/24

382

Architektura komputera – sieci – zapora Linux

ufw – reguły precyzyjne, profile aplikacji i logowanie

- `sudo ufw app list`
 - wyświetla listę dostępnych profili aplikacyjnych zdefiniowanych w systemie dla ufw; gotowe zestawy reguł opisujące typowe usługi, np. SSH, Apache czy Nginx
- `sudo ufw app info OpenSSH`
 - pokazuje szczegóły profilu aplikacyjnego OpenSSH; wyświetla nazwę profilu, jego opis oraz porty i protokoły, które zostaną otwarte po zastosowaniu tego profilu
- `sudo ufw allow OpenSSH`
 - dodaje regułę zapory na podstawie profilu OpenSSH; zamiast ręcznie otwierać port 22/tcp – nazwa profilu usługi

383

Architektura komputera – sieci – zapor Linux

ufw – reguły precyzyjne, profile aplikacji i logowanie

- `sudo ufw logging on`
 - włącza logowanie zdarzeń zapory; system zaczyna rejestrować wybrane informacje o ruchu filtrowanym przez ufw
- `sudo ufw --dry-run allow http`
 - pokazuje, jakie reguły zostałyby wygenerowane dla polecenia `allow http`, ale ich nie stosuje; pozwala sprawdzić efekt planowanej zmiany bez modyfikowania aktywnej konfiguracji zapory

384

Architektura komputera – sieci – zaporą Linux

nftables – nowoczesny model filtrowania w Linuxie – następca iptables

Zalety:

- wspólne reguły dla IPv4 i IPv6
- bardziej deklaratywna składnia
- większa elastyczność
- lepsza organizacja tabel i łańcuchów
- możliwość wydajniejszego przetwarzania reguł

385

Architektura komputera – sieci – zapora Linux

Prosta zapora hostowa: domyślnie blokuj ruch przychodzący, dopuść ruch lokalny loopback, dopuść połączenia już zestawione, otwórz port 22/tcp dla SSH

```
sudo nft add table inet filter
```

```
sudo nft add chain inet filter input '{ type filter hook input priority 0; policy drop; }'
```

```
sudo nft add rule inet filter input iif lo accept
```

```
sudo nft add rule inet filter input ct state established,related accept
```

```
sudo nft add rule inet filter input tcp dport 22 accept
```

