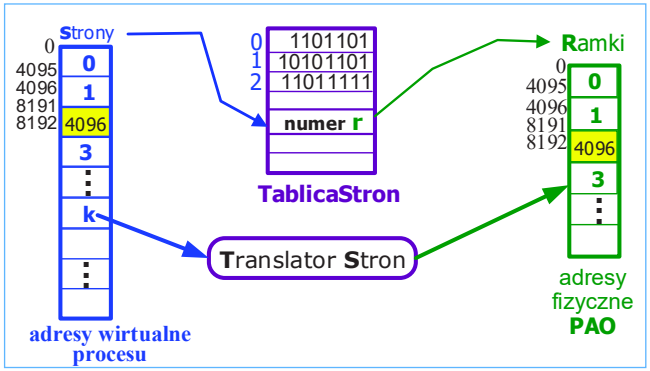
9. PAMIĘĆ WIRTUALNA

Pamięć Wirtualna umożliwia wykonanie procesów, **nie**mieszczących się w całości PAO.

Tworzy abstrakcyjną pamięć główną w postaci dużej, jednolitej Tablicy Stron.



◗ **Cały proces nie musi być trzymany w PAO** w tym samym czasie.

- fragmenty kodu obsługujące błędy, które rzadko występują,
- tablicom często rezerwuje się więcej pamięci, niż potrzeba,
- rzadko wykorzystuje się wszystkie możliwości programu.

◗ **Korzyści trzymania** w **PAO fragmentu procesu** (kilku stron):

- **brak** ograniczeń związanych z wielkością **pamięci fizycznej**
  użytkownik pisze programy w wirtualnej przestrzeni adresowej;
- **mniejszy** obszar **pamięci fizycznej** może proces zajmować,
  można lokować więcej procesów w PAO;
- **maleje** liczba operacji We/Wy koniecznych do ***wymiany procesów*** w PAO.

Pamięć wirtualna można implementować w formie:

- **stronicowania** na **żądanie** *(demand paging)*,
- segmentacji na żądanie *(demand segmentation)*,
- stronicowanego segmentowania.

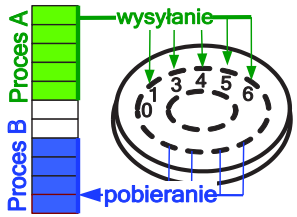
9.1. Stronicowanie na żądanie

Stronicowanie na żądanie zostało po raz pierwszy użyte w systemie Atlas, w komputerze MUSE w Manchester University około 1960 r.

➔ Proces stanowi Ciąg Stron, a nie wielką i ciągłą przestrzeń adresową ➔

Procedura leniwej wymiany:
nie dokonuje się wymiany strony w pamięci, jeśli nie jest to konieczne.
Procedura wymiany może operować całymi procesami.

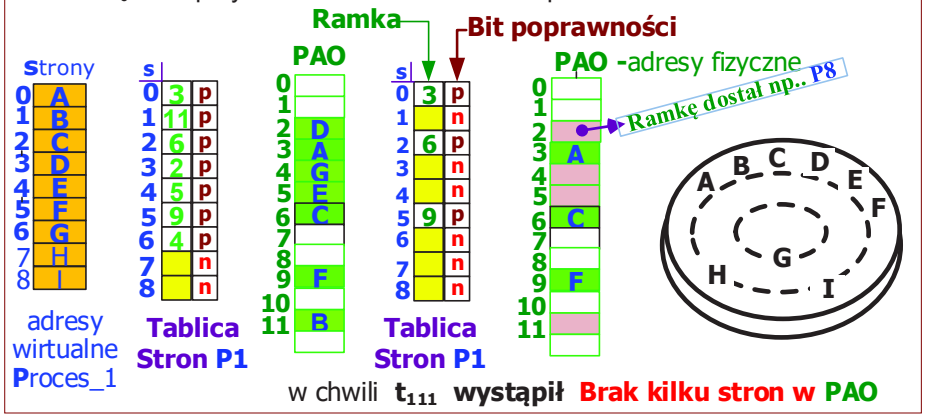
Procedura stronicująca *(pager)*: zajmuje się stronami procesu;
zgaduje, które strony będą w użyciu przed ponownym wysłaniem procesu na dysk.



☛ Nie wymienia całego procesu, gdyż do **PAO sprowadza** tylko **niezbędne strony**.

➔ **Niezbędne są środki** do **odróżnienia stron będących** w **PAO od stron na dysku**.
Można zastosować **bit poprawności**.

W chwili t_0 SO przydzielił 9-cio stronicowemu procesowi **7 ramek**



Wartości **bitu poprawności**:

poprawna -dozwolone odwołanie do danej strony, która znajduje się PAO.

niepoprawna może oznaczać jedną z dwóch możliwości, że strona jest:
-**niedozwolona** (nie należy do logicznej przestrzeni adresowej procesu);
-**dozwolona**, lecz aktualnie znajduje się na **dysku**.

Strona "***nie załadowana***" może być w TablicyStron oznaczona jako:
-***niepoprawna***,
-zawierać **adres dyskowy** strony.

Strona oznaczona jako niepoprawna nie wywołuje skutków, jeśli proces nie odwoła się do niej.

Błąd **"brak strony"** wywołany jest próbą dostępu do strony oznaczonej „***niepoprawna***".

Jeżeli sprzęt stronicujący - tłumacząc adres na podstawie **TablicyStron** - wykryje, że **bit poprawności** ma wartość „**nie**poprawne" to:
w tradycyjnym rozwiązaniu spowoduje ***awaryjne przejście*** do SO.

➔ Pułapka „**brak strony**" to wynik ***nieprzygotowania*** -przez SO- w PAO potrzebnej strony;
(minimalizacja operacji dyskowych, zużycia pamięci),

nie musi oznaczać błąd adresowania-powstałego wskutek użycia niedozwolonego adresu pamięci (np. niewłaściwy wskaźnik do tablicy).

Jeżeli problem ten wystąpi, powinien zostać skorygowany.

Schemat obsługi błędu "brak strony" *(page fault)*:

1. Sprawdzić w wewnętrznej tablicy (w PCB), czy odwołanie do PAO było dozwolone.
2. Jeżeli odwołanie było **niedozwolone** to proces zostaje zakończony.
3. Jeśli było dozwolone, lecz nie było strony w **PAO**, to sprowadzić tę stronę.
4. Wznowić wykonanie procesu.

> Po usunięciu "braku strony" można wznowić proces w tym samym miejscu i stanie -jeżeli strona jest już w PAO- dzięki przechowaniu stanu przerwanego procesu.

