Systemy operacyjne

February 13, 2025

Contents

1 Funkcje systemów operacyjnych

1.1 Przykładowe funkcjl jakie pełni system operacyjny

- Planowanie, wykonywanie i nadzorowanie zadań
- Obsługa i kontrolowanie urządzeń
- Zarządzanie pamięcią masową
- Tworzenie mechanizmu umożliwiającego przechowywanie informacji (np. system plików)
- Stworzenie środowiska, dzięki któremu użytkownik może w sposób wygodny i wydajny zarządzać zadaniami

1.2 Zastosowania rodzajów systemów operacyjnych

Wieloprocesorowe systemy operacyjne - mają możliwość wykonywania wielu procesów, wykorzystując co najmniej dwa procesory lub kilka rdzeni procesora w systemie. **Planują i przydzielają zadania do dostępnych procesorów**, aby zapewnić mn.w. równe wykorzystanie zasobów oraz mn.w. równe obciążenie. Np: Windows, Linux, Unix.

Systemy operacyjne czasu rzeczywistego - systemy czasu rzeczywistego muszą być w stanie obsłużyć zdarzenia w określonym czasie. Twarde systemy czasu rzeczywistego są systemami w których operacje muszą być wykonane bezwzględnie w określonym czasie (systemy kontroli produkcji, systemy bezpieczeństwa, systemy wojskowe, systemy lotnicze, systemy medyczne). Miękkie systemy czasu rzeczywistego są systemami w których wykonanie operacji z niepożądanym opóźnieniem, nie powoduje poważnych strat (systemy multimedialne, systemy rozrywkowe, systemy telekomunikacji, systemy zarządzające budynkiem). Np: FreeRTOS, VxWorks, QNX, eCos.

Systemy operacyjne dla komputerów mainframe - są w stanie obsługiwać znacznie więcej urządzeń wejścia-wyjścia (np. mogą obsługiwać ponad 1000 dysków). Mogą wykonywać wiele zadań jednocześnie i zazwyczaj oferują trzy usługi: **przetwarzania wsadowe, przetwarzanie transakcji oraz podział czasu**.

- **Systemy wsadowe** są systemami, które wykonują zadania automatycznie i sekwencyjnie, bez interakcji z użytkownikiem. Obecnie są stosowane np. w przetwarzaniu transakcji finansowych, w generowaniu raportów sprzedaży.
- **Systemy przetwarzania transakcji** służą do przetwarzania operacji, które prowadzą do zmiany zasobów oraz obsługują dużą liczbę żądań. Obecnie przykładowe zastosowania: przetwarzanie czeków, systemy rezerwacji.
- **Systemy z podziałem czasu** pozwalają wielu zdalnym użytkownikom na jednoczesne uruchomienie zadań na komputerze.

Np: OS/390, BS2000.

Systemy operacyjne dla serwerów - dostosowane do działania na serwerach, czyli komputerach lub innych narzędziach, które **pozwalają na dostarczenie zasobów lub współdzielenie usług, dla wielu użytkowników jednocześnie, przez sieć**. Np: Linux, Windows Server, OS X Server, FreeBSD, Solaris.

Systemy operacyjne dla komputerów osobistych - dostosowane do działania na komputerach osobistych. Większość systemów operacyjnych dla komputerów oferuje wygodny interfejs graficzny dla użytkownika, wygodne zarządzanie plikami i folderami, uruchamianie programów oraz umożliwia dostęp do internetu. Np: Linux, Windows, macOS, FreeBSD.

Systemy operacyjne dla komputerów przenośnych - dostosowane do działania na komputerach przenośnych, tj. smartfony, tablety, notebooki.

Systemy operacyjne wbudowane - dostosowane do działania na urządzeniach wbudowanych, czyli urządzeniach elektronicznych, które są przeznaczone do pełnienia konkretnych zadań lub funkcji. Systemy wbudowane **nie wymagają instalowania aktualizacji i są odporne na szkodliwe oprogramowanie - całe oprogramowanie jest zapisane w pamięci tylko do odczytu**. Np: FreeRTOS, VxWorks, QNX, MicroC/OS.

1.3 Praca komputera w trybie jądra i trybie użytkownika

Większość komputerów pracuje w dwóch trybach:

- **tryb jądra** (inaczej tryb nadzorcy) (ang. supervisor mode) - można uruchomić każdą instrukcję, którą komputer jest zdolny uruchomić.
- **tryb użytkownika** - można uruchomić tylko pewien podzbiór instrukcji, które komputer jest zdolny uruchomić.

System operacyjny pracuje w trybie jądra.

2 Systemy plików

2.1 System plików

System plików to metoda i struktura organizacji danych na nośnikach. Umożliwia on:

- przechowywanie danych
- utworzenie struktury katalogów ułatwiającą szeregowanie plików
- udostępnienie przechowywanych danych przy pomocy ścieżek
- zarzadzanie uprawnieniami dostępu do plików i katalogów
- tworzenie kopii zapasowych i umożliwienie odzyskiwania danych

2.2 Przykładowe systemy plików

- **Ext4** - utworzony dla systemów Linux.
- **NTFS** - utworzony dla systemów Windows.
- **FAT** - utworzony dla systemów Windows.
- **UFS** - utworzony dla systemów Unix.
- **HFS+** - utworzony dla systemów macOS.
- **APFS** - utworzony dla systemów macOS.

2.3 Zastosowania plików w systemie Linux

Plik zwykły - służy do przechowywania danych tekstowych, binarnych, obrazów, dźwięków, filmów, itp.

Katalog - służy do przechowywania plików i innych katalogów. (katalogi organizują strukturę systemu plików)

Plik specjalny - służy do interakcji z urządzeniami wejścia-wyjścia.

- **Urządzenie blokowe** - pliki dla urządzenia obsługujące dane o stałym rozmiarze, tj. dyski twarde, pendrive, karta graficzna.
- **Urządzenie znakowe** - pliki dla urządzenia obsługujące dane o zmiennej długości, tj. klawiatura, mysz, drukarka.

dowiązanie (ang. link) - służy do przechowywania odnośnika do pliku lub katalogu.

- **dowiązanie twarde** - odnośnik do pliku lub katalogu, wskazuje na to samo miejsce w pamięci masowej. Po usunięciu pliku, dowiązanie nadal pozostaje aktywne
- **dowiązanie symboliczne** - zawiera jedynie ścieżkę do pliku lub katalogu. Po usunięciu pliku, na który wskazuje, dowiązanie może zostać nieaktywne.

gniazdo (ang. socket) - służy do komunikacji międzyprocesowej. Stosowane głównie w oprogramowaniu sieciowym

potok nazwany (ang. named pipe) - służy do komunikacji międzyprocesowej na zasadzie potoku. Stosowane głównie na tym samym komputerze.

2.4 Zastosowanie systemowych katalogów w systemie Linux

- / - katalog główny, korzeń systemu plików.
- /**bin** - katalog zawierający pliki wykonywalne dostępne dla wszystkich użytkowników.
- /**boot** - katalog zawierający pliki niezbędne do uruchomienia systemu.
- /**dev** - katalog zawierający pliki specjalne reprezentujące urządzenia.
- /**etc** - katalog zawierający pliki konfiguracyjne, administracyjne i inne pliki systemowe dla systemu lokalnego.
- /**home** - przechowuje katalogi domowe dla użytkowników.
- /**lib** - katalog zawierający współdzielone biblioteki systemowe.
- /**lost+found** - przechowuje pliki uszkodzone lub zgubione (takie, które nie mają przypisanego katalogu).
- /**mnt** - przeznaczony do tymczasowego montowania systemów plików.

- **/opt** - katalog zawierający opcjonalne pakiety oprogramowania.
- **/proc** - katalog zawierający informacje o stanie aktualnie działających procesów.
- **/root** - katalog domowy dla użytkownika root.
- **/run** - system plików tmpfs (system plików w pamięci RAM), który zawiera pliki i katalogi używane przez różne procesy.
- **/sbin** - katalog zawierający pliki wykonywalne, które wykonują czynnności administracyjne.
- **/sys** - przechowuje informacje potrzebne do zarządzania urządzeniami systemu operacyjnego.
- **/tmp** - katalog zawierający pliki tymczasowe.
- **/usr** - katalog zawierający pliki (programy, biblioteki, pliki konfiguracyjne) przeznaczone dla użytkowników systemu.
- **/var** - katalog zawierający pliki, które często ulegają zmianie, tj. logi, kopie zapasowe, maile.

2.5 Prawa dostępu do plików w systemie Linux

Systemy typu Unix umożliwiają określenie praw dostępu dla:

- właściciela pliku lub katalogu;
- grupy do której plik lub katalog należy;
- pozostałych użytkowników, którzy nie są właścicielem, ani nie należą do grupy pliku lub katalogu.

Standardowe prawa dostępu:

- **prawo czytania** (ang. read) pozwala na odczyt pliku lub zawartości do katalogu;
- **prawo zapisu** (ang. write) pozwala na zapisanie lub modyfikacje pliku lub zmianę zawartości katalogu;
- **prawo wykonywania** (ang. execute) pozwala na wykonanie (uruchomienie) pliku (tyczy się skryptów, albo programów) lub otwierania katalogu.

Systemy Unix udostępniają możliwość korzystania z rozszerzonych praw dostępu, takich jak:

- **Bit SUID** (ang. set user ID) - stosowany głównie do plików wykonywalnych. Powoduje, że program dziedziczy uprawnienia właściciela, co umożliwia mu obsługę danych, do których ma dostęp administrator. Jest ustawiany jako prawo wykonania dla użytkownika (właściciela) (s).

- **Bit SGID** (ang. set group ID) - stosowany głównie do katalogów, plików wykonywalnych i skryptów. Powoduje, że nowo utworzony plik lub katalog ma przypisaną grupę katalogu, w którym został utworzony. Jest ustawiany jako prawo wykonania dla grupy (s).
- **Bit lepkości** (ang. sticky bit) - stosowany głównie do katalogów. Powoduje, że użytkownicy, którzy nie są właścicielami ani nie należą do grupy katalogu, nie mogą usuwać zawartości katalogu, której nie są właścicielami, nawet jeśli posiadają prawo zapisu. Jest ustawiany jako prawo wykonania dla innych użytkowników (t).

2.6 chmod

Właściciel			**Grupa**			**Inni**		
r	**w**	**x**	**r**	**w**	**x**	**r**	**w**	**x**
4	2	1	4	2	1	4	2	1

u	**g**	**o**
s	**s**	**t**
4	2	1

Modyfikacja przy pomocy notacji symbolicznej:

- `chmod u=rw direct` - przypisuje właścicielowi prawa do odczytu i zapisu.
- `chmod u+rx file.sh` - dodaje właścicielowi prawa do zapisu i wykonania.
- `chmod go-rwx file.c` - odbiera wszystkie prawa grupie i pozostałym użytkownikom.
- `chmod a+r file.txt` - dodaje wszystkim prawa do odczytu.
- `chmod u+s plik` - dodaje bit SUID dla właściciela pliku.
- `chmod u-s plik` - odbiera bit SUID dla właściciela pliku.
- `chmod g+s katalog` - dodaje bit SGID dla grupy katalogu.
- `chmod g-s katalog` - odbiera bit SGID dla grupy katalogu.
- `chmod o+t katalog` - dodaje bit lepkości dla innych użytkowników.
- `chmod o-t katalog` - odbiera bit lepkości dla innych użytkowników.
- `chmod 3755 katalog` przypisuje bit SGID (2) i bit lepkości (1) (2+1=3)

2.7 umask

umask - umożliwia określenie domyślnych praw dla nowo tworzonych plików i katalogów. Domyślne prawa dla plików to 666, a dla katalogów 777. Umask określa, które prawa zostaną odjęte od tych wartości.