386

Architektura komputera – sieci – zapora Linux

Prosta zapora hostowa: domyślnie blokuj ruch przychodzący, dopuść ruch lokalny loopback, dopuść połączenia już zestawione, otwórz port 22/tcp dla SSH

```
sudo iptables -P INPUT DROP
```

```
sudo iptables -A INPUT -i lo -j ACCEPT
```

```
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
```

```
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

387

Architektura komputera – sieci – zapora Linux

Po wpisaniu poleceń nft albo iptables reguły trafiają do:

jądra Linux – działają od razu, ale niekoniecznie są trwałe po restarcie systemu

Trwała konfiguracja – nftables

- /etc/nftables.conf

Trwała konfiguracja – iptables

- /etc/iptables/rules.v4
- /etc/iptables/rules.v6

Jeżeli używany jest ufw, to:

- /etc/ufw/applications.d

388

Architektura komputera – sieci – narzędzia konfiguracyjne Windows

W Windows 11 podstawowym środowiskiem administracyjnym do konfiguracji sieci jest **PowerShell**

Najważniejsze komendy:

- Get-NetAdapter
– wyświetla podstawowe właściwości kart sieciowych widocznych w systemie
- Get-NetIPConfiguration
– pokazuje konfigurację IP interfejsów, w tym adresy IP, bramę i DNS
- Get-NetIPAddress
– wyświetla adresy IPv4 i IPv6 oraz interfejsy, do których są przypisane
- Get-DnsClientServerAddress
– pokazuje adresy serwerów DNS przypisane do interfejsów

389

Architektura komputera – sieci – narzędzia konfiguracyjne Windows

Konfiguracja statyczna DNS

- Get-NetAdapter
 - pozwala ustalić nazwę i indeks interfejsu, który chcemy skonfigurować
- New-NetIPAddress
 - tworzy statyczny adres IPv4 lub IPv6 dla wskazanego interfejsu
- Set-DnsClientServerAddress ...
 - statycznie ustawia adresy serwerów DNS dla interfejsu

Przykład konfiguracji statycznej

Get-NetAdapter

New-NetIPAddress -InterfaceIndex 5 -IPAddress 192.168.1.50 -PrefixLength 24 -
DefaultGateway 192.168.1.1

Set-DnsClientServerAddress -InterfaceIndex 5 -ServerAddresses ("8.8.8.8","1.1.1.1")

Architektura komputera – sieci – narzędzia konfiguracyjne Windows

- Get-NetRoute
 - pokazuje tablicę routingu widoczną dla systemu
- route print
 - klasycznie wyświetla trasy IPv4 i IPv6
- Test-NetConnection 8.8.8.8
 - wykonuje diagnostykę osiągalności celu (ICMP, DNS, TCP, wybór trasy)
- tracert 8.8.8.8
 - pokazuje kolejne przeskoki między hostem a celem

391

Architektura komputera – sieci – narzędzia konfiguracyjne Windows

Konfiguracja statyczna DNS

- Get-NetAdapter
– pozwala ustalić nazwę i indeks interfejsu, który chcemy konfigurować
- New-NetIPAddress
– tworzy statyczny adres IPv4 lub IPv6 dla wskazanego interfejsu
- Set-DnsClientServerAddress ...
– statycznie ustawia adresy serwerów DNS dla interfejsu

Przykład konfiguracji statycznej

Get-NetAdapter

New-NetIPAddress -InterfaceIndex 5 -IPAddress 192.168.1.50 -PrefixLength 24 -DefaultGateway 192.168.1.1

Set-DnsClientServerAddress -InterfaceIndex 5 -ServerAddresses ("8.8.8.8","1.1.1.1")

392

Architektura komputera – sieci – zapora Windows

Windows Defender Firewall działa w oparciu o **trzy profile**:

1. **Domain** – system działa w domenie
 2. **Private** – sieć zaufana, np. domowa lub firmowa lokalna
 3. **Public** – sieć najmniej zaufana, np. hotspot, sieć hotelowa
- Get-NetFirewallProfile
– pokazuje ustawienia wybranego profilu zapory
 - Get-NetFirewallRule
– zwraca reguły zapory spełniające podane kryteria; same warunki, takie jak porty czy adresy, są reprezentowane przez osobne obiekty filtrów
 - Show-NetFirewallRule -PolicyStore ActiveStore
– pokazuje aktywne reguły w czytelniejszej, rozwiniętej postaci

393

Architektura komputera – sieci – zaporą Windows

Windows Defender Firewall – tworzenie prostych reguł

- New-NetFirewallRule ...
 - tworzy regułę przychodzącą dopuszczającą TCP na porcie lokalnym 22
- Get-NetFirewallRule -DisplayName "Allow SSH"
 - wyszukuje utworzoną regułę po nazwie widocznej

Przykład – otwarcie SSH

New-NetFirewallRule -DisplayName "Allow SSH" -Direction Inbound -Protocol TCP -
LocalPort 22 -Action Allow

Get-NetFirewallRule -DisplayName "Allow SSH"

394

Dziękuję za uwagę

Treści programowe:

Podstawowe zagrożenia bezpieczeństwa,
mechanizmy ochrony oraz tworzenie kopii
zapasowych i odzyskiwanie danych



396

Architektura komputera – zagrożenia bezpieczeństwa

System operacyjny – warstwa pośrednia między sprzętem, aplikacjami i użytkownikiem

Atak – daje atakującemu szeroki dostęp do zasobów

Cel bezpieczeństwa – ograniczenie prawdopodobieństwa naruszenia oraz skutków incydentu

System musi chronić dane użytkownika, własną konfigurację, mechanizmy startowe, konta, usługi i logi

Bezpieczeństwo obejmuje trzy stany pracy:

normalne działanie, sytuację incydentu oraz odzyskiwanie po awarii

Zabezpieczony system powinien: utrudniać przejęcie, ograniczać skutki błędu lub włamania, umożliwiać wykrycie naruszenia, pozwalać na szybkie odtworzenie działania

Architektura komputera – podstawowe atrybuty bezpieczeństwa

- **Poufność (confidentiality)** – dane są dostępne tylko dla podmiotów uprawnionych
- **Integralność (integrity)** – dane i konfiguracja nie zostały zmienione w sposób nieautoryzowany lub przypadkowy
- **Dostępność (availability)** – usługa lub zasób jest osiągalny wtedy, gdy jest potrzebny
