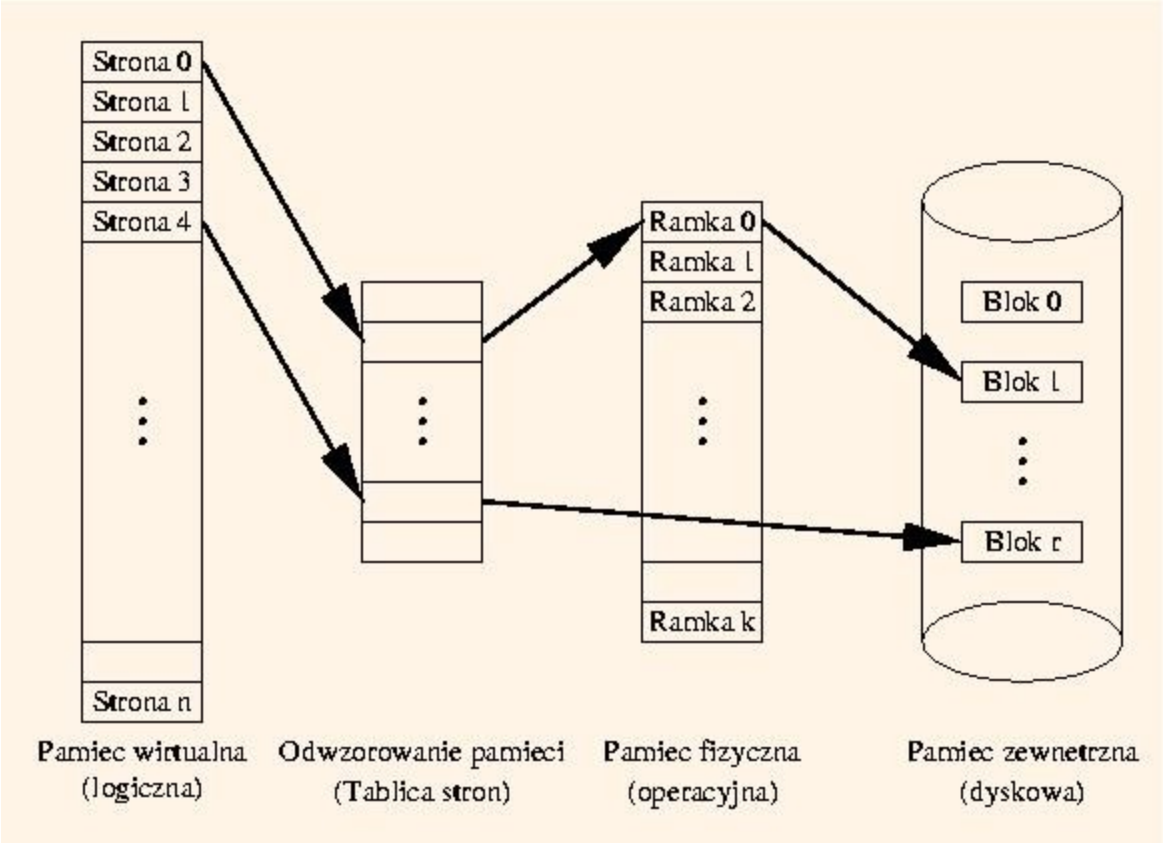
Pamięć wirtualna

Pamięć wirtualna oznacza oddzielenie pamięci fizycznej od pamięci logicznej dostępnej użytkownikowi. U podstaw tej idei leży obserwacja, że w każdej chwili tylko część pamięci procesu musi znajdować się w pamięci operacyjnej komputera -- ta część, która akurat jest używana przez proces. Jest to możliwe dzięki temu, że procesy charakteryzują się *lokalnością* -- w konkretnej chwili proces nie potrzebuje całej przydzielonej mu pamięci, a jedynie jej część. Pamięć logiczna może być więc większa niż fizyczna, wystarczy że nie używane w danej chwili obszary pamięci logicznej zostaną przeniesione na pamięć drugorzędną, np. na dysk. Taka pamięć drugorzędna używana do "udawania" pamięci operacyjnej jest nazywana obszarem wymiany. Adresy w pamięci wirtualnej są odwzorowywane na adresy pamięci fizycznej, jednak przestrzeń adresów logicznych może być większa niż przestrzeń adresów fizycznych. Konstrukcja MMU musi być rozszerzona tak, żeby było wiadomo, czy dany adres logiczny odpowiada adresowi fizycznemu i jakiemu, a jeżeli nie, to jakiemu miejscu w obszarze wymiany odpowiada.

