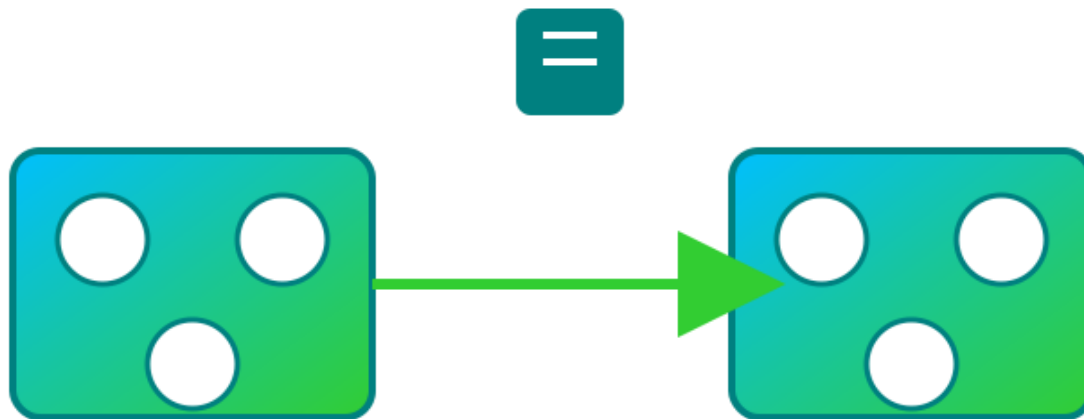


Procesy, wątki i wielozadaniowość.



Cechy systemu operacyjnego

Jednozadaniowość

System wykonuje tylko jedno zadanie w danym momencie.

Kolejne zadanie rozpoczyna się dopiero po zakończeniu poprzedniego.

Przykład: starsze systemy DOS.

Wielozadaniowość

System umożliwia wykonywanie wielu procesów „równocześnie”.

W praktyce na jednym procesorze realizowane jest to poprzez przełączanie kontekstu w krótkich przedziałach czasu (time slicing), co daje wrażenie równoległości.

Jeśli system ma wiele rdzeni/procesorów, część zadań może być faktycznie wykonywana równolegle.

Za planowanie i przełączanie zadań odpowiada jądro systemu operacyjnego (kernel), które zarządza m.in. harmonogramem pracy procesora, komunikacją między procesami i ich priorytetami.



Planista i wielozadaniowość

Planista (scheduler)

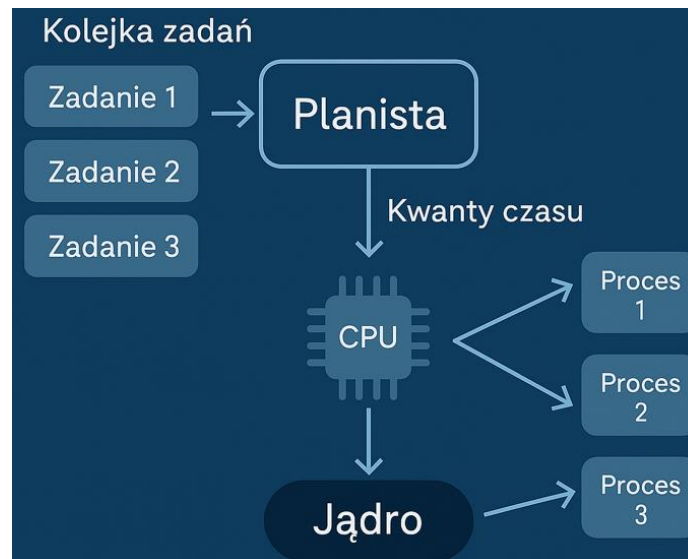
To część jądra systemu operacyjnego odpowiedzialna za realizację algorytmu szeregowania zadań (ang. scheduling). Decyduje, które procesy i w jakiej kolejności otrzymają czas procesora.

Pozorna równoczesność

Jeżeli liczba procesów jest większa niż liczba dostępnych procesorów/rdzeni, system stosuje przełączanie kontekstu w krótkich przedziałach czasu, co daje wrażenie równoległości.

Dzielenie czasu (time sharing)

To podstawowy mechanizm wielozadaniowości w systemach jednoprocessorowych – procesor przydziela każdemu zadaniu kwant czasu, po czym przełącza się na kolejny.

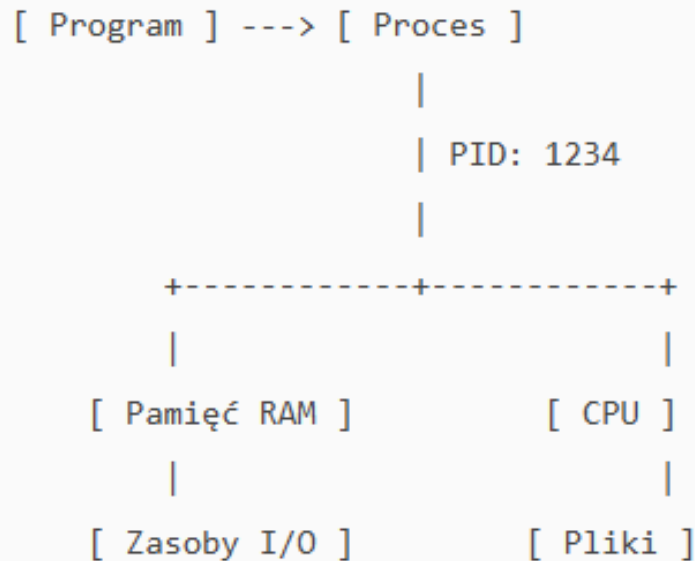


Proces

Proces

- to egzemplarz wykonywanego programu wraz z przydzielonymi przez system operacyjny zasobami, niezbędnymi do jego funkcjonowania.

Każdy nowo powstały proces otrzymuje unikatowy numer PID (process identifier), który go jednoznacznie identyfikuje.



Proces

Proces to działający program wraz z zasobami, które zostały mu przydzielone przez system operacyjny.

Składa się z:

Kod programu (instrukcje do wykonania)

Czas procesora (przydzielany przez planistę w kwantach czasu)

Przydzielona pamięć (obszar w RAM – kod, dane, stos, sarta)

Zestaw danych (zmienne, pliki, dane wejściowe/wyjściowe)

Dostęp do urządzeń wejścia/wyjścia (np. klawiatura, dysk, sieć)

Dodatkowo proces ma identyfikator (PID), stan (np. gotowy, wykonywany, oczekujący) i informacje o priorytecie.



Struktura hierarchiczna procesów

Struktura hierarchiczna procesów

- Uruchomienie jednego programu (np. przeglądarki) powoduje utworzenie procesu głównego (rodzica), który może tworzyć procesy potomne (np. obsługa kart, wtyczek, renderowanie).
- Procesy potomne mogą tworzyć kolejne procesy, co tworzy drzewo procesów.

Przykład:

W systemie Linux polecenie **ps -ef --forest** pokazuje hierarchię procesów w formie drzewa lub **htop**.

W Windows **ps** podobnie widać w **Menedżerze zadań** lub narzędziu **Process Explorer**.

```
4077    2826  \_ /usr/libexec/gnome-terminal-server
4127    4077    \_ bash
4199    4127      | \_ sudo -s
4200    4199      |    \_ sudo -s
4201    4200      |        \_ /bin/bash
4756    4201      |            \_ htop
4769    4077    \_ bash
4776    4769      \_ sudo -s
4777    4776      \_ sudo -s
4778    4777      \_ /bin/bash
5112    4778      \_ ps -eo pid,ppid,cmd --forest
root@ubuntu2204:/home/ubuntu# ps -eo pid,ppid,cmd --forest > procesy.txt
```

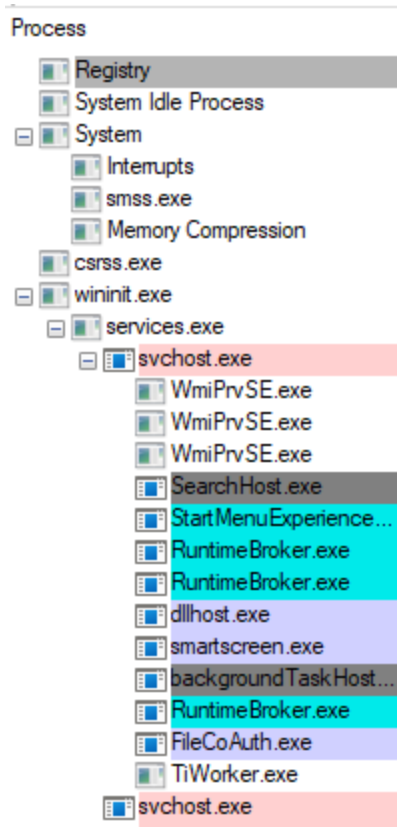
PS C:\WINDOWS\system32> ps

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
139	9	2228	11020	0,19	5268	0	AggregatorHost
215	16	8772	22252	0,09	4604	2	chrome
325	27	15952	50464	0,73	4988	2	chrome
1423	56	42264	139440	7,69	5012	2	chrome
169	10	2092	8592	0,03	6016	2	chrome
246	20	16028	42680	0,16	6532	2	chrome
221	19	13000	28756	0,17	7584	2	chrome

