Systemy Operacyjne — Pamięć wirtualna cz. 1

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 6 grudnia 2020

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

1. Wprowadzenie
   1. Zasady lokalności czasowej i przestrzennej
   2. Pamięć wirtualna
   3. Sposoby implementacji pamięci wirtualnej
2. Stronicowanie na żądanie
   1. Podstawy
   2. Wymiana stron
   3. Efektywność
3. Algorytmy wymiany stron
   1. Algorytm FIFO
   2. Algorytm OPT
   3. Algorytm LRU
   4. Algorytmy zbliżone do LRU
   5. Algorytmy ad hoc

Wprowadzenie

Na poprzednim wykładzie poznaliśmy trzy metody pozwalające zmniejszyć zużycie miejsca w pamięci operacyjnej. Tymi metodami były: łączenie dynamiczne, ładowanie dynamiczne i nakładanie. Szczególnie interesujące podejście stosują dwie ostatnie techniki: fragment kodu procesu nie jest wprowadzany do pamięci komputera dopóki nie ma faktycznego zapotrzebowania na jego wykonanie. Dzięki temu fragmenty kodu, które są wykonywane rzadko (czasem wręcz w ogóle nie są wykonywane) nie zajmują miejsca w pamięci. Te fragmenty obejmują najczęściej:

- procedury obsługi wyjątków,
- struktury danych, które nie są w pełni wykorzystywane,
- fragmenty kodu, których wykonanie jest objęte warunkiem, który rzadko jest spełniony.

W każdym z tych podejść to programista aplikacji musiał stworzyć odpowiedni system gospodarowania pamięcią. Możliwe jest jednak całkowite wyeliminowanie takiej potrzeby, polegające na zastosowaniu takiego zarządzania pamięcią operacyjną przez system operacyjny, które uwzględniałoby przedstawioną regułę.

Wprowadzenie

Na poprzednim wykładzie poznaliśmy trzy metody pozwalające zmniejszyć zużycie miejsca w pamięci operacyjnej. Tymi metodami były: łączenie dynamiczne, ładowanie dynamiczne i nakładanie. Szczególnie interesujące podejście stosują dwie ostatnie techniki: fragment kodu procesu nie jest wprowadzany do pamięci komputera dopóki nie ma faktycznego zapotrzebowania na jego wykonanie. Dzięki temu fragmenty kodu, które są wykonywane rzadko (czasem wręcz w ogóle nie są wykonywane) nie zajmują miejsca w pamięci. Te fragmenty obejmują najczęściej:

- procedury obsługi wyjątków,
- struktury danych, które nie są w pełni wykorzystywane,
- fragmenty kodu, których wykonanie jest objęte warunkiem, który rzadko jest spełniony.

W każdym z tych podejść to programista aplikacji musiał stworzyć odpowiedni system gospodarowania pamięcią. Możliwe jest jednak całkowite wyeliminowanie takiej potrzeby, polegające na zastosowaniu takiego zarządzania pamięcią operacyjną przez system operacyjny, które uwzględniałoby przedstawioną regułę.

Wprowadzenie

Na poprzednim wykładzie poznaliśmy trzy metody pozwalające zmniejszyć zużycie miejsca w pamięci operacyjnej. Tymi metodami były: łączenie dynamiczne, ładowanie dynamiczne i nakładanie. Szczególnie interesujące podejście stosują dwie ostatnie techniki: fragment kodu procesu nie jest wprowadzany do pamięci komputera dopóki nie ma faktycznego zapotrzebowania na jego wykonanie. Dzięki temu fragmenty kodu, które są wykonywane rzadko (czasem wręcz w ogóle nie są wykonywane) nie zajmują miejsca w pamięci. Te fragmenty obejmują najczęściej:

- procedury obsługi wyjątków,
- struktury danych, które nie są w pełni wykorzystywane,
- fragmenty kodu, których wykonanie jest objęte warunkiem, który rzadko jest spełniony.

W każdym z tych podejść to programista aplikacji musiał stworzyć odpowiedni system gospodarowania pamięcią. Możliwe jest jednak całkowite wyeliminowanie takiej potrzeby, polegające na zastosowaniu takiego zarządzania pamięcią operacyjną przez system operacyjny, które uwzględniałoby przedstawioną regułę.