Mechanizm pamięci wirtualnej stanowi rozszerzenie mechanizmów stronicowania lub segmentacji, omówionych w poprzednim rozdziale. Są to, odpowiednio, mechanizmy *stronicowania na żądanie* albo *segmentacji na żądanie*. W przypadku segmentacji na żądanie, tablica segmentów tłumaczy numer segmentu na spójny obszar w pamięci fizycznej lub w obszarze wymiany. W przypadku stronicowania na żądanie, tablica stron tłumaczy numery stron na numery ramek lub numery bloków dyskowych w obszarze wymiany. Stronicowanie na żądanie jest dużo bardziej elastyczne niż segmentacja na żądanie, dlatego też w praktyce wyparło segmentację na żądanie. W dalszej części wykładu ograniczymy się do stronicowania na żądanie.

Mechanizm pamięci wirtualnej umożliwia też współdzielenie przestrzeni adresowej przez wiele procesów, oraz pozwala efektywniej tworzyć procesy.



Na powyższym rysunku widać przykładowy stan pamięci wirtualnej systemu. Strona 0 w pamięci logicznej jest przyporządkowana (odwzorowana w) ramce nr 0 w pamięci fizycznej, która z kolei odpowiada blokowi 1 w pamięci dyskowej. Aktualny stan strony nr 0 jest zapisany w ramce 0. Jeśli ta strona nie była modyfikowana od czasu ostatniego ładowania z dysku, to zawartość bloku 1 i ramki 0 są takie same. W przeciwnym przypadku tylko w ramce 0 jest aktualny stan strony 0. Strona 4 nie jest natomiast odwzorowana w pamięci operacyjnej. Jej zawartość znajduje sie wyłącznie na dysku w bloku nr *r*. Jeśli jakiś proces chciałby skorzystać ze strony 4 musi być ona sprowadzona do pamięci operacyjnej, co wiąże się z zarezerwowaniem dla niej jakiejś pustej ramki.

Stronicowanie na żądanie

Stronicowanie na żądanie jest najczęstszym sposobem implementacji pamięci wirtualnej. W skrócie można go określić jako sprowadzanie strony do pamięci operacyjnej tylko wtedy, gdy jest ona potrzebna. Nazywa się to także *procedurą leniwej wymiany*. Takie rozwiązanie zmniejsza liczbę wykonywanych operacji wejścia/wyjścia i zapotrzebowanie na pamięć operacyjną, ponieważ nie sprowadza się niepotrzebnych stron (pojedyncze wywołanie dużego wielofunkcyjnego programu może wymagać sprowadzenia jedynie niewielkiej części jego kodu). Dzięki temu zmniejsza się czas reakcji systemu. Można też obsłużyć większą liczbę użytkowników.

Gdy proces odwołuje się do pamięci, mogą zajść trzy sytuacje:

- odwołanie jest niepoprawne,
- odwołanie jest poprawne i strona jest w pamięci, oraz
- odwołanie jest poprawne, ale strony nie ma w pamięci.

W pierwszym przypadku zlecenie jest odrzucane, w drugim jest po prostu obsługiwane, a w trzecim musi dodatkowo obejmować sprowadzanie żądanej strony z dysku do pamięci.

Bit poprawności

Rozpoznawanie stanu strony, do której nastąpiło odwołanie, musi być wspomagane sprzętowo. W przeciwnym przypadku nie będzie to mechanizm efektywny. Takim wsparciem może być np. *bit poprawności*, towarzyszący każdej pozycji w tablicy stron. Jeśli jest równy 1, strona jest w pamięci operacyjnej. Jeśli jest równy zero, strony nie ma w pamięci operacyjnej, albo taki adres strony jest niepoprawny. Początkowo wszystkie bity poprawności są równe 0. Jeśli w trakcie procesu tłumaczenia adresu, w tablicy stron znajdziemy bit poprawności ustawiony na 0, następuje przerwanie *braku strony*, które dalej jest szczegółowo obsługiwane.