Struktura hierarchiczna procesów

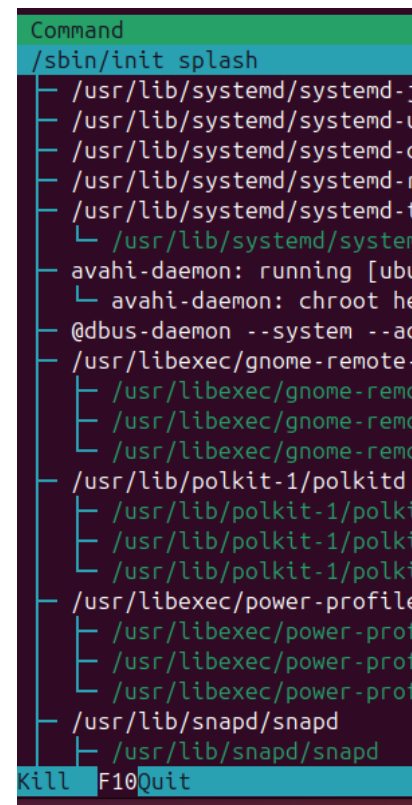
Struktura hierarchiczna procesów

- Uruchomienie jednego programu (np. przeglądarki) powoduje utworzenie procesu głównego (rodzica), który może tworzyć procesy potomne (np. obsługa kart, wtyczek, renderowanie).
- Procesy potomne mogą tworzyć kolejne procesy, co tworzy drzewo procesów.



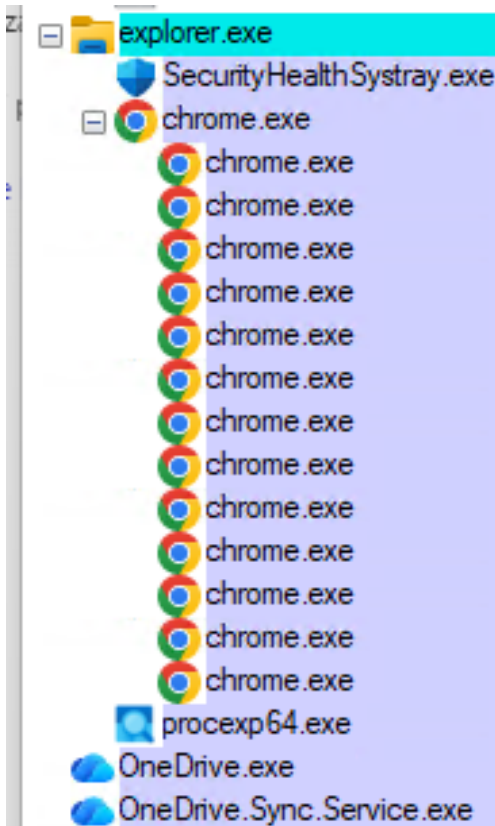
• *procesy nadrzędne*

• *procesy potomne*

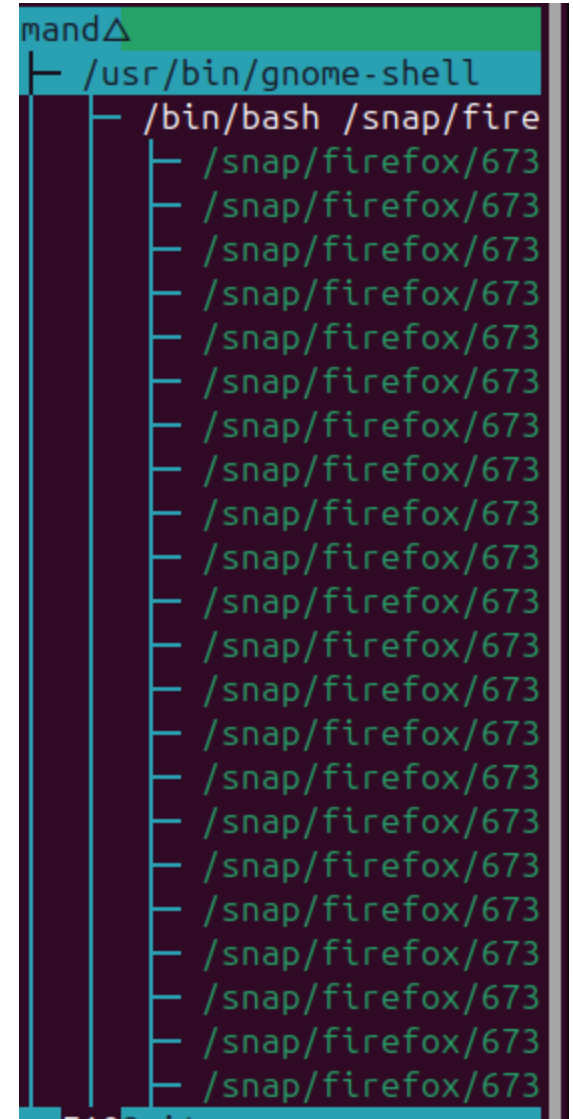


Hierarchiczna struktura procesów - drzewo

- Hierarchiczna struktura procesów - drzewo



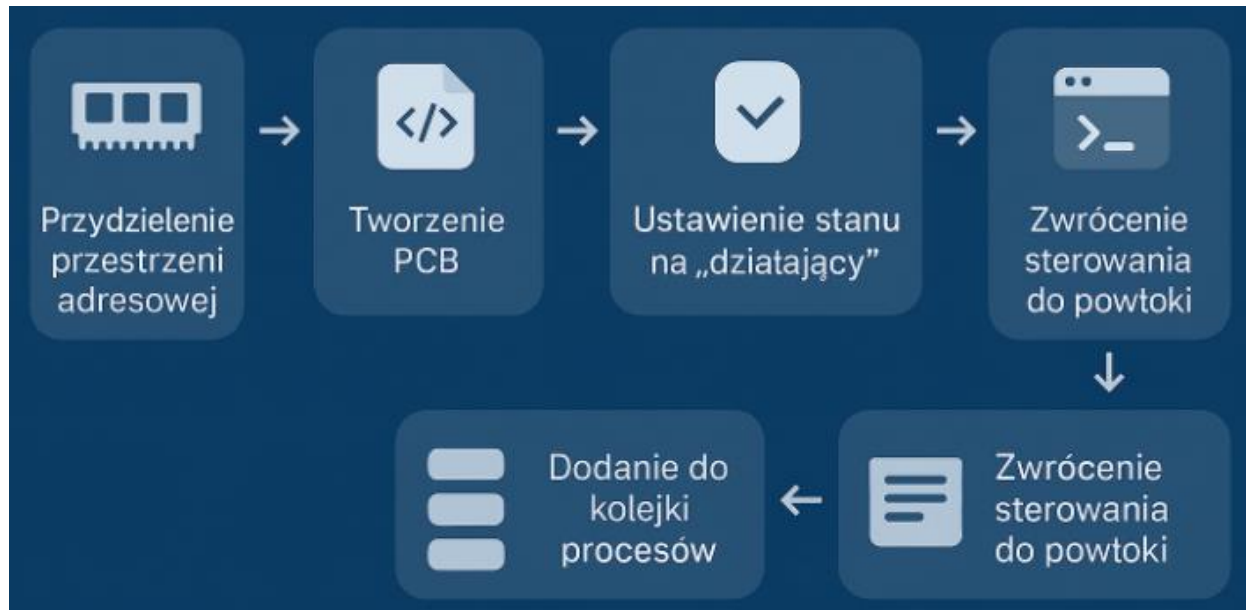
Procesy równorzędne



Uruchomienie procesu

Uruchomienie procesu SO:

- **Przydzielenie przestrzeni adresowej** – system rezerwuje pamięć dla procesu.
- **Wypełnienie struktury opisującej proces** – tworzenie PCB (Process Control Block).
- **Kopiowanie kodu i danych** – ładowanie programu do pamięci.
- **Ustawienie stanu na „działający”** – zmiana statusu w PCB.
- **Dodanie do kolejki procesów** – planista ustala priorytet.
- **Zwrócenie sterowania do powłoki** – użytkownik może kontynuować pracę.