Umask	Prawa dla plików	Prawa dla katalogów
000	666	777
022	644	755
XYZ	666 - X*100 - Y*10 - Z	777 - X*100 - Y*10 - Z

2.8 Deskryptory i potoki

Deskryptor pliku to identyfikator mający na celu identyfikację źródła pobierania i wysyłania danych.

- **0** - standardowy strumień wejścia (`stdin`) to miejsce, z którego program pobiera dane (np. pobrane z klawiatury).
- **1** - standardowy strumień wyjścia (`stdout`) to miejsce, gdzie program może wysłać dane (np. do drukarki, wyświetlane na ekran).
- **2** - standardowy strumień błędów (`stderr`) to miejsce, gdzie program może wysłać informacje o błędach (np. na ekran).

Przekierowanie to mechanizm umożliwiający zmianę miejsca z którego są pobierane lub przesyłane dane. Operatory przekierowania:

- `polecenie < plik` - przekierowanie standardowego wejścia.
- `polecenie > plik` - przekierowanie standardowego wyjścia (nadpisanie).
- `polecenie » plik` - przekierowanie standardowego wyjścia (dopisanie). Powyższe operatory mogą być poprzedzone numerem deskryptora.
- `polecenie &> plik` - przekierowanie standardowego wyjścia i błędów (nadpisanie).
- `polecenie &» plik` - przekierowanie standardowego wyjścia i błędów (dopisanie).

Operatory zarządzające deskryptorami:

- `>& numer` - przekierowanie wyjścia na deskryptor o podanym numerze.
- `<&-` - zamknięcie standardowego wejścia.
- `>&-` - zamknięcie standardowego wyjścia.
- `<& numer` - pobiera wejście z deskryptora o podanym numerze.

- `numer <> plik` - otwiera deskryptor o podanym numerze i przypisuje do niego plik.

Potok (ang. pipe) służy do przekazywania standardowego wyjścia jednego polecenia na standardowe wejście innego polecenia.

Operatory potoków:

- `polecenie1 | polecenie2` - przekazuje standardowe wyjście polecenia1 na standardowe wejście polecenia2.
- `polecenie1 |& polecenie2` - przekazuje standardowe wyjście i błędy polecenia1 na standardowe wejście polecenia2.

FIFO, inaczej potok nazwany (ang. named pipe) to specjalny rodzaj pliku, który umożliwia komunikację między procesami. Przykłady użycia FIFO:

- **Utworzenie FIFO**: `mkfifo myfifo`
- **FIFO jako wyjście dla programu**: `ls > myfifo`
- **FIFO jako wejście dla programu**: `cat < myfifo`
- **Oczekiwanie na dane w FIFO**: `cat myfifo`

3 Powłoki

3.1 Powłoka

Powłoka (ang. shell) to interpreter poleceń oraz język programowania, który pozwala na tworzenie skryptów.

Podpowłoka (ang. subshell), to proces, który jest tworzony przez bieżącą powłokę i symuluje pracę powłoki. W podpowłoce wykonywane są zadania oraz przekazywane są informacje odnośnie wyjścia i błędów do powłoki.

Zmienne środowiskowe są przechowywane i zarządzane przez powłokę. Zmienne środowiskowe są dostępne dla wszystkich procesów potomnych.

3.2 Przykładowe rodzaje powłok

Pierwsza **powłoka sh** (Bourne Shell) dla systemu Unix Version 7, została stworzona przez Stephena Bourne'a w latach siedemdziesiątych. Wprowadziła ona wiele koncepcji w dziedzinie skryptów, poleceń i zarządzania systemem Unix.

Powłoki **kompatybilne z sh:**

- **Bash** (Bourne Again Shell)
- **Dash** (Debian Almquist Shell)
- **Ksh** (Korn Shell)

Powłoki **niekompatybilne z sh:**

- **Csh** (C Shell)
- **Fish** (Friendly Interactive Shell)

Powłoki systemu Windows:

- **Cmd** (Command Prompt)
- **PowerShell**
- **WSL** (Windows Subsystem for Linux)

3.3 Zastosowanie zmiennych środowiskowych

Przykładowe zmienne środowiskowe:

- **HOME** - ścieżka do katalogu domowego użytkownika.
- **PATH** - ścieżka do katalogów, w których znajdują się pliki wykonywalne.
- **PWD** - ścieżka do katalogu roboczego.
- **USER** - nazwa użytkownika.
- **LANG** - ustawienia językowe.
- **SHELL** - ścieżka do powłoki.
- **IFS** - separator pól (domyślnie spacja).
- **PS1** - znak zachęty.
- **PS2** - wtórny znak zachęty.

3.4 Zastosowanie plików startowych i konfiguracyjnych

Pliki startowe zawierają informacje niezbędne do zainicjowana pracy powłoki. Powłoka wykonuje pliki startowe kiedy jest uruchamiana.

- **Plik** `/etc/profile` - plik wykonywany jako pierwszy podczas logowania - zawiera ustawienia dla wszystkich użytkowników np. zmienne środowiskowe
- **Plik** `/etc/profile.d` - katalog, w którym znajdują się skrypty konfiguracyjne, które są uruchomiane po wykonaniu.
- **Plik** `~/.bash_profile, ~/.bash_login, ~/.profile` - pliki, które są wykonywane po wykonaniu pliku `/etc/profile` - zawierają ustawienia dla indywidualnych użytkowników (np. mogą modyfikować globalne ustawienia).

- **Plik** `~/.bash_logout` - polecenia wykonywane podczas wylogowywania, np. usuwanie plików tymczasowych.
- **Plik** `/etc/bashrc` - uruchamiany za każdym razem, kiedy uruchamiana jest nowa sesja powłoki, np. poprzez terminal - zawiera ustawienia dla interaktywnych sesji powłoki dla wszystkich użytkowników.
- **Plik** `~/.bashrc` - wykonywany po wykonaniu pliku /etc/bashrc - zawiera ustawienia dla interaktywnych sesji powłoki dla indywidualnych użytkowników.

3.5 Zarządzanie operatorami logicznymi interpretera poleceń

Operatory logiczne:

- || - operator alternatywy (OR) - wykonuje kolejne polecenia, jeśli poprzednie zakończyły się niepowodzeniem.
- && - operator koniunkcji (AND) - wykonuje kolejne polecenia, jeśli poprzednie zakończyły się sukcesem.

Inne operatory:

- **;** - operator sekwencji - wykonuje kolejne polecenia niezależnie od wyniku poprzednich.
- & - operator tła - uruchamia polecenie w tle.

4 Systemy komputerowe

4.1 System komputerowy

System komputerowy to układ współpracujących ze sobą dwóch składowych: sprzętu komputerowego (ang. hardware) oraz oprogramowania (ang. software).

Warstwy systemu komputerowego:

- **Sprzęt komputerowy** - to wszystkie elementy fizyczne, które tworzą system komputerowy.
- **System operacyjny** - to oprogramowanie, które zarządza zasobami sprzętowymi oraz umożliwia korzystanie z komputera.
- **Programy narzędziowe** - wspomagają zarządzanie zasobami sprzętowymi oraz usprawniają oprogramowanie systemowe.
- **Programy użytkowe** - rozwiązują problemy zadane przez użytkownika (programy obliczeniowe, algorytmy)
- **Użytkownicy** - ludzie, urządzania lub inne podmioty, które realizują zadania przy pomocy programów użytkowych

4.2 Oprogramowanie urządzeń wejścia-wyjścia

Oprogramowanie związane z urządzeniami wejścia-wyjścia przeważnie jest zorganizowane w warstwach:

- **Procedury obsługi przerwania - przerwanie** to działanie za pomocą którego zewnętrze układy sygnalizują zajście jakiegoś zdarzenia i żądają określonego działania. Przerwania powodują wstrzymanie wykonywania programu, wywołanie procedury, która wykonuje określoną czynność i na koniec wydaje instrukcję powrotu do wykonywanego programu.
- **Sterowniki urządzeń** - oprogramowanie odpowiedzialne za komunikację między urządzeniem a pozostałymi elementami systemu komputerowego.
- **Oprogramowanie niezależne od urządzenia** - oprogramowanie działające na poziomie systemu operacyjnego, które ma na celu dostarczenie wspólnego interfejsu dla różnych urządzeń (buforowanie, raportowanie, przydzielanie zwolnienia urządzeń).
- **Oprogramowanie poziomu użytkownika** - oprogramowanie działające na poziomie aplikacji użytkownika, które komunikują się z urządzeniami wejścia-wyjścia za pośrednictwem interfejsów dostarczanych przez warstwę oprogramowania niezależnego od urządzania (wyświetlenie na ekran, pobieranie danych).

4.3 Procesor

Procesor (CPU, ang. Central Processing Unit) jest układem odpowiedzialnym za sterowanie innymi układami lub elementami np. komputera. Podstawowe elementy procesora:

- **Rdzenie** - wykonują instrukcje programu i zawierają między innymi:
  - **Układ sterowania** - kontroluje pracę procesora i wytwarza sygnały potrzebne do pracy z niektórymi elementami komputera.
  - **Arytmometr** (ALU) - służy do wykonywania operacji arytmetycznych, logicznych, przesunięć bitowych itp
  - **Rejestry** - przechowują dane i wyniki.
- **Pamięć podręczna Cache** - przechowuje ostatnio przetwarzane dane, co poprawia szybkość dostępu do informacji, które mają zostać ponownie wykorzystane.
- **Wewnętrzne szyny** - łączą elementu procesora z innymi elementami. które mają zostać ponownie wykorzystane.

4.4 Pamięci ROM i RAM

Pamięć ROM (ang. Read Only Memory) pamięć tylko do odczytu, która jest odporna na spadki napięcia. Zapisuje się na niej informacje, niezbędne do rozruchu urządzenia oraz dotyczące elementów budowy komputera (np. porty dysku).

Pamięć RAM (ang. Random Access Memory) pamięć operacyjna, która przechowuje informacje odnośnie aktualnie przetwarzanych programów. Procesor współpracuje z pamięcią operacyjną, która przechowuje różne dane, instrukcje programów itp. stosując jednobajtowe komórki pamięci oraz ich numery adresów.

4.5 Mechanizmy zarządzania pamięcią operacyjną

Przestrzeń adresowa to zbiór wszystkich dopuszczalnych adresów w pamięci.

- **przestrzeń fizyczna** - zbiór fizycznych adresów przekazanych do układów pamięci głównej.
- **przestrzeń logiczna** - zbiór adresów generowanych przez procesor w kontekście realizowanego zadania. **Adres logiczny** identyfikuje komórkę pamięci na poziomie kodu maszynowego w kontekście realizowanego zadania.

Procesory korzystają z **pamięci wirtualnej**. Mechanizm zarządzania pamięcią wirtualną polega na tym, że zadanie korzysta z obszaru pamięci mając wrażenie że korzysta z ciągłego obszaru, chodź w rzeczywistości **pamięć fizyczna może być pofragmentowana i nieciągła**. Możliwa jest też wymiana z pamięcią na dysku.