Obsługa braku strony

Oto czynności wykonywane w trakcie obsługi braku strony:

1. Sprawdzenie wewnętrznej tablicy (zwykle znajduje się ona w bloku kontrolnym procesu), aby stwierdzić, czy odwołanie do pamięci było poprawne.
2. Znalezienie wolnej ramki w pamięci fizycznej.
3. Sprowadzanie brakującej strony z dysku.
4. Aktualizacja tablicy stron (wstawienie numeru ramki i ustawienie bitu poprawności na 1).
5. Wznowienie instrukcji przerwanej przez wystąpienie braku strony (teraz żądana strona jest już dostępna w pamięci).

Może się jednak tak zdarzyć, że nie ma wolnej ramki w pamięci. Należy wówczas znaleźć jakąś nieużywaną ramkę i przenieść ją na dysk (jeżeli oczywiście na dysku nie ma już jej wiernej kopii). Tym zajmuje się algorytm zastępowania stron. Należy go oczywiście tak opracować, żeby liczba zastępowanych stron była jak najmniejsza.

Sprawność stronicowania na żądanie

Oznaczmy przez *p* prawdopodobieństwo wystąpienia braku strony przy odwołaniu do pamięci. Za jego pomocą wyprowadzimy pewną miarę sprawności pamięci zwaną *efektywnym czasem dostępu* (ang. *effective access time* (*EAT*)) do pamięci stronicowanej. Będzie to średni czas wymagany na obsługę odwołania do pamięci:

EAT = (1 - *p*) * czas dostępu do pamięci zwykłej
+ *p* * (narzut związany z przerwaniem braku strony
+ [zapisanie na dysk strony ze zwalnianej ramki]
+ wczytanie z dysku żądanej strony
+ narzut związany ze wznowieniem procesu
)

Gdy strona jest w pamięci (prawdopodobieństwo 1 - *p*), bierzemy pod uwagę tylko koszt dostępu do pamięci. Gdy strony nie ma w pamięci (prawdopodobieństwo *p*), należy wliczyć koszt wszystkich czynności związanych z obsługą braku strony. Zapisanie na dysk strony ze zwalnianej ramki ma miejsce tylko wtedy, gdy jest to niezbędne, tzn. gdy tę stronę zmieniono od chwili jej załadowania do ramki.

Przykład:

Przyjmijmy następujące założenia:

1. Czas dostępu do konwencjonalnej pamięci to 2 nanosekundy.
2. Połowa z zastępowanych stron wymaga zapisu na dysk.
3. Czas dostępu do dysku to 5 milisekund (5 000 000 nanosekund).
4. Zaniedbujemy narzuty związane z obsługą przerwania i wznowieniem procesu (są one niewielkie w porównaniu z gigantycznym kosztem operacji dyskowej).

Wówczas:

EAT = (1 - *p*) * 2 +*p* *(50% * 5 000 000 + 5 000 000) ns =
= 2 + *p* * 7 499 998 ns

Widać więc, że różnica między EAT i czasem dostępu do pamięci fizycznej jest wprost proporcjonalna do prawdopodobieństwa wystąpienia przerwania braku strony. Co więcej, żeby EAT był akceptowalny, *p* musi być rzędu kilkudziesięciu miliardowych. (W miarę postępu technologii, dysproporcja między czasem dostępu do pamięci operacyjnej i do dysków rośnie. W rezultacie, wymagania wobec *p* również rosną w miarę upływu czasu.)

Oczywiście efektywność pamięci wirtualnej jest niższa niż pamięci fizycznej. Nie ma nic za darmo. Ograniczenie tego spowolnienia wymaga minimalizacji prawdopodobieństwa wystąpienia braku strony. Dziesięcioprocentowe spowolnienie nastąpi już wtedy, gdy jedno na 17 499 9980 odwołań do pamięci skończy się brakiem strony.

Zastępowanie stron

Gdy wystąpi brak strony a nie będzie już wolnych ramek, trzeba wybrać jedną z ramek i zastąpić obecną w niej stronę żądaną stroną. Dzięki zastępowaniu stron pamięć logiczna może być większa niż pamięć fizyczna.