Uruchomienie procesu

Tabela 1 – Windows 11 (PowerShell)

Etap tworzenia procesu	Polecenie PowerShell
Uruchomienie procesu	<code>Start-Process notepad.exe</code>
Pobranie PID	<code>\$p = Start-Process notepad -PassThru; \$p.Id</code>
Informacje o procesie (PCB)	<code>`Get-Process -Id \$p.Id</code>
Ścieżka i moduły (ładowanie kodu)	<code>`(Get-Process -Id \$p.Id).Modules</code>
Status procesu	<code>`Get-Process -Id \$p.Id</code>
Priorytet procesu	<code>(Get-Process -Id \$p.Id).PriorityClass</code>
Zmiana priorytetu	<code>\$proc = Get-Process -Id \$p.Id; \$proc.PriorityClass = 'AboveNormal'</code>
Sprawdzenie czy działa	<code>try { Get-Process -Id \$p.Id } catch { 'Proces nie działa' }</code>

Aby zobaczyć listę modułów w czytelnej formie, użyj:

`(Get-Process -Id $p.Id).Modules | Select-Object ModuleName, FileName`

Właściwość jest pusta, bo proces nie ma dodatkowych modułów poza podstawowymi.

Uruchomienie procesu

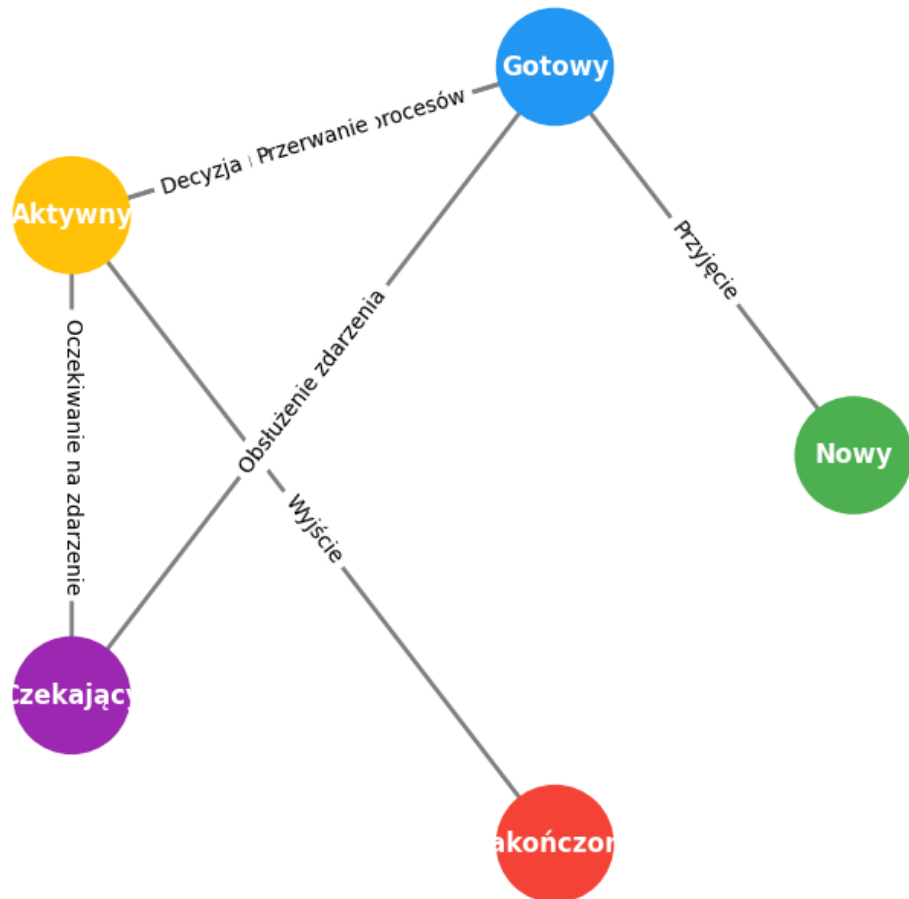
Tabela 2 – Ubuntu 24.04 (bash)

Etap tworzenia procesu	Polecenie Linux
Uruchomienie procesu w tle	<code>nano /etc/hosts &</code>
Pobranie PID	<code>echo \$! # PID procesu nano</code>
Informacje o procesie (PCB)	<code>ps -p <PID> -o pid,ppid,state,pri,cmd</code>
Mapowanie pamięci (ładowanie kodu)	<code>cat /proc/<PID>/maps</code>
Status procesu	<code>ps -p <PID> -o state</code>
Priorytet i polityka planisty	<code>ps -p <PID> -o pid,pri,policy</code>
Szczegóły polityki RT	<code>chrt -p <PID></code>

Wykonanie procesu może przyjmować kilka stanów:

Wykonanie procesu może przyjmować kilka stanów:

- aktualnie wykonywany przez procesor,
- czekający na udostępnienie przez system operacyjny zasobów,
- uśpiony
- przeznaczony do zniszczenia,
- proces zombie,
- właśnie tworzony itd.



Wykonanie procesu może przyjmować kilka stanów:

Tabela 1 – Stany procesu w Linux (Ubuntu 24.04)

Stan	Opis	Polecenie
R – Running	Proces aktualnie wykonywany przez CPU	<code>ps -p <PID> -o state,cmd</code>
S – Sleeping	Proces uśpiony, czeka na zdarzenie	<code>ps -p <PID> -o state,cmd</code>
D – Uninterruptible	Czeka na I/O, nie można przerwać	<code>ps -p <PID> -o state,cmd</code>
T – Stopped	Zatrzymany (np. sygnał STOP)	<code>ps -p <PID> -o state,cmd</code>
Z – Zombie	Proces zakończony, PCB istnieje	<code>`ps aux</code>
I – Idle	Bezczynny (wątek jądra)	<code>ps -p <PID> -o state,cmd</code>

Wykonanie procesu może przyjmować kilka stanów:

Tabela 2 – Stany procesu w Windows 11 (PowerShell)

Stan	Opis	Polecenie
Running	Proces działa	<code>Get-Process -Id <PID></code>
Suspended / Blocked	Proces wstrzymany lub czeka na zasób	<code>`Get-CimInstance Win32_Process</code>
Terminated	Proces zakończony	Brak w <code>Get-Process</code> (zwraca błąd)
Zombie	Brak w Windows – proces usuwany natychmiast	—
Tworzony	Bardzo krótki stan, niewidoczny w PowerShell	—

Pole `ExecutionState` w WMI/CIM może przyjmować:

- 3 – Running
- 4 – Blocked
- 5 – Suspended
- 6 – Terminated

Kończenie programu – proces zwraca SO kod zakończenia:

Kończenie programu – proces zwraca SO kod zakończenia

Etapy kończenia procesu:

1. Proces **zwraca kod zakończenia** do systemu operacyjnego (np. exit code – informacja o powodzeniu lub błędzie).
2. System operacyjny **ustawia stan procesu na „zakończony”** (terminated) lub „do usunięcia” (zombie, jeśli czeka na odczyt kodu zakończenia przez proces macierzysty).
3. Zwalnianie zasobów:
 - pamięci RAM przydzielonej procesowi,
 - uchwytów do plików,
 - urządzeń wejścia/wyjścia.
4. Kończenie procesów potomnych (jeśli istnieją).
5. Usunięcie przestrzeni adresowej procesu.
6. Usunięcie procesu z kolejki procesów i aktualizacja harmonogramu.
7. Przydzielenie procesora innemu procesowi (przełączenie kontekstu).