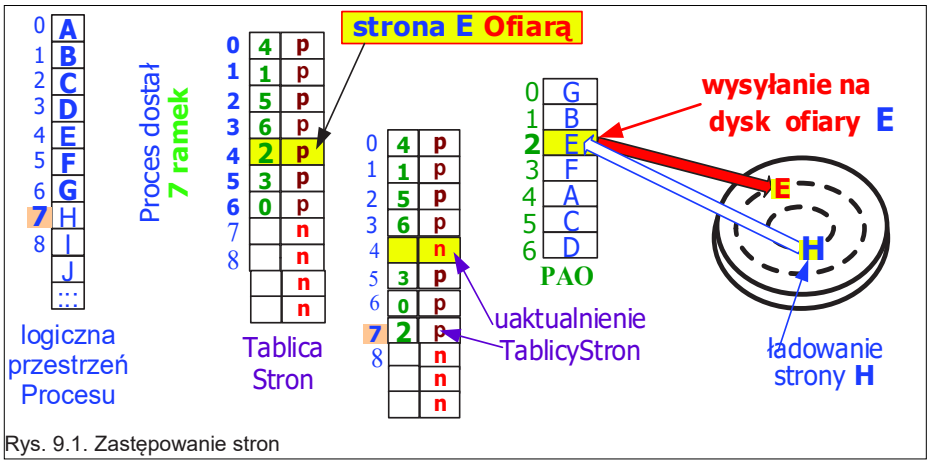
Procedura obsługi braków stron:

1. Zlokalizować potrzebną **stronę** na dysku.
2. Odnaleźć wolną **ramkę**:
   -jeśli istnieje wolna ramka, to wykorzystać ją;
   -jeśli nie, to wykonać **algorytm zastępowania stron** aby wytypować ***ramkę-ofiarę*** *(victim frame);*
   -**ramkę**-ofiarę zapisać na dysku; towarzyszy temu stosowna zmiana w **T**ablicy**S**tron.
3. Wczytać potrzebną **stronę** do (świeżo) zwolnionej **ramki**
4. Zmodyfikować struktury danych procesu i TablicęStron (odnotowując, że strona jest już w PAO).
5. Wznowić działania procesu (przerwanego rozkazu - odwołał się do niedozwolonego adresu).

Koncepcja zastępowania stron (wszystkie ramki są zajęte → Proces dostał **7 ramek**)

- znaleźć **stronę**, która **nie jest** ***właśnie*** używana i zwolnić **ramkę**, która ją odwzorowuje.
- **ramkę** zwolnić, zapisując jej zawartości na dysk i dokonać zmian w **T**ablicy**S**tron wskazując, że **strona** (odwzorowana na ramkę) nie jest już w PAO.
- do zwolnionej **ramki** wprowadzić **stronę**, której brak przerwał wykonywanie procesu.

W chwili $\mathbf{T_i}$ proces odwołał się do adresu na stronie **7** (**H**), której nie ma w PAO (rys. 9.1).
Co zrobić gdyż brakuje ramek?



Rys. 9.1. Zastępowanie stron

Gdy nie ma wolnych **ramek**, następuje **dwukrotne** przesyłanie stron
(jedno na dysk i jedno z dysku)

Można zastosować **bit modyfikacji** zwany **bitem zabrudzenia** *(modify, dirty bit), co* zmniejszy liczbę kosztownych operacji dyskowych.

➔ Każdą stronę lub ramkę można wyposażyć w sprzętowy **bit modyfikacji**.

Bit ten ustawiany jest sprzętowo na **1** (przy zapisie dowolnego bajta lub słowa na stronie), i wskazuje, że dana **strona** ***uległa zmianie***.

➔Przy wyborze strony do zastąpienia **sprawdza** się **jej bit modyfikacji**.

Ustawiony **bit modyfikacji:** stan strony zmienił się od czasu jej przeczytania z dysku.
W tym wypadku stronę należy zapisać na dysku.

0	4	p	1
1	1	p	1
2	5	p	0
3	6	p	1
4	2	p	0
5	3	p	1
6	0	p	1

Nie ***ustawiony*** (zero): zawartości strony **nie** zmieniono od czasu przeczytania jej do PAO.

Jeśli kopia strony na dysku nie została zniszczona (np. przez inną stronę), to nie trzeba przesyłać takiej strony na dysk, ponieważ ona już tam jest.

Proces można wykonywać, gdy niektórych jego części nie ma w PAO.

Proces próbuje sięgnąć do komórki, której nie ma w pamięci, wówczas:
-sprzęt spowoduje przejście do SO (brak strony), który wczyta potrzebną stronę do PAO,
-SO wznowi proces tak, jakby ta strona była stale w pamięci.

➔ Proces można rozpocząć wykonywać bez żadnej strony w PAO.

SO ustawi **licznik rozkazów** na pierwszy rozkaz procesu, znajdujący się na stronie nieobecnej w pamięci, wtedy w procesie wystąpi natychmiast brak strony.

Po sprowadzeniu tej strony do pamięci proces będzie wykonywany, z ewentualnymi przerwami na uzupełnianie dalszych brakujących strony w PAO.

> Teoretycznie proces może potrzebować **kilku stron** PAO na wykonanie **każdego rozkazu** (jednej z powodu rozkazu i wielu z powodu danych), co grozi wystąpieniem wielu braków stron przypadających na jeden rozkaz.
> Prawdopodobieństwo takiego zdarzenia jest małe.

Sprzęt musi gwarantować wznowienie wykonania rozkazu po wystąpieniu braku strony.

Brak strony może wystąpić w dowolnym odwołaniu do pamięci.

Brak strony wystąpił przy:
-*pobraniu* ***rozkazu*** do wykonania: wznowienie wymaga ponownego pobrania tego rozkazu.
-*pobieraniu* ***argumentu*** rozkazu: -należy pobrać ten rozkaz ponownie,
-powtórnie go zdekodować,
-znowu pobrać argument.

Rozkaz trójadresowy typu **C** = DODAJ(A, B).

Jeśli niepowodzenie zdarzy się przy podstawianiu do zmiennej **C** (C jest na stronie, której nie ma w PAO), to trzeba: -sprowadzić wymaganą stronę,
-uaktualnić TablicęStron,
-***wykonać ponownie*** rozkaz (pobrać, zdekodować, itd.).

◗ Rozkaz zmienia kilka różnych komórek.

Rozkaz przesyła z jednego miejsca na drugie do 256 B (mogą zachodzić na siebie).

Jeżeli blok bajtów (źródłowy lub docelowy) przekracza granicę strony, to brak strony może powstać po częściowym wykonaniu przesyłania.

Jeżeli blok docelowy zachodzi na blok źródłowy, to dane w bloku źródłowym mogą ulec zniekształceniu, wykluczając proste wznowienie tego rozkazu.

Pierwsze rozwiązanie: **mikroprogram oblicza położenie krańców obu bloków** `i usiłuje do nich dotrzeć.`

Jeśli miałby się pojawić brak strony, to wystąpi już w tym kroku, zanim cokolwiek zostanie zmienione.

☛ Przesyłanie wykona się wtedy gdy **niezbędne strony** znajdują się w PAO.

Drugie rozwiązanie: **rejestry przetrzymują wartości przesyłanych pól**.

Jeśli wystąpi brak strony, to wszystkie poprzednie wartości będą z powrotem przepisane do PAO, zanim wystąpi pułapka.

Nastąpi odtworzenie stanu pamięci sprzed wykonania rozkazu, wobec czego można go powtórzyć.