Procesor realizując zadanie korzysta z **mechanizmu segmentacji**, co oznacza, że pamięć jest dzielona na segmenty, które pełnią określone przeznaczenie.

W systemach wieloprocesowych procesor przechowuje **selektory w rejestrach segmentowych**, które **opisują własności początkowych adresów segmentów**. Budowa selektora:

Indeks	**TI**	**RPL**

Deskryptor to struktura, która **zawiera informacje dotyczące pewnego zasobu lub segmentu pamięci**. Deskryptor może zawierać informacje tj. **prawa dostępu, adres bazowy, rozmiar, typ zasobu itp**. **Tablica deskryptorów** to struktura, która przechowuje deskryptory. Tablica deskryptorów jest tworzona w pamięci operacyjnej i jest zarządzana przez system operacyjny (jest często tworzona i używana w systemach komputerowych, które korzystają z segmentacji pamięci). Tablica deskryptorów jest przydzielana każdemu procesowi oraz istnieje tablica deskryptorów, która definiuje opisy segmentów pamięci. Tablica deskryptorów może zawierać informacje odnośnie numeru

deskryptora pliku. System operacyjny przydziela numer deskryptora (przeważnie od 3 wzwyż) podczas otwarcia pliku lub innego zasobu wejścia/wejścia. Istnieją stałe numery deskryptorów jak 0 (stdin), 1 (stdout), 2 (stderr).

Procesor umożliwia korzystanie z mechanizmu zwanym **stronicowaniem**, który stosuje dwupoziomową tablicę stron: katalog stron zawierający adresy do tablicy stron i tablicę stron zawierającą adresy do stron.

32-bitowy adres w mechanizmie stronicowania:

Katalog (31-22)	**Strona** (21-12)	**Offset** (11-0)

Schemat wyznaczania adresu pamięci:

1. Na podstawie selektora z rejestru segmentowego wyznaczany jest deskryptor z tablicy deskryptorów.
2. Do wyznaczonego deskryptora dodawane jest przesunięcie (offset).
3. Jeżeli stronicowanie jest wyłączone na podstawie adresu deskryptora i przesunięcia wyznaczany jest adres.
4. Jeżeli stronicowanie jest włączone na podstawie deskryptora i przesunięcia wyznaczany jest adres katalogu stron, adres strony i przesunięcie (offset). Na podstawie wartości katalogu stron wyznaczany jest katalog stron. Na podstawie wyznaczonego katalogu stron wyznaczany jest adres strony. Na podstawie wyznaczonego adresu strony i przesunięcia wyznaczany jest adres.

4.6 Dysk twardy

Dysk twardy (inaczej: dysk magnetyczny) to urządzenie pamięci masowej, które może być używane w komputerach i innych urządzeniach, oraz służy do długoterminowego przechowywania danych.

- **Dysk twardy składa się z jednego lub więcej talerzy** obracających się z szybkością 5400 obr./min. lub 7200 obr./min. (rzadziej 10800 obr./min.). Nad talerzami przesuwa się mechaniczne ramię, które zawiera głowice odczytujące dane.
- **Ścieżka** to okrężny tor, na którym mogą być odczytywane i zapisywane dane na dysku twardym.
- **Cylinder** to zbiór ścieżek na wszystkich talerzach, które znajdują się na tym samym poziomie.
- Ścieżka zawiera **sektory** w których zapisywane są dane. Zazwyczaj każdy sektor ma 512 bajtów lub 4 kilobajty. Budowa sektora:

preambuła	**dane**	**error correction code (ECC)**

4.7 Przygotowanie dysku twardego

Formatowanie niskopoziomowe to proces mający na celu przygotowanie dysku twardego do zapisu danych. Pojęcia związane z formatowanie niskopoziomowym i fizyczną organizacją dysku twardego:

- **Przeplot** - określa sposób zapisywania danych (np. co drugi sektor na dysku) - mechanizm ułatwia przeszukiwanie sektorów w celu zapisu lub odczytu danych.
- **Przekos** - określa czasowe opóźnienie między odczytem lub zapisem kolejnych sektorów - mechanizm zwiększa wydajność.

Partycjonowanie to proces podziału dysku twardego na logiczne obszary, które nazywamy **partycjami**. Partycjonowanie jest przeważnie wykonywane po wykonaniu formatowania niskopoziomowego. W pierwszym sektorze znajduje główny rekord startowy, który zawiera m.in **kod uruchamiający system operacyjny i tablicę partycji, która zawiera informację na temat sektora początkowego**.

Formatowanie wysokopoziomowe to proces mający na celu utworzyć blok administracyjny tj. katalog główny oraz system plików.

Defragmentacja to proces polegający na optymalizacji wydajności pracy dysku twardego, który polega na przesuwaniu fragmentów danych na dysku. Podczas tworzenia, edytowania, usuwania danych, dane mogą być rozmieszczane na różnych miejscach dysku (fragmentacja), co może się przyczynić do pogorszenia wydajności. Dyski twarde często posiadają sektory zapasowe i korzystają z mechanizmów, które pozwalają na zmianę lokalizacji uszkodzonego sektora na sektor zapasowy.

4.8 Technologie zarządzania wieloma dyskami (RAID)

RAID (ang. Redundant Array of Independent Disks) to technologia, która służy do łączenia wielu dysków twardych w jeden logiczny system. Ma to na celu zwiększenie pojemności, niezawodności lub wydajności. Przeważnie polega to na zastosowaniu szafy dyskowej obok komputera (przeważnie serwera) oraz zastąpieniu kontrolera dysków na kontroler macierzy RAID. RAID jest tworzony z wielu fizycznych dysków twardych, ale z poziomu oprogramowania (w tym systemu operacyjnego) zazwyczaj jest widziany jako pojedynczy dysk logiczny. Możliwych jest kilka organizacji macierzy RAID.

- **RAID 0** (paskowanie, ang. striping) polega na **podzieleniu pojedynczego dysku na fragmenty składające się z sektorów**. RAID 0 zwiększa wydajność odczytu i zapisu danych (możliwe jest używanie kilku fragmentów jednocześnie). RAID 0 nie zabezpiecza przed utratą danych.

- **RAID 1** (lustrzany zapis, ang. mirroring) polega na **kopiowaniu danych na innych dysk**. RAID 1 zwiększa wydajność odczytu, ponieważ dane mogą być odczytywane z dwóch dysków jednocześnie, ponadto zwiększa niezawodność tzn. jeśli jeden dysk ulegnie awarii, dane będą na innym dysku. RAID 1 umożliwia korzystanie jedynie z połowy dostępnej pojemności dysku.

- **RAID 2** (dzielenie bitów, ang. bit slicing) polega na **zapisywaniu każdego bitu na fragmentach różnych dysków**. RAID 2 zwiększa niezawodność, tzn. uszkodzenie jednego dysku, spowoduje utratę tylko jednego bitu informacji. Do zapisu danych można można używać kodów korekcyjnych tj. kod Hamminga. RAID 2 wymaga korzystania z dużej liczby dysków oraz zsynchronizowanej rotacji talerzy.

- **RAID 3** polega na **zapisywaniu każdego bitu na fragmentach różnych dysków oraz bitu parzystości** (ang. parity). RAID 3 zwiększa niezawodność, tzn. uszkodzenie jednego dysku, umożliwia odzyskanie danych przy pomocy bitu parzystości (wynik operacji XOR pozostałych bitów). RAID 3 wymaga korzystania z dużej liczby dysków oraz zsynchronizowanej rotacji talerzy, uszkodzenie dysku parzystości uniemożliwia odzyskanie danych.

- **RAID 4** polega na **podzieleniu pojedynczego dysku na fragmenty oraz zapisu bitu parzystości każdego fragmentu na jednym dysku**. RAID 4 zwiększa niezawodność, tzn. uszkodzenie jednego dysku, umożliwia odzyskanie danych przy pomocy bitu parzystości (wynik operacji XOR pozostałych bitów). RAID 4 zapewnia niską wydajność zapisu (jeśli zmieni się jeden sektor trzeba aktualizować wszystkie dyski), uszkodzenie dysku parzystości uniemożliwia odzyskanie danych.

- **RAID 5** polega na **rozmieszczeniu bitów parzystości na wszystkich dyskach (rozkłada dane i bity parzystości w różnych miejscach)**. RAID 5 większą niezawodność, ponieważ przetrzymuje bity parzystości na różnych dyskach. RAID 5 zapewnia niską wydajność zapisu (jeśli zmieni się jeden sektor trzeba aktualizować wszystkie dyski)

- **RAID 6** polega na **rozmieszczeniu dwóch odpowiedników bitów parzystości na wszystkich dyskach**. RAID 6 ma większą niezawodność, ponieważ przetrzymuje dwa odpowiedniki bitu parzystości na różnych dyskach. RAID 6 zapewnia niższą wydajność zapisu (trzeba aktualizować wszystkie dyski, w tym dwa odpowiedniki bitów parzystości).

4.9 Rozruch systemu komputerowego

W pamięci stałej plyty glownej (read only), znajdują się oprogramowania sprzętowe (ang. firmware) tj. program **BIOS** (starsze komputery), albo **UEFI** (nowsze komputery). **BIOS** (ang. Basic Input Output System) i **UEFI** (ang. Unified Extensible Firmware Interface) to **programy odpowiedzialne za inicjalizację i zarządzanie sprzętem systemu komputerowego**. BIOS i UEFI zawierają miedzy innymi procedury, które umożliwiają odczytywanie danych z klawiatury, czy wypisywania tekstu na ekran.

Schemat rozruchu systemu komputerowego:

1. Po włączeniu komputera uruchamia się **BIOS lub UEFI**.
2. Sprawdzana jest **ilość zainstalowanej pamięci RAM oraz poprawność działania urządzeń systemu komputerowego** (skanowane są złącza ISA i PCI).
3. BIOS lub UEFI **rejestrują oraz ewentualnie konfigurują nowe urządzenia**.
4. **Określane jest urządzenie rozruchowe** poprzez sprawdzenie urządzeń zapisanych na liście w pamięci (urządzenia te można zmienić w programie konfigurującym BIOS lub UEFI).
5. Jeśli nie zostaną wykryte dyskietki, ani płyty startowe w CD-ROM, BIOS lub UEFI probiera **MBR** (ang. Master Boot Record) (starsze komputery), albo **GPT** (ang. GUID Partition Table) (nowsze komputery) z pierwszego sektora dysku twardego, które zawierają informacje o strukturze partycji na dysku. MBR zawiera program rozruchowy; w przypadku GPT program rozruchowy jest zapisany w osobnej partycji.
6. **Program rozruchowy** (ang. bootloader) wczytuje system operacyjny i uruchamia go na aktualnej partycji.

5 Wielozadaniowość

5.1 Procesy i wątki

- **Proces** to instancja programu działającego w systemie operacyjnym.
- **Demon** to rodzaj procesu, który działa w tle i wykonuje określone zadania bez interakcji z użytkownikiem.
- **Wątek** (ang. thread) to fragment zadania, który jest wykonywany współbieżnie w procesie systemu operacyjnego. **Proces może zawierać kilka wątków.**
- **Sygnał** to mechanizm komunikacji między procesami i z systemem operacyjnym.

