



ADRIAN GORAL, ISA '22

Systemy Operacyjne

Cały semestr na jednej prezentacji, czyli...
nauczmy się tego w jedną noc!

Autorzy merytoryczni

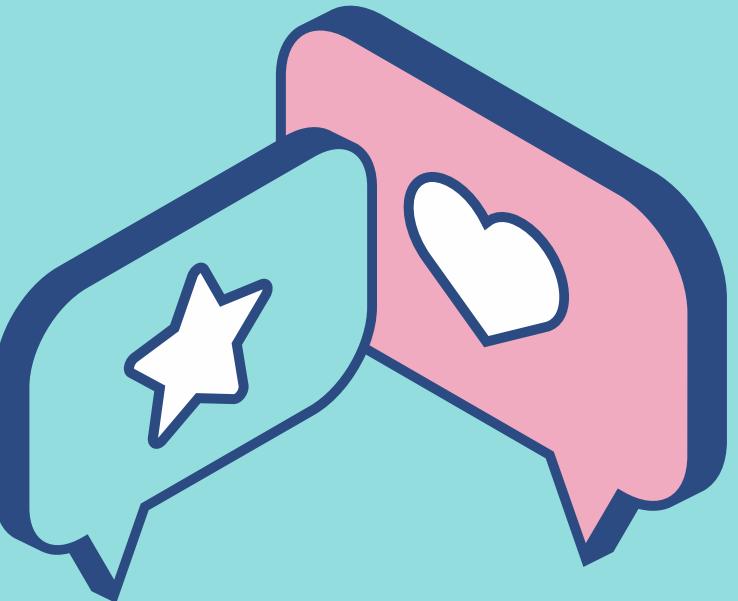
- Adrian Goral
- Stefan Wojciechowski
- Michał Kaszowski
- Adam Makarewicz
- Konrad Zarzecki
- Mikołaj Mariasz
- Julia Szymańska
- Bartłomiej Kuk
- Mateusz Moskal
- Filip Zioło
- Konrad Landzberg
- Mateusz Zubrzycki

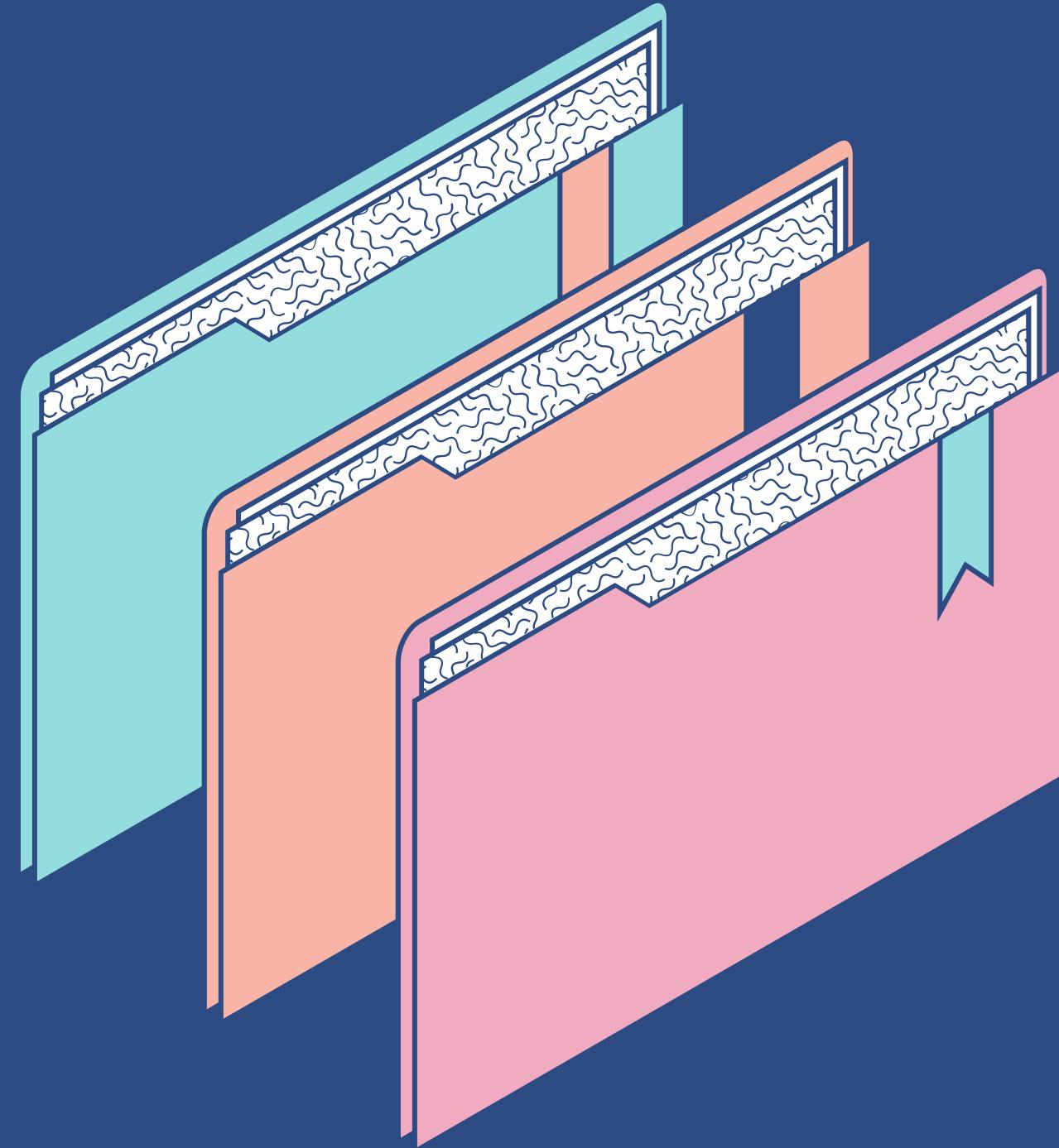
i wiele innych osób!



Oprawa wizualna

- Paulina Szulc
- Adrian Goral
- Mateusz Zubrzycki
- Michał Kaszowski





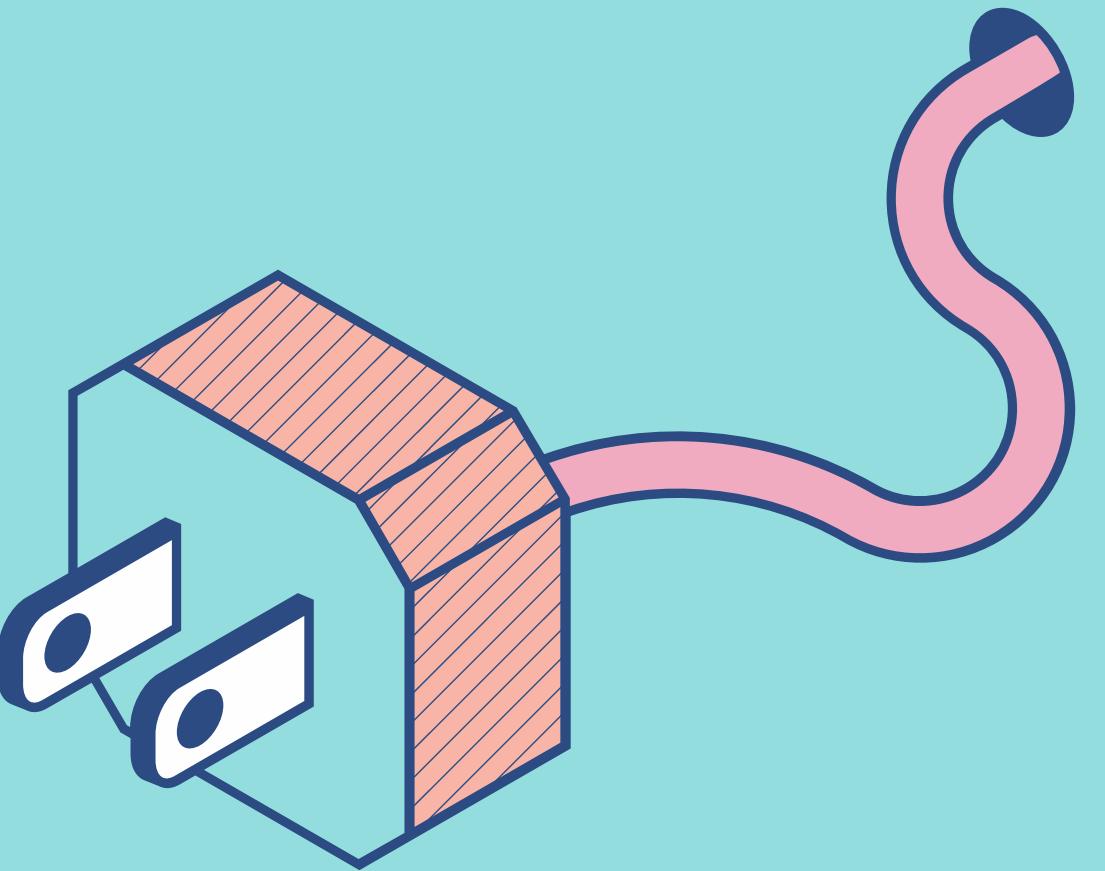
Agenda

ROZKŁAD JAZDY? JAZDA Z... SYSTEMAMI!

- Przejdziemy sobie przez wykłady 2-14
- Po wykładzie 6 i 9 (hehe) będzie chwila przerwy

Wykład 2

System Operacyjny
podstawa, budowa, klasyfikacja, powłoka





System Operacyjny

Oprogramowanie (program) zarządzające systemem komputerowym (sprzęt, oprogramowanie) i udostępniające jego zasoby i funkcje użytkownikowi. Tworzy środowisko do uruchamiania i wykonywania zadań (programów).

Zadania systemu operacyjnego

- 1** Tworzenie, obsługa i kontrola procesów
- |
- 2** Planowanie i przydział czasu procesora do zadań (procesów).
- |
- 3** Synchronizacja procesów i komunikacja międzyprocesowa
- |
- 4** Przydział pamięci operacyjnej do zadań, kontrola i obsługa pamięci.
- 5** Obsługa sprzętu (sterowniki), dostarczanie jednolitego interfejsu do obsługi sprzętu.
- |
- 6** Zarządzanie plikami (systemy plików).
- |
- 7** Obsługa sieci komputerowej.
- |
- 8** Zapewnienie bezpieczeństwa (pamięć, pliki itd.), autoryzacja dostępu

SPRAWDŹ SIĘ!

Zarządzanie plikami,
synchronizacja procesów i
dynamiczne linkowanie są
przykładami zadań wykonywanych
przez system operacyjny.



TAK

NIE

SPRAWDŹ SIĘ!

Zarządzanie plikami,
synchronizacja procesów i
dynamiczne linkowanie są
przykładami zadań wykonywanych
przez system operacyjny.



NIE





Rodzaje systemów operacyjnych

→ Sieciowy system operacyjny

OS z wbudowaną obsługą stosu sieciowego (protokołów).

Współcześnie wyspecjalizowany OS dla urządzeń sieciowych (np. ruterów)

→ Rozproszony system operacyjny

Zespół fizycznie odrębnych węzłów sieciowych wraz z ich komponentami systemowymi działającymi i widocznymi dla użytkownika końcowego jako jeden system

→ System Operacyjny Czasu Rzeczywistego (RTOS)

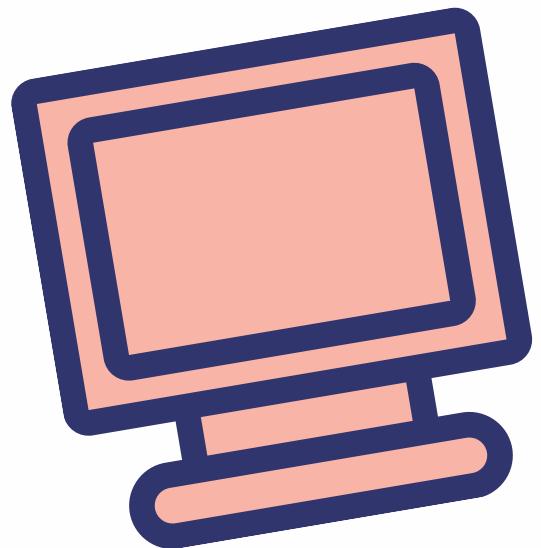
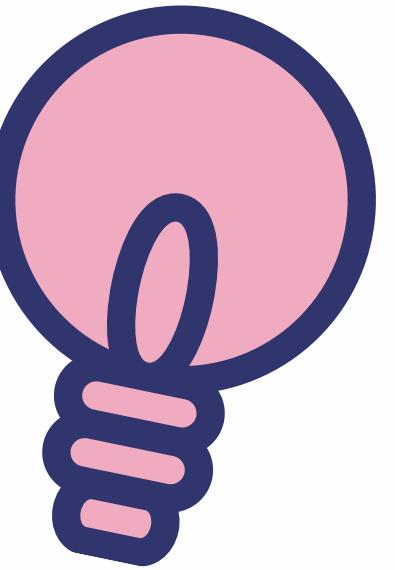
To system operacyjny gwarantujący wykonywanie operacji w zadanym czasie, nawet w przypadku wersji wielozadaniowej (np. sterowanie priorytetem)

→ Wbudowany System Operacyjny

System operacyjny dla urządzeń wbudowanych, wydajny, o niewielkim zapotrzebowaniu na zasoby. Często są też jednocześnie RTOS. Przykłady to: Windows CE, Minix 3.

Elementy Systemu Operacyjnego

1. Jądro
2. Powłoka
3. System plików
4. Pozostałe oprogramowanie systemowe

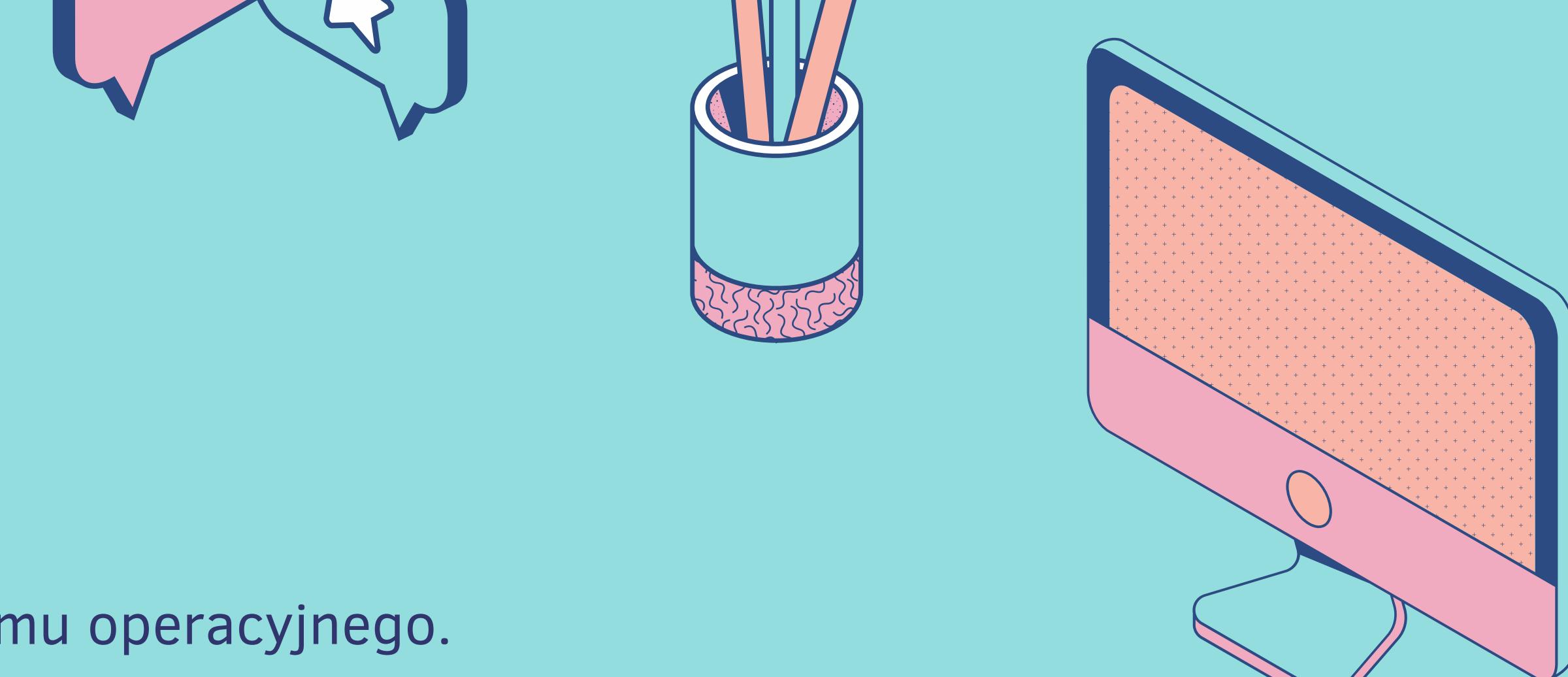


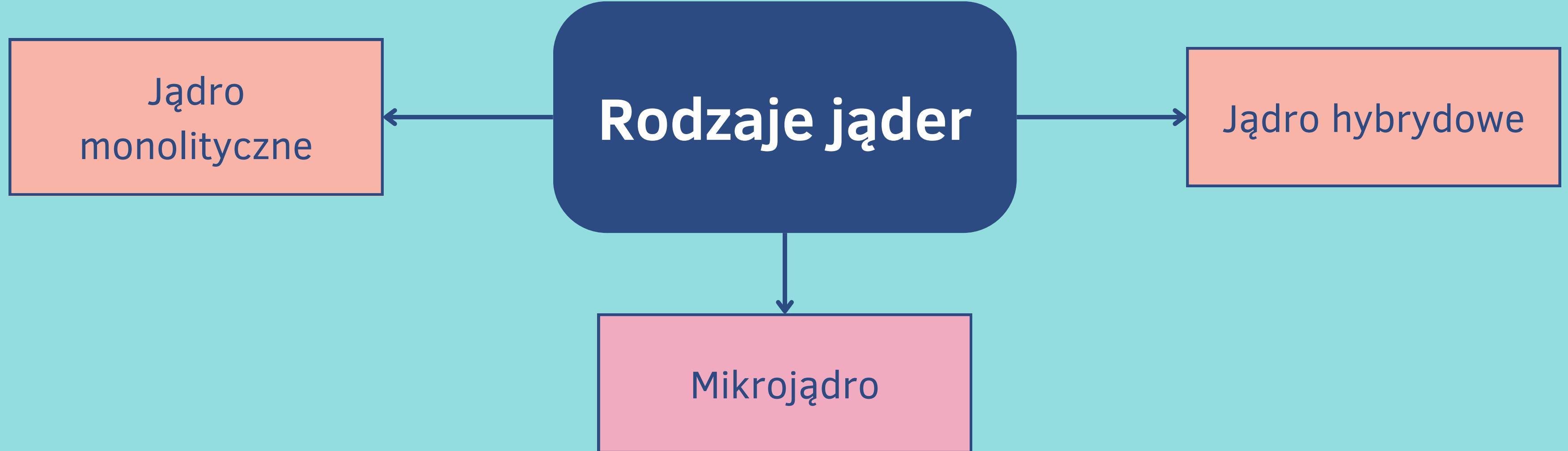
Jądro (kernel)

- Podstawowa część systemu operacyjnego.

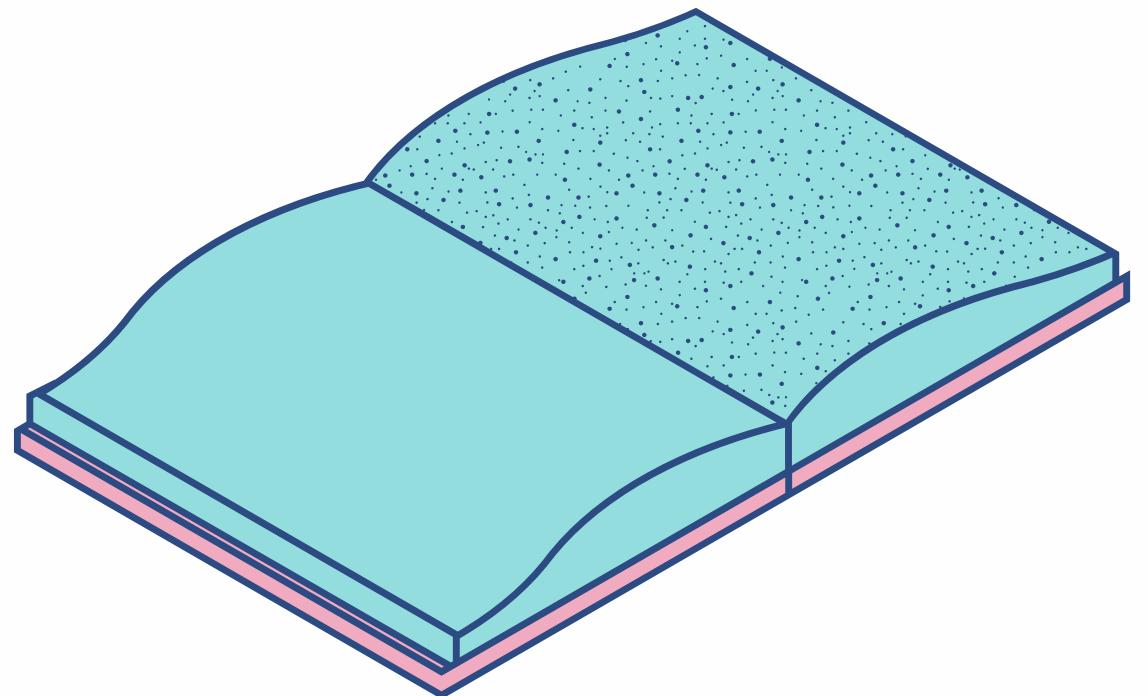
W zależności od typu jądra odpowiada albo za najważniejsze funkcje systemu, albo za praktycznie wszystkie funkcje.

-
- Użytkownicy korzystają z aplikacji (w tym systemowych, jak powłoka), które z kolei korzystają z usług jądra poprzez jego API (czyli wywołania systemowe)
 - Jądro pracuje w trybie uprzywilejowanym (pełny dostęp do zasobów)





Jądro monolityczne



Jądro to pojedynczy obraz w pamięci

Wysoka wydajność

Podatność na błędy

Możliwa modularność

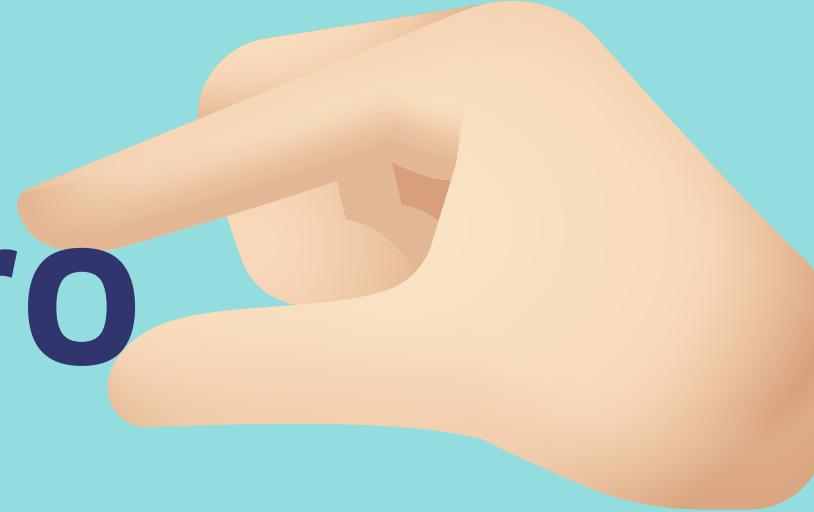
Obsługuje “wszystkie” funkcje od planisty i obsługi pamięci, po sieć, sterowniki i systemy plików

FreeBSD, Linux (tworzące GNU/Linux)

Wszystkie zadania są wykonywane przez jądro, które jest jednym dużym programem

Prostota, stabilność, łatwość komunikacji pomiędzy różnymi członkami jądra

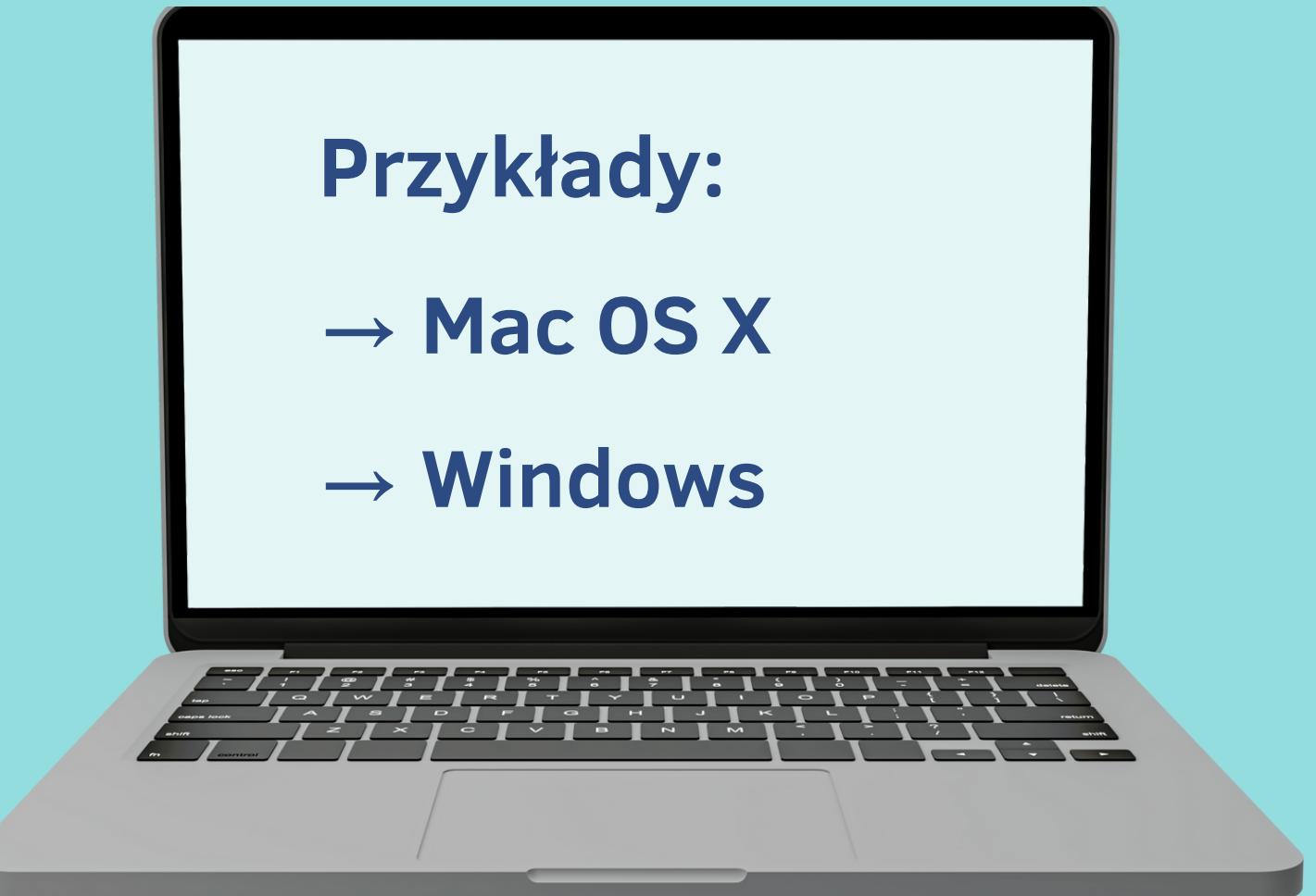
Mikrojądro



- 1** Wykonuje tylko nieliczne podstawowe funkcje
(np. wątki, IPC, planista)
- 2** Reszta funkcji obsługiwane przez serwery w
przestrzeni użytkownika
- 3** Większa odporność na błędy
- 4** Niższa wydajność
- 5** Nanokernel
- 6** Amoeba, MINIX, QNX, Mach i L4 (wraz z Hurd
tworzący GNU/Hurd)

Jądro hybrydowe

- Połączenie idei jądra **monolitycznego** i **mikrojądra**
- Wszystkie lub prawie wszystkie elementy w przestrzeni jądra
- Jądro składa się z **wielu osobnych elementów** (obszarów pamięci)
- Duża **wydajność**, mniejsze skutki błędów
- Nie zawsze wyróżniane jako osobny typ





Moduły jądra

- Fragmenty dołączane opcjonalnie
- Dynamiczne ładowane i usuwane z jądra
- Korzystają z wewnętrznego API jądra, a **nie** z wywołań systemowych (np. `printk` zamiast `printf`)
- Kontrola wielkości jądra
- Elastyczność mimo monolityczności
- Problem z błędami i testowaniem
- Pliki modułów (rozszerzenie `.ko`) zwykle umieszczane w `/lib/modules`
- Funkcje `init module` oraz `cleanup module`



Moduły jądra

- Ładowanie poprzez insmod
- Uwzględnianie zależności poprzez modprobe
- Lista zależności poprzez depmod
- Usuwanie poprzez użycie komendy rmmod
- Do modułów można przekazywać parametry: insmod plik modułu zm="my dev" nr=81
- Listing poprzez lsmod lub /proc/modules
- **Użycie demona kerneld:**
 - Automatyzacja zarządzaniem i usuwaniem modułów
 - Jądro musi być skompilowane z wsparciem

SPRAWDŹ SIĘ!

Jądro systemu operacyjnego zajmuje się jedynie najważniejszymi funkcjami systemu operacyjnego (szeregowanie procesów, pamięć wirtualna itp.).

TAK

NIE



SPRAWDŹ SIĘ!

Jądro systemu operacyjnego zajmuje się jedynie najważniejszymi funkcjami systemu operacyjnego (szeregowanie procesów, pamięć wirtualna itp.).

TAK

NIE



Praca z powłoką

Ctrl + D

symulacja końca pliku (kończenie
prac z niektórymi programami)

Ctrl + C

sygnał przerwania (**SIGINT**)

Ctrl + Z

zatrzymanie procesu (**SIGSTP**),
dodatkowo komendy `fg`, `bg` oraz &

Ctrl + W

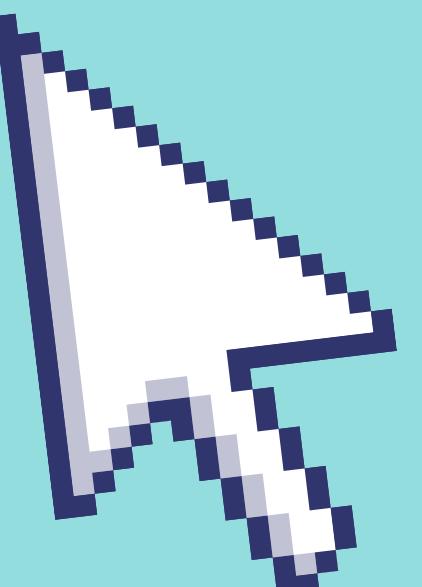
skasowanie poprzedniego słowa

Ctrl + Shift + C/V

kopiuj/wklej (terminale graficzne)

Ctrl + U/Y

wytnij/wklej obecny wiersz



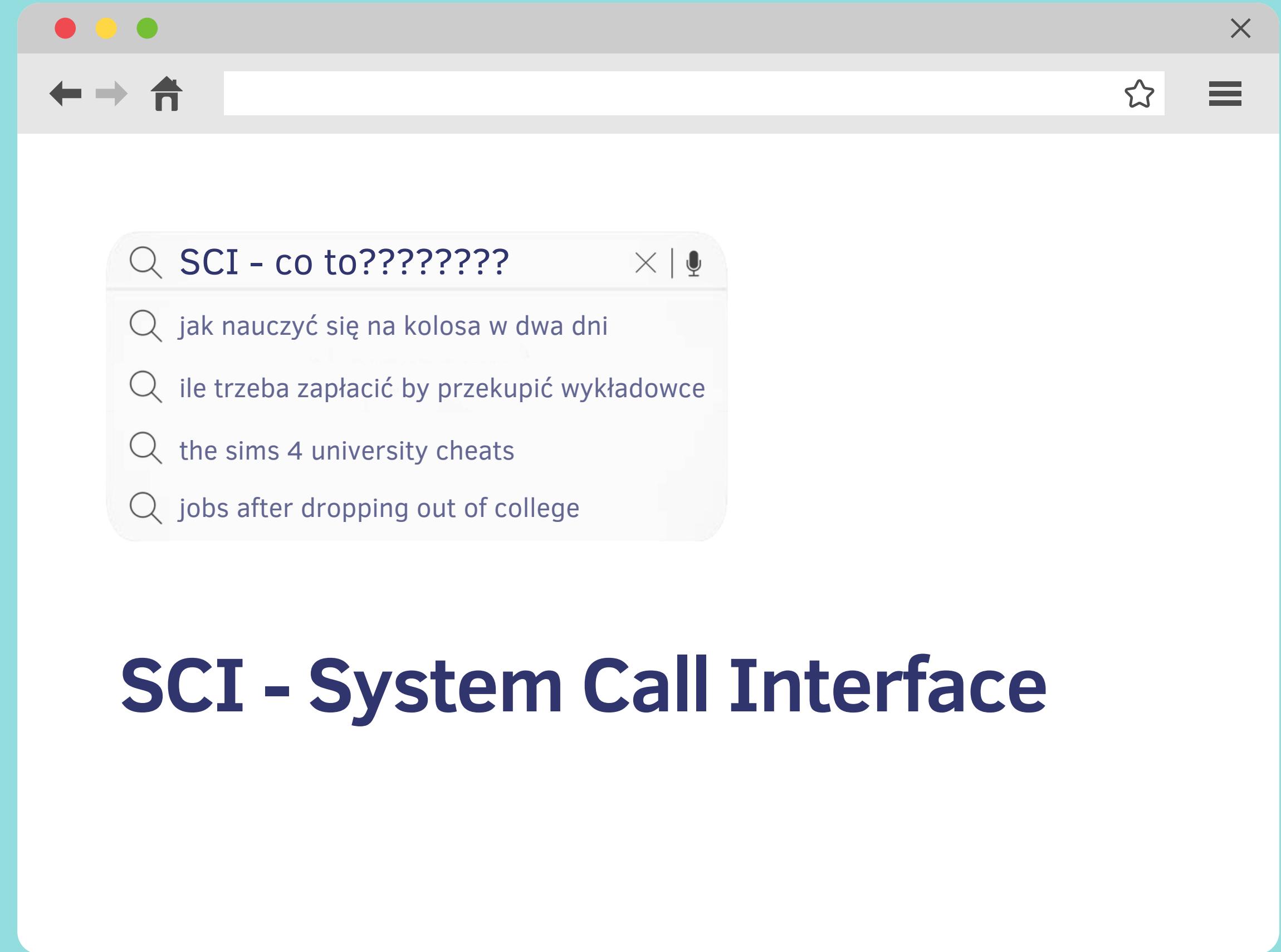
- **kursor up/down** - ostatnie komendy
- **cd** - zmiana katalogu (CWD - Current Working Directory):
 - bez parametru przenosi nas do katalogu domowego użytkownika
 - możemy przejść do poprzedniego katalogu przy użyciu .
- przekierowanie na wejście pliku (<) lub tekstu (<<)
- przekierowanie wejścia do pliku (> oraz >>)
- przekierowanie konkretnych deskryptorów (**2 >** oraz **2 > &1**)
- połączenie komand potokiem (|)



- wykonanie warunkowe (&& oraz ||)
- wykonanie w tle (&)
- podgląd procesów (**top**, **ps**)
- wyświetlanie plików (**cat**, **head**, **tail**, **less**, **more**, **most**)
- edytory tekstu (**nano**, **emacs**, **vi(m)**)
- **screen** - multiplekser terminala
- **script** - zapis przebiegu sesji (wraz z kolorami)

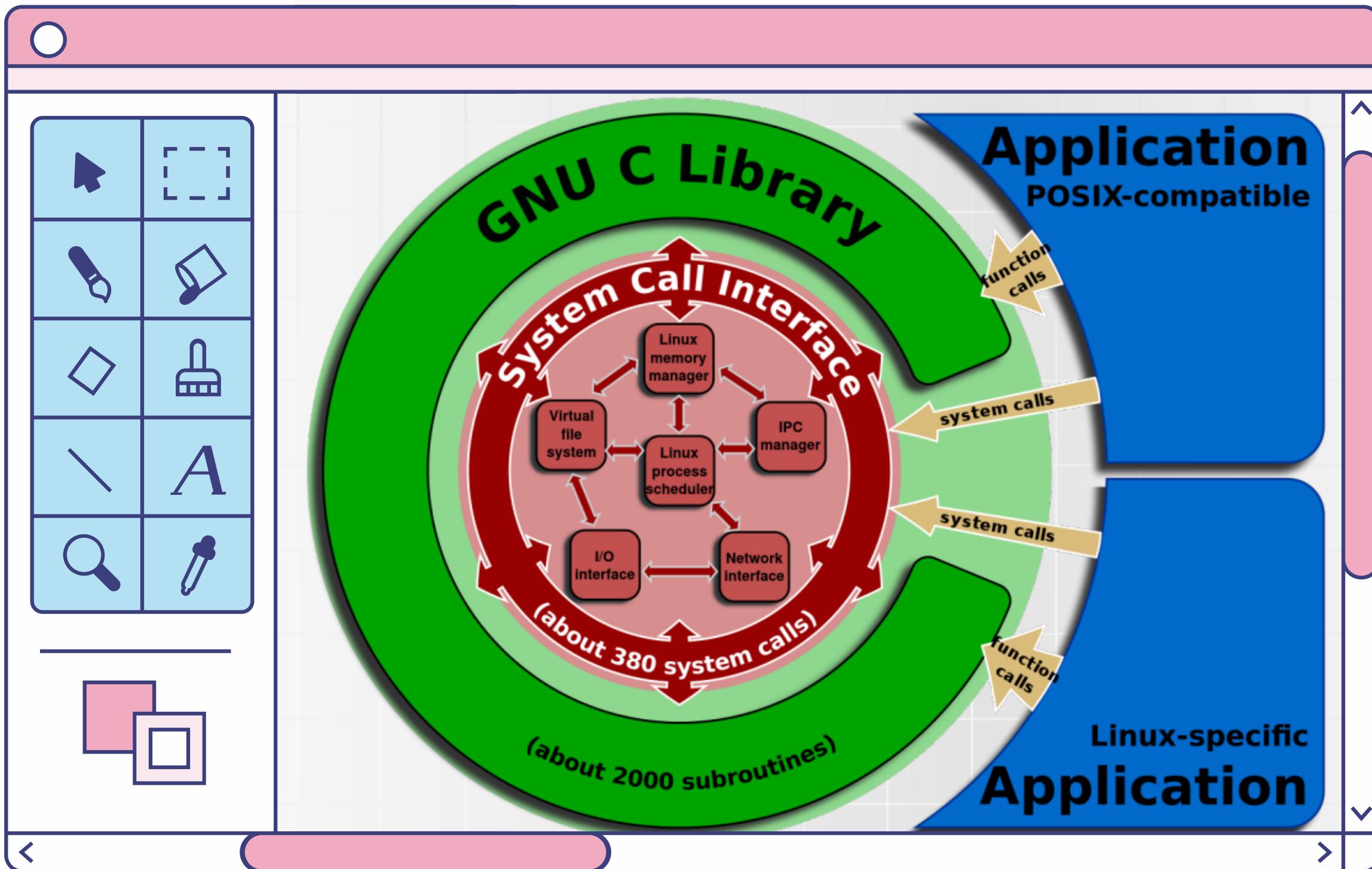
Praca z powłoką





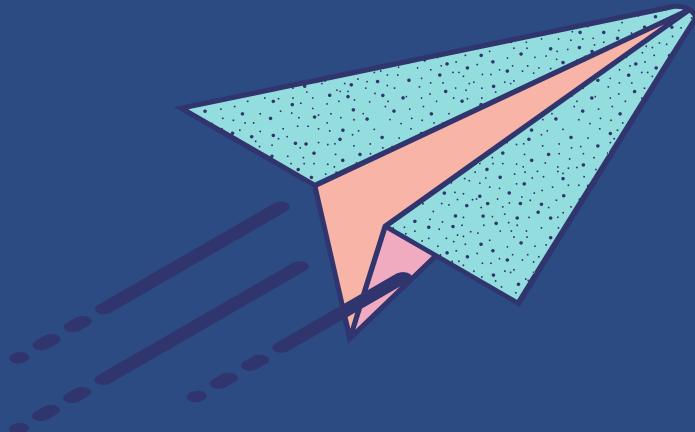
SCI - System Call Interface

System Call Interface



- API do wywoływania kodu jądra
- Z przestrzeni użytkownika (userspace)
- Biblioteki pośrednie (glibc)
- Najczęściej realizowane poprzez „zwykłe” wywołania funkcyjne w bibliotece uruchomieniowej (wrappery)
- Stosowane przez powłokę, programy systemowe oraz programy użytkowe

Powłoka jądra



1

Dowolne oprogramowanie pośredniczące pomiędzy użytkownikiem a systemem operacyjnym.



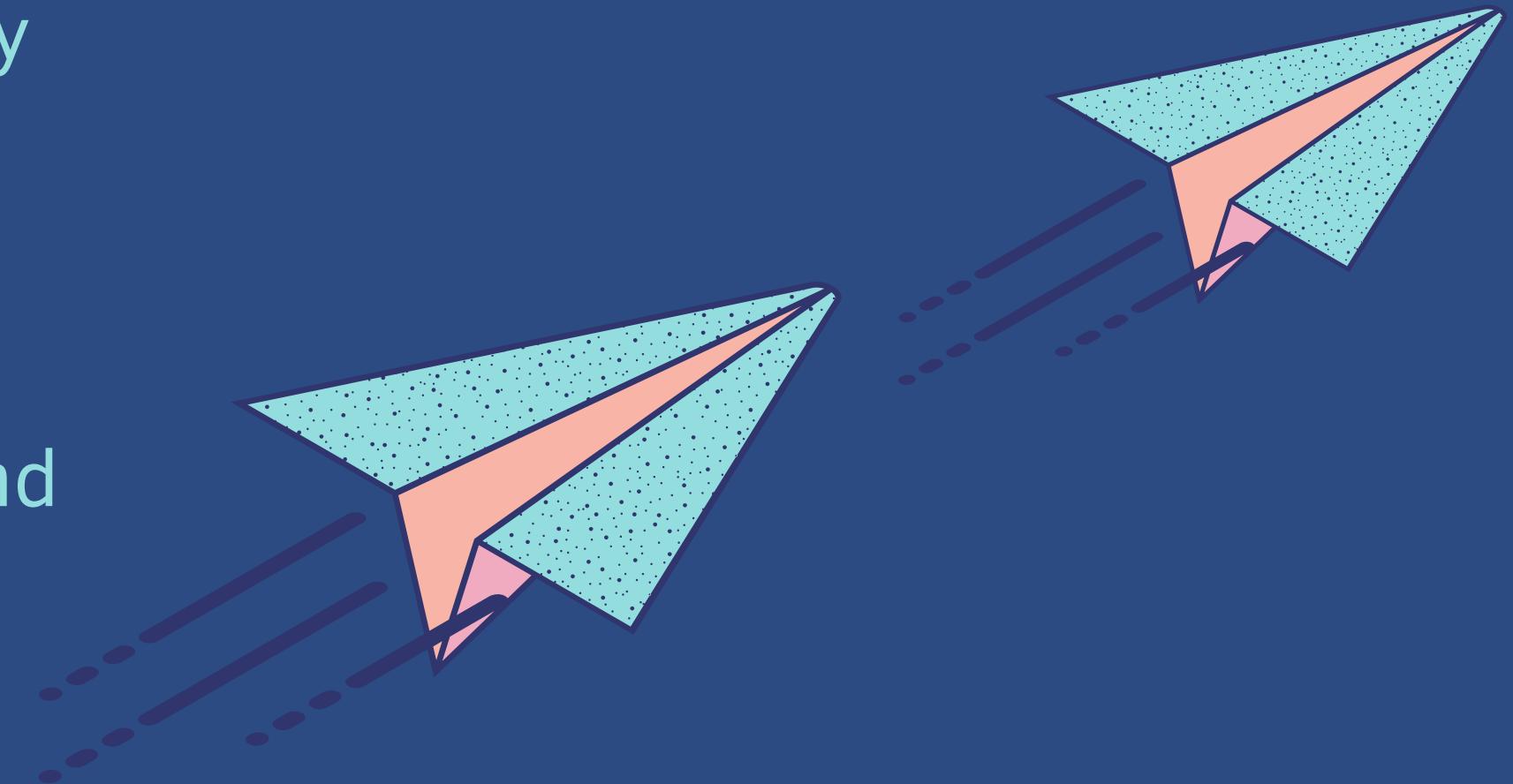
Forma interfejsu dowolna: **wierszowy (CLI)**, **tekstowy (TUI)**, **graficzny (GUI)**

2

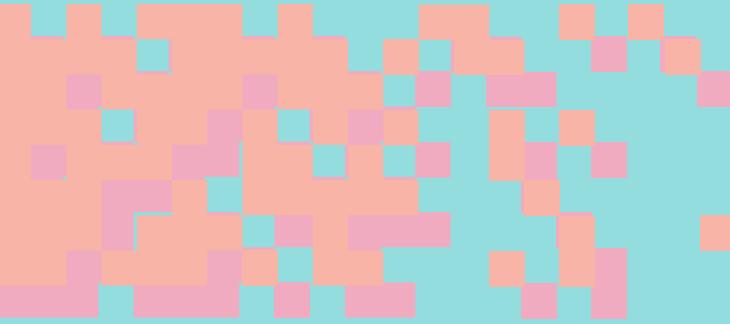
Oprogramowanie pośredniczące pomiędzy użytkownikiem a systemem operacyjnym w postaci interfejsu wiersza poleceń.