Wprowadzenie

Na poprzednim wykładzie poznaliśmy trzy metody pozwalające zmniejszyć zużycie miejsca w pamięci operacyjnej. Tymi metodami były: łączenie dynamiczne, ładowanie dynamiczne i nakładanie. Szczególnie interesujące podejście stosują dwie ostatnie techniki: fragment kodu procesu nie jest wprowadzany do pamięci komputera dopóki nie ma faktycznego zapotrzebowania na jego wykonanie. Dzięki temu fragmenty kodu, które są wykonywane rzadko (czasem wręcz w ogóle nie są wykonywane) nie zajmują miejsca w pamięci. Te fragmenty obejmują najczęściej:

- procedury obsługi wyjątków,
- struktury danych, które nie są w pełni wykorzystywane,
- fragmenty kodu, których wykonanie jest objęte warunkiem, który rzadko jest spełniony.

W każdym z tych podejść to programista aplikacji musiał stworzyć odpowiedni system gospodarowania pamięcią. Możliwe jest jednak całkowite wyeliminowanie takiej potrzeby, polegające na zastosowaniu takiego zarządzania pamięcią operacyjną przez system operacyjny, które uwzględniałoby przedstawioną regułę.

Zasady lokalności

Uważna analiza działania procesu ujawnia, że podlega on dwóm *zasadom lokalności*.

Zasada lokalności przestrzennej

Jeśli nastąpiło odwołanie procesu do pewnej lokacji w pamięci komputera, to następne odwołania z dużym prawdopodobieństwem będą dotyczyły lokacji położonej w pobliżu tej do której nastąpiło to odwołanie.

Zasada lokalności czasowej (temporalnej)

Jeśli nastąpiło odwołanie procesu do określonej lokacji w pamięci komputera, to w niedalekiej przyszłości to odwołanie zostanie wielokrotnie powtórzone.

Jeśli system zarządzania pamięcią uwzględnia te dwie zasady, to w pamięci operacyjnej komputera pozostają tylko informacje, które w danej chwili są niezbędne do działania procesu.

Pamięć wirtualna

Dzięki uwzględnieniu w systemie zarządzania pamięcią przedstawionych wcześniej zasad otrzymujemy szereg udogodnień:

- Pamięć procesu użytkownika staje się teoretycznie nieograniczona.
- W systemach wielozadaniowych następuje zwiększenie stopnia wielozadaniowości - ponieważ procesy zajmują mniej miejsca w pamięci, to można w niej zmieścić większą ich liczbę.
- Zmniejsza się czas trwania operacji wejścia-wyjścia koniecznych do wprowadzenia procesu do pamięci.

Technikę pozwalającą wykonać, przy wsparciu systemu operacyjnego i sprzętu, proces, który tylko częściowo jest załadowany do pamięci nazywamy *pamięcią wirtualną* (ang. virtual memory). Zasadniczą zaletą tego rozwiązania jest odseparowanie pamięci logicznej procesu użytkownika od pamięci fizycznej komputera. Dzięki temu każdy proces dysponuje teoretycznie nieograniczoną pamięcią, a w praktyce może wykonywać się nawet, kiedy ilość fizycznie dostępnej pamięci jest o wiele niższa od jego wymagań.

Pamięć wirtualna

Dzięki uwzględnieniu w systemie zarządzania pamięcią przedstawionych wcześniej zasad otrzymujemy szereg udogodnień:

- Pamięć procesu użytkownika staje się teoretycznie nieograniczona.
- W systemach wielozadaniowych następuje zwiększenie stopnia wielozadaniowości - ponieważ procesy zajmują mniej miejsca w pamięci, to można w niej zmieścić większą ich liczbę.
- Zmniejsza się czas trwania operacji wejścia-wyjścia koniecznych do wprowadzenia procesu do pamięci.