❒ Sprawność stronicowania na żądanie

Efektywny czas dostępu *(effective access time)* do pamięci stronicowanej na żądanie określa:

(**1 - p**) • **cd** + **p** • **C**zas**O**bsługi**B**raku**S**trony

gdzie **p** -prawdopodobieństwo braku strony,
cd - czas dostępu do PAO [ns].

Czynniki wpływające na **C**zas **O**bsługi **B**raku **S**trony:

1. obsługa przerwania wywołanego brakiem strony,
2. czytanie strony,
3. wznowieniem procesu.

Przyjmując średni czas obsługi braku strony **25** ms, czas dostępu do pamięci **100 ns**, mamy:

efektywny czas dostępu = (1 - p) • 100 +p • 25 [ms] = 100 + 24999900 • **p** [nS]

Jeśli jeden dostęp na **1000** powoduje brak strony, to efektywny czas dostępu = **25 µS**.

Stronicowanie na żądanie spowolniło komputer **250**-krotnie!

Kiedy pogorszenie będzie **mniejsze** niż **10%** ?

110 > 100 + 25 000 000 • p ∧ **p < 0.0000004**

Utrzymanie spowolnienia (powodowanego stronicowaniem) na poziomie **10%** wymaga mniej niż **1 brak strony** na **2.500.000** odniesień do PAO.

Brak strony powoduje ciąg zdarzeń:

1. Przejście do systemu operacyjnego.
2. Przechowanie rejestrów użytkownika i stanu procesu.
3. Określenie przyczyny przerwania (brak strony).
4. Sprawdzenie poprawności odniesienia do strony i poszukiwanie strony na dysku.
5. Wydanie polecenia czytania z dysku do wolnej ramki:
   -czekanie w kolejce do tego urządzenia, aż będzie obsłużone zamówienie na czytanie;
   -czekanie przez czas szukania informacji na urządzeniu i (lub) jej nadchodzenia;
   -rozpoczęcie przesyłania strony do wolnej ramki.
6. Przydzielenie CPU innemu użytkownikowi na czas oczekiwania bieżącego użytkownika.
7. Otrzymanie przerwania z dysku (zakończona operacja We/Wy).
8. Przechowanie rejestrów i stanu procesu innego użytkownika.
9. Określenie, czy przerwanie pochodziło od dysku.
10. Skorygowanie zawartości Tablicy Stron -odnotowanie, że strona jest obecnie w pamięci.
11. Czekanie na powtórne przydzielenie CPU danemu procesowi.
12. Odtworzenie stanu rejestrów użytkownika, stanu procesu i nowej Tablicy Stron, po czym wznowienie przerwanego rozkazu.

❒ Nadprzydział pamięci

Wzrost stopnia wieloprogramowości prowadzi do **nadprzydziału** *(over-allocating)* pamięci.

Wykonuje się **6** procesów, każdy ma rozmiar **10** stron, faktycznie używa **5** stron.
Fizycznie dostępnych jest **40** ramek.

Może się zdarzyć, że każdy z **6**-ciu procesów potrzebuje nagle wszystkich **10** stron (np.: szczególny zestaw danych), co spowoduje zapotrzebowanie na **60** ramek.

Prawdopodobnej takiego zdarzenia rośnie ze wzrostem stopnia wieloprogramowości, gdy średnie wykorzystanie pamięci zbliża się do ilości pamięci dostępnej fizycznie.

Skutki nadprzydziału:

- Podczas wykonywania procesu użytkownika wystąpi brak strony.
- Sprzęt zaalarmuje SO, który sprawdzi swoje wewnętrzne tablice, aby stwierdzić, że to jest brak strony, a nie próba niedozwolonego dostępu do pamięci.
- SO zlokalizuje na dysku potrzebną **stronę**, lecz okaże się, że brak jest wolnych **ramek**.

W tej sytuacji system operacyjny mógłby:

➢ zakończyć proces użytkownika.

Użytkownicy nie powinni być świadomi, że ich procesy są wykonywane w systemie stronicowanym.

Zakończenie procesu **nie** jest dobrym wyborem.

➢ wymienić proces, zwalniając jego ramki i zmniejszając stopień wieloprogramowości.

➢ zastosować technikę **zastępowania stron** *(page replacement).*

➔ **Mechanizm pamięci wirtualnej jest przezroczysty dla procesów użytkownika** ➔

➔ **Mechanizm stronicowania umieszczony jest między CPU a PAO** ➔

➔ **Zastępowanie stron jest podstawą stronicowania na żądanie** ➔

Należy udostępnić algorytmy: -**przydziału ramek** *(frame allocation algorithm)*
-**zastępowania stron** *(page-replacement algorithm).*

- Dla kilku procesów w **PAO** należy zdecydować, ile ramek przydzielić do każdego procesu.
- Gdy zajdzie konieczność zastąpienia **strony**, trzeba **umieć** wskazać **ramkę** do wymiany.

9.1.1. Algorytmy zastępowania stron

Algorytm zastępowania winien minimalizować **częstość braków stron** *(page fault rate).*

- Algorytm ocenia się wykonując go na pewnym ***ciągu odniesień*** do pamięci oraz sumując liczby braków stron.

Ciąg odniesień *(reference string)* to zapis ***adresu*** każdego odwołania do pamięci na podstawie śledzenia systemu.

Może być modelowany generatorem liczb losowych.

Liczba danych ulega redukcji gdyż:

1. dla znanego rozmiaru stron można brać pod uwagę tylko ***numer strony***, a nie cały adres.
2. kolejne odwołanie do strony **s**, następujące bezpośrednio po odwołaniu do strony **s**, **nie spowoduje braku strony**.

Strona **s** jest już w PAO po pierwszym odniesieniu.

Należy znać **liczbę** dostępnych **ramek**, aby określić liczbę brakujących stron dla ciągu odniesień

Adresy$_{16}$: 0100, 0423, 0103, 0621, 0111, 0115, 0134, 0634, 0145, 0147, 0152, 0643, 0112, 0115

Dla rozmiaru strony 256 B ≡ 100_{16} otrzymuje się **ciąg odniesień: 1, 4, 1, 6, 1, 6, 1, 6, 1**

Dane są **3 puste** ramki.

Dla ciągu wystąpią **3** braki **stron** (przy pierwszym odwołaniu do nowej strony).

1
4
6

Dla **2-ch** pustych ramek wystąpią **3 braki** stron z jednym **zastąpieniem**.

1	4

1	6

9.1.1.1. Algorytm FIFO

Algorytm FIFO kojarzy z każdą stroną **czas wprowadzenia** jej do PAO.

Do zastąpienia strony wybiera się **stronę najstarszą** (najdłużej istniejącą w PAO).
Czas wykorzystania strony jest nieistotny.

Kolejka FIFO może przechowywać wszystkie strony przebywające w PAO.

- Do zastąpienia pobierana jest strona z **czoła kolejki**.
- Stronę wprowadzoną do **PAO** lokuje się na **końcu** kolejki.