3

W wąskim znaczeniu to interpreter komend (skryptowy język programowania).



Powłoki wiersza poleceń



Interfejs wiersza poleceń (Command-line Interface, CLI):

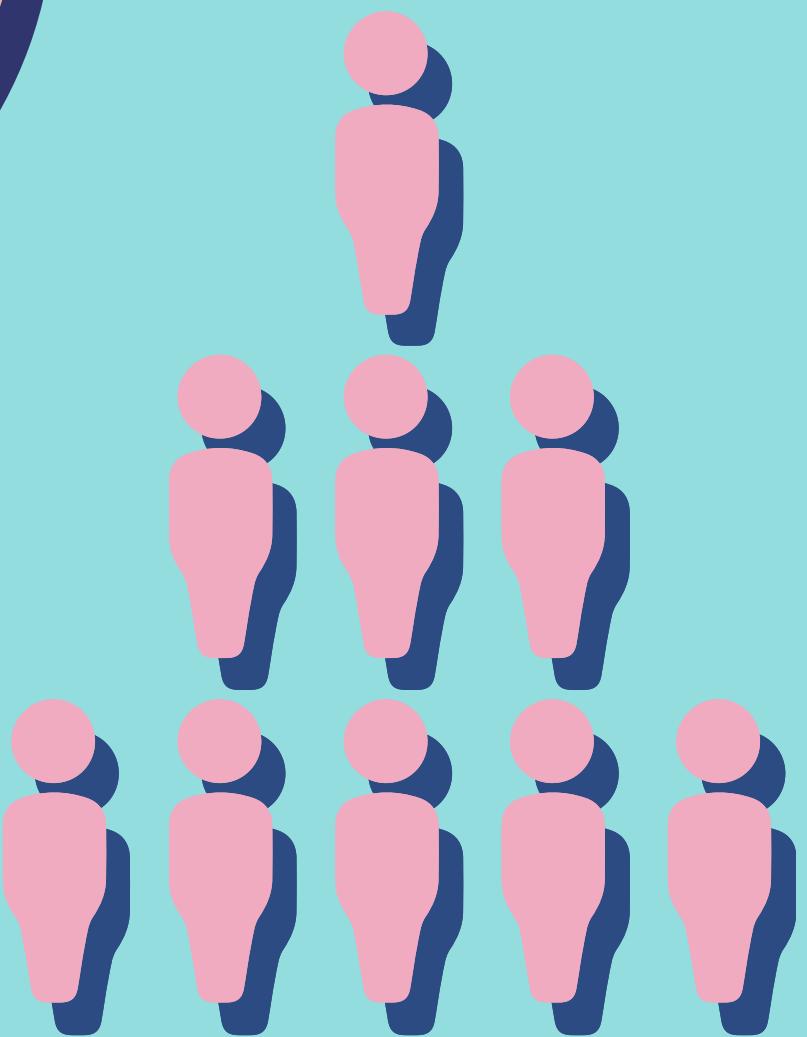
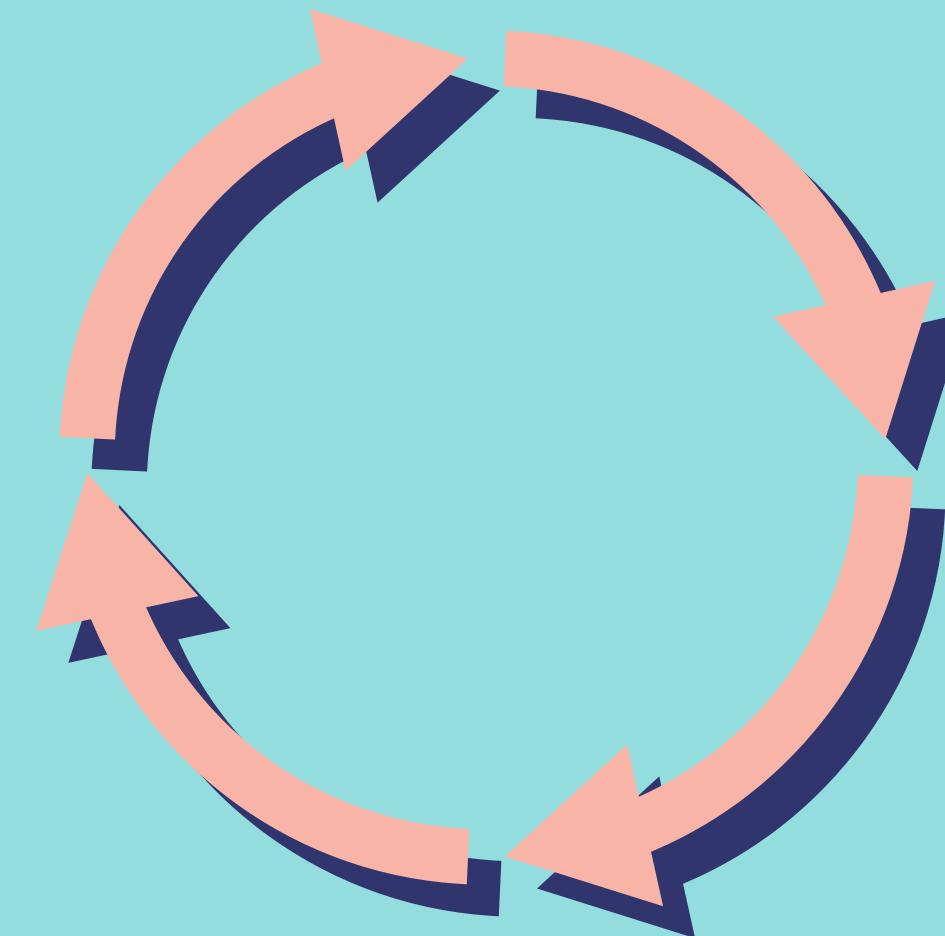
- ▶ **terminale** (np. /dev/tty2),
- ▶ **pseudoterminale** (np. /dev/pts/0),
- ▶ **okna terminala** (np. GNOME terminal, xterm).

Interpretery wiersza poleceń:

- ▶ **Bourne shell** (sh),
- ▶ **Bourne-Again shell** (bash),
- ▶ **Korn shell** (ksh),
- ▶ **Almquist shell** (ash) – podstawa dla powłok dla FreeBSD i NetBSD,
- ▶ **Z shell** (zsh) – dla macOS,
- ▶ **C shell** (csh) – bazowany na języku C.

Wykład 3

Procesy (stan, cykl życia, hierarchia)

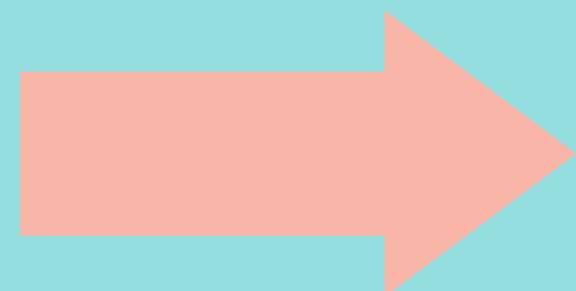


Proces
Egzemplarz
uruchomionego
programu.



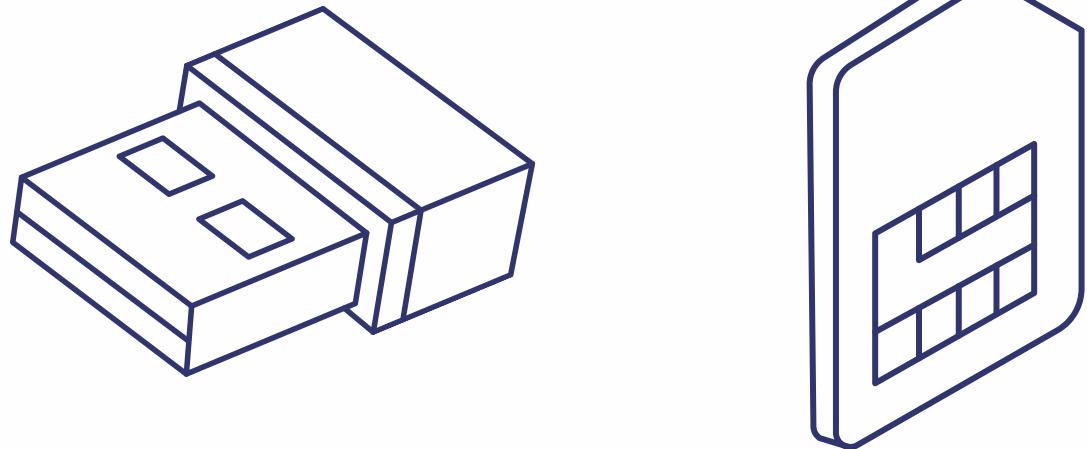
**Program ma „tylko” instrukcje i
wartości zmiennych globalnych,
procesy mają dużo więcej.**

Użytkownik

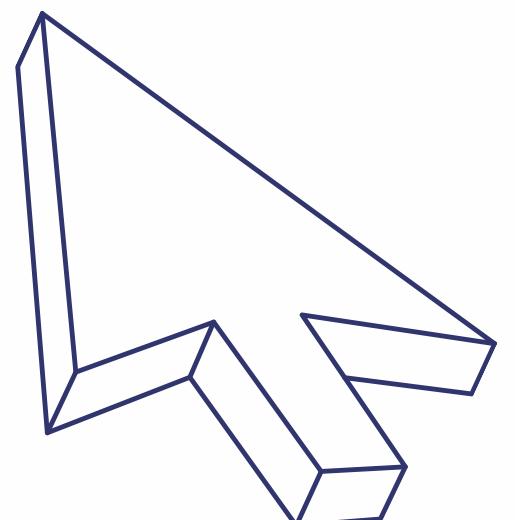


**Podmiot (tożsamość) o unikatowej
nazwie/numerze (UID).
Użytkownicy korzystają z systemu
informatycznego i jego zasobów.**

Informacje o procesie



- **PID.**
- **Informacje o użytkowniku (UID, GID itd.).**
- Wątek główny i lista wątków.
- Rodzic, procesy potomne, procesy siostrzane.
- Aktualny katalog (CWD).
- **Stan, priorytet.**
- Stan rejestrów.
- Informacje o przydzielonej pamięci (stronach).
- Limity pamięci.
- Segment (kod, dane), wskaźnik stosu.
- Licznik programu (PC/IP).
- Czas procesu (walltime, user pace, kernel space).
- Deksryptory (otwarcie pliki).
- Informacje dla planisty (w tym priorytet, pozostały czas itd.).
- Informacje statystyczne.
- Nieobsłużone sygnały.
- Uprawnienia.



SPRAWDŹ SIĘ!

Każdy proces jest identyfikowany
przez unikalną liczbę zwaną PPID.

TAK

NIE



SPRAWDŹ SIĘ!

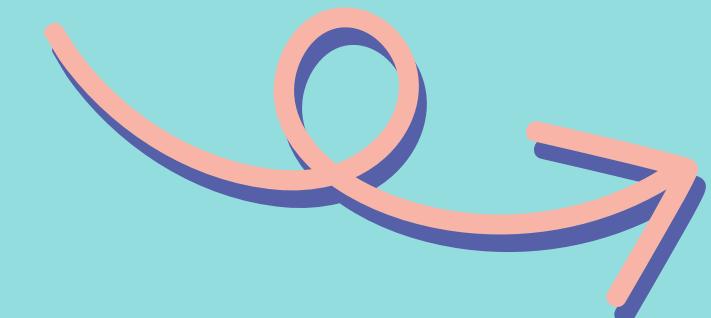
Każdy proces jest identyfikowany
przez unikalną liczbę zwaną PPID.

TAK



Stany procesu

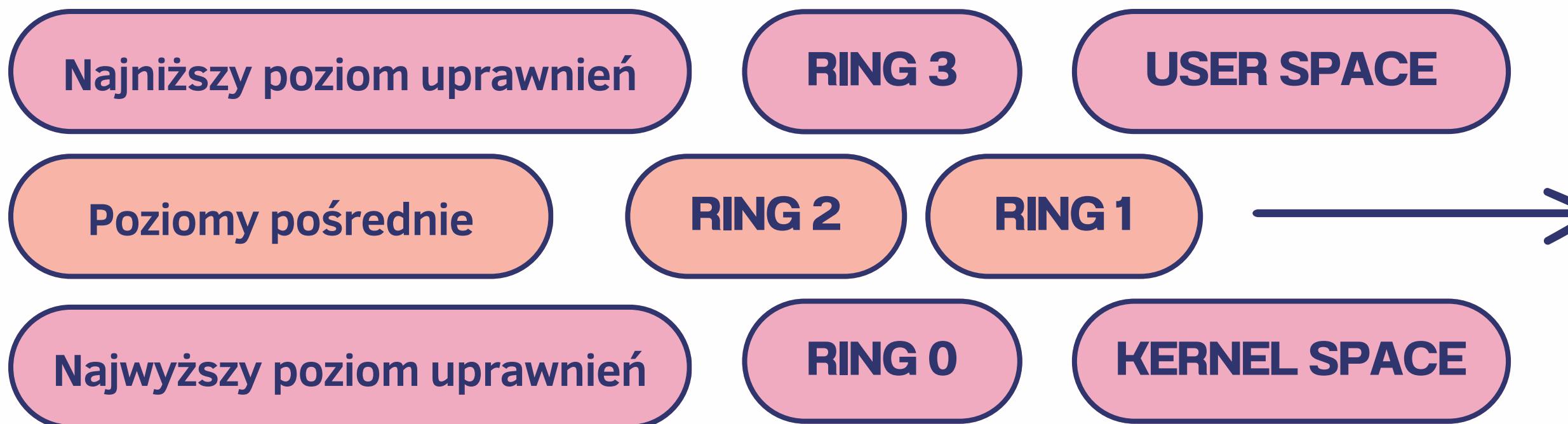
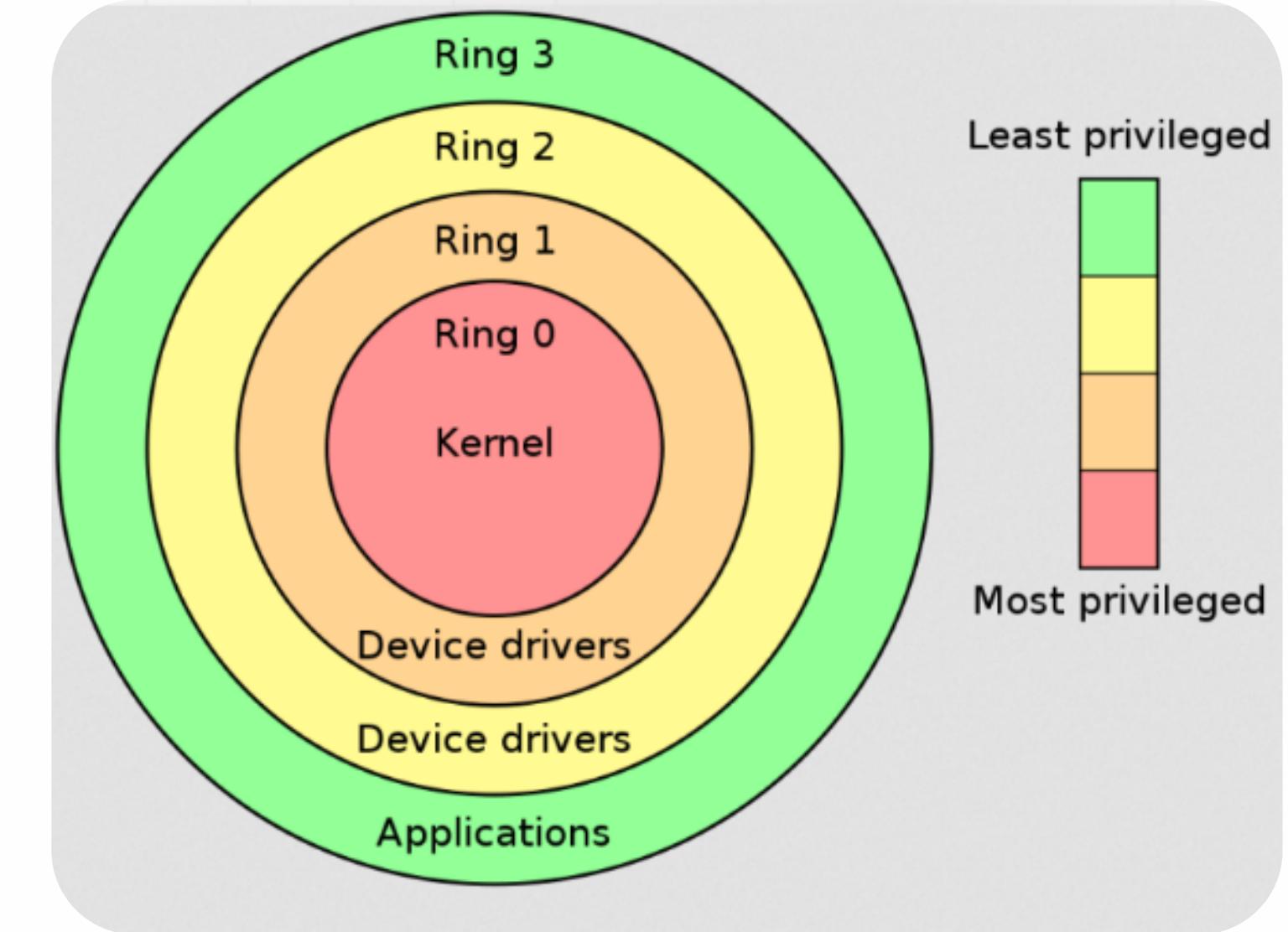
1. Utworzony
2. Gotowy w kolejce (“czekający”)
3. Wykonywany
4. Zablokowany / czekający
5. Zakończony
6. Swapped out



Swapped out to stan, w którym proces został przeniesiony z RAM do swap space na dysku twardym. Gdy proces zostanie wznowiony, jego zawartość została ponownie przeniesiona do przestrzeni RAM. Używane w celu optymalizacji zarządzania pamięcią RAM.

POZIOMY UPRAWNIĘŃ

- Wymaga wsparcia sprzętowego (CPU mode).
- Definiują do czego proces (kod) ma dostęp.
- Przełączanie trybu jest kosztowne (setki instrukcji maszynowych).
- “Ujemne” pierścienie (hypervisor).



Stany procesu



Tablica procesów i PCB

- System operacyjny przechowuje podstawowe informacje o procesach w tzw. **tablicy procesów**.
- Rozmiar tablicy jest **ograniczony**, wartość w /proc/sys/kernel/pid max.
- Procesy są **tworzone, wykonywane, kolejkowane, usypiane, zatrzymywane i zakańczane** z pomocą systemu operacyjnego. Wskaźniki na Process Control Block dla każdego procesu.

SPRAWDŹ SIĘ!

W Linuksie proces czekający na operacje I/O znajduje się w stanie:
(A) STOPPED, (B) SLEEPING.

A

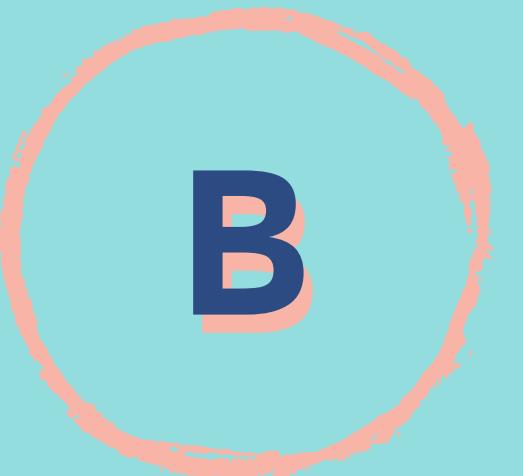
B



SPRAWDŹ SIĘ!

W Linuksie proces czekający na operacje I/O znajduje się w stanie:
(A) STOPPED, (B) SLEEPING.

A



Wywołania Systemowe

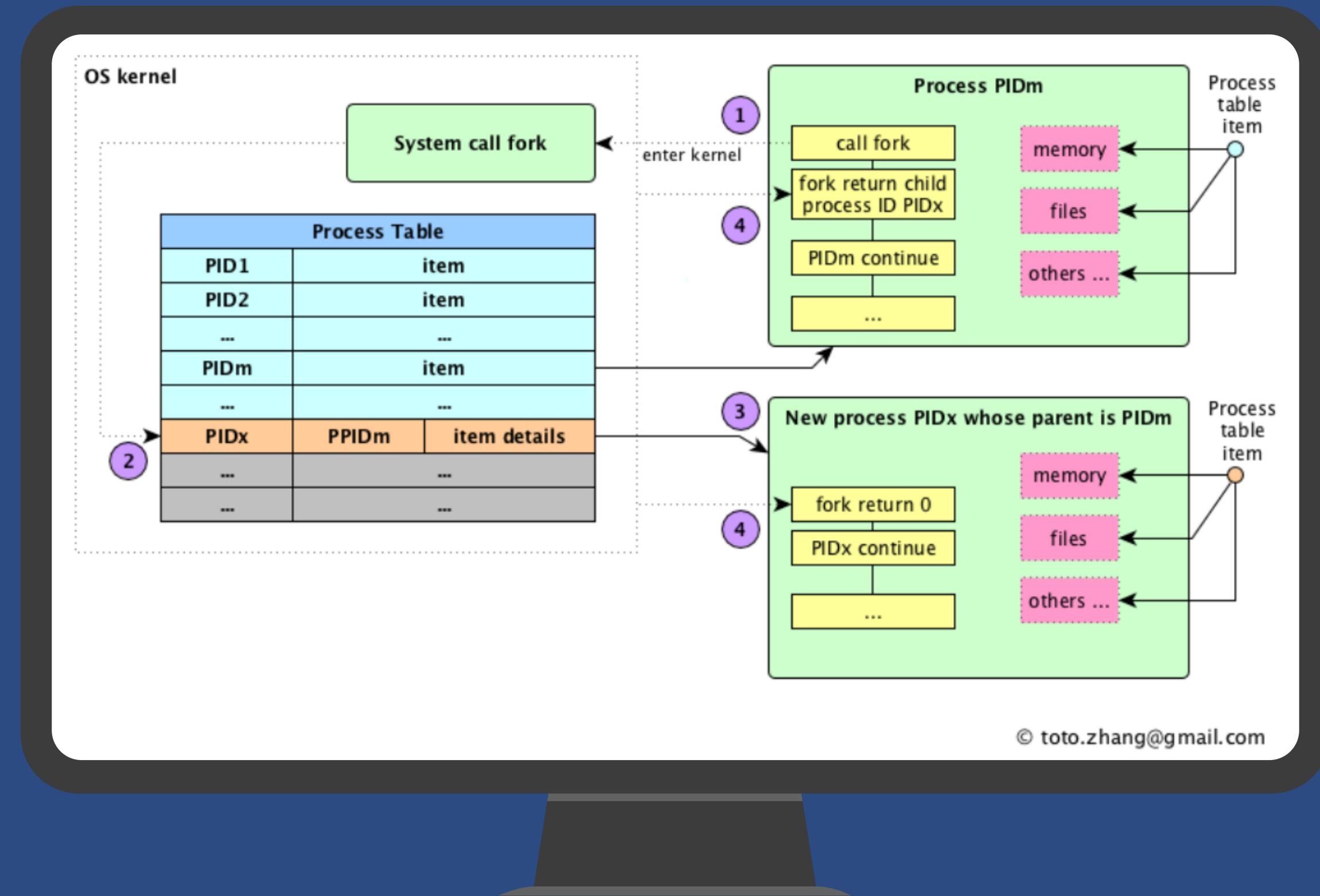
- Argumenty i numer wywołania do rejestrów.
- Przerwanie (int 0x80), przejście do trybu jądra.
- Kernel wywołuje własne system call().
- Argumenty na stos jądra oraz sprawdzenie.
- Wykonanie zadania (wartość zwracana).
- Wrócenie do wrappera, przejście do trybu użytkownika (ustawienie errno).

**Wywołanie
wrappera (glibc),
argumenty na
stosie.**

Wywołanie systemowe - fork

Tworzy nowy proces
*(osobny wpis
w tablicy procesów)*

- Proces potomny jest prawie idealną kopią procesu rodzica.
- Współdzielone deskryptory plików.
- Różna wartość zwracana w obu procesach.
- Dokładniejsza kontrola poprzez clone.

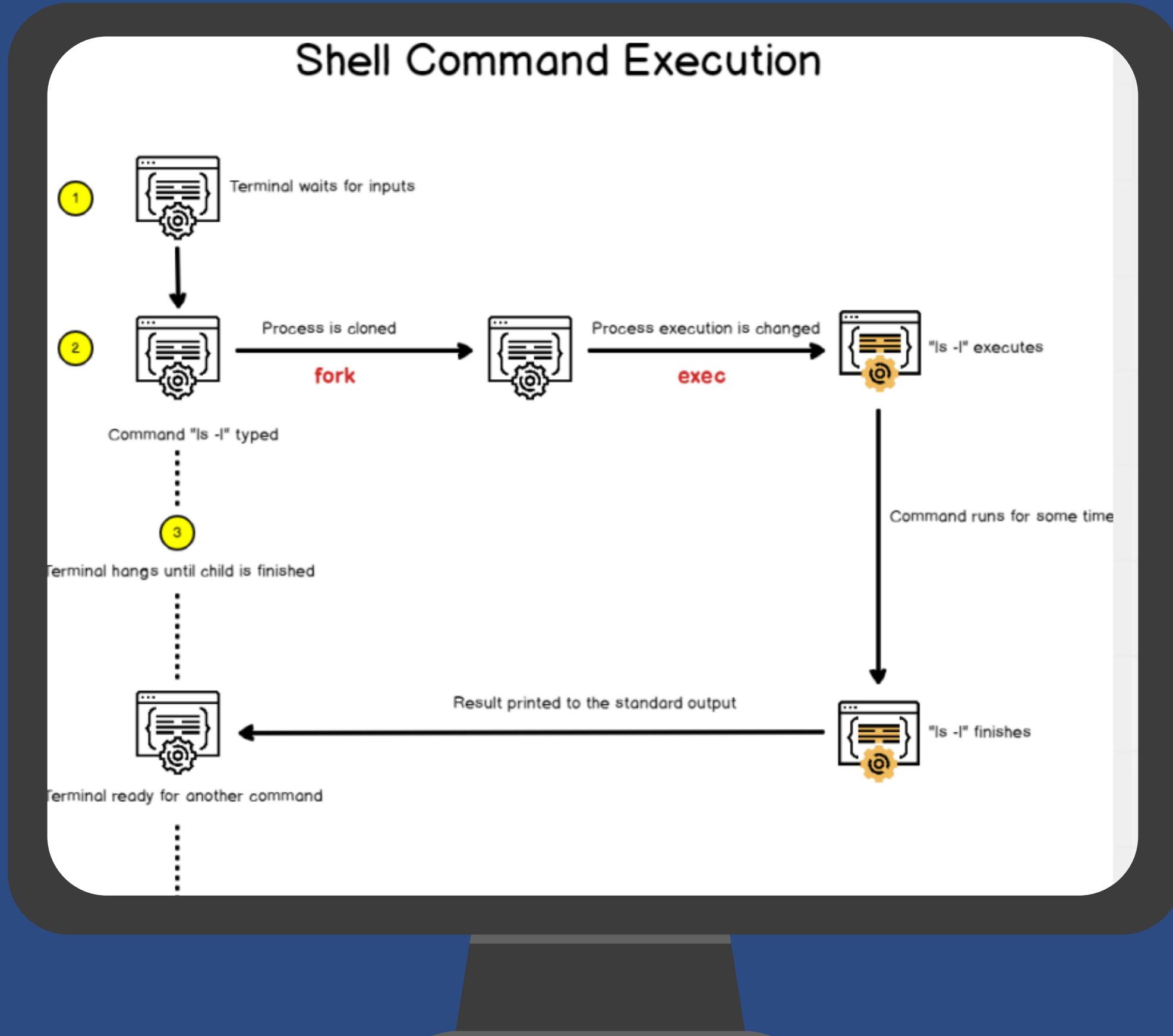


Wywołanie systemowe - exec

Zastąpienie programu innym.

- Często łączone z fork.
- Przekazywanie argumentów.
- Wartość zwracana.
- Wiele wariantów.
- Możliwość przekazania własnego środowiska (zmiennych).
- Możliwość wykorzystania PATH

*Wywołanie system – wygodne,
ale wolniejsze i mniej bezpieczne.*



SPRAWDŹ SIĘ!

W Linuksie wywołania systemowe są interfejsem do komunikacji z usługami jądra, z którego jednakowo korzystają programy przestrzeni użytkownika i inne usługi jądra.

TAK

NIE



SPRAWDŹ SIĘ!

W Linuksie wywołania systemowe są interfejsem do komunikacji z usługami jądra, z którego jednakowo korzystają programy przestrzeni użytkownika i inne usługi jądra.

TAK





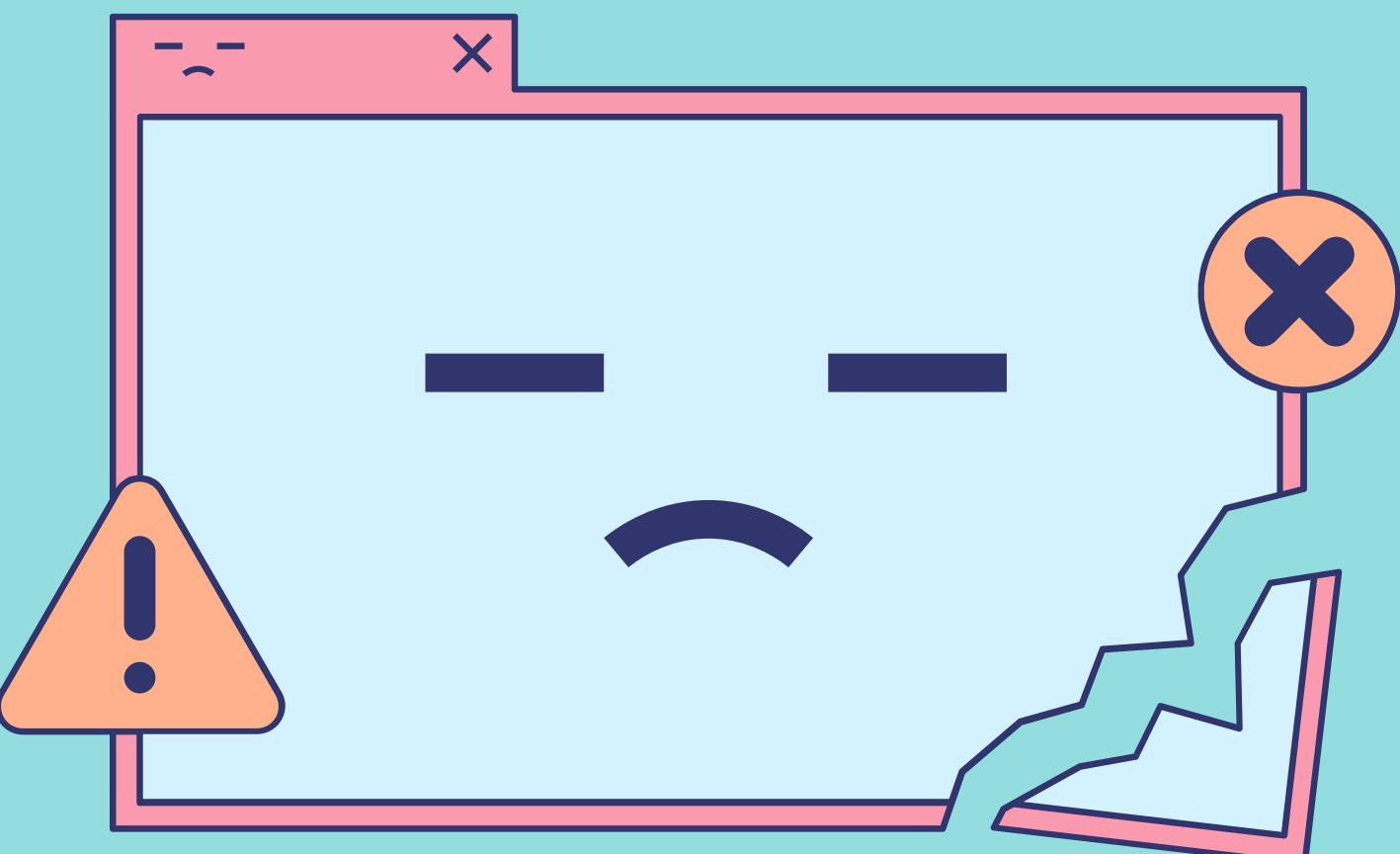
Co gdy proces się kończy?

→ PRZYCZYNY

- Poprzez wywołanie systemowe exit() (jawne lub niejawne).
- Ostatni wątek się kończy lub wykona pthread exit().
- Poprzez wywołanie abort() (SIGABRT).
- Poprzez niektóre nieobsłużone sygnały.

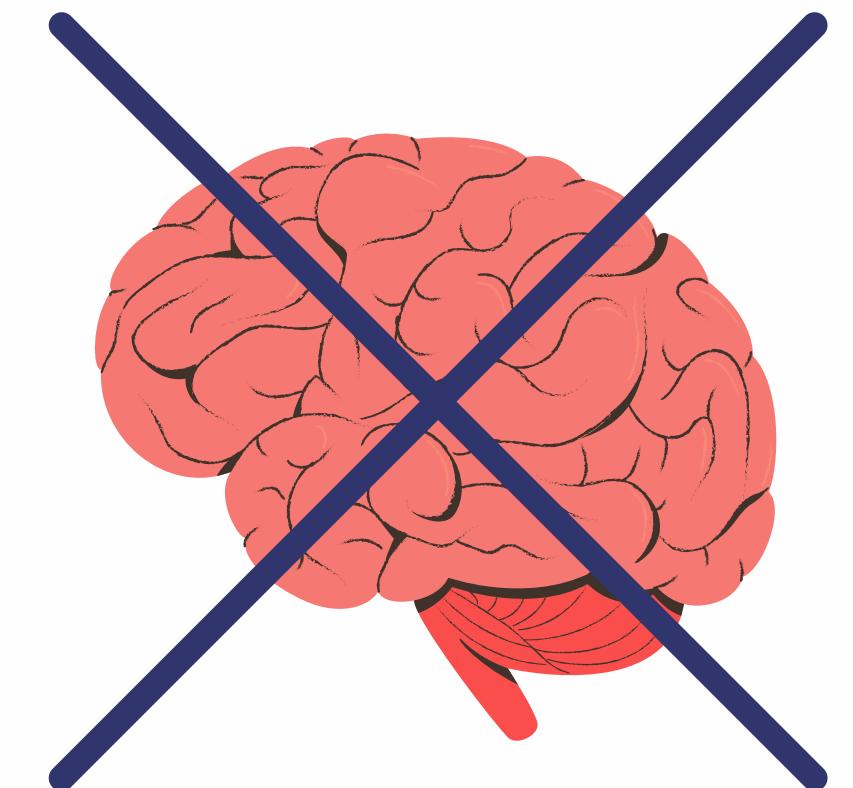
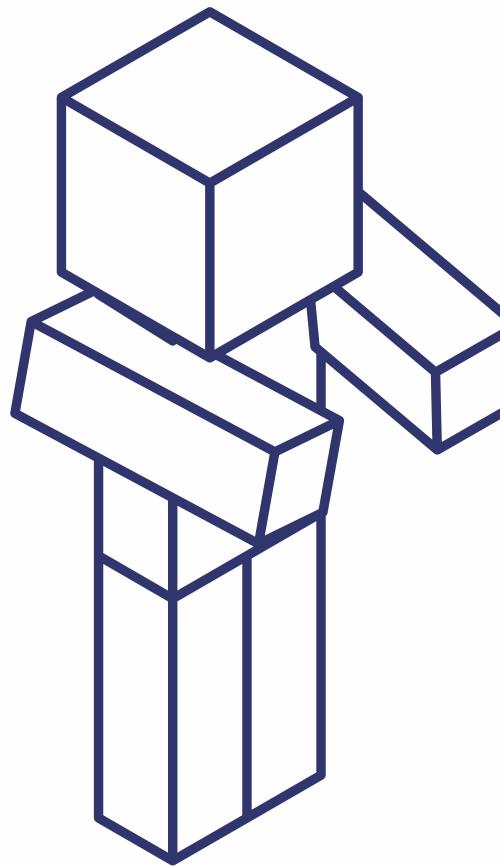
→ EFEKTY

- Przejście do stanu terminated.
- Wysłanie SIGCHLD do procesu rodzica.
- Zwolnienie zasobów, ale **nie wpisu w tablicy procesów!**



PROCES ZOMBIE

Proces zombie to wpis w tablicy opisujący program, którego wykonanie w systemie operacyjnym zostało zakończone, ale którego zamknięcie nie zostało jeszcze obsłużone przez proces rodzica.



Zakończony proces wciąż zajmuje wpis w tablicy procesów (proces zombie)

Wpisy są małe, czyli brak problemów?

Duża liczba procesów zombie może prowadzić do destabilizacji systemu, gdy cała tablica procesów posiadająca ograniczony rozmiar zostaje zajęta przez wpisy zombie.

Czy proces zombie można zabić poprzez sygnal SIGKILL?

Proces zombie jest już martwy, więc SIGKILL nie ma na niego wpływu. SIGKILL kończy proces, ale nie usuwa jego wpisu w tablicy procesów.

Rodzic musi wykonać wait/waitpid. Rodzic zawieszony funkcją wait/waitpid czeka na exit status procesu potomnego. Następnie usuwany jest wpis procesu zombie z tablicy.

SPRAWDŹ SIĘ!

Procesy zombie wciąż posiadają przydzielone zasoby (w tym pamięć), których nie mogą użyć inne procesy.

TAK

NIE



SPRAWDŹ SIĘ!

Procesy zombie wciąż posiadają przydzielone zasoby (w tym pamięć), których nie mogą użyć inne procesy.

TAK



Hierarchia procesów

Procesy systemu operacyjnego startują z poziomu kodu jądra.

Pierwszym procesem (PID=1) jest init:

- 1 /sbin/init (często dowiązaniem np. na popularnego demona systemd),
- 2 wszystkie inne procesy są potomkami (bezpośrednimi lub nie) init,
- 3 osierocone procesy stają się potomkami któregoś init,
- 4 do init (PID=1) można wysyłać tylko sygnały, które on obsługuje.

Każdy terminal ma proces jego obsługi (getty):

- 1 Proces getty wykonuje exec /bin/login wraz z naszym loginem,
- 2 Proces login przeprowadza uwierzytelnianie, ustawia środowisko (UID, katalog domowy, powłoka itp.) i wykonuje exec powłoka.
- 3 Procesy zalogowanych użytkowników stają się potomkami powłoki.

Konfiguracja init

Konfiguracja znajduje się w /etc

np. /etc/inittab

Format wpisów w inittab:

id:poziomy:akcja:proces

Niektóre dostępne „akcje”:

respawn

wait

once

bootwait

boot

sysinit

initdefault



SPRAWDŹ SIĘ!

W Linuksie niemożliwe jest istnienie wielu procesów typu init (lub jego odpowiedników jak systemmd).

TAK

NIE



SPRAWDŹ SIĘ!

W Linuksie niemożliwe jest istnienie wielu procesów typu init (lub jego odpowiedników jak systemmd).

TAK



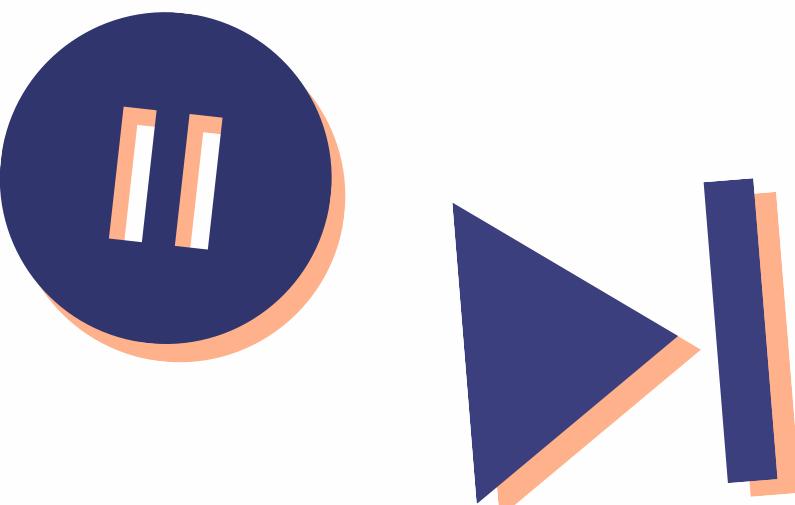
POZIOMY URUCHOMIENIA

Typowe znaczenie poziomów 0–6 dla Linuksa:

- 0 – halt,
- 1 – jeden użytkownik (*cele administracyjne*),
- 2 – wielu użytkowników, bez sieci,
- 3 – standard bez GUI (*wielu użytkowników, sieć*),
- 4 – zdefiniowany przez użytkownika,
- 5 – podobnie jak poziom 3, ale dodaje GUI,
- 6 – reboot.

- Poziom domyślny określony przez `initdefault`.
- Spotyka się też poziom **S** lub **s** (zwykle oznacza poziom 1 lub podobny).
- Do zmiany poziomu uruchomienia służy komenda `telinit`.

Przełączanie kontekstu



Zapisanie pewnych elementów stanu zadania (w szczególności wartości rejestrów procesora), by móc je później **odtworzyć** i **wznowić zadanie**.

Zadaniem **nie musi być proces**. Przełączanie jest względnie **kosztowne**.

Kiedy mamy do czynienia z przełączeniem kontekstu? 

Z przełączeniem kontekstu mamy do czynienia w przypadku m.in.:

- Wywołania systemowego (*dyskusyjne*).
- Przerwanie (*dyskusyjne*).
- Przełączania wątku.
- Przełączania procesu.

PRZYPADEK

1

wywołanie systemowe

- Zachowanie kontekstu (część rejestrów procesora).
- Wykonanie kodu (usługi) jądra.
- Przywrócenie kontekstu.
- Powrót do oryginalnego kodu.

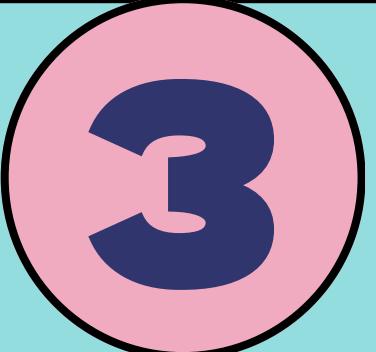
PRZYPADEK

2

przerwanie

- Zachowanie kontekstu (część rejestrów procesora).
- Obsługa przerwania.
- Przywrócenie rejestrów.
- Powrót do poprzednio wykonywanego kodu.

PRZYPADEK



przełączenie wątku

- Zachowanie kontekstu (rejestry, PC, stos itd) starego wątku.
- Wczytanie kontekstu dla nowego wątku.
- Udział planisty.