Przykładowe sygnały Unix/Linux:

- **SIGTSTP** - zatrzymanie procesu
- **SIGSTOP** - zatrzymanie procesu
- **SIGCONT** - wznowienie procesu
- **SIGHUP** - zakończenie połączenia
- **SIGCHLD** - zakończenie procesu potomnego
- **SIGUSR1/SIGUSR2** - sygnały użytkownika (brak domyślnego działania)

5.2 Hierarchiczna struktura procesów

- **Proces główny** (ang. main process) - proces, który pelni kluczową rolę podczas wykonywania zadania.
- **Proces nadrzędny** (ang. parent process) - proces, który utworzył proces podrzędny.
- **Proces podrzędny** (ang. child process) - proces, który został utworzony przez proces nadrzędny.

5.3 Stany procesów

Możliwe stany procesów:

- **Nowy** - proces został utworzony
- **Gotowy** - proces oczekuje na przydzielenie zasobów
- **Aktywny** - proces jest wykonywany
- **Czekający** - proces oczekuje na zdarzenie
- **Zakończony** - proces zakończył działanie

5.4 Metody synchronizacji

Region krytyczny to fragment programu z którego może skorzystać kilka procesów lub wątków (np. pamięć współdzielona).

Muteks to mechanizm synchronizacyjny, który jest stosowany do zarządzania wzajemnego wykluczania się.

- Zablokuj Muteks
- Region krytyczny - dostęp do współdzielonych zasobów
- Odblokuj Muteks

Muteks, zazwyczaj przyjmuje dwie wartości: odblokowany i zablokowany.

Semafor to mechanizm synchronizacyjny, który umożliwia zarządzanie dostępem do współdzielonych zasobów poprzez inkrementację i dekrementację wartości semafora.

- Jeżeli semafor jest równy 0 - zatrzymaj się
- Jeżeli semafor jest większy od 0 - dekrementuj wartość semafora
- Region krytyczny - dostęp do współdzielonych zasobów
- Inkrementuj wartość semafora
- Jeżeli istnieją zatrzymane procesy - uruchom jeden z nich

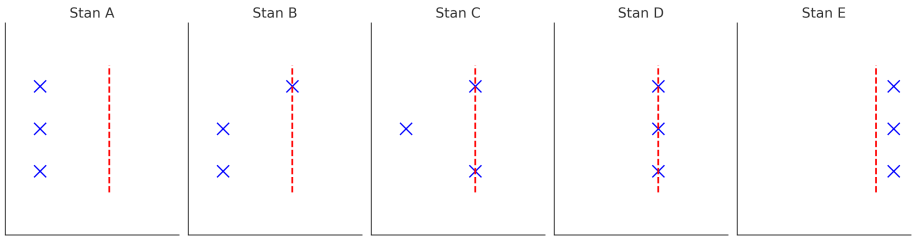
Semafor binarny może przyjąć jedynie wartości: 0, 1. Semafor licznikowy może przyjąć większą liczbę wartości.

Monitor to mechanizm synchronizacyjny, który zawiera pogrupowane dane (zmienne) i możliwe operacje na tych danych (procedury).

```
monitor myMonitor {
    int n;
    bool b;

    func update() {
        ...
    }
    func check() {
        ...
    }
}
```

Bariera to mechanizm synchronizacyjny, który polega na nakazaniu procesom lub wątkom zaczekanie na siebie, zanim zaczną kontynuację swojego zadania.



5.5 Procesy działające na pierwszym planie, w tle i niezależne od terminala

- `jobs` - wyświetla listę procesów działających w tle
- `fg` - wznawia proces w pierwszym planie
- `bg` - wznawia proces pozostawiając go w tle

Przykładowe wyjście polecenia `jobs`

```
[1]   Stopped    vim zad1.py
[2]-  Running       ls -l
[3]+  Stopped  python zad2.py
```

Wyświetlany jest numer zadania oraz informacja, czy zadanie jest aktywne, czy czekające. Poprzez + oznaczone jest ostatnio uruchomione zadanie, natomiast - oznacza przedostatnie zadanie.

Przykład wznawiania poleceń:

- `fg` 2 wznawia, na pierwszym planie, zadanie o numerze 2.
- `bg` wznawia, w tle, ostatnio wykonane zadanie.

`nohup` - uruchamia proces, który nie jest zależny od terminala

- `nohup ./skrypt.sh &` - uruchamia proces w tle, który nie jest zależny od terminala. Proces nie jest zatrzymywany, gdy otrzyma sygnał SIGHUP.

5.6 Elementy identyfikujące procesy

- `kill` - wysyła sygnał do procesu o podanym PID
- `killall` - wysyła sygnał do procesów o podanej nazwie
- `pkill` - wysyła sygnał do procesów, których nazwa pasuje do podanego wzorca
- `pgrep` - wyświetla PID procesów, których nazwa pasuje do podanego wzorca
- `kill 1234` - wysyła (domyślny) sygnał SIGTERM do procesu o PID 1234
- `kill -s SIGKILL 1234` - wysyła sygnał SIGKILL do procesu o PID 1234
- `kill -s 9 1234` - wysyła sygnał SIGKILL do procesu o PID 1234
- `killall -s SIGKILL vim` - wysyła sygnał SIGKILL do procesów o nazwie vim

Priorytet określa jak często proces otrzymuje dostęp do zasobów. Priorytety w Unix/Linux zazwyczaj mogą przyjąć wartości od -20 do 19 (domyślnie proces ma priorytet 0). Niższe wartości oznaczają wyższy priorytet, czyli -20 to najwyższy priorytet, 19 to najniższy priorytet. Im niższy priorytet (większa liczba) tym proces posiada mniej czasu procesora.

- `nice` - pozwala na uruchomienie procesu z określonym priorytetem
- `renice` - pozwala na zmianę priorytetu procesu
- `nice -n 10 ./skrypt.sh` - uruchamia proces z priorytetem 10
- `renice -n 15 -p 1234` - zmienia priorytet procesu o PID 1234 na 15

Inne przydatne komendy:

- `ps` - wyświetla listę aktualnie uruchomionych procesów, do których użytkownik ma dostęp
- `ps -L` - wyświetla informacje o wątkach
- `ps -f` - wyświetla informacje odnośnie listingu, tj. numer PPID czy liczba wątków w polaczeniu z `-L`
- `ps -e` - wyświetla wszystkie procesy systemowe
- `pstree` - wyświetla drzewo procesów
- `pstree -p` - wyświetla PID procesów
- `pstree -t` - wyświetla pełne nazwy wątków

- `exit` - powoduje zakończenie działania procesu i przekazuje jego kod wyjścia, który jest liczba jednobajtowa
- `return` - zwraca wartość funkcji, która jest liczba jednobajtowa
- `wait` - powoduje, że proces nadrzędny czeka na zakończenie procesu potomnego
- `trap` - przechwytuje sygnał i określa polecenia, które mają być wykonane po otrzymaniu sygnału
- `trap -l` - wyświetla listę sygnałów i ich identyfikatorów
- `trap -p` - wyświetla polecenia, jakie mają być wykonane po otrzymaniu sygnału
- `top, atop, htop, btop, glances` - interaktywne narzędzia do monitorowania procesów

6 Zarządzanie zasobami

6.1 Szeregowanie procesów

Jeżeli w komputerze wykorzystywana jest wielozadaniowość, często procesy lub wątki znajdujące się w tym samym stanie rywalizują o procesor.

Program szeregujący (ang. scheduler) to część systemu, która dokonuje wyboru procesu, który jako następny uzyska dostęp do procesora. Program szeregujący odpowiada również za wydajne działanie procesora podczas przełączania procesów.

W systemach operacyjnych można spotkać kilka **algorytmów szeregowania procesów**. Ich dwa główne typy to:

- **Algorytmy szeregowania bez wywłaszczania** - proces może być zakończony jedynie przez zablokowanie lub zakończenie działania.
- **Algorytmy szeregowania z wywłaszczaniem** - scheduler może zatrzymać proces po określonym czasie, przed zakończeniem wykonania zadania i przydziela procesor innemu procesowi.

Schemat przełączania procesów:

1. Przełączenie się z trybu użytkownika na tryb jądra;
2. Papisanie stanu bieżącego procesu, i ewentualnie mapę pamięci, tak by można było ponownie uruchomić proces;
3. Wybór nowego procesu na podstawie algorytmu szeregującego i załadowanie modułu z mapy pamięci;
4. Uruchomienie innego procesu.

Kiedy należy wykonać szeregowanie?

- Podczas tworzenia nowego procesu; scheduler musi zdecydować kiedy ma zostać uruchomiony.
- Podczas zakończenia pracy procesu; scheduler musi jaki proces ma go zastąpić.
- Podczas blokowania procesu przez semafor.
- Kiedy proces jest gotowy do wykonania działania.
- Kiedy wystąpi przerwanie.

Procesy można podzielić na dwa główne rodzaje ze względu na ich zadania i charakterystykę pracy:

- Procesy **zorientowane na obliczenia** (ang. CPU-bound) to procesy, które przez większość czasu wykonują obliczenia.
- Procesy **zorientowane na wejście-wyjście** (ang. I/O-bound) to procesy, które przez większość czasu oczekują na zakończenie operacji wejścia-wyjścia

6.2 Algorytmy szeregowania

Algorytmy szeregowania są rożne w zależności od specyfiki środowiska, ponieważ rożne systemy realizują rożne zadania.

Przykładowe cele algorytmów szeregowania:

- **Wszystkie systemy** - sprawiedliwe przydzielenie czasu procesora dla każdego procesu, sprawdzanie czy procesy przestrzegają zasad itp.
- **Systemy wsadowe** - maksymalizacja liczby wykonywanych zadań, zapewnienie ciągłości pracy procesora, minimalizacja czasu pomiędzy przetwarzaniem zadań, a jego zakończeniem. **Wywłaszczanie nie jest stosowane, albo jest stosowane z bardzo długim przydziałem czasowym dla procesora**.
- **Systemy interaktywne** - szybka odpowiedź na żądanie, proporcjonalne spełnianie oczekiwań użytkowników. **Wywłaszczanie pełni ważną rolę**, aby proces w nieskończoność nie korzystał z zasobów procesora (np. z powodu błędu w programie).
- **Systemy czasu rzeczywistego** - dotrzymywanie terminów, unikanie utraty danych. **Wywłaszczanie nie zawsze jest potrzebne**, ponieważ procesy po zakończeniu zadania szybko się blokują.

Szeregowanie w systemach wsadowych

Pierwszy zgłoszony, pierwszy obsłużony (ang. first come, first served) polega na tym, że procesy otrzymują dostęp do procesora w kolejności w której zażądały do niego dostępu.

Przykładowe zalety algorytmu:

- **łatwy do zrozumienia i zaimplementowania** (nie wykonuje skomplikowanych obliczeń);
- **Procesy są traktowane tak samo**, co może być korzystne w sytuacji, gdy mamy procesy o mniej więcej równym priorytecie.

Przykładowe wady algorytmu:

- **Może doprowadzić do marnowania czasu** jeśli pierwszy proces w kolejce będzie trwał znacznie dłużej niż procesy, które będą później;
- **Nie dostosowuje się do wymagań** tj. priorytety, czy czas wykonania.

Najkrótsze zadania najpierw obsłużone (ang. shortest job first lub shortest job next) polega na tym, że program szeregujący wybiera proces o najkrótszym czasie działania.

Przykładowe zalety algorytmu:

- **Algorytm minimalizuje średni czas przetwarzania wszystkich procesów**. Przykładowo dla zadań, które działają o czasach kolejno 8, 4, 2 minuty. przetwarzanie kolejnych zadań zajmie niecałe 12 min. ((8+12+14)/3);Początkowe przetwarzanie najkrótszych zadań zajmie niecałe 8 min. ((2+6+14)/3).

