

Pamięć wirtualna

Wykład 8.

Idea pamięci wirtualnej

- Pamięć wirtualna -- technika umożliwiająca wykonywanie procesów, które nie są w całości przechowywane w pamięci operacyjnej.
- Skutki (zalety i wady):
 - programy mogą być większe niż pamięć fizyczna
 - pamięć wirtualna umożliwia utworzenie abstrakcyjnej pamięci głównej w postaci dużej, jednolitej tablicy do magazynowania informacji, można oddzielić pamięć logiczną od pamięci fizycznej
 - zastosowanie pamięci wirtualnej może znacznie obniżyć wydajność systemu
- Sposoby realizacji pamięci wirtualnej:
 - stronicowanie na żądanie (ang. demand paging)
 - segmentacja na żądanie (ang. demand segmentation)

Czy konieczne jest ładowanie całego programu do pamięci na czas jego wykonania ?

- programy często zawierają fragmenty do obsługi wyjątkowo pojawiających się błędów
- tablice, listy i wykazy mają często przydzielone więcej pamięci, niż rzeczywiście potrzebują
- pewne procedury programu są rzadko stosowane

Czy konieczne jest ładowanie całego programu do pamięci na czas jego wykonania (2)?

- Własność lokalności
 - Program i referencje do danych procesie zazwyczaj grupują się w klaster
 - Tylko niewielka część procesu (kodu i danych) będzie potrzebna w najbliższym czasie
 - Dlatego są możliwe inteligentne próby odgadywania, które części procesu będą potrzebne w najbliższym czasie i dokonywanie odpowiedniego ładowania ich do pamięci, co umożliwia uniknięcie szamotania (utrzymanie błędnych odgadywań na niskim poziomie)

Zalety ładowania do PO jedynie części programów

- Program nie jest ograniczony wielkością dostępnej pamięci fizycznej. Możliwe jest pisanie programów w wielkiej wirtualnej przestrzeni adresowej, co ułatwia programowanie.
- Każdy program użytkownika może zajmować mniejszy obszar pamięci fizycznej. Można w tym samym czasie wykonywać więcej programów, zwiększyć wykorzystanie procesora i przepustowość, bez zwiększania czasu odpowiedzi i czasu cyklu przetwarzania.
- Maleje liczba operacji wejścia-wyjścia koniecznych do załadowania lub wymiany programów użytkowych w pamięci, zatem każdy program powinien się wykonywać szybciej.

Stronicowanie na żądanie

- procesy przebywają w pamięci pomocniczej
- proces, który należy wykonać zostaje wprowadzony do pamięci operacyjnej
- procedura leniwej wymiany (ang. lazy swapper) -
- nigdy nie wykonuje się wymiany stron w pamięci, jeśli nie jest to konieczne
- procedura stronicująca (ang. pager) zajmuje się wyborem stron
 - przed wprowadzeniem procesu do pamięci określa, które strony będą potrzebne przed ponownym wysłaniem procesu na dysk
 - procedura stronicująca wprowadza do pamięci tylko niezbędne strony

Stronicowanie na żądanie (2)

- Środki sprzętowe do odróżnienia stron w pamięci i na dysku
 - zastosowanie bitu poprawności (ang. *valid-invalid bit*)
 - bit ma wartość poprawne - strona znajduje się w pamięci operacyjnej i odwołanie do niej jest dozwolone
 - bit ma wartość niepoprawne - strona jest niedozwolona (nie należy do logicznej przestrzeni adresowej procesu) lub jest dozwolona, ale znajduje się na dysku

Obsługa błędu "*brak strony*"

- Odwołanie się do strony, która nie jest w pamięci powoduje wystąpienie błędu *brak strony* (ang. page fault).
- Obsługa braku strony:
 1. Sprawdzenie wewnętrznej tablicy (najczęściej przechowywanej w bloku kontrolnym procesu) , aby określić, czy odwołanie do pamięci jest dozwolone, czy nie.
 2. Jeśli odwołanie jest niedozwolone, wówczas proces jest kończony, jeśli jest dozwolone, ale właściwej strony nie ma w pamięci, to strona jest wprowadzana.
 3. Znajdowana jest wolna ramka (np. biorąc ramkę z listy wolnych ramek)
 4. Strona z dysku jest odczytywana i ładowana do nowo przydzielonej ramki
 5. Po zakończeniu czytania z dysku, modyfikowana jest tablica wewnętrzna procesu i tablica stron, odnotowywane zostaje, że strona jest w pamięci
 6. Wznowienie wykonania przerwanych rozkazów (do tego trzeba przechowywać stan przerwanych procesu rejestry, kody warunków, licznik rozkazów)

Sprawność stronicowania na żądanie

Efektywny czas dostępu (ang. effective access time) do pamięci stronicowanej na żądanie.

- **Nie występują braki stron**, wtedy efektywny czas dostępu jest równy czasowi dostępu do pamięci.
- **Występują braki stron.**

cd - czas dostępu do pamięci (około 10-200 ns)

p - prawdopodobieństwo braku strony (braki stron powinny być nieliczne, dlatego p powinno być bliskie 0)

$$\text{efektywny czas dostępu} = (1-p) \times cd + p \times \text{czas-obługi-braku-strony} \quad (*)$$

Główne składniki wpływające na czas obsługi braku strony:

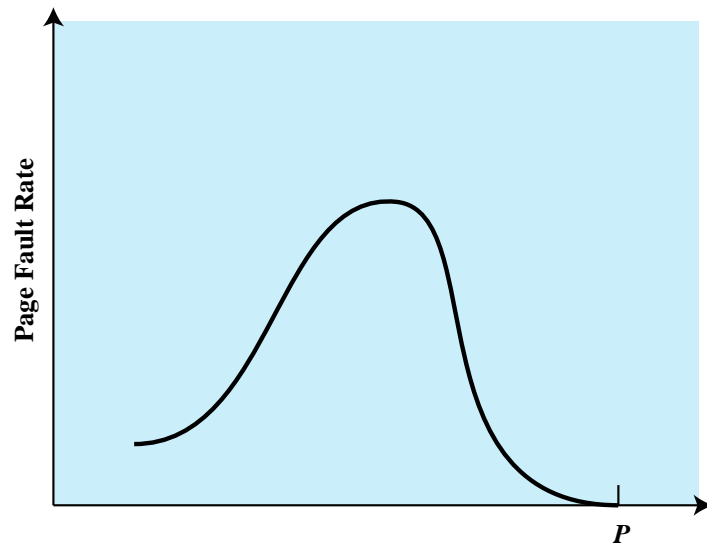
1. Obsługa przerwania wywołanego brakiem strony (1 do 100 μ s)
2. Czytanie strony (~ 25 ms)
3. Wznowienie procesu (1 do 100 μ s)

