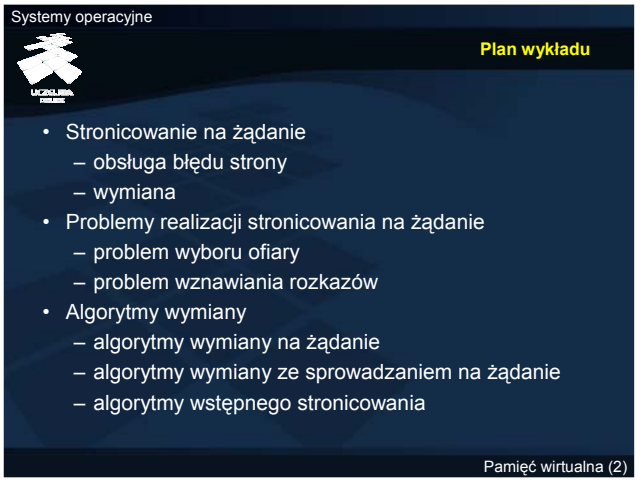
Pamięć wirtualna jest organizacją zasobów pamięci, zrealizowaną w oparciu o tzw. przestrzeń wymiany w pamięci drugiego rzędu (na dysku). Pamięć operacyjna (fizyczna) jest dla tych zasobów tylko pewnym oknem, przechowującym część zawartości na potrzeby bieżącego przetwarzania.

Stosowanie pamięci wirtualnej ma wiele zalet nie tylko związanych z możliwością powiększenia zasobów pamięci ponad dostępną pamięć fizyczną. Umożliwia bardziej racjonalne wykorzystanie pamięci operacyjnej, gdyż programy tworzone są często z nadmiarem w stosunku do typowych potrzeb. Na przykład rozmiary tablic statycznych ustala się z nadmiarem w stosunku do typowych potrzeb, w kodzie uwzględnia się obsługę sytuacji wyjątkowych do których może nigdy nie dojść. Ten nadmiar nie musi być ładowany do pamięci. Zastosowanie pamięci wirtualnej może też zmniejszyć czas odpowiedzi, gdyż skraca czas ładowania kodu, który często odwzorowywany jest w przestrzeń adresową procesu bezpośrednio z pliku i sprowadzany w niewielkich porcjach na żądanie.

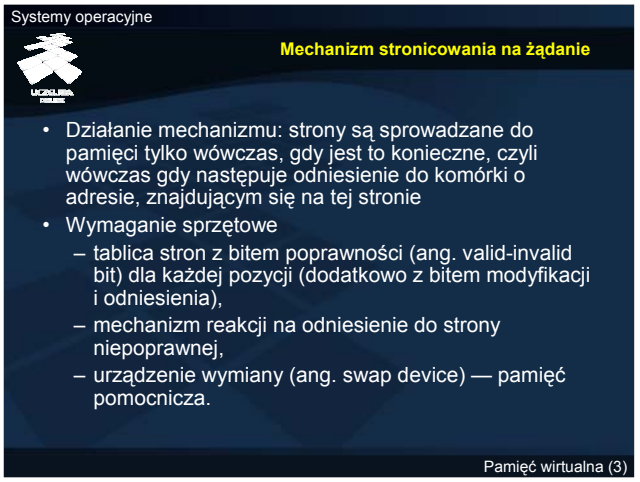
Celem wykładu jest przedstawienie zasady działania i problemów realizacji pamięci wirtualnej oraz omówienie algorytmów wymiany stron pomiędzy pamięcią operacyjną a pamięcią drugiego rzędu (obszarem wymiany).



Podstawą funkcjonowania pamięci wirtualnej jest mechanizm stronicowania na żądanie, od omówienia którego rozpoczyna się wykład. Działanie mechanizmu oparte jest na stronicowaniu i polega na sprowadzaniu do pamięci operacyjnej stron adresowanych przez procesor. Sprowadzenie strony jest zadaniem systemu operacyjnego i realizowane jest w przypadku wystąpienia błędu strony. Ponieważ pamięć operacyjna jest na ogół mniejsza od pojemności zasobów pamięci wirtualnej, sprowadzenie może wymagać usunięcia innej strony — dochodzi wówczas do wymiany. Realizacja pamięci wirtualnej oprócz wsparcia sprzętowego wymaga rozwiązania dwóch zasadniczych problemów na poziomie systemu operacyjnego:

- problemu wyboru ramki do wymiany strony, jeśli zajdzie potrzeba wymiany,
- problemu wznawiania rozkazów, którego rozwiązanie sprowadza się do zapewnienia dostępności odpowiednio dużej liczby ramek dla procesu.

Rozwiązanie problem wyboru ofiary bazuje na przesłankach o charakterze losowym i związane jest ściśle z algorytmem wymiany. Klasyfikacja i omówienie algorytmów wymiany stanowią ostatnią część wykładu.

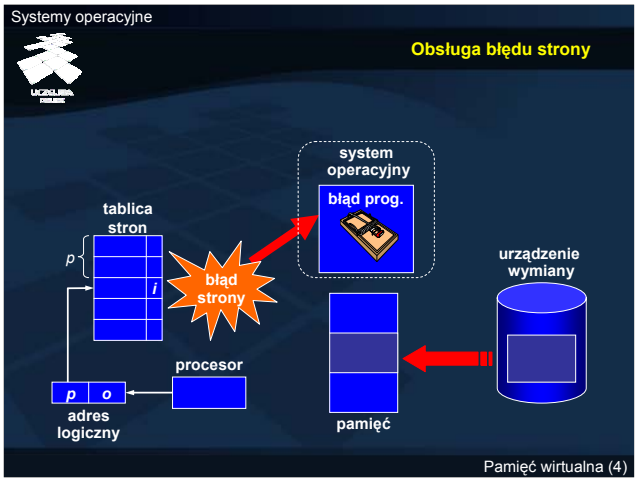


Stronicowanie na żądanie związane jest przede wszystkim z wymianą stron pomiędzy pamięcią pierwszego rzędu (operacyjną, fizyczną) i drugiego rzędu (masową, dyskową). Dzięki wykorzystaniu pamięci masowej można rozszerzyć wirtualną przestrzeń adresową i tym samym zwiększyć stopień wieloprogramowości lub uruchamiać zadania, których rozmiar wykracza poza dostępną pamięć operacyjną. Kosztem wprowadzenia takiego mechanizmu jest złożoność zarządzania pamięcią i narzut czasowy związany z dostępem, wynikający z wykorzystania stosunkowo wolnej pamięci dyskowej.

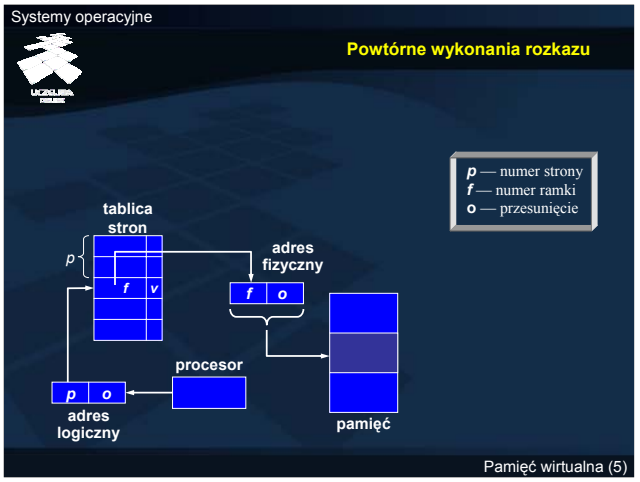
Adresowanie stron odbywa się tak samo, jak w prostym stronicowaniu, omówionym w poprzednim module. W tablicy stron przechowywany jest jednak bit poprawności, informujący o stanie strony. Strona poprawna (ang. valid) to taka, która zlokalizowana jest w pamięci operacyjnej. Odniesienie do takiej strony nie wymaga jej sprowadzania. Jeśli zgodnie z wartością bitu poprawności strona jest uznana za niepoprawną, występuje błąd braku strony i zgłaszane jest odpowiednie przerwanie diagnostyczne. W ramach obsługi błędu strony następuje sprowadzenie strony z obszaru wymiany (ang. swap space), umieszczenie w wolnej ramce i ponowne wykonanie rozkazu. Takie działanie nazywa się *leniwą wymianą* (ang. lazy swapping).

Urządzeniem wymiany jest najczęściej dysk, na którym znajduje się plik wymiany lub specjalnie wyodrębniona strefa (tzw. partycja wymiany).

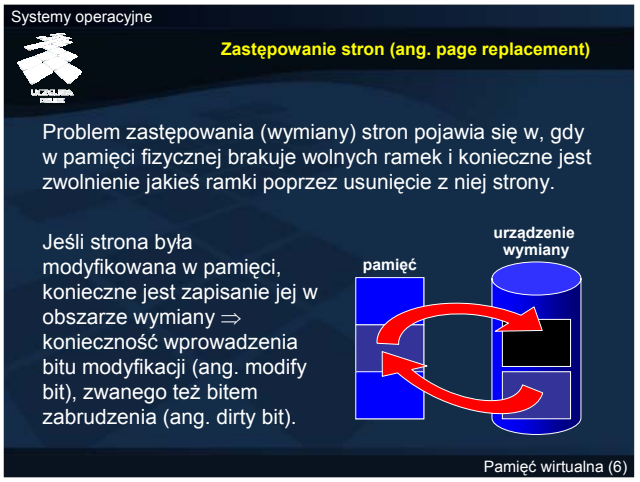
W dalszej części wykładu zamiennie będzie używane sformułowanie: *adresowanie strony* i *odniesienie do strony*, oznaczające wystawienie przez procesor adresu logicznego komórki, zlokalizowanej na tej stronie.



Zgodnie z zasadą transformacji adresu w pamięci stronicowanej, w celu zamiany numeru strony na numer ramki lokalizowany jest odpowiedni wpis w tablicy stron. Wpis może być jednak w danej chwili niepoprawny (ang. invalid), gdyż strona mogła zostać usunięta z pamięci — zastąpiona inną stroną. W przypadku odwołania się do strony, opisanej jako niepoprawna, następuje zgłoszenie błędu (przerwania diagnostycznego), które obsługuje system operacyjny. W ramach obsługi adresowana strona sprowadzana jest z obszaru wymiany do pamięci operacyjnej.



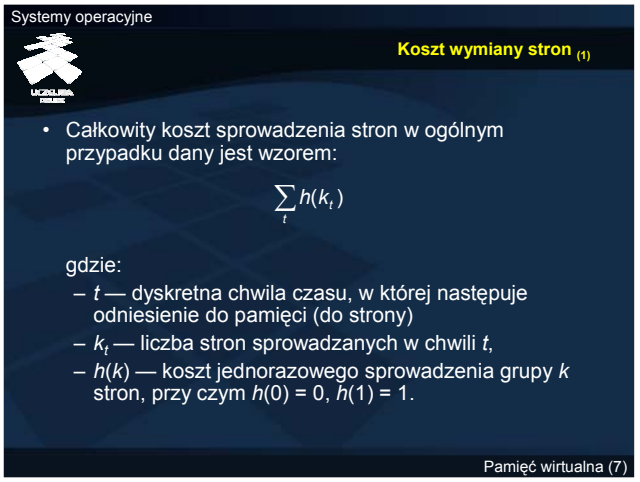
Po sprowadzeniu strony do pamięci operacyjnej odpowiedni bit poprawności w tablicy stron jest ustawiany. Przyjmijmy, że ustawienie bitu poprawności oznacza, że strona jest poprawna (ang. valid). Następnie rozkaz, który spowodował błąd strony, wykonywany jest ponownie przez procesor. To odniesienie do pamięci, które spowodowało wcześniej błąd strony powinno teraz wykonać się poprawnie.



Ramkę, która jest użyta do wymiany określa się jako *ramkę ofiarę* (ang. victim frame), chociaż to raczej strona jest ofiarą. W dalszej części używane będą zależnie od kontekstu terminy: *ramka ofiara*, *strona ofiara*, lub po prostu *ofiara*.

Brak bitu modyfikacji w tablicy stron oznacz konieczność zapisania strony na urządzeniu wymiany, jeśli strona to mogła być potencjalnie modyfikowana. Jeśli bit modyfikacji istnieje, ustawiany jest przed jednostkę zarządzania pamięcią zawsze, gdy wystąpił cykl maszynowy zapisu na tej stronie, nawet jeśli zapis niczego w stanie strony nie zmienił (np. nastąpiło wpisanie do komórki pamięci takiej samej wartości, jaka była tam wcześniej).