Przykładowe wady algorytmu:

- **Konieczna jest znajomość czasów lub szacunkowych czasów wykonania zadań** co może być trudne do uzyskania w niektórych środowiskach;
- **Wszystkie zadania muszą być dostępne jednocześnie**, aby algorytm działał optymalnie.

Szeregowanie w systemach interaktywnych

Szeregowanie cykliczne (ang. round robin) polega na tym, że każdemu procesowi przydzielany jest stały czas (kwant) w którym może wykonywać swoje działanie. Jeżeli proces nie zakończył swojego zadania po upływie czasu, zostaje on wywłaszczony i czeka na ponowny przydział do wykonania zadania. Jeżeli proces zakończył lub zatrzymał działanie, procesor przełącza kolejne zadanie.

Przykładowe zalety algorytmu:

- **łatwy do zrozumienia i zaimplementowania**;
- **procesy otrzymują taką samą szansę na dostęp do procesora**, co może być korzystne w sytuacji, gdy mamy procesy o mniej więcej równym priorytecie;

- **znany czas opóźnienia**, tzn. procesy wiedzą kiedy otrzymają ponownie dostęp.

Przykładowe wady algorytmu:

- **wydłużony czas wykonywania procesów**, które mają długi czas działania;
- **niska efektywność podczas wykonywania zadań o różnym czasie działania** (czasami krótkotrwałe zadania muszą czekać, aż długotrwałe zadania wykonają swoje czynności);
- **nie dostosowuje się do obciążenia systemu**, ani do parametrów procesów tj. priorytety.

Szeregowanie według priorytetów (ang. priority scheduling) polega na tym, że każdemu procesowi jest przydzielany priorytet i program szeregujący wybiera proces o najwyższy priorytecie.

Przykładowe zalety algorytmu:

- **dostosowanie szeregowania do charakterystyki zadania**, poprzez możliwość ustawienia różnych priorytetów;
- **pozwala na dynamiczne dostosowywanie priorytetów procesów w trakcie działania systemu**.

Przykładowe wady algorytmu:

- **procesy o niższych priorytetach mogą być blokowane przez procesy o wyższych priorytetach**;
- **właściwe określenie priorytetów dla różnych zadań czasami jest trudne**;
- **może dojść do "zapętlenia", jeżeli dochodzi do stałej zmiany priorytetów**.

Szeregowanie z wykorzystaniem kolejki priorytetów (ang. multilevel queue scheduling) polega na podzieleniu procesów na różne poziomy priorytetów, przy czym procesy mogą zmieniać swoje położenie w zależności od wykonanych działań.

Zasada działania kolejki priorytetów:

- Procesy należące do klasy o najwyższym priorytecie działają przez jeden kwant.
- Procesy należące do kolejnej klasy w hierarchii działają przez dwa kwanty.
- Procesy należące do kolejnej klasy w hierarchii działają przez cztery kwanty.
- Procesy należące do kolejnej klasy w hierarchii działają przez osiem kwantów itd.

- Jeżeli proces wykorzysta wszystkie swoje kwanty dostępne w danej klasie, jest przenoszony do klasy o niższym priorytecie.

Przykładowe zalety algorytmu:

- **automatyczne dostosowanie priorytetów i kwantów czasu do rodzaju zadania** (krótkotrwałe zadania są wykonywane szybciej, a długotrwałe przenoszone do klas o niższym priorytecie).

Przykładowe wady algorytmu:

- **bardziej skomplikowana implementacja**;
- różne implementacji mogą działać różnie w zależności od ustawień kolejki priorytetów (**brak jednoznacznych reguł**).

Szeregowanie gwarantowane (ang. guaranteed scheduling) polega na zapewnieniu, że każdy proces otrzyma minimalny czas działania np. każdy z n użytkowników otrzyma 1/n mocy procesora.

Przykładowe zalety algorytmu:

- zapewnienie, że **określone zadania zostaną wykonane w określonym czasie**;
- **procesy mogą być obsługiwane zgodnie z wymaganiami sprzętowymi**.

Przykładowe wady algorytmu:

- **możliwe marnowanie zasobów procesora**, gdy nie są wykorzystywane wszystkie;
- **przeciążenie systemu w przypadku dużej liczby procesów**;
- **wymagana jest precyzyjna ocena czasu wykonania zadań**.

Szeregowanie loteryjne (ang. lottery scheduling) polega na tym, że każdy proces otrzymuje tzw. bilety, następnie program szeregujący wykonuje "losowanie" biletu, który otrzyma dostęp do procesora (im więcej biletów ma proces tym większa szansa, że zostanie wybrany).

Przykładowe zalety algorytmu:

- **procesy, które mają mało biletów, nadal mają szansę na dostęp do zasobów procesora co oznacza, że nie zostaną zablokowane**;
- **procesy mogą dynamicznie zmieniać liczbę swoich biletów w trakcie działania systemu**.

Przykładowe wady algorytmu:

- **przydzielenie właściwej liczby biletów może być trudne**;

- **losowanie wprowadza dodatkowe obliczenia**, które mogą powodować opóźnienie wykonywanego zadania.

Algorytmy szeregowania mogą mieć przypisane wagi dla użytkowników, którzy uruchamiają procesy.

Przykład: Załóżmy, że z systemu, który wykorzystuje algorytm cykliczny, korzysta dwóch użytkowników.

- Pierwszy użytkownik uruchomił cztery procesy (P1P2P3P4), drugi uruchomił jeden proces (P5).
- Jeżeli każdy użytkownik ma 50% czasu procesora: możliwa kolejność wykonywania procesów: P1P5P2P5P3P5P4P5P1P5P2P5...
- Jeżeli pierwszy użytkownik ma dwa razy więcej czasu procesora niż drugi: możliwa kolejność wykonywania procesów: P1P2P5P3P4P5P1P2P5P3P4P5...

6.3 Zakleszczenie

Zakleszczenie (ang. deadlock) to sytuacja w której dochodzi zablokowania pewnego zbioru procesów, np. procesy oczekują na pewien zasób, który jest zajmowany przez inne procesy.

Zasobem może być urządzenie sprzętowe, blok informacji itp. Rodzaje zasobów:

- **zasoby z możliwością wywłaszczania**, to zasoby, które można odebrać procesom bez skutków ubocznych, np. pamięć;
- **zasoby bez możliwości wywłaszczania**, to zasoby, których nie można odebrać procesom bez skutków ubocznych, np. wypalana płyta CD

Warunki powstania zakleszczenia zasobów. Wszystkie warunki muszą być spełnione by doszło do zakleszczenia.

- **wzajemne wykluczanie** - każdy zasób jest przypisany do jednego procesu, albo jest dostępny;
- **procesy, które mają przydzielony zasób, żądają nowych zasobów**;
- **zasoby przydzielone wcześniej nie mogą być odebrane siłą**;
- **m**usi istnieć cykl, składający się z co najmniej dwóch procesów, w których **każdy proces oczekuje na zasób, który jest kontrolowany** przez następny proces w cyklu.

6.4 Przykładowe sposoby postępowania

Przykładowy algorytm ignorowania zakleszczeń

Algorytm strusia polega na tym, że "wkładamy głowę w piasek" i udajemy, że problemu nie ma.

Przykładowe algorytmy wykrywania zakleszczeń

Przykładowy **algorytm działający dla zasobów jednego typu**, np. dla systemu posiadającego jedną drukarkę, jedną nagrywarkę itd. Schemat działania:

1. Początkowo tworzymy graf procesów i zasobów.
2. Tworzymy pustą listę i umieszczamy "losowo" wybrany wierzchołek grafu.
3. Następnie umieszczamy losowo wybrany **wierzchołek, który ma łuk wychodzący z naszego wierzchołka**
4. Jeżeli nie istnieje żaden łuk wychodzący **usuwamy ostatnio wybrany wierzchołek** i sprawdzamy czy możemy wybrać dla niego łuk.
5. Jeżeli lista jest pusta, **sprawdzamy czy istnieje inny wierzchołek od którego wcześniej nie zaczynaliśmy**, jeżeli, tak, **umieszczamy go w liście**, w przeciwnym razie kończymy działanie wiedząc, że **nie ma zakleszczeń**.
6. Podczas umieszczania wierzchołka w liście sprawdzamy, czy nie został on już wcześniej umieszczony. Jeżeli tak, wiemy, że **graf posiada cykl i występuje zakleszczenie**.

Przykładowy **algorytm działający dla wielu zasobów każdego typu**.

1. Tworzymy dwa wektory określające istniejące (V_E) i dostępne zasoby (V_A) oraz dwa macierze określające przydział (M_C) i zapotrzebowanie (M_R) zasobów dla procesów.
2. Wybieramy wcześniej niewybrany proces, którego wiersz macierzy M_R jest mniejszy lub równy niż wektor V_A.
3. Jeżeli taki proces nie istnieje kończymy działanie z informacją, że **niewybrane procesy są zakleszczone**. Jeżeli zostaną wybrane wszystkie procesy **kończymy działanie z informacją, że zakleszczenia nie występują**.
4. Jeżeli wybierzemy proces dodajemy do V_A odpowiedni wiesz z M_C.

Przykładowe sposoby usuwania wykrytych zakleszczeń

Jeżeli zakleszczenie zostanie wykryte, algorytmy podejmują działania mające na celu usunięcie zakleszczenia.

- **Kończenie pracy procesów poprzez wysłanie sygnału** - można zakończyć pracę jednego lub więcej procesów, które są zakleszczone lub procesu, który nie jest zakleszczony, by zwolnić zasoby.
- **Cofanie operacji** - stan procesu jest rejestrowany (stan procesu jest zapisywany do pliku) i w przypadku powstania zakleszczenia można cofnąć proces do poprzedniego stanu.
- **Wywłaszczanie procesów** - zakleszczony proces można wywłaszczyć, dzięki czemu zwolni zasoby

6.5 Sposoby unikania zakleszczeń zasobów

Algorytm bankiera polega na sekwencyjnym przyznawaniu procesom zasobów w taki sposób, aby nie dopuścić do sytuacji, w której jakiś proces nie będzie w stanie zakończyć swojego działania. Algorytm bankiera może nie działać efektywnie, ponieważ procesy nie zawsze wiedzą z góry jakie zasoby są im potrzebne i wymaga informacji na temat przyszłych żądań Schemat działania algorytmu bankiera:

- **Wybieramy proces, który spełnia warunek** - proces, który nie został zakończony i dla którego potrzebne zasoby są mniejsze niż zasoby dostępne.
- **Przydzielamy zasoby procesowi** - jeżeli proces spełnia warunek, przydzielamy zasoby i oznaczamy proces jako zakończony.
- **Zwalniamy zasoby** - jeżeli proces zakończył działanie i zwalniamy zasoby.
- **Kończymy działanie** - jeżeli wszystkie procesy zostały zakończone, kończymy działanie.

Unikanie zakleszczeń w obecnych systemach polega na usunięciu jednego z czterech warunków, który powoduje powstanie zakleszczenia:

- wzajemne wykluczanie - **umożliwienie na dostęp do zasobu przez wiele procesów**.
- przydzielenie i oczekiwanie zasobu - **ustalenie z góry, jakie zasoby są potrzebne** (wówczas można zastosować algorytm bankiera).
- brak możliwości odebrania zasobu - **możliwość odebrania zasobów procesowi (np. wywłaszczanie)**.
- cykl składający się z dwóch procesów - **wprowadzenie reguł, które sprawią, że proces zwolni zasób jeśli potrzebuje kolejnego**.

