Stronicowanie i pamięć wirtualna

Wykład #9

- Stronicowanie hierarchiczne
- Odwrócona tablica stron
- Segmentacja
- Segmentacja ze stronicowaniem
- Pamięć wirtualna

wersja:20.01.2022

Stronicowanie hierarchiczne

współczesne systemy komputerowe stosują bardzo duże przestrzenie adresów logicznych:

- dla 32-bitowych adresów i stron o rozmiarze 4 KB → ponad milion stron → każda pozycja w tablicy stron ma 4 B → potrzeba 4 MB fizycznej przestrzeni adresowej na samą tablicę stron dla każdego procesu!
- możliwe rozwiązanie → stronicowanie dwupoziomowe: tablica stron jest podzielona na strony
- w architekturze 64 bitowej → stronicowanie trzypoziomowe etc.

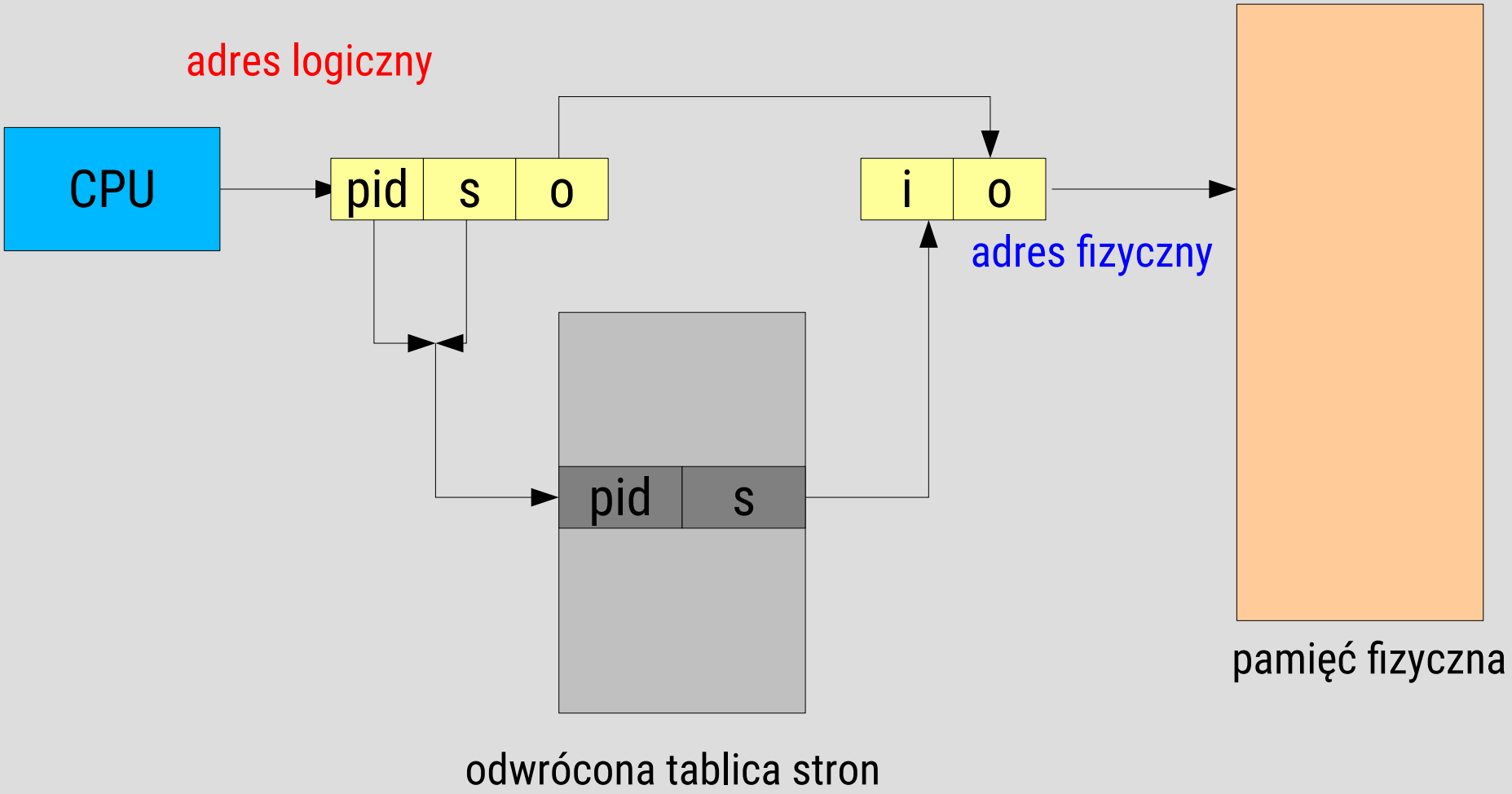
Odwrócona tablica stron

- tablica stron → duże zużycie pamięci fizycznej – z każdym procesem związana jest tablica stron
- rozwiązanie → **odwrócona tablica stron** (*inverted page table*)
  - posiada po jednej pozycji dla każdej ramki
  - indeksowana jest numerami ramek
  - każda pozycja zawiera adres wirtualny (logiczny) strony przechowywanej w ramce oraz informacje o procesie, do którego strona należy