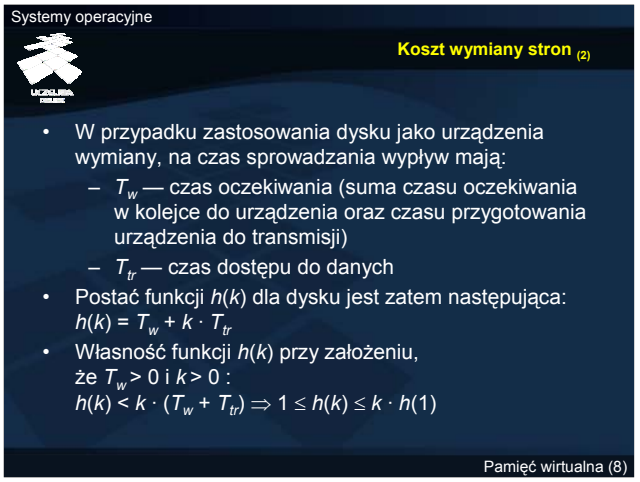
Jeśli bit modyfikacji nie jest ustawiony, to znaczy, że w obszarze wymiany jest aktualna kopia strony, która znajduje się również w pamięci. Zastąpienie takiej strony sprowadza się do jej nadpisania w pamięci. Przy ustawionym bicie modyfikacji strona zastępowana musi być najpierw zapisana w obszarze wymiany, co istotnie zwiększa koszt operacji wymiany.



Z punktu widzenia efektywności przetwarzania koszt wymiany można utożsamiać z czasem realizacji wymiany. W celu uproszczenia analizy przyjmuje, że koszt usunięcia strony stanowi stałą część kosztu jej sprowadzenia. Koszt wynika zatem z czasu sprowadzania, który w ogólności zależy od:

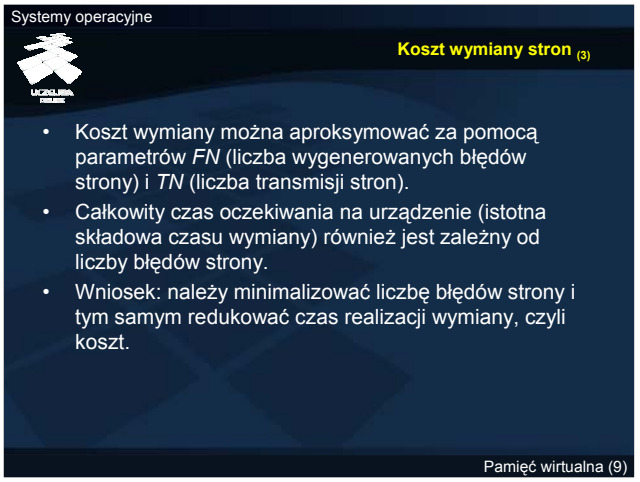
- ciągu odniesień,
- liczby dostępnych ramek,
- algorytmu wymiany.

Przyjmując funkcję kosztu sprowadzenia zbioru stron — h, koszt realizacji wymiany jest po prostu sumą kosztów w chwilach odniesienie do pamięci. Jeśli żadna strona nie jest sprowadzana, koszt wynosi 0. Jeśli sprowadzana jest jedna strona, koszt wynosi 1.



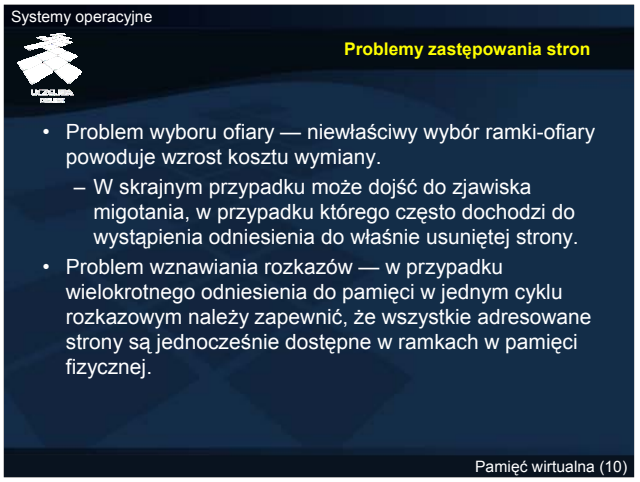
W przypadku dysku żądanie odczytu bloków czeka w kolejce, aż urządzenie dyskowe będzie wolne, czyli skończy obsługę wcześniej zgłoszonych żądań. Czas oczekiwania nie zależy od wielkości zgłaszanego żądania, ale wielkość ta ma wpływ na łączny czas dostępu, na który składa się czas wyszukiwania ścieżki, opóźnienie obrotowe, czas przesyłania. Czas dostępu dotyczy każdego sektora, można jedynie minimalizować czas wyszukiwania i opóźnienie obrotowe, gdyż sektory z zawartością strony są lokalizowane blisko siebie.

Jak wynika z własności funkcji kosztu dla dysku, sprowadzenie kilku stron w wyniku realizacji **jednego żądania** jest bardziej kosztowne niż sprowadzenie jednej strony, ale mniej kosztowne niż sprowadzanie poszczególnych stron w wyniku osobnych żądań.



Konieczność sprowadzenia strony pojawia się dopiero przy wystąpieniu błędu strony. Sprowadzanie stron w okolicznościach innych niż obsługa błędu strony nie ma sensu, ale jak wynika z zaprezentowanej analizy, w ramach obsługi błędu strony sensowne może być sprowadzenie kilku stron, przewidując przeszłe odniesienia. Koszt sprowadzenie jest więc ściśle uzależniony od liczby błędów strony i liczby transmisji stron.

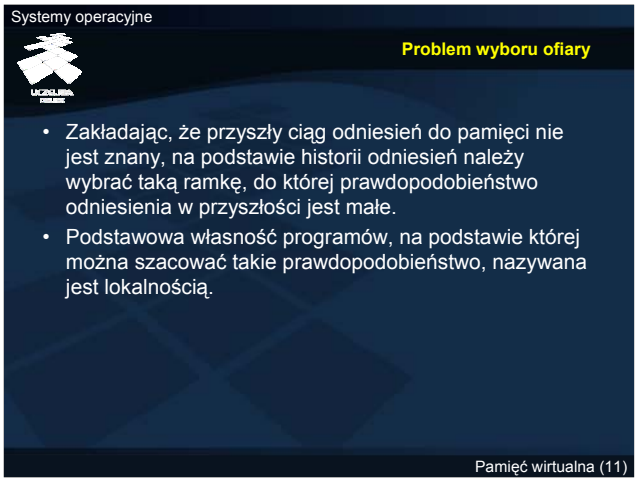
Liczba błędów strony wpływa również na długość kolejki żądań dostępu do dysku, co wydłuża czas oczekiwania. Można zatem wyciągnąć ogólny wniosek, że w celu redukcji kosztów wymiany należy minimalizować liczbę błędów strony, gdyż wpływa ona pośrednio lub bezpośrednio na wszystkie składowe kosztu.



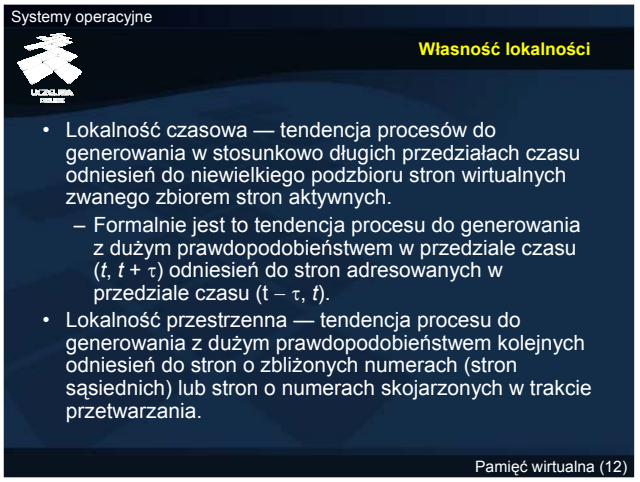
Zasadnicze problemy wymiany stron dotyczą decyzji odnośnie usuwanej strony oraz zagwarantowania dostępności wszystkich stron wymaganych do realizacji cyklu rozkazowego.

Usunięcie z pamięci strony, która będzie potrzebna przyszłości, oznacza konieczność ponownego sprowadzenia jej do pamięci, a więc koszt, którego być może dałoby się uniknąć. Nasilenie tego zjawiska określane jest jako *migotanie* i oznacza drastyczny spadek efektywności działania systemu komputerowego, gdyż większość czasu cyklu przetwarzania proces spędza w stanie oczekiwania na gotowość urządzenia dyskowego. Rozwiązanie problemu wyboru ofiary oparte jest na przesłankach o charakterze losowym.

Problem wznawiania rozkazów wiąże się z koniecznością ponownego wykonania całego cyklu rozkazowego, w którym wystąpił błąd strony. Cykl rozkazowy może składać się z kilku cykli maszynowych, z których każdy może się wiązać z odniesieniem do innej strony pamięci. W niesprzyjających okolicznościach mogłoby dojść do sytuacji, w której w wyniku obsługi błędu strony następuje usunięcie z pamięci innej strony, adresowanej w tym samym cyklu rozkazowym. Wznowienie rozkazu ponownie zakończy się błędem strony w jednym z cykli maszynowych. W skrajnym przypadku rozkaz mógłby się nigdy nie wykonać. Problem wznawiania rozkazów można rozwiązać, zapewniając procesowi pewną minimalną liczbę ramek — nie mniejszą niż liczba różnych adresów, wystawianych na magistrali w jednym cyklu rozkazowym.

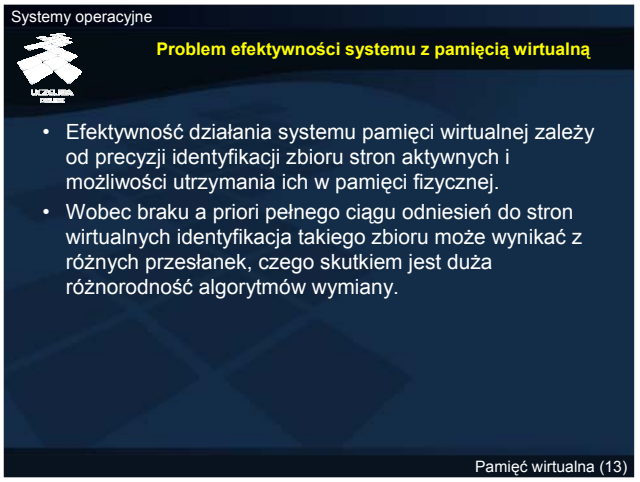


Problem z wyborem ofiary wynika z faktu, że nie znamy przyszłego ciągu odniesień do stron. Przyszłe odniesienia możemy jedynie przewidywać z pewnym prawdopodobieństwem na podstawie różnych przesłanek z odniesień w przeszłości. Większość typowych programów charakteryzuje się własnością lokalności. Często również optymalizatory kodu dokonują takiego rozmieszczenia obiektów, aby w trakcie wykonywania ujawniała się taka własność.



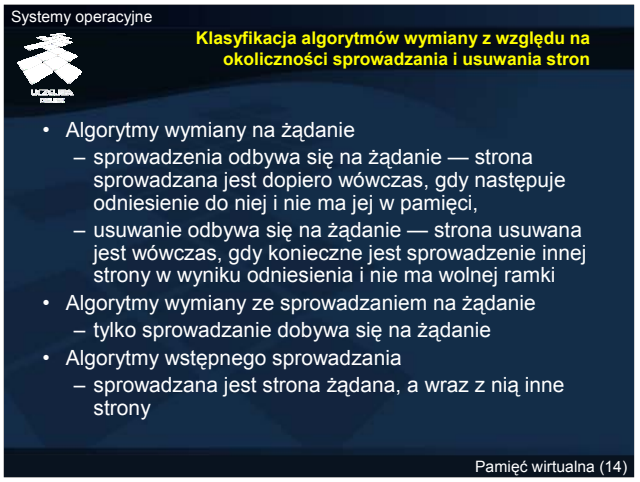
Lokalność można rozważać w dwóch aspektach: czasowym i przestrzennym. Lokalność czasową można sprowadzić do wielokrotnego odwołania w krótkim czasie do tej samej strony. Przypadek taki ma miejsce przy wykonywaniu programu, chyba że występują częste i dość dalekie (wychodzące poza stronę) skoki. Przypadek taki dotyczy również danych, np. tablic, a nawet pojedynczych zmiennych, gdyż programiści definiują często kilka zmiennych na potrzeby obsługi jakiegoś krótkiego fragmentu programu, np. pętli. Z punktu widzenia stronicowania oznacza to, że jeśli strona zostanie sprowadzona do pamięci operacyjnej, przyda się wielokrotnie w czasie wykonywania określonego fragmentu programu. Własność lokalności czasowej wykorzystywana jest często przy podejmowaniu decyzji o usunięci strony z pamięci.