Sprawność stronicowania na żądanie (2)

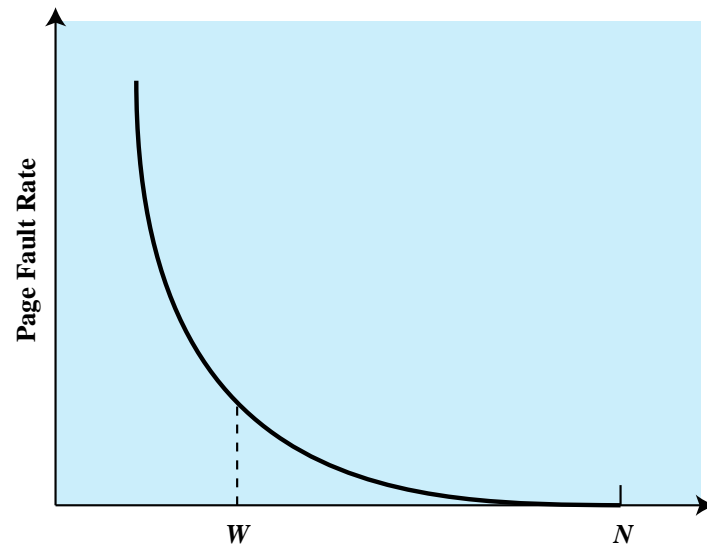
(*)

- jeśli 1 dostęp na 1000 powoduje brak strony, to komputer zostaje spowolniony 250 razy
- aby pogorszenie nie przekraczało 10%, dopuszczalny jest 1 brak strony na 2.5 miliona odniesień do pamięci

Typowe zachowanie programu podczas stronicowania



(a) Page Size



(b) Number of Page Frames Allocated

P = size of entire process

W = working set size

N = total number of pages in process

Realizacja obsługi braku strony - wersja bardziej szczegółowa

1. Przejście do systemu operacyjnego
2. Przechowanie rejestrów użytkownika i stanu procesu
3. Określenie, że przerwanie było spowodowane brakiem strony
4. Sprawdzenie, czy odniesienie do strony było dopuszczalne i określenie położenia strony na dysku
5. Wydanie polecenia czytania z dysku do wolnej ramki:
 - czekanie w kolejce do odpowiedniego dysku, aż będzie obsłużone zamówienie na czytanie
 - czekanie przez czas szukania informacji na dysku i (lub) jej nadchodzenie
 - rozpoczęcie przesyłania strony do wolnej ramki
6. Przydzielenie procesora innemu użytkownikowi na czas oczekiwania bieżącego użytkownika
7. Otrzymanie przerwania od dysku (zakończenie operacji we/wy)
8. Przechowanie rejestrów i stanu procesu innego użytkownika (jeśli wykonano 6)
9. Określenie, czy przerwanie pochodziło od dysku
10. Skorygowanie zawartości tablicy stron i innych tablic, w celu odnotowania, że strona jest teraz w pamięci
11. Czekanie na powtórne przydzielenie procesora danemu procesowi
12. Odtworzenie stanu rejestrów użytkownika, stanu procesu i nowej tablicy stron, oraz wznowienie przerwane rozkazu

Zastępowanie stron

- właściwy dobór stron ładowanych do pamięci pozwala na ograniczenie operacji we/wy i zwiększenie stopnia wieloprogramowości
- nadprzydział pamięci (ang. *over-allocating*) - brak wolnych ramek do załadowania potrzebnej strony -- stronicowanie na żądanie musi znaleźć rozwiązanie niewidoczne dla użytkownika:
 - wymiana procesu - zwolnienie wszystkich jego ramek i zmniejszenie poziomu wieloprogramowości
 - **zastępowanie stron** (ang. *page replacement*) -> trzeba wybrać spośród zajętych ramek tę, która nie jest używana i zwolnić ją, a następnie wprowadzić na jej miejsce potrzebną stronę
- W celu ograniczenia nakładu czasu na obsługę braku strony stosuje się:
 - bit modyfikacji/zabrudzenia (ang. *modify (dirty) bit*) - stowarzyszony z każdą stroną, wskazuje, że strona uległa modyfikacji
- Podstawowe problemy stronicowania na żądanie:
 - algorytm przydziału ramek (ang. *frame-allocation algorithm*) - określa, ile ramek należy przydzielić każdemu procesowi
 - algorytm zastępowania stron (ang. *page-replacement algorithm*) - wybiera ramki do wymiany

Algorytmy zastępowania stron

- **pożądane właściwości algorytmu zastępowania stron:**
 - minimalizacja częstości braków stron (ang. page-fault rate)
 - łatwość i koszt realizacji algorytmu
- **ocena algorytmu** - na podstawie jego wykonania na pewnym ciągu odniesień (ang. *reference string*) do pamięci i po zsumowaniu liczby braków stron
- **sposoby tworzenia ciągów odniesień:**
 - sztucznie (np. losowo)
 - na podstawie śledzenia jakiegoś systemu - zapisując wszystkie odwołania do pamięci (numery stron, do których nastąpiły odwołania)

Algorytm FIFO

(pierwszy na wejściu-pierwszy na wyjściu, ang. first-in-first-out)

- z każdą stroną jest kojarzony jej czas wprowadzenia do pamięci
- zamiast zapamiętywać czasy dla każdej strony, można przechowywać odsyłacze do stron w kolejce FIFO
- w razie konieczności wymiany – wymieniana jest najstarsza strona
- **Zalety:**
 - algorytm łatwy do zrozumienia i zaprogramowania
- **Wady:**
 - strona dawno załadowana może być nadal potrzebna
 - możliwość wystąpienia tzw. anomalii Belady'ego (ang. *Belady's anomaly*) – w pewnych sytuacjach liczba braków stron dla większej ilości ramek może być wyższa od liczby braków stron, gdyby do dyspozycji była mniejsza ilość ramek
 - anomalia Belady'ego oznacza, że współczynnik braków może czasem wzrastać wraz z ilością ramek