PRZYPADEK

4

przełączanie procesu

- Przełączanie wątku.
- Zapisanie kontekstu starego procesu.
- Wczytanie kontekstu nowego procesu.
- Wyczyszczenie Translation Lookaside Buffer.
- Ma dodatkowe konsekwencje.
- Wywołanie planisty.

Skrypty startowe

Skrypty startowe najczęściej umieszczone są w katalogu /etc/init.d.

Skrypty te uruchamiane są z argumentem start, stop, restart lub reload.

Które z powyższych skryptów należy wykonać na danym poziomie uruchomienia i w jakiej kolejności, precyzują katalogi

/etc/rcX.d gdzie X jest poziomem uruchomienia.

Katalog /etc/rcX.d zawiera dowiązania do części skryptów z /etc/init.d, lecz ze zmienionymi nazwami:

- ▶ pierwszy znak to S (start) lub K (kill/stop),
- ▶ dwa kolejne znaki to numer (kolejność uruchomienia),
- ▶ przykładowo **S19cron** oznacza uruchomienie skryptu cron.
- ▶ Komenda update-rc.d aktualnia dowiązania /etc/rcX.d.

Wykład 4

Szeregowanie procesów, planista



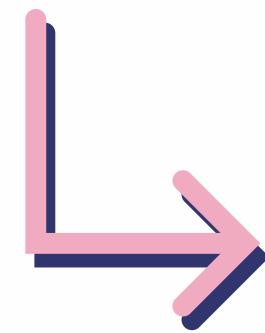
Czym jest planista?

Planista, lub inaczej Scheduler, to mechanizm systemu operacyjnego odpowiedzialny za przydział zadań (procesów) do zasobów (CPU).

Mechanizm SO odpowiedzialny za przydział zadań (procesów) do zasobów (CPU):

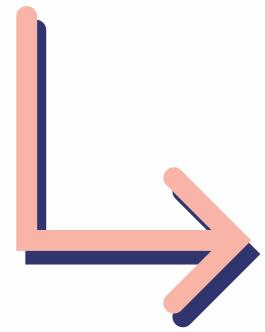
- Planista powinien działać szybko dla dużej liczby zadań
- Kontrola stopnia wielozadaniowości systemu

Rodzaje planistów



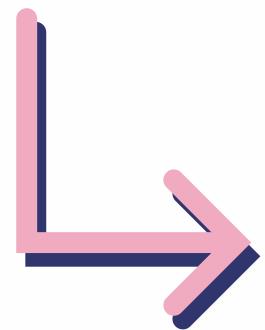
Planista długoterminowy

Long-Time Scheduler



Planista srednioterminowy

Medium-Term Scheduler



Planista krótkoterminowy

Short-Time Scheduler



PLANISTA DŁUGOTERMINOWY

LONG-TIME SCHEDULER

- Nazywany też job scheduler
- Decyduje czy i kiedy utworzone zadania przechodzą w stan gotowości
- Kontroluje poziom wielozadaniowości
- Istotny w RTOS
- Ograniczony lub nieistniejący w typowych systemach z podziałem czasu

PLANISTA ŚREDNIOTERMINOWY

MEDIUM-TIME SCHEDULER

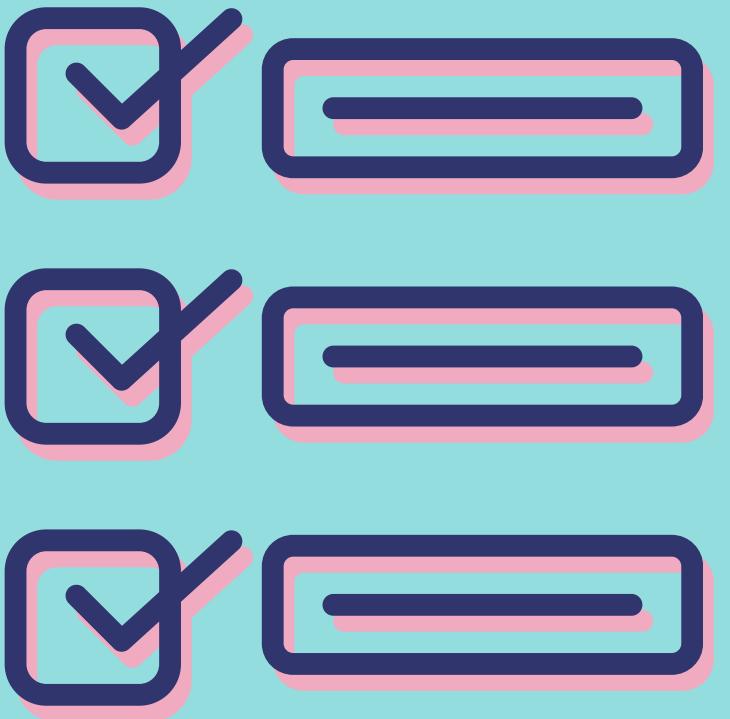
- Nazywany też swapping scheduler
- Decyduje o przeniesieniu procesów do pamięci masowej i z powrotem (swap out)
- Kontroluje poziom wielozadaniowości
- Często łączony z planistą długoterminowym
- Różne kryteria pracy

PLANISTA KRÓTKOTERMINOWY

SHORT-TIME SCHEDULER

- Nazywany też CPU Scheduler
- Decyduje które z gotowych zadań zostanie wykonane następne
- Może decydować o czasie wykonania (time slice)
- Konieczna duża wydajność dla dużej liczby zadań

Kryteria Szeregowania



- Maksymalizacja przepustowości (*throughput*)
- Minimalizacja czasu oczekiwania (*wait time*)
- Minimalizacja czasu odpowiedzi/przepływu (*response time/flow time*)
- Maksymalizacja uczciwości (*fairness*)
- Maksymalizacja wykorzystania CPU (*utilization*)

Strategia CFS Scheduler

- aktualny planista w jądrze Linux
- Completely Fair Scheduler
- maksymalizuje uczciwość (*fairness*) w dostępie do CPU
- Virtual Runtime (*ważony z użyciem dobroci*)
- wykorzystuje drzewo czerwono-czarne ($O(\log n)$)
- osobne kolejki dla poszczególnych CPU
- możliwość szeregowania i uczciwości dla grupy zadań



SPRAWDŹ SIĘ!

Planista CFS wykorzystuje drzewa czerwono-czarne.

TAK

NIE



SPRAWDŹ SIĘ!

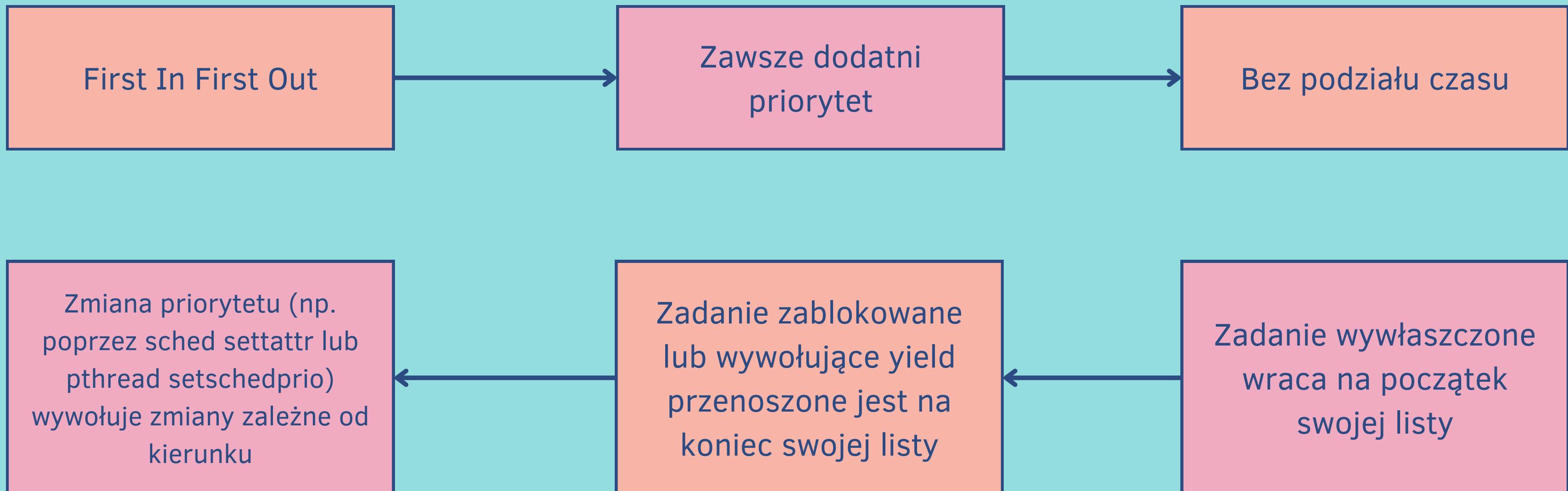
Planista CFS wykorzystuje drzewa czerwono-czarne.



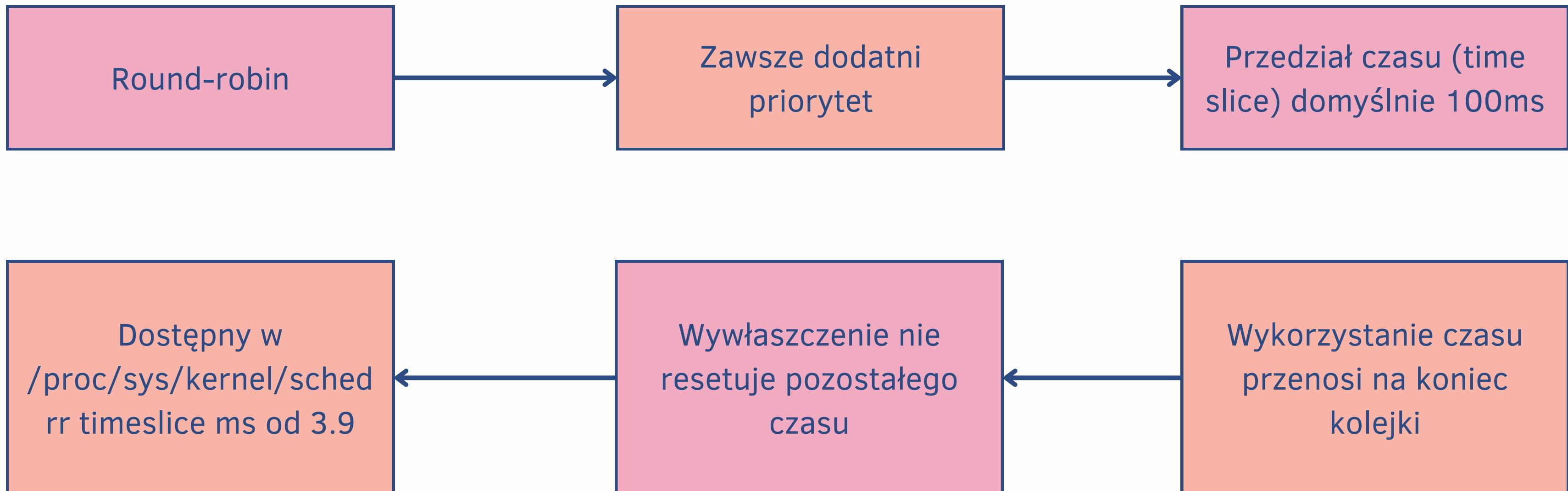
NIE



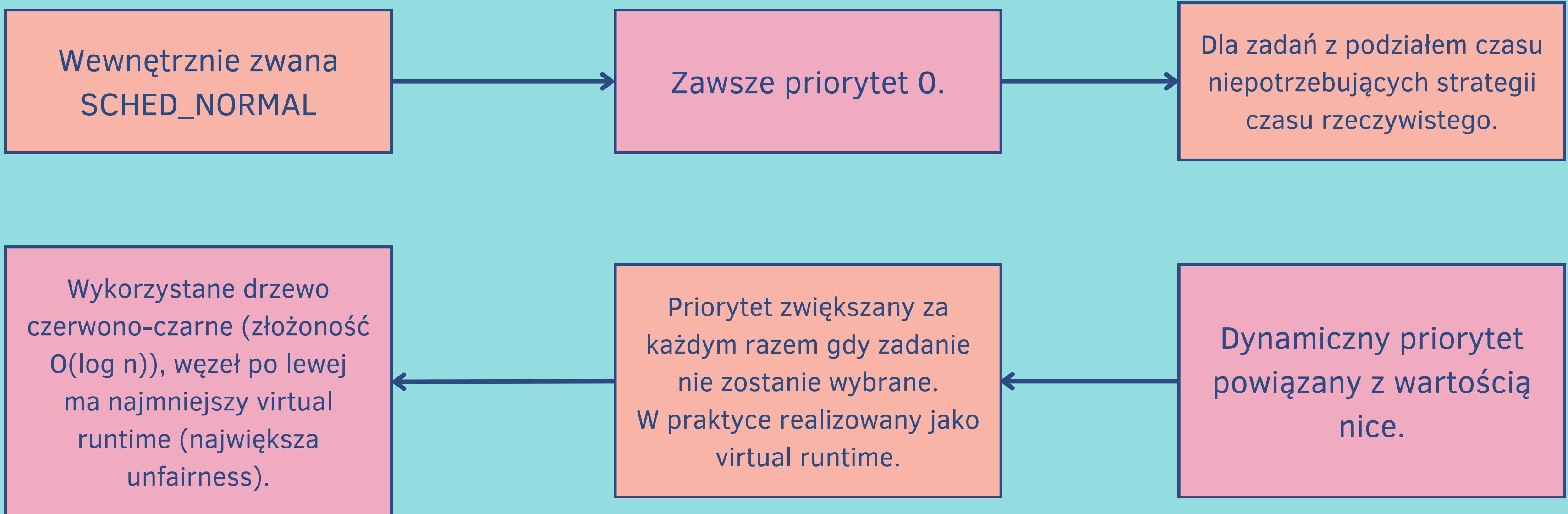
Strategia SCHED_FIFO



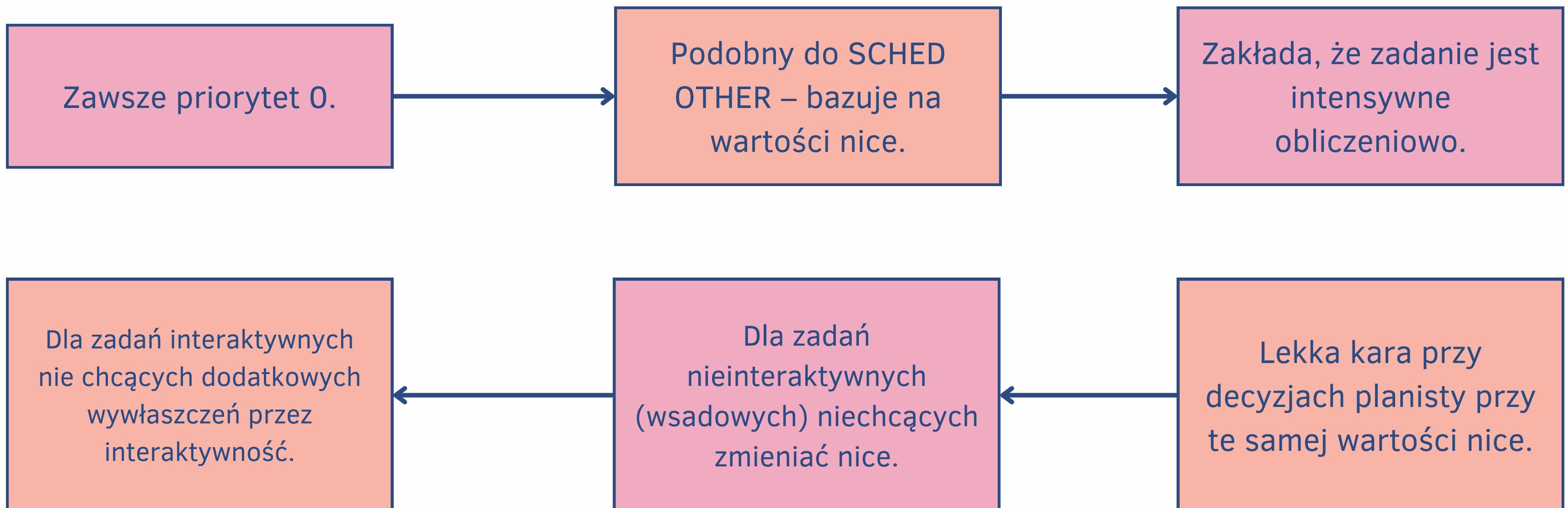
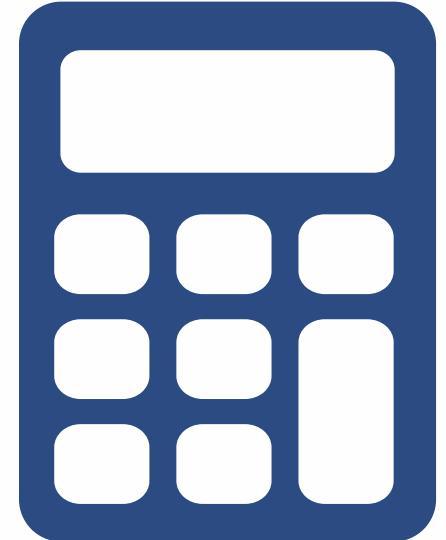
Strategia SCHED_RR



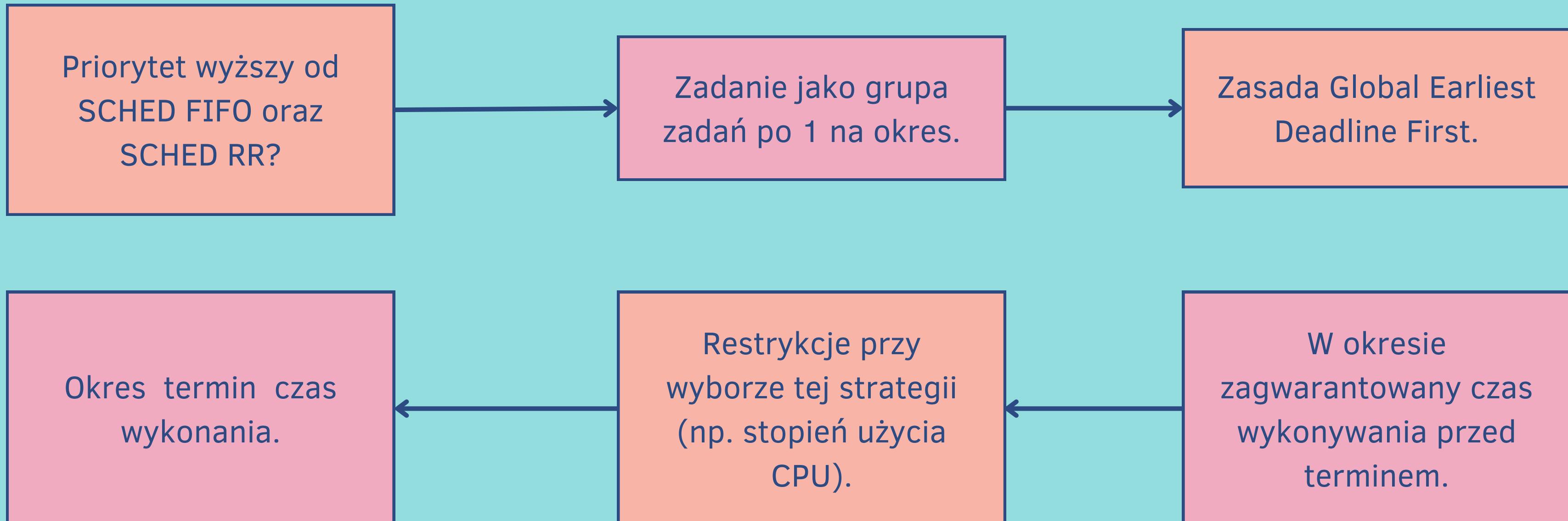
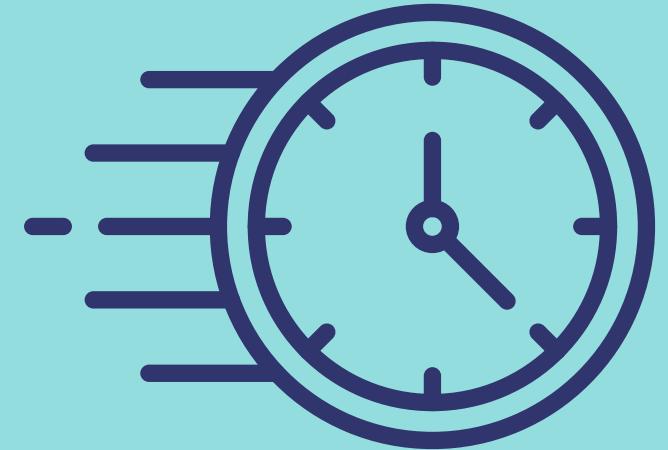
Strategia SCHED_OTHER



Strategia SCHED_BATCH



Strategia SCHED_DEADLINE



SPRAWDŹ SIĘ!

Głównym kryterium planisty CFS jest: (A) maksymalizacja Uczciwości, (B) minimalizacja czasu oczekiwania.

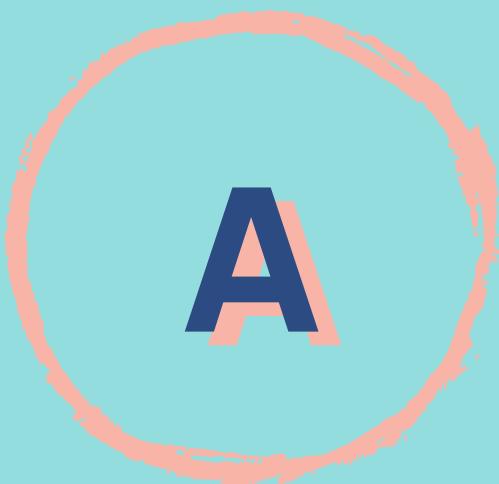
A

B



SPRAWDŹ SIĘ!

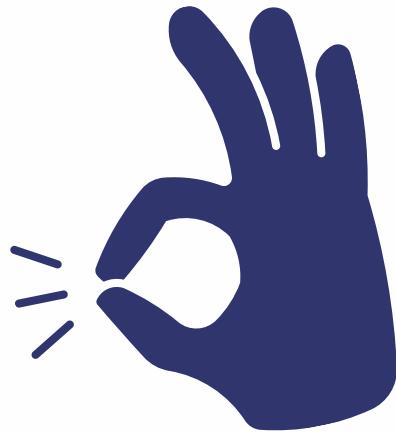
Głównym kryterium planisty CFS jest: (A) maksymalizacja Uczciwości, (B) minimalizacja czasu oczekiwania.



B



NICE



- Wartość nice jest potrzebna do ustalenia priorytetu, dla schedulerów SCHED_OTHER i SCHED_BATCH.
- Na podstawie tej wartości, schedulery ustalają priorytet, na podstawie którego przydzielają czas procesora procesom.
- Im mniejsza wartość nice, tym większy priorytet, na przykład proces z wartością nice -19 jest wybierany najchętniej.



Proces z wartością nice o 1 mniejszą, ma $1.25x$ większy priorytet.

Czyli na przykład proces z nice -19 jest wybierany $1.25x$ razy częściej niż proces z wartością -18, 1.25^2 razy częściej niż -17 itd.

Dlatego proces z wartością nice -19 jest wybierany 1.25^{39} (ok 6k) razy częściej niż proces z wartością 20.

SPRAWDŹ SIĘ!

Proces z wartością nice = -20 jest wybierany przez planistę około 6000x częściej niż proces z nice:
(A) 0, (B) 19.

A

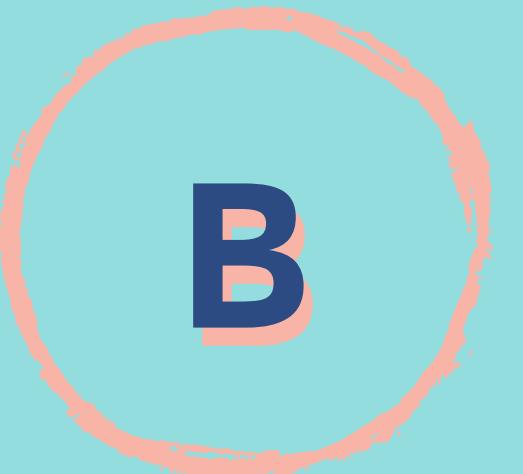
B



SPRAWDŹ SIĘ!

Proces z wartością nice = -20 jest wybierany przez planistę około 6000x częściej niż proces z nice:
(A) 0, (B) 19.

A



Wykład 5

Sygnały, zarządzanie procesami



POSIX

Portable Operating System Interface



Rodzina standardów dla zachowania zgodności pomiędzy systemami operacyjnymi (Unix) autorstwa IEEE.



Linuks od samego początku był tak rozwijany, aby być zgodny z standardem POSIX.

» Grupy i sesje procesów «

GRUPA PROCESÓW (*shell job*)

Zbiór (najczęściej spokrewnionych) procesów, które mogą wspólnie otrzymywać sygnały.

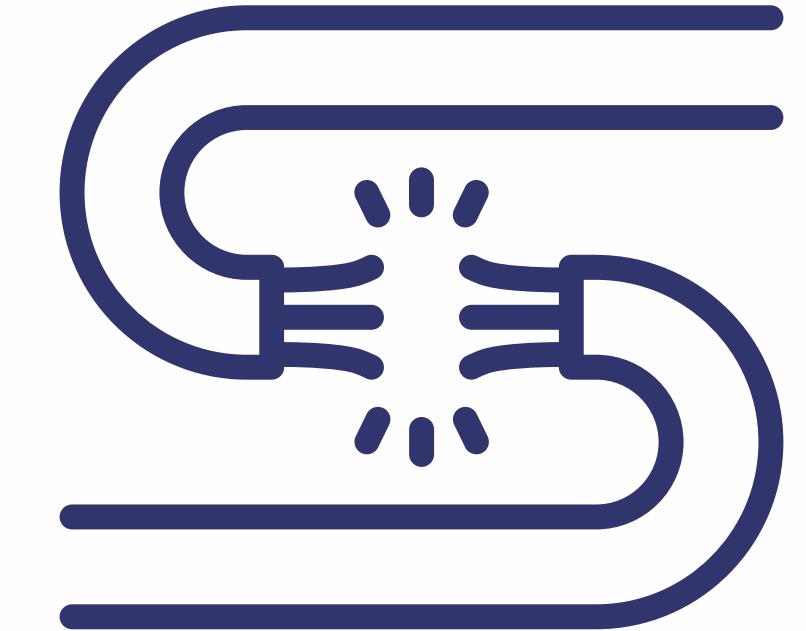
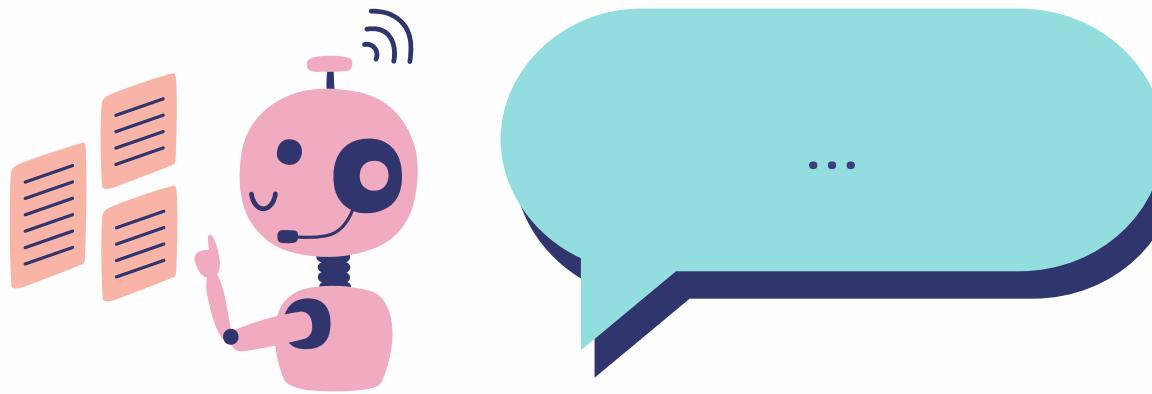
SESJA

Zbiór grup procesów albo podłączonych do wspólnego kontrolującego terminala albo niepodłączonych do żadnego.

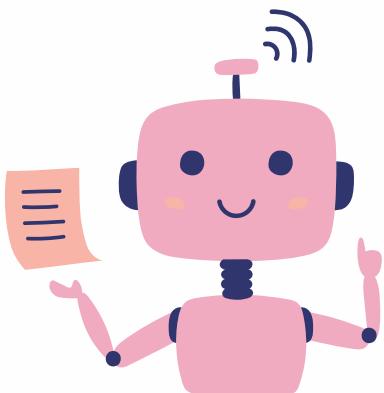
Grupy i sesje wpływają m.in. na sygnały (np. SIGINT, SIGTSTP, SIGHUP)
oraz kontrolę procesów pierwszego i drugiego planu.



Przerwania



- Asynchroniczne zdarzenia wymagające uwagi procesora.
- Przejście do kodu obsługi przerwania (*interrupt handler*).
- **Źródło:** sprzęt (np. gotowe dane, błąd) lub programy (np. *system calls*).
- **Mediacja:** procesor (*tablica przerwań*).
- **Obsługa:** system operacyjny (*tablica przerwań*).
- Przerwania maskowalne i niemaskowalne.



Sygnały

Sygnały to chyba
na innym
przedmiocie były...

My na pewno na
dobrym wykładzie
jesteśmy?

(Najczęściej) asynchroniczne zdarzenia wymagające uwagi procesu.

- Źródło: proces lub system operacyjny (często na skutek przerwania/błędu).
- Mediacja: system operacyjny.
- Obsługa: proces lub wątek (signal handler).
- Część stanowi minimalistyczną formę komunikacji międzyprocesowej (IPC).
- Może być atrakcyjna dla programów wbudowanych.

- Możliwość zablokowania sygnału (maska).
- Zwykłe sygnały nie kolejkują się podczas blokady.
- Oczekiwanie na sygnał ze zbioru: `sigwait()`/`sigtimedwait()`.

Wywołanie systemowe sigaction – możliwość ustawienia akcji (domyślna, ignorowanie, funkcja) i flag dla sygnału:

- Możliwość “systemowego” usunięcia procesów zombie.
- Wywołanie systemowe signal – podobne do sigaction, ale mniej przenośne.
- SIGKILL i SIGSTOP nie mogą być obsłużone, zignorowane lub zablokowane.

SPRAWDŹ SIĘ!

W Linuksie wszystkie sygnały
mogą zostać zignorowane lub
obsłużone.

TAK

NIE



SPRAWDŹ SIĘ!

W Linuksie wszystkie sygnały mogą zostać zignorowane lub obsłużone.

TAK



Lista wybranych sygnałów POSIX

1

NAZWA

NUMER

DOMYSLNA AKCJA

OPIS/UWAGI

SIGCHLD

-

brak

zmiana statusu procesu potomnego

SIGQUIT

3

zakonczenie (core dump)

zakonczenie od terminala (zwykle Ctrl+\)

SIGSTOP

-

zatrzymanie

zatrzymanie programu (nie można obsłużyć)

SIGHUP

1

zakonczenie

utrata terminala (daemony)

SIGTERM

15

zakonczenie

prosba o zakonczenie

Lista wybranych sygnałów POSIX

2

NAZWA

NUMER

DOMYSLNA AKCJA

OPIS/UWAGI

SIGKILL

9

zakonczenie

kończy proces (nie można obsłużyć)

SIGSYS

-

zakonczenie (core dump)

niewłaściwe wywołanie systemowe

SIGSEGV

11

zakonczenie (core dump)

błąd odwołania do pamięci

SIGINT

2

zakonczenie

„Przerwanie” od terminala (zwykle Ctrl+C)

SPRAWDŹ SIĘ!

Do sygnału SIGKILL jest standardowo przypisany numer 9.

TAK

NIE



SPRAWDŹ SIĘ!

Do sygnału SIGKILL jest standardowo przypisany numer 9.



NIE



Komenda kill

Do czego służy komenda kill w systemach operacyjnych?

Komenda kill w systemach operacyjnych Unix/Linux służy do wysłania sygnału do procesu w celu zarządzenia jego działaniem.

Głównym celem komendy kill jest zakończenie procesu lub modyfikacja jego działania poprzez wysłanie określonego sygnału.

Komenda kill



- Wysyłanie sygnałów z poziomu powłoki.
- Możliwość podania wielu PID-ów.
- Lista sygnałów: **kill -l** lub **kill -L**.
- Tłumaczenie **nazwy** sygnału na **numer** **kill -l nazwa**.
- Tłumaczenie **numeru** sygnału na **nazwę** **kill -l numer**.
- Określenie sygnału S: **-S**, **-s**, **S**, **--signal S**.
- Sygnał może być podany w trzech formach np. **9**, **KILL**, **SIGKILL**.
- Domyślnie wysyła **SIGTERM**.

SPRAWDŹ SIĘ!

Domyślnym sygnałem polecenia kill jest SIGTERM.

TAK

NIE



SPRAWDŹ SIĘ!

Domyślnym sygnałem polecenia kill jest SIGTERM.

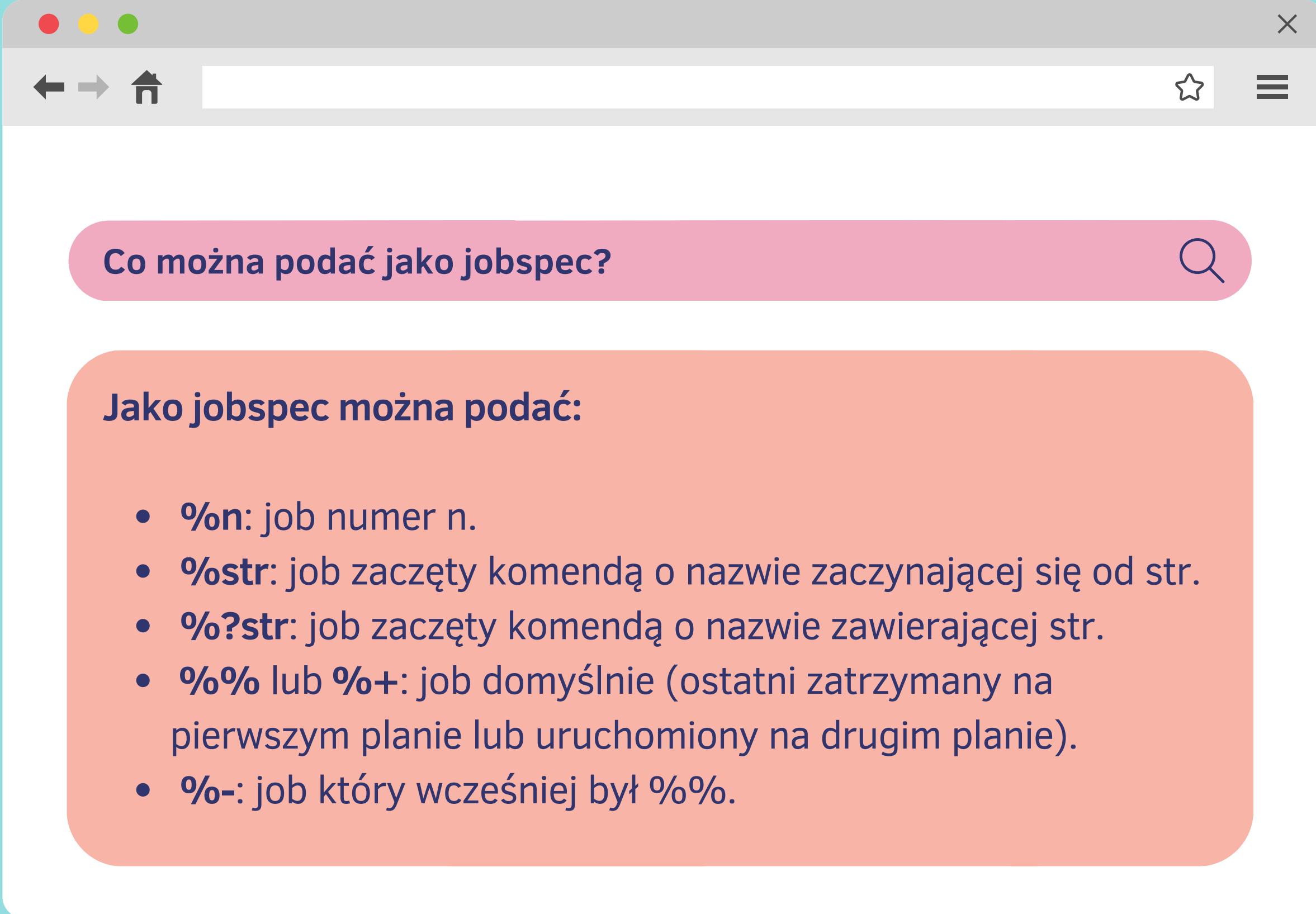


NIE



Job control

- Komendy powłoki (w tym potok) wykonywane jako osobna grupa procesów.
- Jedna grupa jest na pierwszym planie (dostęp do I/O terminala, sygnały **SIGINT**, **SIGQUIT**, **SIGTSTP**).
- Reszta grup na drugim planie (**SIGTTIN**/**SIGTTOU** przy próbie read/write).
- Start procesu w tle (&).
- **fg jobspec** – wznowienie zatrzymanego joba jobspec na pierwszym planie.
- **bg jobspec** – wznowienie zatrzymanego joba jobspec na drugim planie.
- **jobs** – status jobów. Opcja -l oraz ograniczenie do jednego joba poprzez dodanie jobspec.



Co można podać jako jobspec? 🔍

Jako jobspec można podać:

- **%n**: job numer n.
- **%str**: job zaczęty komendą o nazwie zaczynającej się od str.
- **%?str**: job zaczęty komendą o nazwie zawierającej str.
- **%%** lub **%+**: job domyślnie (ostatni zatrzymany na pierwszym planie lub uruchomiony na drugim planie).
- **%-**: job który wcześniej był %%.



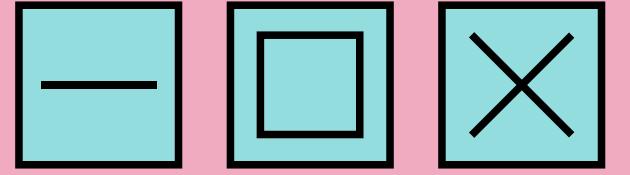
**jobspec musi pasować do jednego joba
lub otrzymamy błąd!**

Komendy



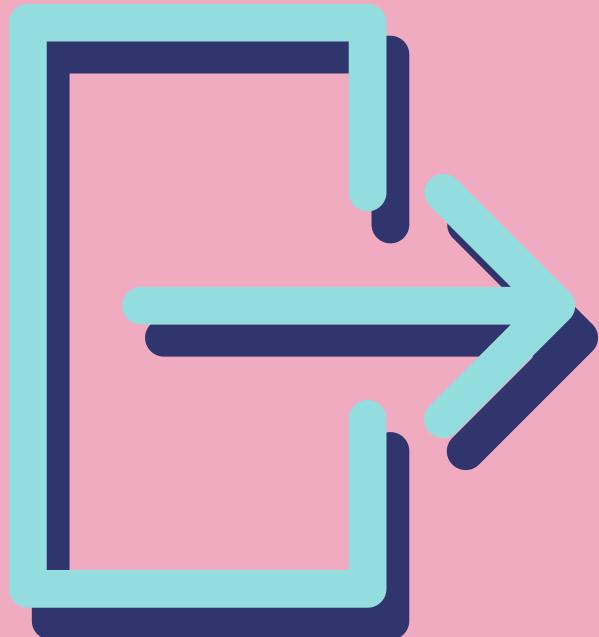
Komenda ps

- Wypisuje **migawkę** (snapshot) pewnego zestawu procesów.
- Przyjmuje różne standardy opcji:
 - 1.Opcje UNIX-owe np. -p.
 - 2.Opcje BSD np. p.
 - 3.Opcje GNU np. --pid.
- Rozbudowana i wymagająca ostrożności.
- Komenda **pstree** wypisuje procesy jako drzewo.



ps – wyjście

Mocno zależny od wielu opcji, w szczególności **opcji formatowania -m**, ale też **-f, -F, -L, u**.



co na wyjściu?

PID, PPID, UID.

TTY – controlling terminal.

TIME – spędzony czas.

STIME – czas rozpoczęcia.

CMD – komenda, która uruchomiła proces.

C – użycie procesora.

STAT – status procesu (waiting, runnable, zombie itd.).

ps – niektóre opcje

-f, -F – „pełny” i „ekstra pełny” format.

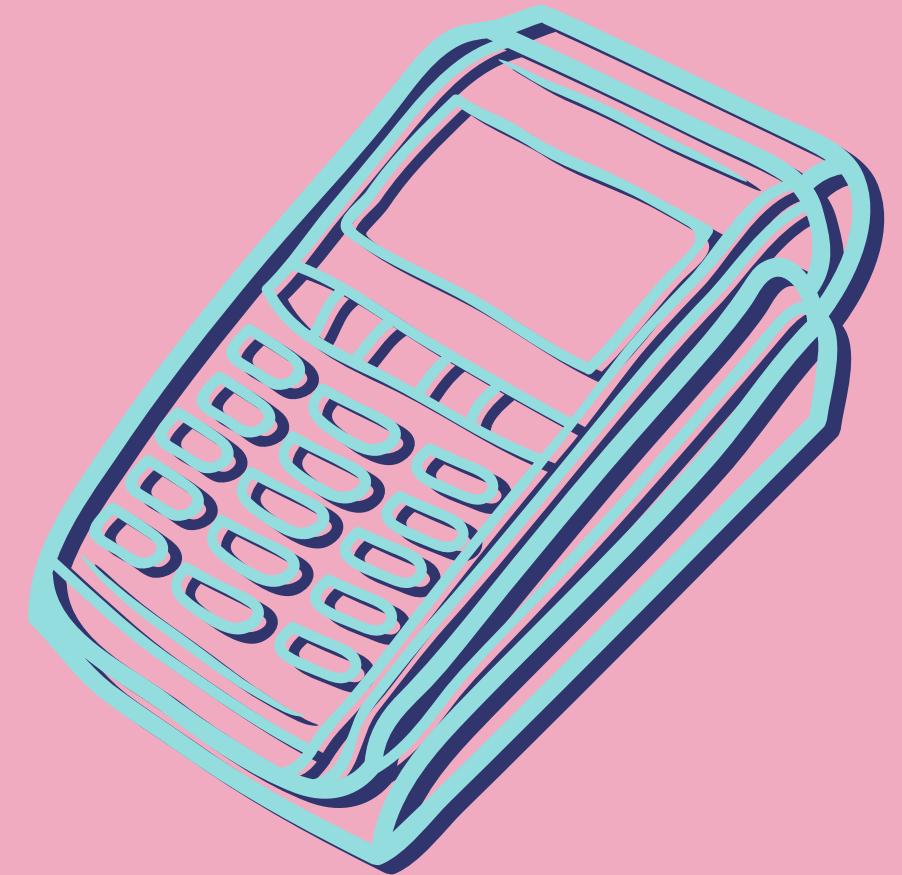
a – bez ograniczenia „tylko własne”.

x – bez ograniczenia „musi być terminal”.

-A, -a – (prawie) wszystkie procesy.

-u, U, --user – procesy podanego użytkownika.

-C – procesy podanej komendy.



Komenda top

Komenda interaktywna (w przeciwieństwie do ps).

Stan procesów na bieżąco (odświeżane snapshoty).