Zastępowanie strony obejmuje następujące czynności:

1. Znalezienie położenia żądanej strony na dysku.
2. Znalezienie strony-ofiary, która ma być usunięta z pamięci.
3. Zapisanie ofiary na dysk -- o ile na dysku nie ma jej wiernej kopii.
4. Oznaczenie w tablicy stron odwołania do ofiary jako niepoprawnego.
5. Wczytanie żądanej strony do zwolnionej ramki.
6. Oznaczenie w tablicy stron odwołania żądanej strony jako poprawnego.

W p. 3 musimy umieć stwierdzić, czy strona-ofiara ma na dysku swoją wierną kopię. Inaczej mówiąc, czy od ostatniego sprowadzenia do pamięci była modyfikowana. Służy do tego specjalny sprzętowy bit związanego z każdą ramką, tzw. *bitu modyfikacji*. Po załadowaniu strony ten bit jest ustawiany na 0. Przy zapisie dowolnego bajtu w danej ramce, bit ów jest sprzętowo ustawiany na 1. Jeżeli bit modyfikacji strony-ofiary jest równy 0, to znaczy, że od czasu sprowadzenia do pamięci nie była ona modyfikowana i nie trzeba jej zapisywać na dysku. Ponownie warto zwrócić uwagę na sprzętowość tego rozwiązania. Użycie tu rozwiązania software'owego byłoby niedopuszczalnie powolne.

Z całego tego scenariusza najciekawszy jest punkt 2: znalezienie ofiary. Tą czynnością zajmują się *algorytmy zastępowania stron*. Jest to jedyne miejsce, gdzie wybór odpowiedniego algorytmu w systemie operacyjnym może wpłynąć na efektywność pamięci wirtualnej. Istnieje wiele takich algorytmów. Poniżej omówimy niektóre z nich i postaramy się je ocenić. Najważniejszym kryterium oceny będzie *liczba braków stron*. Im jest ona mniejsza, tym algorytm lepszy.

First-In-First-Out (FIFO)

Przy tym algorytmie ofiarą staje się strona, która najdłużej przebywa w pamięci. Inaczej mówiąc, strony znajdujące się w pamięci fizycznej tworzą kolejkę FIFO. Strony sprowadzane do pamięci są wstawiane na koniec kolejki, a strony-ofiary do usunięcia z pamięci są brane z początku kolejki. Algorytm FIFO w ogóle nie bierze pod uwagę właściwości obsługiwanych procesów i w zasadzie zastępuje strony na chybił-trafił. Zaletą jest prostota jego implementacji i niewielkie narzuty związane z jego obsługą.

Przyjrzyjmy się działaniu FIFO na przykładzie następującego ciągu odwołań do stron:

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

Oto zawartość ramek w poszczególnych chwilach, gdy mamy do dyspozycji 3 ramki (pogrubiony numer strony oznacza, że obsługa odwołania była związana z brakiem strony):

Chwila	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Odwołanie	1	2	3	4	1	2	5	1	2	3	4	5
Ramka 1	**1**	1	1	**4**	4	4	**5**	5	5	5	5	5
Ramka 2		**2**	2	2	**1**	1	1	1	1	**3**	3	3
Ramka 3			**3**	3	3	**2**	2	2	2	2	**4**	4

Łącznie oznacza to 9 braków stron. Przyjrzyjmy się teraz, co się stanie, gdy liczba dostępnych ramek wyniesie 4:

Chwila	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Odwołanie	1	2	3	4	1	2	5	1	2	3	4	5
Ramka 1	**1**	1	1	1	1	1	**5**	5	5	5	**4**	4
Ramka 2		**2**	2	2	2	2	2	**1**	1	1	1	**5**
Ramka 3			**3**	3	3	3	3	3	**2**	2	2	2
Ramka 4				**4**	4	4	4	4	4	**3**	3	3

10 braków stron! Zwiększenie pamięci spowolniło więc działanie systemu! Ten nieoczekiwany rezultat nosi nazwę *anomalii Beladiego*.

Least Recently Used (LRU)

Przy tym algorytmie ofiarą staje się strona, która nie była używana przez najdłuższy okres. Można powiedzieć, że jest to przybliżenie algorytmu optymalnego, które zamiast przyszłości bierze pod uwagę przeszłość. Taki algorytm da się już zaimplementować. Przyjrzyjmy się wykonaniu tego algorytmu dla poprzednio podanego ciągu odwołań i 4 ramek:

Chwila	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Odwołanie	1	2	3	4	1	2	5	1	2	3	4	5
Ramka 1	**1**	1	1	1	1	1	1	1	1	1	1	**5**
Ramka 2		**2**	2	2	2	2	2	2	2	2	2	2
Ramka 3			**3**	3	3	3	**5**	5	5	5	**4**	4
Ramka 4				**4**	4	4	4	4	4	**3**	3	3

Razem 8 braków strony. To istotnie lepiej niż FIFO z 10 brakami stron.

