Ćw 4 - FrameAllocation

sobota, 23 marca 2024 13:27

Postępująca komplikacja zad. 3.
- Założyć, że: w systemie działa pewna ilość (rzędu ~10) procesów,
- każdy korzysta z własnego zbioru stron (zasada lokalności wciąż obowiązuje),
- globalny ciąg odwołań jest wynikiem połączenia **sekwencji odwołań** generowanych przez poszczególne procesy (każdy generuje ich wiele, nie jedną),
- zastępowanie stron odbywa się zgodnie z LRU, każdemu system przydziela określoną liczbę ramek, na podstawie następujących metod:
    1. Przydział proporcjonalny
    2. Przydział równy
    3. Sterowanie częstością błędów strony
    4. Model strefowy

• Jak strategie przydziału ramek wpływają na wyniki (***liczbę braków strony*** - globalnie, dla każdego procesu)?
• Program powinien wypisywać na ekranie przyjęte założenia symulacji.

Wymagana możliwość sterowania założeniami symulacji (na poziomie kodu programu).

Najprostszym, najpowszechniej stosowanym, a jednocześnie najskuteczniejszym sposobem implementacji pamięci wirtualnej jest stronicowanie na żądanie (ang. demand paging).

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.

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

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

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

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:

- Jeśli nie istnieje wolna ramka, należy wytypować ramkę-ofiarę.
- Strona-ofiara jest zapisywana na dysku i aktualizowana jest tablica stron.
- 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.

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 Belady'ego. 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.

Przydział ramek

Wstęp

W poprzedniej części wykładu ustaliliśmy, że wpływ na efektywność stronicowania na żądanie, oprócz algorytmu wymiany stron, może mieć również metoda przydziału wolnych ramek procesom. Kiedy w systemie pracują tylko dwa procesy, system operacyjny i proces użytkownika, to przydział ramek jest prosty. Podział zbioru wolnych ramek jest dokonywany tak, aby proporcja była korzystna dla procesu użytkownika (w końcu systemy komputerowe są tworzone po to by wykonywać procesy użytkownika, a system operacyjny ma tylko nimi zarządzać i ułatwiać ich pracę). Jeśli się skończy pula wolnych ramek, to strony procesu użytkownika lub systemu operacyjnego podlegają wymianie. Możliwe są również scenariusze, w których system przekazuje część swoich ramek na rzecz procesu użytkownika (np. ramek przeznaczonych na bufory, które aktualnie nie są używane) lub w których system utrzymuje zawsze pewną liczbę wolnych ramek, aby usprawnić proces wymiany stron. Problem przydziału ramek komplikuje się, gdy mamy do czynienia z systemem wielozadaniowym.

Minimalna liczba ramek

Aby proces mógł wykonać choć jeden rozkaz, to w pamięci komputera muszą się znajdować jednocześnie wszystkie strony, których ten rozkaz może dotyczyć. Ponieważ nie można określić jaki to będzie rozkaz przed załadowaniem programu do pamięci, to system operacyjny zakłada najgorszy scenariusz i przydziela tyle ramek, aby mogły się w nich zmieścić wszystkie strony konieczne do wykonania najbardziej złożonego rozkazu na liście rozkazów procesora. Przez złożoność rozumiemy tu złożoność jego trybu adresowania. Najbardziej korzystnie pod tym względem wypadają procesory RISC, w których istnieją zazwyczaj tylko proste tryby adresowania, najgorzej procesory, które pozwalają na wielopoziomowe adresowanie pośrednie (należy ograniczyć liczbę poziomów do pewnej liczby, po której przekroczeniu generowany jest wyjątek).

Algorytmy przydziału ramek

W systemach wielozadaniowych możliwych jest wiele strategii przydziału programom użytkownika wolnych ramek. Najprostszy jest przydział równy w którym każdemu procesowi przydziela się taką samą liczbę ramek. Liczba ta jest określona wzorem m/n, gdzie m jest całkowitą liczbą wolnych ramek, a n jest liczbą procesów. Ponieważ każdy z nich ma inne zapotrzebowanie na pamięć, taki przydział nie jest optymalny. Bardziej sprawiedliwy jest przydział proporcjonalny, który uwzględnia rozmiar każdego z procesów. Załóżmy, że si jest wielkością pamięci wirtualnej procesu pi . Sumaryczny rozmiar pamięci wirtualnej wszystkich procesów jest zatem równy $S = \Sigma_i s_i$ Liczba ramek przydzielona programowi pi jest określona wzorem $a_i \approx s_i / S \cdot m$. Ta wartość musi zostać zaokrąglona do najbliższej wartości całkowitej, większej od minimalnej liczby ramek, jaką należy procesowi przydzielić. Metody wywodzące się z przydziału proporcjonalnego oprócz rozmiaru procesu mogą uwzględniać też inne jego właściwości, np. priorytet.