Lokalność przestrzenna sprowadza się do wnioskowania na podstawie odwołań do pewnych stron o przyszłym odwołaniu do innych stron. Podstawą takiego skojarzenia może być numeracja stron. Jeśli program lub większa struktura danych (tablica) zajmuje kilka kolejnych stron, to prawdopodobne jest, że po odniesieniu do pierwszej z tych stron nastąpi odniesienie do drugiej, a później do trzeciej. Skojarzenie takie można również oprzeć o obserwację wcześniejszych odniesień. Jeśli strona p_2 często była adresowana zaraz po stronie p_1, to być może na stronie p_1 jest skok do instrukcji, znajdującej się na stronie p_2. Ten rodzaj lokalności można wykorzystać przy podejmowaniu decyzji o usuwaniu, ale również przy sprowadzaniu stron, kiedy stosowane jest tzw. sprowadzanie wstępne (sprowadzanie kilku stron w wyniku jednego błędu strony).



Podstawą efektywnego funkcjonowania pamięci wirtualnej jest oczywiście precyzyjna identyfikacja zbioru stron aktywnych. Wykorzystanie tej informacji polega na utrzymaniu stron aktywnych w pamięci. Jeśli ze względu na zbyt duże potrzeby procesów strony aktywne muszą być usuwane z pamięci, to i tak trzeba ponieść koszt ich ponownego sprowadzenie. W tego typu sytuacji dochodzi najczęściej do migotania stron (szamotania, ang. trashing), jednak identyfikacja stron aktywnych umożliwia wykrycie takiego ryzyka i podjęcie pewnych działań zapobiegawczych.

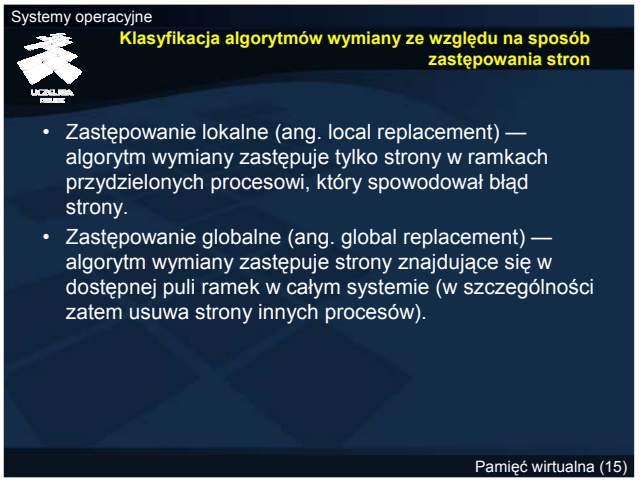
Sama identyfikacja wynikać może z różnych przesłanek, a precyzja tej identyfikacji uzależniona jest zasadności przyjętych przesłanek.



Klasyfikacja związana jest z okolicznościami, w jakich podejmowana jest decyzja o sprowadzaniu lub usunięci strony. W algorytmach wymiany na żądanie zarówno sprowadzanie, jak i usuwanie wykonywane jest wówczas, gdy jest absolutna konieczność. W przypadku sprowadzania oznacza to, że jeśli strony nie ma w pamięci, to nie zostanie sprowadzona wcześniej, niż po wystąpieniu odniesienia do niej (zaadresowania jej przez procesor). W przypadku usuwania absolutna konieczność występuje dopiero wówczas, gdy brakuje miejsca w pamięci operacyjnej (nie ma wolnej ramki), a ze względu na występujące odniesienia do stron, znajdujących się poza pamięcią operacyjną, konieczne jest zwolnienie ramki na potrzeby sprowadzanej strony. Ponieważ strona do sprowadzenia jest jednoznacznie określona przez stan systemu (stan pamięci oraz adres wystawiony przez procesor), specyfika algorytmu tej klasy zależy od wyboru strony usuwanej z pamięci.

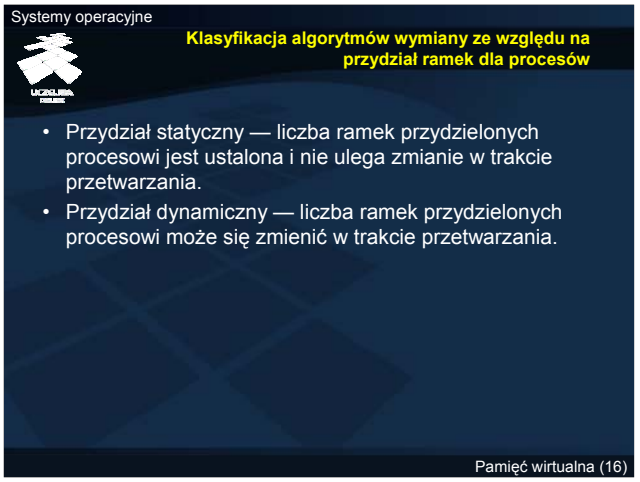
W klasie algorytmów wymiany ze sprowadzaniem na żądanie o specyfice algorytmu również decyduje sposób wyboru strony usuwanej, a często stron usuwanych. W tych algorytmach strona (lub kilka stron) może być usunięta w dowolnym momencie, niekoniecznie przy okazji obsługi błędu strony.

W klasie algorytmów ze wstępnym sprowadzaniem odniesienie do strony nieobecnej w pamięci operacyjnej skutkuje błędem strony, ale przy okazji jego obsługi może nastąpić sprowadzenie oprócz strony żądanej innych stron. Istotą algorytmów tej klasy jest wybór stron wstępnie sprowadzanych, można więc przyjąć dowolny sposób usuwania, łącząc w ten sposób ideę wstępnego sprowadzania z różnymi koncepcjami utrzymania stron w pamięci.



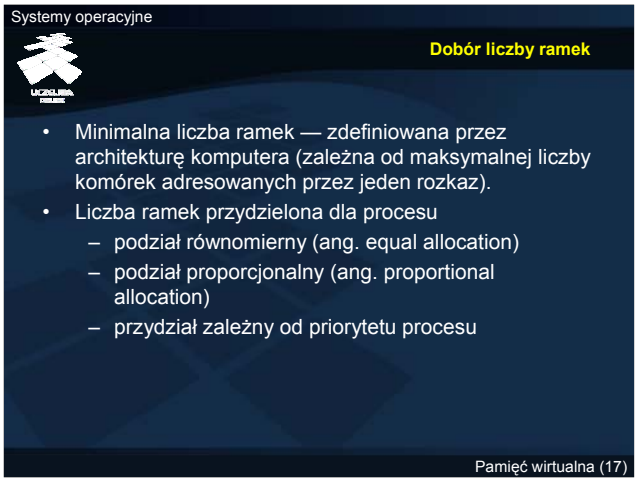
Klasyfikacja wiąże się z zakresem ramek uwzględnianych przy poszukiwaniu ofiary. W przypadku zastępowania lokalnego proces dysponuje pewną liczbą ramek, w których muszą się pomieścić jego strony. Błąd strony tego procesu skutkuje ewentualnym usunięciem innej jego strony. Można w ten sposób ograniczyć (ale nie zlikwidować) negatywny wpływ zbyt częstego generowania błędów strony przez proces na efektywność działania całego systemu.

W zastępowaniu globalnym poszukiwanie ramki ofiary dotyczy całej puli ramek niezależnie od bieżącego przydziału. Może więc dojść do podkradania ramek, kiedy jeden proces traci ramkę na rzecz innego. Takie podejście ułatwia adaptację liczby ramek używanych przez procesy do zróżnicowanych potrzeb, ale jest niebezpieczne dla efektywności całego systemu, gdy liczba ramek jest zbyt mała. Może wówczas dojść do zjawiska szamotania.



Klasyfikacja związana jest z adaptacją przydziału zasobów do zmieniających się potrzeb procesów. W przydziale statycznym proces otrzymuje pewną liczbę ramek na cały czas cyklu przetwarzania. W przydziale dynamicznym liczba ramek przydzielonych procesowi może się zmienić zależnie od intensywności generowania błędów strony. Jeśli liczba błędów strony, generowanych przez proces jest stosunkowo duża to w zależności od możliwości, jakimi dysponuje system, można przydzielić dodatkowe ramki. Dodatkowe ramki mogą pochodzić z puli ramek odebranych procesom, które generowały mało błędów strony.

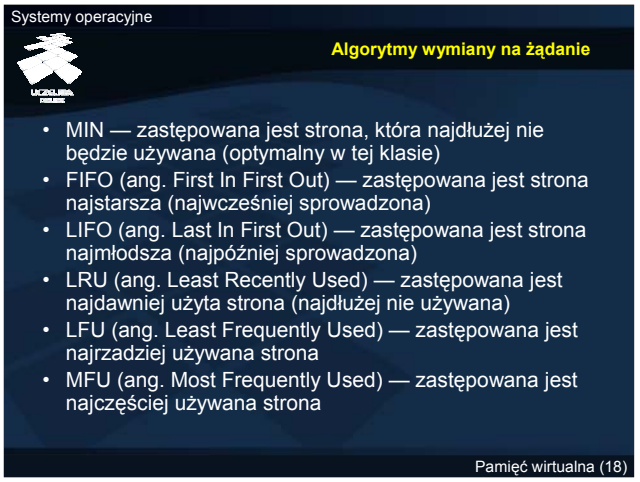
Zastępowanie globalne w naturalny sposób prowadzi do przydziału dynamicznego, nie ma natomiast sensu w przypadku przydziału statycznego. Przydział statyczny wymaga zastępowania lokalnego, ale w przypadku zastępowania lokalnego również można zastosować przydział dynamiczny. W ramach obsługi błędu strony wymiana jest lokalna, ale okresowo liczba ramek może się np. zwiększyć, co wyeliminuje na jakiś czas problem zastępowania w przypadku kolejnych błędów strony. Liczba ramek przydzielonych procesowi może też oczywiście się zmniejszyć.



Z przydziałem ramek wiąże się ustalanie ich liczby. Jest to szczególnie istotne w przydziale statycznym, ale dotyczy również początkowej liczby ramek w przydziale dynamicznym. Niezależnie od sposobu przydziału proces, który jest w pamięci powinien mieć do dyspozycji pewną minimalną liczbę ramek, gwarantującą uniknięcie problemu wznawiania rozkazów. Liczba ramek powinna być nie mniejsza, niż maksymalna liczba różnych adresów, jakie mogą być wystawione na magistrali w jednym cyklu rozkazowym.

Zakładając, że spełnione jest minimum dla każdego procesu przebywającego w pamięci, liczba ramek może być:

- w przypadku przydziału równomiernego równa dla wszystkich procesów,
- proporcjonalna do rozmiaru obrazu procesu,
- uzależniona od priorytetu procesu — proces o wyższym priorytecie będzie miał więcej ramek, żeby jego przetwarzania odbywało się sprawniej.

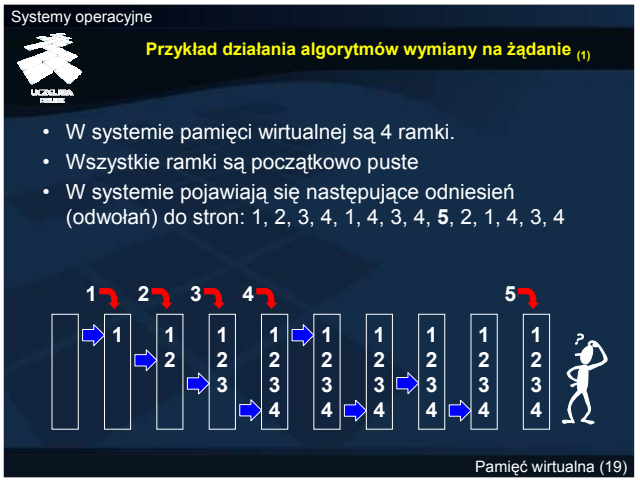


Algorytmy wymiany na żądanie stosowane są w systemach jednozadaniowych lub w systemach wielozadaniowych ze statycznym przydziałem ramek.

Algorytm MIN oparty jest na nierealnej przesłance, wymagającej znajomości **przyszłego** ciągu odniesień. Z drugiej strony jest to algorytm optymalny w tej klasie, dlatego wykorzystywany jest dla celów porównawczych. Można w ten sposób sprawdzać, ile tracimy, opierając się na przesłankach z historii dotychczasowych odniesień. Jest to zatem swego rodzaju miara zasadności tych przesłanek.

Algorytmy FIFO i LIFO oparte są na kolejności sprowadzania stron do pamięci. FIFO usuwa strony w kolejności ich sprowadzania LIFO w kolejności odwrotnej. FIFO sprawdza się dobrze w przypadku programów, w których jest prosty, sekwencyjny przepływ sterowania do początku programu do końca, z małą liczbą pętli, czy wywołań podprogramów. LIFO natomiast właściwy jest dla przypadków pętli, gdyż ponowne przejście sterowania do tej samej instrukcji nastąpi dopiero w następnej iteracji. Dla pętli istnieje jeszcze inny algorytm — LD (ang. loop detection), którego prezentację tu pominięto.