Implementacja LRU

Implementacja LRU nie jest łatwym zadaniem, ponieważ trzeba jakoś sprawić, żeby każde odwołanie do pamięci pozostawiało w systemie jakiś ślad niezbędny przy późniejszym wyznaczaniu ofiary. To oznacza, że bez specjalnych udogodnień sprzętowych się nie obejdzie. Realizacja za pomocą oprogramowania byłaby zbyt wolna.

LRU można zaimplementować za pomocą **znaczników czasu** (ang. *time-stamps*). Każda strona ma wówczas specjalne pole znacznika. Gdy jakiś proces odwołuje się do strony, do tego licznika kopiuje się stan zegara systemowego. Gdy trzeba wybrać ofiarę, szuka się strony z najmniejszą wartością znacznika. Oczywiście ustawianie znacznika strony musi odbywać się sprzętowo, inaczej jest nieefektywne.

Inna możliwość to wykorzystanie **stosu**. Na stosie trzymamy numery stron, do których były odwołania, w kolejności czasu ostatniego odwołania. Odwołanie do strony powoduje przesunięcie jej numeru na wierzchołek stosu. W ten sposób strona, która jest najdłużej nieużywana znajduje się na spodzie stosu. Gdy stos ma postać listy dwukierunkowej, przeniesienie na wierzchołek oznacza konieczność zmiany 6 wskaźników. To powoduje, że taka implementacja może być efektywna tylko przy zastosowaniu oprogramowania systemowego albo mikroprogramów.

Przybliżenia LRU

Jak widać, niestety implementacja LRU wymaga bardzo wymyślnych udogodnień sprzętowych, które istnieją w niewielu systemach. Dlatego zwykle przybliża się algorytm LRU za pomocą jakiegoś innego algorytmu. Zwykle wymaga to również specjalnych udogodnień sprzętowych, ale już nie tak wymyślnych. Jedno z takich przybliżeń polega na skojarzeniu z każdą ramką *bitu odwołania*. Wstępnie wszystkie bity odwołania są równe 0. Przy każdym odwołaniu do strony odpowiadający jej bit jest (sprzętowo) ustawiany na 1. Algorytmy przybliżający LRU jako ofiarę wybierają jedną ze stron, których bit odwołania jest równy 0. Jeden z takich algorytmów opisano poniżej.Oczywiście pozostaje jeszcze kwestia kiedy bity odwołania są ustawiane na 0.

Algorytm drugiej szansy

Algorytm drugiej szansy przypomina algorytm FIFO. Wszystkie strony znajdujące się w pamięci fizycznej tworzą kolejkę FIFO. Strony są sprowadzane do pamięci na żądanie, czyli na skutek odwołania do nich. Tak więc, gdy wstawiamy stronę na koniec kolejki FIFO, to jej bit odwołania jest ustawiany na 1. Gdy chcemy wybrać stronę-ofiarę, to kandydatem jest strona na początku kolejki FIFO. Możliwe są dwa przypadki:

- Jeśli bit odwołania tej strony jest równy 0, to faktycznie staje się ona ofiarą.
- Jeśli jej bit odwołania jest równy 1, to dostaje ona "drugą szansę":
    1. Jej bit odwołania jest ustawiany na 0.
    2. Strona pozostaje w pamięci.
    3. Jest ona przenoszona na koniec kolejki FIFO.
    4. Kandydatką na ofiarę jest następna strona w kolejce (ta strona również może dostać drugą szansę itd.).

Oto przykład działania algorytmu drugiej szansy. W dolnej części tabelki podano zawartość kolejki FIFO w kolejnych momentach. Bity odwołania zaznaczono jako indeksy.