Dane są **3 puste ramki** r 12 r 34 r 76

s	r	
0	34	p
1	76	p
2		n
3		n
4		n
5		n
6		n
7	12	p

TS

s	r	
0	34	p
1	76	p
2	12	p
3		n
4		n
5		n
6		n
7	12	n

TS

i ciąg odwołań do **stron**:

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

← **STRONY**

15 braków stron

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

← **kolejka**

Trzy pierwsze odwołania (**7, 0, 1**) wymagają sprowadzenia **stron** do pustych **ramek**.

Odwołanie (**2**) wymaga zastąpienie ***strony* 7**, gdyż została ona wprowadzona jako pierwsza.

Kolejne odniesienie **(0)** nie spowoduje błędu gdyż strona **0** jest już w pamięci.

Pierwsze odwołanie do strony (**3**) spowoduje zastąpienie strony **0**, gdyż była ona pierwszą z trzech stron wprowadzonych do pamięci w ciągu (**0, 1, 2**).

➔ Następne odniesienie do strony **(0)** spowoduje błąd i strona **0** wejdzie na miejsce strony **1**.

Łączna liczba **braków** stron wynosi **15**.

Algorytmu FIFO nie zawsze jest **optymalnym rozwiązaniem**

- Strona zastępowana może zawierać ***zmienną*** będącą **ciągle** w użyciu.

Do czasu wybrania do zastąpienia strony, która jest właśnie używana, wszystko działa poprawnie.

Po wysłaniu z pamięci aktywnej **strony** w celu sprowadzenia nowej prawie natychmiast z powodu jej braku zostanie ona zamówiona ponownie.

☛ Aby **sprowadzić** z powrotem wcześniej aktywną stronę do PAO, trzeba **usunąć** inną.

Zły wybór przy zastępowaniu zwiększa liczbę braków stron i spowalnia wykonanie procesu, choć ***nie powoduje*** niepoprawnego działania.

❒ **Anomalia Belady'ego** *(Belady's anomaly)*:
w niektórych algorytmach zastępowania stron, współczynnik braków stron może **wzrastać** ze wzrostem wolnych ramek.

Anomalie Belady'ego ilustruje ciąg odniesień: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
W algorytmie FIFO liczba braków stron dla **4-ch** ramek wynosi **10**,
zaś dla **3-ch** ramek wynosi **9**.

Zadanie. Wykazać, że dla w/wym. ciągu odniesień zachodzi anomalia Belady'ego

9.1.1.2. Algorytm optymalny OPT lub MIN

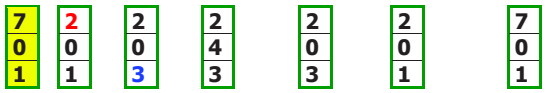
Zastąp tę **stronę,** która **najdłużej** ***nie będzie*** *(przyszłość)* **używana.**

Algorytm optymalny jest wolny od anomalii Belady'ego.

Algorytm gwarantuje najmniejszą liczbę braków stron (dla danej liczby ramek).

Indeks: → 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Ciąg odwołań: **7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1**



9 braków stron.

Trzy pierwsze odwołania (**7, 0, 1**) wymagają sprowadzenia stron do pustych ramek.

Odwołanie do strony (**2**) zastąpi stronę **7**, ponieważ ***nie zostanie użyta*** aż do **18-go** odwołania.

Odwołanie do strony (**3**) zastąpi stronę **1**, ponieważ ***nie zostanie użyta*** aż do **14-go** odwołania.

Algorytm zastępowania wymaga wiedzy o przyszłej postaci ciągu odniesień.

Przydatny w studiach porównawczych.

Wiedza, że jakiś algorytm różni się od optymalnego tylko o kilka % jest przydatną informacją.

9.1.1.3. Algorytm LRU *(Least Recently Used)* - zastępuje najdawniej **używane** strony

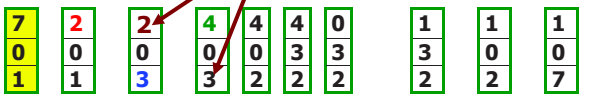
Zastąpiona zostaje **strona,** która **nie była** **używana** od ***najdłuższego*** czasu

Algorytm LRU kojarzy z każdą stroną czas jej ostatniego użycia.

Algorytm LRU jest wolny od anomalii Belady'ego.

Indeks: ← 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Ciąg odwołań: **7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1**



12 braków stron.

Odwołanie do strony **4** zastąpi stronę **2**, ponieważ z trzech stron (2, 0, 3), pozostających w pamięci strona **2** była używana najdawniej (4-te odwołanie).

Algorytm LRU ***zastąpi*** **stronę 2**, nie mając informacji,
że będzie ona za chwilę potrzebna.

Gdy chwilę później wystąpi brak **strony 2**, algorytm LRU zastąpi **stronę 3**, gdyż z trzech stron w pamięci (4, 0, 3), ona była używana najdawniej.

Problemem implementacji: jak ustalić czas ostatniego użycia ramek, aby zachować właściwy porządku ich wymiany.

❒ Dwie metody implementacji algorytmu: Licznik i Stos.

◗ **Licznik:** do każdej pozycji w **TablicyStron** dołączamy ***rejestr czasu użycia***.

Wykorzystuje się zegar logiczny lub licznik.

☛ Wskazania zegara zwiększane są z każdym odniesieniem do PAO.

Gdy występuje odniesienie do PAO, kopiowana jest zawartość **rejestru zegara** do ***rejestru czasu użycia*** należącego do danej strony w **TablicyStron**.

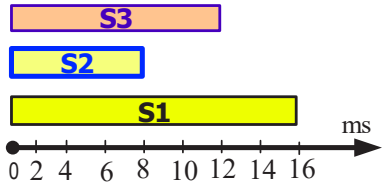
Dla każdej strony znany jest **czas** ostatniego do niej odniesienia.

☛ **Zastępujemy** stronę z ***najmniejszą*** wartością **czasu**.

Schemat wymaga: -***przeglądania*** **TablicyStron** w celu znalezienia strony najdawniej,
-**zapisywania** w **TablicyStron** stosownej informacji o czasach użycia.

s	r		Rejestr
0		n	
1	76	p	16000
2	34	p	8000
3	12	p	12000
4		n	
5		n	
6		n	

TablicaStron



Rejestry czasu użycia wymagają uaktualnień również przy ***wymianach*** TablicStron (rotacja przydziału CPU).

◗ **Stos:** budowany jest stos numerów **stron**

Każde **odwołanie** do strony powoduje ***wyjęcie*** numeru strony ze stosu i ***umieszczenie*** jej na szczycie.

Na **szczycie** stosu jest strona **ostatnio** użyta;
na **spodzie** jest strona ***najdawniej*** użyta.

ciąg odniesień

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

Trzeba wydobyć pozycje z wnętrza stosu.

Stosuje się listę dwukierunkową ze wskaźnikami do czoła i końca listy.

☛ **Do zastąpienia** strony **nie** trzeba przeszukiwać listy, ponieważ wskaźnik wskazujący na **dno** stosu, identyfikuje **najdawniej** używaną stronę.