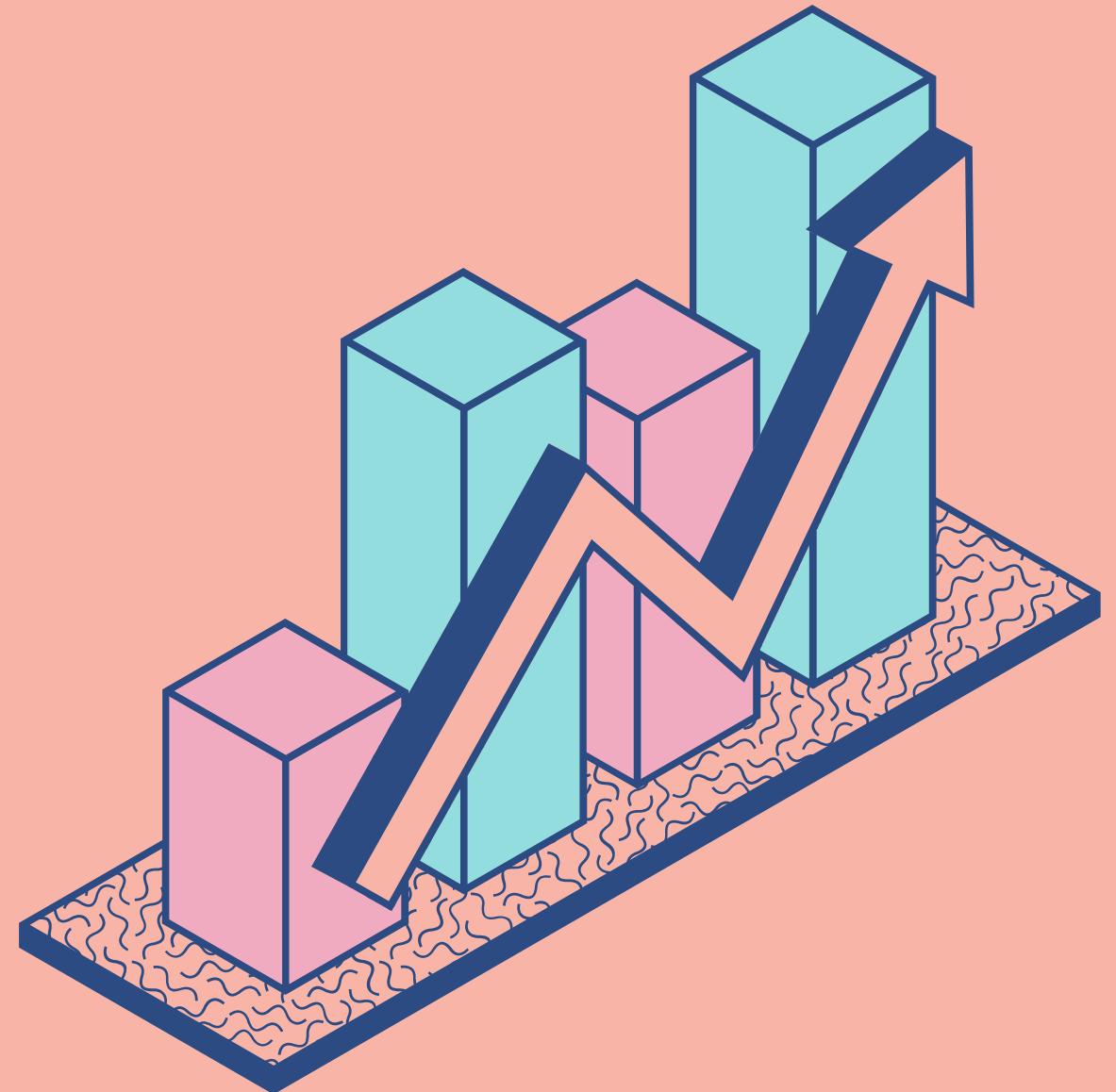
Ogólny stan systemu:

1. Liczba zadań (i ich stan).
2. Liczba użytkowników.
3. Średnie Obciążenie systemu (ostatnie 1, 5, 15 minut).
4. Uptime systemu.
5. Zużycie procesora.
6. Zużycie pamięci.



Podstawowe kolumny:

- **PID, User**
- **PR, NI** – całociowy priorytet (w tym rt) i wartość nice.
- **VIRT, RES, SHR** – zajęta pamięć.
- **S** – stan procesu.
- **%CPU, %MEM** – procentowe użycie procesora i pamięci.
- **TIME+** – łączny czas procesora wykorzystany przez proces.
- **Command.**



SPRAWDŹ SIĘ!

W bashu ampersand (&) służy do przeniesiania działającego procesu z pierwszego planu do drugiego planu (tła).

TAK

NIE



SPRAWDŹ SIĘ!

W bashu ampersand (&) służy do przeniesiania działającego procesu z pierwszego planu do drugiego planu (tła).

TAK



Wykład 6

Zarządzanie pamięcią



Pamięć operacyjna

- Procesy przechowują **kod i dane** w pamięci operacyjnej
- Konieczność ochrony procesu:
 - przed **innymi procesami**
 - przed **samym sobą**
 - jądra przed procesami
- SO ma dostęp do **całej pamięci**



**PLIKI
WYKONYWALNE**

W Linuksie formatem
plików wykonywalnych
jest Executable and
Linkable Format (ELF).



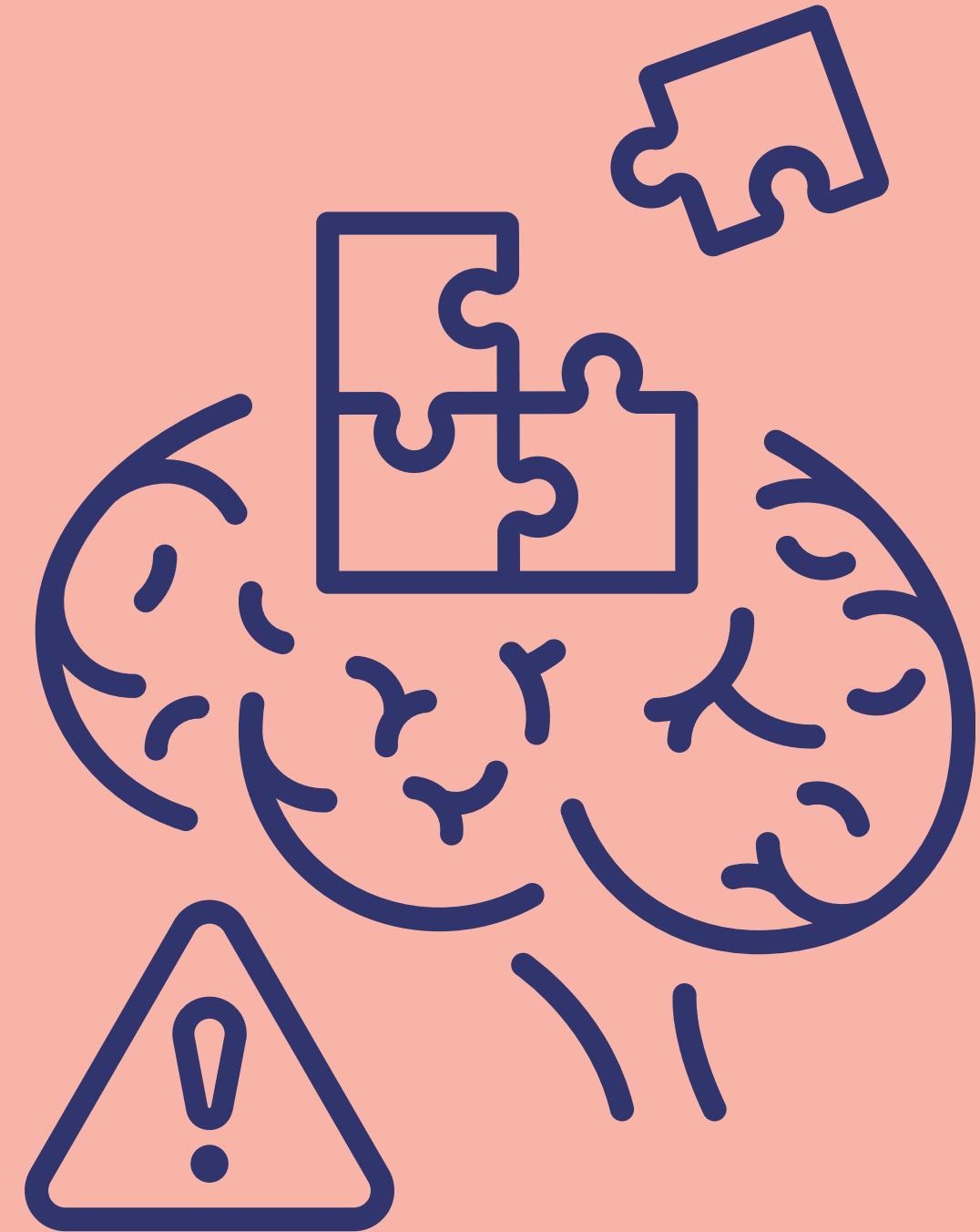
Zawartość pliku ELF:

- Nagłówek ELF.
- Program header table (lista segmentów).
- Segmente/sekcje.
- Section header table (lista sekcji).

Ładowanie pliku wykonywalnego:

- Odbywa się za pomocą wywołania **execve**, które sprawdza plik wykonywalny lub skrypt i analizuje go.
- Lista segmentów wykorzystywana jest przez **loader** do załadowania obrazu.
 - Lista sekcji wykorzystywana jest przez **linker** w procesie **dynamicznego linkowania** (konsolidacja).
 - Do sprawdzania informacji o plikach elf można wykorzystać m.in. **komendy file, readelf oraz size**.
 - Strategia **copy-on-write**.

Pamięć procesu



Text/code (.text) - instrukcje wykonywalne, tylko do odczytu, często współdzielony.

↳ **Sekcje .init oraz .fini.**

Uninitialized data segment (.bss) - globalne/statyczne zmienne z nieustalona (lub zerowa) wartość.

(Initialized) data segment - globalne/statyczne zmienne z ustalona niezerowa wartość.

↳ **Read-only data (.rodata).**

↳ **Read-write data (.data).**



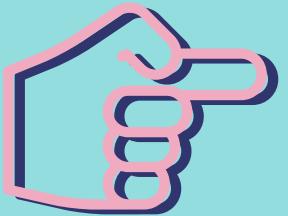
stos (stack)



sterta (heap)



obszary zmapowane za pomocą mmap



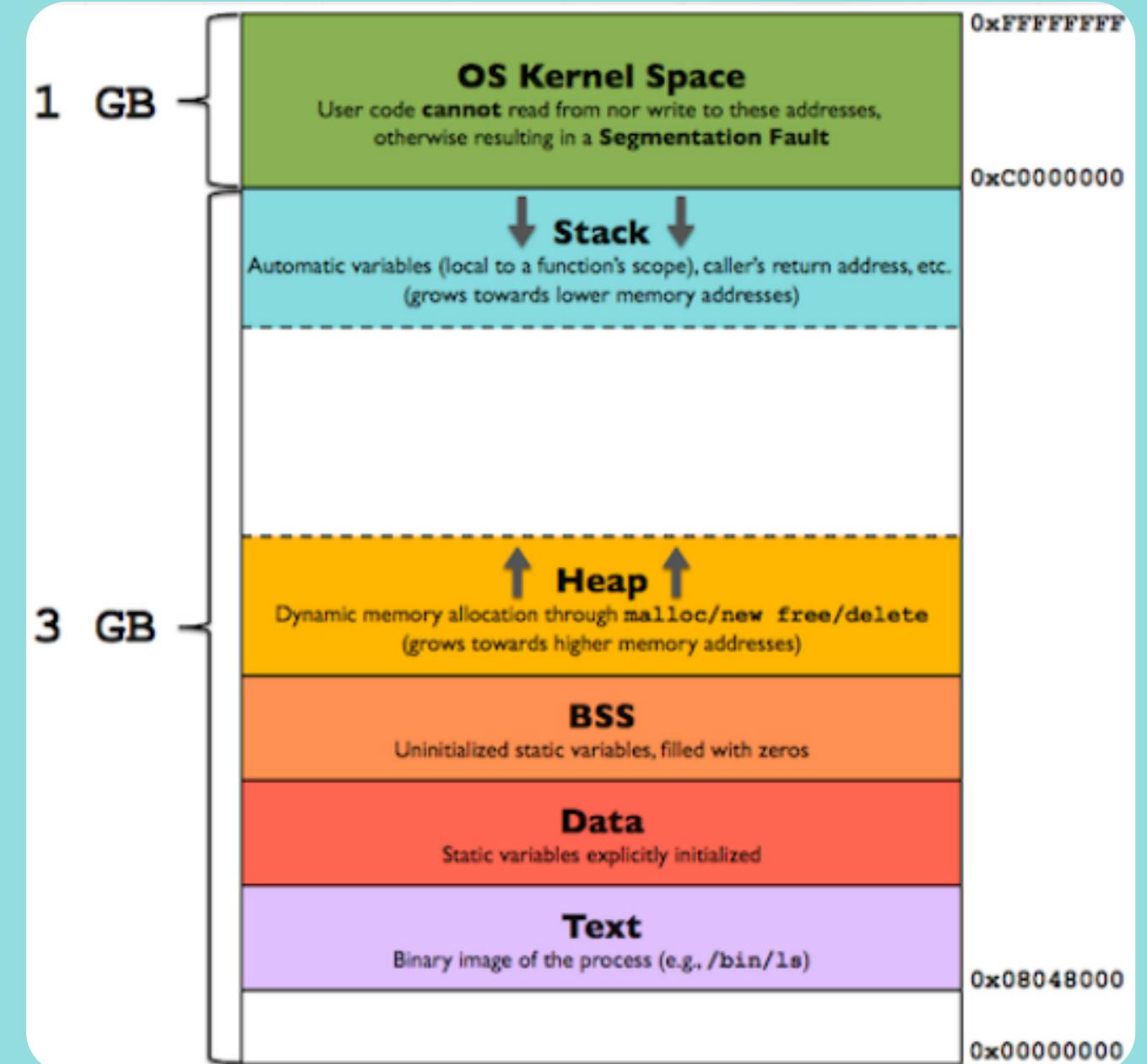
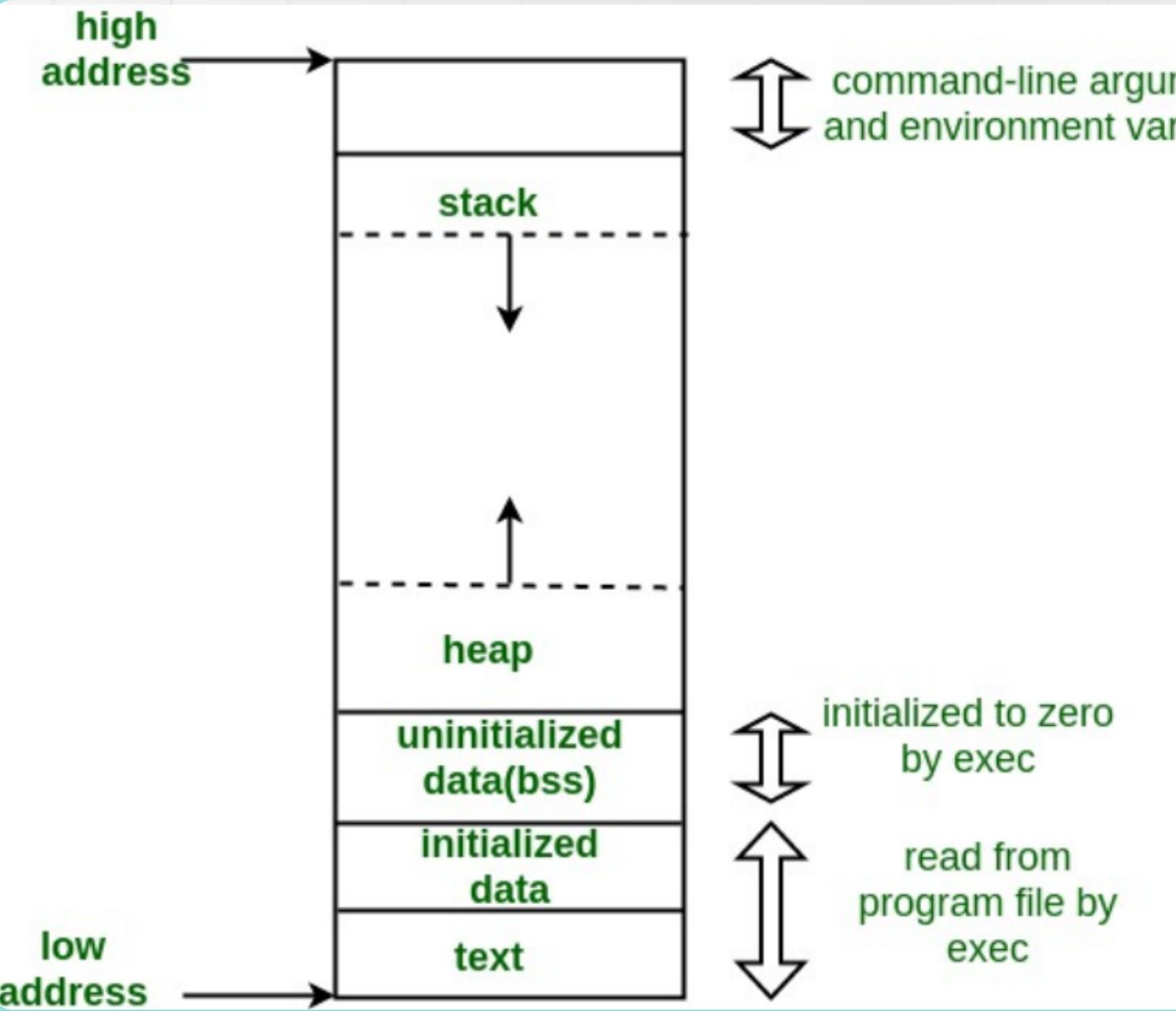
biblioteki współdzielone



argumenty i zmienne środowiskowe



pamięć jądra



STOS

- najczęściej limitowany (**SIGSEGV**) lub rośnie do czasu napotkania innego obszaru (np. sterta)
- przechowuje adresy powrotu funkcji
- przechowuje zmienne lokalne funkcji (w tym argumenty)
- rośnie “w dół”

- osobny na każdy wątek
- zmiana domyślnego limitu stosu:
 - 1) komenda ulimit -s size
 - 2) wywołanie setrlimit
 - 3) plik konfiguracyjny /etc/security/limits.conf
- kernel posiada swój własny mały stos na każdy wątek

STERTA

- pamięć przydzielana **dynamicznie**
- wywoływanie biblioteczne **malloc, calloc, realloc, free**
- początkowy **rozmiar 0 i rośnie w górę**
- przydzielanie pamięci dodatkowej za pomocą **brk/sbrk**
(zwłaszcza przy wyczerpaniu ciągłego obszaru sterty)
 - może rosnąć do czasu wyczerpania wolnej pamięci
 - chunki przekazywane w **dwukierunkowej liście wiązanej**
 - osobne kubełki (bins) dla obszarów o różnej wielkości
 - łączenie przylegających wolnych obszarów
 - **problem fragmentacji zewnętrznej**

SPRAWDŹ SIĘ!

Organizacja sterty typowo bazuje na zasadzie: (A) drzew czerwono-czarnych (B) list wiązanych.

A

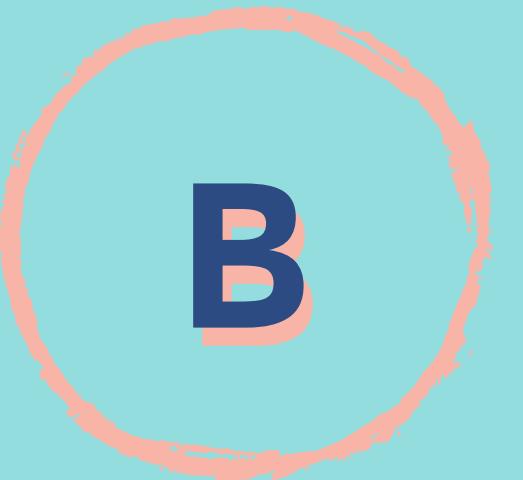
B



SPRAWDŹ SIĘ!

Organizacja sterty typowo bazuje na zasadzie: (A) drzew czerwono-czarnych (B) list wiązanych.

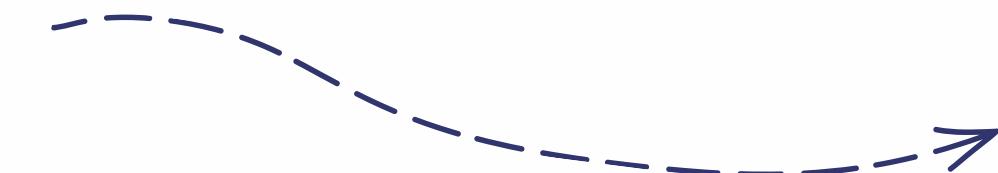
A



1

PAMIĘĆ WIRTUALNA

- Każdy proces widzi swoją przestrzeń tak samo (kod, dane, stos i sterta w tym samym miejscu).
- Każdy proces widzi pamięć jako ciągły obszar.
- Każdy proces w systemie 32-bitowym widzi 3–4 GiB pamięci.



Nawet na maszynie z zainstalowanymi 2 GiB RAM?

Tak

Dlaczego to działa?

Pamięć wirtualna

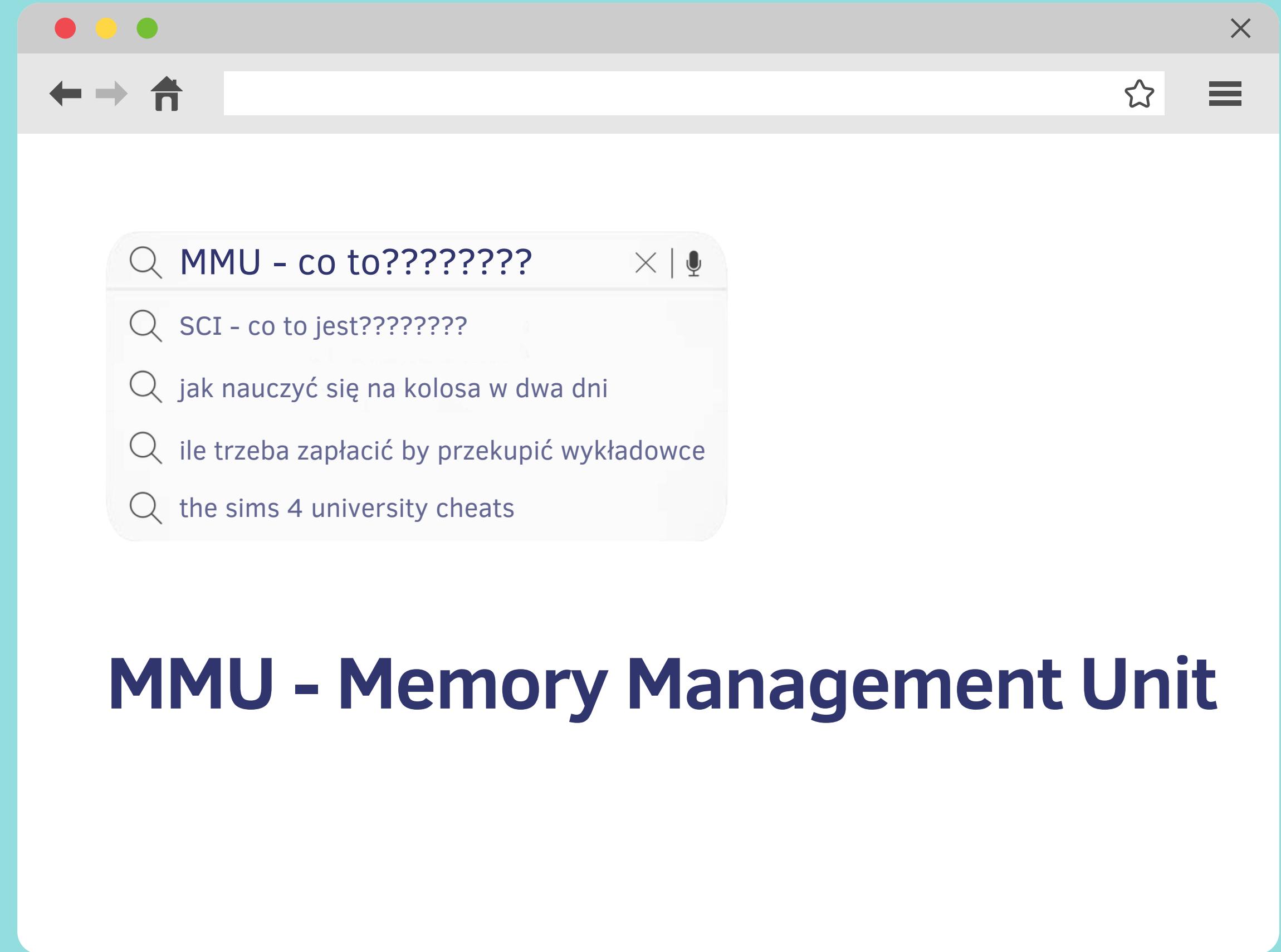


2

PAMIĘĆ WIRTUALNA

Powyższe generuje szereg paradoksów:

- Proces widzi adresy logiczne – w teorii operuje na całej dostępnej przestrzeni adresowej procesora.
- System widzi adresy fizyczne – ograniczone przez zainstalowaną pamięć RAM (oraz pamięć swap).
- Faktyczny rozkład pamięci procesu jest inny niż widzi to sam proces.
- Konieczność translacji adresów logicznych na adresy fizyczne przy wsparciu systemu operacyjnego i sprzętu.

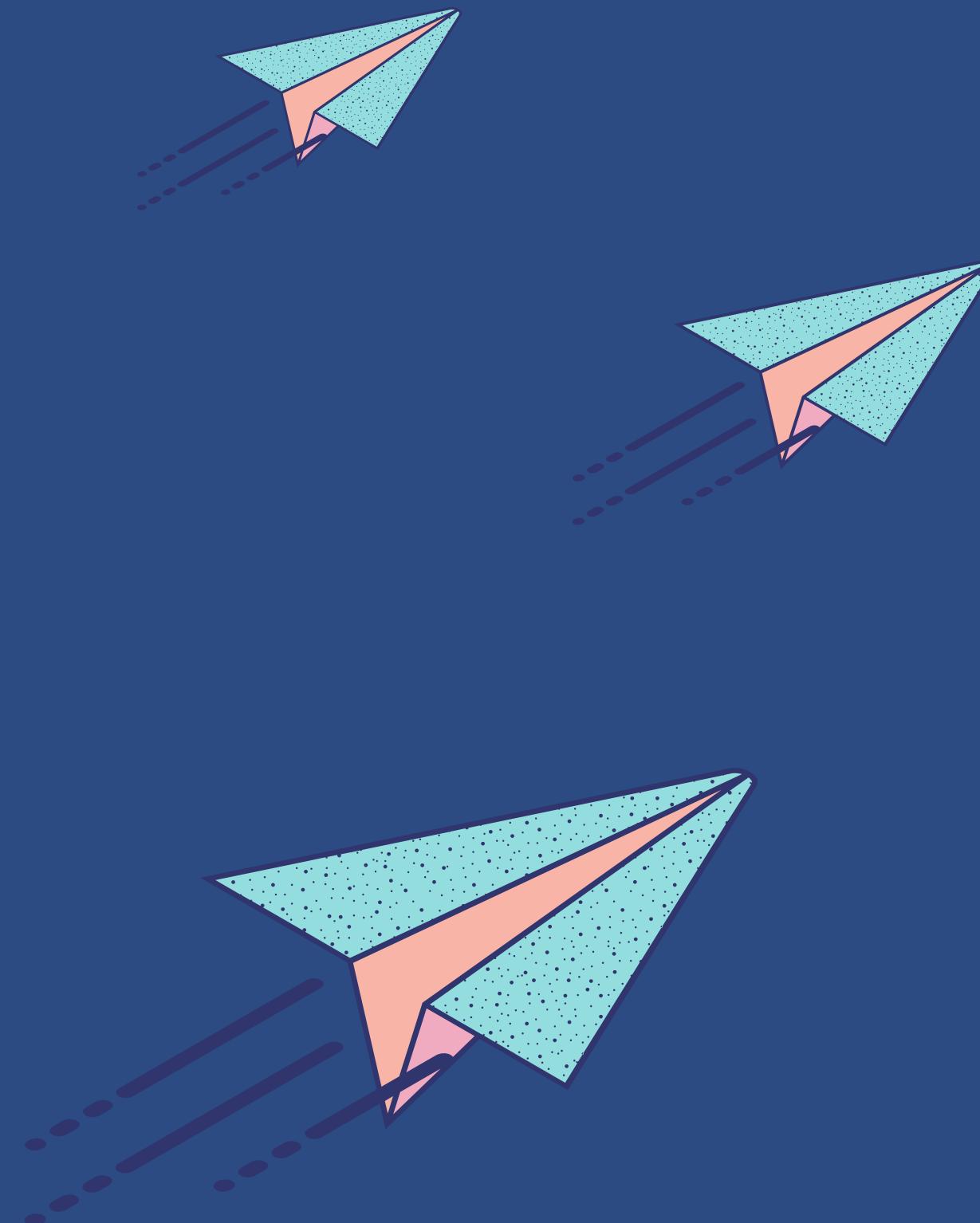


MMU - Memory Management Unit

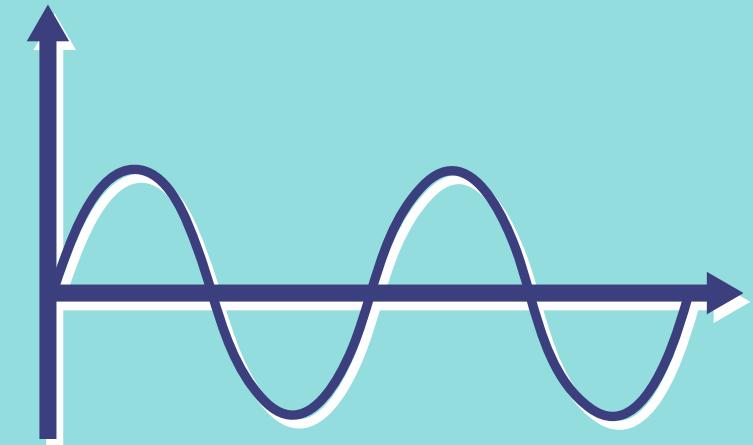
MMU to układ sprzętowy do zarządzania pamięcią.

Zajmuje się:

- 1 Translacjią adresów logicznych na fizyczne.
- 2 Ochroną pamięci.
- 3 Obsługą pamięci podręcznej.
- 4 Obsługą szyny pamięci.
- 5 Przełączaniem banków pamięci.



PRZYDZIAŁ CIĄGŁY



- Każdy proces otrzymuje jeden spójny obszar pamięci adresy logiczne od 0 do końca przydzielonego obszaru pamięci MMU sprawdza czy nie przekroczono przydzielonego obszaru i oblicza adres fizyczny na podstawie początku przydziału i przesunięcia.
- Fragmentacja zewnętrzna:
 1. Można zmieścić jedynie proces nie większy niż największy wolny obszar - łączna dostępna pamięć może być dużo większa
 2. Kompaktyfikacja - kosztowne scalanie wolnych obszarów
 3. Zwykle jest wiele wolnych miejsc do przydziału
- Różne strategie przydziału
 1. First-fit
 2. Best-fit
 3. Worst-fit
 4. Algorytm optymalny musiałby być jasnowidzący (dostępny tylko offline)
- Podobne strategie stosuje się w przypadku zarządzania stertą.

SEGMENTACJA



- Każdy proces otrzymuje kilka spójnych obszarów pamięci - segmentów
- Segmente na kod, dane, stos, stertę itd.
- Możliwa alokacja dodatkowych segmentów
- Te same strategie przydziału, problem fragmentacji zewnętrznej i możliwość kompatyfikacji, jak dla przydziału ciągłego
- Inna budowa i ograniczenia MMU

➤ BUDOWA MMU:

- Tablica rozmiarów oraz tablica przemieszczeń (adresy bazowe) w RAM
- Dwuetapowy dostęp do pamięci: tablica => obliczony adres fizyczny

➤ OPTYMALIZACJE:

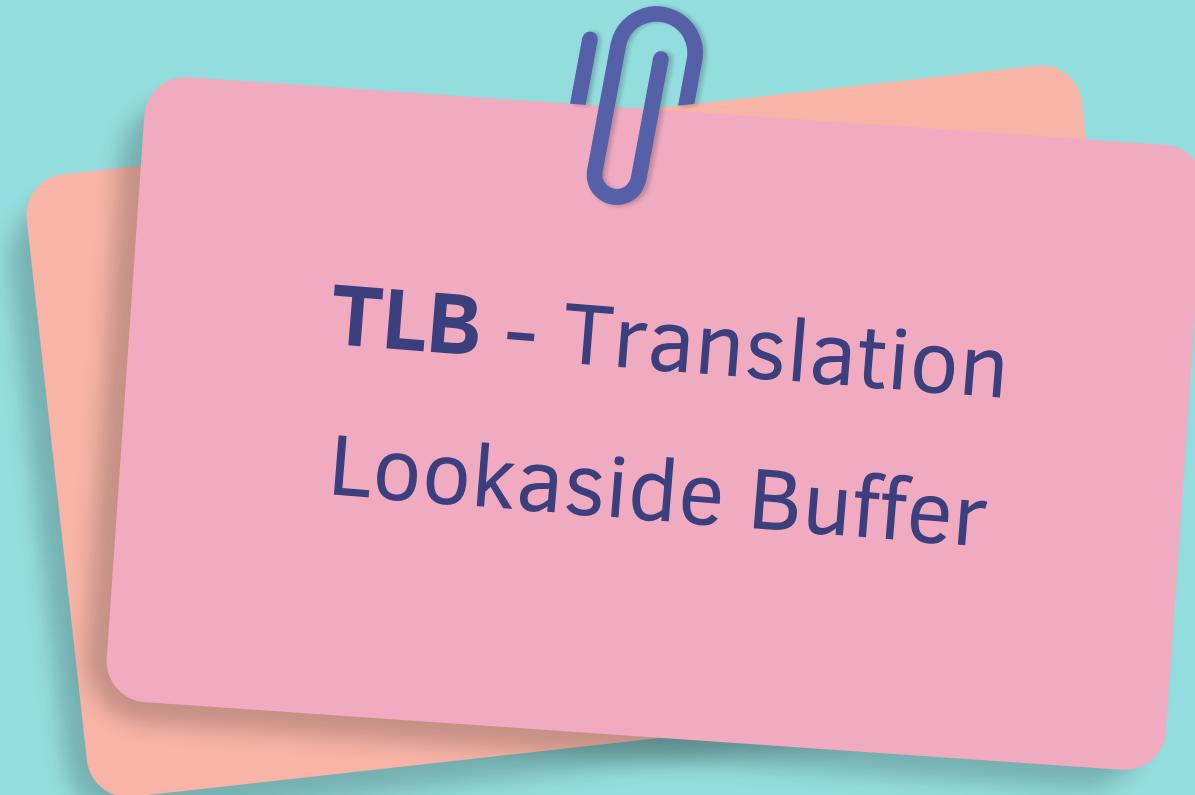
- Mała liczba segmentów, przechowanie ich w rejestrach procesora
- Mała pamięć podręczna MMU przechowuje część tablicy segmentów
- Przełączanie kontekstu - zamiana wartości tablic lub wskaźników procesora
- Możliwość współdzielenia segmentów

STRONICOWANIE

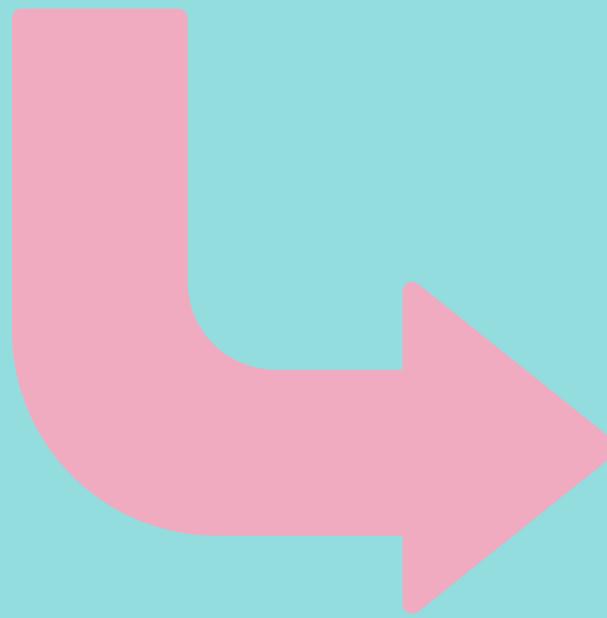
- Ciągły obszar pamięci **wirtualnej**, ale nieciągły w pamięci **fizycznej**
- Pamięć wirtualna podzielona na strony, a pamięć fizyczna na ramki
- Strony i ramki są równej wielkości, najczęściej **4kiB**
- Przydzielony obszar jest **zawsze** wielokrotnością rozmiaru strony (ramki)
- System może umieścić dowolną stronę w dowolnej ramce
- Translacja numerów stron na numery ramek jest przechowywana w tablicy stron procesu

MMU dla stronicowania:

- Adres dzielony jest na co najmniej dwie części, **wyższe bity** (np. 20) opisują **numer strony**, a **niższe** (np. 12) opisują **przesunięcie w ramach strony**
- Tablica stron znajduje się w pamięci, więc stosuje się podręczną tablicę stron: TLB
- Zasada lokalności zapewnia, że TLB wystarcza w **większości** wypadków
- TLB **wymaga czyszczenia** przy zmianie kontekstu (przełączanie procesów)
- Przetłumaczone strony umieszczane są w TLB
- Przy przepełnieniu należy **usunąć** jakąś stronę z TLB



- Stronicowanie unika fragmentacji zewnętrznej
- Część (średnio połowa) ostatniej strony jest niewykorzystana - zachodzi zjawisko fragmentacji wewnętrznej
- Duży rozmiar strony przy wielu procesach powoduje znaczną fragmentację
- Mały rozmiar strony zwiększa liczbę stron, a tym samym zwiększa rozmiar tablicy stron



kompromis

- Podział strony na co najmniej dwie części: **katalog stron i tablica stron**, najczęściej **po 10 bitów** (dla 32-bitowego systemu)
- **Physical Address Extension** - trzypoziomowa translacja, adresy fizyczne do 36 bitów, adres logiczny pozostaje bez zmian
- W systemach **64-bitowych** stronicowanie jest zwykle na **4 poziomach**, pozwalając zaadresować **256 TiB** pamięci z wykorzystaniem **48 bitów adresu**

SPRAWDŹ SIĘ!

Większy rozmiar strony przy stronicowaniu zwiększa: (A) fragmentację (B) rozmiar tablicy stron.

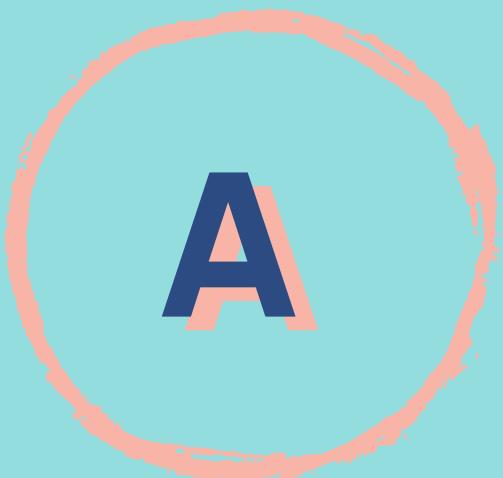
A

B



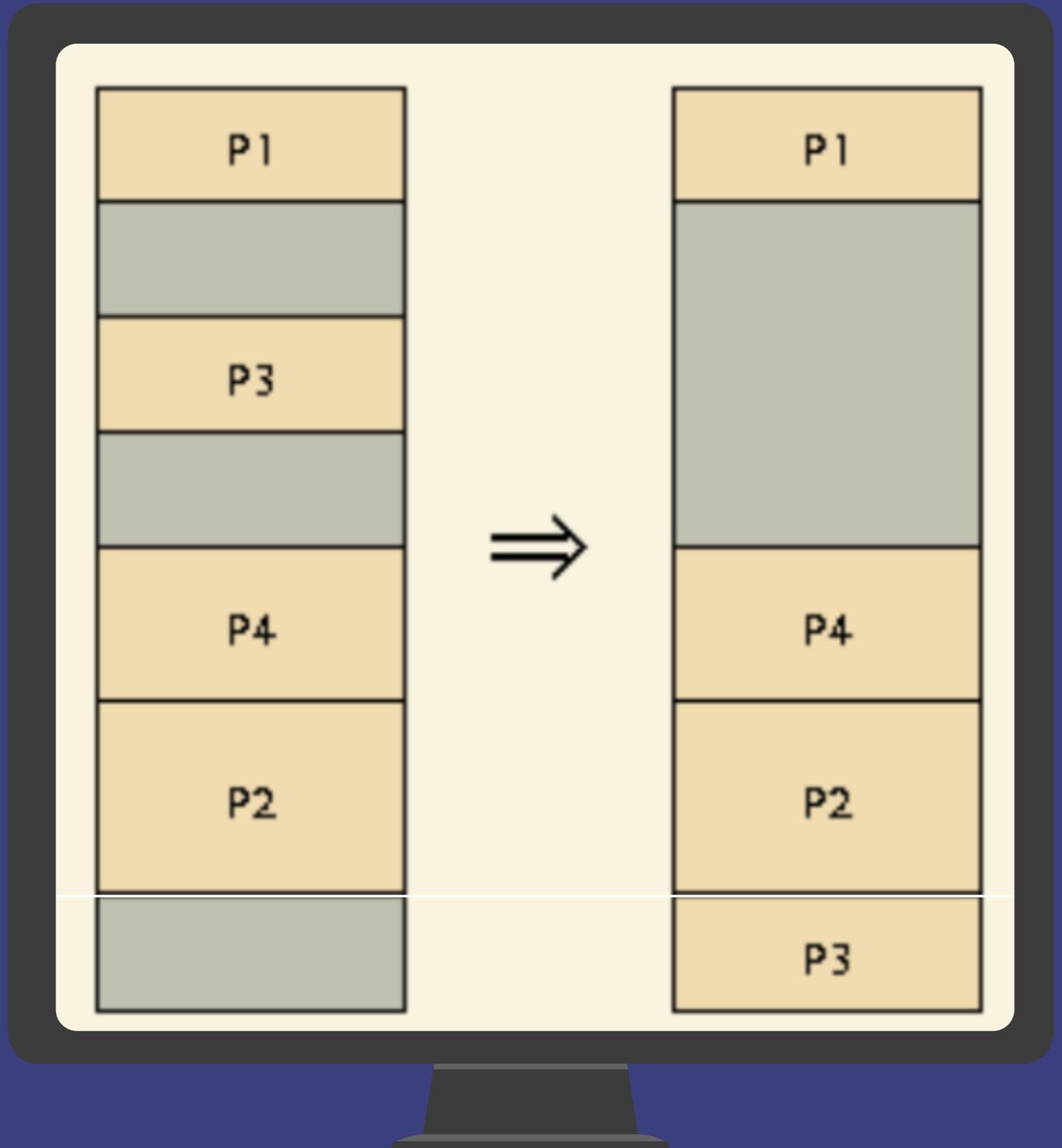
SPRAWDŹ SIĘ!

Większy rozmiar strony przy stronicowaniu zwiększa: (A) fragmentację (B) rozmiar tablicy stron.