Algorytm optymalny (OPT, MIN)

- cechuje go najniższy współczynnik braków stron
- nie jest dotknięty anomalią Belady'ego
- używany głównie do studiów porównawczych
- różnice między algorytmem FIFO i algorytmem optymalnym
 - w FIFO jest brany pod uwagę czas wprowadzania strony do pamięci, a w optymalnym – czas w którym strona ma być użyta
 - FIFO „patrzy” wstecz, a optymalny – do przodu
- **Idea algorytmu:**
 - zastąpiona zostaje strona, która najdłużej nie będzie używana
- **Wady algorytmu idealnego:**
 - trudność realizacyjna
 - wymaga wiedzy o przyszłej postaci ciągu odniesień

Algorytm LRU (zastępowanie najdawniej używanych stron, ang. least recently used)

- zastępowana jest strona, która najdawniej nie była używana
- do oszacowania sytuacji w przyszłości jest używane badanie stanów w niedawnej przeszłości
- z każdą stroną jest kojarzony czas jej ostatniego użycia
- algorytm stosowany często i uważany za dość dobry
- trudność -- implementacja zastępowania stron wg. kolejności wynikającej z algorytmu LRU
 - może być konieczne znaczne wsparcie sprzętowe
 - problem, jak określić kolejność ramek na podstawie ich ostatniego użycia
- zastępowanie stron metodą LRU nie jest dotknięte anomalią Belady'ego

Sposoby realizacji LRU:

- zastosowanie licznika
- zastosowanie stosu

Algorytmy stosowe (ang. stack algorithms)

- dla tych algorytmów można wykazać, że zbiór stron w pamięci w przypadku istnienia n ramek jest zawsze podzbiorem zbioru stron znajdujących się w pamięci przy $n+1$ ramkach
- nie są one dotknięte anomalią Belady'ego

Algorytm LRU. Implementacja z licznikiem

Idea:

- do każdej pozycji w tablicy stron jest dołączany rejestr czasu użycia
- do procesora jest dodawany zegar logiczny, lub licznik
- wskazania zegara są zwiększane wraz z każdym odniesieniem do pamięci
- przy odniesieniu do strony, zawartość zegara jest kopiowana do rejestru czasu użycia należącego do danej strony tablicy stron
- zastępowana jest strona z najmniejszą wartością czasu

Wymagania i problemy:

- realizacja algorytmu wymaga przeglądania tablicy stron w celu znalezienia strony najdawniej używanej
- rejestry czasu użycia są aktualizowane przy odwołaniach do strony i przy wymianach stron
- trzeba rozwiązać problem nadmiaru w rejestrze zegara

Algorytm LRU. Implementacja stosowa

Idea:

- utrzymywanie stosu numerów stron
- przy każdym odwołaniu do strony jej numer wyjmuje się ze stosu i umieszcza na jego szczycie
- na szczycie stosu jest zawsze numer strony użytej jako ostatnia, a poniżej kolejne strony używane coraz dawniej

Koszt:

- trzeba wydobywać pozycje z wnętrza stosu,
- najlepsza implementacja oparta jest na zastosowaniu listy dwukierunkowej ze wskaźnikami do czoła i końca listy
- wyjęcie strony i ulokowanie jej na szczycie wymaga w najgorszym razie aktualizacji 6 wskaźników
- nie jest konieczne przeszukiwanie listy
- metoda przydatna do implementacji algorytmu LRU za pomocą oprogramowania systemowego lub mikroprogramu

Algorytmy przybliżające metodę LRU

- aby algorytm LRU mógł działać z odpowiednią szybkością wymagane jest znaczne wsparcie sprzętowe
- wiele systemów nie ma wystarczającego wsparcia, co powoduje konieczność używania innych algorytmów (np. FIFO)
- w wielu systemach są stosowane tylko ograniczone środki wspomaganie – bity odniesienia, związane z każdą pozycją w tablicy stron
- bit odniesienia jest ustawiany dla danej strony przez sprzęt wtedy, gdy występuje do niej odniesienie (czytanie lub pisanie dowolnego bajta na stronie)

Działanie algorytmu z bitami odniesienia

- na początku wszystkie bity odniesienia są zerowane
- w trakcie wykonania procesu, bit związany ze stroną do której następuje odwołanie jest ustawiany przez sprzęt (ma nadawaną wartość 1)
- na podstawie wartości bitów odniesienia można określić, które strony były używane i usuwać tylko te strony, które używane nie były
- nie można określić (za mało danych):
 - porządku użycia stron
 - częstotliwości użycia stron

Algorytmy przybliżające metodę LRU:

Algorytm dodatkowych bitów odniesienia

- stany bitów odniesienia są odnotowywane w regularnych odstępach czasu – daje to algorytmowi więcej informacji o użyciu stron
- dla każdej strony przeznaczona jest bajt (8 bitów) i przechowywana jest w pamięci operacyjnej
- w regularnych okresach czasu, przerwanie zegarowe powoduje przekazanie sterowania do systemu operacyjnego
 - dla każdej strony system operacyjny wprowadza bit odniesienia na najbardziej znaczącą pozycję odpowiedniego bajta (który pełni rolę rejestru przesuwanego związanego z daną stroną)
 - pozostałe bity są przesuwane o jedną pozycję w prawo; na pozycji najmniej znaczącej następuje utrata bitu
 - bit odniesienia jest zerowany
- rejestry przesuwne zawierają historie użycia wszystkich stron podczas 8 ostatnich okresów
 - strona o wartości rejestru 00000000 nie była używana podczas 8 ostatnich okresów
 - strona o wartości rejestru 11111111 była używana podczas każdego z 8 ostatnich okresów
- rejestry można traktować jako 8-bitowe liczby bez znaku i porównywać je – najmniejsza wartość odpowiada stronie najdawniej używanej
- wartości rejestrów przesuwne mogą być równe dla wielu stron – można wtedy wymienić wszystkie strony o najmniejszej wartości, albo np. zastosować alg. FIFO

Algorytmy przybliżające metodę LRU:

Algorytm drugiej szansy (zegarowy)