- **Autentyczność (authenticity)** – pewność co do tożsamości użytkownika, systemu lub źródła danych
- **Rozliczalność (accountability)** – możliwość przypisania operacji do konkretnego podmiotu na podstawie logów i mechanizmów audytu

Naruszenie jednego z tych atrybutów wpływa także na pozostałe, np. ransomware uderza jednocześnie w integralność i dostępność

398

Architektura komputera – przedmiot ochrony

Dane użytkownika: dokumenty, bazy danych, poczta, konfiguracje aplikacji, klucze kryptograficzne

Dane systemowe: pliki startowe, pliki konfiguracyjne, biblioteki współdzielone, logi, rejestr systemowy

Tożsamości i poświadczenia: hasła, tokeny, bilety Kerberos, certyfikaty, sesje uwierzytelnione

Usługi systemowe i sieciowe: SSH, RDP, SMB, HTTP, DNS, serwery baz danych

Infrastruktura uruchomieniowa: BIOS/UEFI, bootloader, jądro, sterowniki

Zasoby sprzętowe: nośniki danych, interfejsy sieciowe, porty USB, moduł TPM

399

Architektura komputera – źródła zagrożeń

Czynnik ludzki: słabe hasła, uruchamianie nieznanych plików, błędna konfiguracja, przypadkowe usunięcie danych

Atakujący zewnętrzny: skanowanie portów, exploit usług sieciowych, phishing, przejęcie zdalnego dostępu

Atakujący wewnętrzny: nadużycie legalnych uprawnień, kopiowanie danych, sabotaż

Złośliwe oprogramowanie: trojany, ransomware, rootkity, spyware

Awaria sprzętu: uszkodzenie dysku, błędy pamięci RAM, problemy z zasilaniem

Błędy oprogramowania: podatności w systemie, bibliotekach, sterownikach i aplikacjach

Błędy proceduralne: brak kopii zapasowych, brak testów odtwarzania, brak segmentacji sieci

400

Architektura komputera – powierzchnia ataku

Powierzchnia ataku – zbiór wszystkich elementów systemu, przez które możliwe jest jego rozpoznanie, zakłócenie działania lub przejęcie kontroli

Obejmuje: usługi sieciowe, lokalne mechanizmy systemowe

Każda aktywna usługa, otwarty port, uruchomiony demon, interfejs administracyjny, harmonogram zadań czy komponent zewnętrzny zwiększa liczbę potencjalnych punktów wejścia dla atakującego

Wraz ze wzrostem liczby usług rośnie ryzyko: błędnej konfiguracji, pozostawienia domyślnych ustawień, niezafatanej podatności, niepotrzebnie szerokich uprawnień, trudności w monitorowaniu i audycie

W praktyce atakujący nie musi przełamać całego systemu – wystarczy mu jeden słabo zabezpieczony komponent, np. nieużywana usługa z otwartym portem, stary interfejs WWW lub źle skonfigurowany mechanizm zdalnego dostępu

401

Architektura komputera – powierzchnia ataku

Hardening – utwardzanie systemu: zestaw działań konfiguracyjnych i administracyjnych, których celem jest zwiększenie bezpieczeństwa systemu operacyjnego, aplikacji lub urządzenia przez ograniczenie liczby możliwych podatności i punktów ataku

Podstawowa zasada hardeningu = minimalizacja powierzchni ataku:

wyłączanie zbędnych usług, usuwanie niepotrzebnych pakietów i modułów, zamykanie nieużywanych portów, ograniczanie interfejsów zdalnych, redukcja uprawnień do niezbędnego minimum

Zasada minimalizmu bezpieczeństwa: system powinien udostępniać tylko te funkcje, które są rzeczywiście wymagane do jego działania

Mniej aktywnych elementów = mniej podatności do wykorzystania, prostsza administracja, łatwiejszy audyt, mniejsze ryzyko błędu konfiguracyjnego

402

Architektura komputera – scenariusze naruszeń

Użytkownik otwiera złośliwy załącznik i uruchamia kod z uprawnieniami swojej sesji

Atakujący wykorzystuje niezafataną podatność w usłudze sieciowej i uzyskuje zdalne wykonanie kodu

Przejęte konto użytkownika jest używane do dalszej eskalacji uprawnień

Malware szyfruje dane lokalne, zasoby sieciowe i kopie dostępne online

Niepoprawna konfiguracja praw dostępu powoduje ujawnienie danych lub ich skasowanie

Awaria nośnika bez działającego backupu prowadzi do trwałej utraty danych

Kradzież laptopa bez szyfrowania dysku daje fizyczny dostęp do plików

403

Architektura komputera – zagrożenie, podatność, atak, ryzyko

Zagrożenie – potencjalne źródło szkody, np. malware, błąd użytkownika, pożar, awaria

Podatność – konkretna słabość, np. brak aktualizacji, zbyt szerokie uprawnienia, domyślne hasło

Atak – praktyczne wykorzystanie podatności

Ryzyko – opisuje oczekiwaną szkodę i zależy od:

- prawdopodobieństwa wystąpienia
- skali skutków

Ten sam system może mieć tę samą podatność, ale inne ryzyko zależnie od środowiska i wartości chronionych danych

Zarządzanie bezpieczeństwem = redukcja ryzyka

404

Architektura komputera – klasy podatności

Błędy pamięci: przepełnienie bufora, użycie pamięci po jej zwolnieniu , przepełnienie zakresu liczby

Błędy walidacji danych wejściowych: niepoprawne parsowanie danych, wstrzyknięcie polecenia, przejście poza dozwoloną ścieżkę katalogów

Błędy logiki bezpieczeństwa: brak kontroli autoryzacji, błędy w sprawdzaniu ról

Błędy konfiguracji: usługa wystawiona publicznie, zbyt szerokie ACL, konto bez hasła

Podatności łańcucha dostaw: złośliwe lub podatne biblioteki i pakiety

Błędy kryptograficzne: słabe algorytmy, złe przechowywanie kluczy, brak weryfikacji certyfikatu

405

Architektura komputera – klasy złośliwego oprogramowania

Wirus – dołącza się do innego pliku i wymaga uruchomienia nośnika lub programu gospodarza