Algorytmy LRU, LFU i MFU oparte są na przesłankach, wymagających monitorowania odniesień do pamięci. Dla LRU istotny jest czas ostatniego odniesienia, a dla LFU i MFU liczba odniesień w przeszłości. LRU jest typowym algorytmem dla programów, charakteryzujących się lokalnością czasową odniesień do pamięci. Algorytmy LFU i MFU (tzw. algorytmy licznikowe) oparte są na zupełnie przeciwnych przesłankach: LFU usuwa stronę, do której było najmniej odniesień do początku przetwarzania lub od momentu sprowadzenia do pamięci (są to dwa warianty algorytmów licznikowych), a MFU usuwa stronę, do której było najwięcej odniesień.



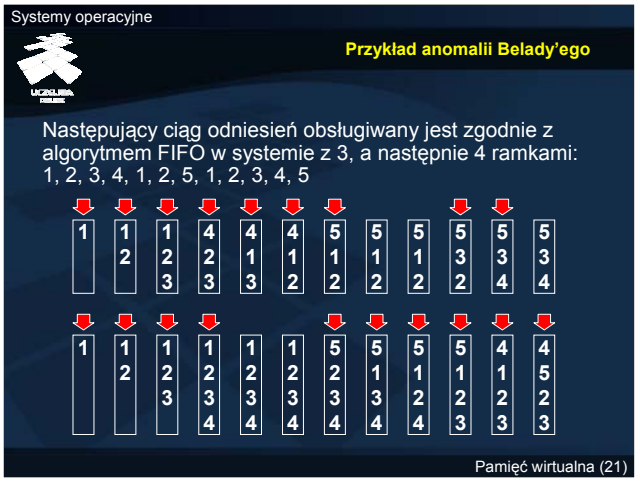
Przedstawiony przykład pokazuje przebieg odniesień do stron o numerach od 1 do 5 w obrębie 4 początkowo pustych ramek. Pierwsze 4 odniesienia powodują błąd strony, ale nie występuje problem zastępowania, gdyż dostępne są wolne ramki. Dopiero odniesienie do strony nr 5 wymaga zwolnienia jakieś ramki. Wybór strony do usunięcia zależy od algorytmu wymiany.

Na kolejnych rysunkach pokazanych jest stan poszczególnych ramek, zależnie od zastosowanego algorytmu. Przy okazji pokazane jest odniesienie do pamięci, które spowoduje kolejny błąd błąd strony i tym samym problem zastępowania.

Usunięcie tej samej strony w przypadku algorytmów LFU i LRU oraz MFU i LIFO jest przypadkowe i kolejna wymiana w przypadku tych algorytmów może wykazać różnice.

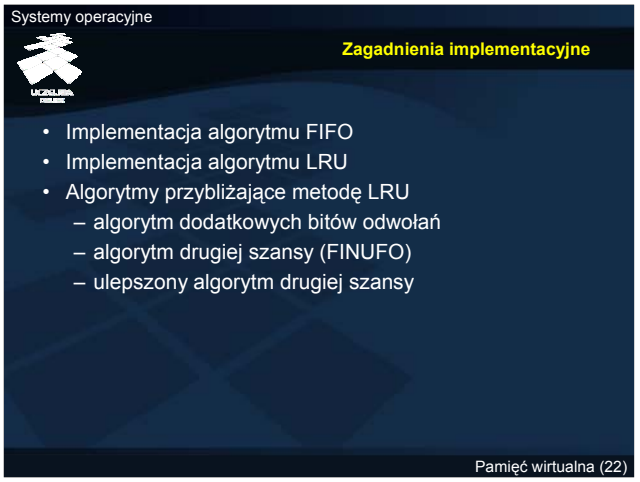
Warto też przy tej okazji zwrócić uwagę, że dla algorytmów licznikowych wybór może być niejednoznaczny, gdyż liczba odniesień do różnych stron może być taka sama. Jako kryterium rozstrzygnięcia można przyjąć czas sprowadzenia, czyli strony z taką samą wartością licznika usuwane są w kolejności FIFO.

Przy założeniu, że strony sprowadzane są pojedynczo nie ma tego problemu w algorytmach opartych na kolejności zdarzeń w czasie, czyli MIN, FIFO, LIFO i LRU.

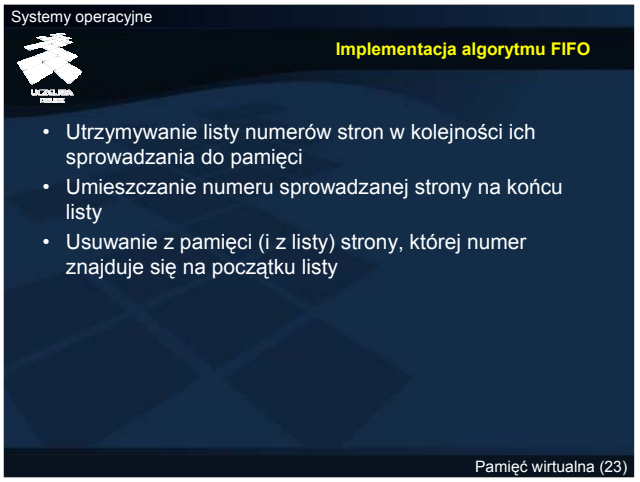


W przypadku 3 ramek pierwsze 7 odniesień do stron kończy się błędem, ale kolejna 2 odniesienia przebiegają bez błędów, a później pojawiają się jeszcze 2 błędy. W sumie jest ich 9.

W przypadku 4 ramek początek jest bardziej optymistyczny, gdyż po 4 pierwszych odniesieniach z błędem, wypełnione ramki zdają się gwarantować stabilność przetwarzania. Jednak sprowadzenie strony nr 5 powoduje usunięcie strony numer 1, która sprowadzana jest jako następna, usuwając z kolei potrzebną później stronę nr 2 itd. Ostatecznie mamy 10 błędów strony pomimo **zwiększenia liczby ramek**. Paradoks ten, nazywany anomalią Belady'ego i dotyczy wybranych algorytmów, między innymi algorytmu FIFO. Anomalia nie ujawnia się być może zbyt często, ale może wzbudzać obiekcje co do efektów przydziału dodatkowych zasobów.

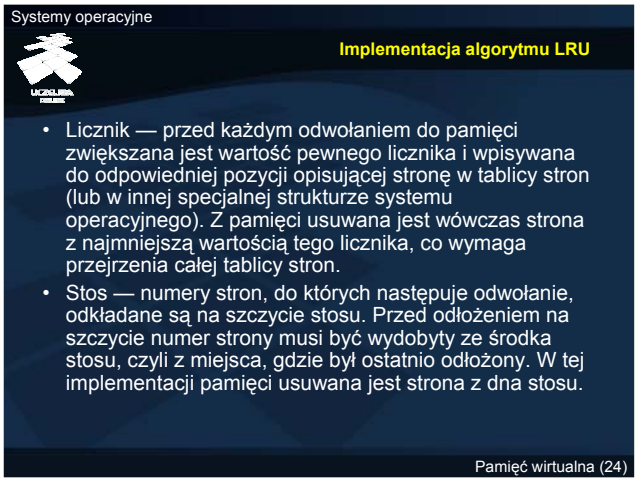


Jak już wcześniej wspomniano, kluczowa dla efektywności funkcjonowania systemu z pamięcią wirtualną jest minimalizacja liczby błędów strony. W celu ograniczenia kosztów czasowych dostępu do pamięci stosowane są odpowiednie algorytmy wymiany. Na przykładzie algorytmów FIFO i LRU wyjaśnione zostaną natomiast zagadnienia związane z **kosztem działania samego algorytmu**.



Implementacja algorytmu FIFO bazuje na kolejce FIFO, którą można łatwo zaimplementować za pomocą listy jednokierunkowej.

Warto zwrócić uwagę, że aktualizacja listy następuje w ramach obsługi błędu strony, gdyż stronę umieszczamy na liście po sprowadzeniu. W przypadku usuwania na żądanie, również usuwanie strony z listy wykonywane jest przy obsłudze błędu strony.

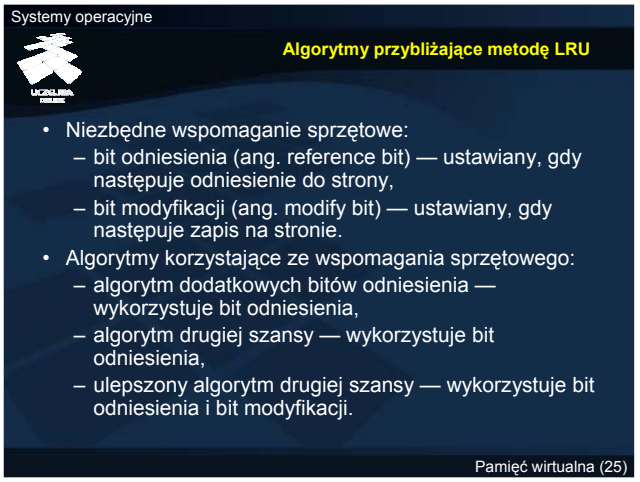


Rozwiązanie z licznikiem wymaga zwiększenia licznika odniesień do pamięci i stosownej aktualizacji opisu strony (lub ramki). Jest to operacja stosunkowo szybka w porównaniu z wyszukiwaniem strony do usunięcia w przypadku konieczności zwolnienia ramki. Pewnym problemem jest ryzyko przepełnienia licznika.

W przypadku rozwiązania opartego na stosie jest odwrotnie. Aktualizacja stosu zbudowanego na liście wymaga zmiany zaledwie kilku wskaźników, ale wyszukanie strony w głębi stosu jest bardziej czasochłonne. Wybór strony do usunięcia w przypadku konieczności zwolnienia ramki jest za to natychmiastowy.

Porównując oba podejścia należałoby zauważyć, że błąd strony jest zjawiskiem nieporównywalnie rzadszym niż odniesienie do strony. W przeciwnym przypadku stosowanie pamięci wirtualnej nie miałoby żadnego sensu. Preferowane byłoby zatem rozwiązanie licznikowe. Jednak implementacja tego rozwiązania na poziomie systemu operacyjnego wymaga przekazania sterowania do programu jądra **przy każdym odniesieniu do strony**. Nawet jeśli aktualizacja licznika i tablicy stron wymaga wykonania zaledwie kilku rozkazów, to i tak powoduje to kilkunastokrotne spowolnienie **każdego dostępu** do pamięci. Programowe rozwiązanie oparte na stosie byłoby jeszcze bardziej czasochłonne.

Mogłoby się zatem okazać, że koszt algorytmu LRU pochłania zysk, wynikający z trafności decyzji co do stron wymienianych. Konieczne jest zatem odpowiednie wsparcie na poziomie architektury komputera.



Implementacja na poziomie architektury komputera nie może być zbyt kosztowana, gdyż nie wiadomo, jaki algorytm wymiany będzie implementowany w systemie operacyjnym i czy w ogóle realizowana będzie pamięć wirtualna z wymianą stron. Jeśli zatem ma dostarczane jest wsparcie dla implementacji wymiany, powinno ono być na tyle uniwersalne, żeby można była je zaadaptować do różnych algorytmów.

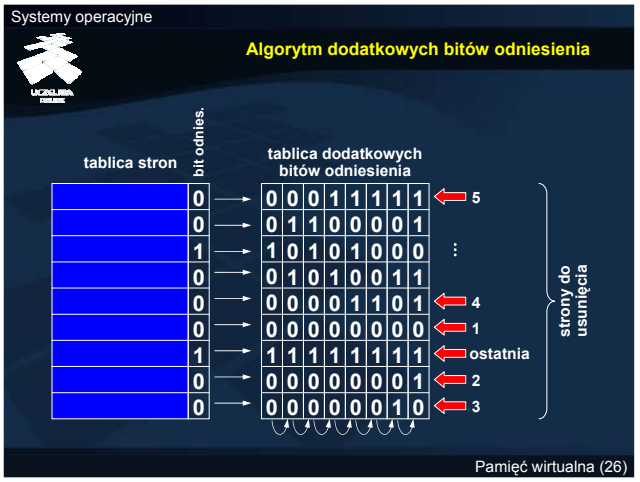
W ramach wsparcia na poziomie architektury jednostka zarządzania pamięcią ustawia odpowiednie bity na właściwej pozycji w tablicy stron w przypadku wykrycia odniesienia do strony:

- bit odniesienia (ang. reference bit) — ustawiany dla danej strony zawsze, gdy następuje zapis lub odczyt jakieś komórki na tej stronie,
- bit modyfikacji (ang. modify bit) — ustawiany dla danej strony przez sprzęt zawsze, gdy następuje zapis na tej stronie. (Wspomniany przy omawianiu problemu wymiany).