Szamotanie - definicja i przyczyna

W systemie, w którym procesom są przyporządkowane priorytety można pozwolić, aby proces o wysokim priorytecie odbierał w razie potrzeby ramki przydzielone procesom o niższym priorytecie. Przy takim rozwiązaniu, jeśli liczba ramek, którą dysponuje proces niskopriorytetowy spadnie poniżej określonej wartości, to proces ten zaczyna intensywnie wymieniać swoje strony. Jeśli proces zużywa więcej czasu na wymianę stron niż na wykonanie to mówimy o zjawisku szamotania (ang. trashing). Szczególnie intensywnie to zjawisko przebiegało w pierwszych systemach, które stosowały równocześnie planowanie długoterminowe i stronicowanie na żądanie. Szamotanie choć jednego z procesów powoduje zwiększenie obciążenia urządzenia wymiany, a zmniejszenie obciążenia procesora. Ponieważ zadaniem planisty długoterminowego jest dbanie o to, aby procesor był maksymalnie wykorzystany, to żeby zwiększyć jego obciążanie wprowadzał on do pamięci nowy proces. Ten nowy proces otrzymywał do dyspozycji ramki, które zabierane były procesom już rezydującym w pamięci. To z kolei powodowało zwiększenie intensywności szamotania procesów, które obejmowało do tej pory to zjawisko i doprowadzenie do niego innych procesów. W efekcie planista długoterminowy uzyskiwał efekt odwrotny do zamierzonego: małe obciążenie procesora, a duże jednostki stronicującej.

Zbiór roboczy

Aby wyeliminować szamotanie należy zadbać o to, by proces zawsze dysponował wystarczającą liczbą ramek, aby pomieścić jednocześnie w pamięci wszystkie strony niezbędne w danej chwili do jego działania. Wykonanie procesu podlega modelowi strefowemu. Strefą nazywamy zbiór stron, które pozostają we wspólnym użyciu (muszą razem być obecne w pamięci operacyjnej). Taką strefę mogą tworzyć np. strony na których umieszczony jest jakiś podprogram lub jakaś struktura danych. Program składa się wielu stref, niekoniecznie rozłącznych. Podczas wykonania procesu sterowanie przechodzi od bieżącej strefy do następnej. Model zbioru roboczego jest ściśle powiązany z modelem stref. Przyjmuje się pewien parametr Δ, który określa szerokość okna zbioru roboczego, czyli liczbę ostatnich odniesień procesu do stron, które powinien system operacyjny zapamiętać. Zbiorem roboczym (ang. working set) nazywamy wszystkie strony do których nastąpiło ostatnich Δ odniesień. Należy zadbać o to by każdy proces miał tyle ramek, aby mógł utrzymać w pamięci swój zbiór roboczy. Problemem jest dobranie takiego parametru Δ aby model zbioru roboczego działał. Do szamotania nie dojdzie, jeśli rozmiar sumy zbiorów roboczych WSSi wszystkich procesów będzie zawsze mniejszy od liczby dostępnych ramek D, co można zapisać nierównością $D > \Sigma i\ WSSi$.

Częstotliwość błędów strony

Prostszym sposobem niedopuszczania do zaistnienia szamotania jest monitorowanie częstotliwości występowania błędów stron we wszystkich procesach. Zakładamy przy tym pewną maksymalną i minimalną wartość graniczną. Jeśli któryś z procesów wykazuje za mało błędów stron, to część ramek jest mu odbierana i przekazywana procesowi, który wykazuje zbyt dużą liczbę błędów stron.

Przydział lokalny i globalny

Istnieją dwie strategie przeprowadzania wymiany strony dla pojedynczego procesu wykonywanego w systemie wielozadaniowym. Pierwsza, nazywana wymianą globalną, polega na objęciu algorytmem wymiany wszystkich ramek w pamięci fizycznej. Oznacza to, że proces (a właściwie system operacyjny w imieniu procesu) może odebrać ramkę innemu procesowi i umieścić w niej swoją stronę. Druga strategia jest nazywana wymianą lokalną. W tym przypadku system operacyjny wymienia strony należące jedynie do procesu, który wykazał błąd strony. Dodatkowo, liczba ramek przydzielonych procesowi nie ulega zmianie, jeśli w systemie nie ma wolnych ramek. Wymiana lokalna ogranicza nasilanie zjawiska szamotania, ale z kolei wymiana globalna daje lepszą przepustowość systemu i pozwala dostosować liczbę ramek procesu do jego zbioru roboczego.