Robak – propaguje się samodzielnie przez sieć bez udziału użytkownika

Trojan – wygląda jak legalna aplikacja, ale realizuje ukrytą funkcję złośliwą

Spyware – zbiera informacje: hasła, historię, zrzuty ekranu, dane formularzy

Rootkit – ukrywa obecność atakującego i modyfikuje mechanizmy wykrywania

Ransomware – szyfruje dane i wymusza okup

Bot – pozwala na zdalne sterowanie zainfekowaną maszyną i włączenie jej do botnetu

406

Architektura komputera – zasady projektowania bezpieczeństwa

Minimalne uprawnienia – użytkownik, proces i usługa powinny działać wyłącznie z zakresem praw koniecznym do wykonania zadania

Domyślny zakaz – dostęp, ruch sieciowy lub operacja są blokowane, jeżeli nie istnieje jawna reguła zezwalająca

Ochrona warstwowa – zabezpieczenia powinny działać na wielu poziomach, np. uwierzytelnianie, kontrola dostępu, zaporę sieciową, izolacja procesu, logowanie i szyfrowanie

Rozdział ról i obowiązków – funkcje administracyjne, operacyjne i audytowe powinny być przypisane różnym kontom lub osobom

407

Architektura komputera – zasady projektowania bezpieczeństwa

Minimalizacja systemu – należy ograniczać liczbę usług, pakietów, kont, otwartych portów, reguł i aktywnych komponentów

Bezpieczne zachowanie przy błędzie – w przypadku awarii, braku odpowiedzi komponentu, błędu autoryzacji lub niespójności system powinien odmawiać dostępu, a nie go przyznawać

Uwierzytelnianie – potwierdzenie tożsamości użytkownika lub procesu

Autoryzacja – decyzja, jakie operacje wolno wykonać po poprawnym uwierzytelnieniu

Audyt – rejestracja, kto, kiedy i co wykonał

Poprawne logowanie nie oznacza jeszcze prawa do wszystkich zasobów

Bez audytu nie ma wiarygodnej analizy incydentu ani rozliczalności działań

408

Architektura komputera – hasła i poświadczenia

Hasło – najprostsza forma potwierdzenia tożsamości użytkownika – powinno być długie, unikalne i nieużywane w wielu systemach

Konto uprzywilejowane – większy zakres dostępu niż konto zwykłe – przejęcie daje napastnikowi znacznie większe możliwości – silniejsza ochrona

Dane uwierzytelniające – wszystkie informacje lub obiekty, które system wykorzystuje do potwierdzenia tożsamości użytkownika, procesu albo usługi

Do danych uwierzytelniających należą (oprócz haseł):

klucze prywatne – tajne klucze kryptograficzne używane np. do logowania lub podpisu

tokeny sesyjne – identyfikatory potwierdzające, wcześniejsze poprawne zalogowanie

bilety uwierzytelniające – system uwierzytelniania potwierdzający tożsamość bez hasła

certyfikaty klienta – cyfrowe dokumenty tożsamości

409

Architektura komputera – funkcja skrótu

Funkcja skrótu odwzorowuje dane na skrót o ustalonej długości – służy do wykrywania modyfikacji danych

Funkcja skrótu nie potwierdza tożsamości nadawcy – każdy może obliczyć skrót tych samych danych

Uwierzytelnianie wiadomości z tajnym kluczem łączy dane i sekret współdzielony przez obie strony – zapewnia integralność i autentyczność w modelu wspólnego klucza

Podpis cyfrowy wykorzystuje klucz prywatny nadawcy i klucz publiczny weryfikujący – zapewnia integralność, autentyczność i możliwość przypisania podpisu do właściciela klucza – nie jest równoważny zwykłemu szyfrowaniu treści

System operacyjny wykorzystuje je do: weryfikacji pakietów, podpisywania sterowników, sprawdzania aktualizacji, uwierzytelniania komunikacji między komponentami

410

Architektura komputera – bezpieczne przechowywanie haseł

Hasło nie jest przechowywane jawnie, tylko w postaci skrótu kryptograficznego

Dla każdego hasła generuje się losową sól, aby dwa identyczne hasła nie dawały tego samego wyniku

Do haseł stosuje się funkcje celowo wolne i kosztowne pamięciowo, np. Argon2, bcrypt, scrypt

Po przejęciu bazy napastnik wykonuje atak offline, czyli lokalnie zgaduje hasła i porównuje otrzymane skróty

Sól uniemożliwia skuteczne użycie gotowych tablic skrótów i utrudnia masowe łamanie wielu haseł naraz

Parametry funkcji powinny być dobrane tak, aby pojedyncza próba była kosztowna obliczeniowo

411

Architektura komputera – uwierzytelnianie wieloskładnikowe

Uwierzytelnianie wieloskładnikowe wymaga przedstawienia co najmniej dwóch niezależnych składników z różnych klas:

- wiedza – hasło, kod PIN
- posiadanie – token sprzętowy, aplikacja generująca kod jednorazowy, karta
- cecha biologiczna – odcisk palca, obraz twarzy

Przejęcie jednego sekretu nie wystarcza do uzyskania dostępu

Mechanizm ogranicza skuteczność: przejęcia hasła, ataku słownikowego na pojedynczy składnik, ponownego użycia wykradzionych danych logowania, ataków phishingowych

Najsilniejszy wariant – klucze kryptograficzne związane z urządzeniem i domeną usługi

Uwierzytelnianie wieloskładnikowe dla: kont uprzywilejowanych, dostępu zdalnego, systemów przechowujących dane wrażliwe, interfejsów administracyjnych

412

Architektura komputera – modele kontroli dostępu

Model uznaniowy – właściciel obiektu definiuje prawa dostępu do zasobu; decyzja zależy od właściciela i metadanych obiektu

Model obowiązkowy – reguły dostępu wynikają z centralnej polityki bezpieczeństwa; użytkownik ani właściciel obiektu nie mogą ich dowolnie osłabić

Model oparty na rolach – prawa są przypisywane do ról, a użytkownik uzyskuje uprawnienia przez przypisanie do roli