W dalszej części omówione zostaną 3 algorytmy, w których wykorzystywane jest omówione wspomaganie sprzętowe:

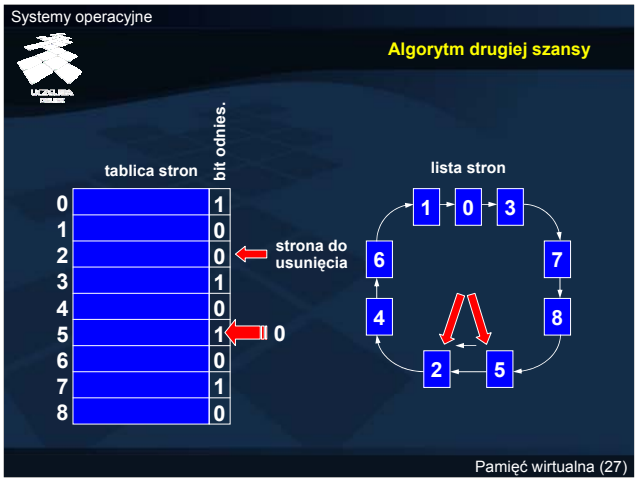
- algorytm dodatkowych bitów odniesienia (wykorzystanie bitu odniesienia),
- algorytm drugiej szansy (wykorzystanie bitu odniesienia),
- ulepszony algorytm drugiej szansy (wykorzystanie bitu odniesienia i bitu modyfikacji).

Poza tym, bity te wykorzystywane są w niektórych algorytmach wymiany ze sprowadzaniem na żądanie.



W algorytmie dodatkowych bitów odniesienia okresowo sprawdzane są bity odniesienia każdej ze stron i kopiowane do dodatkowej struktury, zwanej tablicą dodatkowych bitów odniesienia. Bity kopiowane są na najbardziej znaczącą pozycje, ale prze skopiowaniem następuje logiczne przesunięcie bitów w prawo na każdej pozycji. Bit najmniej znaczący jest tracony. Tablica przechowuje zatem informacje o wystąpieniu błędu strony w kilku ostatnich okresach kontrolnych. Ciąg samych zer oznacza, że nie było żadnego odniesienia w ostatnim czasie. Ogólnie im mniejsza wartość tym bardziej odległe w czasie jest ostatnie odniesienie lub mniejsza jest liczba odniesień. Horyzont czasowy tej informacji zależy od liczby bitów na każdej pozycji oraz długości okresu kontrolnego. Oczywiście im dłuższy okres kontrolny, tym mniej precyzyjna jest informacja, gdyż reprezentowana jest tylko przez 1 bit.

W przypadku konieczności wymiany poszukiwana jest strona, dla której wartość w tablicy jest najmniejsza, a ramka tej strona jest użyta do wymiany. Precyzja działania algorytmu wymaga uwzględnienia również bieżącej wartości bitu odniesienia na najbardziej znaczącej pozycji wektora. Można przyjąć, że tuż przed wybraniem strony ofiary następuje przesunięcie bitów w tablicy i skopiowanie bitów odniesienia.



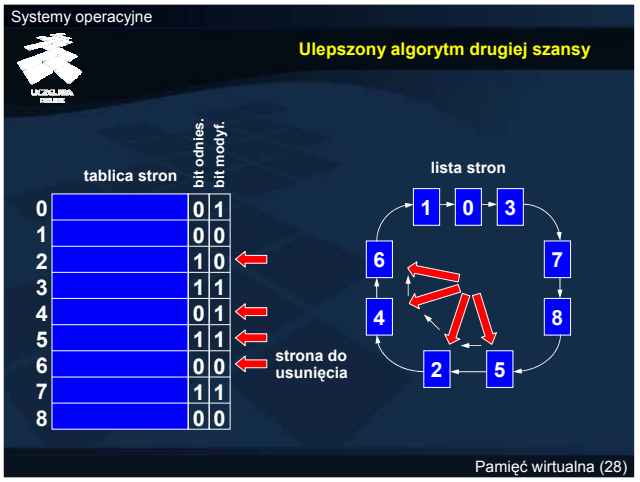
Algorytm drugiej szansy (FINUFO, ang. First In Not Used First Out) jest jednym z tzw. algorytmów zegarowych (wskazówkowych), w których numery stron (lub ramek, w tym przypadku ramek danego procesu) tworzą listę cykliczną (tarczę zegara), a wskazówka wskazuje kandydatkę do usunięcia z pamięci w przypadku błędu strony. Jeśli jednak strona, wskazywana do usunięcia była ostatnio adresowana (ma ustawiony bit odniesienia), otrzymuje drugą szansę. Bit odniesienia dla tej strony jest kasowany, a wskazówka przesuwa się na stronę następną. Sprowadzana strona zastępuje więc stronę na wskazanej ostatecznie pozycji, po czym wskazówka przesuwa się do następnej strony.

W przedstawionym przykładzie wskazywana jest strona nr 5, ale bit odniesienia do tej strony jest ustawiony, więc po jego skasowaniu wskazówka przesuwa się na stronę nr 2. Dla tej strony bit odniesienie jest skasowany, więc nastąpi jej usunięcie, konkretnie ramka tej strony zostanie użyta do wymiany, a wskazówka przesunie się na stronę nr 4.

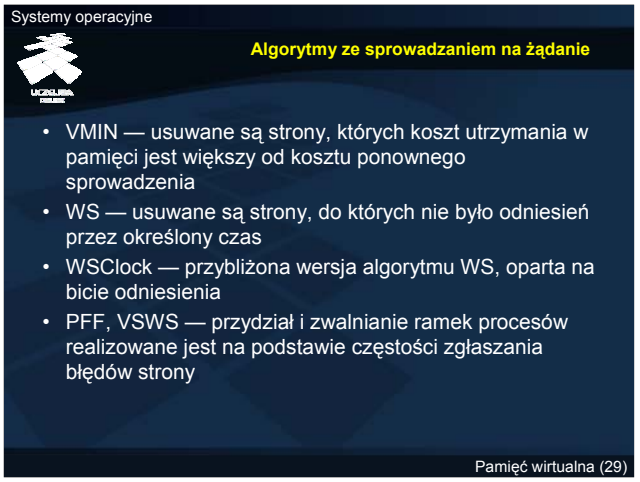
W niektórych publikacjach odróżnia się algorytm zegarowy od algorytmu drugiej szansy (FINUFO), przedstawiając ten drugi, jako algorytm FIFO, w którym strona na czole kolejki jest:

- usuwana, jeśli nie ma ustawionego bitu odniesienia, albo
- przesuwana jest na koniec, jeśli jest ustawiony bit odniesienia, który jest przy tym kasowany.

Algorytm zegarowy natomiast przedstawia się jako listę cykliczną ramek, w których wymieniane są strony zgodnie z położeniem wskazówki.



Ulepszenie algorytmu drugiej szansy w tym przypadku polega na uwzględnieniu bitu modyfikacji. W celu redukcji kosztów unika się po prostu wymiany stron modyfikowanych, które muszą być dodatkowo zapisane na urządzeniu wymiany. Wyszukiwanie strony do wymiany polega więc na analizie wartości 2-bitowej, złożonej z bitu odniesienia na bardziej znaczącej pozycji i bitu modyfikacji na mniej znaczącej pozycji. W przestawionym przykładzie „najlepsza” do usunięcia zgodnie z obiegiem wskazówki jest strona nr 6, gdyż ma wyzerowane oba bity.



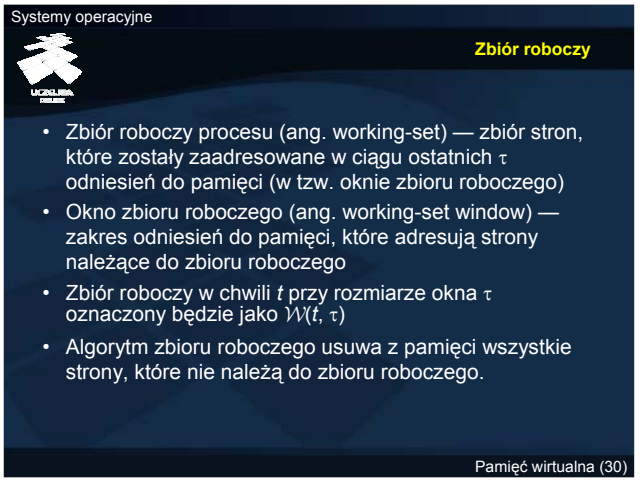
Istotą algorytmów ze sprowadzaniem na żądanie jest zwalnianie ramek, uznanych za nie wykorzystywane, wcześniej niż to wynika z potrzeb realizacji samej wymiany. Celem wcześniejszego zwalniania jest uzyskanie ramek na potrzeby innych procesów, którym tych ramek brakuje. W skrajnych przypadkach niewystarczająca liczba ramek może być przyczyną zawieszenie procesu i przeniesienia całego obrazu w obszar wymiany. Uzyskanie wolnych ramek umożliwia z kolei aktywację procesu — wprowadzenie do pamięci operacyjnej niezbędnych jego stron.

Algorytm VMIN (ang. Variable-space MIN) jest optymalny w klasie algorytmów ze sprowadzaniem na żądanie, ale oparty jest na znajomości **przyszłego** ciągu odniesień do stron. Decyzja o usunięci strony z pamięci wynika z porównania kosztu ponownego sprowadzenia, oraz kosztu utrzymania do czasu następnego odniesienia. Jeśli koszt ponownego sprowadzania jest mniejszy, strona jest usuwana.

Algorytm WS (ang. Working Set) oparty jest na koncepcji *zbioru roboczego*, czyli zbioru stron, do których było odniesienie w ostatnim okresie czasu w przeszłości. Długość tego okres mierzona liczbą odniesień, jest parametrem algorytmu. Algorytm WS wymaga monitorowania odniesień do pamięci, więc wymaga wsparcia sprzętowego. Wykorzystywany jest tutaj bit odniesienia.

Przybliżoną realizacją idei zbioru roboczego, wykorzystującą bit odniesienia, jest również algorytm WSClock.

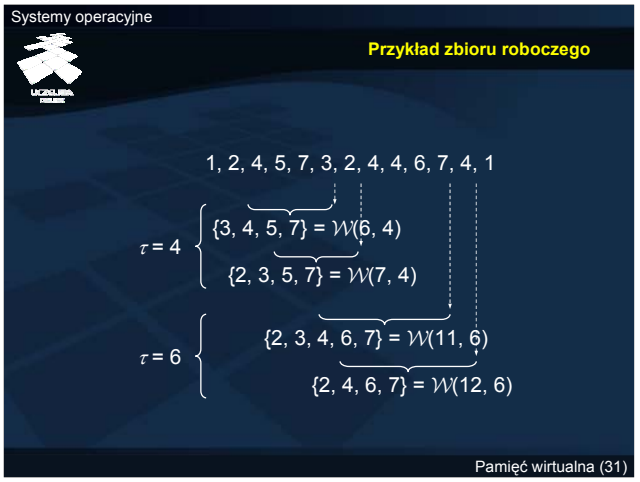
Algorytmy PFF (ang. Page Fault Frequency) i VSWS (ang. Variable-Interval Sampled Working Set) dają efekt podobny do zbioru roboczego, przy czym oparte są na częstość generowania błędów strony.



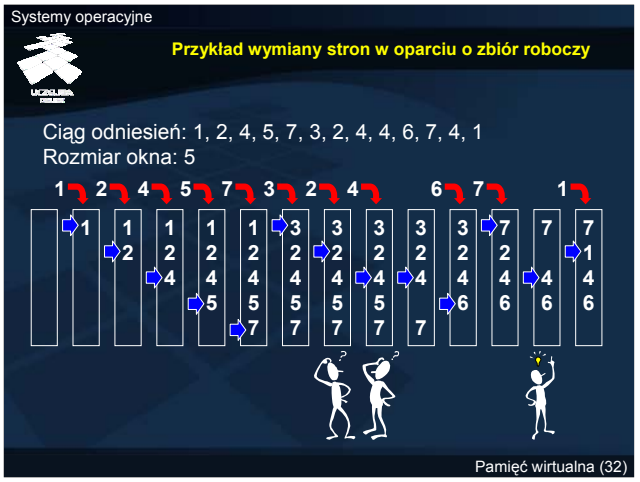
Przedstawiona definicje — zbioru roboczego oraz okna zbioru roboczego — oznaczają właściwie to samo. Pierwsza definiuje zbiór roboczy w oparciu o pojęcie okna, a druga definiuje okno w oparciu o pojęcie zbioru roboczego. Z punktu widzenia strony, pozostaje ona w zbiorze roboczym tak długo, jak długo ostatnie odwołanie do niej pozostaje w bieżącym oknie. Oznacza to, że od momentu ostatniego zaadresowania strony, pozostaje ona w zbiorze roboczym przez τ odwołań.