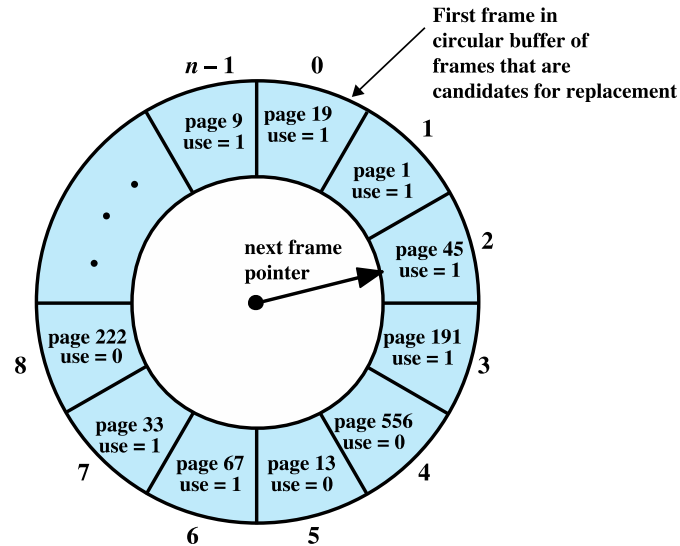
- liczba bitów historii w *algorytmie dodatkowych bitów odwołań* może być różna, w skrajnym przypadku może zostać zredukowana do 0 (pozostaje tylko bit odniesienia) → algorytm zastępowania stron na zasadzie drugiej szansy
- algorytm oparty na algorytmie FIFO
 - po wybraniu strony sprawdza się jej wartość jej bitu odniesienia
 - jeżeli jest 0, to strona zostaje zastąpiona
 - jeżeli jest 1, to strona pozostaje w pamięci, ale jej bit odniesienia jest zerowany, a jej czas przybycia ustawiany na czas bieżący

Sposoby realizacji algorytmu drugiej szansy:

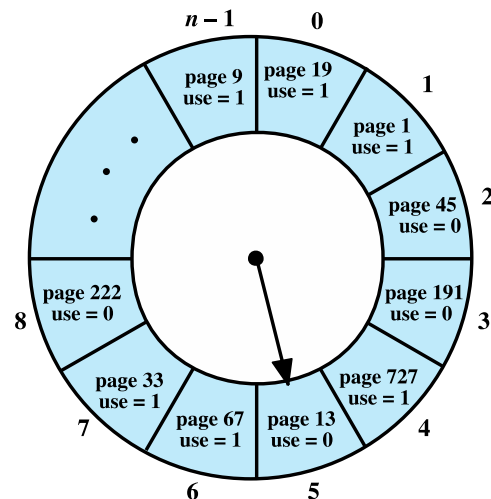
- kolejka cykliczna
- wskaźnik pokazuje na stronę, która może być zastąpiona w następnej kolejności
- po zapotrzebowaniu na ramkę wskaźnik przemieszcza się naprzód, aż zostanie odnaleziona strona z wyzerowanym bitem odniesienia
- podczas przemieszczania wskaźnika, analizowane bity odniesienia są zerowane

Algorytmy przybliżające metodę LRU:

Algorytm drugiej szansy (zegarowy) - przykład



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Algorytmy przybliżające metodę LRU:

Ulepszony algorytm drugiej szansy

- brane są pod uwagę: bit odniesienia i bit modyfikacji, są one traktowane jak uporządkowana para

Istnieją 4 klasy stron

- (0,0) – ostatnio nie używana i nie zmieniana – najlepsza strona do zastąpienia
- (0,1) – ostatnio nie używana, ale zmieniona – nieco gorsza kandydatka, gdyż trzeba ją będzie zapisać na dysk przed zastąpieniem
- (1,0) – używana ostatnio, ale nie zmieniona – niedługo może być używana
- (1,1) – używana ostatnio i zmieniona – niedługo może być używana, w przypadku wymiany trzeba ją będzie zapisać na dysk

Działanie algorytmu:

- postępowanie analogiczne jak w algorytmie zegarowym, ale sprawdza się, do której klasy strona należy
- zastępowana jest pierwsza napotkana strona z najniższej niepustej klasy

Algorytmy zliczające: Algorytm LFU i MFU

- wprowadzenie liczników odwołań do każdej ze stron
- oba poniższe algorytmy nie są popularne (implementacja dość kosztowna i nie przybliżają wystarczająco dobrze algorytmu OPT)

LFU (alg. zastępowania strony najrzadziej używanej, ang. least frequently used)

- zakłada się, że strona aktywnie użytkowana powinna mieć wysoką wartość licznika odwołań
- problemy:
 - strona może być aktywnie używana w przeszłości, a potem przestaje być używana – jednym z rozwiązań jest przesuwanie liczników w prawo o 1 bit w regularnych odstępach czasu,
 - strony dopiero wprowadzone do pamięci mogą mieć stosunkowo niską wartość licznika

MFU (alg. zastępowania strony najczęściej używanej, ang. most frequently used)

- zakłada się, że strona z najmniejszą wartością licznika została prawdopodobnie dopiero wprowadzona do pamięci