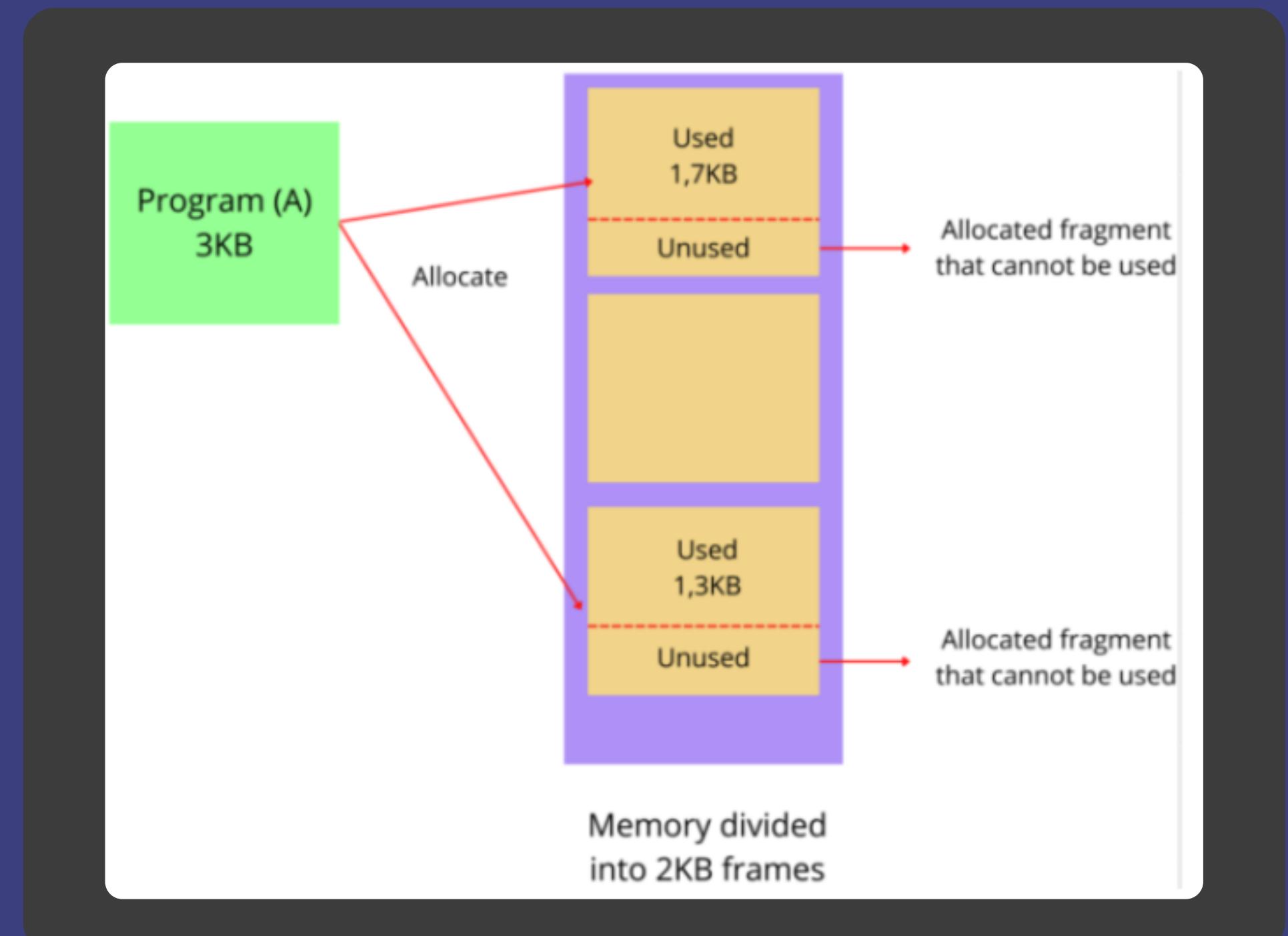


Fragmentacja Zewnętrzna

- Można zmieścić jedynie proces **niewiększy niż największy wolny obszar.**
- Łączna dostępna pamięć może być **dużo większa.**
- **Kompaktyfikacja** – kosztowne scalanie wolnych obszarów.



Fragmentacja Wewnętrzna



Różnice między fragmentacją zewnętrzną a wewnętrzną:

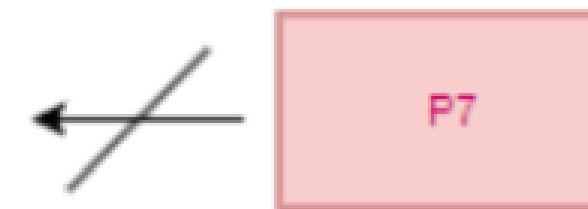
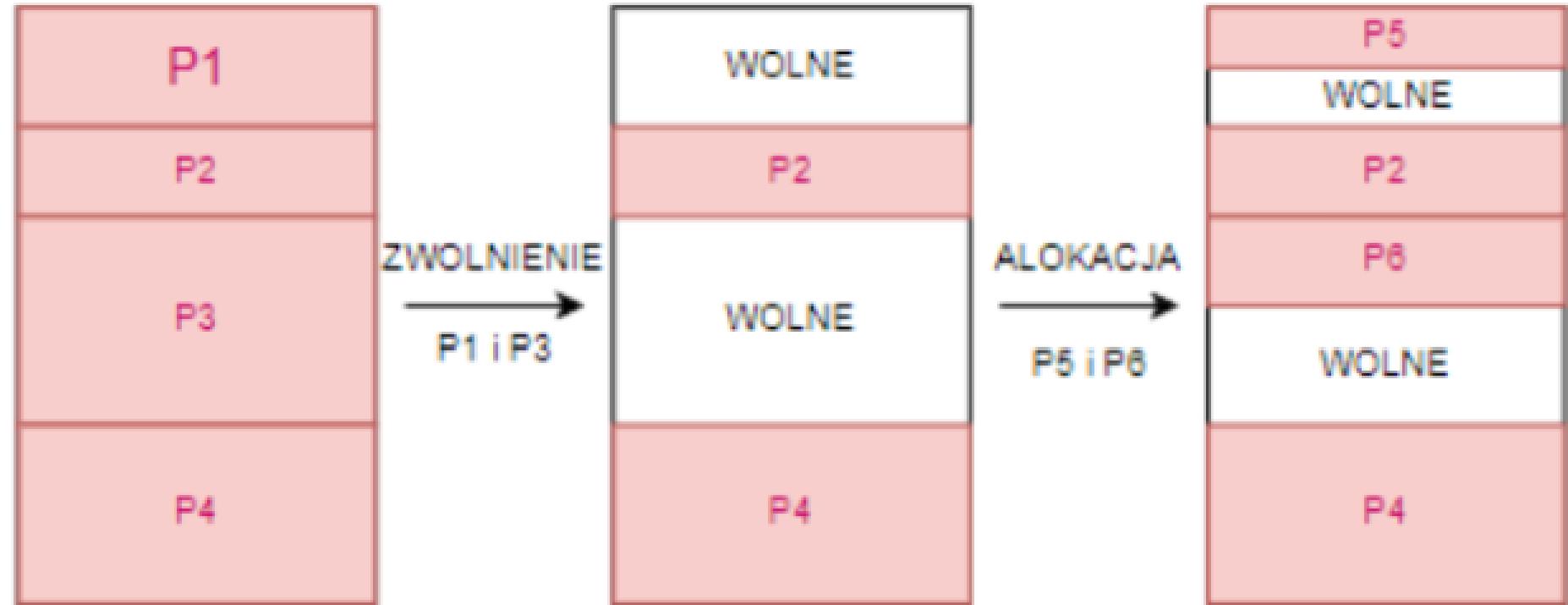


Fragmentacja zewnętrzna występuje gdy:

- pamięć jest alokowana dynamicznie, gdzie po załadowaniu i zwolnieniu kilku gniazd tu i tam, wolna przestrzeń jest raczej dystrybuowana, niż ciągła.
- w trakcie działania aplikacji różne obszary pamięci są zwalniane i alokowane; powoduje to powstawanie małych, nieciągłych obszarów, niespełniających wymaganej wielkości dla danego procesu.

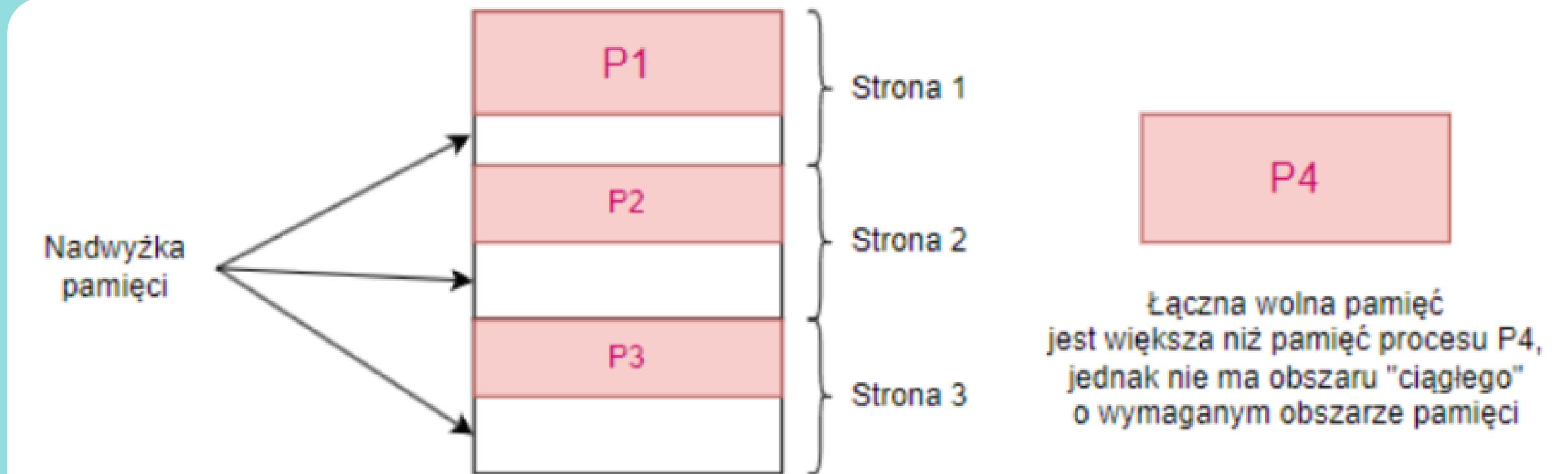
Fragmentacja wewnętrzna występuje gdy:

- alokacja pamięci jest oparta na partycjach o stałym rozmiarze, gdzie po przypisaniu aplikacji o małym rozmiarze do gniazda marnowane jest pozostałe wolne miejsce w tym gnieździe.
- zarządzana pamięć jest podzielona na pewne statyczne obszary; kiedy przydzielona jest pamięć, "nadwyżka" w bloku nie jest używana przez aplikację, ale też nie może zostać przydzielona innej aplikacji.



Łączna wolna pamięć
jest większa niż pamięć procesu P7,
jednak nie ma obszaru "ciągłego"
o wymaganym obszarze pamięci

Problem fragmentacji zewnętrznej



Problem fragmentacji wewnętrznej

SPRAWDŹ SIĘ!

W przypadku stronicowania mamy do czynienia z problemem fragmentacji: (A) wewnętrznej, (B) zewnętrznej.

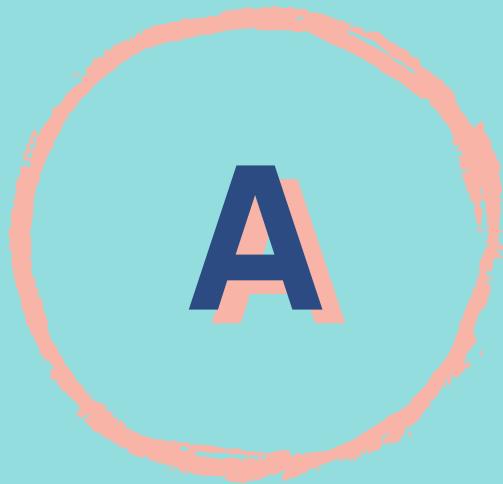
A

B



SPRAWDŹ SIĘ!

W przypadku stronicowania mamy do czynienia z problemem fragmentacji: (A) wewnętrznej, (B) zewnętrznej.



mmap

Służy do mapowania pliku do przestrzeni adresowej procesu.
To oznacza, że umożliwia dostęp do danych z pliku poprzez
operacje na pamięci.

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);

addr -> preferowany adres początkowy mapowania (NULL = system sam wybierze)

length -> rozmiar obszaru do zmapowania w bajtach

prot -> uprawnienia dla obszaru `PROT_READ/WRITE/EXEC`

flags -> flagi np. MAP_SHARED (mapowanie udostępnione), MAP_PRIVATE (mapowanie prywatne)

fd -> deskryptor pliku, z którego mapujemy dane (jeśli mapujemy plik)

offset -> przesunięcie od początku pliku, od którego rozpoczynamy mapowanie

Wykład 7

Zarządzanie pamięcią - cd.



Segmentacja ze stronicowaniem

- Pamięć logiczna składa się z segmentów, widocznych dla procesów jako **spójny obszar**
- Możliwość **wielu** segmentów i ich **współdzielenia**
- Segmente są stronicowane, **brak** fragmentacji zewnętrznej
- Segment i przesunięcie tłumaczone na adres liniowy
- Adres liniowy tłumaczony na adres fizyczny (ramkę)
- Możliwość **wielu** poziomów i pamięci podręcznej
- Linux korzysta z jednego segmentu o **adresie 0x0** i **rozmiarze 4GiB** (32 bitowa wersja)

Atrybuty stron

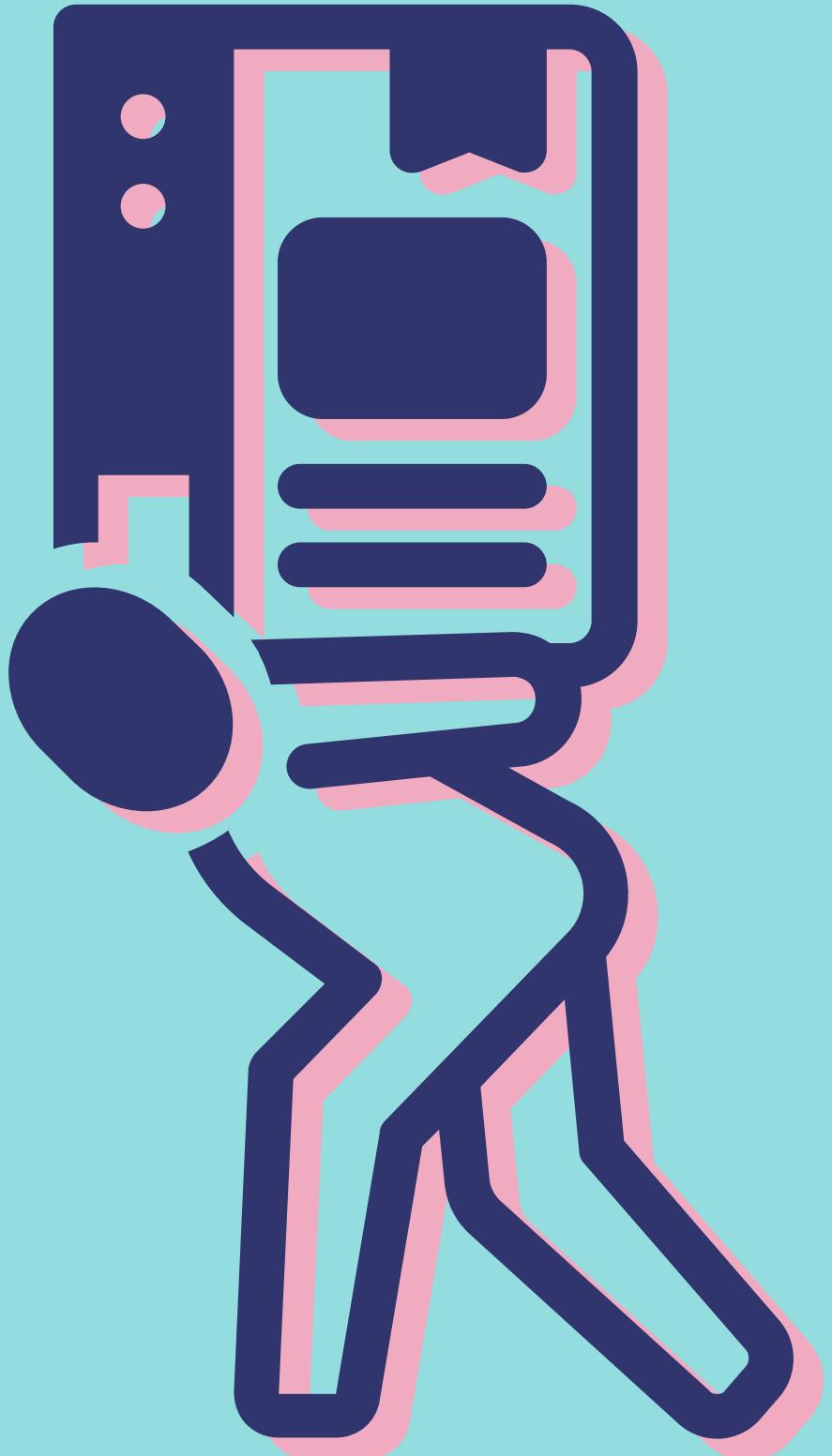
Wpis w tablicy stron może zawierać:

- ▶ Numer ramki.
- ▶ Presence bit – 0 powoduje błąd strony.
- ▶ Read/write bit, Execute bit.
- ▶ User/Supervisor bit.
- ▶ Access/reference bit.
- ▶ Cache enabled/disabled bit.
- ▶ Dirty/modified bit.
- ▶ Swap information



HUGE PAGES

- ▶ Strony rozmiaru 4 kiB ograniczają fragmentację, ale również wydajność.
- ▶ Huge pages – dodatkowe większe rozmiary stron.
- ▶ Wsparcie sprzętowe (dla x86 64 są to 2 MiB oraz 1 GiB).
- ▶ Aplikacje muszą mieć ich świadomość.
- ▶ Transparent Huge Pages.



BŁĄD STRONY

- Zapis do strony z bitem NX (no execute) - wysłanie SIGSEGV
- Strona w pamięci, ale nie w tablicy stron (minor) - uzupełnienie
- Strony nie ma w pamięci (major) - próba uzupełnienia:
 1. Pierwsze odwołanie
 2. Strona na dysku



Odwołanie do strony spoza pamięci procesu:

- błędny adres (**invalid**) - wysłanie **SIGSEGV** do procesu
- adres blisko końca stosu - przydzielenie pamięci
- dla stosu głównego **mmap(MAP GROWSDOWN)**



SPRAWDŹ SIĘ!

(na logikę)

W przypadku stronicowania
wystąpienie błędu strony nigdy
nie powoduje zakończenia
procesu

TAK

NIE



SPRAWDŹ SIĘ!

W przypadku stronicowania
wystąpienie błędu strony nigdy
nie powoduje zakończeniu
procesu

TAK



Procedura uzupełnienia strony:

- 1** Zlokalizowanie strony na dysku (jeśli jest w swapie)
- |
- 2** Zlokalizowanie wolnej ramki w pamięci fizycznej
- |
- 3** Wczytanie strony do ramki
- |
- 4** Uzupełnianie tablic strony/TLB
- |
- 5** Wznowienie programu (na tej samej instrukcji)

Thrashing



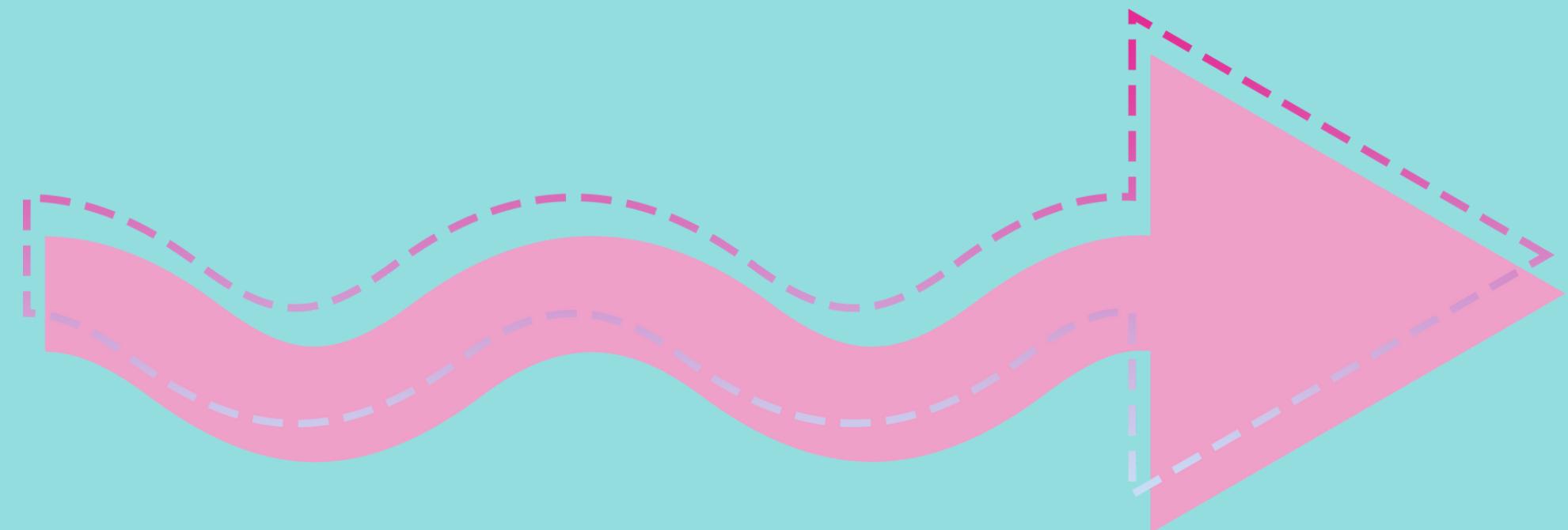
Pamięć potrzebna teraz i w najbliższej przyszłości określana jest mianem working set.

- Jeśli working set jest mały względem całej pamięci, to narzut obsługi błędów stron jest **niski** lub **pomijalny**.
- Jeśli working set jest duży względem rozmiaru pamięci, to strony zrzucone na dysk, szybko znów będą potrzebne.
- Istnieje **punkt krytyczny**, powyżej którego system traci większość czasu na obsługę błędów stron (komunikacja z dyskiem)



Zjawisko takie nazywamy thrashingiem (szarpaniem/szamotaniem)

Wybrane techniki ładowania i usuwania stron



Ładowanie wszystkich stron od razu

- Mniej błędów stron.
- Dłuższe ładowanie programów przy uruchomieniu.
- Mniej pamięci dostępnej dla innych procesów.
- Niemożliwe w 100% – obecność sterty i brk/srk.

Demand paging

- ładowanie stron kiedy są potrzebne (na żądanie)

- Krótsze ładowanie programów.
- Więcej pamięci dostępnej do innych programów.
- Część stron może nie być w ogóle potrzebna.

Wady Demand paging

- Większy koszt pierwszego odwołania do strony.
- Może być niedostępne dla niektórych systemów (urządzenia wbudowane).
- Więcej błędów stron, większe zagrożenie szamotaniem.
- Niemożliwe w 100% – obecność sterty i brk/srk.

SPRAWDŹ SIĘ!

W porównaniu do ładowania wszystkich stron od razu, ładowanie na żądanie typowo powoduje: (A) większą dostępność pamięci dla innych procesów, (B) mniejszą dostępność pamięci dla mniejszych procesów.

A

B



SPRAWDŹ SIĘ!

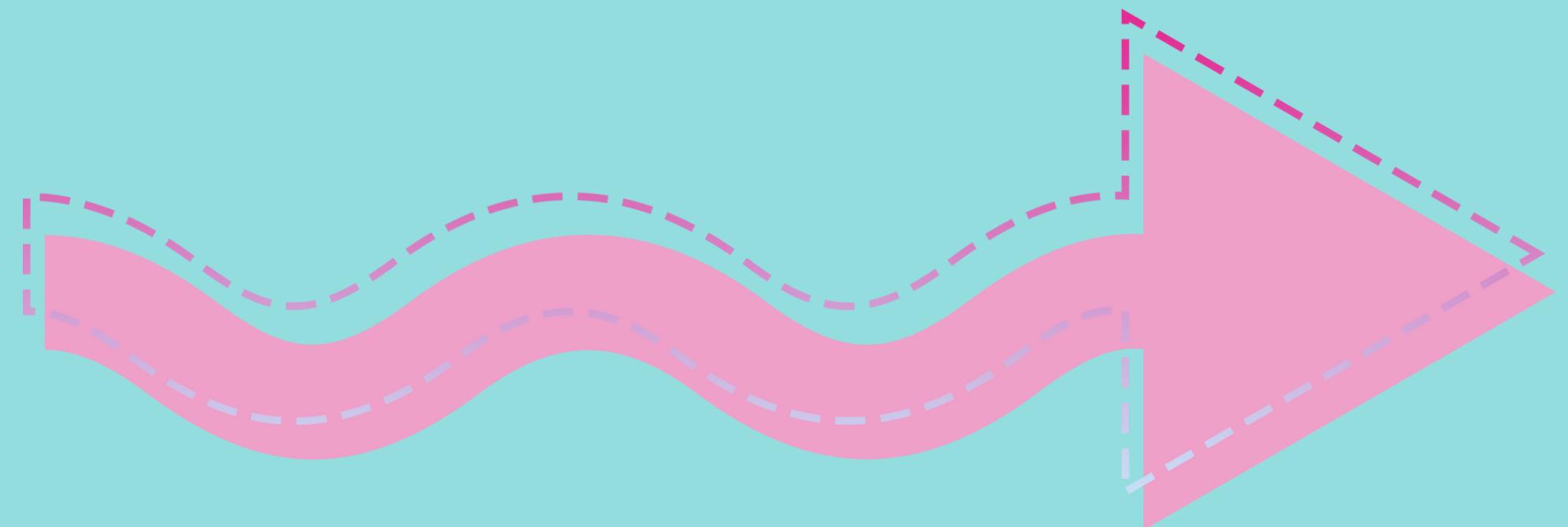
W porównaniu do ładowania wszystkich stron od razu, ładowanie na żądanie typowo powoduje: (A) większą dostępność pamięci dla innych procesów, (B) mniejszą dostępność pamięci dla mniejszych procesów.

A

B



Wybrane Algorytmy Zastępowania Stron



Algorytm OPT

- Algorytm **optymalny**
- Usuwana strona, która będzie potrzebna najpóźniej
- Wymaga **przewidywania przyszłości**.
- Teoretycznie możliwy w niektórych systemach
- Możliwe bardzo dobre algorytmy przybliżone



Algorytm NRU

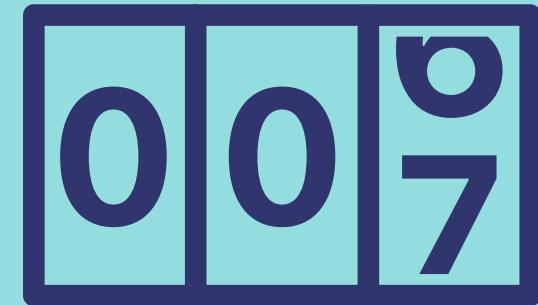
- Not Recently Used (and/or Modified)
- Bit odwołania czyszczony okresowo
- Cztery kategorie stron:
 1. Bez odwołania i bez modyfikacji.
 2. Modyfikacja bez odwołania.
 3. Odwołanie bez modyfikacji.
 4. Odwołanie i modyfikacja.
- Usuwana losowa strona z najniższej niepustej kategorii.

Algorytm LRU

- **Least Recently Used**
- Koncepcja podobna do NRU, ale **stopień użycia opisywany dokładniej**
- Wysoka średnia skuteczność
- Skomplikowana implementacja:
 - Lista wiązana – wymaga częstej reorganizacji.
 - Przy dostępie do strony zapisywana wartość licznika – wymaga wsparcia sprzętowego.
 - Niska skuteczność w niektórych typowych przypadkach np. zapętlony dostęp do tablicy.

Algorytm NFU

- Not Frequently Used
- Licznik dla każdej strony
- Licznik zwiększany o 1 jeśli do strony było odwołanie w danym okresie
- Usuwana strona o najmniejszym liczniku
- Ignoruje okres czasu
- Średnio słabszy od LRU, ale generuje mniej błędów stron w niektórych przypadkach



Algorytm Aging



- Rozbudowa NFU poprzez uwzględnienie okresu czasu
- Stara wartość jest dzielona przez 2 przed dodaniem nowej wartości
- Horyzont ograniczony do 16 lub 32 okresów
- Efektywność bliska optimum
- Umiarkowana złożoność.

Algorytm Aging



- Przy założeniu 6 okresów, licznik Aging wynosi:
1. dla ciągu 0, 0, 0, 1, 1, 1 – 111000 (56)
2. dla ciągu 1, 1, 1, 0, 0, 0 – 000111 (7)
3. dla ciągu 1, 0, 0, 1, 1, 0 – 011001 (25)
- We wszystkich powyższych przypadkach licznik NFU wyniesie 3

SPRAWDŹ SIĘ!

Optymalny algorytm
zastępowania stron jest możliwy
w niektórych szczególnych
przypadkach.

TAK

NIE



SPRAWDŹ SIĘ!

Optymalny algorytm
zastępowania stron jest możliwy
w niektórych szczególnych
przypadkach.



NIE

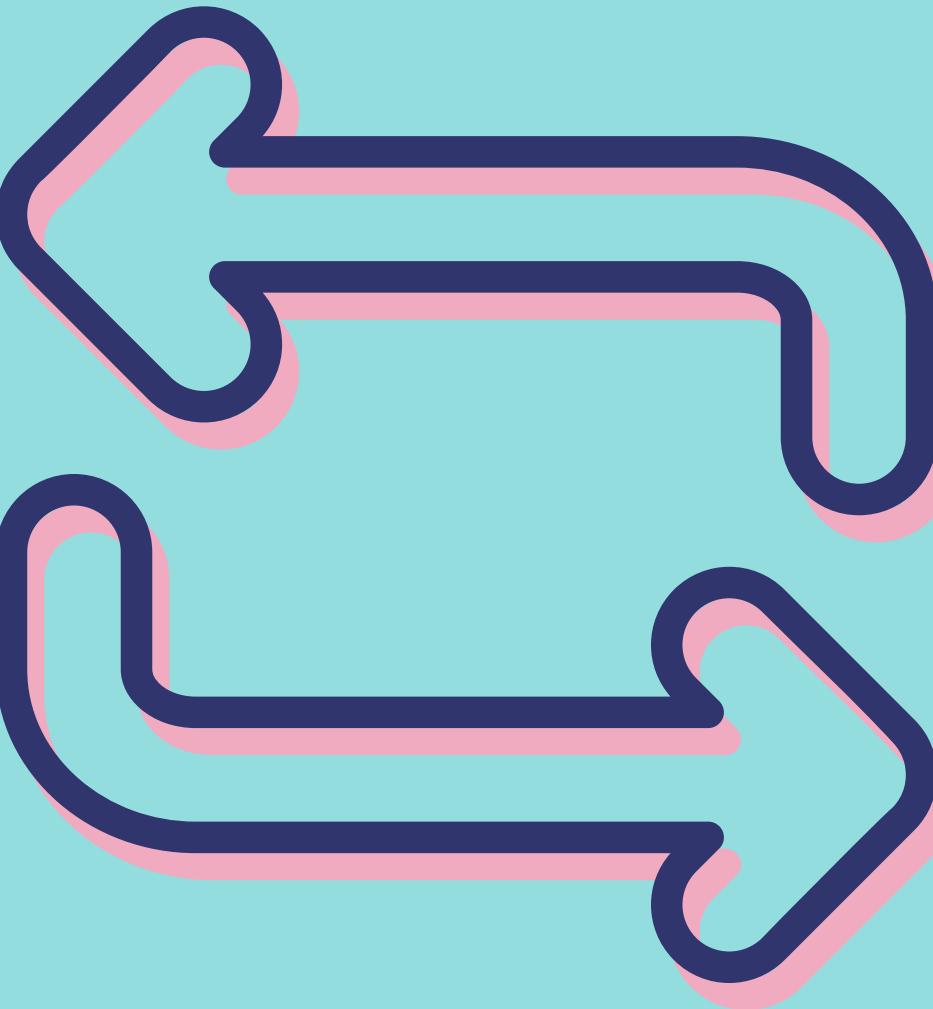


Wykład 8

Zarządzanie pamięcią - cd.

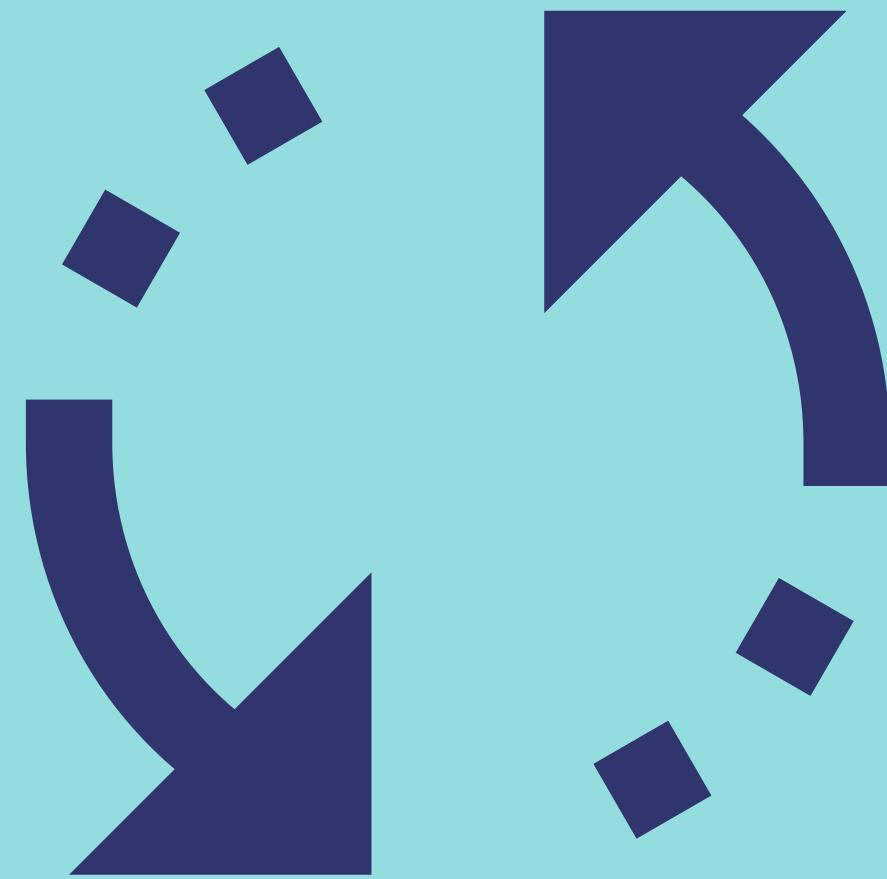


Pamięć Swap



Pamięć swap

- Wykorzystanie pamięci masowej do przechowywania stron z pamięci operacyjnej.
- Może być traktowane jako swoiste **rozszerzenie RAM**.
- Kłopoty w przypadku starszych urządzeń SSD.



Swap w Linuksie:

- **Swap partitions** – szybsze (zwłaszcza przed wersją jądra 2.6), ale nie mogą łatwo zmienić rozmiaru.
- **Swap files** – elastyczne, ale mogą być wolniejsze.
- Możliwość zdefiniowania **wielu swap partitions/files** wraz z priorytetem.
- Parametr **swappiness** (0 do 100, domyślnie 60) – pozwala sterować proporcją pomiędzy usuaniem stron anonymous i file-backed.

SPRAWDŹ SIĘ!

Pamięć swap w Linuksie przyjmuje postać pliku lub całej partycji. Rozmiar obu nie może być łatwo zmieniony.

TAK

NIE



SPRAWDŹ SIĘ!

Pamięć swap w Linuksie przyjmuje postać pliku lub całej partycji. Rozmiar obu nie może być łatwo zmieniony.

TAK



Page cache

- Pamięć operacyjna **nigdy nie jest pusta** – wolne strony mogą zostać wykorzystane do innych celów.
- Page cache (disk cache, buffer cache).
- Zapis do pliku jest najpierw dokonywany w **buforze**.
- Drugi odczyt pliku wykonywany na szybkiej pamięci.
- Rozmiar buforu zależny od „wolnej” pamięci.

SPRAWDŹ SIĘ!

Ładowanie małych (np. kilkaset bajtów) plików do pamięci page cache jest wydajne pod względem wykorzystania pamięci.

TAK

NIE



SPRAWDŹ SIĘ!

Ładowanie małych (np. kilkaset bajtów) plików do pamięci page cache jest wydajne pod względem wykorzystania pamięci.

TAK



Nakładkowanie

- Technika wykorzystywania **jednego** fragmentu pamięci do ładowania **wielu** fragmentów pliku – tylko jeden naraz.
- Musi być zdefiniowana przez programistę.
- Fragmenty pamięci organizowane w postaci **drzewa**.
- Mało użyteczna w czasach wszechobecnej pamięci wirtualnej.
- Może być przydatna dla systemów wbudowanych (brak MMU) i aplikacji czasu rzeczywistego.

Mapowanie plików

- Dostęp do pamięci jest rzędu wielkości **szyszy** niż wywołanie systemowe i dostęp do dysku.
- Możliwość dostępu do dowolnego fragmentu pliku.
- Ładowanie fragmentów plików (lazy loading).
- **Problem fragmentacji** w przypadku małych plików.
- Problem dużych plików przy małym rozmiarze pamięci logicznej/fizycznej.
- Błędy I/O powodują błędy **SIGSEGV** lub **SIGBUS**.
- Koszt związany z błędami stron.

Brak pamięci

- **Out of memory** (OOM) – brak pamięci fizycznej (oraz przestrzeni swap) na bieżące potrzeby.
- Obecnie ma mniejsze znaczenie, ale wciąż jest spotykane.
- Proces typu **OOM Killer** uruchamiany w przypadku krytycznie małej ilości wolnej pamięci.
- **OOM Killer** zakończy część procesów.

Czym jest wyciek pamięci?

Wyciek pamięci to sytuacja, gdy fragment pamięci jest już niepotrzebny, ale nie został zwolniony.

Wyciek pamięci

- Powoduje powstanie fragmentów pamięci, których nie można wykorzystać.
- Po zakończeniu procesu:
 1. Zasoby (w tym pamięć) są automatycznie zwalnianie przez system.
 2. Dotyczy to m.in. fragmentów sterty nie zwolnionych za pomocą funkcji free() lub operatora delete.
 3. Zasoby mogą być wykorzystane przez inny proces – brak wycieku pamięci.
- Proces zombie nie powoduje wycieku pamięci.

SPRAWDŹ SIĘ!

Jeśli proces zaalokował pamięć
przez malloc()/new(), lecz nie
zwolnił jej poprzez free()/delete()
to po zamknięciu takiego procesu
dojdzie do wycieku pamięci.



TAK

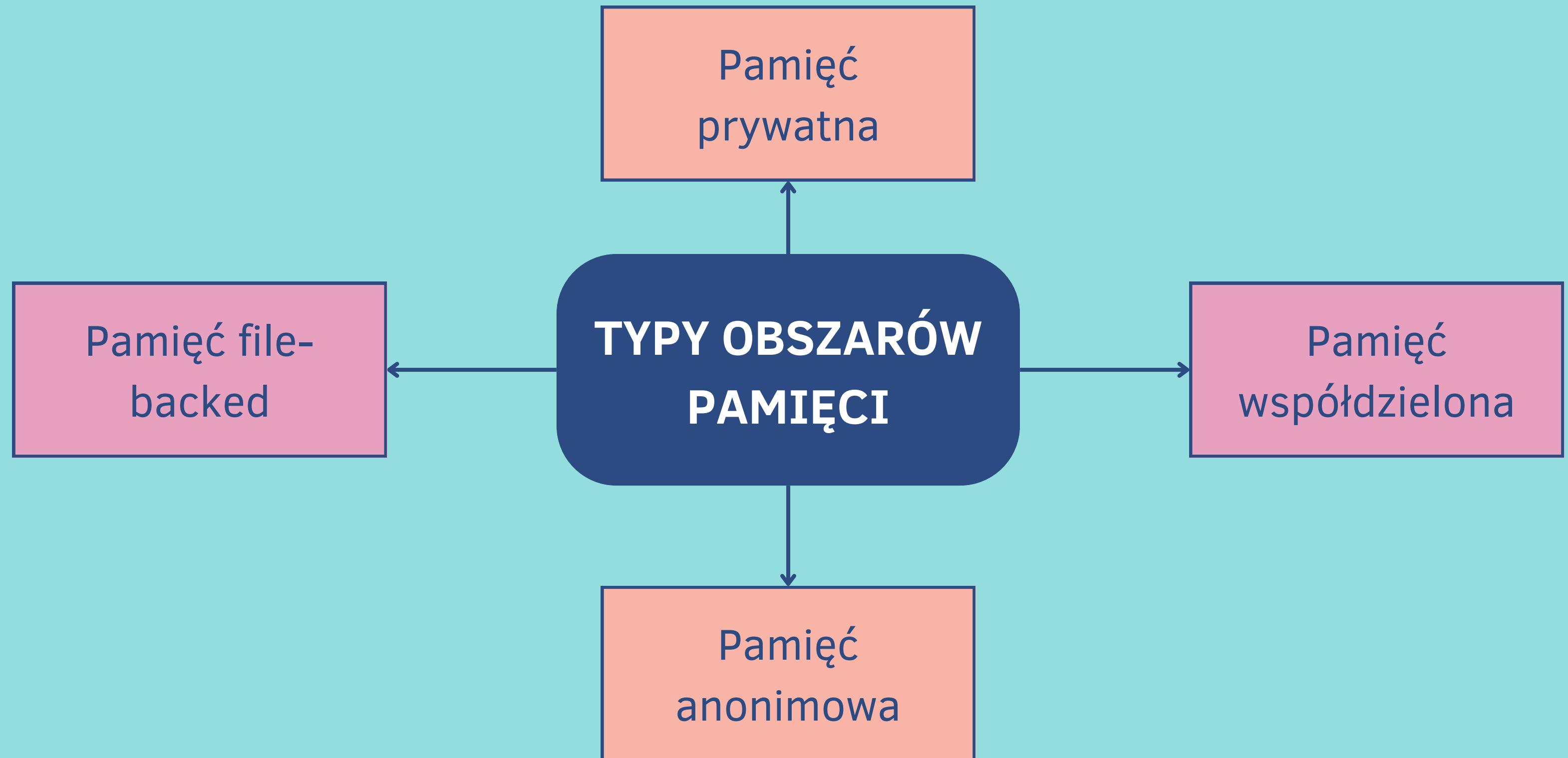
NIE

SPRAWDŹ SIĘ!

Jeśli proces zaalokował pamięć
przez malloc()/new(), lecz nie
zwolnił jej poprzez free()/delete()
to po zamknięciu takiego procesu
dojdzie do wycieku pamięci.

TAK





Pamięć prywatna

- Dostępna dla konkretnego, pojedynczego procesu.
- **Większość** wykorzystywanej pamięci.
- Procesy mogą „współdzielić” prywatne fragmenty danych/bibliotek tylko do odczytu.
- W wersji file-backed zmiany nie są zapisywane do pliku, zmiany w pliku mogą nie być odzwierciedlone w pamięci.

Pamięć współdzielona

- Przeciwieństwo pamięci prywatnej.
- Stosowana do komunikacji międzyprocesowej.
- Dostęp za pomocą rodziny wywołań systemowych shm* lub odpowiedniego wywołania mmap.
- Zmiana dokonana przez jeden proces jest widoczna we wszystkich procesach współdzielących pamięć.
- W wersji file-backed zmiany będą widoczne także w samym pliku.

Pamięć anonimowa

- Znajduje się wyłącznie w **RAM** (nieprzypisana do pliku).
- Może być jednak **wymieniona** (swap).
- Możliwość **zarezerwowania** dużej ilości bez natychmiastowego przypisania pamięci fizycznej.
- **Memory overcommitment.**

Pamięć file-backed

- Przeciwieństwo pamięci anonimowej.
- Dane ładowane z pliku.
- Powiązanie z systemem plików i konkretnym deskryptorem.
- Możliwość ładowania fragmentarycznego.
- Możliwość ładowania prefetch.

SPRAWDŹ SIĘ!

Obszar pamięci stosu należy do kategorii: (A) anonymous, (B) file-backed.

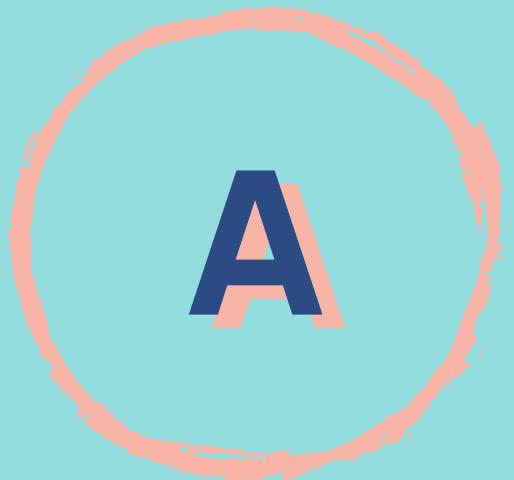
A

B



SPRAWDŹ SIĘ!

Obszar pamięci stosu należy do kategorii: (A) anonymous, (B) file-backed.



B



Komenda free

Podstawowa informacja o pamięci wolnej i zajętej przez wszystkie procesy (całkowita zainstalowana pamięć fizyczna plus swap).

Zwraca total / used / free / shared / buff cache / available

-b, -k, -m, -g -> wielkość w bajtach, kB, MiB, GiB

--kilo, --mega, --giga -> wielkość w kB, MB, ...