Ogólna (teoretyczna) koncepcja zbioru roboczego polega utrzymywaniu takiego zbioru dla każdego procesu. Strony, należące do zbioru roboczego, pozostawiane są w pamięci, natomiast ramki stron spoza zbioru roboczego są zwalniane. Zapotrzebowanie na ramki w systemie jest sumą zbiorów roboczych wszystkich procesów. Jeśli liczba ta jest większa niż dostępna liczba ramek, może dojść do szamotania. Zadaniem zarządcy pamięci jest wówczas przeniesienie całego obrazu któregoś z procesów w obszar wymiany i odzyskanie jego ramek. Po uzyskaniu odpowiednio dużej liczby wolnych ramek (w przypadku zmniejszenia się zbiorów roboczych) można aktywować któryś z procesów zawieszonych. Wybór procesu do usunięcia lub aktywowania jest zadaniem planisty średnioterminowego.

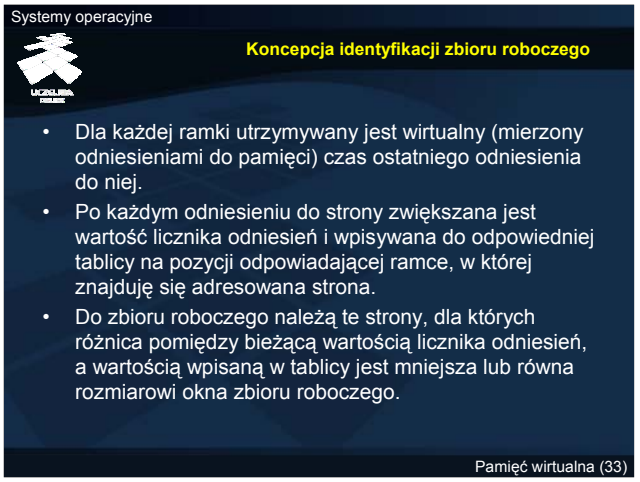
Osobnym problemem jest dobór wielkości okna. Okno powinno być na tyle duże, żeby objąć tzw. strefę, czyli zbiór stron niezbędnych do wykonania określonego fragmentu programu (np. procedury). W praktyce zależy to od częstości generowania błędów strony przez proces.



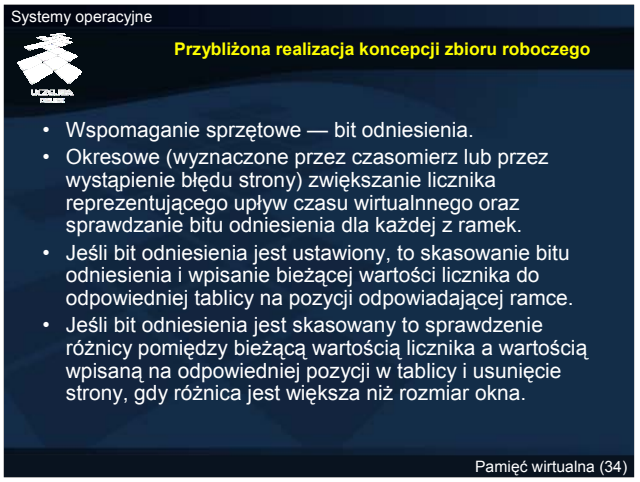
Dla przykładowego ciągu odniesień przedstawiono 2 zbioru robocze przy rozmiarze okna 4 oraz 2 zbiory robocze przy rozmiarze okna 6. Ze względu na powtarzające się odniesienia moc zbioru roboczego (liczba elementów) może być mniejsza niż rozmiar okna, co widać w przypadku okna o rozmiarze 6.



Przykład pokazuje realizację ciągu odniesień z poprzedniego slajdu przy rozmiarze okna 5. Proces ma więc do dyspozycji 5 ramek. Początkowo wymiana przebiega tak samo, jak w przypadku FIFO lub LRU. Ciekawym momentem jest drugie odniesienie do strony nr 2. Formalnie strona nr 2 została właśnie usunięta ze zbioru roboczego, ale nastąpiło do niej odniesienie. Jest więc błąd strony, chociaż jego obsługa nie wymaga sprowadzania strony ponownie z pliku wymiany. Podobna sytuacja ma miejsce przy następnym odniesieniu do pamięci — do strony nr 4. Po kolejnym odniesieniu — ponownie do strony nr 4 — poza bieżącym oknem znajduje się ostatnie odniesienie do strony nr 5, więc strona ta jest usuwana ze zbioru roboczego i tym samym z pamięci operacyjnej. Po ostatnim odniesieniu do strony numer 4 zbiór roboczy redukuje się do 3 stron.



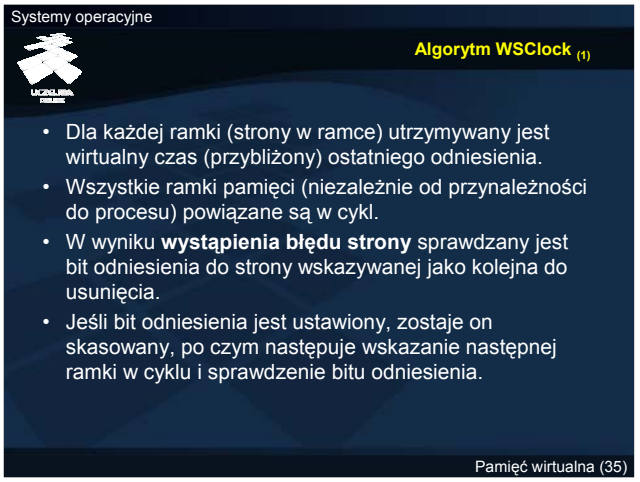
Precyzyjna identyfikacja zbioru roboczego mogłaby być zaimplementowana podobnie jak algorytm LRU. Zaprezentowana koncepcja jest adaptacją licznika odniesień do stron. Problemem podobnie, jak w przypadku LRU, jest monitorowanie odniesień do stron oraz złożoność struktur danych na potrzeby pamiętania licznika. Ze względu na koszt takiej realizacji algorytm WS implementowany jest w sposób przybliżony.



Czas wirtualny i tym samym czas odniesienia do strony wyznaczany jest w sposób przybliżony — z dokładnością do okresu kontrolnego, wyznaczanego przez czasomierz lub błąd strony. Wystąpienie odniesienia do strony sprawdzane jest na podstawie bitu odniesienia w tablicy stron. Bieżąca wartość licznika staje się wirtualnym czasem odniesienia dla tych stron, dla których bit odniesienia jest ustawiony. Brak odniesienia do strony może oznaczać, że ostatnie odniesienie jest już poza oknem i stronę należy usunąć. W tym celu sprawdzana jest różnica pomiędzy bieżącą wartością licznika (bieżącym czasem wirtualnym), a czasem ostatniego odniesienia, odczytanym dla danej strony/ramki z przeznaczonej na to tablicy.

Upływ czasu wirtualnego musi być rejestrowany oddzielnie dla każdego procesu i tylko wówczas, gdy jest on wykonywany. Przerwanie zegarowe powinno więc zwiększać licznik w tym procesie, w kontekście którego jest obsługiwane.

Pomimo uniknięcia konieczności monitorowania wszystkich odniesień do pamięci, implementacja tego algorytmu i tak jest kosztowana, gdyż po każdym przerwaniu zegarowym wymagane jest testowanie bitu odniesienia **każdej** ze stron procesu.

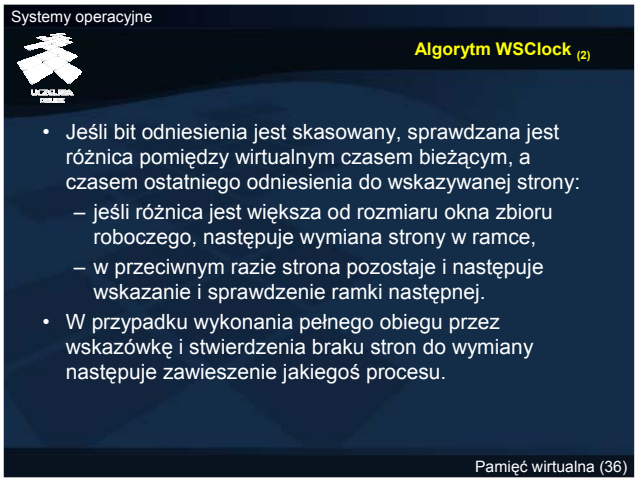


Jak sama nazwa wskazuje, algorytm łączy podejście oparte na zbiorze roboczym z koncepcją algorytmu zegarowego. Podobnie jak w przybliżonej realizacji algorytmu zegarowego, utrzymywany jest wirtualny czas ostatniego odniesienia do strony, mierzony osobno dla każdego procesu.

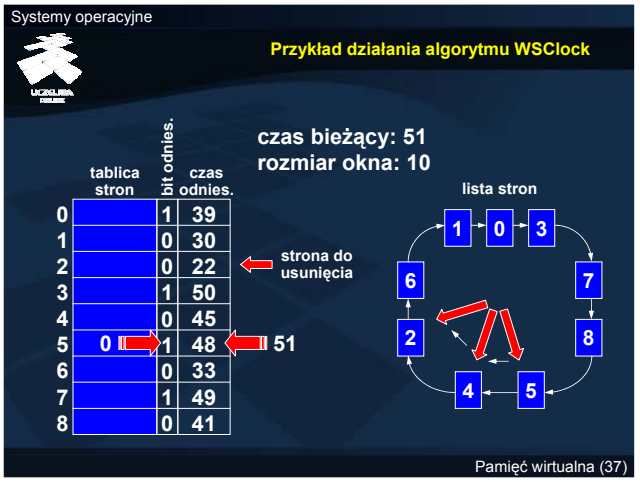
Warto zwrócić uwagę na dwie istotne cechy algorytmu:

- lista cykliczna (tarcza zegara) obejmuje wszystkie ramki w systemie, a nie ramki procesu, jak w przypadku klasycznego algorytmu zegarowego,
- próba usunięcia strony podejmowana jest w reakcji na błąd strony, co oznacza, że jest to właściwie algorytm wymiany na żądanie.

Zasadniczą różnicą w stosunku do algorytmów wymiany na żądanie jest zastępowanie globalne. Podobieństwo do koncepcji zbioru roboczego przejawia się natomiast w usuwaniu pierwszej napotkanej strony, nie należącej do zbioru roboczego swojego procesu.

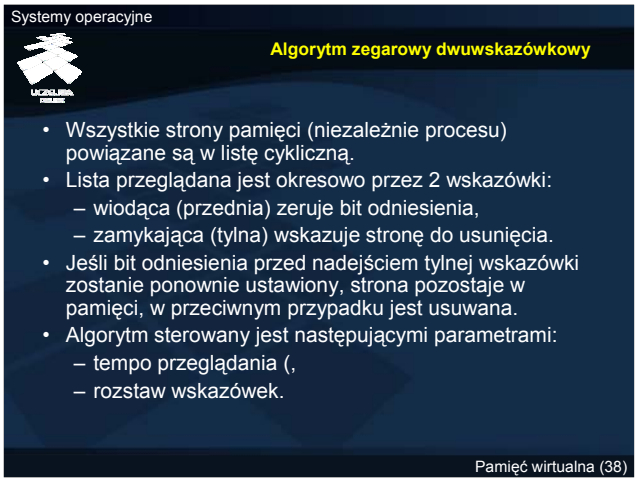


Postępowanie w drugiej części algorytmu jest podobne, jak w przypadku właściwego przybliżenia algorytmu WS, czyli strona, do której nie było odniesienia, weryfikowana jest pod kątem przynależności do zbioru roboczego. Wyszukiwanie kończy się jednak na znalezieniu pierwszej ramki ofiary. Brak takiej ramki (brak strony ofiary) zgodnie z ideą zbioru roboczego oznacza, że suma rozmiarów zbiorów roboczych wyczerpuje limit dostępnych ramek pamięci fizycznej i należy zwiesić jakiś proces.