Kończenie programu – proces zwraca SO kod zakończenia:

Tabela 1 – Windows 11 (PowerShell)

Etap kończenia procesu	Polecenie PowerShell
Sprawdzenie kodu zakończenia	<code>\$p = Start-Process notepad -PassThru</code> <code>\$p.WaitForExit()</code> <code>\$p.ExitCode</code>
Stan „zakończony”	<code>Get-Process -Id <PID> → błąd (proces nie istnieje)</code>
Zwalnianie zasobów	Automatyczne – można monitorować pamięć: <code>`Get-Process</code>
Procesy potomne	<code>`Get-CimInstance Win32_Process</code>
Usunięcie z kolejki / przełączenie kontekstu	Automatyczne przez system – brak bezpośredniego polecenia

Kończenie programu – proces zwraca SO kod zakończenia:

Tabela 2 – Linux (Ubuntu 24.04)

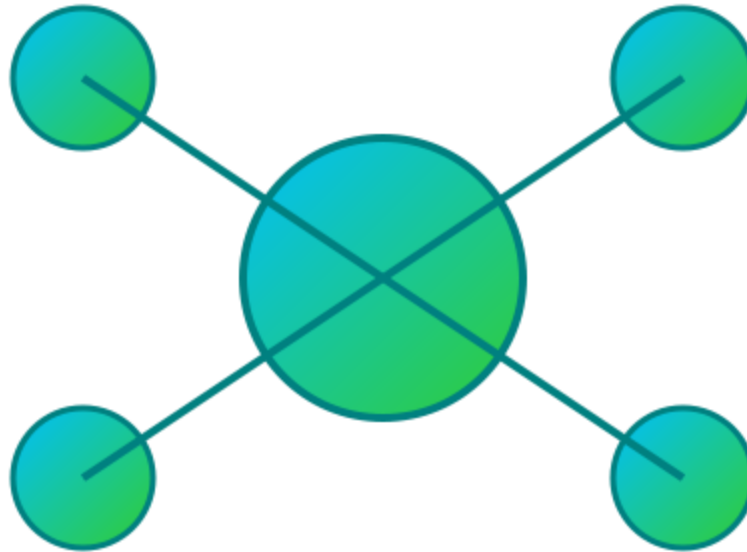
Etap kończenia procesu	Polecenie Linux
Sprawdzenie kodu zakończenia	<pre>sleep 5 & PID=\$! wait \$PID echo \$?</pre>
Stan „zakończony” / „zombie”	<pre>ps -p <PID> -o pid,state,cmd (stan Z = zombie)</pre>
Zwalnianie zasobów	Po wait katalog /proc/<PID> znika
Procesy potomne	<pre>pstree -p</pre>
Usunięcie z kolejki / przełączenie kontekstu	Widoczne w top / htop – proces znika, CPU przydzielane innym

Wielowątkowość

Wielowątkowość cecha, dzięki której w ramach jednego procesu system może wykonywać kilka zadań.

Nowe zadania to kolejne ciągi instrukcji wykonywane oddzielnie.

- Wątek (ang.thread) - część programu wykonywana współbieżnie w obrębie jednego procesu, w jednym procesie może istnieć wiele wątków.
- Wszystkie wątki działające w danym procesie współdzielą przestrzeń adresową oraz wszystkie inne struktury systemowe.
- Procesy posiadają niezależne zasoby.



Wielowątkowość

Tabela 1 – Windows 11 (PowerShell)

Czynność	Polecenie PowerShell
Uruchomienie procesu	<code>\$p = Start-Process notepad -PassThru</code>
PID procesu	<code>\$p.Id</code>
Lista wątków w procesie	<code>`(Get-Process -Id \$p.Id).Threads</code>
Liczba wątków	<code>(Get-Process -Id \$p.Id).Threads.Count</code>
Zasoby procesu (pamięć, uchwyt)	<code>`Get-Process -Id \$p.Id</code>
Ładowane moduły (DLL)	<code>`(Get-Process -Id \$p.Id).Modules</code>

Wielowątkowość

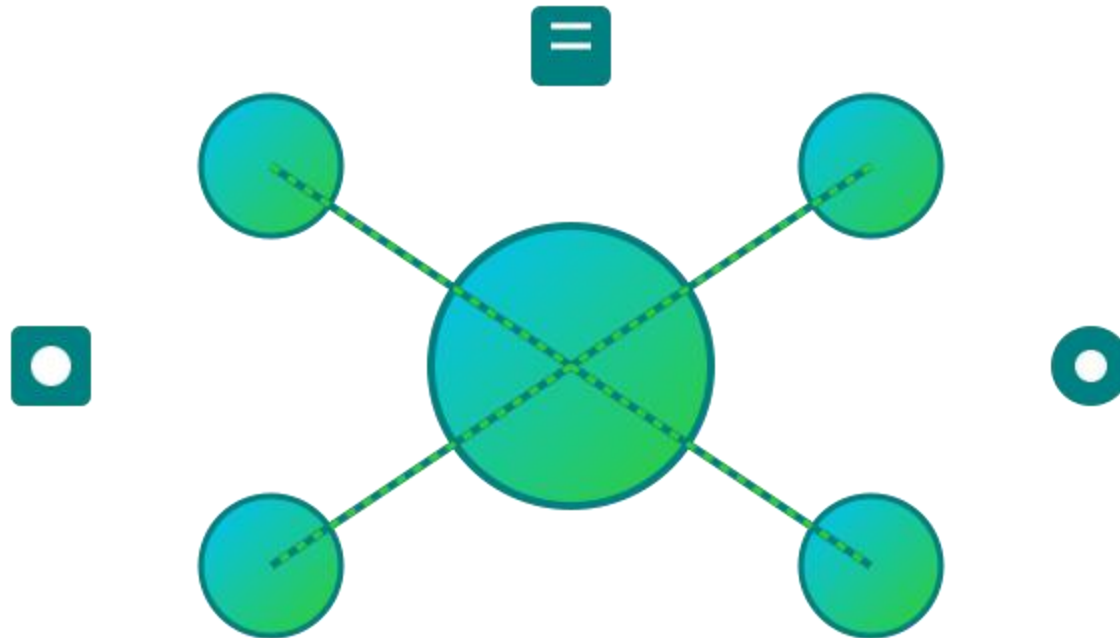
Tabela 2 – Linux (Ubuntu 24.04)

Czynność	Polecenie Linux
Uruchomienie procesu w tle	<code>sleep 300 & PID=\$!</code>
PID procesu	<code>echo \$PID</code>
Lista wątków w procesie (TID)	<code>ps -L -p \$PID -o pid,tid,psr,pri,stat,cmd</code>
Liczba wątków	<code>`ls /proc/\$PID/task</code>
Współdzielona przestrzeń adresowa	<code>`cat /proc/\$PID/maps</code>
Zasoby procesu (deskryptory plików)	<code>`ls -l /proc/\$PID/fd</code>

Cechy wątków

Cechy wątków

- wymagają mniej zasobów do działania i mniejszy jest czas ich tworzenia;
- dzięki współdzieleniu przestrzeni adresowej (pamięci) wątki jednego zadania mogą się między sobą komunikować w bardzo łatwy sposób, niewymagający pomocy ze strony systemu operacyjnego
- w systemach **wieloprocessorowych**, a także w tzw. systemach z wywłaszczaniem, wątki mogą być wykonywane równocześnie (współbieżnie).



Cechy wątków

Tabela 1 – Windows 11 (PowerShell)

Cecha wątków	Polecenie PowerShell
Mniejszy czas tworzenia / mniej zasobów	<code>`Get-Process -Id</code>
Lista wątków w procesie	<code>`(Get-Process -Id).Threads</code>
Liczba wątków	<code>(Get-Process -Id <PID>).Threads.Count</code>
Współdzielona przestrzeń adresowa (moduły)	<code>`(Get-Process -Id).Modules</code>
Równoczesne wykonywanie (wieloprocessorowość)	<code>`(Get-Process -Id).Threads</code>

Cechy wątków

Tabela 2 – Linux (Ubuntu 24.04)

Uruchom proces i pobierz jego PID

sleep 300 &

PID=\$!

echo \$PID # np. 3224

Cecha wątków	Polecenie Linux
Mniejszy czas tworzenia / mniej zasobów	<code>ps -p <PID> -o pid,%cpu,%mem,cmd</code>
Lista wątków w procesie (TID)	<code>ps -L -p <PID> -o pid,tid,psr,pri,stat,cmd</code>
Liczba wątków	<code>ls /proc/\$PID/task</code> # Wyświetli listę TID (Thread IDs) w tym procesie <code>ls /proc/\$PID/task wc -l</code> # policzy wątki
Współdzielona przestrzeń adresowa	<code>cat /proc/\$PID/maps head -n 20</code>
Równoczesne wykonywanie (wieloprosesorowość)	<code>ps -L -p <PID> -o pid,tid,psr,stat</code> (kolumna psr = numer CPU)

Cechy wielowątkowości

Cechy wielowątkowości:

1. Wszystkie wątki działają w ramach jednego procesu – wątki są częścią programu, dzielą jego zasoby.
2. Umożliwia przetwarzanie współbieżne – realizowane na wielu procesorach lub poprzez przełączanie kontekstu na jednym procesorze (symulacja równoległości).
3. Może obniżyć wydajność – konieczność stosowania mechanizmów synchronizacji (np. semaforey, muteksy) w celu uniknięcia konfliktów przy dostępie do wspólnych danych.
4. Współdzielona przestrzeń adresowa – wszystkie wątki mają dostęp do tych samych zmiennych, obiektów i struktur w pamięci oraz korzystają z tych samych zasobów systemowych (np. uchwytów do plików).