-h -> wielkości w najmniejszej sensownej jednostce

-s -> odświeżanie co podany czas

Pamięć procesu

Podstawowa informacja dla procesu PID umieszczona w /proc/PID/statm:

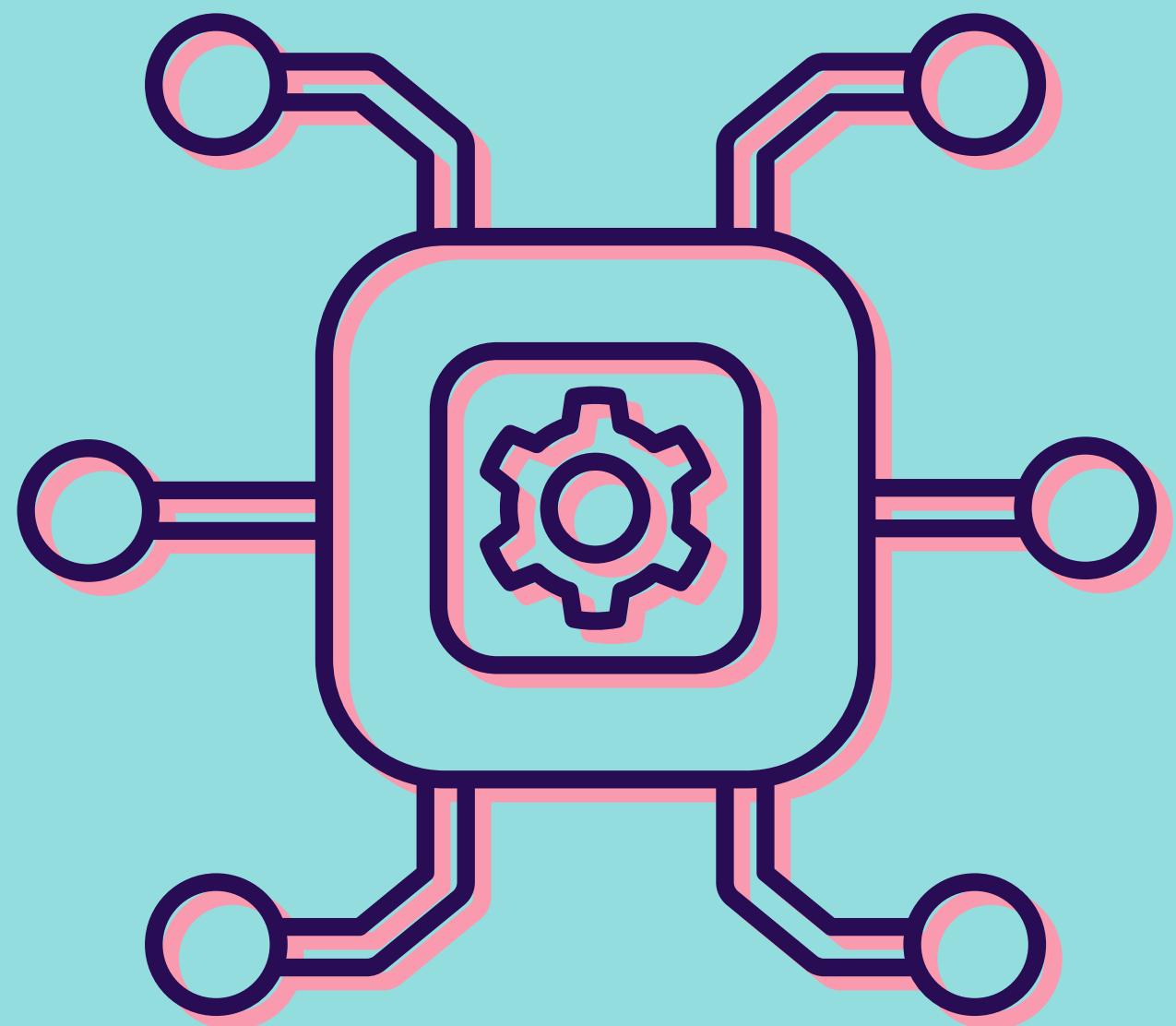
- **size** – wszystkie strony.
- **resident** – strony w pamięci fizycznej.
- **shared** – rezydentne współdzielone strony.
- **text** – kod programu.
- **data** – data oraz stos.

Część wartości może być niedokładna.
Dokładne wyniki w /proc/PID/smaps.



Wykład 9

Wątki, programowanie wielowątkowe



Wątki definicje

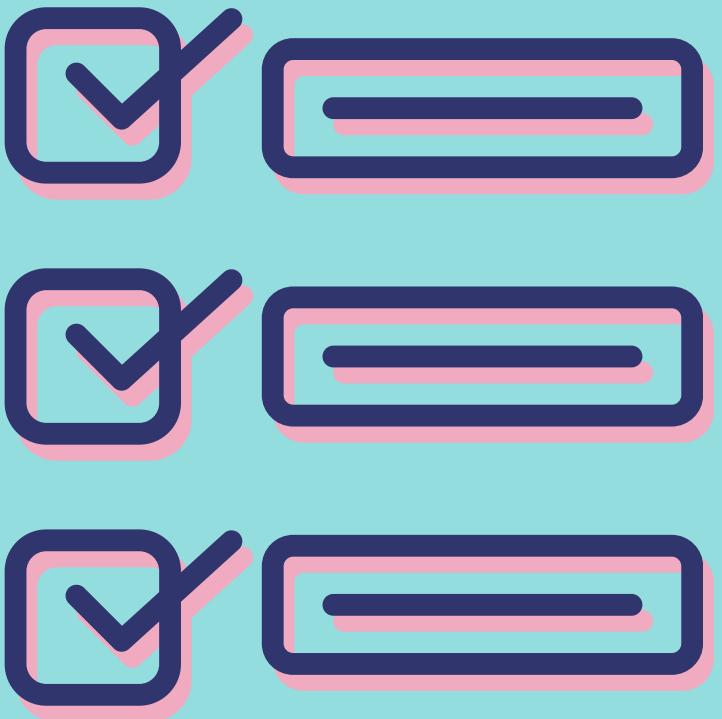


Thread (of execution)

- Wydzielona część programu wykonywana współbieżnie w ramach procesu.
- Najmniejsza jednostka kodu wykonywalnego podlegająca szeregowaniu przez system operacyjny.

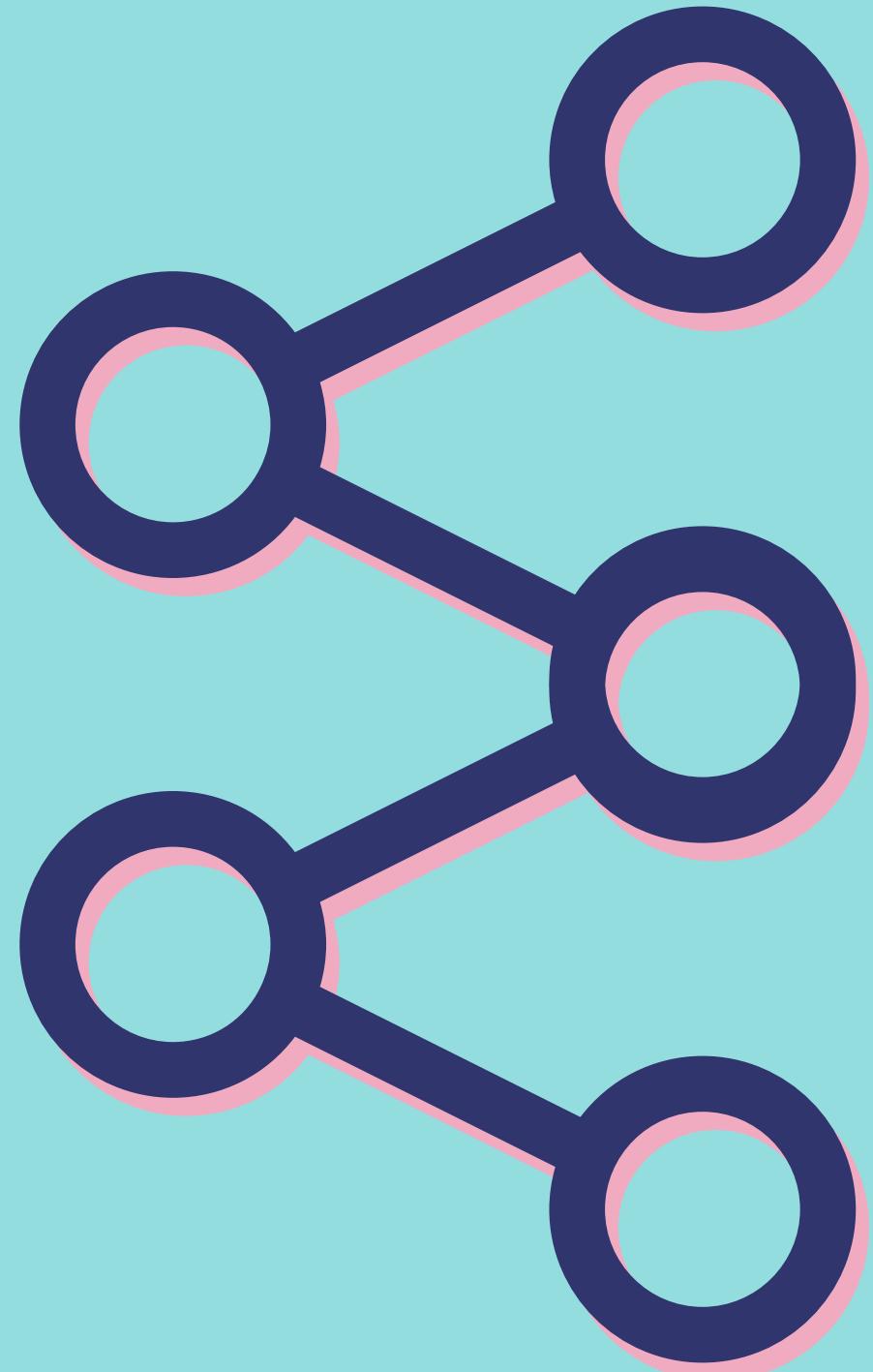
Wątki (2)

- Wątki nazywane są też lekkimi procesami (lightweight process, LWP)
- Proces składa się z jednego lub więcej wątków.
- Wątek główny i wątki poboczne.
- Każdy wątek posiada unikalny identyfikator (TID).
- System operacyjny dostarcza interfejsów oraz implementacji do obsługi wątków (w tym synchronizacji).



Wątki a procesy

- Procesy są “jednostką” zasobów, wątki są “jednostką” kodu/wykonania.
- Wątki współdzielą przestrzeń adresową (poza stosem) oraz większość innych zasobów procesu.
- Błędy we wspólnej przestrzeni.
- Procesy nie współdzielą przestrzeni adresowej (poza bibliotekami, pamięcią dzieloną itp.).
- Komunikacja między wątkami jest szybsza (ale wciąż wymaga synchronizacji).
- Wątki są szybsze w tworzeniu i przełączaniu kontekstu niż procesy.



SPRAWDŹ SIĘ!

Wpływ wspólnej przestrzeni adresowej wątków na konsekwencje błędów jest: (A) pozytywny, (B) negatywny.

A

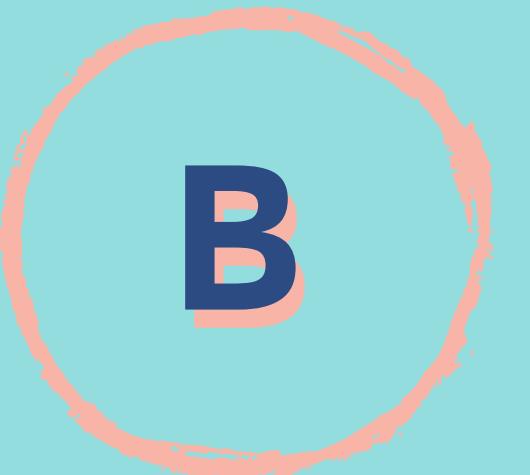
B

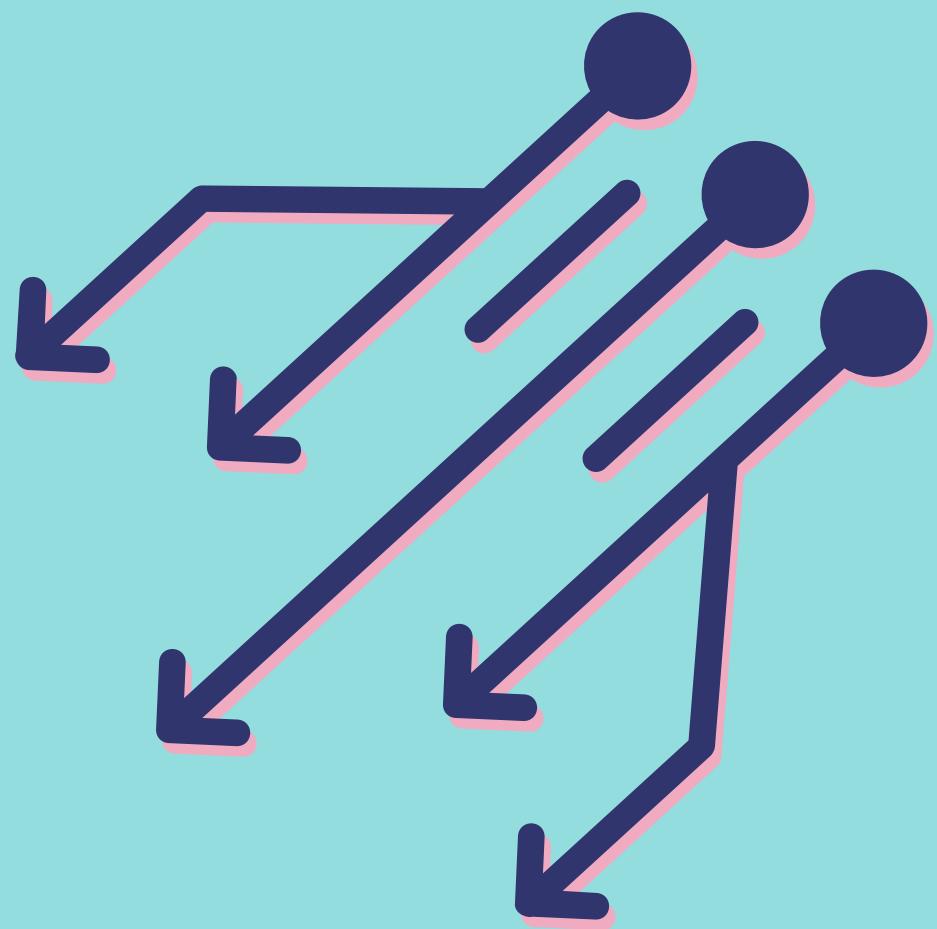


SPRAWDŹ SIĘ!

Wpływ wspólnej przestrzeni adresowej wątków na konsekwencje błędów jest: (A) pozytywny, (B) negatywny.

A





Współbieżność a równoległość

Współbieżność a równoległość

- **Procesy sekwencyjne**

Procesy są sekwencyjne jeżeli następny proces ze zbiory procesów rozpoczyna się po zakończeniu procesu poprzedniego.

- **Procesy współbieżne**

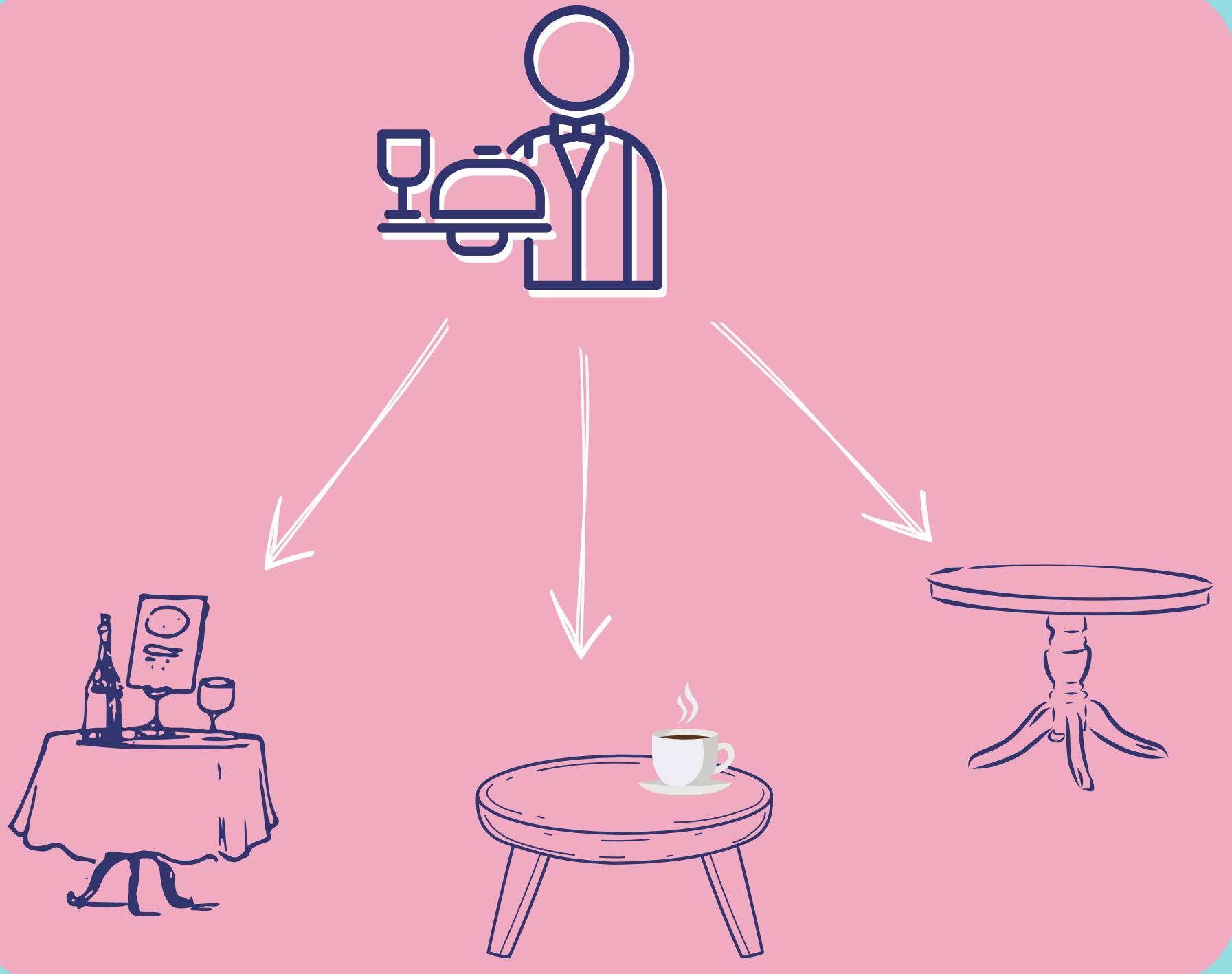
Dwa procesy są współbieżne jeżeli jeden z nich rozpoczyna się przed zakończeniem drugiego.

- **Procesy równoległe**

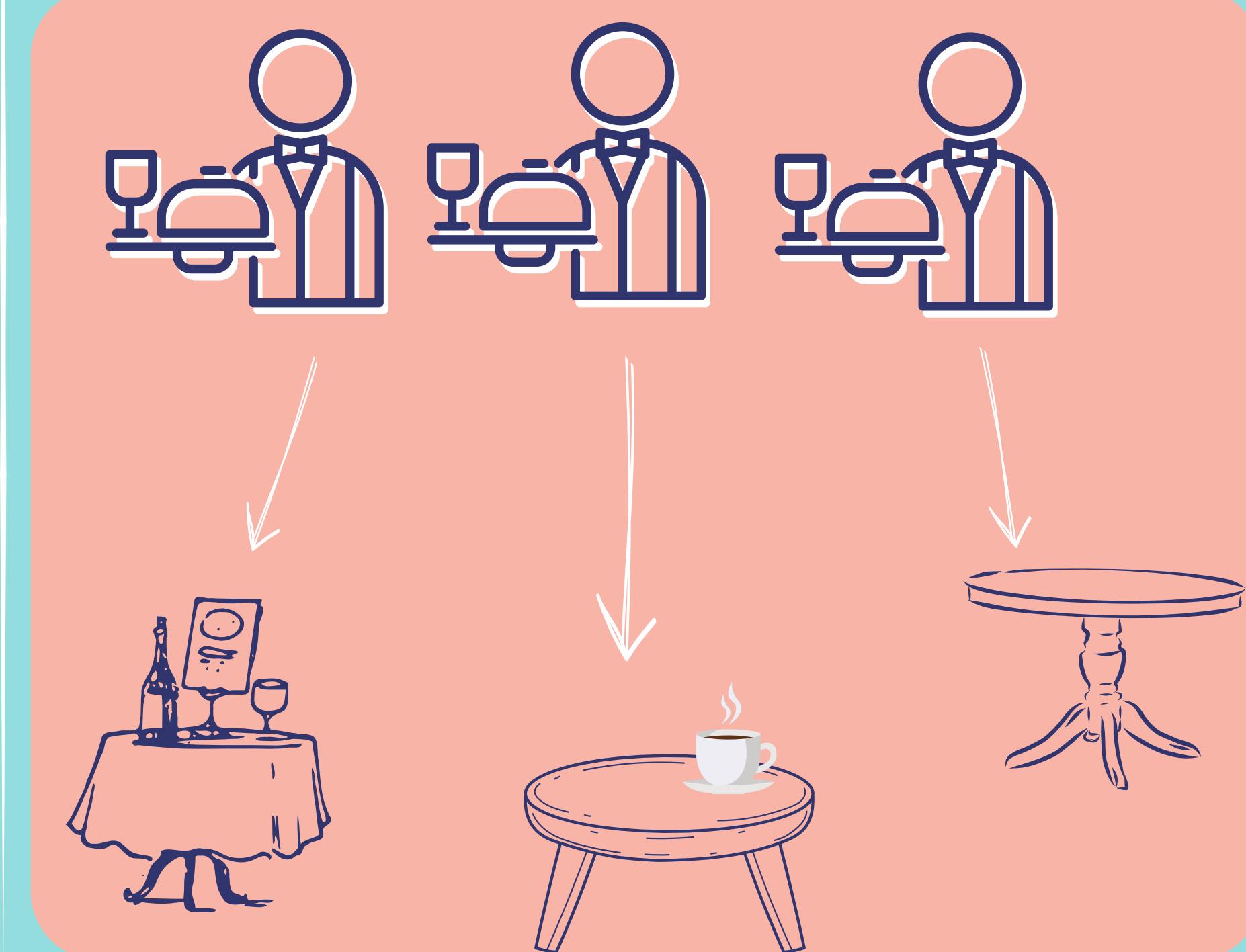
Dwa procesy są równoległe jeżeli jeden z nich rozpoczyna się przed zakończeniem drugiego i wykonywane są jednocześnie na oddzielnych procesorach.



Współbieżność (Concurrency)



Równoległość (Parallelism)



Współbieżność (Concurrency)

Jesteś jeden, zarządzasz wieloma stolikami, ale w danym momencie zajmujesz się tylko jednym.

Równoległość (Parallelism)

Jest wielu z was, każdy zajmuje się innym stolikiem, wszystko dzieje się na raz.



SPRAWDŹ SIĘ!

Aby procesy p_1 i p_2 były współbieżne, musi istnieć chwila czasu t , w której oba procesy już się zaczęły, ale żaden z nich się jeszcze nie skończył.

TAK

NIE



SPRAWDŹ SIĘ!

Aby procesy p_1 i p_2 były współbieżne, musi istnieć chwila czasu t , w której oba procesy już się zaczęły, ale żaden z nich się jeszcze nie skończył.



NIE



PThreads

PThreads, znane również jako **POSIX threads**, to standardowa biblioteka wielowątkowa dla systemów Linux, umożliwiająca efektywne tworzenie i zarządzanie wątkami w aplikacjach C i C++. Kluczowe cechy obejmują:

- Linkowanie z opcją `-pthread` dla optymalnego zarządzania wątkami.
- Model jeden do jednego, zapewniający każdemu wątkowi użytkownika odpowiadający wątek jądra.
- Kompleksowe wsparcie systemowe obejmujące funkcje biblioteczne, wywołania systemowe, oraz metody komunikacji i synchronizacji wątków.
- Bogata dokumentacja wspierająca programistów w implementacji i debugowaniu.

creatingThread.cpp



Synchronizacja procesu/wątku

**Wymuszenie zależności czasowych
(synchronizacja w czasie) pomiędzy
czynnościami dwóch lub więcej
procesów/wątków.**

Zależność może być „wprost” (jednoczesne rozpoczęcie czynności) lub „odwrotna” (jeden proces zaczyna czynność gdy inny jakiś kończy).

Synchronizacja i komunikacja to nie to samo:

- 
- ▶ Synchronizacja korzysta z komunikacji.
 - ▶ Komunikacja może być synchroniczna lub asynchroniczna.
 - ▶ Blokowanie jest nieodłącznym elementem poprawnej synchronizacji.

Kończenie wątków:

Możliwe sposoby zakończenia wątku:

- **Jawne wywołanie funkcji pthread exit(), wraz z podaniem wskaźnika.**
- **Wyjście (niejawne lub jawne return) z funkcji wątku.**
- **Anulowanie wątku (pthread cancel()).**
- **Którykolwiek wątek wykona exit().**
- **Wątek główny wyjdzie z funkcji main().**
- **Zakończenie wszystkich wątków kończy proces.**
- **Wartość zwracana wątku.**

JOIN

- Funkcja `pthread join()` jest wątkowym odpowiednikiem procesowego `wait()`.
- Czeka na zakończenie podanego wątku (chyba że już się zakończył) i zapisuje jego kod powrotu (lub PTHREAD CANCELED).
- Synchronizuje wyjście z `pthread join()` z zakończeniem wątku.
- Wątek musi być joinable.
- Join na już „zjoinowanym” wątku stanowi niezdefiniowane zachowanie.
- Wiele jednoczesnych joinów stanowi niezdefiniowane zachowanie (wyścig).
- Wątki zombie.

SPRAWDŹ SIĘ!

Wywołanie T.join() po zakończeniu wątku T: (A) jest błędem, (B) może prowadzić do niespójności.

A

B



**pthread cancel() wysyła żądanie
anulowania do wątku (i od razu wraca).**

Stan anulowania działa jak maska:

- **włączony (domyślnie).**
- **wyłączony (do ustawiania przez pthread setcancelstate()) – żądanie zakolejkowane aż stan będzie włączony.**

Typ anulowania:

- **deferred** – anulowanie nastapi przy najbliższym tzw.
punkcie anulowania.
- **asynchronous** – anulowanie natychmiast.

Anulowanie poprzez sygnały czasu rzeczywistego.

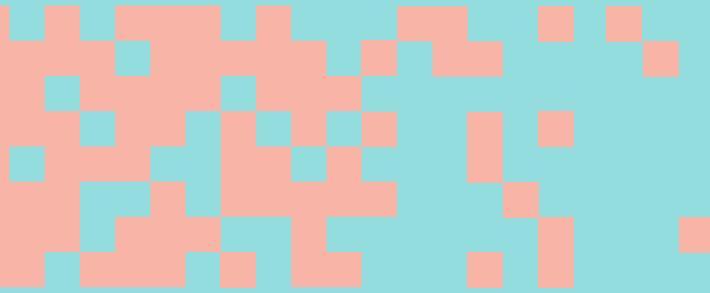


Problemy programowania współbieżnego:

- Problem spójności danych.
- Problem zagłodzenia.
- Problem zakleszczenia.
- Livelock.



Problem spójności danych



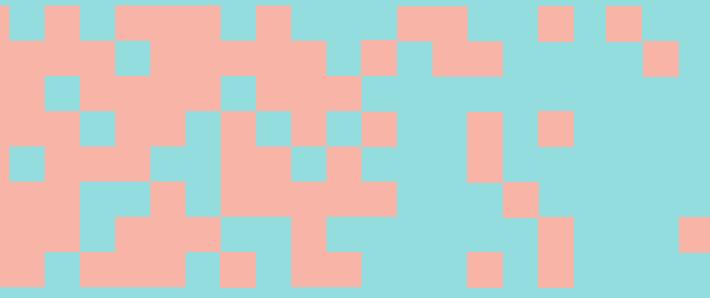
- Problem synchronizacji (problem sekcji krytycznej, problem konta bankowego).

Sekcja krytyczna

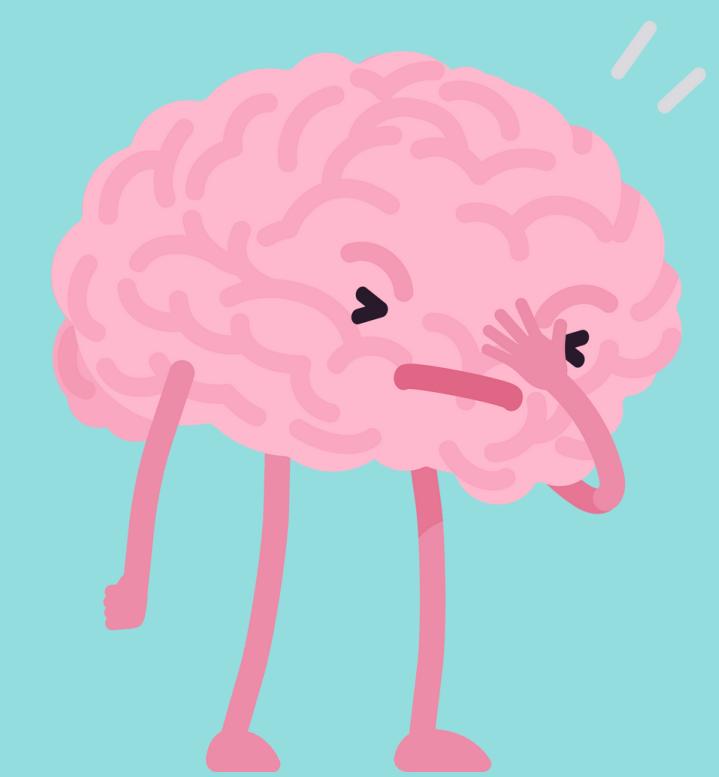
- (Ciągły) fragment kodu korzystający z zasobu dzielonego, którego wykonanie przez wiele wątków musi być ograniczone.



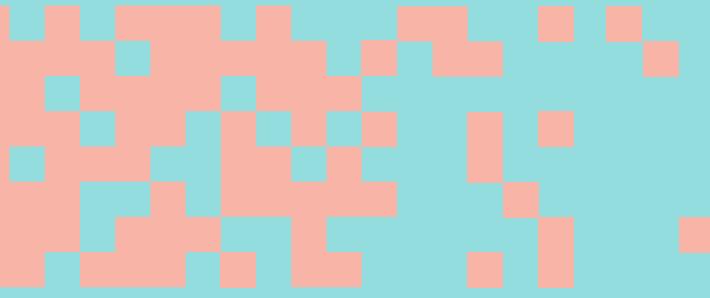
Problem spójności danych



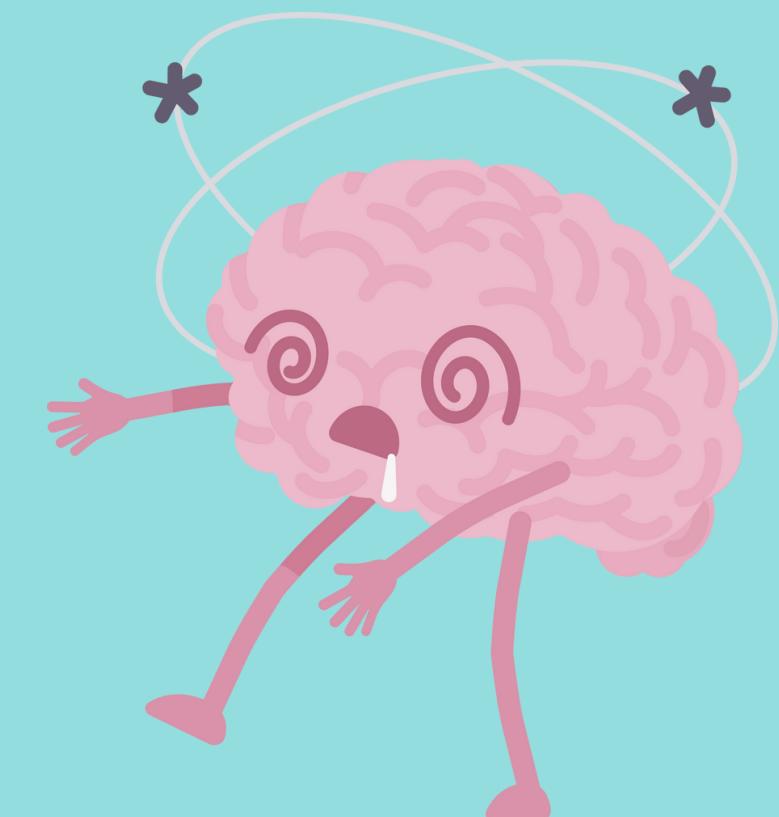
- Sekcja krytyczna powinna być atomowa – powinna wykonać się cała lub nie wykonać się wcale (w przybliżeniu odpowiada to transakcji z baz danych).
- Najczęściej sekcję krytyczną może wykonywać tylko jeden wątek naraz (algorytm typu **MUTEX - MUTual EXclusion**) – jeśli jeden wątek wykonuje sekcję krytyczną, inny wątek nie może być do niej wpuszczone.
- Sporadyczne sytuacje gdy do sekcji krytycznej może wejść więcej niż jeden wątek (wielokortny zasób).



Problem spójności danych



- Problem spójności występuje zarówno przy zapisie jak i odczycie („brudny odczyt”, CPU cache).
- Szczególnie widoczny w przypadku modyfikacji całych struktur lub obiektów.
- Występuje też przy modyfikacji pojedynczej zmiennej, nawet boolowskiej – to czy zostanie to zrobione pojedynczą instrukcją zależy od architektury i implementacji.
- Zbyt krótka (ale też zbyt dобра!) sekcja krytyczna może być błędna.
- Zbyt dобра sekcja może być niewydajna (ogranicza współbieżność/równoległość).
- Spójność jest ważniejsza niż wydajność!



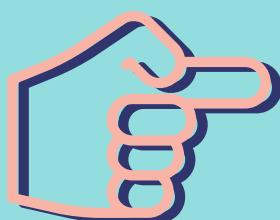
3 algorytmy rozwiązuające problem sekcji krytycznej.



Algorytm Dekkera



Algorytm Petersona



Algorytm piekarniany



Blokady (zamek)

Mechanizm synchronizacji oraz ADT(Abstract Data Type)/struktura danych kontrolująca dostęp do zasobu (sekcji krytycznej).

- Blokady obowiązkowe lub opcjonalne
- Konieczne zapewnienie atomowości
- Blokada może opierać się na ciągłym sprawdzaniu stanu (polling) lub na blokowaniu do czasu zwolnienia zasobu
- Przykłady: semafory, mutexy, monitory

SPRAWDŹ SIĘ!

Odczyt w wątku zmiennej, którą może modyfikować inny wątek jest (A) zawsze bezpieczne, (B) może prowadzić do niespójności.

A

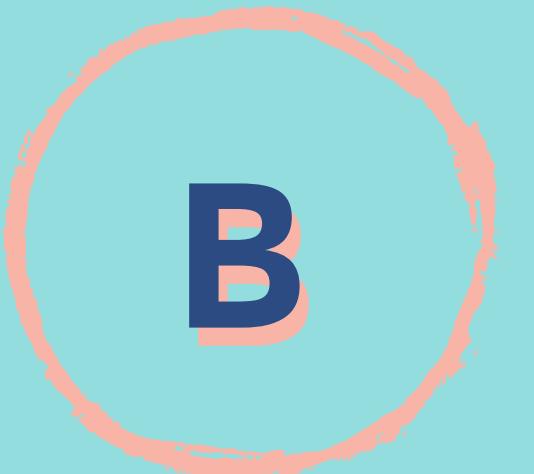
B



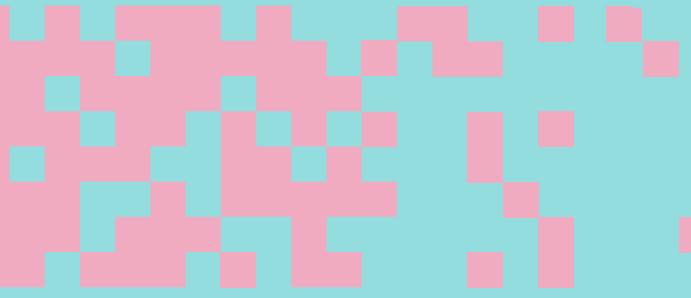
SPRAWDŹ SIĘ!

Odczyt w wątku zmiennej, którą może modyfikować inny wątek jest (A) zawsze bezpieczne, (B) może prowadzić do niespójności.

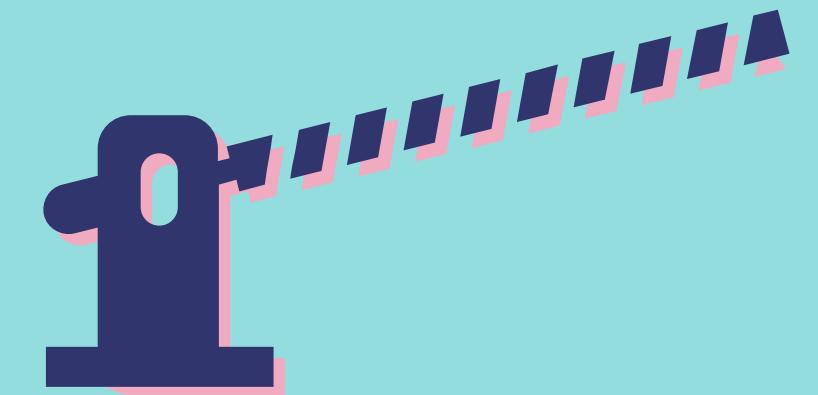
A



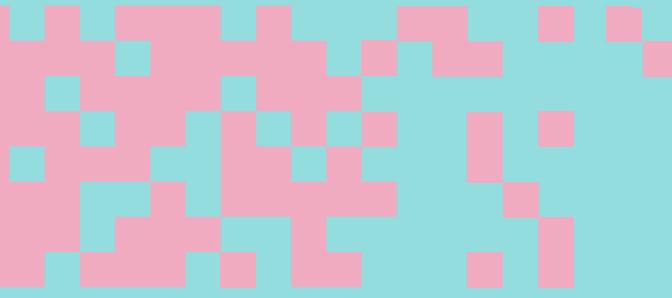
Semafora



- Wprowadzone przez Dijkstrę jako rozwinięcie algorytmu Dekkera.
- Licznik oznaczający liczbę wolnych egzemplarzy zasobu.
 - 1) Semafora binarne.
 - 2) Semafora zliczające.
- Atomowa operacja zajęcia (wait, lock, acquire, down, P) – czekaj dopóki licznik dodatni, po czym zmniejsz licznik.
- Atomowa operacja zwolnienia (signal/post, unlock, release, up, V) – zwiększ licznik.



Semafony w Linuksie



- Semafony POSIX-owe mogą być zarówno współdzielone przez wątki jak i przez procesy.
- Semafony anonimowe/nienazwane (unnamed).
- Semafony nazwane (named).
- Zajmowanie: sem wait(), sem try wait(), sem timed wait().
- Zwalnianie: sem post().
- Konieczne podanie adresu struktury semafora.
- Semafor może zostać zwolniony przez dowolny proces mający do niego dostęp.



SPRAWDŹ SIĘ!

W Linuksie semafor nazwany może być użyty przez wątki należące do różnych procesów.

TAK

NIE



SPRAWDŹ SIĘ!

W Linuksie semafor nazwany może być użyty przez wątki należące do różnych procesów.

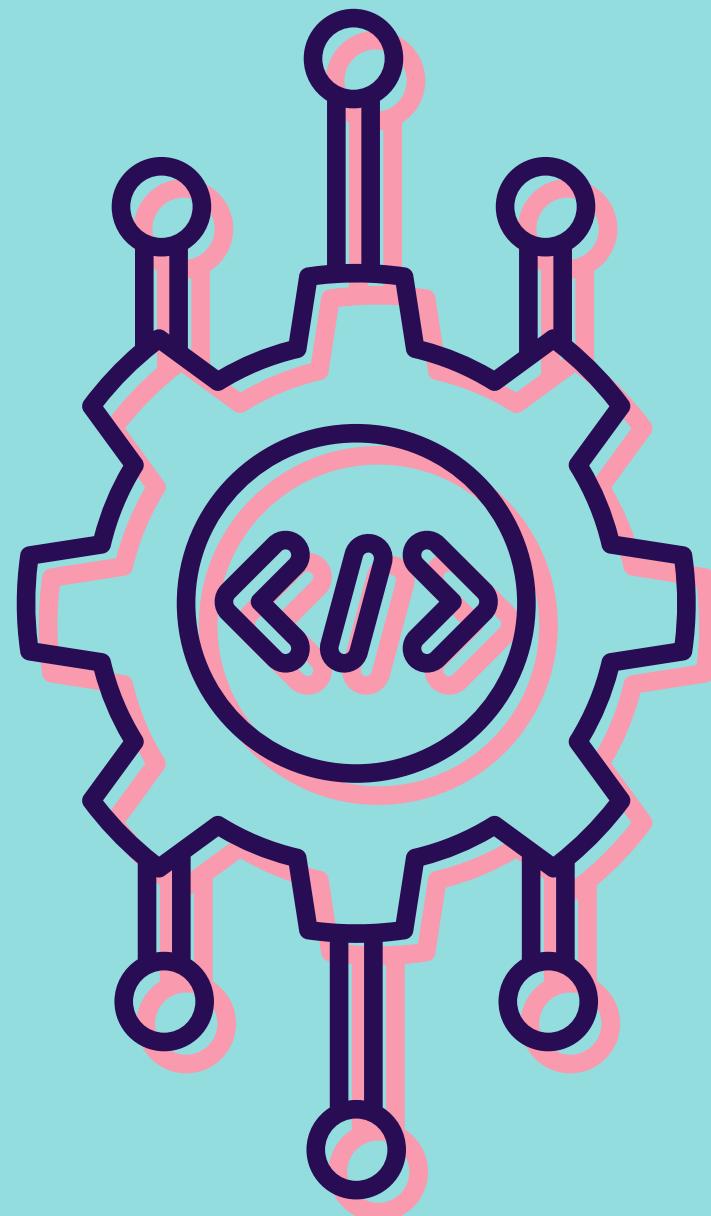


NIE



Wykład 10

Programowanie wielowątkowe





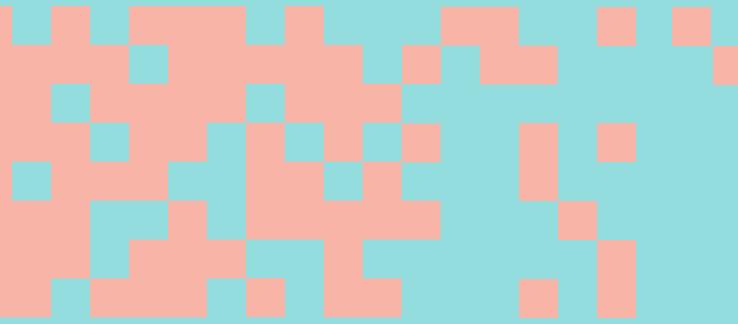
Mutexy

- Mutexy są bardziej zaawansowane niż zwykłe semafory.
- Pojęcie właściciela – thread owns mutex after locking it.
- Tylko właściciel może odblokować dany mutex.
- Różne typy mutexów.
- W pthreads mutexy domyślnie działają w obrębie procesu.

Przed użyciem zmienną typu mutex należy zainicjalizować:

- Funkcja pthread mutex init(), do której podajemy mutex i atrybuty (NULL dla domyślnych).
- Dla mutexów globalnych z domyślnymi atrybutami możliwy zapis pthread mutex t zmienna = PTHREAD_MUTEX_INITIALIZER;
- Po zakończeniu pracy, na mutexie należy wykonać funkcję pthread mutex destroy()
- Użycie na zablokowanym (lock) mutexie to nieokreślone zachowanie.

Mutexy - podstawowe funkcje



→ **pthread mutex lock()**

→ **pthread mutex unlock()**

→ **pthread mutex trylock()**

- Synchronizacja odbywa się na mutexie, nie na jego kopiiach!
- Mutex można przekazywać przez wskaźnik, ewentualnie przez referencję.



SPRAWDŹ SIĘ!