Model oparty na atrybutach – decyzja dostępu wynika z oceny atrybutów podmiotu, obiektu, operacji i kontekstu

System może jednocześnie stosować kilka modeli, np. prawa plików, listy kontroli dostępu i obowiązkową politykę bezpieczeństwa – większa złożoność = większe znaczenie praw efektywnych, a nie pojedynczych wpisów konfiguracyjnych

413

Architektura komputera – prawa dostępu w systemach plików

System plików przechowuje metadane określające właściciela, grupę, typ obiektu i prawa dostępu

Decyzje dostępu dla pliku: odczyt zawartości, modyfikacja zawartości, wykonanie jako programu

Decyzje dostępu dla katalogu: odczyt listy wpisów, przechodzenie ścieżki, tworzenie i usuwania wpisów, zmiana nazw obiektów

Uprawnienia do pliku i uprawnienia do katalogu nadrzędnego są rozpatrywane niezależnie: możliwa jest sytuacja, w której użytkownik ma prawo zapisu do pliku, ale nie może go usunąć z powodu braku praw do katalogu

Błędna konfiguracja praw katalogów: ujawnienie nazw i struktury danych, nieautoryzowane usuwanie lub podmiana plików, blokowanie dostępu procesom

414

Architektura komputera – rozszerzone listy kontroli dostępu

Rozszerzone listy kontroli dostępu – pozwalają przypisywać prawa konkretnym użytkownikom i grupom poza prostym modelem właściciel-grupa-pozostali

Wpis listy określa: podmiot, zestaw operacji dozwolonych lub zabronionych, reguły dziedziczenia – jeśli system je obsługuje

Decyzja dostępu wynika z przetworzenia całej listy, a nie z pojedynczego wpisu

Przy analizie bezpieczeństwa należy wyznaczać prawa efektywne po uwzględnieniu: kolejności reguł, masek, dziedziczenia, reguł odmowy

Rozbudowana lista kontroli dostępu zwiększa precyzję polityki, ale utrudnia audyt i wykrywanie nadmiarowych praw

Błąd w jednej regule może rozszerzyć dostęp do całego drzewa katalogów

415

Architektura komputera – konta uprzywilejowane

Konto uprzywilejowane może wykonywać operacje wpływające na cały system: modyfikację konfiguracji, instalację i usuwanie oprogramowania, zarządzanie usługami, odczyt danych innych użytkowników, zmianę polityki bezpieczeństwa, modyfikację logów i mechanizmów startowych

Konto uprzywilejowane musi być odseparowane od konta używanego do pracy codziennej

Dostęp do takiego konta wymaga: silnego uwierzytelniania, rejestracji operacji, ograniczenia czasu użycia, kontroli źródła logowania

Procesy usługowe nie powinny działać na koncie uprzywilejowanym bez wymuszonej potrzeby architektonicznej

Przejęcie konta uprzywilejowanego umożliwia pełną zmianę stanu hosta i unieważnia większość lokalnych mechanizmów ochrony – minimalna liczba kont uprzywilejowanych

416

Architektura komputera – izolacja procesów

Każdy proces posiada: własną przestrzeń adresową, własne deskryptory plików, własny kontekst wykonania, identyfikatory bezpieczeństwa, własne mapowania pamięci

Proces w trybie użytkownika nie ma bezpośredniego dostępu do pamięci innego procesu ani pamięci jądra

Operacje uprzywilejowane są wykonywane wyłącznie przez jądro po przejściu przez interfejs wywołań systemowych

Izolacja obejmuje także separację: kontekstu użytkownika i grup, przestrzeni nazw zasobów, kanałów komunikacji międzyprocesowej

Naruszenie izolacji umożliwia: odczyt cudzych danych, modyfikację stanu innych procesów, eskalację uprawnień, obejście modelu bezpieczeństwa systemu

Izolacja procesów = podstawowy mechanizmem ochrony systemów wielozadaniowych

417

Architektura komputera – ograniczanie możliwości procesów

System może ograniczać sam proces przez politykę wykonania i izolację zasobów

Polityka procesu może definiować: do jakich plików i katalogów ma dostęp, jakie urządzenia może otwierać, z jakimi hostami i portami może się komunikować, jakich wywołań systemowych może używać, jakie mechanizmy komunikacji międzyprocesowej są dostępne

Ograniczenie procesu zmniejsza skutki kompromitacji aplikacji przetwarzającej dane niezaufane

Mechanizm ten ma znaczenie dla: przeglądarek, interpreterów języków skryptowych, usług sieciowych, parserów dokumentów i multimediów, kontenerów

Kompromitacja procesu ograniczonego polityką nie daje automatycznie pełnego dostępu do hosta

418

Architektura komputera – ochrona pamięci

Jednostka zarządzania pamięcią – realizuje translację adresów wirtualnych na fizyczne oraz kontroluje prawa dostępu do stron pamięci

Każda strona może być oznaczona atrybutami: odczyt, zapis, wykonanie, dostęp tylko z trybu uprzywilejowanego

Dostęp niezgodny z atrybutami strony powoduje wyjątek sprzętowy obsługiwany przez jądro

Mechanizm ten umożliwia: separację procesów, separację przestrzeni użytkownika i jądra, ochronę kodu przed nadpisaniem, blokowanie wykonywania kodu z obszarów danych

Ochrona pamięci zależy od poprawnej konfiguracji tablic stron i praw segmentów mapowanych do procesu

Wyłączenie lub obejście ochrony pamięci = destabilizacja systemu i eskalacja uprawnień

419

Architektura komputera – ochrona pamięci

Mechanizmy utrudniające wykorzystanie błędów pamięci:

- Losowe rozmieszczenie przestrzeni adresowej zmienia położenie: stosu, sterty, bibliotek współdzielonych, obrazu programu
- Zakaz wykonywania danych uniemożliwia uruchamianie kodu z obszarów przeznaczonych na bufor danych lub stertę