Cechy wielowątkowości

Wielowątkowość w Windows 11 (PowerShell)

Cecha	Narzędzia / Polecenia	Co obserwujesz
Wątki w obrębie jednego procesu	<code>Start-Process + Get-Process Threads</code>	Wątki należą do jednego PID, współdzielą zasoby procesu.
Liczba wątków / stan wątków	<code>(Get-Process -Id \$p.Id) .Threads.Count</code> i <code>`... .Threads</code>	<code>Select Id, ThreadState, PriorityLevel`</code>
Współbieżność na wielu rdzeniach	Task Manager / Performance Monitor	Równoległe wykonywanie wątków na różnych rdzeniach.
Synchronizacja (muteksy, semaforey)	Process Explorer (uchwyty synchronizacji)	Mechanizmy synchronizacji widoczne w uchwytach.
Współdzielona przestrzeń adresowa	<code>`(Get-Process -Id \$p.Id).Modules</code>	<code>Select ModuleName`</code>
Zasoby procesu (RAM, uchwyty)	<code>`Get-Process -Id \$p.Id</code>	<code>Select WorkingSet, HandleCount`</code>
Identyfikacja PID vs TID	<code>Threads . Id</code> (TID) wewnątrz procesu (PID)	Rozróżnienie PID (proces) i TID (wątek).

Uruchom proces i pobierz jego PID

sleep 300 &

PID=\$!

echo \$PID # np. 3224 Wielowątkowość w Linux (Ubuntu 24.04)

Cechy wielowątkowości

Cecha	Narzędzia / Polecenia	Co obserwujesz
Wątki w obrębie jednego procesu	<code>ps -L -p \$PID,</code> <code>/proc/\$PID/task</code>	Wątki należą do jednego PID, współdzielą zasoby procesu.
Liczba wątków / stan wątków	<code>ps -L -p \$PID -o</code> <code>pid,tid,psr,stat,cmd</code>	Liczba wątków, ich stan i CPU.
Współbieżność na wielu rdzeniach	Kolumna <code>psr</code> w <code>ps</code>	Wątki działają na różnych rdzeniach.
Synchronizacja (muteksy, semafony)	<code>strace, lsof</code>	Mechanizmy synchronizacji widoczne w syscall'ach i uchwytach.
Współdzielona przestrzeń adresowa	<code>cat /proc/\$PID/maps</code>	Te same mapowania pamięci dla wszystkich wątków.
Zasoby procesu (RAM, deskryptory)	<code>/proc/\$PID/status, smaps</code>	Wątki współdzielą pamięć i deskryptory.
Identyfikacja PID vs TID	<code>pid vs tid w ps -L</code>	Rozróżnienie PID (proces) i TID (wątek).

Komunikacja i zagrożenia wielowątkowości

Komunikacja i zagrożenia wielowątkowości

- **Komunikacja między wątkami** jest bardzo łatwa – wystarczy odwoływać się do tych samych zmiennych i obiektów.
- **Komunikacja między procesami** wymaga zastosowania mechanizmów IPC (InterProcess Communication).
- **Współdzielenie przestrzeni adresowej** niesie zagrożenie:
 - – jeden „wadliwy” wątek może zagrozić wykonaniu całego programu.
- Każdy wielowątkowy system operacyjny zapewnia **mechanizmy synchronizacji wątków**, które należy bezwzględnie stosować.



Komunikacja i zagrożenia wielowątkowości

Windows 11 – Wielowątkowość i komunikacja

Cecha	Jak sprawdzić w Windows	Obserwacja
Komunikacja między wątkami	Wątki współdzielą zmienne i moduły w obrębie procesu. Sprawdź: <code>(Get-Process -Id \$p.Id).Modules</code>	Wszystkie wątki widzą te same moduły i przestrzeń adresową.
Komunikacja między procesami (IPC)	Named Pipes, Shared Memory, Sockets. Możesz podejrzeć uchwyt w Process Explorer .	Procesy wymagają mechanizmów IPC – nie współdzielą przestrzeni adresowej.
Zagrożenie wadliwego wątku	W aplikacji testowej (np. C#, Python) wywołaj wyjątek w jednym wątku.	Wadliwy wątek może zakończyć cały proces.
Mechanizmy synchronizacji	W kodzie: <code>Mutex</code> , <code>Semaphore</code> , <code>Monitor</code> . W systemie: uchwyt w Process Explorer .	Synchronizacja jest konieczna, aby uniknąć warunków wyścigu.
Identyfikacja wątków	PowerShell: <code>(Get-Process -Id \$p.Id).Threads</code>	<code>Select Id, ThreadState`</code>

Komunikacja i zagrożenia wielowątkowości

Linux Ubuntu 24.04 – Wielowątkowość i komunikacja

Cecha	Jak sprawdzić w Linux	Obserwacja
Komunikacja między wątkami	Wątki w tym samym PID współdzielą pamięć. Sprawdź: <code>ps -L -p \$PID</code> <code>cat /proc/\$PID/maps</code>	Wszystkie wątki mają te same mapowania pamięci.
Komunikacja między procesami (IPC)	Polecenie: <code>ipcs</code> (pamięć współdzielona, semaforey, kolejki)	Procesy wymagają mechanizmów IPC – nie współdzielą przestrzeni adresowej.
Zagrożenie wadliwego wątku	W aplikacji testowej (np. C, Python) wywołaj <code>abort()</code> w jednym wątku.	Wadliwy wątek kończy cały proces.
Mechanizmy synchronizacji	W kodzie: <code>pthread_mutex</code> , <code>semaphores</code> . W systemie: <code>strace -p \$PID (syscall futex)</code> .	Synchronizacja jest konieczna, aby uniknąć warunków wyścigu.
Identyfikacja wątków	Polecenie: <code>ps -L -p \$PID -o pid,tid,psr,stat</code>	Wątki mają własne TID, ale współdzielą PID procesu.

Dzielenie czasu

Dzielenie czasu (time sharing):

- Mechanizm stosowany w **wielozadaniowych systemach operacyjnych**, który pozwala na wykonywanie wielu zadań „jednocześnie” na jednym procesorze poprzez **przydzielanie kwantów czasu** każdemu procesowi (lub wątkowi).
- W rzeczywistości jest to **przełączanie kontekstu** w krótkich odstępach czasu, co daje wrażenie równoległości.
- W systemach **wieloprocesorowych** część zadań może być faktycznie wykonywana równolegle na różnych rdzeniach/procesorach.

Planista (scheduler) odpowiada za:

- Kolejność wykonywania procesów (algorytmy: FCFS, Round Robin, priorytetowe).
- Przydziały czasowe (kwanty czasu dla każdego procesu).



Dzielenie czasu

Windows 11 – Dzielenie czasu i planista

Cecha	Jak sprawdzić w Windows	Obserwacja
Mechanizm dzielenia czasu	System używa wielopoziomowej kolejki z priorytetami (Round Robin + priorytety).	Procesy i wątki otrzymują kwanty czasu, przełączanie kontekstu daje wrażenie równoległości.
Przełączanie kontekstu	Performance Monitor → licznik: Thread → Context Switches/sec.	Widać częstotliwość przełączania wątków.
Priorytety procesów/wątków	PowerShell: (Get-Process -Id \$PID).Threads	Select Id, PriorityLevel Zmień priorytet: Get-Process notepad
Algorytmy planisty	Nie można zmienić algorytmu, ale można obserwować efekty w Task Manager i Process Explorer .	Windows stosuje Round Robin z priorytetami.
Wieloprocessorowość	Task Manager → Wydajność → CPU (obciążenie rdzeni).	Widać równoległe wykonywanie na wielu rdzeniach.

Dzielenie czasu

Linux Ubuntu 24.04 – Dzielenie czasu i planista

Cecha	Jak sprawdzić w Linux	Obserwacja
Mechanizm dzielenia czasu	Linux używa Completely Fair Scheduler (CFS) dla zwykłych procesów.	Procesy otrzymują proporcjonalny czas CPU, przełączanie kontekstu daje wrażenie równoległości.
Przełączanie kontekstu	Polecenie: <code>vmstat 1</code> → kolumna <code>cs</code> (context switches).	Widać liczbę przełączeń kontekstu na sekundę.
Priorytety procesów	Polecenie: <code>ps -eo pid,comm,policy,pri</code> Zmień priorytet: <code>renice -n -5 -p <PID></code>	Wyższy priorytet zwiększa czas CPU dla procesu.
Polityka planisty	Polecenie: <code>chrt -p <PID></code> Zmień politykę: <code>chrt -r -p 20 <PID></code>	Możesz ustawić Round Robin lub FIFO dla procesu.
Algorytmy planisty	Podjrzysz szczegóły: <code>`cat /proc/sched_debug</code>	<code>head -n 50`</code>
Wieloprocessorowość	Polecenie: <code>htop lub ps -L -p <PID> -o pid,tid,psr</code>	Widać, że wątki działają na różnych rdzeniach.