6.6 Inne problemy

- **Zakleszczenie komunikacyjne** to sytuacja w której dwa lub więcej procesów czeka na odpowiedź od siebie nawzajem. Rozwiązaniem problemu może być ustawienie limitu czasu (ang. timeout).
- **Uwięzienie** (ang. livelock) to sytuacja, w której co najmniej dwa procesy lub wątki, wykonują swoje zadania w taki sposób, że blokują się one nawzajem, co może uniemożliwić dokończenie zadania. Przykładowo, kiedy jeden wątek oczekuje na odpowiedź od drugiego, podczas gdy drugi wątek wykonuje obliczenia w nieskończonej pętli: w tej sytuacji wątki wzajemnie się blokują.

- **Zagłodzenie** (ang. starvation) to sytuacja, w której proces lub wątek, nie może otrzymać dostatecznie dużej liczby zasobów, by kontynuować swoje działanie, co w konsekwencji może doprowadzić do zakleszczenia lub opóźnienia. Przykładowo z zagłodzeniem mamy do czynienia, kiedy algorytm szeregowania wybiera najkrótsze zadania i krótkich zadań jest na tyle dużo, że długie zadania będą w nieskończoność odwlekane.

7 Zagrożenia i bezpieczeństwo

7.1 Rodzaje ataków

- **Tylne drzwi** (ang. backdoor) to technika polegająca na zaprojektowaniu oprogramowania lub systemu, w taki sposób, że możliwy jest dostęp do zasobów z pominięciem mechanizmów zabezpieczających tj. uwierzytelnianie. Jedynym skutecznym sposobem na zabezpieczenia, przed wprowadzeniem tylnych drzwi, są regularne przeglądy kodu, przez zespół programistów pracujący nad projektem.

- **Fałszywe logowanie** (ang. login spoofing) to technika, która ma na celu kradzież danych logowania, polegająca na podszywaniu się pod prawidłowe miejsce logowania. Jedynym skutecznym sposobem na zapobieganie takim działaniom, jest uruchamianie możliwości logowania przez kombinację klawiszy, która jest obsługiwana przez system operacyjny, a nie przez program użytkownika.

- **Fałszywy adres IP** (ang. IP spoofing) to technika, polegająca na fałszowaniu adresu IP, aby ukryć tożsamość w celu uzyskania dostępu. Przykładowe sposoby zapobiegania:

  - **filtracja ruchu** - odrzucanie pakietów, które nie pochodzą z zaufanego źródła;
  - **uwierzytelnianie** - sprawdzanie tożsamości użytkownika;
  - **monitorowanie ruchu sieciowego** - sprawdzanie, czy nie występuje podejrzany ruch sieciowy.
  - **firewall** - zapobieganie dostępowi do zasobów sieciowych.
  - **Stosowanie silnych haseł** - zabezpieczenie przed atakami typu brute force.

- **Uniemożliwienie dostępu** (ang. denial of service, skr. DoS) atak mający na celu uniemożliwienie lub utrudnienie dostępu do zasobów poprzez przepełnienie ich ruchem sieciowym. Popularne rodzaje ataków to:

  - **ping flood** - atak polegający na wysyłaniu dużej liczby pakietów ICMP
  - **SYN flood** - atak polegający na generowaniu dużej liczby połączeń TCP, co powoduje przepełnienie kolejki połączeń

  - **HTTP flood** - atak polegający na generowaniu dużej liczby zapytań HTTP

  Popularnym rodzajem ataku DoS jest **rozproszone uniemożliwienie dostępu** (ang. distributed denial of service, skr. DDoS), który jest przeprowadzany z wielu komputerów (np. zombie).

- **Atak pośrednika** (ang. man in the middle, skr. MitM) atak polegający na umiejscowieniu cyberprzestępcy pomiędzy dwiema stronami komunikującymi się w celu przechwycenia lub modyfikacji przesyłanej wiadomości. Popularne rodzaje ataków:
  - modyfikacja tablicy ARP, która przechowuje powiązania adresów IP z adresemMAC urządzeń w sieci lokalnej (ang. ARP spoofing);
  - modyfikacja konfiguracji DNS, tak aby przekierowywał żądanie na inny adres IP (ang. DNS spoofing);
  - zmuszenie użytkownika do korzystania z niezabezpieczonej strony, która kiedyś była zabezpieczona protokołem SSL/TLS (HTTPS), w taki sposób, że żądania początkowo będą wysyłane do pośrednika, zanim zostaną wysłane do właściwego serwera (ang. SSL strip);
  - stworzenie fałszywego WiFi, które nadsłuchuje ruch.

7.2 Wykorzystanie błędów w kodzie

Exploit to program lub technika wykorzystująca błędy w oprogramowaniu w celu wykonania szkodliwej czynności.

Zero-day exploit wykorzystuje błędy w kodzie, które nie są znane dla twórców oprogramowania.

Przykładowe szkody, jakie mogą wyrządzić exploity:

- otwieranie tylnych drzwi, w celu przejęcia kontroli nad komputerem;
- kradzież danych użytkownika;
- utrudnienie pracy z systemem operacyjnym;
- rozprzestrzenianie złośliwego oprogramowanie.

Atak z wykorzystaniem przepełnienia bufora to nadpisanie kolejnych komórek pamięci, które nie zostały zaalokowane do wykorzystania np. tablicy oraz mogą nadpisać obszar pamięci, który jest używany przez system operacyjny.

7.3 Złośliwe oprogramowanie

Złośliwe oprogramowanie (ang. malware) to oprogramowanie, celem jest powodowanie szkód.

- **Zombie** to komputer, który jest zdalnie kontrolowany przez cyberprzestępce.

- **Botnet** biór komputerów zombie.
- **Wirus** to rodzaj złośliwego oprogramowania, które są zdolne do reprodukcji poprzez dołączenie swojego kodu do innych programów lub systemów. Po napisaniu wirusa, autor może użyć do wstrzyknięcia go do pliku poprzez narzędzie tj. **dropper**.
- **Robak** to rodzaj złośliwego oprogramowania, które są zdolne do reprodukcji poprzez wykorzystanie błędów i niedopatrzeń w innych oprogramowaniach.
- **Koń** trojański to rodzaj złośliwego oprogramowania, które podszywa się pod pozornie nieszkodliwe oprogramowanie, które jest instalowane świadomie przez użytkownika.
- **Bomba logiczna** to rodzaj złośliwego oprogramowania, które jest uruchamiany po wykonaniu określonej czynności, np. po zalogowaniu się użytkownika, czy po uruchomienie programu. Bomby logiczne mogą być potajemnie tworzone przez programistów (np. w ramach ewentualnej zemsty za zwolnienie z pracy). Można wykorzystać wirusy, robaki, konie trojańskie do umieszczania bomblogicznych.
- **Rootkit to rodzaj złośliwego oprogramowania**, którego celem jest ukrywanie istnienia złośliwych oprogramowań przed użytkownikiem lub wykonywania szkodliwych działań na zainfekowanym komputerze. Przykładowe miejsca ukrywania rootkitów to: przestrzenie jądra systemu operacyjnego, sterowniki urządzeń, biblioteki systemowe, rejestry systemowe, aplikacje, pamięć, warstwa wizualizacji.
- **Oprogramowanie szpiegujące** (ang. spyware) to rodzaj złośliwego oprogramowania, które gromadzi dane na temat aktywności użytkownika na komputerze i przesyła zgromadzone informacje.
- **Rejestrator klawiatury** (ang. keylogger) to rodzaj złośliwego oprogramowania, które rejestruje aktywność na klawiaturze użytkownika, oraz wysyła informację na temat wciśniętych klawiszy.
- **Oprogramowanie żądające okupu** (ang. ransomware) to rodzaj złośliwego oprogramowania, które blokuje dostęp do danych znajdujących się na komputerze, a następnie wyświetla informację na temat żądania okupu w zamian za odblokowanie danych.

7.4 Zapora sieciowa

Zapora sieciowa (ang. firewall) to oprogramowanie lub urządzenie, które służy do blokowania niepowołanego dostępu do komputera lub sieci wewnętrznej, przez sieć zewnętrzną, poprzez monitoring i kontrolę ruchu sieciowego. Przykładowe rodzaje zapór sieciowych:

- **zapora sieciowa stanowa** (ang. stateful firewall) to rodzaj zapory sieciowej, która analizuje bieżący stan połączeń sieciowych, dzięki czemu skuteczniej sobie radzą z niektórymi rodzajami ataków tj. ataki polegające na ustawianiu połączeń.

- **zapora sieciowa osobista** (ang. personal firewall) to rodzaj zapory sieciowej, będąca oprogramowaniem, która chroni urządzenie (np. komputer), poprzez monitoring ruchu wychodzącego i przychodzącego oraz kontrolę aplikacji mających dostęp do sieci.

- **zapora sieciowa aplikacji** (ang. application firewall) to rodzaj zapory sieciowej, która analizuje protokoły działające na poziomie aplikacji sieciowej, np. może odpowiadać za kontrolę ruchu sieciowego na poziomie protokołu tj. HTTP, TCP, FTP.

- zapora sieciowa posiadająca zaimplementowany **system wykrywania i zapobiegania włamań** (ang. intrusion detection system, intrusion prevention system, skr. IDS/IPS), może nie tylko kontrolować nagłówki pakietów, ale też wykrywać i zapobiegać atakom sieciowym.

7.5 Programy zwalczające złośliwe oprogramowanie

Program zwalczający złośliwe oprogramowanie umożliwia zwalczenie złośliwego oprogramowania np. robaki, wirusy czy spyware.

- **Skaner antywirusowy** polega na skanowaniu plików znajdujących się na dysku w celu poszukiwania zainfekowanych plików.

- **Weryfikatory integralności** (ang. integrity checking) to narzędzie, które na początku rozpoczyna skanowanie w poszukiwaniu wirusów i po nabraniu pewności o braku wirusów, wyznacza sumę kontrolną dla każdego pliku wykonywalnego. Podczas następnego skanowania sprawdzane jest, czy suma kontrolna jest taka sama.

- **Weryfikatory zachowań** (ang. behavioral checking) to narzędzie, które na na celu monitorowanie aktywności systemu w celu wykrycia podejrzanych działań.

7.6 Inne mechanizy zabezpieczenia

- **Podpisywanie kodu** (ang. code signing) polega na stosowaniu podpisu cyfrowego do utworzonych produktów. Producent generuje klucz publiczny i prywatny oraz generuje kod, który jest zwracany przez funkcję skrótu, uruchomionej dla kodu programu. Producent podpisuje oprogramowanie kodem, poprzez zaszyfrowanie go kluczem prywatnym. Podpis jest przekazywany z oprogramowaniem. Kiedy użytkownik uruchamia program, uruchamiana jest funkcja skrótu, która zapisuje wynik. Mechanizm weryfikujący odszyfrowuje, przy pomocy klucza publicznego, podpis

dołączony do oprogramowania. Jeżeli kod zwrócony przez funkcje skrótu i kod odszyfrowanego podpisu jest taki sam, to program jest akceptowalny, w przeciwnym razie jest uznawany za sfałszowany.