Problemy: **Każde uaktualnienie listy jest czasochłonne.**

Uaktualnienie pól zegara lub stosu następuje przy ***każdym*** odniesieniu do pamięci.
Każde odwołanie do pamięci generuje przerwanie.
Uaktualnianie struktur danych na drodze programowej spowalnia kontakty z PAO.

Dodatkowe koszty zarządzania pamięcią można zredukować tylko dodatkowym sprzętem.

Klasa algorytmów zastępowania stron, zwana **algorytmami stosowymi** *(stack algorithms)*, nie wykazuje anomalii Belady'ego.

9.1.1.4. Algorytmy przybliżające metodę LRU zastąp najdawniej używaną stronę

Niewiele systemów posiada sprzęt do realizacji pełnego algorytmu LRU.

❒ **Bit odniesienia** *(reference bit)*, związany jest ze wszystkimi pozycjami **TablicyStron**, może wspomagać realizacje LRU (na początku SO zeruje wszystkie bity).

W trakcie realizacji procesu każdy **bit** związany ze stroną, do której następuje **odwołanie** (czytanie/pisanie dowolnego bajta na stronie), **jest ustawiany** (wartość 1) przez sprzęt.

Otrzymujemy informację, które strony były używane do określonej chwili.

Nie jest znany **porządek** użycia stron.

☛ Wiadomo tylko, które strony **nie były** u użyciu.

s	r		Bit odn.
0		n	
1	76	p	1
2	34	p	0
3	12	p	1
4		n	
5		n	
6		n	

TablicaStron

❒ **Algorytm dodatkowych bitów odwołań**

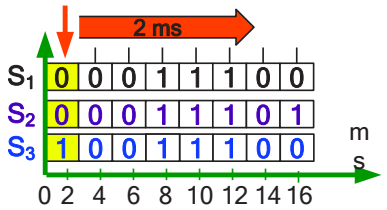
Dla **każdej** strony przeznacza się dodatkowo **bajt odwołań** (8-bitów).

W ustalonych okresach (np. **2ms**) przerwanie zegarowe przekazuje sterowania do SO.

➔ Wówczas dla każdej strony SO wprowadza **bit odniesienia** na najbardziej znaczącą pozycję **bajta odwołań**, przesuwając pozostałe bity o jedną pozycję w **prawo** (tracony jest bit na najmniej znaczącej pozycji).

Ten ***8-bitowy rejestr*** przesuwający zawiera historię użycia strony w ostatnich **8-miu** okresach.

Wartość 00000000 ⇒ odpowiadająca mu strona **nie** była używana przez ***osiem*** okresów.
Wartość 11111111 ⇒ stronę **użyto** w każdym z tych okresów przynajmniej raz.



Stronę skojarzoną z S_1 użyto w 8-mej, 10-tej i 12-tej ms.

Stronę skojarzoną z S_2 użyto w 8-mej, 10-tej, 12-tej i 16–tej ms.

Strona S_2 była częściej używana niż strona S_1.

Za stronę **najdawniej** używaną (typowaną do wymiany) przyjmuje się tą, której odpowiada ***najmniejsza*** **liczba** utworzona z bajta odwołań.

❒ **Algorytm drugiej szansy**

- wykorzystuje algorytm **FIFO** (strona najdłużej będąca w PAO).

Po wybraniu strony do wymiany sprawdza się ***dodatkowo*** jej **bit odniesienia**.

☛ Jeśli **bitu odniesienia ≡ 0**, to strona zostaje zastąpiona.

Gdy **bit odniesienia ≡ 1**, strona dostaje ***drugą szansę*** i jej **bit odniesienia** jest **zerowany**, ZAŚ

pod uwagę bierze się ***następną*** stronę wynikającą z porządku FIFO.

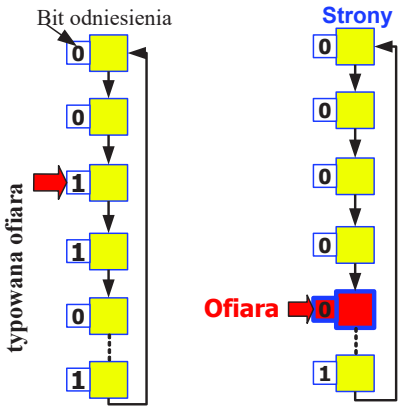
Strona, której dano drugą szansę, **nie będzie** uwzględniona przy zastępowaniu, dopóki nie zostaną zastąpione inne strony (lub otrzymają drugą szansę).

Jeśli strona eksploatowana jest często i utrzymuje swój bit odniesienia ustawiony, to nigdy nie zostanie zastąpiona.

◗ **Implementacja algorytmu drugiej szansy** jako **kolejki cyklicznej**.

Wskaźnik ➧ pokazuje stronę ***typowaną*** do zastąpienia.

Przy zapotrzebowaniu na ramkę wskaźnik przemieszcza się naprzód, aż zostanie znaleziona strona z ***wyzerowanym bitem*** odniesienia.



☛ Podczas przemieszczania wskaźnika bity odniesienia są zerowane.

Jeśli wszystkie bity odniesienia były **ustawione**, wskaźnik obiegnie całą kolejkę, dając każdej stronie drugą szansę, zerując jednocześnie wszystkie bity odniesienia.

Gdy wszystkie bity odniesienia są **ustawione**, to metoda drugiej szansy jest realizacją algorytmu FIFO.

9.1.1.5. Algorytm buforowania stron

❒ **SO przechowuje pulę wolnych ramek** (jeżeli go na to stać)

Po wystąpieniu błędu strony wybiera się ***ramkę_ofiarę***.

Potrzebną stronę **wprowadza się** do wolnej ramki **z puli**, zanim usunie się **stronę_ofiarę** z PAO.

Pozwala to szybko wznowić proces, bez oczekiwania na zapisanie **strony_ofiary**.

Później **stronę_ofiarę** przepisuje się na dysk,
➔ zaś zajmowaną przez nią ramkę **dołącza** do puli wolnych ramek.

❒ **SO utrzymuje listy modyfikowanych stron** zapis na dysk z wyprzedzeniem

Gdy urządzenie stronicujące jest bezczynne, wybiera się ***zmienioną*** stronę i zapisuje na dysk.
☛ Jej **bit modyfikacji** zostaje **zerowany**.

Zwiększa się prawdopodobieństwo, że **stronę wybraną** do **zastąpienia** ***nie trzeba*** będzie **zapisywać** na dysku.

❒ **SO utrzymuje pulę wolnych ramek** ale **pamięta, które strony gdzie rezydowały**.

Zawartość ramki nie ulega zmianie wskutek zapisania jej na dysku.

Póki dana **ramka** nie zostanie użyta na nowo, to pozostająca w niej strona może być użyta ponownie z puli wolnych ramek.

☛ Nie jest potrzebna dyskowa operacja We/Wy.