- Ochrona stosu wprowadza wartość kontrolną wykrywaną przed powrotem z funkcji
- Kontrola przepływu wykonania ogranicza możliwość przekierowania sterowania do nieautoryzowanych adresów

Redukują skuteczność exploitów wykorzystujących: nadpisanie adresu powrotu, przejęcie wskaźników funkcji, wykonanie kodu wstrzykniętego do pamięci

Nie usuwają błędów implementacyjnych – zwiększają koszt i złożoność ich wykorzystania

420

Architektura komputera – moduły bezpieczeństwa systemu

Moduły bezpieczeństwa – szerszy model kontroli dostępu poza klasyczne prawa plików i identyfikatory użytkowników

Polityka może opisywać dozwolone relacje między: procesem, plikiem, katalogiem, gniazdem sieciowym, urządzeniem, usługą systemową

Decyzja dostępu zależy od etykiety bezpieczeństwa, kontekstu procesu lub profilu przypisanego do usługi

Proces uruchomiony z wysokimi uprawnieniami może mieć zablokowane: połączenia sieciowe, zapis do wybranych katalogów, dostęp do urządzeń, wykonanie określonych operacji jądra

Moduły bezpieczeństwa zmniejszają skutki przejęcia usługi i ograniczają ruch boczny wewnątrz hosta

421

Architektura komputera – integralność oprogramowania

Integralność oprogramowania – pewność: pakiet pochodzi od zaufanego dostawcy, nie został zmodyfikowany, jego zawartość odpowiada oczekiwanej wersji

Weryfikacja integralności: podpisy kryptograficzne pakietów, weryfikacja repozytorium, kontrola skrótów plików binarnych i bibliotek, weryfikacja sterowników i komponentów uruchamianych z podwyższonymi uprawnieniami

Kompromitacja jednego pakietu może dostarczyć napastnikowi kod wykonywany z prawami instalatora, usługi albo jądra

Brak kontroli integralności aktualizacji umożliwia: podstawienie fałszywego pakietu, modyfikację legalnego pakietu, atak na łańcuch dostaw oprogramowania

Zaufane źródło i poprawny podpis są warunkiem instalacji komponentów systemowych

422

Architektura komputera – rozruch i łańcuch zaufania

Łańcuch zaufania – rozpoczyna się w oprogramowaniu wbudowanym i obejmuje kolejne etapy rozruchu aż do uruchomienia jądra i składników startowych systemu

Każdy etap musi weryfikować integralność i autentyczność kolejnego etapu przed przekazaniem sterowania

Sekwencja obejmuje: firmware, program rozruchowy, jądro, moduły i komponenty startowe

Przejęcie programu rozruchowego umożliwia: uruchomienie nieautoryzowanego jądra, ukrycie złośliwego kodu przed systemem, przejęcie kontroli przed aktywacją standardowych zabezpieczeń hosta

Ochrona rozruchu wymaga weryfikacji podpisów oraz kontroli nośników i ustawień firmware

423

Architektura komputera – szyfrowanie danych na nośniku

Szyfrowanie nośnika – ochrona danych przed odczytem po fizycznym przejęciu dysku, woluminu lub urządzenia

Szyfrowanie może obejmować: cały dysk, wybrany wolumin, katalog, pojedyncze pliki

Bezpieczeństwo mechanizmu zależy od: siły algorytmu, ochrony klucza szyfrowania, poprawnej procedury odblokowania, integralności środowiska rozruchowego

Szyfrowanie nie chroni danych przed procesem działającym po poprawnym zalogowaniu i odszyfrowaniu woluminu

Chroni przed: odczytem po wyjęciu dysku, analizą nośnika offline, prostą kradzieżą danych z utraconego urządzenia

424

Architektura komputera – szyfrowanie transmisji

Kanał komunikacyjny musi zapewniać: poufność danych, integralność danych, uwierzytelnienie strony zdalnej

Samo szyfrowanie bez weryfikacji tożsamości nie chroni przed atakiem pośrednika

Poprawna konfiguracja obejmuje: weryfikację certyfikatu lub klucza publicznego, użycie aktualnej wersji protokołu, wyłączenie algorytmów uznanych za słabe, ochronę kluczy prywatnych, poprawne zarządzanie zaufanymi urzędami certyfikacji

Błędna konfiguracja może prowadzić do: akceptacji fałszywego certyfikatu, użycia słabych szyfrów, obniżenia poziomu bezpieczeństwa sesji

Bezpieczny kanał transmisyjny jest wymagany dla administracji, logowania i przesyłania danych wrażliwych

425

Architektura komputera – rejestrowanie zdarzeń

System musi rejestrować zdarzenia istotne dla bezpieczeństwa i administracji: logowania udane i nieudane, zmiany uprawnień, uruchomienia i zatrzymania usług, modyfikacje konfiguracji, operacje uprzywilejowane, błędy jądra i sterowników, zdarzenia zapory i mechanizmów ochronnych

Log musi zawierać: źródło zdarzenia, znacznik czasu, identyfikator użytkownika lub procesu, typ operacji, wynik operacji

Logi muszą być chronione przed modyfikacją, usunięciem i nieautoryzowanym odczytem

Rejestrowanie musi być dobrane tak, aby nie utracić zdarzeń krytycznych i nie przeciążyć systemu szumem

Brak logów uniemożliwia analizę incydentu i odtworzenie sekwencji działań napastnika

426

Architektura komputera – rejestrowanie zdarzeń Linux

systemd-journald – podstawowy mechanizm rejestrowania zdarzeń, komunikaty: jądra, usług systemowych, procesów użytkownika, standardowego wyjścia i błędów usług

Logi mogą być przechowywane:

`/run/log/journal` – ulotnie; `/var/log/journal` – trwale

Równolegle może działać `rsyslog` – przepisuje część zdarzeń do `/var/log`

Typowe pliki logów w Ubuntu:

`/var/log/syslog` – ogólne zdarzenia systemowe

`/var/log/auth.log` – logowania, sudo, SSH, uwierzytelnianie