  - jeżeli pojawia się odwołanie do pamięci, to następuje przeszukiwanie odwróconej tablicy stron w celu dopasowania adresu wirtualnego

Odwrócona tablica stron

- **oszczędność pamięci** → w systemie jest tylko jedna tablica stron
- **dłuższy czas potrzebny do przeszukania tablicy** przy odwołaniu do strony (ponieważ tablica ta jest uporządkowana według adresów fizycznych, a przeglądanie dotyczy adresów wirtualnych)

Stronicowanie z odwróconą tablicą stron



Strony dzielone

- zaleta stronicowania → możliwość dzielenia wspólnego kodu przez wiele procesów

- **kod dzielony:**
  - kod wznawialny (*reentrant code*) → kod, który nie modyfikuje sam siebie i jest czystą procedurą (*pure procedure*)

  - kod wznawialny może być współdzielony przez wiele procesów

  - kod dzielony musi znajdować się w tej samej pozycji w fizycznej przestrzeni adresowej wszystkich korzystających z niego procesów.

Strony dzielone

prywatny kod i dane:

- każdy proces posiada własną kopię rejestrów i obszar danych prywatnych
- strony prywatnego kodu i danych mogą pojawiać się w dowolnym miejscu w fizycznej przestrzeni adresowej

Segmentacja

Segmentacja → schemat zarządzania pamięcią urzeczywistniający sposób widzenia pamięci przez użytkownika