- **Wtrącanie do więzienia** (ang. jailing) to mechanizm, który polega na sprawdzeniu zachowania procesu i w razie potrzeby, izolowania im dostępu do zasobów systemowych. Nowy proces jest monitorowany przez zaufany proces systemowy, który decyduje czy żądane wywołanie systemowe może zostać zrealizowane. Oznacza to, że podejrzane zachowania można wykryć i wyeliminować zanim narobią szkód w systemie.

- **Izolowanie** (ang. sandboxing) to mechanizm, który polega na uruchamianiu zadań w odizolowanym kontrolowanym środowisku. Przestrzeń adresów wirtualnych jest podzielona na obszary o tej samej wielkości, które nazywamy piaskownicami (ang. sandboxes). Piaskownice zawierają mechanizmy, które uniemożliwiają możliwość wykonania skoku do innej piaskownicy oraz pobierania z niej danych itp.

- **Wykrywanie włamań** z użyciem modeli statycznych (ang. static modelbased intrusion detection) to mechanizm wykrywania włamań na podstawie modelu. Przykładowy modelem może być graf wywołań systemowych. Można zastosować jailing w którym proces monitorujący kończy zadanie, którego zachowanie jest niezgodne z modelem.

8 Administrowanie

8.1 Rola użytkownika root w Linux

root - Użytkownik o najwyższych uprawnieniach w systemie. może przeprowadzać wszelkie zadania administracyjne, dlatego czasami jest nazywany superużytkownikiem.

- `su` - zmienia tożsamość użytkownika
- `sudo` - pozwala na wykonanie polecenia jako użytkownik root lub jako inny użytkownik (w zależności od konfiguracji)
- `sudo su` - uruchamia powłokę jako użytkownik root

8.2 Zadania cykliczne

`cron` - demon opowiedzialny za uruchomianie zadań wg określonego harmonogramu.

- `crontab` - umożliwia zarządzanie poleceniami cyklicznymi cron
- `crontab -e` - otwiera edytor, umożliwiający zmianę zawartości pliku crontab
- `crontab -l` - wyświetla listę zaplanowanych zadań.

- `crontab -r` - usuwa plik crontab (usuwa zadania).
- `crontab -u` - lmielewczyk -l wyświetla listę zaplanowanych zadań użytkownika lmielewczyk.

Każdy wiersz pliku crontab zawiera komentarze poprzedzone znakiem #, albo wiersz odpowiadający cyklicznemu poleceniu w formacie:
`minuta godzina dzień_miesiąca miesiąc dzień_tygodnia polecenie`

8.3 Pliki

- **/etc/sudoers** - zawiera definicję uprawnień w stosunku do polecenia sudo w systemach Unix/Linux.
- **/etc/passwd** - zawiera listę użytkowników, którzy są rozpoznawani przez Linuxa. Podczas logowania system pobiera dane z pliku, aby określić m.in. UID, czy katalog domowy użytkownika.
- **/etc/group** - zawiera informację na temat grup, które są dostępne w systemie
- **/etc/shadow** - zawiera zaszyfrowane hasła i inne informacje odnośnie zarządzania hasłami. Plik jest chroniony przed programami do łamania haseł i może być odczytywany jedynie przez użytkownika root

8.4 Polecenia

- `chfn` - zmienia informacje o użytkowniku.
- `chown` - zmienia właściciela i przynależność grupy plików i katalogów.
- `chage` - ustawia ważność hasła użytkownika.
- `chsh` - pozwala zmienić domyślną powłokę.
- `useradd` - dodaje nowe konto użytkownika.
- `usermod` - modyfikuje atrybuty konta użytkownika.
- `userdel` - usuwa konto użytkownika.
- `adduser` - dodaje nowego użytkownika.
- `deluser` - usuwa użytkownika.
- `groupadd, addgroup` - dodaje nową grupę.
- `groupdel, delgroup` - usuwa grupę.
- `groupmod` - modyfikuje atrybuty grupy.

8.5 Moduł uwierzytelniania PAM

Moduł uwierzytelniania **PAM** (ang. pluggable authentication modules) to modularny system umożliwiający kontrolę uwierzytelniania w systemach operacyjnych Unix/Linux. Pliki konfiguracyjne PAM znajdują się w katalogu `/etc/pam.d/`. Każda usługa ma własny plik konfiguracyjny, np. `/etc/pam.d/login`, `/etc/pam.d/passwd`, `/etc/pam.d/su`.

8.6 Funkcja pojedynczego logowania

Funkcja pojedynczego logowania (skr. SSO, ang. single sign-on) pozwala na uzyskanie dostępu do dowolnego systemu, usługi lub aplikacji, przy pomocy tego samego loginu i hasła. SSO zazwyczaj zawiera informacje tj. nazwa użytkownika, (zaszyfrowane) hasło, identyfikator użytkownika (UID), adres email.

Podstawowe elementy SSO:

1. **Scentralizowany magazyn katalogów** - miejsce, gdzie przechowywane są informacje na temat użytkowników, takie jak dane uwierzytelniające, uprawnienia i atrybuty. Przykładowe scentralizowane magazyny katalogów:
   - LDAP (ang. lightweight directory access protocol) (np. OpenLDAP)
   - Active Directory
2. **Narzędzie do zarządzania danymi użytkowników** - narzędzie umożliwiające administrację użytkownikami, ich danymi i uprawnieniami. Przykładowe narzędzia do zarządzania danymi użytkowników
   - Apache Directory Studio (dla LDAP)
   - Active Directory Users and Computers (dla Active Directory)
3. **Mechanizm uwierzytelniania** - mechanizm uwierzytelniania potwierdzają tożsamość użytkownika na podstawie dostarczonych danych, tj. login i hasło. Przykładowe mechanizmy uwierzytelniania:
   - Kerbersos (uwierzytelnianie oparte na biletach)
   - SSSD (ang. system security services daemon) (demon pośredniczący między PAM a źródłami danych tj. LDAP, Active Directory)
   - LDAP (bezposrednio)
4. **Scentralizowane funkcje sprawdzające atrybuty użytkownika** - funkcje odpowiadające za weryfikację atrybutów użytkownika, takich jak uprawnienia czy role. przykładowe scentralizowane funkcje sprawdzające atrybuty użytkownika:
   - w systemach Unix/Linux plik /etc/nsswitch.conf zawiera konfiguracje źródeł danych
5. **Protokoły komunikacyjne** - protokoły służące do komunikacji między aplikacjami a serwerem uwierzytelniającym.

8.7 Rejestrowanie dzienników w Linux

System **syslog** to narzędzie, którego celem jest tworzenie plików dziennika oraz daje możliwość konfiguracji rejestrowania zdarzeń. Większość plików dziennika systemu jest zazwyczaj przechowywane w katalogu `/var/log`. System syslog korzysta z konfiguracji znajdującej się w pliku `/etc/rsyslog.conf`. System syslog zarządzany jest przez demona `rsyslogd`, albo przez demona `syslogd`. Syslog umożliwia sortowanie komunikatów ze względu na ich źródło (usługi) i poziom ważności oraz kierowanie ich do różnych miejsc tj. pliki dzienników, pliki terminali, czy inne komputery. Przykładowe usługi systemu syslog: `authenctication, cron, daemon, kernel, mail, mark, syslog, user`.

System **systemd** to narzędzie służące do przechowywania plików dziennika. System systemd przechowuje dzienniki w plikach binarnych, w większości przypadków są przechowywane w folderze `/var/log/journald`. Systemd korzysta z konfiguracji znajdującej się w pliku: `/etc/systemd/journald.conf`, przy czym własne konfiguracje można wprowadzać do folderu: `/etc/systemd/journald.conf.d`. System systemd zarządzany jest przez demona `systemd-journald`.

Systemy syslog i systemd to narzędzia, które są powszechnie stosowane. Oba systemy dostarczają istotne funkcje, które sprawiają, że mogą ze sobą współpracować (np. syslog może odbierać komunikat z różnych wtyczek wejściowych).

Inne narzędzia:

- **logrotate** - umożliwia zarządzanie dziennikami w Linuxie (tj. automatyczne przenoszenie, usuwanie, kompresja)
- **ELK Stack** - połączenie narzędzi:
  - **elasticsearch** - posiada funkcje przeszukiwania, analizy i przechowywania dużej ilości danych,
  - **logstash** - służy do przetwarzania, przekształcania i przesyłania logów z różnych źródeł do elasticsearch,
  - **kibana** - dostarcza interfejs webowy do eksploracji, analizy i wizualizacji danych przechowywanych w elasticsearch

  (dobry wybór jeśli priorytetem jest wszechstronność tzn. obsługa różnych formatów).
- **Graylog** - posiada jeden główny serwer i opcjonalną bazę danych (dobry wybór jeśli priorytetem jest monitorowanie logów lub łatwa konfiguracja).
- **Apache Druid** - system bazodanowy, który służy do przetwarzania i analizy dużej ilości danych (dobry wybór jeśli priorytetem jest szybka analiza danych)

8.8 Narzędzia monitorujące

- **collectd** - umożliwia zbieranie danych odnośnie wydajności systemu. Narzędzie collectd tworzy demona collectd i korzysta z konfiguracji w pliku: `/etc/collectd/collectd.conf`. Collectd zbiera różne dane dotyczące wydajności systemu, takie jak tj. zużycie CPU, obciążenie systemu, ilość dostępnej pamięci, statystyki dysków, informacje o sieci, temperatury itd. Dane te mogą posłużyć do generowania raportów, analizy i monitorowania systemu.
- **Nagios, Icinga** - popularne narzędzia monitorujące systemy, sieci, aplikacje itp. Narzędzia Nagios i Icinga posiadają interfejs WWW umożliwiające wygodne zarządzanie oraz mechanizmy umożliwiające tworzenie raportów i archiwizacji monitorowanych danych. Narzędzia Nagios i Icinga mogą powiadamiać administratorów o awariach i innych istotnych zdarzeniach za pomocą różnych kanałów tj. email, SMS itp. Nagios i Icinga mogą korzystać z danych zbieranych przez collectd, następnie przekazując je do narzędzi wizualizacji danych.

8.9 journalctl i systemctl

- `journalctl` - przeszukuje i analizuje logi wygenerowane przez system systemd.
- `journalctl -u ssh` - wyświetla dzienniki wygenerowane przez ssh.
- `journalctl -n 100 -D /var/log/journal/2000/` - wyświetla ostatnie 100 wersów z dziennika w katalogu.
- `journalctl -since yesterday -until now` - wyświetla komunikaty od wczoraj do chwili obecnej.
- `systemctl` - umożliwia zarządzanie usługami systemowymi
- `sudo systemctl start mysql` - uruchomienie usługi mysql.
- `sudo systemctl stop mysql` - zatrzymanie usługi mysql.
- `sudo systemctl restart mysql` - restart usługi mysql.
- `sudo systemctl status mysql` - sprawdzenie statusu usługi mysql
- `sudo systemctl enable mysql` - włączenie usługi mysql przy starcie systemu.
- `sudo systemctl disable mysql` - wyłączenie usługi mysql przy starcie systemu.