Chwila	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Odwołanie	1	2	3	4	5	3	1	5	2
Ramka 1	$\mathbf{1_1}$	1_1	1_1	1_1	$\mathbf{5_1}$	5_1	5_1	5_1	5_1
Ramka 2		$\mathbf{2_1}$	2_1	2_1	2_0	2_0	$\mathbf{1_1}$	1_1	1_1
Ramka 3			$\mathbf{3_1}$	3_1	3_0	3_1	3_1	3_1	3_0
Ramka 4				$\mathbf{4_1}$	4_0	4_0	4_0	4_0	$\mathbf{2_1}$
FIFO	1	1 2	1 2 3	1 2 3 4	2 3 4 5	2 3 4 5	3 4 5 1	3 4 5 1	5 1 3 2

Algorytmy zliczające

Algorytmy zliczające korzystają ze związanego z każdą stroną licznika odwołań. Oczywiście taki licznik musi być zrealizowany sprzętowo, ponieważ każde odwołanie danej strony powoduje zwiększenie jej licznika o jeden. Można wskazać dwa takie algorytmy oparte na przeciwstawnych założeniach. Oba algorytmy nie są dość popularne, ponieważ ich implementacja jest dość kłopotliwa, a obu daleko do algorytmu optymalnego.

Least Frequently Used (LFU)

Przy tym algorytmie ofiarą staje się strona, do której było najmniej odwołań. Zakłada się tu, że strony o dużych licznikach odwołań są aktywnie użytkowane i nie należy ich usuwać z pamięci.

Most Frequently Used (MFU)

Przy tym algorytmie ofiarą staje się strona, do której było najwięcej odwołań. Zakłada się tu, że strony o małych licznikach odwołań zostały właśnie załadowane i wkrótce będą jeszcze używane.

Algorytm optymalny

Przy tym algorytmie ofiarą staje się strona, która przez najdłuższy okres będzie nieużywana. Przyjrzyjmy się wykonaniu tego algorytmu dla poprzednio podanego ciągu odwołań i 4 ramek:

Chwila	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Odwołanie	1	2	3	4	1	2	5	1	2	3	4	5
Ramka 1	**1**	1	1	1	1	1	1	1	1	1	**4**	4
Ramka 2		**2**	2	2	2	2	2	2	2	2	2	2
Ramka 3			**3**	3	3	3	3	3	3	3	3	3
Ramka 4				**4**	4	4	**5**	5	5	5	5	5

Tylko 6 braków strony. Fenomenalny wynik! Tylko jak zaimplementować tę strategię? Nijak! Nie da się przecież przewidzieć przyszłych odwołań do pamięci. To jest jedynie algorytm teoretyczny, który służy do oceny jakości innych algorytmów. Lepiej niż algorytm optymalny się nie da. Można też próbować go przybliżać.

Przydział ramek

W systemie wieloprogramowym ramki są współdzielone przez kilka procesów. Co więcej, procesy rywalizują o te ramki. Powstaje więc problem, ile ramek powinny dostać poszczególne procesy. Często do działania procesu jest potrzebna pewna **minimalna** liczba ramek (rzędu kilka), choćby po to, żeby mógł on wykonać pewne instrukcje maszynowe odwołujące się równocześnie do kilku różnych miejsc w pamięci. Istnieją dwa zasadnicze schematy takiego przydziału: *przydział lokalny* i *przydział globalny*.

Przydział lokalny i globalny

Przydział lokalny oznacza, że każdy proces ma przydzieloną określoną pulę ramek i zastępowana ramka jest wybierana właśnie z tej puli. Natomiast przydział globalny polega na wyborze ofiary spośród wszystkich ramek w systemie (również tych należących do innych procesów). W przypadku przydziału globalnego proces nie panuje nad zestawem posiadanych ramek a jego efektywność zależy od braków strony powodowanych przez inne procesy. Z drugiej strony przydział lokalny może hamować procesy potrzebujące w danej chwili więcej pamięci, mimo że pamięć lokalnie przydzielona innym procesom nie jest im wcale potrzebna. Właśnie ze względu na większą przepustowość systemu najczęściej stosuje się przydział globalny.