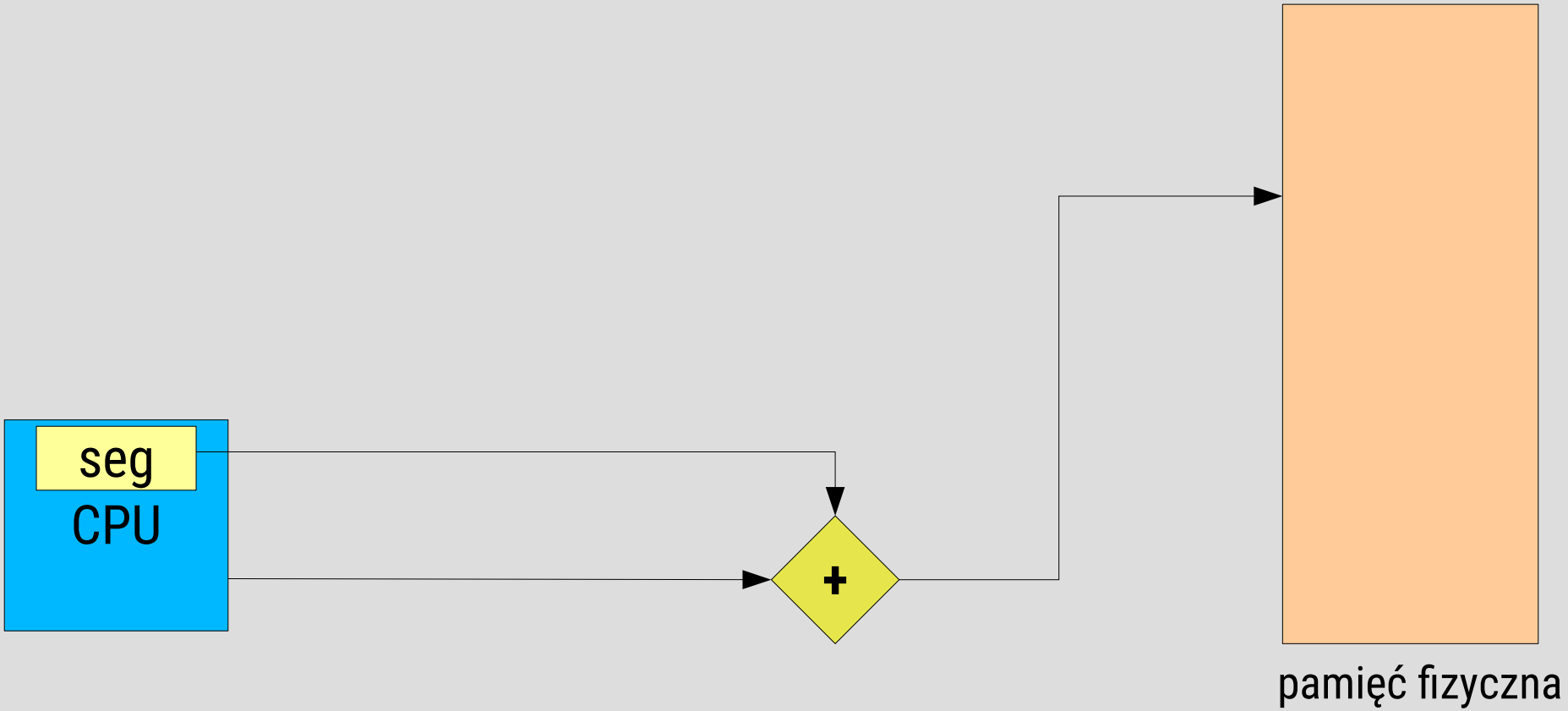
- przestrzeń adresów logicznych jest zbiorem segmentów
- **segment** → logiczny składnik procesu, np. program główny, procedura, funkcja, metoda, obiekt, stos, wykaz, tablica, zmienna lokalna/globalna
- dla ułatwienia implementacji segmenty są ponumerowane
- adres logiczny:
  `<numer_segmentu, odległość_od_początku_segmentu>`
- ponieważ numer segmentu może być krótszy od adresu segmentu (albo wręcz domyślny), można uzyskać efekt skrócenia adresu → krótszy kod wykonywalny

Segmentacja

rejestry segmentów: odwzorowują dwuwymiarowe adresy logiczne w jednowymiarowe adresy fizyczne

- każdy rejestr zawiera początkowy adres fizyczny segmentu w pamięci
- Intel x86 → 4 rejestry segmentowe (kod, stos, dane, dane dodatkowe)

Sprzęt do segmentacji



Segmentacja

- zaleta segmentacji → powiązanie ochrony pamięci z segmentami (segmenty są określonymi semantycznie porcjami programu, które powinny być używane w ten sam sposób)

- z każdym segmentem mogą być związane są:
  - bit poprawności: zero → użycie segmentu jest niedozwolone
  - prawa czytania/pisania/wykonania dla segmentu.