9 Wirtualizacja

9.1 Podstawowa wirtualizacja i konteneryzacja

Wirtualizacja to technologia umożliwiająca tworzenie wirtualnych wersji zasobów sprzętowych lub oprogramowania. Wirtualizacja umożliwia **uruchomienie wielu instancji systemów operacyjnych na tym samym sprzęcie fizycznym**. Oprogramowanie "wirtualizacyjne" w sposób dynamiczny **przydziela zasoby procesora, pamięci oraz operacje wejścia-wyjścia "gościnnym" systemom operacyjnym oraz rozwiązuje konflikty związane z przydzielaniem zasobów**. Powszechne stosowane są wirtualne serwery, które działają wirtualnie na jednym fizycznym serwerze, co pozwala na efektywne wykorzystanie zasobów. **Wirtualizacja serwerów umożliwia efektywne obsługiwane wielu użytkowników**, dzięki czemu sprzęt działa wydajniej, ponadto **technologia wirtualizacji stanowi podstawę dla chmury obliczeniowej i kontenerów**.

Konteneryzacja to metoda wirtualizacji, umożliwiająca uruchomienie i izolację aplikacji oraz ich zależności w środowisku kontenerowym. Konteneryzacja nie korzysta z hipernadzorcy, zamiast tego korzysta z funkcji jądra gospodarza, które pozwalają na izolowanie procesu od reszty systemu. **W kontenerze każdy proces posiada swoją własną przestrzeń nazw systemu plików, przestrzeń nazw PID (Process ID) oraz inne przestrzenie nazw, co pozwala na izolację zadań między kontenerami**. Kontenery **współdzielą to samo jądro systemu operacyjnego z systemem gospodarza**. Kontenerów powszechnie używa się w połączeniu z maszynami wirtualnymi - maszyny wirtualne można wykorzystać do podziału serwerów fizycznych, na bloki, natomiast w kontenerach można uruchomić aplikacje na wirtualnych maszynach.

9.2 Hipernadzorca

Hipernadzorca (ang. hypervisor) to warstwa programowa, która pośredniczy pomiędzy maszynami wirtualnymi, a sprzętem, na którym działają. Rola hipernadzorcy polega na **alokowaniu i zarządzaniu zasobami sprzętowymi tak, aby różne maszyny wirtualne mogły współistnieć**. Hipernadzorcy mogą być klasyfikowane jako:

- **Hipernadzorca typu 1** - (ang. bare-metal) – działa bezpośrednio na sprzęcie fizycznym, nie wymagają dodatkowego systemu operacyjnego. VMware ESXi, XenServer (moduły na poziomie sprzętu fizycznego), KVM (ang. kernel-based virtual machine) (moduł jądra Linux), Bhyve (moduł jądra FreeBSD).
- **Hipernadzorca typu 2** - (ang. hosted) – działa na poziomie systemu operacyjnego gospodarza. Oracle VirtualBox, QEMU (ang. Quick Emulator) (działają w różnych systemach operacyjnych), VMware Workstation

(działa w Linux i Windows), Hyper-V (zawarty w Windows), Parallels Desktop (działa w MacOS).

9.3 Techniki wirtualizacji

- **pełna wirtualizacja** - hipernadzorca emuluje sprzęt, definiując wirtualne odpowiedniki sprzętu, na których będą uruchamiane maszyny wirtualne (systemy operacyjne gościa nie wiedzą, że są w stanie wirtualnym).
- **parawirtualizacja** - systemy operacyjne gościa są modyfikowane, tak aby wykryły swój stan wirtualny i mogły współpracować z hipernadzorcą.
- **wirtualizacja wspomagana sprzętowo** - wirtualizacja jest wspomagana przez specjalne funkcje procesora (np. Intel VT, AMD-V) – procesor i kontroler pamięci są wirtualizowane przez sprzęt oraz są pod kontrolą hipernadzorcy

9.4 Inne pojęcia

- **Sterowniki parawirtualne** - zmodyfikowane sterowniki urządzeń w systemie operacyjnym gościa, które są dostosowane do współpracy z hipernadzorcą.
- **Migracja w locie** (ang. live migration) - technika polegająca na przeniesieniu maszyny wirtualnej, bez przerywania jej działania, z jednego fizycznego hosta do innego, który stanie się jej hostem docelowym. Hipernadzorca kopiuje zmiany z jednego hosta do drugiego oraz kończy działanie migracji, gdy pamięć tych dwóch hostów stanie się identyczna. Technika jest stosowana np. do równoważenia obciążeń, utrzymywania dostępności (np. po awarii sprzętu), optymalizacji wykorzystania zasobów.
- **Obraz maszyny wirtualnej** - szablony systemów operacyjnych, które hipernadzorca może załadować i uruchomić. Obrazy są dostosowane do konkretnego hipernadzorcy, ale dostępne są narzędzia umożliwiające przenoszenia obrazów między hipernadzorcami. Serwery wirtualne tworzy się z obrazów odpowiednio skonfigurowanych systemów operacyjnych.

10 Podstawowe elementy Linuxa i Windowsa

10.1 Licencja GNU i licencja jądra Windows

Linux jest udostępniany z **licencją publiczną GNU**, która opisuje zasady jego stosowania. Użytkownicy mogą **za darmo używać, kopiować, modyfikować i rozpowszechniać kod źródłowy i wersję binarną kodu Linuxa**. Ograniczeniem jest **brak możliwości udostępnienia produktów w formie binarnej, które są tworzone na bazie jądra systemu Linux** (może być dostarczony wraz z produktem, albo na żądanie).

Jądro Windowsa jest **własnościowe**, co oznacza, że jest produktem firmy Microsoft i **jego kod źródłowy nie jest publicznie dostępny**. Microsoft chroni prawa autorskie i własność intelektualną związane z jądrem Windows. Przykładowe punkty, które dotyczą licencji jądra Windowsa:

1. **kod źródłowy jądra nie jest publicznie dostępny** (Microsoft utrzymuje jądro jako zamkniętą technologię i dostęp do jego kodu jest ograniczony).
2. **jądro Windowsa jest objęte prawami autorskimi i innymi prawami własności intelektualnej firmy Microsoft** (kopiowanie, modyfikowanie lub dystrybucja bez zgody Microsoftu jest nielegalna).
3. **dostęp do jądra Windowsa uzyskuje się poprzez zakup licencji na system operacyjny Windows**.
4. **Microsoft oferuje narzędzia programistyczne, tj. Windows Driver Kit, czy Windows Hardware Lab Kit, które pozwalają deweloperom na tworzenie sterowników i oprogramowania, które działa z jądrem Windowsa**.

10.2 Standard POSIX

Standard POSIX (ang. portable operating system interface) opisuje interfejs biblioteki.

- **POSIX nie zawiera bezpośrednio wywołań systemowych**, tylko określa, jakie funkcje biblioteczne powinny być dostępne w systemie, który jest zgodny ze standardem oraz to jakie parametry powinny obsługiwać i jakie wyniki powinny zwracać.
- **Standard obejmuje różne aspekty systemu operacyjnego**, tj. operacje na plikach, zarządzanie procesami, komunikacja międzyprocesowa, obsługa sygnałów itp. Przykładowo, funkcje tj. read, write, fork są opisane w standardzie POSIX jako część interfejsu programistycznego.
- **Standard POSIX jest zbiorem norm**, a implementacje systemów operacyjnych, które chcą być zgodne z POSIX, muszą spełnić wymagania określone w tych normach.

10.3 Ładowalne moduły Linuxa

Ładowalne moduły to dynamicznie ładowane rozszerzenia jądra, tzn. są to fragmenty kodu, które można dynamicznie dołączać lub usuwać z jądra systemu operacyjnego w trakcie jego działania, bez konieczności ponownego uruchamiania całego systemu. Przykładowe zastosowania ładowalnych modułów:

- **dostarczanie sterowników urządzeń**;

- **dostarczenie funkcjonalności obsługującą nowe systemy plików**;
- **dostarczenie funkcjonalności obsługującą nowe protokoły sieciowe**;

Przykłady obsługi modułów:

- `sudo insmod my_module.ko` - załadowanie modułu.
- `sudo rmmod my_module` - usunięcie modułu.
- `sudo lsmod` - wyświetlenie listy załadowanych modułów.

10.4 System32 (Windows)

Katalog System32 w systemach Windows zawiera ważne pliki systemowe i biblioteki DLL (ang. dynamic link libraries), które są niezbędne do prawidłowego działania systemu

Przykładowe pliki, które znajdują się w katalogu:

- **pliki wykonywalne** (z rozszerzeniem *.exe), które są używane przez system operacyjny;
- **biblioteki DLL, które zawierają kod, który może być używany przez wiele aplikacji**, co pozwala na bardziej efektywne zarządzanie zasobami;
- **sterowniki systemowe**, które są potrzebne do obsługi komponentów sprzętowych;
- **niektóre pliki konfiguracyjne i systemowe**;
- **programy systemowe** tj. narzędzia administracyjne.

Modyfikacje lub usuwanie plików w tym katalogu może prowadzić do niestabilności systemu.

10.5 NTOS

NTOS (ang. new technology operating system) - centralna wartość jądra systemowego, wczytywana z pliku ntoskrnl.exe, stosowanego w systemach operacyjnych rodziny Windows NT, tj. Windows NT 4.0, Windows 2000, Windows XP, Windows 7, Windows 8, Windows 10, Windows 11 i ich odpowiedniki serwerowe.

10.6 Środowiska kompatybilności w Windows

Interix było środowiskiem kompatybilności dla systemu Windows, które umożliwiało uruchamianie aplikacji i poleceń przeznaczonych dla systemów opartych na UNIX oraz dostarczało narzędzia programistyczne i skompilowane biblioteki,

które umożliwiały programistom korzystanie z funkcji systemowych i interfejsu POSIX na platformie Windows.

Windows Subsystem for Linux (skr. WSL) umożliwia uruchamianie dystrybucji Linux bezpośrednio na systemie Windows, która działa jako warstwa kompatybilności, pozwalająca na uruchamianie programów i poleceń przeznaczonych dla środowisk Linux na platformie Windows. WSL oferuje pełne środowisko Linux z jądrem systemu operacyjnego.

10.7 Interfejsy programowania NT API i Win32 API

NT API dostarcza niskopoziomowe funkcje jądra systemu operacyjnego Windows NT, które obejmują operacje na niższym poziomie, tj. zarządzanie pamięcią, obsługa plików, operacje na procesach, operacje na rejestrze, obsługa przerwań itp. **Win32 API** dostarcza wyższopoziomowe funkcje bardziej zorientowane na interfejs użytkownika, które obejmują funkcje do tworzenia aplikacji okienkowych, obsługi zdarzeń, komunikacji międzyprocesowej, operacji plikowych i graficznych itp.

10.8 Struktura rejestru systemu Windows

Rejestr systemu (ang. system registry) jest kluczowym elementem systemu operacyjnego Windows, który gromadzi i zarządza informacjami odnośnie konfiguracji i działania systemu.

Rejestr podzielony jest na tzw. gałęzie (ang. hives), które określają hierarchiczną strukturę organizacji kluczy i wartości w rejestrze

10.9 Menadżer zadań w Windows

Menedżer zadań (ang. task manager) to narzędzie dostępne w Windows, które umożliwia monitorowanie działania systemu i zarządzanie zadaniami. Uruchomienie: `Ctrl+Shift+Esc lub Ctrl+Alt+Delete`. Przykładowe funkcjonalności:

- **Procesy**
- **Wydajność**
- **Aplikacje autostartu**
- **Użytkownicy**
- **Usługi**
- **Szczegóły**

10.10 msconfig

Narzędzie msconfig jest wbudowane w Windows i może być używane do zarządzania konfiguracją rozruchu systemu oraz konfiguracji ogólnej systemu. Przykładowe funkcjonalności:

- **Ogólne**
- **Rozruch**
- **Narzędzia**