Przykład działania podstawowych algorytmów

Page address
stream

2 3 2 1 5 2 4 5 3 2 5 2

OPT

2	2	2	2	2	2	4	4	4	2	2	2
	3	3	3	3	3	3	3	3	3	3	3
			1	5	5	5	5	5	5	5	5
				F		F			F		

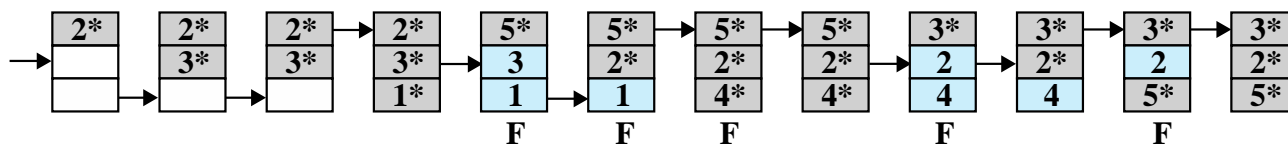
LRU

2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2	2
				F		F		F	F		

FIFO

2	2	2	2	5	5	5	5	3	3	3	3
	3	3	3	3	2	2	2	2	2	5	5
			1	1	1	4	4	4	4	4	2
				F	F	F		F		F	F

CLOCK



F = page fault occurring after the frame allocation is initially filled

Procedura uzupełniająca algorytmy zastępowania stron -

Algorytm buforowania stron

- system przechowuje pulę wolnych ramek
- po wystąpieniu błędu strony ramkę do wywłaszczenia wybiera się jedną z przedstawionych poprzednio metod
- rozwinięcie 1:
 - zanim strona-ofiara zostanie usunięta z pamięci, potrzebna strona jest czytana do którejś z wolnych ramek puli
 - to postępowanie pozwala na maksymalnie szybkie wznowienie procesu, bez oczekiwania na przepisanie strony-ofiary do pamięci operacyjnej
 - jeżeli potem strona-ofiara zostanie przepisana do pamięci zewnętrznej, wówczas zajmowana przez nią ramka jest dołączana do puli wolnych ramek
- rozwinięcie 2 - utrzymywanie listy zmienionych stron
 - gdy urządzenie stronicujące jest bezczynne, wybiera się zmienioną stronę i zapisuje się ją na dysku
 - bit modyfikacji wybranej strony jest zerowany
 - wzrost prawdopodobieństwa, że wybrana do zastąpienia strona nie będzie oznaczona jako zmodyfikowana i nie jest konieczny jej zapis trzeba na dysku
- rozwinięcie 3
 - utrzymywanie puli wolnych ramek i pamiętanie, które strony rezydowały w każdej z ramek
 - jak długo ramka nie zostanie ponownie użyta, tak długo pozostająca w niej strona może być ponownie użyta z puli wolnych ramek

Metody przydziału ramek

- problem: jak podzielić ramki pomiędzy procesy ?
- "czyste stronicowanie" :
 - globalne, dla całego systemu
 - zastępowanie stron dopiero wtedy, gdy wszystkie ramki zostaną wypełnione,
 - brak przydziału limitów ramek dla procesów

Przydział ramek: maksymalna i minimalna liczba ramek

- maksymalna liczba ramek – wynika z ilości pamięci fizycznej
- minimalna liczba ramek, które muszą być przydzielone -- określona przez zbiór rozkazów w architekturze komputera:
 - jeśli brak strony wystąpi przed zakończeniem wykonywania rozkazu, to rozkaz musi być powtórzony
 - trzeba mieć wystarczającą ilość ramek, by przechować wszystkie strony, do których może się odnosić pojedynczy rozkaz
 - rozkaz może przekraczać długość słowa, zatem może się znajdować na sąsiednich stronach
 - trzeba ponadto uwzględnić odniesienia do pamięci, adresowanie pośrednie (i jego poziomy), rozmiary obszarów w pamięci, do których nastąpiło odwołanie itd.

Przydział ramek: Algorytmy przydziału

- n – ilość procesów
- m – ilość ramek
- przydział równy (ang. *equal allocation*)
 - każdy proces otrzymuje m/n ramek
 - ewentualne pozostałe wolne ramki mogą być traktowane jako bufor wolnych ramek
 - słaby punkt: procesy mają różne rozmiary i różne zapotrzebowanie na ramki
- przydział proporcjonalny (ang. *proportional allocation*)
 - każdemu procesowi przydziela się dostępne ramki w zależności od jego rozmiaru
 - s_i – wielkość pamięci wirtualnej procesu p_i
 - $S = \sum s_i$ - suma pamięci wirtualnych wszystkich procesów
 - m_i – ilość ramek przydzielanych procesowi p_i
 - $m_i = s_i / S \cdot m$
- przydział z uwzględnieniem priorytetów
 - wymaga się, że proces o wyższym priorytecie, w celu przyspieszenia jego działania, otrzymuje więcej pamięci niż proces niskopriorytetowy
 - np. przydział proporcjonalny, w którym ilość ramek zależy nie od względnych rozmiarów procesów, ale od priorytetów procesów, lub od kombinacji rozmiaru i priorytetu

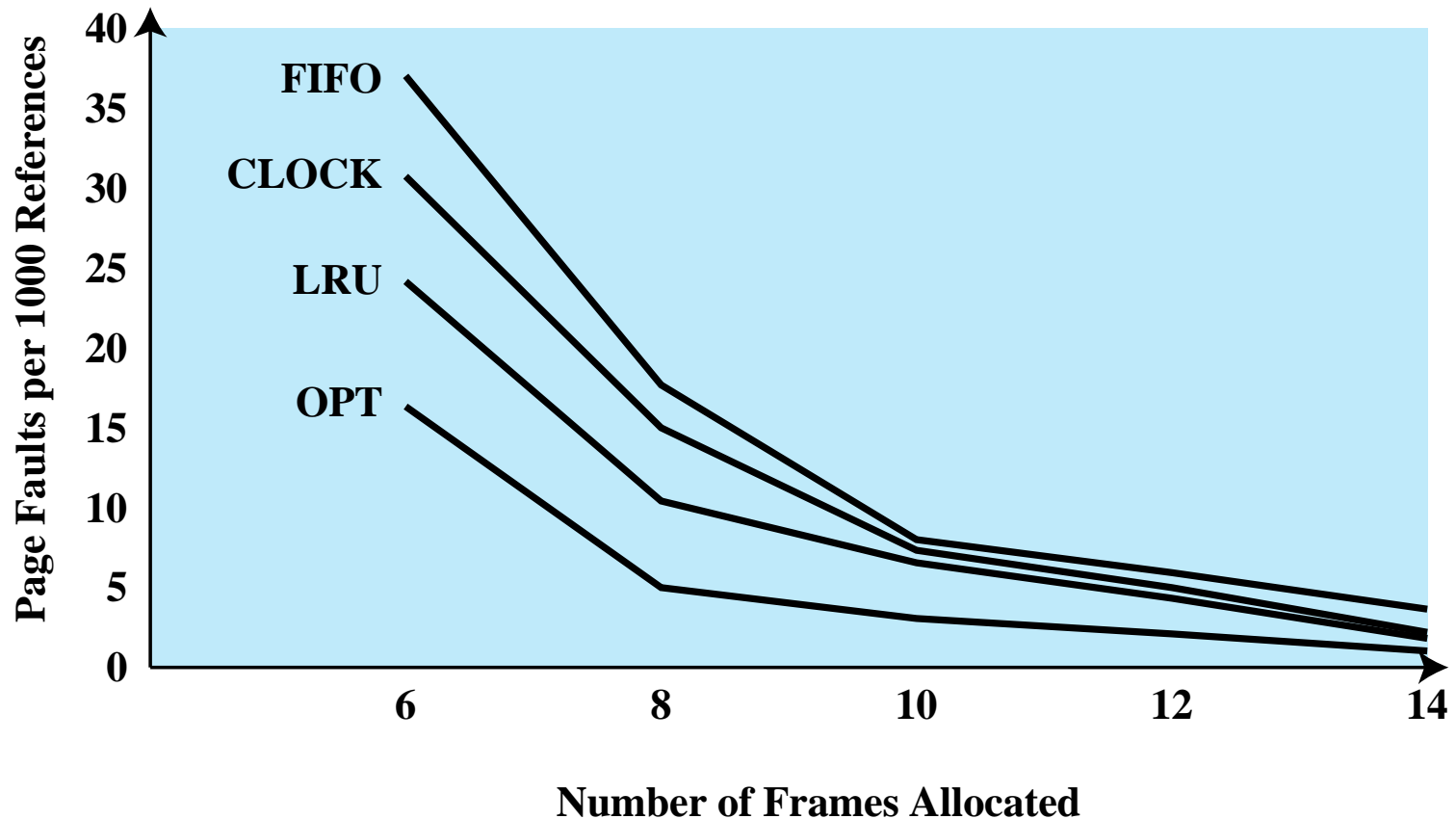
Przydział ramek: Zastępowanie globalne i lokalne