Technikę pozwalającą wykonać, przy wsparciu systemu operacyjnego i sprzętu, proces, który tylko częściowo jest załadowany do pamięci nazywamy *pamięcią wirtualną* (ang. virtual memory). Zasadniczą zaletą tego rozwiązania jest odseparowanie pamięci logicznej procesu użytkownika od pamięci fizycznej komputera. Dzięki temu każdy proces dysponuje teoretycznie nieograniczoną pamięcią, a w praktyce może wykonywać się nawet, kiedy ilość fizycznie dostępnej pamięci jest o wiele niższa od jego wymagań.

Pamięć wirtualna

Dzięki uwzględnieniu w systemie zarządzania pamięcią przedstawionych wcześniej zasad otrzymujemy szereg udogodnień:

- Pamięć procesu użytkownika staje się teoretycznie nieograniczona.
- W systemach wielozadaniowych następuje zwiększenie stopnia wielozadaniowości - ponieważ procesy zajmują mniej miejsca w pamięci, to można w niej zmieścić większą ich liczbę.
- Zmniejsza się czas trwania operacji wejścia-wyjścia koniecznych do wprowadzenia procesu do pamięci.

Technikę pozwalającą wykonać, przy wsparciu systemu operacyjnego i sprzętu, proces, który tylko częściowo jest załadowany do pamięci nazywamy *pamięcią wirtualną* (ang. virtual memory). Zasadniczą zaletą tego rozwiązania jest odseparowanie pamięci logicznej procesu użytkownika od pamięci fizycznej komputera. Dzięki temu każdy proces dysponuje teoretycznie nieograniczoną pamięcią, a w praktyce może wykonywać się nawet, kiedy ilość fizycznie dostępnej pamięci jest o wiele niższa od jego wymagań.

Pamięć wirtualna

Dzięki uwzględnieniu w systemie zarządzania pamięcią przedstawionych wcześniej zasad otrzymujemy szereg udogodnień:

- Pamięć procesu użytkownika staje się teoretycznie nieograniczona.
- W systemach wielozadaniowych następuje zwiększenie stopnia wielozadaniowości - ponieważ procesy zajmują mniej miejsca w pamięci, to można w niej zmieścić większą ich liczbę.
- Zmniejsza się czas trwania operacji wejścia-wyjścia koniecznych do wprowadzenia procesu do pamięci.

Technikę pozwalającą wykonać, przy wsparciu systemu operacyjnego i sprzętu, proces, który tylko częściowo jest załadowany do pamięci nazywamy *pamięcią wirtualną* (ang. virtual memory). Zasadniczą zaletą tego rozwiązania jest odseparowanie pamięci logicznej procesu użytkownika od pamięci fizycznej komputera. Dzięki temu każdy proces dysponuje teoretycznie nieograniczoną pamięcią, a w praktyce może wykonywać się nawet, kiedy ilość fizycznie dostępnej pamięci jest o wiele niższa od jego wymagań.

Implementacja pamięci wirtualnej

Najprostszym, najpowszechniej stosowanym, a jednocześnie najskuteczniejszym sposobem implementacji pamięci wirtualnej jest *stronicowanie na żądanie* (ang. demand paging). Z poznanych na poprzednim wykładzie technik zarządzania pamięcią do implementacji pamięci wirtualnej nadają się jeszcze segmentacja i segmentacja stronicowana, ale ze względu na małą popularność takich implementacji ich użycie będzie omówione tylko pobieżnie w następnej części wykładu.