- potencjalne skrócenie kodu wynikowego oraz możliwość automatycznego ponownego wiązania adresów poprzez zmianę zawartości rejestru segmentowego

Segmentacja

zaleta segmentacji → dzielenie kodu lub danych:

- dzielenie występuje na poziomie całych segmentów, np. kilka procesów może dzielić segmenty zawierające funkcje biblioteczne
- dzielenie segmentów występuje wtedy, gdy rejestry segmentowe różnych procesów wskazują na to samo miejsce w pamięci fizycznej
- dowolna informacja może być dzielona, jeśli została zdefiniowana jako segment

Segmentacja ze stronicowaniem

Intel x86 (386+):

- max liczba segmentów w jednym procesie: 16 K
- max rozmiar segmentu: 4 GB, max rozmiar strony: 4 KB
- przestrzeń adresów logicznych → dwie strefy:
  - strefa prywatnych segmentów procesu: 8K segmentów → tablica lokalnych deskryptorów (*local descriptor table*, LDT)
  - strefa segmentów wspólnych dla wszystkich procesów: 8K segmentów → tablicy globalnych deskryptorów (GDT)
- każdy segment jest stronicowany w schemacie stronicowania dwupoziomowego:
  (s1 = 10 bitów, s2 = 10 bitów, o = 12 bitów) → razem 32 bity

Pamięć wirtualna - podstawy

- pamięć wirtualna → technika umożliwiająca wykonywanie procesów, pomimo że nie są one w całości przechowywane w pamięci operacyjnej
- pamięć wirtualna pozwala stworzyć abstrakcyjną pamięć główną w postaci olbrzymiej, jednolitej tablicy i odseparować pamięć logiczną od pamięci fizycznej

Pamięć wirtualna - podstawy

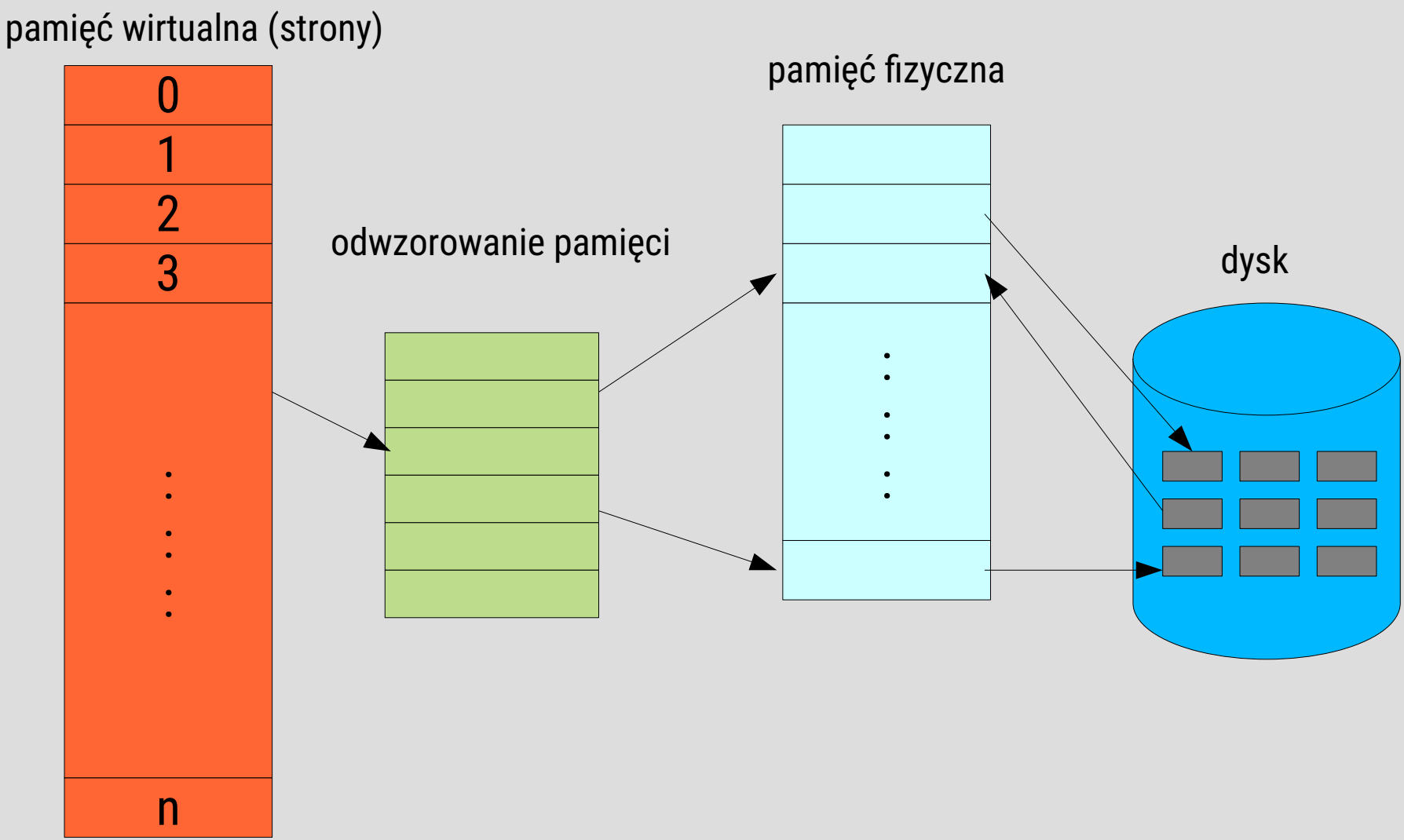
- logiczna przestrzeń adresowa może być większa niż przestrzeń fizyczna
- umożliwia to wykonywanie programu, który tylko w części znajduje się w pamięci fizycznej → program nie jest ograniczony wielkością dostępnej pamięci fizycznej (może być od niej większy)
- powiększenie wieloprogramowości → więcej procesów może być wykonywanych w tym samym czasie → lepsze wykorzystanie procesora i większa przepustowość
- wydajniejsze tworzenie i wymiana procesów → mniej potrzebnych operacji wejścia-wyjścia
- uwalnia użytkownika od potrzeby znajomości organizacji i ograniczeń pamięci fizycznej → ułatwia programowanie