`/var/log/kern.log` – komunikaty jądra

`/var/log/dpkg.log` – instalacja i usuwanie pakietów

`/var/log/apt/history.log` – historia operacji pakietowych

427

Architektura komputera – rejestrowanie zdarzeń Linux

`journalctl` – podstawowym narzędziem analizy logów

Typowe użycie:

`journalctl -b` – logi z bieżącego uruchomienia

`journalctl -k` – komunikaty jądra

`journalctl -u ssh` – zdarzenia konkretnej usługi

`journalctl -p err..alert` – wpisy o wysokim poziomie ważności

`journalctl -f` – śledzenie logów w czasie rzeczywistym

Analiza logów: `less`, `grep`, `tail -f`, `awk`

`logrotate` – rotacja logów tekstowych: archiwizacja starych plików, kompresja, usuwanie po przekroczeniu polityki retencji

Dostęp do logów musi być ograniczony – zawierają: nazwy kont, adresy IP, ścieżki systemowe, błędy usług, fragmenty danych wejściowych lub komunikatów aplikacji

428

Architektura komputera – rejestrowanie zdarzeń Windows

Windows Event Log – mechanizmem rejestrowania zdarzeń

zapisywane w plikach EVTX przechowywane w: C:\Windows\System32\winevt\Logs\

System dzieli logi na kanały zdarzeń, najważniejsze:

Application – zdarzenia aplikacji użytkownika i usług

System – zdarzenia sterowników, usług i składników systemu

Security – logowania, użycie uprawnień, audyt bezpieczeństwa

Setup – instalacja i aktualizacja składników systemu

Applications and Services Logs – szczegółowe dla konkretnych komponentów

Wpis zawiera: źródło zdarzenia, identyfikator zdarzenia, poziom ważności, użytkownika, host, czas wystąpienia, opis operacji

429

Architektura komputera – rejestrowanie zdarzeń Windows

Podgląd zdarzeń (eventvwr.msc) – graficzne narzędzie analizy

wevtutil, Get-WinEvent, Get-EventLog – narzędzia wiersza poleceń

Typowe polecenia:

Get-WinEvent -LogName System – usługi, sterowniki, błędy systemu

Get-WinEvent -LogName Security -MaxEvents 50 – ostatnie zdarzenia bezpieczeństwa: logowania, audyt, uprawnienia

wevtutil qe Security /c:20 /f:text – 20 wpisów z logu bezpieczeństwa w formie tekstowej

Analiza zdarzeń w Windows opiera się na: identyfikatorze zdarzenia, źródle, poziomie, kontekście użytkownika, korelacji czasowej między kanałami

430

Architektura komputera – audyt i monitoring bezpieczeństwa

Audyt – okresowa analiza konfiguracji, uprawnień, usług i zgodności systemu z polityką bezpieczeństwa

Audyt wykrywa: nadmiarowe prawa, nieużywane konta, błędne reguły zapory, nieautoryzowane zmiany konfiguracji, odchylenia od wzorca systemu

Monitoring – ciągła obserwacja zdarzeń, anomalii i wskaźników naruszenia bezpieczeństwa

Monitoring wykrywa: nietypowe logowania, nieznane procesy, podejrzany ruch sieciowy, próby eskalacji uprawnień, modyfikację autostartu i usług

Audyt bada stan statyczny, monitoring bada zachowanie dynamiczne

431

Architektura komputera – audyt i monitoring bezpieczeństwa

Audyt i monitoring w Linux/Ubuntu:

journalctl – analiza logów usług, jądra i systemu

ps, top, htop – obserwacja procesów

ss – analiza aktywnych połączeń i portów

systemctl – kontrola usług i ich stanu

htop / top – obserwacja procesów

dmesg – komunikaty jądra i urządzeń

last – historia logowań

432

Architektura komputera – audyt i monitoring bezpieczeństwa

Audyt i monitoring w Windows 11:

Get-WinEvent, wevtutil – logi systemowe i bezpieczeństwa

tasklist – procesy i powiązane usługi

netstat – połączenia i porty nasłuchujące

sc – stan usług i sterowników

schtasks – zadania automatyczne i autostart zadaniowy

eventvwr.msc – graficzna analiza logów

433

Architektura komputera – zaporą

Zapora hostowa – filtruje ruch na poziomie pojedynczego systemu operacyjnego przed przekazaniem go do aplikacji lub po opuszczeniu hosta

Reguła zapory może uwzględniać: adres źródłowy i docelowy, port źródłowy i docelowy, protokół, interfejs, kierunek ruchu, stan połączenia, identyfikator usługi lub procesu

Polityka bezpieczeństwa hosta powinna: blokować ruch przychodzący domyślnie, dopuszczać tylko wymagane usługi, ograniczać ruch administracyjny do zaufanych źródeł

Linux/Ubuntu: nft list ruleset – pełny aktywny zestaw reguł filtrowania pakietów;
sudo ufw status verbose – polityka i reguły, gdy host używa UFW

Windows 11: Get-NetFirewallProfile – profile zapory i ich ustawienia;
Get-NetFirewallRule – reguły zapory;
netsh advfirewall show allprofiles – konfiguracja wszystkich profili zapory

434

Architektura komputera – filtracja stanowa

Filtracja stanowa – śledzi stan komunikacji i klasyfikuje pakiety jako: inicjujące nowe połączenie, należące do połączenia już ustanowionego, powiązane z istniejącą sesją

Odpowiedzi na legalne połączenia mogą zostać dopuszczone bez otwierania dodatkowych reguł wejściowych

Polityka oparta na stanie: blokuje nieoczekiwane nowe połączenia przychodzące, dopuszcza pakiety związane z istniejącą sesją, jawnie otwiera wyłącznie wymagane usługi

Filtracja zmniejsza liczbę reguł, poprawia zgodność polityki z działaniem protokołów

Linux/Ubuntu: ss -tan – aktywne połączenia TCP i ich stany;
nft list ruleset – reguły zapory współpracujące z tablicą stanu

Windows 11: netstat -ano – aktywne połączenia, porty i PID;
Get-NetFirewallProfile – domyślne akcje profili zapory

435

Architektura komputera – ochrona punktu końcowego