W pthreds zajęcie (lock) mutexa
przez wywołaniem wait zmiennej
warunkowej jest opcjonalne.

TAK

NIE



Zmienne warunkowe

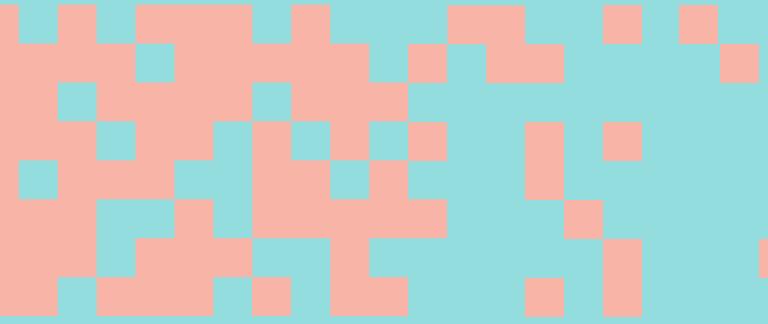
- Pozwalają na kontrolowane wejście do sekcji krytycznej w zależności od spełnienia określonych warunków.
- Są ściśle powiązane z mutexami, co umożliwia bezpieczne zarządzanie dostępem do zasobów.
- Wzorce pthread cond wait i pthread cond timewait pozwalają wątkom oczekiwania na spełnienie warunków, jednocześnie uwalniając mutexy, aby uniknąć zakleszczenia.
- Inicjalizacja, obsługa i niszczenie zmiennych warunkowych są podobne do obsługi mutexów, z dodatkowymi funkcjami jak pthread cond signal i pthread cond broadcast, które odpowiednio wybudzają jeden lub wszystkie oczekujące wątki.

Zmienne warunkowe

-o czym warto pamiętać

- Ważne jest, aby pamiętać o potencjalnych fałszywych wybudzeniach, co wymaga ponownego sprawdzenia warunków po wybudzeniu.
- Użytkowanie zmiennych warunkowych wymaga starannej kontroli i projektowania, aby zapewnić efektywne i bezpieczne wykonanie w środowisku wielowątkowym.

Monitory



- W podejściu obiektowym sekcje byłyby metodami obiektu, a chronione przez nie zmienne – jego stanem.
- Blokada realizowana dowolnym obiektem, a nie dedykowanym mutexem.
- Powyższe prowadzą nas do idei monitora.

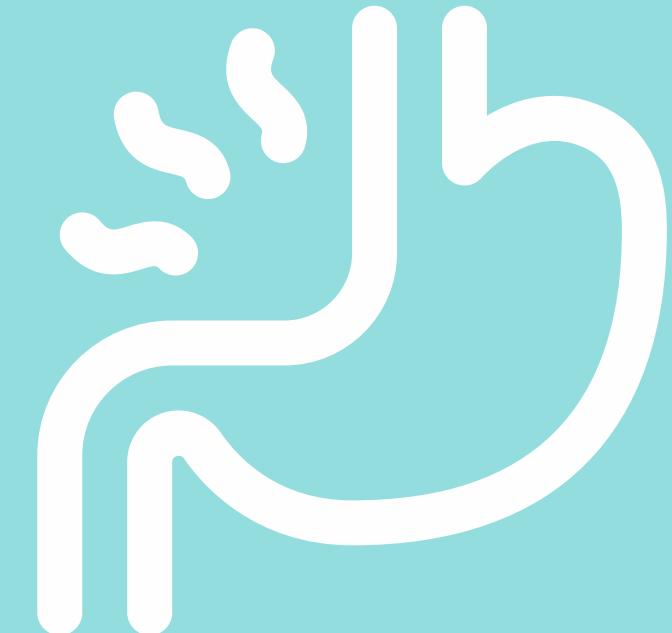
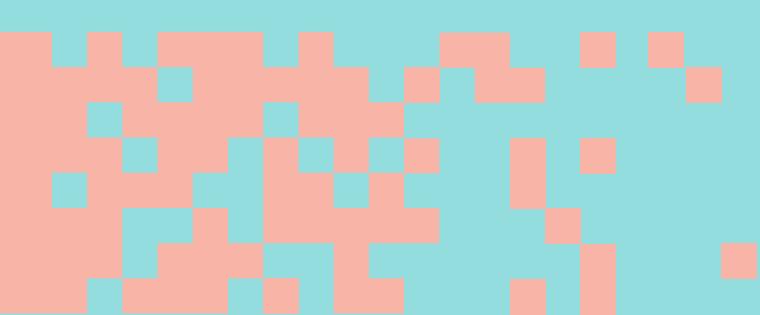
- Posłużymy się przykładem języka Java (kod na następnym slajdzie):
- Każdy obiekt implementuje metody `wait()`, `notify()` oraz `notifyAll()` – odpowiednik zmiennej warunkowej.
- Dostępne tradycyjne blokady/mutexy z metodami `lock()` i `unlock()`.

Przykład reprezentacji monitora



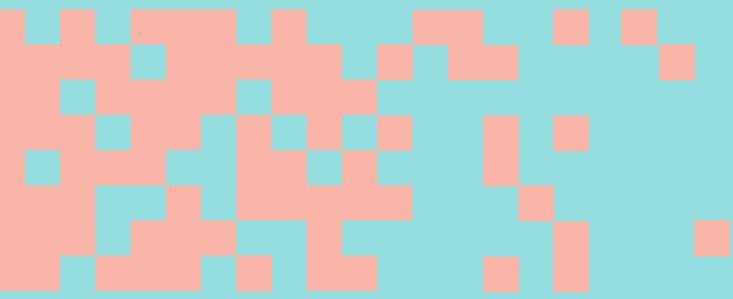
```
1 class MultiThreadCounter {  
2     private int counter = 0;  
3     private static int globalCounter = 0;  
4     public synchronized void increment() {  
5         counter++;  
6     }  
7     public static synchronized void incrementGlobal() {  
8         globalCounter++;  
9     }  
10 }
```

Zagłodzenie (Starvation)



- To sytuacja gdy proces/wątek nie może zakończyć działania (brak postępów) ze względu na brak przydziału zasobów.
- Dochodzi do tego, kiedy zasoby nie zostaną przydzielone nigdy lub nie zostaną przydzielone zbyt długo.
- Na poziomie systemu operacyjnego ten problem zwykle nie występuje – planista systemowy odpowiednio kolejkuje żądania dostępu do sekcji krytycznej
- Każdy zgłaszający wątek w końcu otrzyma dostęp do sekcji (zakładając odpowiednie unlock i signal/broadcast).
- Brak zagłodzeń i zakleszczeń.
- Gwarancja pewnego poziomu uczciwości (fairness).

Zakleszczenie (Deadlock)



- To sytuacja gdy co najmniej dwa wątki/procesy czekają na siebie nawzajem, tak że żaden nie może dokonać postępu (zmienić stanu).
- Livelock – procesy zmieniają stan, ale jednak nie dokonują postępu.

Zakleszczenie wystąpi tylko jeśli spełnione są łącznie 4 warunki:

- Wzajemne wykluczenie (mutual exclusion) zasobów.
- Przetrzymwanie zasobów w oczekiwaniu na kolejne.
- Żądania tworzy cykliczny graf skierowany.
- Brak wywłaszczeń – zasoby nie są oddawane bez wykonania zadania.

Zakleszczenie - ciąg dalszy

- Zakleszczenie jest znacznie bardziej niebezpieczne niż zagłodzenie.
- Zakleszczenie dotyczy zwykle grupy wątków (co najmniej 2).
- Zakleszczone wątki są zagłodzone.
- Zakleszczone wątki mogą zakleszczyć/zagłodzić inne wątki, które zaczną na nie czekać.
- Zakleszczenie nigdy nie rozwiąże się samo.

Sposoby na radzenie sobie z zakleszczeniami:

- Ignorowanie
- Zapobieganie
- Detekcja zakleszczenia,
które wystąpiło lub nastąpi

SPRAWDŹ SIĘ!

Zakleszczenie niekoniecznie prowadzi do zagłodzenia.

TAK

NIE



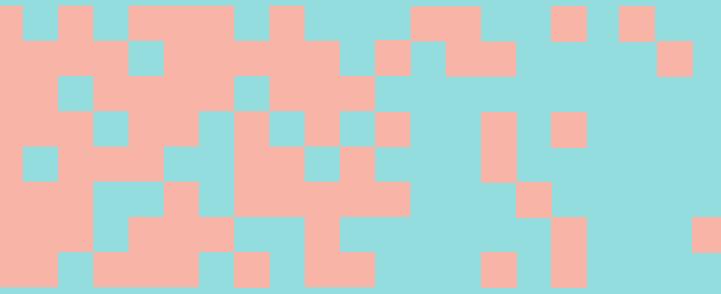
SPRAWDŹ SIĘ!

Zakleszczenie niekoniecznie prowadzi do zagłodzenia.

TAK



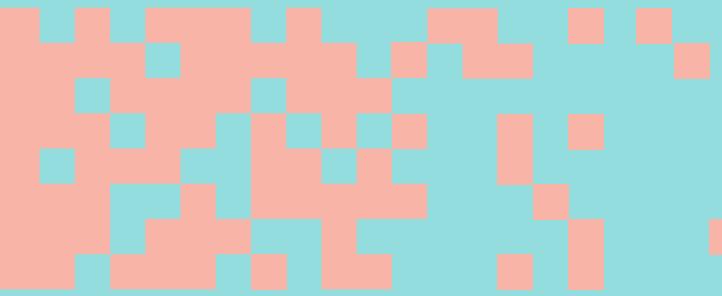
Bariera



To metoda synchronizacji niezwiązańia z kończeniem wątków (w przeciwieństwie do `join()`) lub dostępem do zasobów (w przeciwieństwie do `lock()`).

`pthread barrier init()` – inicializacja: należy podać barierę i liczbę C.
`pthread barrier destroy()` – „zniszczenie” podanej bariery.
`pthread barrier wait()` – czekanie na podanej barierze

Futex



- Fast User-space muTEX – mechanizm i wywołanie systemowe.
- Często cegiełka, z której powstają inne mechanizmy (semafony, zmienna warunkowa).
- Zmienna licznikowa w pamięci.
- Większość operacji odbywa się w przestrzeni użytkownika (sprawdzanie i modyfikacja licznika za pomocą atomowych instrukcji procesora). Jeśli nie ma konfliktów, operacja się kończy.
- Jeśli jest konfikt, wykorzystane jest wywołanie systemowe futex() w celu wykonania operacji FUTEX WAKE lub FUTEX WAIT (dostęp do systemowej kolejki oczekujących).

SPRAWDŹ SIĘ!

W Linuksie (pthreads) próba zablokowania już zablokowanego mutexa zawsze spowoduje zakleszczenie.

TAK

NIE



SPRAWDŹ SIĘ!

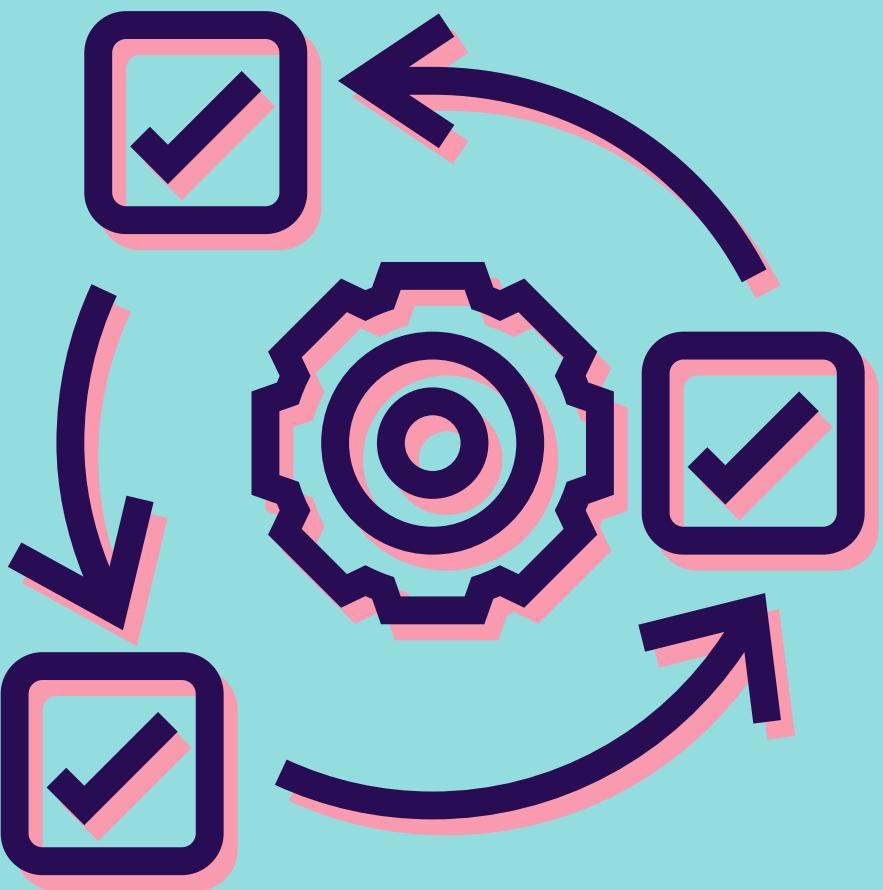
W Linuksie (pthreads) próba zablokowania już zablokowanego mutexa zawsze spowoduje zakleszczenie.

TAK



Wykład 11

Komunikacja międzyprocesowa

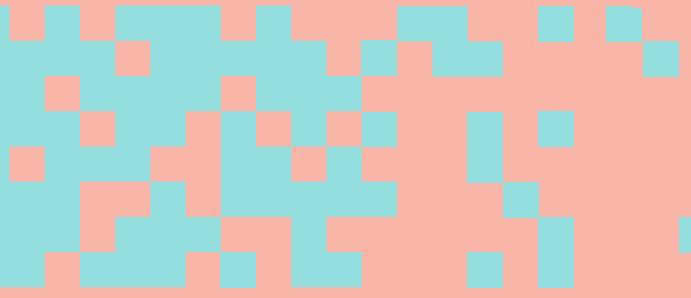


Mechanizmy komunikacji międzyprocesowej

- ▶ Sygnały.
- ▶ Pliki (w tym mapowane).
- ▶ Potoki anonimowe
- ▶ Potoki nazwane.
- ▶ Kolejka komunikatów.
- ▶ Gniazdka Unixowe.
- ▶ Gniazdka sieciowe.
- ▶ Pamięć współdzielona.



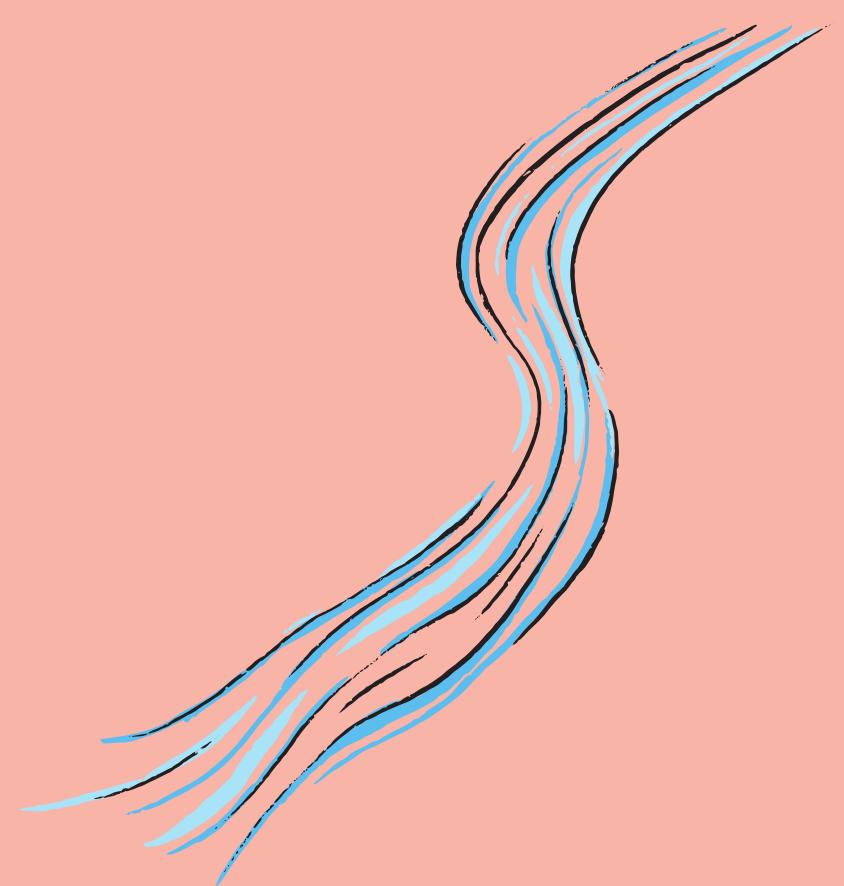
Rodzaje potoków



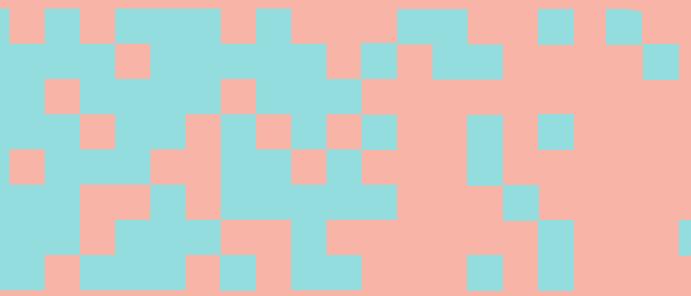
potoki anonimowe



potoki nazwane



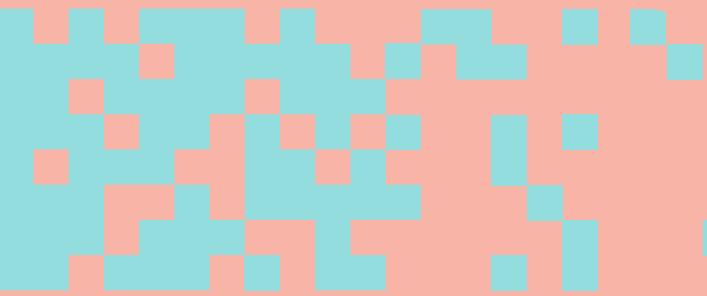
Potoki anonimowe



Anonymous pipe:

- Komunikacja jednokierunkowa.
- Komunikacja dwukierunkowa poprzez dwa przeciwnie skierowane potoki.
- Nieużywany koniec potoku zwykle jest zamykany.
- Użycie jak w przypadku plików (`read()`, `write()`).

Potoki anonimowe



- Komunikacja odbywa się poprzez bufor w jądrze systemu plików.
- Komunikacja niewidoczna dla innych.
- Ograniczona do procesów współdzielących odpowiednie deskryptory plików.
- Wyjście/wejście generowane/konsumowane na bieżąco – zwiększa współbieżność.
- Brak potrzeby używania plików lub dużych zmiennych.
- Potok istnieje tak długo jak procesy.

SPRAWDŹ SIĘ!

Komunikacja potokiem anonimowym (nienazwanym) działa z pośrednictwem bufora w jądrze systemu operacyjnego.

TAK

NIE



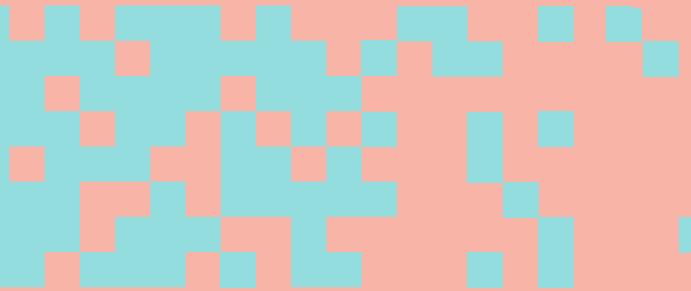
SPRAWDŹ SIĘ!

Komunikacja potokiem anonimowym (nienazwanym) działa z pośrednictwem bufora w jądrze systemu operacyjnego.

TAK



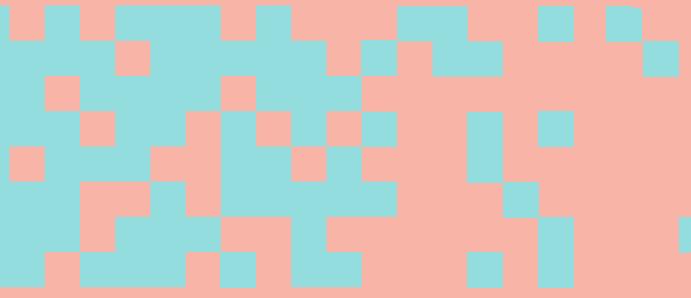
Potoki nazwane



Named pipe (FIFO)

- **Potok identyfikowany przez nazwę (ścieżkę) w systemie plików.**
- **Specjalny typ pliku.**
- **Cechy jak każdy plik (m.in. uprawnienia).**
- **Może być wykorzystany przez dowolny proces, który ma do niego dostęp.**
- **Używany jak zwykły plik.**
- **Typowe komendy wykorzystywane w potoku cat, less, head, tail, sort, grep, cut itp.**

Potoki nazwane



- **Może być używane przez wiele procesów.**
- **Standardowo otwarcie jest blokujące (czeka na otwarcie z drugiej strony).**
- **Możliwe otwarcie nieblokujące lub otwarcie w obie strony (nie w POSIX-ie).**
- **System plików służy jedynie jako referencja – sam plik istnieje (i zachowuje się pomiędzy restartami systemu), ale nie zawiera przesyłanych danych.**
- **Dane przesyłane przez bufor jądra systemu operacyjnego.**

SPRAWDŹ SIĘ!

Potoki FIFO mają odwołanie w systemie plików jako pliki zwykłe (regularne).

TAK

NIE



SPRAWDŹ SIĘ!

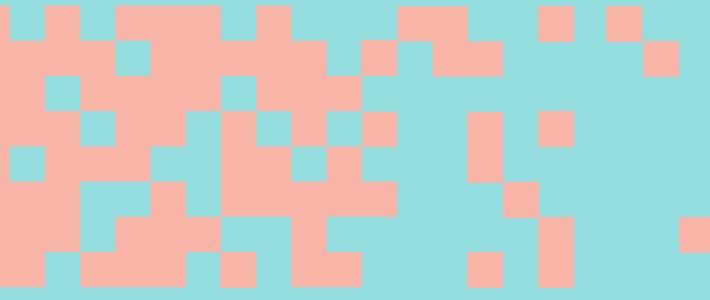
Potoki FIFO mają odwołanie w systemie plików jako pliki zwykłe (regularne).



NIE



Kolejki komunikatów



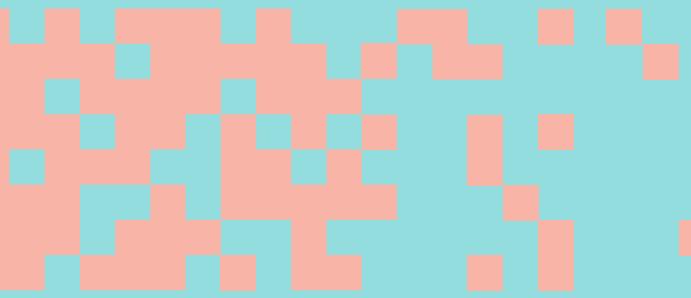
Message queue

- Osobne wiadomości („pakiety”) o określonej długości zamiast nieroróżnialnego ciągu bitów.
- Możliwe ustalenie maksymalnej liczby wiadomości i maksymalnej długości wiadomości.
- Wiadomości mają priorytet – wiadomości kolejkowane są wg priorytetu.
- Kolejka przechowuje atrybuty (maksima, liczba oczekujących wiadomości, flagi).

Dostępne też starsze API dla kolejek komunikatów z System V.
Mniej możliwości, ale szerzej dostępne.



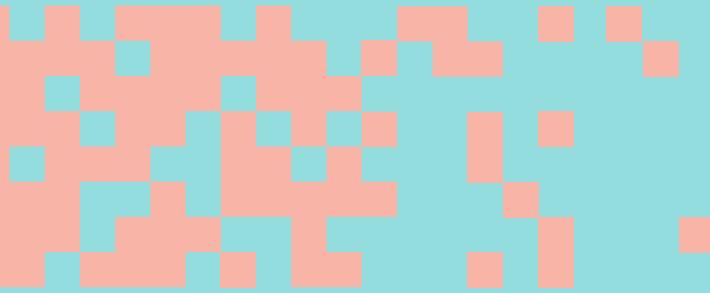
Kolejki komunikatów



Limity

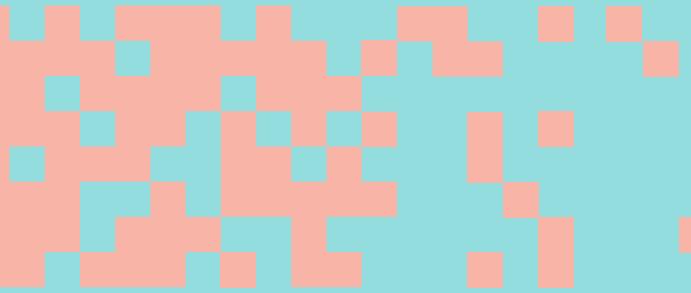
- ▶ Limity dostępne przez wirtualny system plików /proc:
 - Domyślny, maksymalny i twardy maksymalny limit liczby wiadomości.
 - Domyślny, maksymalny i twardy maksymalny limit długości wiadomości.
 - Limit liczby kolejek w systemie.
- ▶ Limit RLIMIT_MSGQUEUE – limit rozmiaru kolejek stworzonych przez danego użytkownika (real UID).

Kolejki komunikatów



- Funkcja mq open() – tworzy lub otwiera istniejącą kolejkę komunikatów.
 - ▶ Funkcja mq close() – zamyka kolejkę dla procesu.
 - ▶ Funkcja mq unlink() – „zaznacza” kolejkę do usunięcia.
- Funkcja mq send() – dodanie wiadomości do kolejki
- Funkcja mq receive() – odebranie wiadomości z kolejki

Kolejki komunikatów



- Do kolejek komunikatów jest osobny wirtualny system plików.
- Montowanie przez:

`mkdir /dev/mqueue`

`mount -t mqueue none /dev/mqueue`

- Kolejka nie jest umocowana na dysku – znika po restarcie systemu.
- Funkcja `mq_notify()` – informowanie o nowej wiadomości (np. sygnałem), tylko jeden proces naraz.

SPRAWDŹ SIĘ!

Kolejność odbierania wiadomości w linuksowej kolejce komunikatów zależy jedynie od kolejności ich wysyłania.

TAK

NIE



SPRAWDŹ SIĘ!

Kolejność odbierania wiadomości w linuksowej kolejce komunikatów zależy jedynie od kolejności ich wysyłania.

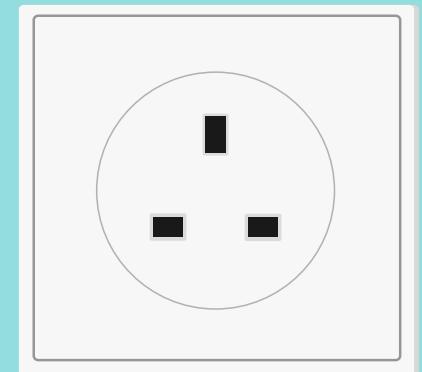
TAK





Gniazdka

- Metoda komunikacji działająca przy użyciu punktów końcowych (endpoint).
- Gniazdka domeny UNIX:
 - ▶ Domena AF UNIX lub AF LOCAL.
- Adresem jest ścieżka w systemie plików, W Linuksie połączenie i pisanie do gniazdka wymaga odpowiednich praw na pliku gniazdka lub katalogu w którym się znajduje.
- Gniazdka IPv4:
 - ▶ Domena AF_INET
 - ▶ Adres endpointu w postaci pary: adres IP i numer portu.
- Gniazda Netlink:
 - ▶ Domena AF_NETLINK.
 - ▶ Adres endpointu w postaci PID (w tym 0).



SPRAWDŹ SIĘ!

Gniazdka domeny AF_INET służą do komunikacji z procesem (A) tylko lokalnym, (B) lokalnym, lub zdalnym.

A

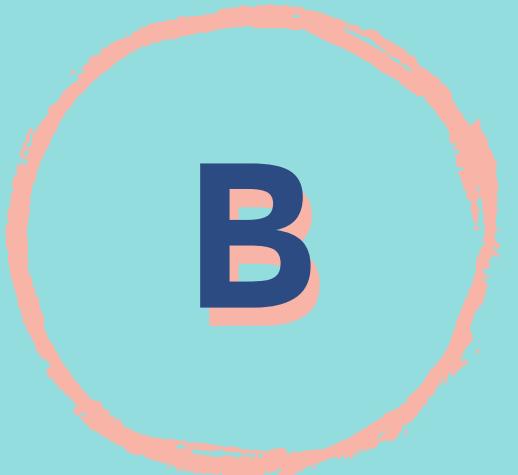
B



SPRAWDŹ SIĘ!

Gniazdka domeny AF_INET służą do komunikacji z procesem (A) tylko lokalnym, (B) lokalnym, lub zdalnym.

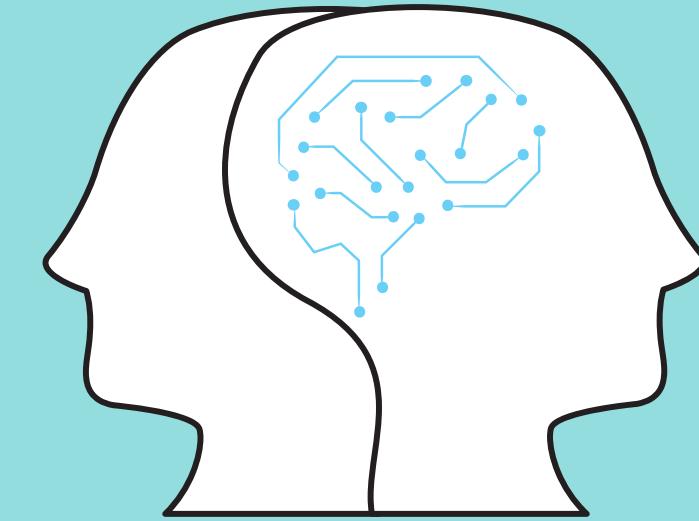
A



Pamięć współdzielona

Pamięć (współ)dzielona w wersji POSIX-owej.

- Brak konieczności wykorzystania wywołań systemowych (poza inicjalizacją)
- Brak interakcji z systemem plików.
- Teoretycznie bardzo wydajny sposób komunikacji.
- Konieczność własnoręcznego zapewnienia synchronizacji (np. semafory)!
- Możliwość w zasadzie dowolnej organizacji przydzielonej pamięci.



SPRAWDŹ SIĘ!

W porównaniu z innymi sposobami, komunikacja z użyciem pamięci współdzielonej jest: (A) szybka, (B) wolna.

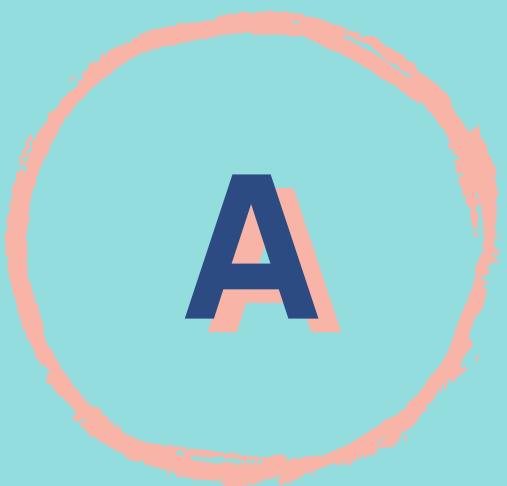
A

B



SPRAWDŹ SIĘ!

W porównaniu z innymi sposobami, komunikacja z użyciem pamięci współdzielonej jest: (A) szybka, (B) wolna.

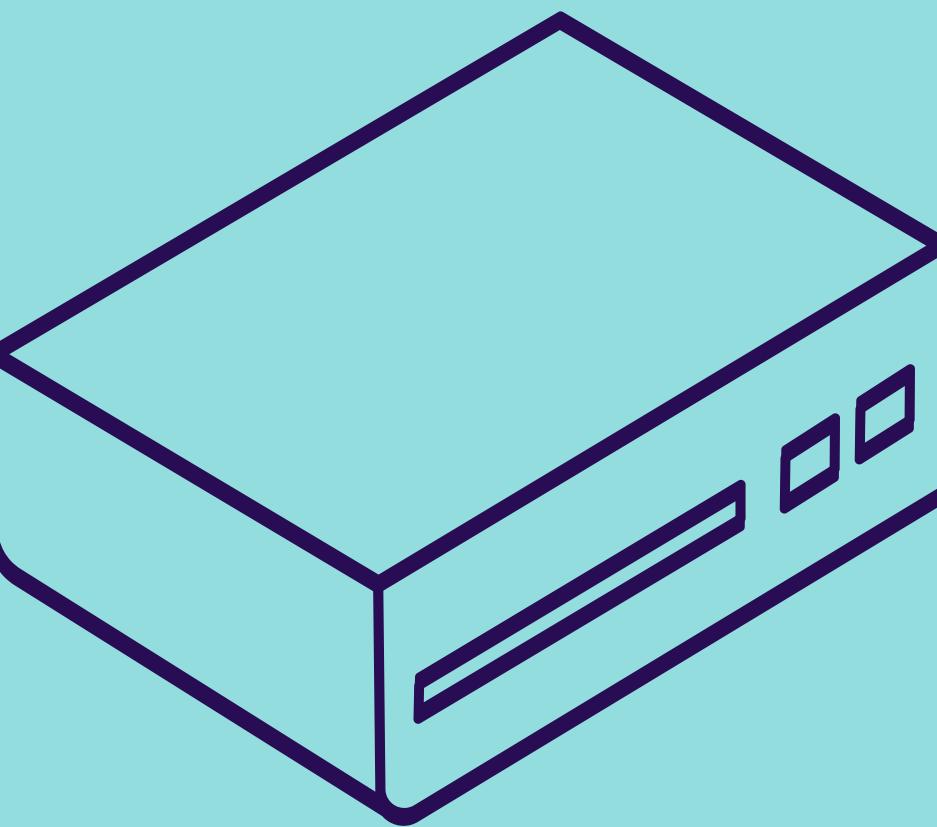


B



Wykład 12

System plików, urządzenia



Sterowniki

- Część jądra systemu operacyjnego odpowiedzialna za **obsługę urządzenia wejścia-wyjścia** (ogólnie urządzenia peryferyjnego).
- Dostarcza **abstrakcyjny interfejs** do obsługi urządzenia, bez konieczności znajomości jego szczegółów. Czasami jeden sterownik obsługuje wiele urządzeń.
- **Sterowniki dołączane są do jądra statycznie (w czasie komplikacji) lub dynamicznie (poprzez moduły).**



PODZIAŁ:

- **urządzenia znakowe**
(character devices)
- **urządzenia blokowe**
(block devices)
- **interfejsy sieciowe**
(network interfaces)

Sterowniki znakowe i blokowe

- Widoczne jako pliki specjalne w `/dev/` z własną ścieżką (np. `/dev/tty0`, `/dev/null`).
- Typ pliku (widoczny np. po użyciu `ls -l`) oznaczany jako **c** (urządzenia znakowe) lub **b** (urządzenia blokowe).
- Identyfikowane z wykorzystaniem liczby głównej i pobocznej:
 - * liczba główna (major) identyfikuje sterownik,
 - * liczba poboczna (minor) jest przekazywana do sterownika, identyfikuje konkretne urządzenie lub tryb pracy,
 - * widoczne poprzez `ls -l` (zamiast rozmiaru pliku).
- Czasami jedno urządzenie dostępne jest zarówno przez urządzenie znakowe jak i blokowe.



Urządzenia znakowe

- 1 Dane dostępne są **sekwencyjnie** (strumieniowo), znak (najczęściej bajt) za znakiem.
- 2 Wskaźnik plikowy ma tylko jedną pozycję: **aktualną**. Ręczna zmiana położenia wskaźnika plikowego (seeking) jest **zabroniona**.
- 3 Buforowanie **nie** jest wymagane (ale jest możliwe).
- 4 Sterowniki urządzeń znakowych są **prostsze, łatwiejsze** w przygotowaniu i wymagają **mniej uwagi**.
- 5 Zapis i odczyt są **blokujące** (wywołania read i write wracają gdy operacja się zakończy).
- 6 **Przykłady:** porty szeregowe, porty równoległe, karty dźwiękowe, klawiatura, terminale.

Urządzenia blokowe

- 1 Zawsze mają dostęp **swobodny** (random-access).
- 2 Zwykle dotyczą fizycznych urządzeń, które zapisują i odczytują bloki o określonej długości.
- 3 Dane są **buforowane** (obecność cache'a).
- 4 Wsparcie dla zmiany pozycji wskaźnika plikowego (seeking).
- 5 Sterowniki blokowe są **trudniejsze w tworzeniu i obsłudze**.
- 6 Żądania zapisu i odczytu wywoływane są przez system buforowania. **Mogą być asynchroniczne**.
- 7 Przykłady: dyski twardy, pendrive'y, CD-ROM, DVD, kamery USB.

System Plików



Czym jest **system plików**?

System plików to struktura danych i operacje na niej, stosowane przez system operacyjny do zarządzania, zapisywania i odczytywania danych.

System plików

- Dotyczy głównie **pamięci masowej**, ale nie tylko.
- Czasami system plików traktowany jest jako część systemu operacyjnego.
- System plików umożliwia **niezawodny i wydajny** dostęp do danych **niezależnie** od urządzenia końcowego i jego specyfikacji.

Wirtualny system plików (VFS)

- **ujednolicony interfejs** dostępu do różnych konkretnych systemów plików, korzystają z niego funkcje systemowe,
- mapuje wirtualne funkcje na operacje specyficzne dla danego FS (łatwe dodawanie nowych FS przez implementację open, read, write itd.),
- **część FS** (np. /tmp) **omija VFS**.

urządzenia blokowe:

- obsługują zapis i odczyt urządzeń fizycznych, pojedyncze odpowiada za jedną (wirtualną) partycję

urządzenia znakowe:

- występują opcjonalnie, wykorzystywane do obsługi urządzeń (np. tworzenie partycji)

SPRAWDŹ SIĘ!

W Linuksie katalogi /tmp oraz /var/tmp służą do tego samego (stosowny jeden lub drugi, zależnie od dystrybucji).

TAK

NIE



SPRAWDŹ SIĘ!

W Linuksie katalogi /tmp oraz /var/tmp służą do tego samego (stosowny jeden lub drugi, zależnie od dystrybucji).

TAK



Najważniejsze elementy uniksowego systemu plików

Superblok

przechowuje ważne informacje ogólne (globalne metadane) o systemie plików

Tablica i-węzłów

zawiera i-węzły (i-nodes), które przechowują metadane poszczególnych plików; zajmuje około 20-30% miejsca.

Bloki danych

przechowują zawartość samych plików (zależnie od typu pliku, dla plików zwykłych są to dane użytkownika); zajmują większość miejsca.

Superblok

- Przechowuje między innymi:
 - * **rozmiar FS, rozmiar bloku itd.,**
 - * **status FS,**
 - * **parametry techniczne,**
 - * **umiejscowienie** innych elementów (m.in. tablicy i-węzłów).
- Błędy w superbloku mogą **uniemożliwić zamontowanie** (a tym samym odczyt i zapis danych) systemu plików.
- FS przechowuje **kopie superbloku.**

I-węzeł



Przechowuje metadane pliku:

- typ pliku,
- właściciel, właściciel grupowy,
- prawa dostępu,
- rozmiar pliku (dla ext4 limit 16 TiB),
- liczba (twardych) dowiązań,
- czas ostatniego dostępu, modyfikacji i zmiany stanu i-węzła,
- położenie na dysku (tablica indeksowa na bloki danych, wskaźniki bezpośrednie i pośrednie).

I-węzeł



Dla rozmiaru bloku 4 kB i rozmiaru
wskaźnika 4 bajty (16 TB miejsca):

- bezpośrednie wskazanie: około 50 kB,
- pośrednie 1-poziomu: około 4 MB,
- pośrednie 2-poziomu: około 4 GB,
- pośrednie 3-poziomu: około 4 TB.

SPRAWDŹ SIĘ!

W uniksowym systemie plików za pomocą odwołań pośrednich 1-go poziomu i bezpośrednich da się zapisać plik do około: (A) 100 KiB, (B) 4 MiB.

A

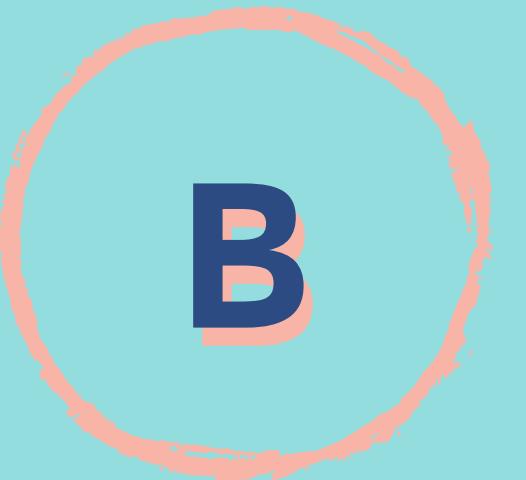
B



SPRAWDŹ SIĘ!

W uniksowym systemie plików za pomocą odwołań pośrednich 1-go poziomu i bezpośrednich da się zapisać plik do około: (A) 100 KiB, (B) 4 MiB.

A



Typy plików:

pliki zwykłe

katalogi

dowiązania symboliczne

urządzenia blokowe i znakowe

gniazdka UNIX-owe

Katalogi

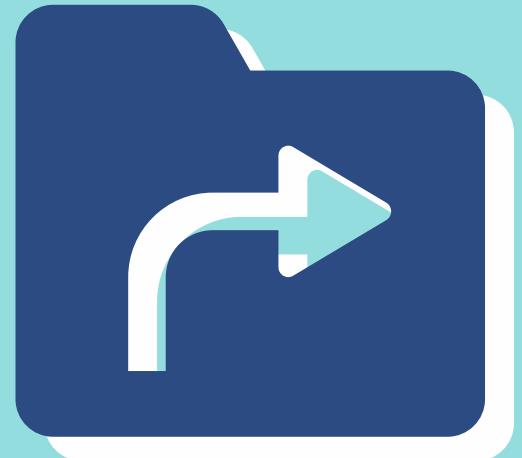
- zawierają wpisy katalogowe w postaci mapowania **nazwa pliku → i-węzeł**
- wpisy stanowią **dowiązania** (tzw. dowiązania twardé)
- wpisy . oraz .. – „specjalne” dowiązania twardé tworzony automatycznie
- na ten sam i-węzeł (i tym samym plik fizyczny) może istnieć **wiele dowiązań twardych** (niekoniecznie w jednym katalogu); system może usunąć plik, dopiero gdy usuniemy wszystkie dowiązania (poza . i ..)
- katalog główny (korzeń) FS to katalog wskazany w superbloku; jego katalogiem nadziedzonym jest on sam

Dowiązania symboliczne (tzw. miękkie, symlinki)

Przechowuję one ścieżkę do innego pliku tzw. celu.