Podstawy

Aby stronicowanie mogło służyć do implementacji pamięci wirtualnej, należy je przekształcić w stronicowanie na żądanie. Zasadnicza różnica między tymi technikami polega na sposobie ładowania procesu do pamięci. W stronicowaniu, zanim proces mógł się rozpocząć, wszystkie jego strony musiały być załadowane do pamięci operacyjnej. Stronicowanie na żądanie stosuje technikę *leniwej wymiany* (ang. lazy swapper), tzn. wprowadza do pamięci operacyjnej stronę, dopiero wtedy, gdy nastąpi do niej odwołanie. Dzięki temu proces zajmuje niewielką ilość pamięci, jest tylko częściowo załadowany, ale jak wynika to z reguł lokalności czasowej i przestrzennej może się wykonywać. Ta technika wymaga wzbogacenia każdej pozycji w tablicy stron o dodatkowy bit, nazywany *bitem poprawności odniesienia*, który sygnalizuje, czy strona do której odwołuje się proces jest załadowana do pamięci operacyjnej. Jeśli nastąpi odwołanie do strony, której nie ma w pamięci fizycznej, to generowany jest wyjątek nazywany *błędem strony* (ang. page fault). Przyczyny tego błędu mogą być dwie: proces odwołuje się do nieistniejącej strony i powinien zostać zakończony lub proces odwołuje się do strony, która istnieje, ale zamiast w pamięci operacyjnej znajduje się na dysku i powinna być do niej sprowadzona. Należy zauważyć, że proces może zacząć się wykonywać, mając wyłącznie puste ramki, bez żadnej strony. To wykonanie rozpocznie się oczywiście błędem strony, którego obsługa spowoduje wprowadzenie do pamięci potrzebnej strony. Ta technika nazywa się *czystym stronicowaniem na żądanie*. Stronicowanie na żądanie wymaga, aby w pamięci masowej (najczęściej na dysku) został wydzielony obszar nazywany *obszarem wymiany* lub *przestrzenią wymiany*. Jest to plik lub partycja, które są zoptymalizowane pod względem przechowywania stron procesu i nazywane odpowiednio *plikiem wymiany* lub *partycją wymiany*. Urządzenie pamięci masowej, na którym osadzone są takie struktury nazywa się *urządzeniem stronicującym* lub *pamięcią pomocniczą*.

Obsługa błędu strony

Błąd strony jest obsługiwany przez odpowiednią procedurę obsługi przerwania (wyjątku). Jednakże inne elementy systemu operacyjnego także wykonują dodatkowe czynności związane z obsługą tego błędu:

1. zachowanie stanu bieżącego procesu,
2. sprawdzenie poprawności adresu, który wygenerował wyjątek (jeśli adres był nieprawidłowy to kończone jest wykonanie procesu),
3. rozpoczęcie wykonania operacji wejścia-wyjścia, której celem jest załadowanie odpowiedniej strony do wolnej ramki w pamięci operacyjnej,
4. (nieobowiązkowo) przydzielenie procesora innemu procesowi na czas oczekiwania na zrealizowanie transmisji,
5. obsługa przerwania od dysku twardego, sygnalizującego zakończenie operacji sprowadzania strony do pamięci,
6. uaktualnienie tablicy stron,
7. oczekiwanie na przydzielenie procesowi dla którego została sprowadzona strona procesora,
8. wykonanie przerwanego przez błąd strony rozkazu.

W zależności od sposobu implementacji stronicowania na żądanie lista tych czynności może być uzupełniona o inne działania.

Wymiana stron

Każdy proces otrzymuje określoną liczbę ramek w pamięci fizycznej. Dopóki są wolne ramki, dopóty można sprowadzać nowe strony do pamięci operacyjnej. Co jednak stanie się, kiedy wolnych ramek zabraknie? Jednym z rozwiązań jest na pewno zawieszenie wykonania procesu, który nie ma wolnej ramki do której można by załadować żądaną przez niego stronę. Istnieje jednak inne, korzystniejsze i częściej stosowane rozwiązanie. Jest nim *wymiana stron*. Polega ona na odnalezieniu strony, co do której istnieje podejrzenie, że nie będzie już używana, albo nie będzie używana w najbliższym czasie, wysłaniu jej do przestrzeni wymiany i sprowadzeniu w jej miejsce żądanej strony. Należy więc uzupełnić scenariusz obsługi błędu strony o następujące czynności:

1. Jeśli nie istnieje wolna ramka, należy wytypować ramkę-ofiarę.
2. Strona-ofiara jest zapisywana na dysku i aktualizowana jest tablica stron.
3. Do zwolnionej ramki wczytywana jest żądana strona.

Do wytypowania ramki-ofiary należy zastosować możliwie najbardziej skuteczny algorytm, który będzie działał szybko i w miarę precyzyjnie typował strony, które będą nieużywane. Algorytmy wymiany stron zostaną zaprezentowane w dalszej części wykładu.