Ochrona punktu końcowego analizuje zachowanie hosta na poziomie: plików, procesów, pamięci, sieci, rejestru lub konfiguracji systemowej, mechanizmów startowych

Wykrywane: uruchamianie nieznanych procesów, modyfikacje pamięci wykonywalnej, uruchamianie skryptów, nietypowe połączenia sieciowe, próby eskalacji uprawnień, trwałe mechanizmy autostartu

Reakcja: zatrzymanie procesu, izolację hosta, zablokowanie konta, zebranie artefaktów do analizy

Linux/Ubuntu: aa-status – stan profili AppArmor;
journalctl -k – komunikaty jądra, naruszenia polityk i zdarzenia ochronne

Windows 11: Get-MpComputerStatus – stan Microsoft Defender;
Get-WinEvent -LogName Security -MaxEvents 50 – ostatnie zdarzenia bezpieczeństwa

436

Architektura komputera – aktualizacja bezpieczeństwa

Aktualizacja bezpieczeństwa – usuwa znaną podatność w systemie, bibliotece, aplikacji, sterowniku albo firmware

Publiczne ujawnienie błędu – pojawienie się działającego exploitu

Brak aktualizacji wydłuża okno podatności między publikacją luki a jej usunięciem w systemie docelowym

Zarządzanie aktualizacjami obejmuje: klasyfikację krytyczności, testy zgodności, plan wdrożenia, kontrolę wersji, możliwość wycofania zmiany

Linux/Ubuntu: `sudo apt update` – odświeżenie list pakietów;
`apt list --upgradable` – pakiety z dostępną aktualizacją;
`sudo apt upgrade` – instalacja aktualizacji

Windows 11: Get-HotFix – lista zainstalowanych poprawek;
Get-WinEvent -LogName System – przegląd zdarzeń systemowych po wdrożeniu aktualizacji

437

Architektura komputera – kopia zapasowa

Kopia zapasowa – mechanizm odzyskiwania: awaria sprzętu, błąd logiczny, usunięcie danych, uszkodzenie systemu plików, atak szyfrujący

Kopia zapasowa nie jest równoważna: nadmiarowości macierzy dyskowej, synchronizacji danych, migawce przechowywanej w tym samym środowisku administracyjnym

Nadmiarowość – zwiększa dostępność – nie przywraca wcześniejszego poprawnego stanu logicznego danych

Synchronizacja – może powielić uszkodzenie, nadpisanie lub zaszyfrowanie plików

Kopia zapasowa – odseparowana logicznie lub fizycznie od środowiska produkcyjnego

Linux/Ubuntu: `tar -cpf backup.tar /etc /home/user` – kopia plikowa do archiwum

Windows 11: `Compress-Archive -Path C:\Dane -DestinationPath C:\Backup\dane.zip` – kopia plikowa do archiwum

438

Architektura komputera – kopia zapasowa

Punkt odtworzenia – określa maksymalny dopuszczalny ubytek danych wyrażony jako różnica czasu między ostatnim poprawnym stanem a stanem po awarii

Czas odtworzenia – określa maksymalny czas niedostępności usługi od momentu incydentu do przywrócenia działania

Parametry te określają: częstotliwość wykonywania kopii, liczbę przechowywanych wersji, wybór nośnika i repozytorium, stopień automatyzacji procedury odtworzeniowej, architekturę redundancji i przełączania awaryjnego

Mniejszy dopuszczalny ubytek danych = częstszy harmonogram kopii zapasowej

Krótszy dopuszczalny czas niedostępności = większe wymagania wobec automatyzacji i gotowości środowiska zapasowego

Parametry odtwarzania muszą być zdefiniowane przed wyborem technologii

439

Architektura komputera – kopia zapasowa – rodzaje

- Kopia pełna – zawiera cały zbiór danych; upraszcza odtwarzanie kosztem czasu wykonania i pojemności
- Kopia przyrostowa – zawiera zmiany od poprzedniej kopii; minimalizuje objętość danych kosztem bardziej złożonej procedury odtwarzania
- Kopia różnicowa – zawiera zmiany od ostatniej kopii pełnej; stanowi kompromis między czasem wykonania a czasem odtworzenia

Rozróżnienie: spójność systemu plików, spójność aplikacyjna, spójność transakcyjna

Linux/Ubuntu: `tar -cpf backup.tar katalog/` – kopia pełna na poziomie plików
`sha256sum backup.tar` – obliczenie skrótu archiwum do kontroli integralności

Windows 11: `Compress-Archive -Path C:\Dane -DestinationPath C:\Backup\dane.zip` – kopia pełna na poziomie plików; `Get-FileHash C:\Backup\dane.zip` – obliczenie skrótu pliku kopii

440

Architektura komputera – kopia zapasowa – odtwarzanie i odzyskiwanie

Kopia zapasowa bez testu odtworzenia nie stanowi potwierdzonego mechanizmu odzyskiwania

Testy obejmują: odtworzenie pojedynczego pliku, odtworzenie katalogu, odtworzenie konfiguracji, odtworzenie całej usługi, pomiar rzeczywistego czasu powrotu do pracy

Po incydencie nie wolno przywracać systemu przed: identyfikacją przyczyny, izolacją zagrożenia, usunięciem mechanizmu kompromitacji

Proces odzyskiwania obejmuje: analizę incydentu, eliminację źródła problemu, odtworzenie danych i usług, weryfikację integralności, ponowne włączenie systemu do pracy

441

Architektura komputera – kopia zapasowa – odtwarzanie i odzyskiwanie

Test odtworzenia weryfikuje nie tylko nośnik kopii, ale cały proces administracyjny

Skuteczność kopii zapasowej mierzy się zdolnością do odtworzenia poprawnego stanu w wymaganym czasie, a nie samym faktem wykonania kopii

Linux/Ubuntu: `tar -xpf backup.tar -C /tmp/restore-test` – test odtworzenia archiwum
`sha256sum backup.tar` – kontrola integralności pliku kopii

Windows 11: `Expand-Archive -Path C:\Backup\dane.zip -DestinationPath C:\RestoreTest`
– test odtworzenia archiwum

`Get-FileHash C:\Backup\dane.zip` – kontrola integralności pliku kopii

Dziękuję za uwagę