Przydział stały

Przydział stały jest rodzajem kompromisu między przydziałem lokalnym i globalnym. Każdy proces ma na stałe przydzieloną pewną liczbę ramek. O pozostałe nie przydzielone ramki procesy mogą dowolnie rywalizować. Na stałe przydzielona procesowi liczba ramek powinna umożliwić mu normalne działanie, bez względu na braki stron występujące w innych procesach. Z drugiej strony, rywalizowanie procesów o pozostałe ramki zapewnia efektywne działanie systemu. Przydział stały może być *równy* (każdy proces dostaje tyle samo ramek) albo *proporcjonalny* (każdy proces dostaje liczbę ramek proporcjonalną do swojego rozmiaru). Przy zastosowaniu przydziału stałego może się okazać, że ramki są w posiadaniu procesu, który wcale ich tyle nie potrzebuje, dlatego lepszym rozwiązaniem wydaje się przydział proporcjonalny.

Przydział priorytetowy

W wypadku przydziału priorytetowego każdy proces ma przypisany priorytet (statycznie albo dynamicznie). Jest to inna forma kompromisu pomiędzy przydziałem lokalnym i globalnym. Gdy proces spowoduje brak strony, do zastąpienia jest wybierana ramka ze zbioru złożonego z ramek tego procesu i ramek procesów o niższym priorytecie. To oczywiście oznacza pogorszenie warunków działania procesów o niższym priorytecie, ale na tym właśnie polega priorytetowość. Nadal braki stron występujące w innych procesach mogą wpływać na działanie danego procesu (jak w przydziale globalnym), ale dotyczy to jedynie procesów o wyższym priorytecie.

Zliczanie częstości braków stron

Strategia priorytetowa może też polegać na obliczaniu częstości braków stron i przydzielaniu dodatkowej ramki procesowi, u którego częstość braków stron jest wyższa niż jakaś ustalona zawczasu akceptowalna wielkość. Dzieje się też odwrotnie. Jeśli częstość braków stron jest niższa niż jakaś ustalona wcześniej wielkość (być może inna), to proces traci jedną ze swoich ramek. Można więc powiedzieć, że częstość braków stron wyznacza priorytet procesu. Taka strategia pozwala dość skutecznie zapobiegać szamotaniu, o którym powiemy sobie za chwilę.

Szamotanie

Gdy proces ma mniej ramek niż liczba aktywnie używanych stron, dochodzi do *szamotania*. Ramek jest mniej niż trzeba, więc co chwilę należy sprowadzić nową stronę usuwając jedną z załadowanych stron. Ta usunięta strona jest za chwilę potrzebna i znów trzeba ją sprowadzić usuwając jakąś, która zaraz będzie znów potrzebna itd. System zaczyna się zajmować jedynie wymianą stron.

To jednak nie wszystko. Taki scenariusz prowadzi do niskiego wykorzystania procesora, więc planista długoterminowy może stwierdzić, że system nie jest dobrze wykorzystywany. Dodaje więc nowy proces do systemu zwiększając stopień wieloprogramowości, żeby poprawić parametry wykorzystania procesora. To oczywiście pogarsza tylko sytuację. Wydajność systemu może spaść tak bardzo, że wszystkie procesy wydają się być "zawieszone", a komputer jedynie "rzęzi" dyskiem.

Zastanówmy się więc, dlaczego dochodzi do szamotania. Zwykle proces korzysta tylko z niewielkiej części swojej pamięci. Ujęto to w tzw. *modelu strefowym*: proces w trakcie wykonania przechodzi z jednej strefy programu do innej. *Strefa* to zbiór stron używanych jednocześnie. Oczywiście strefy mogą na siebie nachodzić. W tym modelu odpowiedź na postawione wcześniej pytanie jest łatwa. Do szamotania dochodzi, gdy jakaś strefa jest większa niż przydzielona procesowi pula ramek.

Zbiór roboczy

Jedną z metod zapobiegania szamotaniu jest wyznaczanie tzw. *zbiorów roboczych* procesów. Zbiór roboczy, to zbiór stron, do których nastąpiło odwołanie w ciągu ostatnich I instrukcji (dla pewnej ustalonej stałej I, np. $I = 100\,000$). Zbiór roboczy ma przybliżać strefę, w której znajduje się program. Należy dobrze dobrać parametr I, ponieważ jego zbyt mała wartość uniemożliwi uchwycenie całej strefy, a zbyt duża spowoduje, że łączy się rozmiary kilku stref. Znając wielkości zbiorów roboczych, możemy przydzielić procesom pamięć proporcjonalnie do wielkości ich zbiorów roboczych. Jeśli suma rozmiarów zbiorów roboczych wszystkich procesów jest większa niż rozmiar dostępnej pamięci, prawdopodobnie mamy do czynienia z szamotaniem. Należy wówczas wstrzymać jeden z procesów, aby nie pogarszać sytuacji (robimy więc zupełnie coś odwrotnego niż planista w poprzednim punkcie). Nie bez znaczenia jest tu kwestia implementacji, ponieważ nie da się wyznaczać zbioru roboczego przy każdym odwołaniu do pamięci. Z pomocą ponownie przychodzi bit odwołania. Dla każdej strony procesu utrzymujemy bit odwołania. Co I instrukcji następuje