Proces vs Wątek - wizualizacja

Proces vs Wątek – wizualizacja pokazująca analogię:

Proces = mieszkanie (ma własne zasoby),

Wątek = pokój w mieszkaniu (dzieli pamięć i zasoby z innymi wątkami w tym samym procesie).

proces = mieszkanie,

wątek = pokój.



Wywłaszczenie

Wywłaszczenie

- **Wywłaszczenie** – technika stosowana w środowiskach wielozadaniowych, w której scheduler może wstrzymać aktualnie wykonywane zadanie (proces lub wątek), aby umożliwić działanie innemu.
- Zawieszenie jednego procesu **nie blokuje całego systemu operacyjnego**.
- Umożliwia dokładne określenie czasu, w jakim dany proces może korzystać z procesora.
- W systemach **bez wywłaszczenia** zadania same informują scheduler, kiedy przekazać sterowanie innym zadaniom (brak zgłoszenia = spowolnienie pracy).
- Wywłaszczanie jest często ograniczane – np. procedury obsługi przerwań sprzętowych są zwykle **niewywłaszczalne**.



Wywłaszczenie

Windows 11 – Wywłaszczenie

Cecha	Jak sprawdzić w Windows	Obserwacja
Czy system jest wywłaszczeniowy?	Windows NT (w tym 11) jest preemptive. Sprawdź przełączanie kontekstu: <code>perfmon</code> → licznik <code>Thread</code> → <code>Context Switches/sec</code> .	Wysoka wartość oznacza częste odbieranie czasu procesom przez scheduler.
Zmiana priorytetu procesu	PowerShell: <code>`Get-Process notepad</code>	<code>ForEach-Object { \$_.PriorityClass = 'High' }</code>
Dowód wywłaszczenia	Uruchom dwa procesy obciążające CPU (np. <code>while(\$true){}</code> w PowerShell). Zmieniaj priorytety i obserwuj w Task Manager .	Scheduler odbiera czas jednemu procesowi i daje drugiemu.
Polityka planisty	Windows używa wielopoziomowej kolejki z priorytetami (Round Robin + priorytety). Nie można zmienić algorytmu, ale można obserwować efekty w Process Explorer .	Wywłaszczenie działa na podstawie priorytetów i kwantów czasu.
Procedury niewywłaszczalne	Obsługa przerw sprzętowych jest niewywłaszczalna – nie można tego zmienić, ale jest to cecha systemu.	Krótkie procedury ISR działają bez przerwania przez scheduler.

Wywłaszczenie

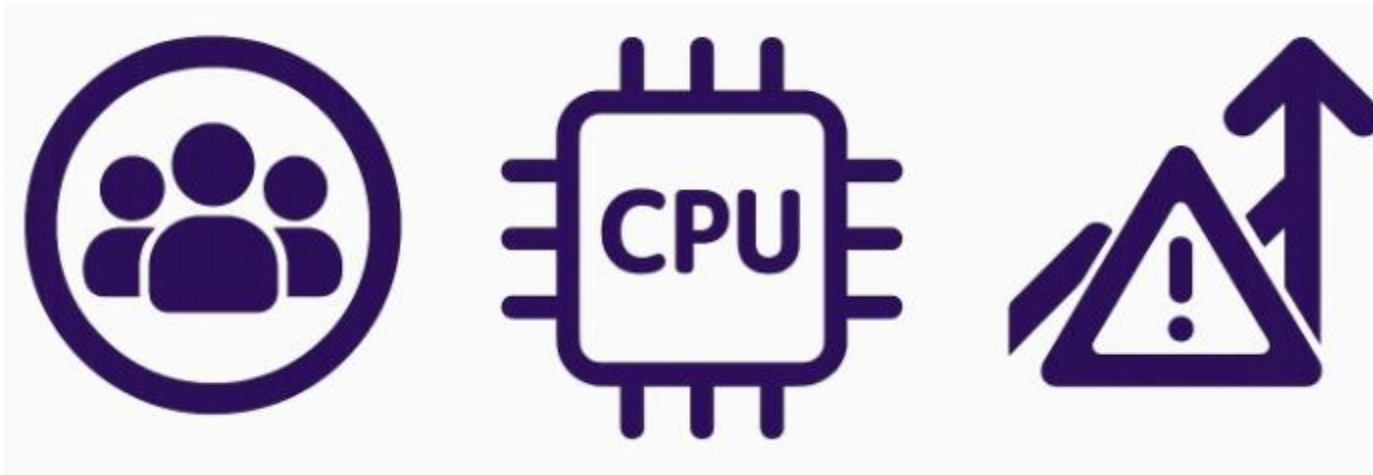
Linux Ubuntu 24.04 – Wywłaszczenie

Cecha	Jak sprawdzić w Linux	Obserwacja
Czy system jest wywłaszczeniowy?	Linux z jądrem 5.x używa Completely Fair Scheduler (CFS) – w pełni wywłaszczeniowy. Sprawdź: <code>vmstat 1</code> → kolumna <code>cs</code> (context switches).	Widać liczbę przełączeń kontekstu na sekundę.
Polityka planisty	Polecenie: <code>ps -eo pid,comm,policy,pri</code>	TS = time-sharing (wywłaszczeniowe), RR = Round Robin, FIFO = niewywłaszczeniowe.
Zmiana priorytetu	Polecenie: <code>renice -n -10 -p <PID></code>	Proces z wyższym priorytetem dostaje więcej czasu CPU.
Dowód wywłaszczenia	Uruchom dwa procesy obciążające CPU: <code>yes > /dev/null & yes > /dev/null &</code> Zmień priorytet jednego: <code>renice -n -10 -p <PID></code>	Scheduler odbiera czas jednemu procesowi i daje drugiemu.
Procedury niewywłaszczalne	Zmień politykę na FIFO: <code>chrt -f -p 99 <PID></code>	Proces z FIFO i wysokim priorytetem może „zablokować” CPU (brak wywłaszczenia).

Wielodostępność, wielobieżność i skalowalność

Wielodostępność, wielobieżność i skalowalność

- **Wielodostępność** – cecha systemu umożliwiająca pracę wielu użytkowników jednocześnie (np. komputer z terminalami).
- **Wielobieżność** – cecha jądra systemu operacyjnego, pozwalająca na jednoczesne wykonywanie kilku procesów w trybie jądra.
- **Skalowalność** – utrzymanie wydajności przy zwiększaniu liczby elementów lub ich miniaturyzacji.



Wielodostępność, wielobieżność i skalowalność

Windows 11 – Wielodostępność, Wielobieżność, Skalowalność

Cecha	Jak sprawdzić w Windows	Obserwacja
Wielodostępność	Polecenie: <code>quser</code> lub: <code>`Get-CimInstance Win32_ComputerSystem</code>	Select UserName`
Wielobieżność (kernel)	Windows NT jest wielobieżny. Sprawdź przełączanie kontekstu: <code>perfmon → licznik Thread → Context Switches/sec.</code>	Wysoka wartość oznacza, że wiele wątków działa w trybie jądra jednocześnie.
Skalowalność	Polecenie: <code>`Get-CimInstance Win32_Processor</code>	Select NumberOfCores, NumberOfLogicalProcessors`

Wielodostępność, wielobieżność i skalowalność

Linux Ubuntu 24.04 – Wielodostępność, Wielobieżność, Skalowalność

Cecha	Jak sprawdzić w Linux	Obserwacja
Wielodostępność	Polecenie: <code>who</code> lub <code>w</code>	Widać wielu zalogowanych użytkowników (lokalnie i przez SSH).
Wielobieżność (kernel)	Linux jest wielobieżny. Sprawdź przełączanie kontekstu: <code>vmstat 1</code> → kolumna <code>cs</code> .	Wysoka wartość oznacza, że wiele wątków działa w trybie jądra jednocześnie.
Skalowalność	Polecenie: <code>ls cpu</code>	Widać liczbę rdzeni i wątków. Obserwuj w <code>htop</code> , jak procesy są rozkładane na rdzeniach.