Efektywność stronicowania na żądanie

Ponieważ obsługa wystąpienia błędy strony trwa stosunkowo długo, można się zastanowić jak wpływa ona na czas działania procesu, a w szczególności jak wpływa na czas dostępu do pamięci. Efektywny czas dostępu do pamięci w stronicowaniu na żądanie możemy wyrazić następującym wzorem:

$$t_{ema} = (1 - p) \cdot t_{ma} + p \cdot t_{pfh},$$

gdzie p-prawdopodobieństwo wystąpienia błędu, t_{ma} czas dostępu do pamięci, t_{pfh} czas obsługi błędu strony. Na czas obsługi błędu strony składają się czasy wykonania wszystkich czynności wymienionych na planszy pt. „Obsługa błędu strony". Ogólnie czas ten jest długi, dlatego też dąży się nie tylko do jego skrócenia, ale również do zminimalizowania liczby błędów strony. Na tę ostatnią wielkość mają wpływ takie czynniki, jak liczba ramek, które zostały procesowi przydzielone i sprawność algorytmu wymiany. Problemem przydziału ramek zajmiemy się w drugiej części wykładu, teraz warto jednak wspomnieć, że aby proces w ogóle mógł się wykonać, musi w pamięci rezydować zawsze tyle ramek ile wymaga najdłuższy rozkaz procesora. Istnieją dodatkowe również wymagania nakładane na listę rozkazów i działanie procesora. Ponieważ błąd strony jest przerwaniem nieprecyzyjnym, to należy ponowić wykonanie rozkazu, który on przerwał. Może to być trudne, jeśli rozkaz stosuje tryb adresowania z preautoinkrementacją lub preautodekrementacją. Również wielokrotny tryb pośredni nie jest wskazany, gdyż wymaga obecności w pamięci dużej liczby stron. Generalnie rozkazy, które bezpośrednio modyfikują zawartość w pamięci operacyjnej są problematyczne w stronicowaniu na żądanie.

Efektywność stronicowania na żądanie

Opisując efektywność stronicowania na żądanie należy wspomnieć też o fragmentacji zewnętrznej, która występuje na poziomie nie poszczególnych lokacji pamięci, ale całych stron. Nie ma ona znaczenia dla aplikacji użytkownika, które posługują się adresami wirtualnymi (tak w przypadku pamięci wirtualnej nazywa się adresy logiczne). Stanowi ona za to problem dla urządzeń korzystających z transmisji DMA, ponieważ one posługują się wyłącznie adresami fizycznymi i pamięć na bufory dla nich musi być przydzielana w sposób ciągły. Nowsze systemy komputerowe wyposażone są w jednostkę IOMMU, która przeznaczona jest dla urządzeń wejścia-wyjścia i eliminuje konieczność przydzielania pamięci operacyjnej na bufory w sposób ciągły.

Algorytmy wymiany-wprowadzenie

Wybierając algorytm wymiany stron zależy nam na tym, aby generował on jak najmniejszą liczbę błędów strony. Implementację takiego algorytmu można przetestować tworząc za pomocą generatora liczb pseudolosowych *ciąg odwołań* (ang. reference string), czyli ciąg numerów stron, do których hipotetyczny proces mógłby się odwoływać. Należy również założyć pewną liczbę wolnych ramek, którymi będzie dysponował ten proces. Najczęściej działanie algorytmu bada się dla kilku różnych wartości tego czynnika, co pozwala sprawdzić, czy algorytm zachowuje się poprawnie, tzn. czy wraz ze wzrostem liczby ramek maleje liczba błędów stron.

Algorytm FIFO

Algorytm FIFO jest najprostszym algorytmem wymiany stron, zawsze zastępuje tę stronę, która przebywa najdłużej w pamięci operacyjnej. Niestety takie działanie nie gwarantuje, że strona wymieniana nie będzie w najbliższym czasie potrzebna, dlatego algorytm FIFO generuje dużą liczbę błędów stron, a dodatkowo obciążony jest *anomalią Belady'ego*, tzn. wraz ze wzrostem liczby ramek może wzrastać liczba błędów stron. Działanie tego algorytmu jest przedstawione na następnej planszy. Górny wiesz tabeli, to ciąg odwołań, a trzy kolejne symbolizują ramki. Wystąpienie błędu strony sygnalizowane jest żółtym tłem kolumny tabeli.