Gdy wystąpi brak **strony** sprawdza się, czy potrzebna strona jest w puli **wolnych ramek**.
Jeśli jej tam nie ma, to należy wybrać wolną ramkę i przeczytać do niej brakującą stronę.

Technikę zastosowano w systemie VAX/VMS wraz z algorytmem zastępowania FIFO.

Gdy w wyniku zastępowania według algorytmu FIFO zdarzy się usunięcie strony, która jest właśnie używana, wówczas stronę tę szybko odzyskuje się z bufora **wolnych ramek** bez potrzeby wykonywania operacji We/Wy.

9.2. Przydział ramek

Jak rozdzielać 123 wolne ramki na 3 procesy?

Istnieje minimalna liczba ramek, które muszą być przydzielone, wynika to ze zbiorem rozkazów w architekturze komputera.

Jeśli brak strony wystąpi przed zakończeniem wykonania rozkazu, to rozkaz musi być powtórzony.

☛ Należy mieć tyle ramek, ile potrzeba do wykonania pojedynczego ***rozkazu***.

Rozkaz przesyłania w niektórych trybach adresowania składa się z więcej niż jednego słowa, rozkaz ten może znaleźć się na **dwu** sąsiednich stronach.

Ponadto każdy z dwu argumentów może być adresowany pośrednio, co stawia w pogotowiu łącznie sześć stron.

❒ **Schemat przydziału równego** *(equal allocation)*.

Mając **m** ramek i **n** procesów przyznaje się każdemu jednakową porcją ***min*** ramek.

Pozostałą nadwyżkę ramek można przeznaczyć na **bufor** wolnych ramek.

❒ **Schemat przydziału proporcjonalnego** *(proportional allocation).*

Każdemu procesowi przydziela się dostępne **ramki** odpowiednio do jego **rozmiaru**.

Niech s_i oznacza wielkość pamięci wirtualnej procesu P_i, zaś **m** ogólną liczbę ramek.

Procesowi P_i przydziela się a_i ramek: $a_i = s_i / \sum s_i \cdot m$

➡ Wielkość przydziału dla każdego procesu zależy od stopnia **wieloprogramowości**.

Wzrost stopnia wieloprogramowości powoduje utratę przez procesy nieco **ramek**, aby zapewnić pamięć potrzebną dla nowych procesów.

Obniżenie stopnia wieloprogramowości sprawia, że **ramki** należące do zakończonych procesów, można rozdzielić innym procesom.

W innej wersji przydziału proporcjonalnego liczba **ramek** zależy od **priorytetów** procesów albo od kombinacji rozmiaru i priorytetu.

❒ **Dwie modele zastępowania stron**

1. globalne zastępowanie *(global replacement)* umożliwia procesom wybór **ramki** ze zbioru **wszystkich** **ramek**, nawet, gdy **ramka** jest przydzielona do innego procesu.

☛ Jeden proces może zabrać **ramkę** drugiemu procesowi.

Proces o **wyższym priorytecie** może zwiększać liczbę swoich **ramek** kosztem procesu o niższym priorytecie.

2. lokalne zastępowanie *(local replacement)* umożliwia wybór **ramki tylko** ze zbioru **ramek** przydzielonych do danego procesu.

W algorytmie zastępowania **globalnego** proces:

- może wybierać **ramki** przydzielone ***innym*** procesom, zwiększając liczbę przydzielonych mu ramek

  (przy założeniu, że inne procesy nie będą wybierały jego **ramek** do zastępowania),
- **nie** może kontrolować **własnej częstości** występowania braków stron.

Zbiór stron procesu w pamięci zależy nie tylko od zachowania się danego procesu przy stronicowaniu, ale również od sposobu stronicowania innych procesów.

Ten sam proces w jednych warunkach realizacji wykonuje się 0.5 s, a przy innych **15.5** s.

☛ Nie dzieje się tak w przypadku algorytmu zastępowania lokalnego.

W zastępowaniu **lokalnym** strony mniej **używane** mogą być **niedostępne**.

Zastępowanie globalne daje lepszą przepustowość systemu i jest częściej stosowane.

9.3. Szamotanie

Jeśli proces nie dostanie nominalnej liczby ramek, to szybko wystąpi brak strony.

Wtedy któraś ze **stron** zostanie zastąpiona.

☛ Strony są **aktywnie używane**, i trzeba zastąpić stronę, która za chwilę będzie potrzebna.

☛ W procesie występują kolejno po sobie braki stron.

P	1	0	7	2	3	5	4	2
1	11							
0	72							
7	34							

Proces wymienia **stronę**, po czym
- z powodu jej braku - sprowadza **ją** z powrotem.

Tak zwiększona aktywność to **szamotanie** *(thrashing).*

Proces szamoce się, jeśli spędza więcej czasu na stronicowaniu niż na wykonaniu.

9.3.1. Przyczyna szamotania

Zastosowano **globalny** algorytm zastępowania stron (nie uwzględnia do jakich procesów należą).

☛ **Proces** wchodzi w nową fazę działania i potrzebuje więcej **ramek**.

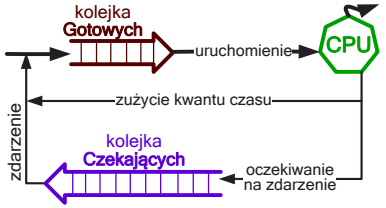
Zaczyna wykazywać braki stron i ***zabiera*** strony innym procesom (które potrzebują tych stron), więc wykazują ich braki i przyczyniają się do ***odbierania*** stron ***kolejnym*** procesom.

Rozważmy 4-ry procesy, którym przydzielono **3 ramki** i odpowiadające im ciągi odwołań do **stron**:

P1	1	0	7	2	3	1	0	2	P2	1	0	7	2	3	1	0	2	P3	1	0	7	2	3	1	0	2	P4	1	0	7	2	3	1	0	2
1																																				
0																																			
7																																			

W jakiejś chwili czasowej procesy **P2**, **P3** i **P4** mogą wykazywać braki stron.

Procesy ustawiają się w kolejce do urządzenia stronicującego aby wymienić strony, a jednocześnie **opróżnia** się kolejka procesów **Gotowych** do wykonywania.



☛ **Oczekiwanie** procesów na urządzenie stronicujące **zmniejsza** wykorzystanie CPU.

Planista przydziału **CPU** widzi spadek jego wykorzystania, więc **zwiększa** stopień wieloprogramowości.

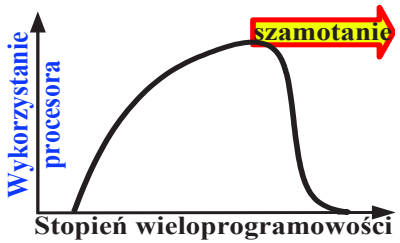
Nowy proces zabiera **ramki** innym procesom, co zwiększa liczbę braków **stron** i wydłuża kolejkę do urządzenia stronicującego.

Wykorzystanie CPU spada jeszcze bardziej i planista przydziału procesora **znowu** zwiększa stopień wieloprogramowości.

Powstaje szamotanina, przepustowość systemu gwałtownie maleje i wzrasta częstość występowania braków stron.