Stronicowanie wstępne

Jeśli proces ulega szamotaniu, to może być całkowicie wycofany z pamięci operacyjnej do przestrzeni wymiany, do momentu aż w pamięci głównej pojawi się odpowiednia liczba ramek pozwalająca na jego prawidłowe wykonanie. Aby program nie był sprowadzany do pamięci strona po stronie, tak jak ma to miejsce na początku jego wykonywania, to można zastosować stronicowanie wstępne, czyli wprowadzić do pamięci wszystkie jego strony, które zostały wycofane. Skuteczność tej techniki zależy od tego ile z tych stron będzie przydatnych podczas dalszej pracy procesu. Jeśli zbyt mało, to nie opłaca się tej techniki stosować.

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

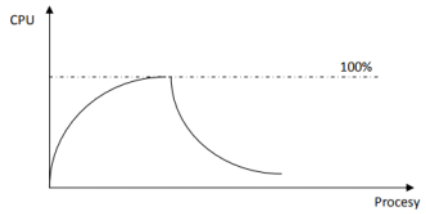
Jedną z metod zapobiegania szamotaniu jest wyznaczanie tzw. *zbiorów roboczych* procesów. Zbiór roboczy, to zbiór stron, do których nastąpiło odwołanie w ciągu ostatnich I instrukcji (dla pewnej ustalonej stałej I, np. I = 100 000). Zbiór roboczy ma przybliżać strefę, w której znajduje się program. Należy 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.

ALGORYTMY PRZYDZIAŁU RAMEK

Algorytmy odpowiadają na pytanie, ile miejsca powinna dostać aplikacja ładowana do pamięci. Jednak nie można odpowiedzieć na to pytanie. Są 2 rozwiązania:

- **Przydział równy** – każdy proces dostaje tę samą ilość ramek
- **Przydział proporcjonalny** – każdemu procesowi przydziela się dostępną pamięć odpowiednio do jego rozmiaru



Istnieje limit podziału. W momencie, gdy odwołań jest coraz więcej, SO musi czekać na HDD. Aplikacji jest coraz więcej, ale procesor nic nie robi, nazywamy to **szamotaniem**.

SPOSOBY LIKWIDACJI SZAMOTANIA

1. **Model strefowy** – służy do zapewnienia aplikacji tylu ramek, ile potrzebuje; minimalizuje to szanse na wystąpienie błędu strony.

Strony 1, 2, 7, 5, 5, 8, 3, 9, 10, ... →

Δ – okno zbioru roboczego, ilość ostatnich odwołań do stanu, które będziemy rozpatrywać
Z_R – zbiór roboczy aplikacji lub procesu

Δ = 6
Z_R = {1, 2, 5, 7} -> | Z_R | = 4

Wielkość | Z_R | jest prawie o połowę mniejsza od Δ -> na tej podstawie SO przydziela 4 ramki.

2. **Strategia sterowania częstością błędu strony.**

Błędy ↑

1) **Przydział równy**

Najprostszy sposób statycznego przydziału ramek. Niech N będzie liczbą procesów działających w systemie, a F liczbą ramek dostępnych w systemie.

$$f_i = \frac{F}{N}$$

f_i – liczba ramek przydzielona procesowi p_i

2) **Przydział proporcjonalny**

Przydział proporcjonalny polega na przydziale procesowi p_i liczba ramek f_i proporcjonalna do liczby używanych przez niego stron s_i.

$$S = \sum_{i=1}^{N} s_i$$

$$f_i = \frac{s_i}{N}$$

3) **Sterowanie częstością błędów strony**

Algorytm sterowania częstością błędów strony jest algorytmem dynamicznego przydziału ramek do procesu. Należy tutaj przyjąć 2 graniczne progi częstości błędów stron *l* oraz *u*, a samą częstość błędów (*PPF*, ang. page fault frequency) wyznaczymy:

$$PPF_i = \frac{e_i}{\Delta t}$$

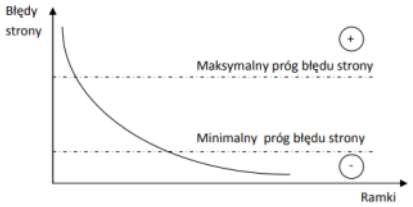
Gdzie:

- e_i - liczba błędów strony generowanych przez proces p_i
- Δt – przyjęte okno czasowe dla pomiaru PPF

Na początku działania algorytmu przydzielamy każdemu procesowi ramki z użyciem standardowego algorytmu przydziału proporcjonalnego. W trakcie działania systemu (symulacji) monitorujemy współczynnik *PPF* dla każdego procesu, jeżeli dojdzie do sytuacji, że *PPF* przekorczy próg *u* dany proces otrzymuje dodatkową ramkę. W przypadku spadku *PPF* poniżej progu *l* dany proces zwalnia jedną z ramek. W przypadku, gdy nie ma dostępnych wolnych ramek dany proces jest wstrzymywany. Do algorytmu sterowania częstością błędów strony można wprowadzić

Wielkość | Z_n | jest prawie o połowę mniejsza od Δ -> na tej podstawie SO przydziela 4 ramki.

2. Strategia sterowania częstością błędu strony.



SO oddaje ramki do procesu, gdy przekroczy maksymalny próg. Natomiast procesowi, który przekroczy minimalny próg zabiera ramki.

standardowego algorytmu przydziału proporcjonalnego. W trakcie działania systemu (symulacji) monitorujemy współczynnik *PPF* dla każdego procesu, jeżeli dojdzie do sytuacji, że *PPF* przekorczy próg *u* dany proces otrzymuje dodatkową ramkę. W przypadku spadku *PPF* poniżej progu *l* dany proces zwalnia jedną z ramek. W przypadku, gdy nie ma dostępnych wolnych ramek dany proces jest wstrzymywany. Do algorytmu sterowania częstością błędów strony można wprowadzić modyfikację polegającą na akceptowaniu przekroczenia progu *u* przez wartość PPF, utrzymaniu aktywności procesu, a wstrzymywanie go dopiero po przekroczeniu pewnej wartości *h* (odpowiednio wysokiej), która określa, kiedy należy proces wstrzymać. W tym wariancie, procesowi przydzielamy dodatkowe ramki po przekroczeniu *u* ale wstrzymujemy dopiero przy przekroczeniu *h*.

Istotne dla symulacji tego algorytmu jest dopieranie odpowiednich wartości progowych oraz okna czasu.

4) Model strefowy

Model strefowy wykorzystuje koncepcję zbioru roboczego, który jest związany z występowaniem zjawiska lokalności odwołań. Ponownie przyjmujemy Δt, tym razem do wyznaczenia rozmiaru zbioru roboczego (*WSS*, ang. working set size). *WSS* jest liczbą stron, do których proces p_i wygenerował odwołania w czasie Δt. Liczbę aktualnie potrzebnych ramek *D* możemy wyznaczyć:

$$D = \sum_{i=1}^{N} WSS_i$$

Dopóki *D* jest mniejsze niż liczba dostępnych w systemie ramek każdy z procesów otrzymuje do wykorzystania tyle ramek, ile wynosi jego *WSS*. W momencie przekroczenia liczby dostępnych ramek przez współczynnik *D* jeden z aktywnych procesów musi zostać wstrzymany, a uwolnione ramki przekazane do pozostałych procesów (zgodnie z zasadą proporcjonalności). Strategie wyboru procesu do wstrzymania mogą być różne: zatrzymanie procesu o najmniejszym *WSS* (ryzyko konieczności wstrzymania wielu procesów), zatrzymanie procesu o największym *WSS* (ryzyko zatrzymania dużego, intensywnie pracującego procesu), zatrzymanie procesu o najniższym priorytecie (konieczność określania priorytetów) czy zatrzymanie procesu o wymaganej liczbie ramek (wzrost złożoności algorytmu + wprowadzenie pewnej losowości w wyborze procesu).

Równie istotne jest określenie momentu wyznaczania *WSS*. Obliczanie tej wartości przy każdym odwołaniu powodowałoby znaczące obciążenie systemu, więc byłoby nieefektywne (a wielu przypadkach zbędne). Rozsądnym wydaje się przyjęcie pewnej wartości *c* takiej, że $c < \Delta t$. Sugeruję poeksperymentować z konkretnymi wartościami zaczynając od $c = ½\ \Delta t$.