- **zastępowanie globalne** (ang. *global replacement*)
 - proces może pobierać ramki ze zbioru wszystkich ramek, nawet jeżeli dana ramka jest aktualnie przydzielona innemu procesowi
 - np. proces wysokopriorytetowy może dokonać wymiany strony w swojej ramce, albo zabrać ramkę procesowi niskopriorytetowemu
 - proces nie może kontrolować własnej czynności występowania braków stron – zbiór stron procesu w pamięci zależy nie tylko od zachowania danego procesu, ale także od zachowania innych procesów
- **zastępowanie lokalne** (ang. *local replacement*) –
 - liczba ramek przydzielonych do procesu nie ulega zmianom
 - proces dokonuje wymian tylko w obrębie swoich ramek
 - zbiór stron procesu w pamięci zależy tylko od stronicowania odnoszącego się do danego procesu
 - ze względu na ograniczenie się tylko do fragmentu pamięci, może nastąpić spowolnienie wykonania się procesu wskutek istnienia niedostępnych, mniej używanych stron pamięci

Szamotanie: Przyczyny

- Szamotanie (ang. *trashing*)
 - sytuacja, w której proces musi dokonywać częstych wymian stron
 - proces szamocze się, jeśli spędza więcej czasu na stronicowaniu, niż na wykonaniu
- Przykładowa sytuacja:
 - proces wchodzi w nową fazę działania i potrzebuje więcej ramek
 - wykazuje braki stron i powoduje utratę stron przez inne procesy
 - inne procesy także wykazują braki stron
 - wskutek oczekiwania procesów na przydział stron zmniejsza się wykorzystanie procesora
 - jeśli spada zużycie procesora, wtedy planista może chcieć zwiększyć poziom wieloprogramowości przyjmując nowe procesy i przydzielając im pamięć
- Sposób ograniczenia szamotania:
 - lokalny(lub priorytetowy) algorytm zastępowania – szamocący się proces nie doprowadza do szamotania innych procesów
 - należy dostarczyć procesowi tyle ramek, ile potrzebuje

Porównanie popularnych algorytmów wymiany stron dla stałych liczb alokowanych stron dla procesów i z lokalnym algorytmem zastępowania stron



Szamotanie: Model zbioru roboczego

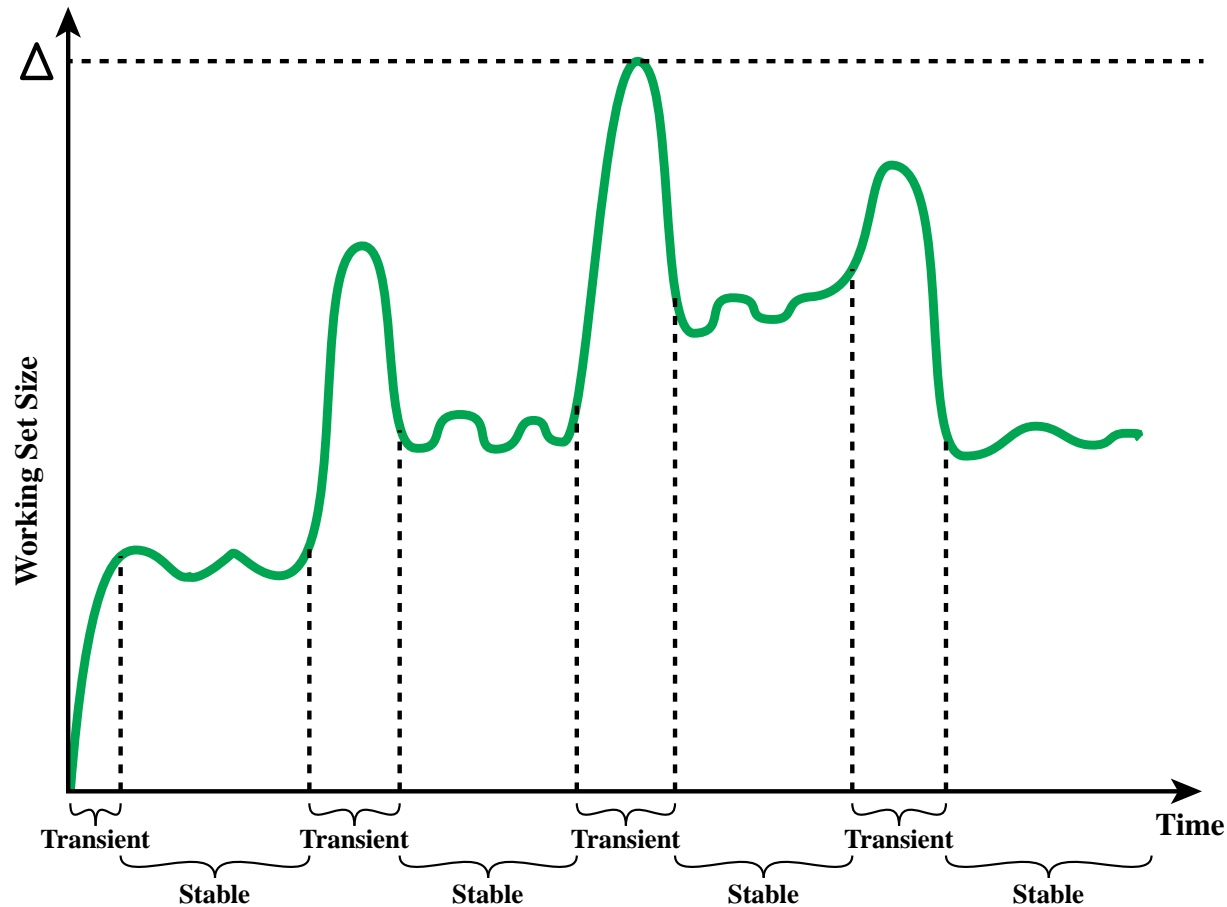
- Model strefowy :
 - podczas swego wykonania program przebywa w różnych strefach, w każdej z nich odwołuje się do innych zbiorów stron
- model zbioru roboczego (ang. working-set model)
 - zakłada się, że program ma charakterystykę strefową
 - Δ -- definiuje okno zbioru roboczego (ang. working-set window), sprawdza się Δ ostatnich odwołań do pamięci
 - zbiór roboczy – zbiór stron, do których nastąpiło Δ ostatnich odwołań
 - konsekwencje: jeśli strona jest używana, to powinna się znaleźć w zbiorze roboczym
 - jeśli strona przestaje być używana, to jest usuwana ze zbioru roboczego
 - proces powinien mieć w pamięci strony ze zbioru roboczego
 - obliczając dla każdego procesu rozmiar zbioru roboczego, można określić aktualne zapotrzebowanie na ramki w systemie
 - jeśli ogólna liczba ramek jest mniejsza od ilości potrzebnych ramek, to dochodzi do szamotania
 - stosując zbiór roboczy można utrzymywać poziom wieloprogramowości na możliwie najwyższym poziomie, przy którym nie dochodzi do szamotania