Na rysunku widać, że w miarę wzrostu stopnia wieloprogramowości wykorzystanie CPU rośnie, choć coraz wolniej, aż osiąga maksimum.

Dalsze zwiększanie wieloprogramowości prowadzi do szamotania i wykorzystanie CPU ostro maleje.



Aby zwiększyć wykorzystanie CPU i powstrzymać szamotanie, należy ***zmniejszyć*** stopień wieloprogramowości.

Efekt szamotania można ograniczyć za pomocą ***lokalnego*** lub ***priorytetowego*** algorytmu zastępowania.

Przy **zastępowaniu lokalnym**, gdy jakiś proces zaczyna się szamotać, wówczas ***nie wolno*** mu kraść ramek innego procesu i doprowadzać inny proces do szamotania.

☛ Rośnie średni czas obsługi braku strony gdyż wydłuża się kolejka do urządzenia stronicującego.

Czas efektywnego dostępu wzrasta nawet dla procesów, które się nie szamocą.

Zapobiec szamotaniu można dostarczając procesowi tyle ramek, ile potrzebuje.

Skąd wiadomo, ile ramek proces będzie potrzebować ?

9.3.2. Model strefowy *(locality model)*

Zakłada, że w trakcie wykonania proces przechodzi z jednej strefy programu do innej.

Strefa programu: zbiór stron pozostający we wspólnym użyciu.

Wywoływany podprogram (funkcja) definiuje nową strefę.
W strefie: odniesienia do pamięci dotyczą rozkazów danego podprogramu, jego zmiennych lokalnych i podzbioru zmiennych globalnych.
Wychodząc z podprogramu, proces opuszcza daną strefę, gdyż zmienne lokalne i rozkazy podprogramu przestają być dalej aktywnie używane.

Zazwyczaj program składa się z wielu stref o różnych wielkościach.

Strefy programu określone są przez jego strukturę i struktury danych.

☛ **Procesowi przydziela** się **ramki niezbędne** dla **bieżącej strefy**.

Proces wykaże braki stron zanim wszystkie strony danej strefy nie znajdą się w pamięci, następnie braki zanikną do czasu zmiany strefy.

Przydzielenie **mniej** ramek, niż rozmiar bieżącej strefy, wywoła **szamotanie** procesu.

9.3.3. Model zbioru roboczego *(working set model)*

Opiera się na założeniu, że program ma charakterystykę strefową.

Parametr **N** definiuje **OKNO zbioru roboczego** *(working-set window)*.

Model sprawdza **N** ostatnich ***odniesień*** do **stron** zawartych w **OKNIE**.

Zbiór roboczy *(working set)*: zbiór **stron**, do których nastąpiło **ostatnich** **N** odniesień.

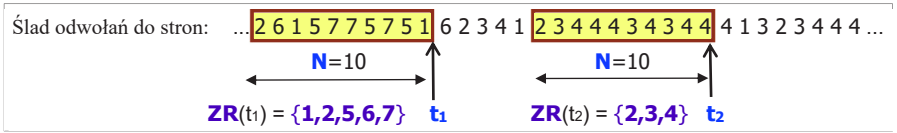
☛ Jeśli strona jest aktywnie używana to znajduje się w ***zbiorze roboczym***.

Gdy strona przestanie być używana, to wypada ze zbioru roboczego po **N** jed. czasu, odliczonych od ostatniego do niej odwołania.

Zbiór roboczy przybliża strefę programu.

Dany jest ciągu odniesień do pamięci.

Przyjmujemy **N** = **10** odniesień do pamięci.



☛ Dokładność zbioru roboczego zależy od wartości parametru **N**.

Jeśli parametr **N** będzie za mały, to nie obejmie całego zbioru roboczego.

Jeśli parametr **N** będzie za duży, to może zachodzić na kilka stref programu.

Gdy parametr **N** jest nieskończony, zbiorem roboczym jest zbiór stron, do których odwoływał się procesu podczas swojego działania.

☛ Najważniejszą cechą zbioru roboczego jest jego rozmiar.

➔ Dla znanego rozmiaru zbioru roboczego $\mathbf{R\_ZR_i}$ każdego procesu w systemie, globalne zapotrzebowanie na ramki określa wzór: $\mathbf{Z} = \sum \mathbf{R\_ZR_i}$

☛ Każdy proces używa strony ze swojego zbioru roboczego;

I **-ty** proces potrzebuje $\mathbf{R\_ZR_i}$ ramek.

Jeśli łączne **z**apotrzebowanie jest większe niż liczba dostępnych ramek (***z*** > *m*), to wystąpi szamotanie, gdyż niektóre procesy nie otrzymają wystarczającej liczby ramek.

❒ **Zastosowanie modelu zbioru roboczego**

SO analizuje **zbiór roboczy** każdego procesu i przydziela każdemu procesowi tyle ramek, ile wynika z **rozmiaru *jego* zbioru roboczego**.

Pozostałe dodatkowe ramki umożliwiają rozpoczęcie nowego procesu.

Gdy rozmiary zbiorów roboczych procesów **wzrastają** i zaczynają przekraczać liczbę dostępnych ramek w SO, wówczas ***wstrzymany*** zostaje jakiś Proces.

☛ **Usuwane** są strony procesu z PAO, a jego ramki przydzielane innym procesom.

❒ **Trudności** w **utrzymaniu śladu zbioru roboczego**

Zawartość **OKNA zbioru roboczego** zmienia się w czasie.

Każde odwołanie do PAO wpisuje na **początku** okna **nowe** odniesienie, a z jego końca **usuwa** najstarsze odniesienie.

Strona należy do zbioru roboczego, jeśli występuje w oknie roboczym.

Można modelować **Zbiór roboczy** za pomocą zegara generującego przerwania w stałych odstępach czasu i **bitu odniesienia**.

9.3.1. Częstość braków stron

Model ***zbioru roboczego*** **nie jest** wygodną metodą nadzorowania szamotania.

Szamotanie charakteryzuje duża częstość występowania braków stron.

Można nadzorować częstości braków stron.

Pomiar **Częstości Braków Stron** (*Page-Fault Freąuency* - **PFF**) jest rozwiązaniem prostszym.

Duża wartość **PFF** sygnalizuje, że proces potrzebuje więcej **ramek**.

Mała wartość **PFF** może oznaczać, że proces ma zbyt wiele **ramek**.

☛ Można ustalić **górną** i dolną **granicę** poziomu braków stron w procesie.

Gdy ***częstość braków*** stron: -przekracza **górną granicę**, **przydziela** się dodatkową ramkę.
-spada poniżej ***dolnej granicy***, to usuwa się ramkę z procesu.

☛ Można bezpośrednio mierzyć częstość braków stron i sterować nią przydział ramek.

Jeśli wzrasta liczba braków stron i występuje niedobór wolnych ramek, to trzeba **wstrzymać** wykonanie jakiegoś procesu.

➔ Wolne **ramki** rozdziela się między procesy z ***najwyższymi częstościami*** braków stron.