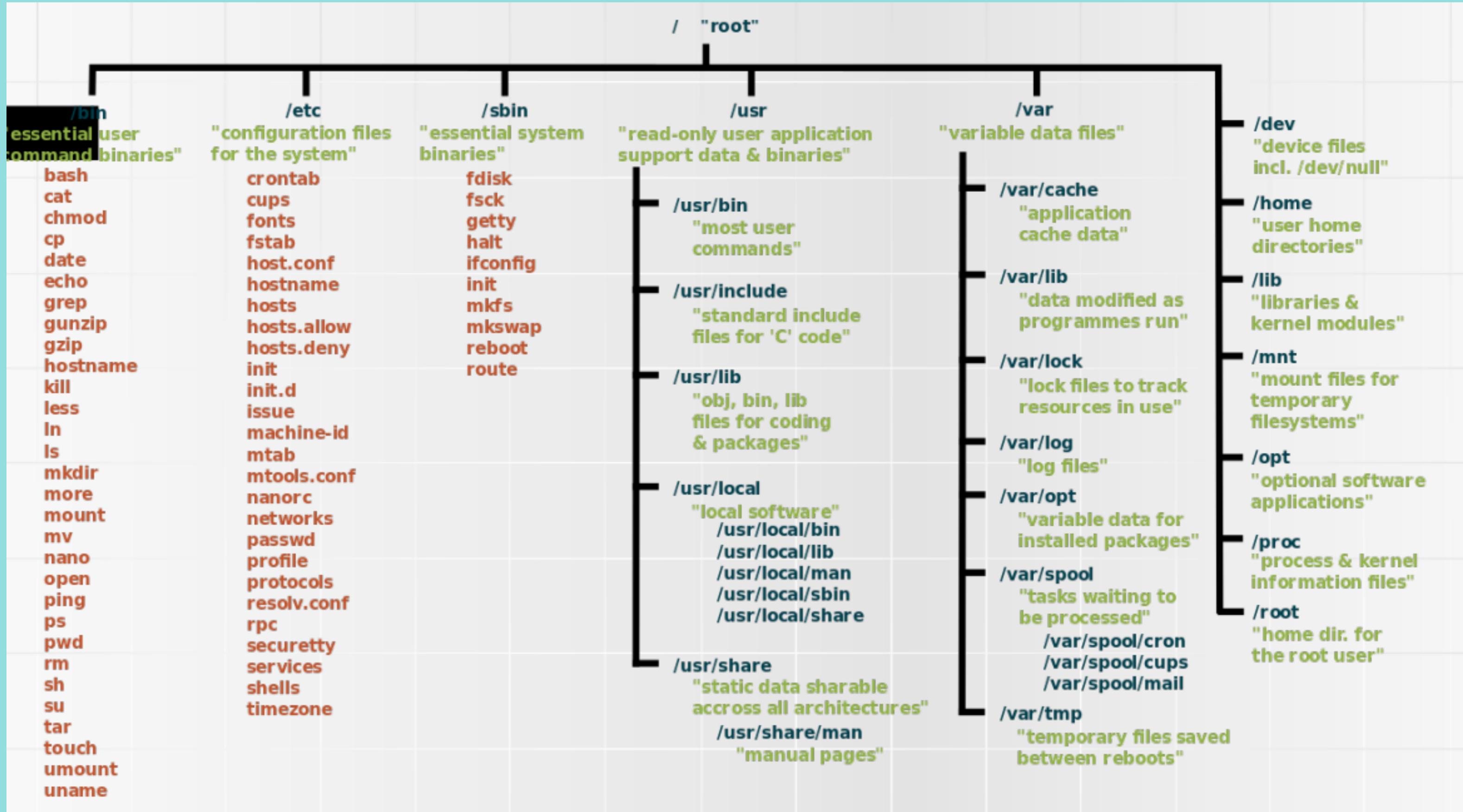
Cel nie musi być na tym samym urządzeniu, ani w ogóle istnieć.

Operacje na symlinku przenoszone są na cel.



Dowiązania twarde

Tworzą nową nazwę dla zasobu, zapisując ją w nowej lokalizacji (nie kasując poprzedniej), a sam link nie odwołuje się do pliku samego w sobie, ale tylko do jego zawartości



NFS - Network File System

- Protokół dla rozproszonych systemów plików.
- Pozwala na dostęp do zdalnych plików tak jakby były to zwykłe pliki lokalne
- Przykładowe opcje udziałów NFS:
 1. **rw** (read-write), **ro** (read-only, domyślna)
 2. **root squash** (domyślna), **no root squash**, **all squash** – zmiana żądającego UID/GID na anonimowy
 3. **anonuid**, **anongid** – UID i GID dla użytkownika anonimowego.
 4. **sync**, **async** – odpowiedź po/przed zaksięgowaniem operacji.
„Brak” domyślnej wartości.
 5. **nohide**, **hide** (domyślna), **crossmnt** – traktowanie „podudziałów”.
 6. **no subtree check**, **subtree check** – sprawdzanie poprawności udziałów. „Brak” domyślnej wartości.

SPRAWDŹ SIĘ!

Ścieżka zaczynająca się od znaku / to ścieżka: (A) bezwzględna,
(B) kanoniczna.

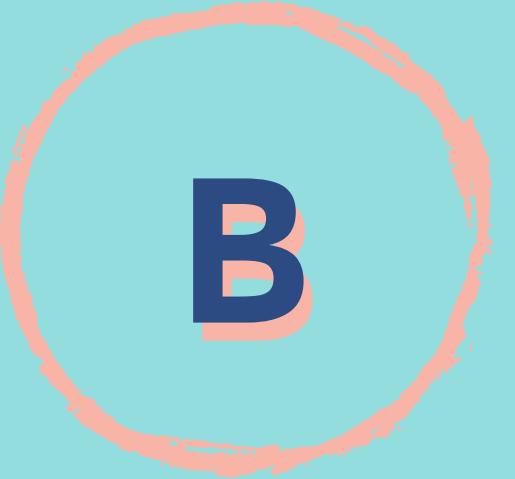
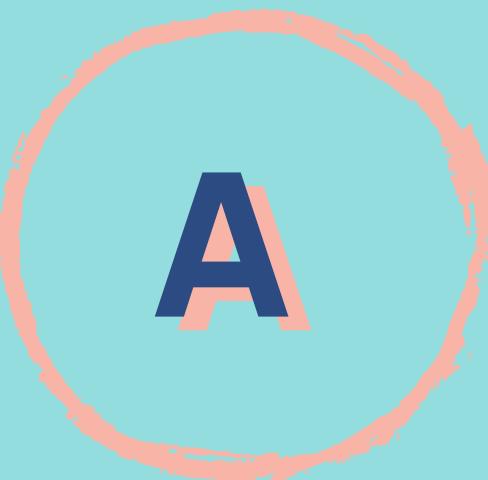
A

B



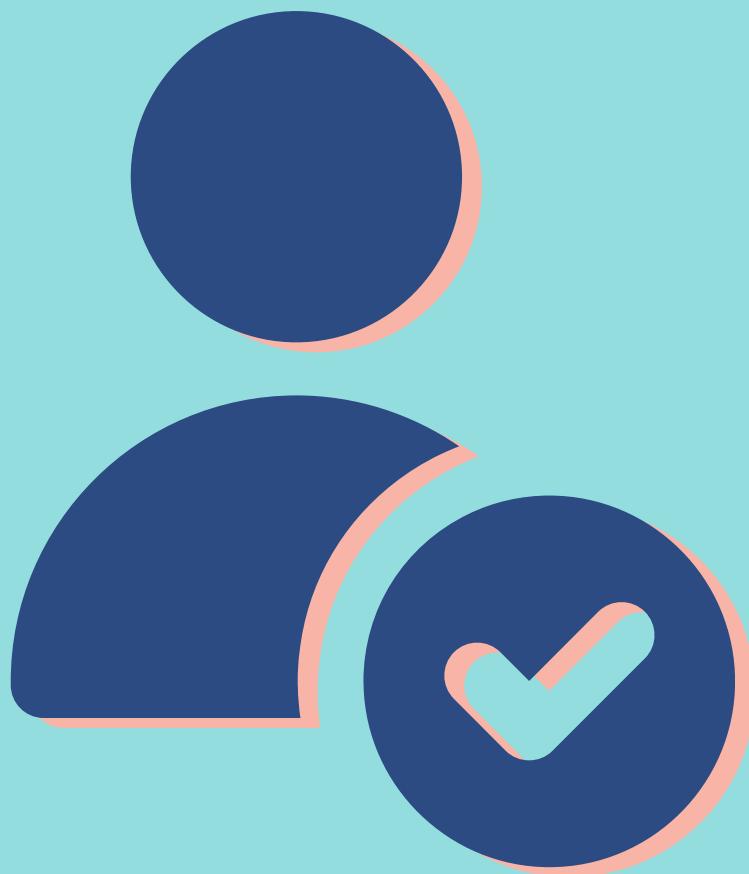
SPRAWDŹ SIĘ!

Ścieżka zaczynająca się od znaku / to ścieżka: (A) bezwzględna,
(B) kanoniczna.



Wykład 13

Użytkownicy i autoryzacja



AAA – Authentication, Authorization, Audit.

- 1. Authentication (uwierzytelnianie)** – weryfikacja wiarygodności (potwierdzenie) tożsamości.
- 2. Authorization (autoryzacja)** – nadawanie uprawnień (pozwolenie lub zakaz) do danej czynności, zwykle dostępu do zasobu.
- 3. Audit (audyt)** – kontrola działań tożsamości, najczęściej poprzez rejestrowanie aktywności (logi), rzadziej kontrole finansowe (accounting).

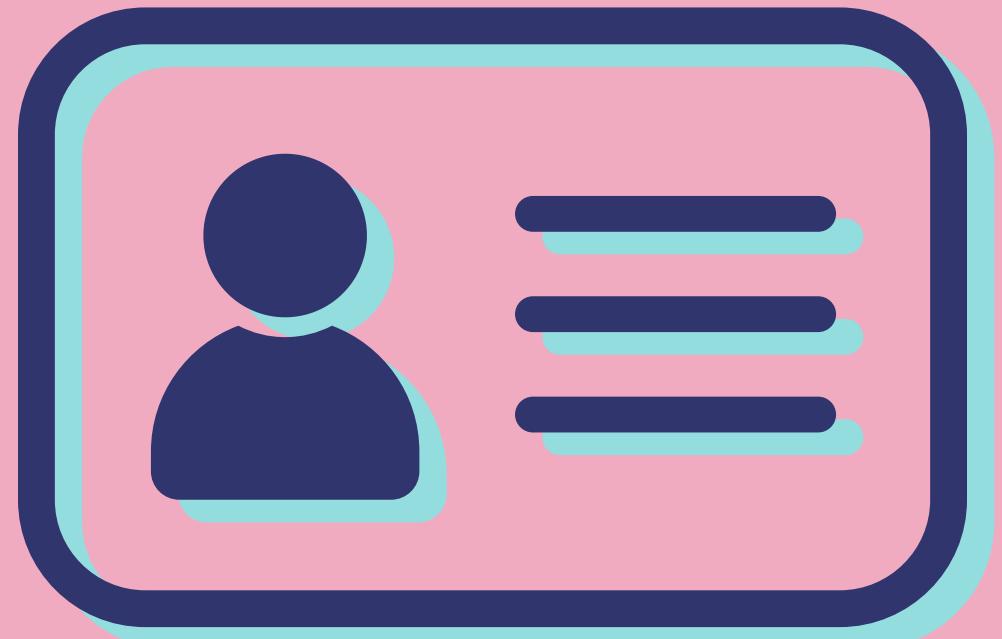
- Identification (identyfikacja) – określenie (zadeklarowanie) tożsamości.
- Określenie czynności – zwykle oczywiste.

Identyfikacja i uwierzytelnianie wykonywane są zwykle jednorazowo lub pozostają ważne przez pewien czas trwania sesji.



Identyfikacja tożsamości

- W jej celu wykorzystuje się numery **UID** (User ID) oraz **GID** (Group ID).
- Określają **właściciela i prawa dostępu do większości obiektów**.
- Użytkownik może należeć do **wielu grup**, z czego jedna jest grupą podstawową



Typowe zakresy UID:

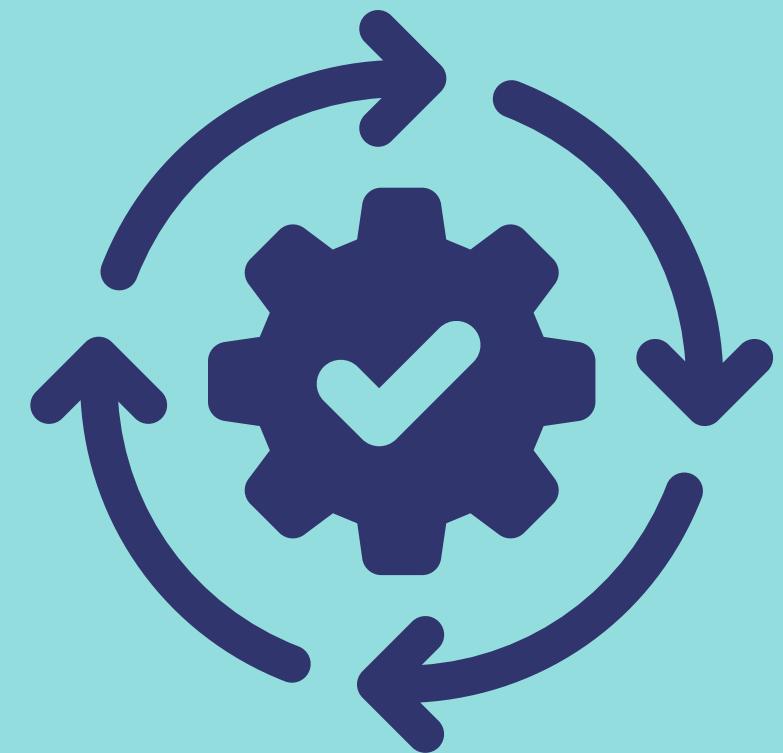
- **0**: superuser (root). Ma prawa do dowolnej czynności.
 - **1–99**: zarezerwowane dla aplikacji systemu (przydział statyczny). Duże ryzyko konfliktu.
 - **100–499**: zarezerwowane dla aplikacji systemu (przydział dynamiczny). Ryzyko konfliktu.
 - **500–999**: generalnie powinny być dostępne. Niskie ryzyko konfliktu.
 - **od 1000**: „zwykli” użytkownicy.
- .
- System generalnie przydziela użytkownikom pierwszy wolny UID wyższy od obecnie zajętych

EUID, EGIG (effective)

To identyfikatory używane do wielu czynności wymagających autoryzacji.

Normalnie są równe RUID/RGID, chyba że:

- „zmieniono” użytkownika (sudo)
- plik który wykonaliśmy miał ustawiony bit setuid



RUID, RGID (real)

To „prawdziwe” identyfikatory właściciela procesu.

Pochodzą od **UID/GID użytkownika**, który uruchomił proces, określają właściciela procesu.



Zmiana UID/GID jest możliwa m.in. z pomocą:

- funkcje **setuid**, **seteuid**, **setgid**,
- komenda **su** – zmiana użytkownika (prostsza, korzysta z wartości w pliku /etc/login.defs),
- komenda **sudo** – wykonanie komendy jako inny użytkownik. Rozbudowana, może służyć do ustalenia złożonych obostrzeń. Podstawowa konfiguracja w /etc/sudoers, ale są inne możliwości (np. LDAP).

SPRAWDŹ SIĘ!

Prawa do wykonywania czynności zasadniczo określone są za pomocą (A)EUID (B)RUID.

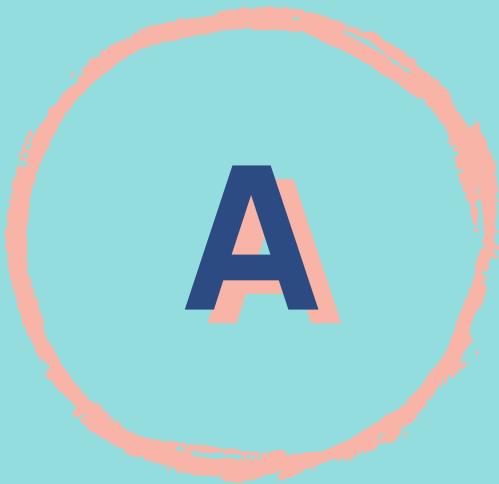
A

B



SPRAWDŹ SIĘ!

Prawa do wykonywania czynności zasadniczo określone są za pomocą (A)EUID (B)RUID.



B



Podstawowe uwierzytelnianie

- Program login.
- Podstawowa baza danych kont użytkowników przechowywana jest w pliku **/etc/passwd**.
- Dane o grupach przechowywane w pliku **/etc/group**
- Podstawowe uwierzytelnianie: porównanie **hasha** podanego **hasła z hashem** w pliku **/etc/passwd**.
- Hashe uzyskiwane są funkcją **crypt**, która udostępnia wiele trybów hashowania, oryginalny bazowany był na szyfrze DES.



- **/etc/passwd** i **/etc/group** muszą być czytelne dla wszystkich, z ich danych (mapowanie nazwy użytkownika na UID, katalog domowy, powłoka) korzysta wiele komend np. ls.
- Obecnie hashe haseł zostały przeniesione do osobnych plików **/etc/shadow** oraz **/etc/gshadow**, z odpowiednią („x”) wzmianką w **/etc/passwd**. Odczyt tych plików wymaga praw roota.
- Edycja tych plików może być ręczna (choć lepiej zostawić to komendom).
- Poprawność (spójność) można sprawdzić poprzez komendy **pwck** oraz **grpck**.

SPRAWDŹ SIĘ!

We współczesnym linuksie
zahasowane hasło użytkownika
przechowywane jest w pliku
`/etc/passwd`.

TAK

NIE



SPRAWDŹ SIĘ!

We współczesnym linuksie
zahasowane hasło użytkownika
przechowywane jest w pliku
`/etc/passwd`.

TAK



Prawa plikowe

W systemach Unix i podobnych do Unix (takich jak Linux), prawa dostępu do plików określają, **kto i w jaki sposób może korzystać z pliku lub katalogu.**

Prawa te są określone przez **zestaw bitów**, które określają uprawnienia dla trzech różnych grup użytkowników:

1. **Właściciel** (ang. **owner**) - osoba, która utworzyła plik lub katalog.
2. **Grupa** (ang. **group**) - zbiór użytkowników, którym przypisano określone uprawnienia do pliku.
3. **Inni** (ang. **others**) - wszyscy pozostali użytkownicy systemu.



Każda z tych grup może mieć nadane trzy rodzaje uprawnień:

- **Czytanie (r - read)** - pozwala na odczytywanie plików lub wyświetlanie zawartości katalogów.
- **Zapis (w - write)** - pozwala na modyfikowanie plików lub zmianę zawartości katalogów.
- **Wykonywanie (x - execute)** - pozwala na uruchamianie plików (jeśli są to pliki wykonywalne lub skrypty) lub przeszukiwanie katalogów.



Przykład



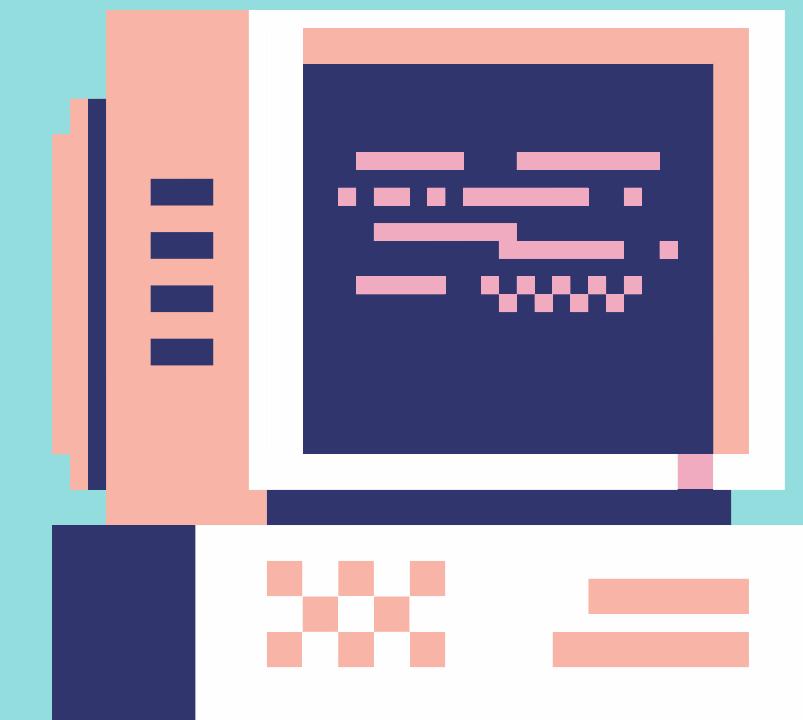
-rwxr-xr-- 1 właściciel grupa rozmiar data nazwa_pliku

Pierwsza część (**-rwxr-xr--**) reprezentuje uprawnienia, i jest podzielona na 10 bitów:

1. Pierwszy bit określa typ elementu (na przykład **-** dla pliku, **d** dla katalogu).
2. Następne trzy bity (**rwx**) określają uprawnienia właściciela.
3. Kolejne trzy bity (**r-x**) określają uprawnienia grupy.
4. Ostatnie trzy bity (**r--**) określają uprawnienia dla innych.

Dodatkowo, istnieją specjalne bity, które mogą być ustawione dla plików i katalogów:

- **Set User ID (SUID)**: Jeśli ten bit jest ustawiony na pliku wykonywalnym, proces uruchamiający ten plik otrzyma uprawnienia właściciela pliku podczas jego wykonania.
- **Set Group ID (SGID)**: Podobnie jak SUID, ale proces otrzymuje uprawnienia grupy pliku.
- **Sticky bit**: Głównie używany na katalogach, zapewnia, że tylko właściciel pliku może usunąć plik w katalogu z ustawionym sticky bit.



Te specjalne bity również mają swoje reprezentacje w notacji uprawnień.

W systemach Unix, te uprawnienia są często reprezentowane także jako trzy cyfry oktalne (od 0 do 7), gdzie każda cyfra jest sumą:

- 4, jeśli ustawione jest uprawnienie do **czytania** (r),
- 2, jeśli ustawione jest uprawnienie do **zapisu** (w),
- 1, jeśli ustawione jest uprawnienie do **wykonywania** (x).

Na przykład, uprawnienia **rwxr-xr--** można przedstawić jako **754** w systemie ósemkowym.

SPRAWDŹ SIĘ!

Jeśli użytkownik “user” wykona plik, którego właścicielem jest “owner” i plik ten ma ustawiony bit set-user-ID, to powstały proces będzie mieć EUID odziedziczony po użytkowniku (A) user (B) owner.

A

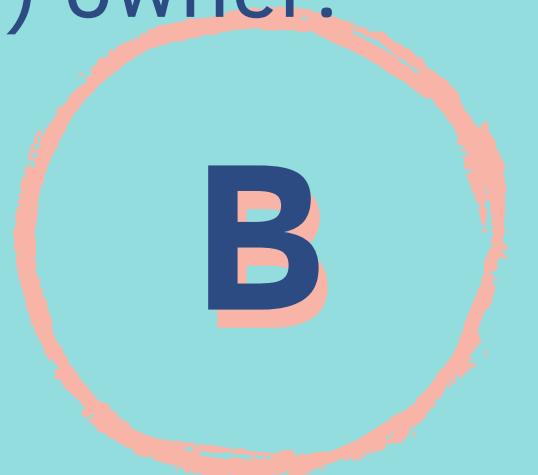
B



SPRAWDŹ SIĘ!

Jeśli użytkownik “user” wykona plik, którego właścicielem jest “owner” i plik ten ma ustawiony bit set-user-ID, to powstały proces będzie mieć EUID odziedziczony po użytkowniku (A) user (B) owner.

A



Access Control Lists



- **POSIX-owe Access Control Lists (ACL)** to nadbudowa na zwykłe prawa plikowe, umożliwiająca dokładniejszą kontrolę uprawnień dla użytkowników.
- Wymaga **wsparcia jądra** (obecnie domyślnie w praktycznie wszystkich dystrybucjach) i zamontowania systemu pliku z opcją acl.
- **Każdy** plik może mieć zwykłe ACL (access ACL).
- Katalogi mogą mieć domyślne ACL (default ACL) – dziedziczone przez tworzone w nich pliki (dla których stają się access ACL).

Limitowanie zasobów



Komenda ulimit



Pakiet quota

Komenda ulimit – raportowanie i ustawianie limitów zasobów wykorzystywanych przez procesy:

- **limit dla dostępnej pamięci** (w tym segmentu danych procesu), **priorytetu procesów** (wartość nice), **liczby otwartych deskryptorów plików**, **liczby wątków**, **liczby procesów**, **rozmiaru potoków/kolejek komunikatów**, **rozmiaru stosu**, **czasu procesora itp.**,
- różne wielkości mają różne jednostki,
- specjalne wartości **soft**, **hard** oraz **unlimited**,
- zwykły użytkownik **nie może** zwiększyć limitu miękkiego poza limit twardy, zwykły użytkownik nie może zwiększyć limitu twardego (ale może go zmniejszyć),
- root **nie ma ograniczeń**.

Pakiet quota – limitowanie przestrzeni dyskowej:



- opcje **usrquota** oraz **grpquota** w pliku **/etc/fstab**,
- komenda **quotacheck** – skanowanie dysków, tworzenie (opcja -c i naprawa plików quota),
- komendy **quotaon** oraz **quotaoff**,
- komenda **edquota** – edycja limitów przestrzeni dyskowej (w blokach) i dozwolonej liczby plików użytkownika lub grupy; uruchamia osobny edytor
- **limit miękki** może być przekroczony do pewnego czasu (grace period),
- **limit twardy** nieprzekraczalny.
- komenda **repquota** – raportowanie ustawionych limitów.

Zarządzanie użytkownikami

- **useradd** - tworzenie użytkownika
- **passwd** – ustawienie lub zmiana hasła.
- **userdel** – usuwanie użytkowników. Można wymusić usunięcie zalogowanego użytkownika oraz skasować pliki w jego katalogu domowym.
- **usermod** – modyfikacja istniejącego użytkownika (opcje podobne do useradd).
- Możliwość przenoszenia katalogu domowego.



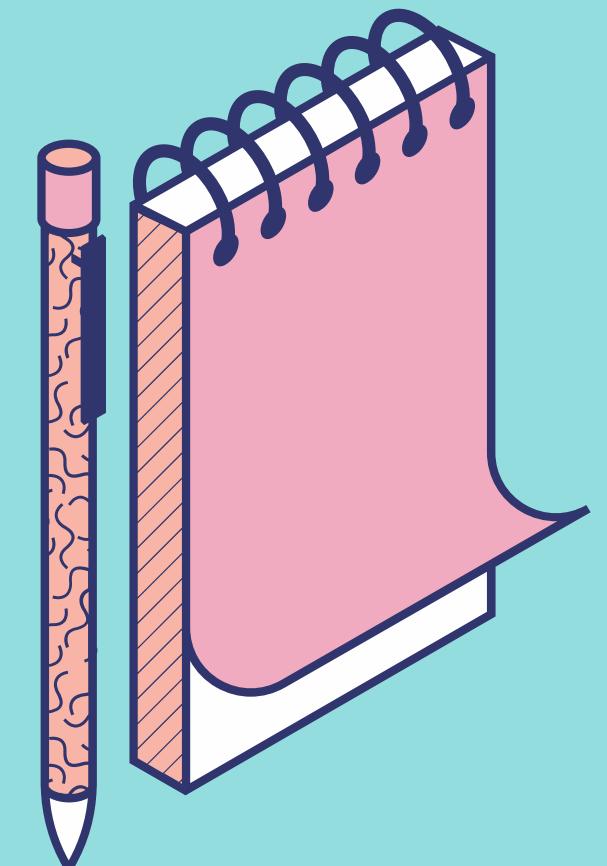
Uwierzytelnianie poprzez LDAP

Lightweight Directory Access Protocol (LDAP)

– otwarty standard dostępu do rozproszonych usług katalogowych.

Pluggable Authentication Module (PAM)

– wysokopoziomowe API dostępu do różnych niskopoziomowych metod zarządzania kontami, uwierzytelnianiem, hasłami i sesją.



SPRAWDŹ SIĘ!

Użytkownik zawsze może zmienić swoje obecne hasło przy użyciu komendy passwd.

TAK

NIE



SPRAWDŹ SIĘ!

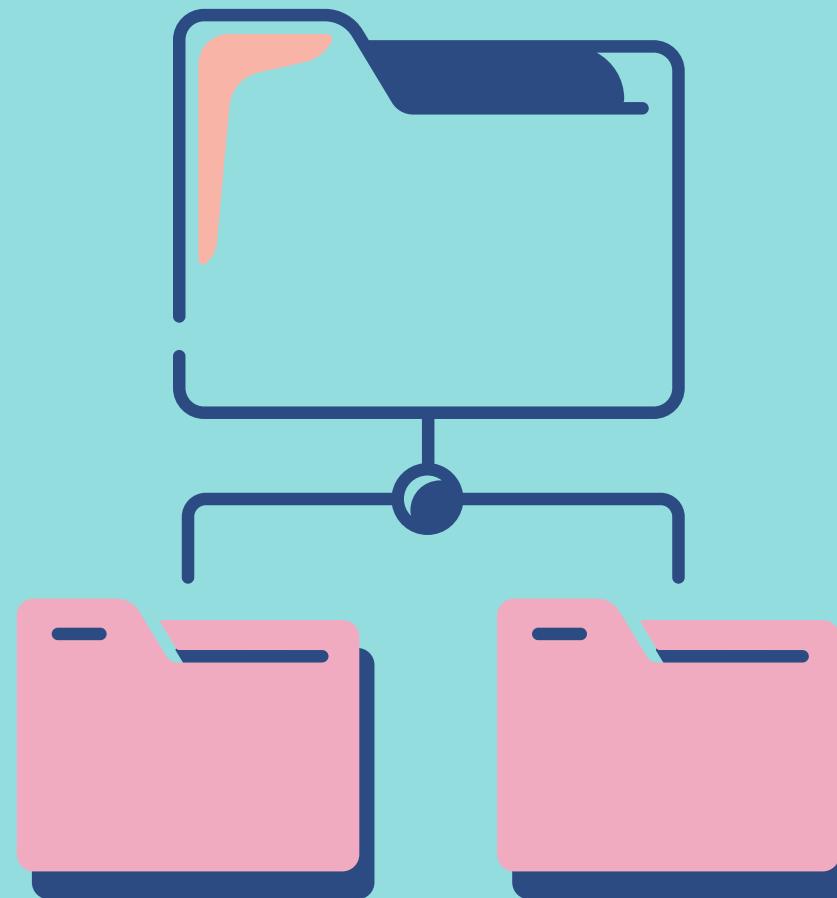
Użytkownik zawsze może zmienić swoje obecne hasło przy użyciu komendy passwd.

TAK



Wykład 14

Systemy plików



EXT 2

- Standardowe **trzy znaczniki czasu**, z rozdzielczością 1 sekundy
- **Brak księgowania (journaling)** powoduje, że czasami wciąż jest w niektórych przypadkach stosowany dla pendrive'ów, USB i dysków SSD
- Błędy operacji w przypadku utraty zasilania (brak księgowania).
- **Problem fragmentacji**

EXT 3

- Znaczna kompatybilność z ext2 (możliwość przekształcania w obie strony)
- Duża wydajność, nawet w porównaniu z ext4
- Duże katalogi przechowywane jako H-drzewa (podobne do B-drzew, niewymagające balansowania)
- Brak zaawansowanych funkcjonalności (np. dynamiczna alokacja i-węzłów, utrudnione odzyskiwanie usuniętych plików).
- Brak natywnego wsparcia dla defragmentacji oraz kompresji plików.



Tryby księgowania

Journal

dziennik zapisuje zarówno **metadane**, jak i **dane pliku** przed przesłaniem do zapisu (commit).

Zapewnia najwyższy poziom **niezawodności**.

Może powodować spadek wydajności (dwukrotny zapis).

Ordered

dziennik zapisuje tylko **metadane**, ale commit jest oznaczany dopiero po zapisie danych pliku.

Dla nowych i dopisywanych plików **bezpieczeństwo** jest takie jak tryb **journal**. Dla plików nadpisywanych może dojść do **korupcji** (brak wersji przed nadpisaniem). **Domyślny tryb dla wielu dystrybucji**.

Writeback

dziennik zapisuje tylko **metadane**, dane dysku commitowane przed lub po aktualizacji dziennika.

Najniższy poziom niezawodności.

EXT 4

- Możliwość wyłączenia dziennika - przewaga nad ext2
- Wsteczna kompatybilność z ext2 i ext3
- Mniejsza fragmentacja
- Wsparcie dla szyfrowania i defragmentacji
- Znaczniki czasu z dokładnością do nanosekundy, znacznik czasu utworzenia
- Nielimitowana liczba podkatalogów (H-drzewa)
- Suma kontrolna w dzienniku

SPRAWDŹ SIĘ!

Ext2 jest systemem plików z księgowaniem.

TAK

NIE



SPRAWDŹ SIĘ!

Ext2 jest systemem plików z księgowaniem.

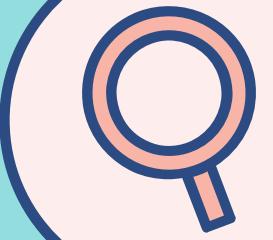
TAK





Co to jest H-Drzewo?

Hash-tree to rodzaj struktury danych, która opiera się na zastosowaniu funkcji haszującej do efektywnego utrzymania równowagi i zapewnienia integralności danych.



H-Drzewo

Równowaga

H-drzewo jest zaprojektowane tak, aby **utrzymywać równowagę** między lewym a prawym poddrzewem dla każdego węzła. Równowaga ta przekłada się na **efektywne operacje** na drzewie, które mają złożoność czasową logarytmiczną względem liczby węzłów.

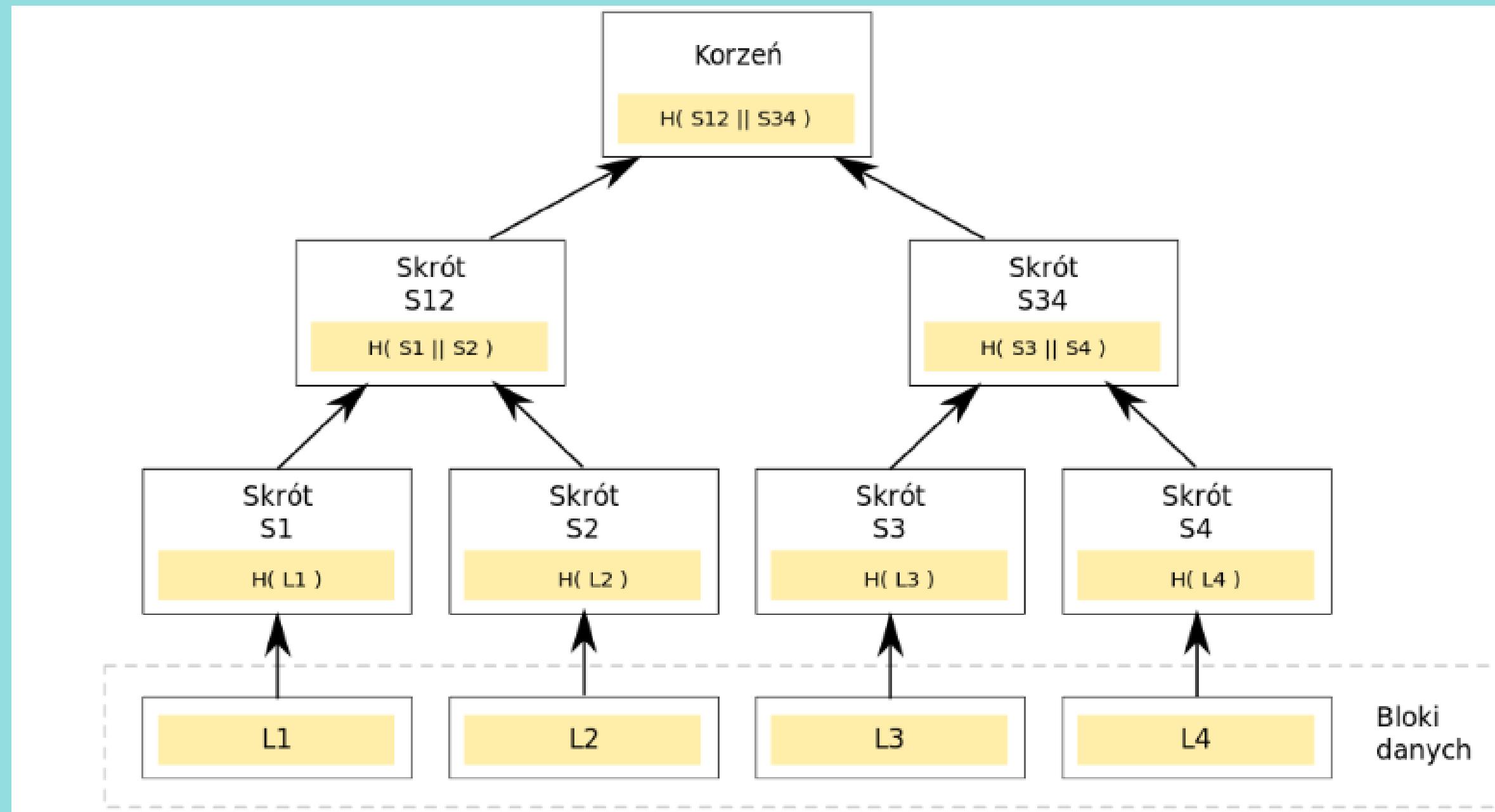
Haszowanie

Każdy węzeł H-drzewa jest haszem swoich dzieci, a korzeń zawiera hasz całego drzewa. Zastosowanie funkcji haszującej umożliwia szybkie sprawdzenie integralności danych poprzez porównanie korzenia (korzenia haszowego) z znany wcześniej haszem.

Zastosowania

H-drzewa są powszechnie stosowane w bazach danych do indeksowania danych przestrzennych, takich jak obszary geograficzne czy przedziały czasowe. Wykorzystywane są również w kryptografii, zwłaszcza w drzewach Merkle'a, do potwierdzania integralności bloków danych, na przykład w kryptowalutach.

Przykładowe H-Drzewo



FAT

- Ze względu na kompatybilność i prostotę **wciąż jest używany** m.in.
w kartach pamięci, pendrive'ach/USB oraz urządzeniach wbudowanych.
- Wspierany przez **wiele urządzeń**.
- Przydatny w środowisku wykorzystującym różne urządzenia i systemy operacyjne.
- Wiele wariantów i rozszerzeń.
- FAT dzieli przestrzeń na sektory, typowo po 4 kiB

FAT zbudowany jest z 4 regionów



Typy wpisów

- Numer kolejnego klastra pliku
- Koniec pliku (ostatni klaster)
- Klaster nieużywany
- Błędny klaster
- Wartości specjalne

Tablica FAT jest tablicą wpisów o klastrach

Atrybuty plikowe

- **Volume** – dodatkowa etykieta woluminu.
- **Directory** – plik jest katalogiem.
- **Archive** – czy plik zsynchronizowany z dyskiem.
- **Read-only**
- **Hidden** – domyślnie nielistowany.
- **System** – nie należy przenosić podczas defragmentacji.

Example of FAT16 table start with several cluster chains																
Offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
+0000	F0	FF	FF	FF	03	00	04	00	05	00	06	00	07	00	08	00
+0010	FF	FF	0A	00	14	00	0C	00	0D	00	0E	00	0F	00	10	00
+0020	11	00	FF	FF	00	00	FF	FF	15	00	16	00	19	00	F7	FF
+0030	F7	FF	1A	00	FF	FF	00	00	00	F7	FF	00	00	00	00	00

SPRAWDŹ SIĘ!

Kolejne klasy pliku w systemie plików FAT tworzą: (A) tablice,
(B) listę wiązaną.

A

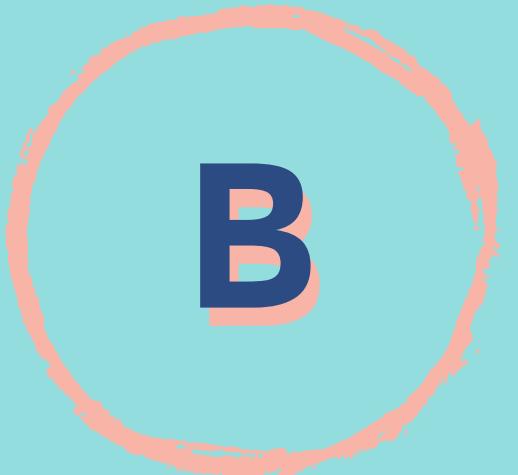
B



SPRAWDŹ SIĘ!

Kolejne klasy pliku w systemie plików FAT tworzą: (A) tablice,
(B) listę wiązaną.

A





NTFS

Struktura

- Partition Boot Sector (**PBS**)
- Master File Table (**MTF**) - tablica metadanych plików (analogiczna do tablicy i-węzłów), plus kopia zapasowa w środku systemu

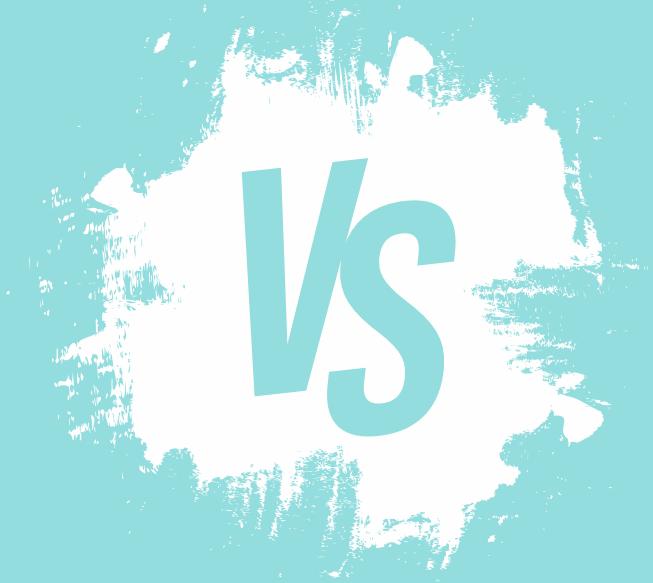
Przestrzeń plików

- **Pliki zwykłe**
- **Katalogi** - mapowana nazwa, ID w formie B-drzewa
- **Metapliki** - “specjalne” pliki, potrzebne od pracy systemy plików

Cechy NTFS

- Księgowanie (journaling)
- Dowiązania twarde, do 1024 na plik
- Opcja kompresji plików i całych katalogów za pomocą algorytmu LZNT1
- Rzadkie pliki (sparse files)
- Shadow copy
- Transakcje plikowe
- Limity dyskowe (quota)
- Szyfrowanie plików i całych katalogów
- Access Control Lists: prawa plikowe plus audyt/logowanie działań

NTFS



VS

FAT



Rozmiar dysku i ilość plików

FAT32 jest bardziej odpowiedni dla mniejszych dysków (karty pamięci, pendrive, USB, urządzenia wbudowane) i zawiera ograniczenia dotyczące maksymalnego rozmiaru partycji i pojedynczego pliku.

NTFS jest bardziej efektywne dla większych dysków, obsługując większe partycje i umożliwiając obsługę większych plików. Niewydajny dla małych partycji (<400MB)



Bezpieczeństwo danych

FAT32 nie oferuje tak rozbudowanych funkcji zabezpieczeń, co może być istotne w przypadku ważnych danych.

NTFS oferuje lepsze funkcje zabezpieczeń, takie jak uprawnienia plików, szyfrowanie plików i foldery, a także funkcje naprawy systemu plików.



Kompatybilność

FAT32 jest bardziej uniwersalne i kompatybilne z różnymi systemami operacyjnymi, co sprawia, że jest bardziej praktyczne, jeśli wymagana jest interoperacyjność pomiędzy różnymi platformami.

NTFS jest bardziej specyficzne dla systemu Windows, a inne systemy operacyjne mogą mieć ograniczoną lub żadną obsługę zapisu na partycjach NTFS.

SPRAWDŹ SIĘ!

Prostota i duża kompatybilność z wieloma urządzeniami to cechy systemu plików: (A)FAT, (B) NTFS.

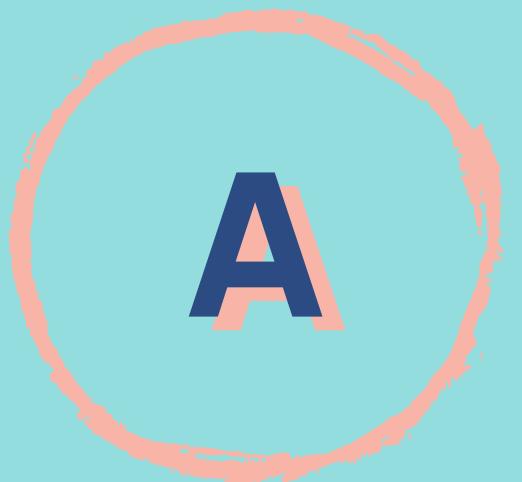
A

B



SPRAWDŹ SIĘ!

Prostota i duża kompatybilność z wieloma urządzeniami to cechy systemu plików: (A)FAT, (B) NTFS.



**“Zmęczył mnie ten przedmiot,
aczkolwiek mam nadzieję, że
wszyscy zdacie, trzymam za
was kciuki z całego serca!”**

ADRIAN GORAL