Dynamika zbioru roboczego – przykład [Bach86]

References

	2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	18 24	•	24 17 18
17	18 17	24 18 17	•	•
17	17	18 17	•	•
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	•
17	24 17	•	•	17 15 24
24	•	24 17	•	•
18	24 18	17 24 18	17 24 18	15 17 24 18

Przykładowe zmiany zbioru rozmiaru roboczego podczas wykonania procesu przy zadanym rozmiarze okna Δ [MAEK87]



Szamotanie: Częstość braków stron

- prostsza metoda nadzorowania szamotania – mierzenie częstości braków stron (ang. *page-fault frequency* PFF)
- wysoka liczba braków stron – proces potrzebuje więcej ramek
- niska liczba braków stron – proces ma zbyt wiele ramek

Metoda ładowania stron (ang. *fetching policy*)

- Stronicowanie na żądanie (ang. *demand paging*)
- Stronicowanie wstępne (ang. *prepaging*)

Własności stronicowania na żądanie

- Strony są ładowane do pamięci tylko gdy następuje referencja do zawartości danej strony
- Wiele błędów stron podczas pierwszego uruchomienia procesu
- Własność lokalności sugeruje, że po ładowaniu większej liczby stron, większość przyszłych referencji będzie dotyczyło stron już załadowanych, a zatem wskaźnik błędów stron powinien spaść do bardzo niskiego poziomu

Stronicowanie wstępne

- w stronicowaniu na żądanie występuje na początku działania procesu duża liczba braków stron (zanim nie zostanie załadowana do pamięci początkowa strefa procesu)
- stronicowanie wstępne (ang. *prepaging*) - jednorazowe wprowadzenie do pamięci stron, o których jest wiadome, że będą potrzebne
- wraz z procesem zapamiętuje się jego zbiór roboczy i odpowiednie strony są ładowane przed wznowieniem wykonania procesu (i załadowaniem go do pamięci)
- trzeba rozstrzygnąć, czy koszt stronicowania wstępnego jest mniejszy od obsługi odpowiednich braków stron

Stronicowanie wstępne - własności

- ładowane są strony inne niż wymagane wskutek błędu braku strony
- wykorzystuje własności większości pamięci drugiego rzędu (pomocniczych)
- jeśli strony procesu są przechowywane w postaci ciągłej w pamięci pomocniczej, jest bardziej wydajnym załadowanie większej liczby stron za jednym razem
- niewydajne, jeśli nie ma referencji do dodatkowych stron
- nie powinno być mylone z wymianą (ang. “*swapping*”)

Wpływ stronicowania na efektywność programu

operacje na tablicach wielowymiarowych;

MAX*sizeof(int) = rozmiar strony

I

```
int A[MAX][MAX];  
for(j=0;j<MAX;j++)  
    for(i=0; i< MAX; i++)  
        A[I][j]=0;
```

II

```
int A[MAX][MAX];  
for(i=0;i<MAX;i++)  
    for(j=0; j< MAX; j++)  
        A[I][j]=0;
```

tablica jest umieszczona w pamięci wierszami

wybór II ogranicza ilość zmian stron około ~ MAX razy

Powiązania stron z operacjami we/wy

- blokowanie (ang. locking) stron w pamięci
- stosowane np., gdy operacja wejścia-wyjścia odnosi się do pamięci wirtualnej procesu
- cel blokowania: uniknięcie sytuacji, że strona zawierająca bufor danych wymienianych przez urządzenie we/wy zostanie wymieniona, gdy proces oczekuje na przydział urządzenia wejścia/wyjścia

Sposoby rozwiązania powyższego problemu:

- zakaz wykonania operacji we/wy bezpośrednio do pamięci użytkownika - transfer jest dokonywany przez pamięć systemu, co jednak powoduje znaczne nakłady czasowe
- blokowanie stron w pamięci
 - każda ramka ma przyporządkowany bit blokowania, jeśli jest ustawiony, to dana ramka nie bierze udziału w wymianie
- blokowane mogą być strony, których nie chcemy odsyłać z pamięci
- blokada zmniejsza ilość ramek, które mogą być udostępnione innym stronom