Pamięć wirtualna - podstawy

możliwe sposoby implementacji pamięci wirtualnej:

- **stronicowanie na żądanie** (*demand paging*) → powszechniejsze
- **segmentacja na żądanie** (*demand segmentation*)

Pamięć wirtualna - schemat



Stronicowanie na żądanie

- procesy podzielone na strony przebywają w pamięci pomocniczej
- strona jest sprowadzana z pamięci pomocniczej do operacyjnej **tylko wtedy, kiedy jest potrzebna**
  - **+**mniej operacji WE/WY koniecznych do załadowania i wymiany programu
  - **+**mniejsze zapotrzebowanie na pamięć
  - **+**krótszy czas odpowiedzi
  - **+**równocześnie może pracować większa liczba procesów/użytkowników

Stronicowanie na żądanie

- strona jest **potrzebna,** kiedy następuje **odniesienie** do niej
- z każdą pozycją w tablicy stron związany jest bit poprawności (*valid-invalid bit*):
  - 1: strona dozwolona i znajduje się w pamięci operacyjnej
  - 0: strona niedozwolona albo jest poza pamięcią operacyjną
- odwołanie do strony z bitem poprawności równym 0 powoduje błąd strony (*page-fault*), po czym następuje awaryjne przejście do systemu operacyjnego

Obsługa błędu strony

- sprawdzamy wewnętrzną tablicę (w PCB), aby określić, czy odwołanie do pamięci było dozwolone
- jeżeli odwołanie było niedozwolone, to kończymy proces
- jeżeli zabrakło właściwej strony w pamięci, to sprowadzamy tę stronę
- znajdujemy wolną ramkę (na podstawie listy wolnych ramek)
- zamawiamy wczytanie z dysku potrzebnej strony do wolnej ramki
- po zakończeniu czytania modyfikujemy wewnętrzną tablicę procesu oraz tablicę stron, zaznaczając, że strona jest w pamięci
- wznawiamy wykonanie rozkazu przerwanego brakiem strony i kontynuujemy wykonywanie procesu (*uwaga! do wykonania rozkazu konieczna może być więcej niż jedna strona*)