zliczenie podniesionych bitów odwołania we wszystkich ramkach procesu i ich wyzerowanie. Liczba podniesionych bitów jest przybliżeniem rozmiaru zbioru roboczego.

Słownik

algorytm drugiej szansy
: Przybliżenie LRU, w którym wykorzystuje się bit odwołania.

anomalia Beladiego
: Zjawisko polegające na tym, że przy zwiększeniu liczby dostępnych ramek pamięci fizycznej, liczba braków stron, zamiast maleć, rośnie.

bit modyfikacji
: Bit związany z ramką pamięci. Jest równy 1 wtedy i tylko wtedy, gdy ta ramka była modyfikowana od chwili załadowania do niej ostatniej strony. Jest to udogodnienie sprzętowe wykorzystywane przy implementacji pamięci wirtualnej.

bit odwołania
: Bit związany z ramką pamięci. Jest równy 1 wtedy i tylko wtedy, gdy do tej ramki było odwołanie od chwili załadowania do niej ostatniej strony. Jest to udogodnienie sprzętowe wykorzystywane przy implementacji pamięci wirtualnej.

bit poprawności
: Bit związany z stroną pamięci. Jest równy 1 wtedy i tylko wtedy, gdy ta strona jest w pamięci operacyjnej. Jest to udogodnienie sprzętowe wykorzystywane przy implementacji pamięci wirtualnej.

brak strony
: Zdarzenie zachodzące w chwili odwołania do strony, której obecnie nie ma w pamięci operacyjnej.

FIFO (zastępowanie FIFO)
: Algorytm zastępowania stron, przy którym ofiarą staje się strona, która najdłużej przebywa w pamięci.

LFU (Least Frequently Used)
: Algorytm zastępowania stron, przy którym ofiarą staje się strona, do której było najmniej odwołań.

lokalność
: Cecha procesów, polegająca na tym, że w danej chwili nie korzystają one z całej przydzielonej im pamięci, a jedynie z jej części.

LRU (Least Recently Used)
: Algorytm zastępowania stron, przy którym ofiarą staje się strona, która nie była używana przez najdłuższy okres.

MFU (Most Frequently Used)
: Algorytm zastępowania stron, przy którym ofiarą staje się strona, do której było najwięcej odwołań.

obszar wymiany
: Obszar pamięci drugorzędnej używany do przechowywania chwilowo nie używanych fragmentów pamięci logicznej.

szamotanie
: Sytuacja, w której proces ma mniej ramek niż liczba aktywnie używanych stron i musi co chwilę sprowadzać jedną ze stron usuwając inną, która za chwilę będzie niezbędna.

pamięć wirtualna
: Technika umożliwiająca wykonanie zadania nie mieszczącego się pamięci w całości.

segmentacja na żądanie
: Metoda implementacji pamięci wirtualnej, która przewiduje sprowadzenie segmentu z pamięci pomocniczej tylko wtedy, gdy jest on potrzebny.

stronicowanie na żądanie

Metoda implementacji pamięci wirtualnej, która przewiduje sprowadzenie strony z pamięci pomocniczej tylko wtedy, gdy jest ona potrzebna.

zastępowanie optymalne

Teoretyczny algorytm zastępowania stron, przy którym ofiarą staje się strona, która będzie nieużywana przez najdłuższy okres.

zastępowanie stron

Czynność systemu operacyjnego polegająca na wyborze jednej z zajętych ramek, opróżnieniu jej i załadowaniu do niej żądanej strony.

zbiór roboczy

Zbiór stron procesu, do których nastąpiło odwołanie w ciągu ustalonej liczby ostatnich instrukcji.