Przetwarzanie w czasie rzeczywistym

- pamięć wirtualna nie sprzyja obliczeniom w czasie rzeczywistym, które powinny mieć minimalne opóźnienia
- systemy czasu rzeczywistego prawie nigdy nie mają pamięci wirtualnej
- systemy, które umożliwiają obliczenia z podziałem czasu i w czasie rzeczywistym
 - proces pracujący w czasie rzeczywistym informuje system o potrzebnych stronach, które zostają zablokowane w pamięci
 - groźba uniemożliwienia innym procesom wykonywania się

Przykład systemu: Windows NT

- Stosowane jest stronicowanie na żądanie z grupowaniem (ang. clustering)
- Grupowanie polega na obsłudze braków stron przez sprowadzanie nie tylko brakującej strony, ale także pewnej liczby stron z jej otoczenia
- Po utworzeniu procesu przypisuje mu się minimum i maksimum zbioru roboczego
 - Minimum zbioru roboczego (ang. working-set minimum) oznacza minimalną liczbę stron, których obecność w pamięci gwarantuje się procesowi
 - Jeśli pamięci jest pod dostatkiem, można przydzielić procesowi więcej stron – aż do wartości maksimum zbioru roboczego (ang. working-set maximum), które w pewnych sytuacjach może być przekroczone
- Zarządca pamięci wirtualnej otrzymuje wykaz wolnych ramek,
 - Z wykazem kojarzy się wartość progową określającą, czy wolnej pamięci jest dosyć, czy też nie
- Po wystąpieniu braku strony w procesie znajdującym się poniżej swego maksimum zbioru roboczego, zarządca pamięci wirtualnej przydziela stronę z wykazu wolnych ramek
- Jeśli proces osiągnął swoje maksimum i wystąpi w nim brak strony, to musi wybrać stronę do zastąpienia przy użyciu lokalnej polityki zastępowania stron
- Gdy ilość wolnej pamięci spada poniżej wartości progowej, zarządca pamięci wirtualnej stosuje tzw. automatyczne przycinanie zbioru roboczego (ang. automatic working-set trimming), aby przywrócić wartość ponad wartością progową
 - Automatyczne przycinanie zbioru roboczego dokonuje się przez określenie liczby stron przydzielonej procesom
 - Jeśli procesowi przydzielono więcej stron, to zarządca pamięci wirtualnej usuwa strony, aż proces osiągnie swoje minimum zbioru roboczego
 - Proces dysponujący tylko minimum zbioru roboczego może uzyskać strony z wykazu wolnych ramek, w miarę możliwości

Przykład systemu: Windows NT (2)

- Algorytm stosowany przy wyznaczaniu strony do usunięcia zależy od rodzaju procesora/ilości procesorów
 - Systemy jednoprocessorowe x86 – odmiana algorytmu zegarowego
 - Systemy wieloprocessorowe x86, procesor Alpha – odmiana algorytmu FIFO
 - Wyczyszczenie bitu odniesienia może wymagać unieważnienia wpisu w buforze translacji adresów stron (TLB) w innych procesorach, aby uniknąć tych komplikacji stosuje się FIFO

Przykład systemu: Solaris 2x (1)

- Jądro utrzymuje listę wolnych stron
- Z wykazem wolnych stron (ang. *free pages*) jest związany parametr **lotsfree** (miejsca pod dostatkim), reprezentujący próg rozpoczęcia stronicowania
 - **lotsfree** jest na ogół ustawione na $1/64$ rozmiaru pamięci fizycznej
 - Cztery razy na sekundę jądro sprawdza, czy ilość wolnej pamięci nie jest mniejsza niż **lotsfree**
- Gdy ilość wolnych ramek spadnie poniżej **lotsfree**, podejmuje działanie proces wymiatania stron (ang. *pageout*).
 - Proces wymiatania stron przypomina algorytm drugiej szansy, znany też jako algorytm zegara dwuwskazówkowego (ang. *two-handed-clock algorithm*)
 - Pierwsza wskazówka obiega wszystkie strony w pamięci, ustawiając ich bity odniesienia na 0
 - W późniejszym czasie, druga wskazówka zegara sprawdza bity odniesień do stron w pamięci i zwraca te strony, które mają ustawioną nadal wartość zero

Przykład systemu: Solaris 2x (2)

- W algorytmie wymiatania stron stosuje się kilka parametrów sterowania szybkością analizowania stron (ang. *scanrate*)
 - Parametry opisujące ilość wolnej pamięci: **minfree** < **desfree** < **lotsfree**
 - Szybkość analizowania jest wyrażana w stronach na sekundę i waha się
 - od **slowscan** (wolna analiza) – domyślna wielkość – 100 stron/sekundę
 - do **fastscan** (szybka analiza) – domyślna wielkość **TotalPhysicalPages/2** stron na sekundę, lecz nie więcej niż 8192 strony
 - Jeśli ilość wolnej pamięci spada poniżej **lotsfree**, analiza rozpoczyna się z prędkością **slowscan** stron na sekundę i zwiększa się do **fastscan** w zależności od ilości dostępnej pamięci
 - Odległość (w stronach) między wskazówkami zegara jest określona przez **handspread** (rozpiętość wskazówek). Ilość czasu między wyczyszczeniem bitu i zbadaniem jego stanu przez wskazówkę tylną zależy od **scanrate** i **handspread**
 - Jeśli **scanrate** wynosi 100 stron na sekundę, a **handspread** – 1024 strony, to między przejściami wskazówek upłynie 10 sekund
 - W rzeczywistości **scanrate** rzędu kilku tysięcy nie należy do rzadkości

Przykład systemu: Solaris 2x (3)

- Proces wymiatania stron sprawdza pamięć 4 razy na sekundę
- Jeśli jednak ilość wolnej pamięci spadnie poniżej **desfree** (pożądana ilość wolnej pamięci), to proces wymiatania stron będzie działał 100 razy na sekundę, w celu utrzymania przynajmniej **desfree** wolnej pamięci
- Jeśli nie uda się w ciągu 30 sekund uzyskać średniej ilości wolnej pamięci na poziomie **desfree**, jądro rozpoczyna proces wymiany procesów (poszukując procesów, które od dłuższego czasu były bezczynne)
- Jeśli system nie jest w stanie utrzymać wolnej pozycji na poziomie **minfree**, proces wymiatania stron jest wywoływany przy każdym zamówieniu strony