Rodzaje wielozadaniowości

Wielozadaniowość kooperatywna

- Kilka aplikacji działa „równocześnie”, ale same aplikacje decydują, kiedy oddają kontrolę (brak wymuszonego przełączania przez system).
- Wadą jest możliwość „zawieszenia” systemu, jeśli aplikacja nie odda czasu procesora.
- Przykład: starsze wersje Windows (3.x).

Wielozadaniowość wywłaszczeniowa (preemptive)

- To najczęściej stosowana forma – jądro systemu operacyjnego decyduje o przełączaniu zadań, przydzielając im kwanty czasu.
- Zapewnia stabilność i sprawiedliwy podział czasu.

Wieloprocessorowość

- Wykorzystanie wielu procesorów/rdzeni do faktycznego równoległego wykonywania zadań.
- W połączeniu z wielozadaniowością daje wysoką wydajność.



Rodzaje wielozadaniowości

Windows 11 – Wielozadaniowość i wieloprocessorowość

Cecha	Jak sprawdzić	Obserwacja
Kooperatywna	Brak – system jest wywłaszczeniowy.	Nie ma możliwości zawieszenia systemu przez jeden proces.
Wywłaszczeniowa	perfmon → licznik Context Switches/sec. Zmień priorytet: PriorityClass.	Scheduler odbiera czas jednemu procesowi i daje innemu.
Wieloprocessorowość	Get-CimInstance Win32_Processor	Widać obsługę wielu rdzeni, rozkład obciążenia w Task Manager.

Rodzaje wielozadaniowości

Linux Ubuntu 24.04 – Wielozadaniowość i wieloprocessorowość

Cecha	Jak sprawdzić	Obserwacja
Kooperatywna	Brak – system jest wywłaszczeniowy.	Nie ma możliwości zawieszenia systemu przez jeden proces.
Wywłaszczeniowa	<code>vmstat 1</code> → kolumna <code>cs</code> . Zmień priorytet: <code>renice</code> .	Scheduler odbiera czas jednemu procesowi i daje innemu.
Wieloprocessorowość	<code>lscpu</code> , <code>htop</code>	Widać obsługę wielu rdzeni, rozkład obciążenia w <code>htop</code> .

Równoległość vs Współbieżność

Równoległość (parallelism)

- Wiele zadań wykonywanych jednocześnie na różnych rdzeniach/procesorach.
- Faktyczna równoczesność dzięki wieloprocessorowości lub wielordzeniowości.
- Analogią jest kilku kucharzy gotujących różne potrawy w tym samym czasie.

Współbieżność (concurrency)

- Wiele zadań przeplata się w czasie na jednym rdzeniu – system przełącza kontekst między nimi.
- Pozorna równoczesność dzięki mechanizmowi dzielenia czasu (time sharing).
- Analogią jest jeden kucharz, który przełącza się między potrawami.



Równoległość vs Współbieżność

Windows 11 – Równoległość vs Współbieżność

Cecha	Jak sprawdzić w Windows	Obserwacja
Równoległość (Parallelism)	Polecenie: <code>`Get-CimInstance Win32_Processor</code>	Select NumberOfCores, NumberOfLogicalProcessors` Obserwuj w Task Manager → CPU .
Współbieżność (Concurrency)	Użyj Performance Monitor : Dodaj licznik Thread → Context Switches/sec. Uruchom dwa procesy obciążające CPU i obserwuj w Task Manager .	Procesy „działają równocześnie”, ale na jednym rdzeniu przełączają się w czasie – pozorna równoczesność.

Równoległość vs Współbieżność

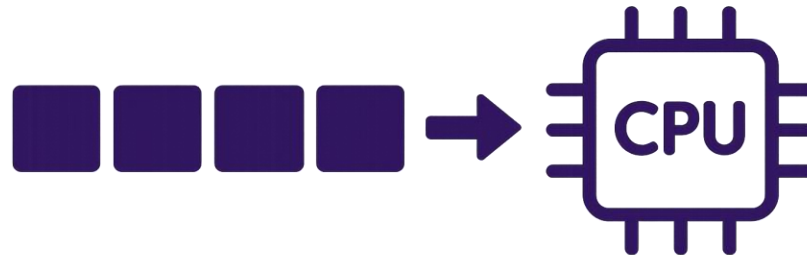
Linux Ubuntu 24.04 – Równoległość vs Współbieżność

Cecha	Jak sprawdzić w Linux	Obserwacja
Równoległość (Parallelism)	Polecenie: <code>lscpu</code> (liczba rdzeni). Obserwuj w <code>htop</code> (kolumna CPU). Uruchom: <code>yes > /dev/null & yes > /dev/null &</code>	Procesy działają jednocześnie na różnych rdzeniach – faktyczna równoległość.
Współbieżność (Concurrency)	Polecenie: <code>vmstat 1</code> → kolumna <code>cs</code> (context switches). Uruchom dwa procesy na jednym rdzeniu: <code>taskset -c 0 yes > /dev/null &</code> <code>taskset -c 0 yes > /dev/null &</code>	Procesy przełączają się na jednym rdzeniu – pozorna równoczesność dzięki dzieleniu czasu.

Schemat działania planisty

Jak działa planista w systemie operacyjnym?

- **Planista** to „kierownik” procesora – decyduje, który proces ma dostać czas na CPU i w jakiej kolejności.
- **Dlaczego to ważne?** Procesor jest jeden (lub kilka), a procesów jest wiele – trzeba ustalić zasady, kto kiedy pracuje.
- **Algorytmy planowania:**
 - Round Robin – każdy proces dostaje „kawałek czasu” po kolei (jak w kolejce do kasy).
 - Priorytety – ważniejsze procesy dostają pierwszeństwo.
 - CFS (Completely Fair Scheduler) – stara się, aby każdy miał sprawiedliwy dostęp do CPU.
- **Kolejka procesów** – wyobraź sobie procesy ustawione w kolejce, czekające na swoją kolej do procesora.



Schemat działania planisty

Windows 11 – Planista (Scheduler)

Cecha	Jak sprawdzić w Windows	Obserwacja
Rodzaj planisty	Windows NT używa wielopoziomowej kolejki z priorytetami (Round Robin w obrębie poziomu).	Procesy z wyższym priorytetem dostają więcej czasu CPU.
Algorytmy	Round Robin + priorytety (dynamiczne i statyczne).	Sprawiedliwy podział czasu w obrębie poziomu priorytetu.
Context Switches	perfmon → licznik Thread → Context Switches/sec.	Wysoka wartość oznacza częste przełączanie kontekstu.
Priorytety procesów	PowerShell: `Get-Process	Select Name, PriorityClass. Zmień priorytet: Get-Process notepad
Kolejka procesów	Task Manager → zakładka Szczegóły .	Procesy sortowane wg priorytetu i użycia CPU.

Schemat działania planisty

Linux Ubuntu 24.04 – Planista (Scheduler)

Cecha	Jak sprawdzić w Linux	Obserwacja
Rodzaj planisty	Linux używa Completely Fair Scheduler (CFS) dla zwykłych procesów.	Sprawiedliwy podział czasu CPU między procesy.
Algorytmy	CFS (domyślny), Round Robin (RR), FIFO (FIFO).	Możesz zmienić politykę na RR lub FIFO.
Context Switches	<code>vmstat 1</code> → kolumna <code>cs</code> .	Wysoka wartość oznacza częste przełączanie kontekstu.
Priorytety procesów	Polecenie: <code>ps -eo pid,comm,policy,pri.</code> Zmień priorytet: <code>renice -n -10 -p <PID>.</code>	Wyższy priorytet zwiększa czas CPU dla procesu.
Kolejka procesów	<code>`cat /proc/sched_debug</code>	<code>head -n 50`.</code>

Bezpieczeństwo i synchronizacja

Problemy:

- Race condition – sytuacja, gdy dwa wątki jednocześnie próbują zmienić tę samą zmienną lub zasób, co prowadzi do nieprzewidywalnych wyników.
- Deadlock – zakleszczenie, gdy wątki czekają na zasoby blokowane przez siebie nawzajem i żaden nie może kontynuować.

Rozwiązania:

- Mutex (mutual exclusion) – blokada zapewniająca, że tylko jeden wątek może korzystać z zasobu w danym momencie.
- Semafor – mechanizm kontroli dostępu do zasobów, pozwala na ograniczoną liczbę wątków korzystających z zasobu.
- Monitor – wyższy poziom synchronizacji, łączący blokadę i warunki synchronizacji w jednym obiekcie.

Dlaczego to ważne?