9.4. Uwagi

Na początku działania procesu występuje duża liczba braków stron w schemacie stronicowania na żądanie

Wznawiany jest proces usunięty z PAO: -wszystkie jego strony znajdują się na dysku,
-powrót każdej z nich następuje po wykryciu jej braku.

Stronicowanie wstępne (*prepaging*) może zapobiegać wysokiej aktywności stronicowania we wstępnej fazie procesu.

Strategia zakłada jednorazowe wprowadzenie do pamięci wszystkich stron, o których wiadomo, że będą potrzebne.

Czy koszt **stronicowania wstępnego** jest mniejszy niż koszt obsługi braków stron?

Czy wszystkie strony sprowadzone do PAO poprzez stronicowanie wstępne zostaną użyte?

Jak wybrać rozmiar strony?

◗ **Jednym** z **aspektów** jest **wielkość TablicyStron**

Zmniejszanie rozmiaru strony to **zwiększanie liczby** stron i rozmiaru TablicyStron.

W pamięci wirtualnej o rozmiarze **4** MB będzie **4096** stron 1024 B, lub **512** stron 8192 B.

Duże strony są wskazane, gdyż każdy aktywny proces ma własną TablicyStron.

Małe strony dają *lepsze wykorzystanie* pamięci gdyż maleje fragmentacja wewnętrzna.

◗ **Czas potrzebny** na **odczytanie** i **zapisanie strony**

Czas operacji We/Wy składa się z czasu ***wyszukiwania***, ***ustawiania*** i ***przesyłania***.

Przeczytanie strony 1024-bajtowej zabierze np. 28 ms, podczas gdy przeczytanie tej samej liczby bajtów w postaci dwu stron 512-bajtowych potrwałoby 56 ms.

Większe rozmiaru strony ułatwiają minimalizację czasu operacji We/Wy.

Większe rozmiary stron pozwalają na minimalizację liczby braków stron.

Mniejsze pozwalają na ***lepsze*** rozłożenie informacji, co może zmniejszyć ilość operacji We/Wy.

Mniejsze strony dają się dokładniej dopasowywać do **stref programu**.

Mniejsze strony dają **lepszą rozdzielczość**, co pozwala na wydzielanie tylko tych fragmentów pamięci, które są rzeczywiście potrzebne.

Mniejszy rozmiar strony może ograniczyć ilości informacji przekazywanej na We/Wy i zmniejszyć ogólny rozmiar przydzielanej pamięci.

Pewne czynniki (wewnętrzna fragmentacja, strefy programu) przemawiają za małymi rozmiarami stron.

Inne czynniki (rozmiar tablic, czas zajmowany przez operacje We/Wy) uzasadniają stosowanie stron o większych rozmiarach.

Stronicowanie na żądanie powinno być przezroczyste dla programu użytkownika.

Znając ideę stronicowania PAO można czasami poprawić działanie programu.

Niech **strony** mają po **1024** bajty.

Tablice trzymane są w pamięci wierszami.

Wiersz tablicy zajmuje 128 x 8 = 1024 B czyli **stronę**.

SO przydzielił na dane **128 ramek**.

```
type vec = array [1..128] of double
var M: array[1..128] of vec;
{------------------------ wersja a }
for i := 1 to 128 do
        for j := 1 to 128 do
                M[j, i] := sin(i + j);
{ ------------------------wersja b }
for i := 1 to 128 do
        for j:=1 to 128 do
                M[i, j] := sin(i + j);
```

Podczas wykonania programu wystąpią braki **stron**:

dla wersji **(a)** 128 x 128 = 16 384 razy.

dla wersji **(b)** 128 razy

Dobór struktur danych i przemyślana konstrukcja programu może spowodować, że strefy programu zacieśnią się.

Może to wpływać na zmniejszenie częstości braków stron i liczby stron w zbiorze roboczym.

❒ **Proces wysyła zamówienie** na **operację We/Wy** i **czeka** w kolejce do urządzenia.

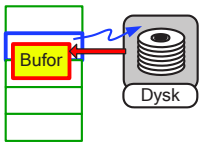
W międzyczasie CPU zostaje przydzielony innym **procesom**.

W **procesach** tych pojawiają się braki stron i -w wyniku działania algorytmu zastępowania globalnego- jeden z procesów zastępuje stronę zawierającą **bufor pamięci** procesu czekającego na operację We/Wy.

Strona zostaje wysłana do pamięci dyskowej.

Po pewnym czasie operacja We/Wy prześle dane pod wskazany ***uprzednio adres***.

☞Jednak z **ramki**, przeznaczonej na bufor, korzysta teraz **strona** należącą do innego procesu.



Dwa możliwe rozwiązania tego problemu:

❶ Zakaz wykonywania operacji We/Wy wprost do pamięci użytkownika.

Aby zapisać blok danych na dysku, najpierw kopiuje się go do pamięci SO i stamtąd na dysk.

Operacje We/Wy są wykonywane tylko między pamięcią **systemu** a urządzeniami We/Wy.

❷ Umożliwienie **blokowania stron** w pamięci.

Każdej **ramce** przyporządkowuje się **bit blokowania**.

Jeśli ramka jest ***zablokowana***, to ***nie bierze*** udziału w zastępowaniu stron.

Dalsze działanie systemu przebiega bez zmian.

Stron zablokowanych nie można zastępować, zwalnia się je dopiero po zakończeniu operacji.

❒ **W niskopriorytetowym procesie zabrakło strony**

Po wybraniu **ramki** do zastąpienia, system stronicujący ***wczytuje stronę*** do pamięci.

Gotowy do działania **niskopriorytetowy** proces czeka w kolejce procesów gotowych, lecz nie otrzymuje CPU ze względu na niski priorytet;

w tym czasie inny proces - **o wysokim priorytecie** -wykazuje brak strony.

☛ System typuje stronę pozostającą w pamięci, do której ***nie było*** odniesień.

☛ Może nią być strona ***nisko****priorytetowego* procesu, ***dopiero co*** sprowadzona.

Kwestią polityki jest decyzja czy proces wysokopriorytetowy może ***zabrać*** stronę procesowi **nisko**priorytetowemu.

Bit blokowania umożliwia ochronę nowo sprowadzonej strony przed zastąpieniem do czasu przynajmniej jednokrotnego jej użycia.

Po wybraniu strony do zastąpienia jej ***bit blokowania*** zostanie ustawiony do czasu, aż oczekującemu procesowi zostanie przydzielony CPU.

Bit blokowania może być niebezpieczny, gdy po jego ustawieniu nie nastąpi zerowanie.

☛ Zablokowana ramka może nigdy nie być udostępniona.

System operacyjny komputera Macintosh stosuje blokowanie stron, gdyż jest to system dla jednego użytkownika i nadmiar zablokowanych stron zaszkodziłby tylko jednemu.

System SunOS dopuszcza ***„zalecenia"*** blokowania, które można odrzucić, jeśli pula wolnych stron staje się zbyt mała lub jeśli dany proces próbuje zablokować zbyt wiele stron w pamięci.