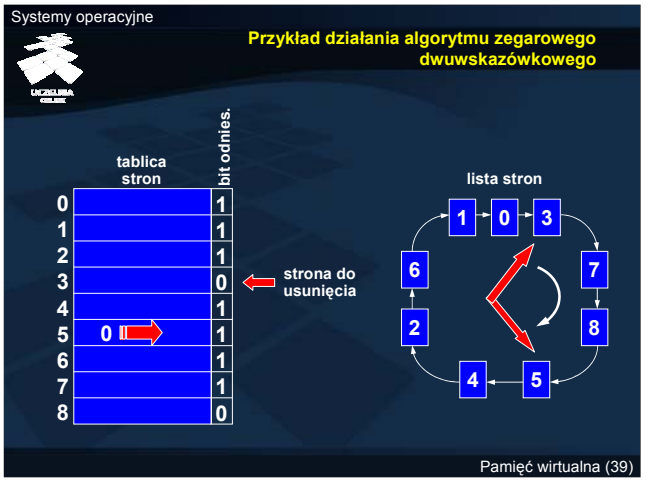
W przykładzie wskazana do usunięcia jest strona nr 5, więc od ramki tej strony rozpocznie się poszukiwanie ofiary. Strona 5 ma ustawiony bit odniesienia więc z założenia jest w zbiorze roboczym i nie zostanie usunięta z pamięci. Wyzerowany zostanie jedynie bit odniesienia i ustawiony wirtualny czas odniesienia do strony na wartość aktualną dla bieżącego procesu, czyli 51. Dla uproszczenia przyjmijmy, że strony 2, 4 i 5 należą do tego samego procesu.

Kolejną kandydatką na liście jest strona nr 4. Bit odniesienia jest dla tej strony skasowany, ale ostatni czas użycia nie jest jeszcze poza oknem (45>51–10), więc strona pozostaje w pamięci, a czas jej użycia się nie zmienia. Następna na liście jest strona nr 2, której ostatnie użycie, zgodnie z wirtualnym czasem, jest poza oknem zbioru roboczego. Strona nr 2 we wskazanej ramce zostanie zastąpiona.

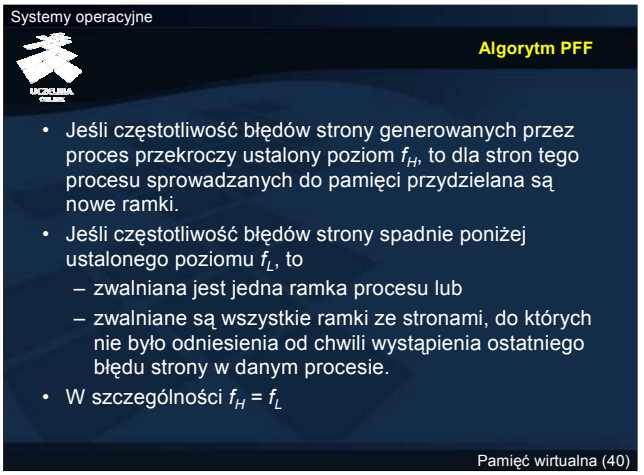


Algorytm dwuwskazówkowy stosowany jest w niektórych systemach rodziny UNIX, np. SVR4. Realizuje on strategię, określaną jako NRU (Not Recently Used), która w tym przypadku oznacza, że jeśli w czasie pomiędzy przejściem wskazówki wiodącej a nadejściem zamykającej nie został ustawiony bit odniesienia, to strona jest usuwana. Czas pomiędzy przejściem tych wskazówek zależy od tempa przeglądania oraz rozstawu wskazówek. Parametry te są natomiast ustawiane zależnie od liczby wolnych ramek w systemie.

Jeśli uruchomienie algorytmu nie doprowadzi do uzyskania wystarczająco dużej liczby wolnych ramek, następuje zawieszenie jakiegoś procesu i zwolnienie jego ramek.



Wskazówka wiodąca w przykładzie ustawiana jest na stronie nr 5, której bit jest zerowany. Wskazówka zamykająca ustawiona jest na stronie nr 3, dla której bit odniesienia nie został ustawiony, więc nastąpi jej usunięcie. Warto zwrócić uwagę, że bit odniesienie został ustawiony dla strony nr 7, więc po przejściu wskazówki zamykającej strona ta pozostanie w pamięci. W międzyczasie może zostać ustawiony bit odniesienia do strony nr 8. Jeśli jednak nie nastąpi to przed nadejściem zamykającej wskazówki, strona ta zostanie usunięta.



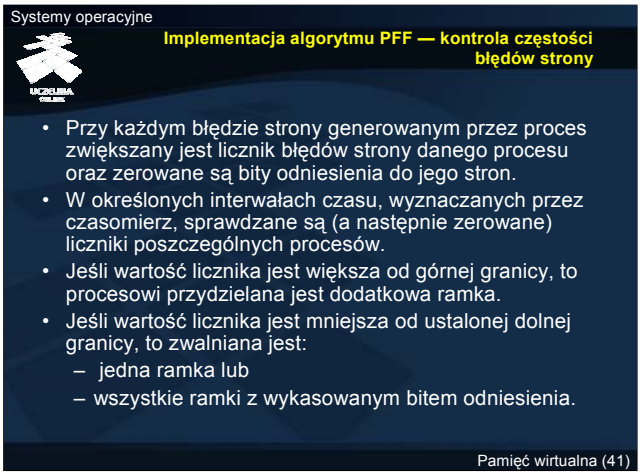
Algorytmy bazujące na zbiorze roboczym są trudne w implementacji, gdyż ze względu na brak możliwości monitorowania odniesień do stron muszą być realizowane w sposób przybliżony.

Podobne efekty przy mniejszym koszcie implementacji można uzyskać stosują podejście oparte na kontroli *częstości błędów strony*. Podejście jest łatwiejsze w implementacji, gdyż wymaga aktualizacji struktur danych **tylko w przypadku wystąpienia błędu strony**, którego obsługa jest i tak dość czasochłonna.

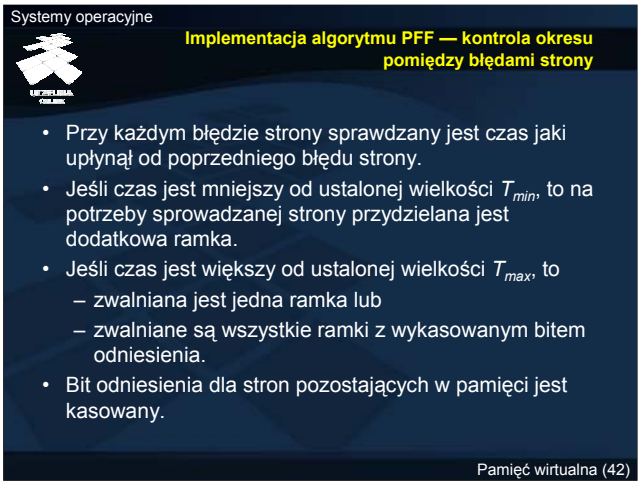
Ogólna idea polega na tym, żeby zabierać ramki procesom zgłaszającym mało błędów strony, a przydzielać procesom, które często generują błędy strony. W przypadku przekroczenia dolnego progu można rozważać dwie strategie postępowania: żwawą i opieszałą. W strategii żwawej zbyt zwalniane są wszystkie ramki, dla których bit odniesienie jest skasowany, a w strategii opieszałej zwalniana jest tylko jedna ramka.

Implementacja algorytmu może być oparta na :

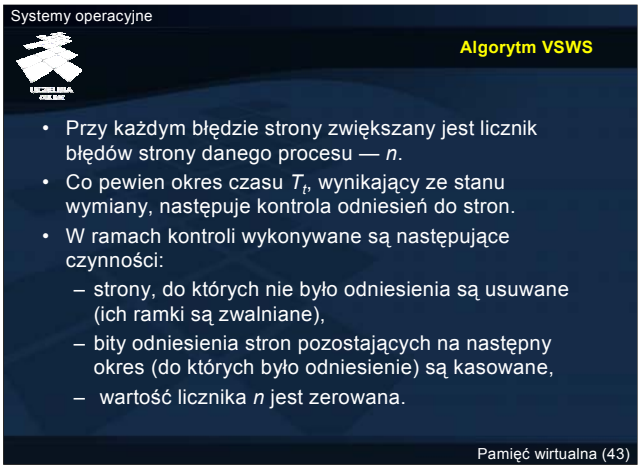
- liczeniu błędów strony, czyli rzeczywistym wyznaczaniu częstotliwości błędów,
- mierzeniu okresu pomiędzy błędami strony.



Kontrola częstości błędów strony wymaga wykonania dodatkowego zadania okresowego, wyznaczanego przez czasomierz, niezależnego od błędu strony. Umożliwia to oparcie decyzji o przydziale lub odebraniu ramek na średniej z dłuższego okresu, w przeciwieństwie do implementacji opartej na sprawdzaniu okresu między kolejnymi błędami strony.

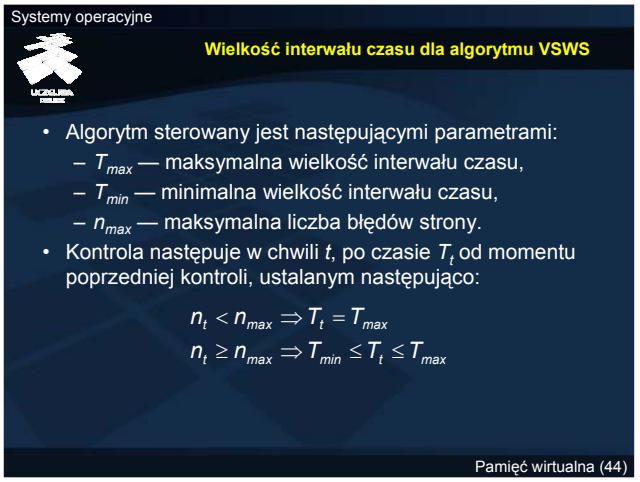


Sprawdzanie okresu między kolejnymi błędami strony nie wymaga dodatkowej kontroli poza obsługą błędu strony, gdyż czas odmierzany jest przez jądro na podstawie taktów zegara, niezależnie od zastosowanego podejścia do wymiany stron. Problemem może być jednak pewna destabilizacja przydziału ramek, wynikająca ze zbyt szybkiej reakcji na przypadkowe błędy strony występujące w krótkim czasie. W teorii automatycznej regulacji można by to podsumować jako dominację członku różniczkującego (duży czas wyprzedzenia).

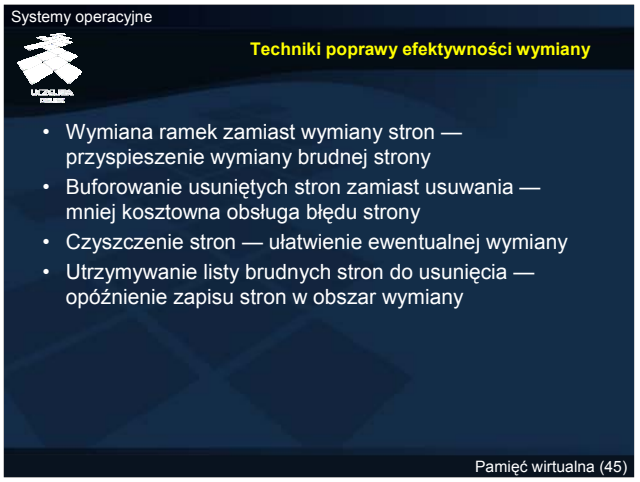


W zależności od implementacji, algorytm PFF albo słabo (wolno) dostosowuje się do zmieniającej się strefy w procesie, albo reaguje zbyt gwałtownie. Wolne dostosowanie polega na tym, że po wejściu w nową strefę zbyt długo utrzymywane są strony ze starej strefy. Może to powodować chwilowy wzrost zapotrzebowania na ramki z wszystkimi negatywnymi skutkami, np. zawieszaniem procesów. Zbyt gwałtowana reakcja może spowodować usunięcie stron potrzebnych do dalszego przetwarzania.

Algorytm VSWS (*zbioru roboczego ze zmiennym okresem próbkowania*) przez odpowiednią parametryzację umożliwia lepsze dostosowanie do tego typu sytuacji. Jego istotą jest wcześniejsze lub późniejsze reagowanie na zmieniającą się strefę, zależnie od liczby błędów strony. Czas reakcji jest jednak obustronnie ograniczony, więc reakcja nie będzie ani zbyt szybka (gwałtowna), ani znacząco spóźniona.



Interwał czasu pomiędzy kolejnymi kontrolami należy do przedziału obustronnie domkniętego od T_{min} do T_{max}, przy czym konkretna wartość zależy od momentu osiągnięcia liczby n_{max} błędów strony. Jeśli liczba błędów strony jest duża i osiągnie wartość n_{max} przed upływem czasu T_{min}, to kontrola następuje po czasie T_{min}. Jeśli upłynął czas T_{min}, to kontrola nastąpi zaraz po osiągnięciu n_{max} błędów strony, nie później jednak niż po czasie T_{max}. A więc najpóźniej kontrola wystąpi po czasie T_{max}, niezależnie do liczby błędów strony.