Obsługa błędu strony

skrajny przypadek: czyste stronicowanie na żądanie (*pure demand paging*) → rozpoczynanie wykonywania procesu bez umieszczania jakiejkolwiek jego strony w pamięci fizycznej (nigdy nie sprowadza się strony do pamięci, zanim nie będzie potrzebna)

Kopiowanie przy zapisie

- tworzenie procesu przy pomocy wywołania systemowego fork() → można początkowo pominąć konieczność stronicowania na żądanie
- po wywołaniu fork() proces potomny i macierzysty dzielą początkowo te same strony, oznaczone jednak jako „kopiowane przy zapisie” (*copy-on-write*)
- jeżeli któryś z procesów próbuje zapisać coś na takiej stronie, to tworzona jest kopia tej strony i odwzorowywana w przestrzeni adresowej tego procesu
- wszystkie nie zmienione strony mogą być dzielone przez procesy

  zaleta → błyskawiczne utworzenie procesu i minimalizacja liczby

Zastępowanie stron

co zrobić, kiedy nie ma wolnej ramki dla potrzebnej strony?

zastępowanie stron (*page replacement*):

- zlokalizowanie potrzebnej strony na dysku
- wykonanie algorytmu zastępowania stron w celu wytypowania ramki-ofiary (*victim frame*)
- zapisanie strony-ofiary na dysku oraz stosowna modyfikacja tablicy stron i tablicy ramek
- wczytanie potrzebnej strony do zwolnionej ramki oraz modyfikacja tablicy stron i tablicy ramek
- wznowienie działania procesu.

Zastępowanie stron

- może powodować dwukrotne wydłużenie czasu obsługi braku strony → przesyłanie z pamięci na dysk i z dysku do pamięci
- aby zmniejszyć narzut związany z zastępowaniem stron, każdą stronę lub ramkę wyposaża się w sprzętowy bit modyfikacji (zabrudzenia - *modify (dirty) bit*) – tylko strony zmodyfikowane są wysyłane na dysk podczas zastępowania
- zastępowanie stron dopełnia oddzielenie pamięci logicznej od pamięci fizycznej

Algorytmy zastępowania stron

- algorytm zastępowania stron powinien minimalizować częstość występowania błędów stron (*page-fault rate*)
- algorytm ocenia się na podstawie wykonania go na pewnym ciągu odniesień (*reference string*) do pamięci i zsumowania błędów stron
- ciąg odniesień można tworzyć sztucznie lub na podstawie śledzenia realnego systemu

Algorytmy zastępowania stron
FIFO

- na ofiarę wybiera się tę ramkę, która najdłużej przebywa w pamięci fizycznej
- **anomalia Belady'ego** (*Belady's anomaly*) – współczynnik błędów stron wzrasta ze wzrostem liczby wolnych ramek

Algorytmy zastępowania stron
OPT

- wybierz na ofiarę tę stronę, która najdłużej nie będzie używana
- algorytm OPT ma najniższy współczynnik braków stron ze wszystkich algorytmów
- jest wolny od anomalii Belady'ego
- jest trudny w realizacji → wymaga wiedzy o przyszłej postaci ciągu odniesień (skąd ją wziąć?)
- algorytm OPT jest używany głównie w studiach porównawczych → punkt odniesienia dla innych algorytmów.

Algorytmy zastępowania stron
LRU

- wybierz na ofiarę ramkę najdawniej używaną (*least recently used* – LRU)
- używa niedawnej historii do oszacowania najbliższej przyszłości
- lepszy od FIFO (mniej błędów stron) i wolny od anomalii Belady'ego – często stosowany
- trudność z zapamiętywaniem historii użycia stron – może wymagać „ciężkiego" zaplecza sprzętowego

LRU - implementacja

implementacja z licznikiem:

- do każdej pozycji w tablicy stron dołącza się pole czasu użycia
- do procesora dodaje się zegar logiczny lub licznik
- wskazania zegara są zwiększane przy każdym odniesieniu do pamięci, a zawartość jego rejestru jest kopiowana do pola czasu użycia danej strony w tablicy stron
- kiedy pojawi się potrzeba zastąpienia jakiejś strony w pamięci, sprawdza się wartości czasu użycia stron i jako ofiarę wybiera się stronę z największą wartością czasu
- duży koszt:

  przeglądanie wszystkich stron

  uaktualnianie rejestrów czasu użycia

LRU - implementacja

Implementacja ze stosem:

- utrzymywanie stosu numerów stron
- przy każdym odwołaniu do strony jej numer wyjmuje się ze stosu i umieszcza na jego szczycie → na szczycie jest zawsze strona ostatnio użyta)
- najlepsza implementacja w postaci listy dwukierunkowej ze wskaźnikami do czoła i końca listy – wyjście strony i umieszczenie jej na szczycie wymaga zmiany co najwyżej 6 wskaźników
- zastąpienie strony nie wymaga przeszukiwania listy (najdawniej używana strona jest na dnie stosu)

LRU - przybliżenia

Algorytm bitów odniesienia:

- z każdą pozycją w tablicy stron związany jest bit odniesienia (*reference bit*), początkowo ustawiony na 0
- kiedy następuje odniesienie do strony, jej bit odniesienia jest ustawiany na 1
- zastępowana jest ta strona, która w porządku FIFO ma bit odniesienia = 0

LRU - przybliżenia

Algorytm dodatkowych bitów odniesienia:

- z każdą stroną związany jest 8-bitowy rejestr
- na początku → 0000000
- w regularnych odstępach czasu (np. co 100 ms) system operacyjny „wsuwa” bit odniesienia na najbardziej znaczącą pozycję rejestru
- jako ofiara wybierana jest strona zawierająca w rejestrze najmniejszą liczbę

LRU - przybliżenia

Algorytm drugiej szansy (zegarowy):

- strony przeglądane są w porządku FIFO
- jeżeli bit odniesienia = 0, to strona zostaje wybrana do zastąpienia
- jeżeli bit odniesienia = 1, to bit ten jest zerowany, a strona dostaje „drugą szansę”, tzn. do zastąpienia bierze się pod uwagę następną stronę etc.
- przeglądanie stron wykonuje się cyklicznie
- ulepszenie – branie pod uwagę pary:

  (bit odniesienia, bit modyfikacji)

LFU

- wprowadzenie liczników odniesień do każdej strony
- **algorytm LFU – najrzadziej używana** (*least fequently used*) → do zastąpienia wybiera się stronę o najmniejszej wartości licznika
- może być obarczony błędami wynikającymi z tego, że strona była na początku intensywnie używana, a potem była bezczynna
- rozwiązanie: przesuwanie liczników w prawo o 1 bit w regularnych odstępach czasu – wykładniczy spadek wartości licznika

MFU

- **algorytm MFU – najczęściej używanej strony** (*most fequently used*) → na ofiarę wybiera się stronę o największej wartości licznika
- uzasadnienie → strona o najmniejszej wartości licznika prawdopodobnie została niedawno wprowadzona do pamięci i będzie jeszcze używana.

Migotanie stron

- jeżeli proces nie ma wystarczającej liczby stron w pamięci operacyjnej, to częstotliwość braków stron będzie wysoka → zmniejszenia wykorzystania procesora
- mniejsze wykorzystanie procesora może być sygnałem dla planisty przydziału procesora do zwiększenia wieloprogramowości
- kolejny proces jest dodawany do systemu, co jeszcze zwiększa częstość błędów stron
- wykorzystanie procesora nadal się zmniejsza, co jest sygnałem dla planisty do dalszego zwiększania wieloprogramowości → błędne koło
- powstaje migotanie (szamotanina) → przepustowość systemu

Migotanie stron

migotanie (*thrashing*) – sytuacja, w której proces jest zajęty głównie/wyłącznie wymianą stron (przesyłaniem między dyskiem a pamięcią operacyjną)