Algorytm FIFO

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1

Razem: 15 błędów stron

Algorytm optymalny

Dla zagadnienia wymiany stron istnieje algorytm optymalny, czyli taki, który powoduje najmniejszą liczbę wymian stron, a zarazem najmniejszą liczbę błędów stron. Ten algorytm oznaczany jest skrótem OPT lub MIN. Jego działanie można scharakteryzować jednym zdaniem: „Wymień tę stronę, która najdłużej nie będzie używana". W praktyce jednak nigdy tego algorytmu się nie stosuje, ponieważ nie można go zaimplementować. Nie istnieje żaden stuprocentowo pewny sposób na określenie, która ze stron będzie najdłużej niepotrzebna. Możemy jednak porównywać rzeczywiste algorytmy z algorytmem OPT i badać w jakiej skali go przybliżają. Działanie tego algorytmu zostanie zaprezentowane na następnej planszy.

Algorytm optymalny

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1

Razem: 9 błędów stron

Algorytm LRU

Algorytm LRU (ang. Least Recently Used) wymienia tą stronę, która najdawniej była używana. Po analizie działania tego algorytmu można stwierdzić, że stanowi on w pewnym sensie odwrotność działania algorytmu OPT. Używając języka potocznego, można napisać, że algorytm OPT „patrzy w przyszłość", żeby znaleźć stronę do wymiany, a algorytm LRU „patrzy w przeszłość". Algorytm LRU jest najpopularniejszym algorytmem stosowanym do wymiany stron. Efektywność jego działania jest zbliżona do efektywności algorytmu OPT. Następna plansza zawiera ilustrację działania tego algorytmu.

Algorytm LRU

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7

Razem: 12 błędów stron

Implementacja LRU

Implementacja algorytmu LRU jest dosyć trudna i może wymagać wsparcia ze strony sprzętu. Aby określić, która ze stron była najdawniej używana stosuje się liczniki, albo stos. Zastosowanie liczników polega na wyposażeniu każdej pozycji tablicy stron w miejsce na wartość zegara logicznego. Wartość tego zegara jest zwiększana przy każdym odwołaniu do strony i jednocześnie zapisywana w miejscu tablicy stron, które jest związane ze stroną do której nastąpiło to odwołanie. Porównując te wartości dla każdej strony możemy sprawdzić, do której odwoływano się ostatnio. Problemem może być powstanie nadmiaru w zegarze logicznym. Stos zawiera numery stron do których się odwoływał proces ułożone na nim w ten sposób, że numer strony, która została ostatnio użyta jest na szczycie tego stosu, a numer strony, która najdawniej była nieużywana jest na jego dnie. Liczba elementów na stosie jest równa liczbie ramek. Ten stos jest najczęściej implementowany w postaci listy dwukierunkowej, której obsługa jest wspomagana sprzętowo. Zastosowanie stosu w implementacji algorytmu LRU pozwoliło określić klasę algorytmów nazywanych *algorytmami stosowymi*, które nie prowadzą do anomalii Beladyego. Algorytm stosowy to taki algorytm dla którego zbiór stron obecnych w pamięci przy n dostępnych ramkach jest podzbiorem zbioru stron obecnych w pamięci, gdyby było dostępnych $n+1$ ramek.

Algorytmy zbliżone do LRU

Jeśli w systemie nie ma odpowiednich środków sprzętowych do realizacji algorytmu LRU, to można zastosować metodę, która będzie dawała rezultaty zbliżone do rezultatów tego algorytmu. Na następnych planszach zostanie opisanych kilka takich algorytmów.