Bez synchronizacji wątki mogą nadpisywać dane, powodować błędy logiczne i utratę spójności danych.



Bezpieczeństwo i synchronizacja

Windows 11 – Problemy i mechanizmy synchronizacji

Element	Jak sprawdzić w Windows	Obserwacja
Race Condition	Napisać prosty program w PowerShell/C# z wieloma wątkami modyfikującymi wspólną zmienną bez synchronizacji.	Wyniki nieprzewidywalne (np. suma nie zgadza się).
Deadlock	Zasymulować w kodzie (dwa wątki blokują zasoby w odwrotnej kolejności). Podejrzec wątki w Process Explorer (stan „Wait”).	Wątki czekają na siebie, brak postępu.
Mutex	W kodzie: <code>System.Threading.Mutex</code> lub <code>SemaphoreSlim</code> . Uchwyty widoczne w Process Explorer (Handle → Mutex).	Tylko jeden wątek korzysta z zasobu w danym momencie.
Semafor	W kodzie: <code>SemaphoreSlim</code> (C#, PowerShell).	Ograniczona liczba wątków korzysta z zasobu jednocześnie.
Monitor	W kodzie C# (<code>lock</code>) lub PowerShell (<code>Monitor.Enter</code>).	Synchronizacja + warunki w jednym obiekcie.

Bezpieczeństwo i synchronizacja

Windows 11 – Problemy i mechanizmy synchronizacji

Element	Jak sprawdzić w Windows	Obserwacja
Race Condition	Napisać prosty program w PowerShell/C# z wieloma wątkami modyfikującymi wspólną zmienną bez synchronizacji.	Wyniki nieprzewidywalne (np. suma nie zgadza się).
Deadlock	Zasymulować w kodzie (dwa wątki blokują zasoby w odwrotnej kolejności). Podejrzec wątki w Process Explorer (stan „Wait”).	Wątki czekają na siebie, brak postępu.
Mutex	W kodzie: <code>System.Threading.Mutex</code> lub <code>SemaphoreSlim</code> . Uchwyty widoczne w Process Explorer (Handle → Mutex).	Tylko jeden wątek korzysta z zasobu w danym momencie.
Semafor	W kodzie: <code>SemaphoreSlim</code> (C#, PowerShell).	Ograniczona liczba wątków korzysta z zasobu jednocześnie.
Monitor	W kodzie C# (<code>lock</code>) lub PowerShell (<code>Monitor.Enter</code>).	Synchronizacja + warunki w jednym obiekcie.

Bezpieczeństwo i synchronizacja

Linux Ubuntu 24.04 – Problemy i mechanizmy synchronizacji

Element	Jak sprawdzić w Linux	Obserwacja
Race Condition	Napisać program w C/Python z <code>pthread</code> bez synchronizacji.	Wyniki nieprzewidywalne (np. błędna suma).
Deadlock	Zasymulować w kodzie (dwa wątki blokują zasoby). Sprawdzić wątki: <code>ps -L -p <PID></code> (stan D).	Wątki czekają na siebie, brak postępu.
Mutex	W kodzie: <code>pthread_mutex_lock/unlock</code> . W systemie: <code>strace -p <PID></code> (<code>syscall futex</code>).	Tylko jeden wątek korzysta z zasobu w danym momencie.
Semafor	W kodzie: <code>sem_init, sem_wait, sem_post</code> .	Ograniczona liczba wątków korzysta z zasobu jednocześnie.
Monitor	W językach wysokiego poziomu (Java: <code>synchronized</code> , Python: <code>threading.Lock()</code>).	Synchronizacja + warunki w jednym obiekcie.

Nowoczesne trendy

Wielordzeniowe procesory i Hyper-Threading

- Zwiększają wydajność poprzez równoległe wykonywanie wielu wątków.
- Hyper-Threading (Intel) pozwala jednemu rdzeniowi obsługiwać dwa wątki logiczne, co poprawia wykorzystanie zasobów CPU.

Programowanie asynchroniczne (async/await)

- Popularne w językach takich jak Python (moduł asyncio) i JavaScript (Promise + async/await).
- Umożliwia obsługę wielu operacji I/O bez blokowania głównego wątku, co zwiększa responsywność aplikacji.

Konteneryzacja (Docker) i mikroserwisy

- Zapewniają izolację procesów i środowisk w chmurze.
- Ułatwiają skalowanie aplikacji i zarządzanie zasobami w architekturze rozproszonej.



Nowoczesne trendy

Windows 11 – Technologie równoległości i skalowania

Element	Jak sprawdzić w Windows	Obserwacja
Wielordzeniowość / Hyper-Threading	PowerShell: <code>`Get-CimInstance Win32_Processor</code>	Select NumberOfCores, NumberOfLogicalProcessors` Task Manager → Wydajność → CPU
Programowanie asynchroniczne	Skrypt w C# (<code>async/await</code>) lub Python (<code>asyncio</code>). Przykład w PowerShell: <code>Start-Job</code> dla zadań równoległych.	Operacje I/O nie blokują głównego wątku, zwiększając responsywność aplikacji.
Konteneryzacja / mikroserwisy	Zainstaluj Docker Desktop. Sprawdź: <code>docker version</code> <code>docker run hello-world</code>	Kontenery izolują środowiska, umożliwiają skalowanie i zarządzanie mikroserwisami.

Nowoczesne trendy

Linux Ubuntu 24.04 – Technologie równoległości i skalowania

Element	Jak sprawdzić w Linux	Obserwacja
Wielordzeniowość / Hyper-Threading	Polecenie: <code>lscpu</code> (liczba rdzeni i wątków logicznych). Obserwuj w <code>htop</code> (kolumna CPU).	Widać rdzenie i wątki logiczne (Hyper-Threading = podwójna liczba CPU względem rdzeni).
Programowanie asynchroniczne	Skrypt w Python (<code>asyncio</code>) lub Node.js (<code>async/await</code>). Przykład: <pre>async def main(): await asyncio.sleep(1)</pre>	Operacje I/O nie blokują głównego wątku, zwiększając responsywność aplikacji.
Konteneryzacja / mikroserwisy	Zainstaluj Docker: <code>sudo apt install docker.io</code> Sprawdź: <code>docker --version</code> <code>docker run hello-world</code>	Kontenery izolują środowiska, umożliwiają skalowanie i zarządzanie mikroserwisami.

Podsumowanie

Podsumowanie

- **Proces** = osobne mieszkanie (ma własne zasoby).
- **Wątek** = pokój w tym mieszkaniu (dzieli wspólne zasoby).
- **Wielozadaniowość** = wielu „lokatorów” korzysta z zasobów jednocześnie.
- Kluczowe elementy:
 - **Planista** – decyduje, kto i kiedy korzysta z CPU.
 - **Synchronizacja** – żeby wątki się nie „zderzały”.
 - **Bezpieczeństwo** – ochrona danych i zasobów.



Zadania dla uczniów

Zadania dla uczniów

1. Wyjaśnij różnicę między współbieżnością a równoległością na przykładzie codziennych czynności.
2. Wymień trzy algorytmy planowania i opisz ich działanie.
3. Co to jest race condition i jak można go uniknąć?
4. Jakie są zalety wielordzeniowych procesorów i Hyper-Threadingu?
5. Dlaczego wielowątki wymagają mechanizmów synchronizacji?



Pytania

1. Wyjaśnij, czym jest proces w systemie operacyjnym i jakie informacje zawiera jego PCB (Process Control Block).
2. Opisz różnicę między współbieżnością a równoległością, podając praktyczny przykład.
3. Jakie są główne etapy tworzenia i uruchamiania procesu w systemie operacyjnym?
4. Na czym polega wielowątkowość i jakie korzyści oraz zagrożenia się z nią wiążą?
5. Wyjaśnij, czym jest race condition i jak można mu zapobiegać.
6. Opisz mechanizmy synchronizacji w systemach wielowątkowych (np. mutex, semafor).
7. Jak działa planista zadań w systemie operacyjnym? Podaj przykłady algorytmów planowania.
8. Dlaczego wielowątkowość może w niektórych przypadkach obniżyć wydajność systemu?
9. Jakie są różnice między systemem jednozadaniowym a wielozadaniowym?
10. Wyjaśnij pojęcie „przełączanie kontekstu” i jego wpływ na wydajność systemu.
11. Jakie problemy mogą wystąpić w systemach wielowątkowych oprócz race condition? Podaj przykłady.
12. Wymień i opisz nowoczesne trendy w rozwoju systemów operacyjnych (np. konteneryzacja, wirtualizacja, bezpieczeństwo).