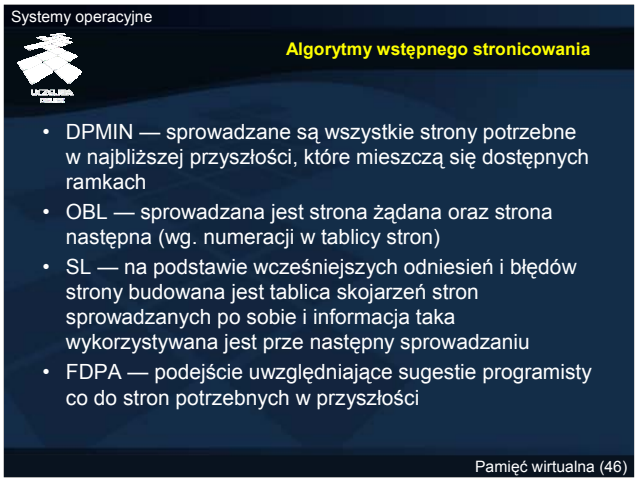
Przedstawione techniki służą do zmniejszenia czasu obsługi błędu strony. Są one szczególnie przydatne w algorytmach ze sprowadzaniem na żądanie.

Jeśli na potrzeby wymiany została znaleziona ramka, zawierająca zmodyfikowaną stronę, a w systemie dostępne są inne wolne ramki, to zamiast wymiany stron, wymagającej czasochłonnego zapisu, dla sprowadzanej strony przydzielana jest nowa ramka. Ramka z brudną stroną dołączana jest natomiast do puli ramek zwolnionych. W ten sposób następuje wymiana ramek zamiast wymiany stron.

Ramki zwolnione w przypadku stosowania algorytmów ze sprowadzaniem na żądanie nie zawsze są natychmiast wykorzystywane na potrzeby innych procesów. Można więc pozostawić zawartość strony bez zmian do czasu przydziału tej ramki. Jeśli jednak przed ponownym przydziałem wystąpi żądanie dostępu do usuniętej strony, to błąd strony można obsłużyć bez wykonywania czasochłonnej operacji wejścia-wyjścia, przydzielając ramkę ze stroną ponownie.

Czasami system jest niedociążony przez bieżące przetwarzanie. Taki chwilowy okres przestoju można wykorzystać do zapisu zmodyfikowanych stron. Kasowany jest wówczas bit modyfikacji i stronę łatwiej jest wymienić.

W przypadku zwalniania ramek stron modyfikowanych operację zapisu w obszarze wymiany można odłożyć w czasie, dołączając ramkę do specjalnej listy. W miarę dostępności zasobów system stopniowo będzie zapisywał strony w ramkach z tej listy w obszarze wymiany i dołączał ramki do puli oczyszczonych wolnych ramek, gotowych w każdej chwili do użycia



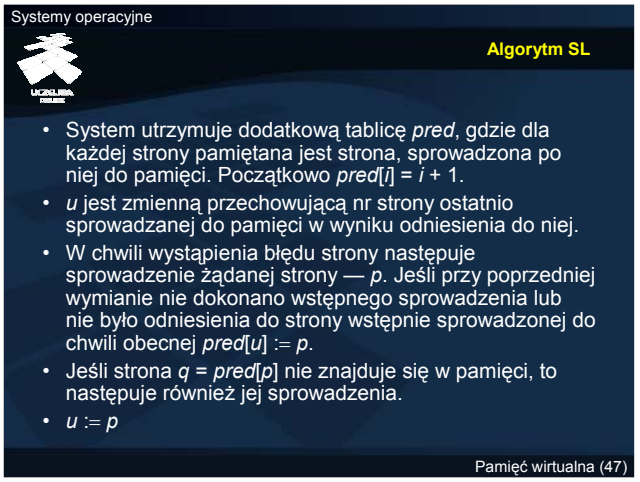
Sens wstępnego sprowadzania wynika z analizy efektywności wymiany, przedstawionej wcześniej. W praktyce, skuteczność tego podejścia zależy od trafności decyzji o wstępnym sprowadzeniu. Na koszt wymiany wpływa również trafność decyzji o usuwaniu stron, ale sprowadzanie wstępne nie jest obligatoryjne, a więc daje potencjalny zysk, jednak obarczony ryzykiem. Przed omówieniem algorytmów warto przypomnieć, że sprowadzanie (również wstępne) przeprowadza się w wyniku wystąpienia błędu strony.

Algorytm DPMIN (ang. Demand Prepaging MIN) — jest optymalny w klasie algorytmów ze sprowadzaniem na żądanie, ale oparty jest (podobnie jak wcześniej przedstawione algorytmy optymalne) na znajomości **przyszłego** ciągu odniesień do stron. W wyniku wystąpienia błędu strony wszystkie dostępne ramki są wypełniane stronami, do których nastąpi odniesienie w najbliższej przyszłości.

Algorytm OBL (ang. One Block Lookahead) wykorzystuje założenie o lokalności przestrzennej odniesień do pamięci i wraz ze stroną, której zaadresowanie spowodowało błąd, sprowadza stronę następną (jeśli nie ma jej jeszcze w pamięci). Skojarzenie stron ma tu charakter statyczny.

Algorytm SL (ang. Spatial Lookahead) również wykorzystuje założenie o lokalności przestrzennej, ale informacja o powiązaniu stron budowana jest w czasie działania algorytmu, ma więc charakter dynamiczny.

Algorytm FDPA (ang. Fixed-space Demand Prepaging) opiera się na wskazówkach programisty lub wnioskach ze statycznej analizy kodu, przekazywanych do systemu poprzez wykonania specjalnych instrukcji. Instrukcje to dotyczą zarówno wskazania stron potrzebnych w przyszłości, jak i zbędnych. Algorytm uwzględnia więc również wskazówki odnośnie usuwania stron.

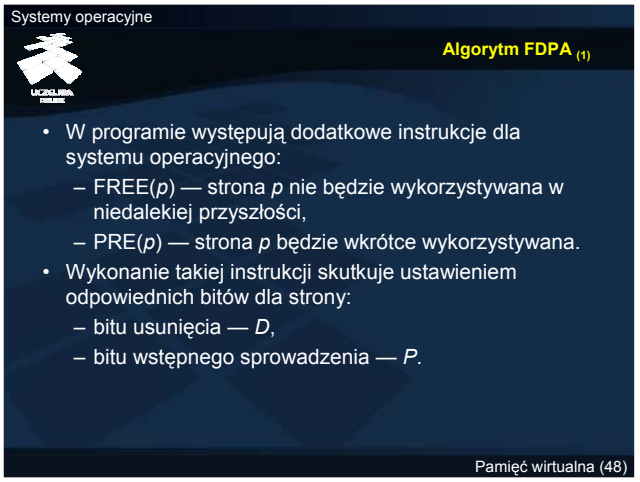


Krótka charakterystyka algorytmów DPMIN i OBL przy opisie wcześniejszego slajdu jest wystarczająca, natomiast omówienia wymaga algorytm SL.

Celem tablicy *pred* jest przechowywanie informacja o numerze strony, którą należy sprowadzić wstępnie w przypadku błędu braku strony. Jeśli więc wystąpi błąd strony przy odniesieniu do strony p, to wstępnie sprowadzona zostanie również strona, której numer jest na pozycji p w tablicy *pred* — *pred*[p]. Początkowo przyjmujemy, że będzie to strona następna, tak, jak w przypadku OBL.

Zmienna u przechowuje numer strony, do której odniesienie spowodowało ostatnio obsłużony błąd strony. Jeśli zatem następny błąd strony występuje przy odwołaniu do strony p (p jest kolejną sprowadzaną stroną po stronie u), to być może jest jakiś związek pomiędzy tymi stronami i w przyszłości również warto po wystąpieniu błędu strony przy dostępie do strony u wstępnie sprowadzić również stronę p. Numer strony — p — powinien zostać zapisany w tablicy *pred* na pozycji odpowiadającej stronie u. Należy jednak upewnić, że dotychczasowe skojarzenie dla strony u jest nieodpowiednie. Jeśli zatem przy sprowadzaniu strony u nie było wstępnego sprowadzenia innej strony lub strona wstępnie sprowadzona nie została użyta, skojarzenie dla u lepiej zmienić.

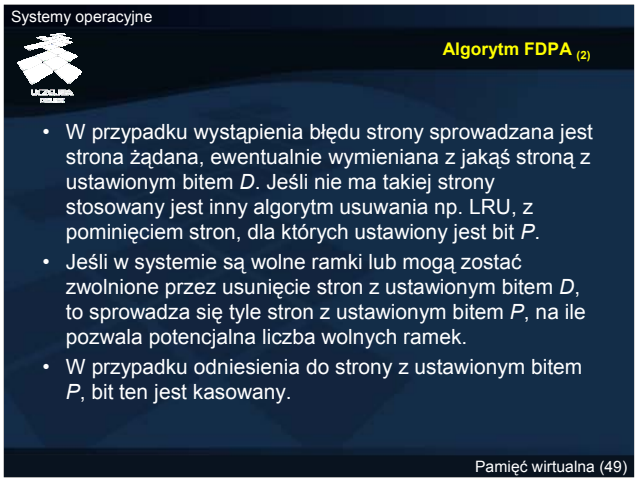
Na końcu następuje wstępne sprowadzenie strony, wynikające z bieżącego skojarzenia dla p. To, czy skojarzenie jest właściwe, okaże się przy wystąpieniu następnego błędu strony. W tym celu wartość p podstawiana jest do zmiennej u.



W wyniku analizy kodu w odpowiednich miejscach w programie pojawiają się instrukcje dla systemu operacyjnego, informujące o przyszłych potrzebach (lub ich braku) w odniesieniu do pewnych stron. Zgodnie z tymi instrukcjami w tablicy stron lub innej strukturze ustawiane są odpowiednie bity. W toku realizacji wymiany bity te są kasowane.

Bit *P* oznacza, że strona będzie w najbliższej przyszłości potrzebna, a jego wykasowanie nastąpi przy odwołaniu do strony po jej sprowadzaniu do pamięci.

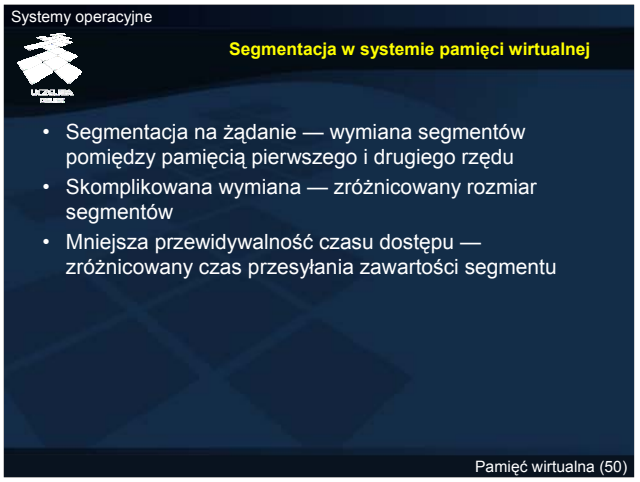
Bit *D* oznacza, że strona nie będzie na razie potrzebna. Gdyby jednak strona została usunięta z pamięci i ponownie sprowadzana w wyniku błędu strony, bit ten zostanie skasowany.



System sprowadza oczywiście strony na żądanie i dokonuje wymiany w przypadku braku wolnych ramek. Usuwane są strony z ustawionym bitem *D*, a gdy takich nie ma, zgodnie z innym algorytmem, np. LRU lub w praktyce z którymś z jego przybliżeń. Algorytm usuwania nie jest tu najbardziej istotny poza preferencją dla stron „wskazanych” przez bit *D*.

Strona z ustawionym bitem *P* może być zarówno w pamięci, jak i poza nią. Będąc poza pamięcią, strona taka czeka na sprowadzenia. Bit *P* ustawiony dla strony przebywającej w pamięci operacyjnej oznacza, że stronę tę wstępnie sprowadzono, ale nie było do niej jeszcze odniesienia, dlatego nie powinna być usunięta przy wymianie.

System stara się sprowadzać do pamięci wszystkie strony z ustawionym bitem *P* w miarę dostępnych ramek. W bilansie ramek uwzględnia się zwolnienie ramek przez strony z ustawionym bitem *D*. Jak już wcześniej wspomniano, gdyby bit D ustawiony był dla strony sprowadzanej, to zostanie wykasowany.



W niektórych systemach (między innymi IBM OS/2) podjęto próbę realizacji segmentacji na żądanie. Zarządzanie wymianą segmentów jest jednak znacznie bardziej skomplikowane, gdyż sprowadzenie segmentu do pamięci może wymagać upakowania i/lub przesunięcia w obszar wymiany jednego lub kilku segmentów znajdujących się w pamięci fizycznej. Optymalizacja decyzji jest w takim przypadku jest więc procedurą czasochłonną. W przypadku stronicowania czas sprowadzenia stron do pamięci oraz zapisu na dysku jest zmienną losową, w przypadku segmentacji, ze względu na zróżnicowany rozmiar segmentów, sam czas przesyłania danych i tym samym czas oczekiwania w kolejce do dysku jest trudniej przewidzieć.