Algorytm dodatkowych bitów odniesienia

Wiele systemów stosuje *bity odniesienia*, które są ustawiane dla stron do których następowało odwołanie. Zamiast pojedynczego bitu można zastosować w tablicy stron cały bajt. Co określony czas system operacyjny modyfikuje wartości tych bajtów, przesuwając ich zawartość o jedno miejsce w prawo i ustawiając na najstarszym bicie jedynkę, jeśli ostatnio nastąpiło odniesienie do strony lub zero jeśli tego odniesienia nie było. Interpretując powstały w ten sposób wzorzec binarny jako liczbę naturalną można określić kolejność odwołań do stron.

Algorytm drugiej szansy

Mając tylko jeden bit odniesienia można zastosować algorytm drugiej szansy, nazywany również algorytmem zegarowym. W tym algorytmie przeszukiwana jest tablica stron w poszukiwaniu strony, która ma wyzerowany bit odniesienia. Jeśli strona ma bit odniesienia ustawiony na jeden, to algorytm go ustawia na zero, ale nie wymienia tej strony dając jej drugą szansę na pozostanie w pamięci operacyjnej. Jeśli jednak przy kolejnym wykonaniu tego algorytmu bit odniesienia tej strony będzie równy zero, to strona ta zostanie wymieniona.

Algorytmy LFU i MFU

Algorytm LFU (ang. Least Frequently Used) wymienia te strony, do których najrzadziej się odwoływano. Aby określić częstotliwość odwoływania się do konkretnej strony, z każdym elementem tablicy stron związany jest licznik odwołań. Odwrotnie działa algorytm MFU (ang. Most Frequently Used) - wymienia on tę stronę, do której najczęściej się odwoływano, wychodząc z założenia, że nie będzie już potrzebna. Oba algorytmy są rzadko stosowane, bo nie przybliżają dobrze algorytmu OPT.

Bit odniesienia i bit modyfikacji

Jeśli tablica stron wyposażona jest w *bit modyfikacji* sygnalizujący, czy do danej strony został wykonany zapis, to można w połączeniu z bitem odniesienia wykorzystać go do określenia przydatności strony do wymiany. Możemy wyróżnić cztery klasy stron, w zależności od ustawienia tych bitów (pierwszy jest bitem odniesienia, drugi modyfikacji):

- **(0,0)** - strona nie była używana ostatnio i nie jest zmieniona, idealna kandydatka do wymiany, nie trzeba jej nawet zapisywać do przestrzeni wymiany,
- **(0,1)** - strona nie była używana ostatnio, a więc może być wymieniona, ale trzeba ją zapisać do pamięci pomocniczej, bo jej stan uległ zmianie,
- **(1,0)** - strona używana, ale nie zmieniona, może być potrzebna, ale ewentualna jej wymiana nie wymagałaby zapisu jej zawartości na dysk,
- **(1,1)** - strona używana i zmieniona, najgorsza kandydatka do wymiany

Wymianę stron zaczyna się od tych, które należą do pierwszej klasy, jeśli nie ma takowych, to brane są pod uwagę strony z następnych klas.

Algorytmy ad hoc

Dosyć często opisane wcześniej algorytmy są „wzbogacane" o dodatkowe elementy usprawniające ich działanie. Takim elementem może być stała pula wolnych ramek. Kiedy trzeba wymienić stronę, to typowana jest ramka-ofiara, ale stronę umieszcza się w pierwszej wolnej ramce i wznawia się wykonanie procesu. Kiedy urządzenie stronicujące nie ma innych zleceń, to zapisuje stronę z ramki-ofiary, a ramka trafia do puli ramek wolnych. Inne rozwiązanie polega na utrzymywaniu listy stron zmodyfikowanych i sukcesywnym ich zapisywaniu do przestrzeni wymiany, jeśli urządzenie stronicujące nie ma innych zleceń. Jeszcze inne rozwiązanie polega na utrzymywaniu puli wolnych ramek, wraz z informacją, które strony zawierały te ramki. Ponieważ do czasu modyfikacji, zawartość żadnej z tych ramek nie ulegnie zmianie, to jeśli nastąpi odwołanie do strony, która w takiej ramce się znajdowała, będzie łatwo tę stronę „odzyskać".

Pytania

?

Koniec

Dziękuję Państwu za uwagę!