L10.Z1. Rozważamy zarządzanie pamięcią wirtualną. Kiedy system korzysta z polityki **ładowania**, **przydziału miejsca**, **zastępowania** i **usuwania**? Porównaj **stronicowanie wstępne** (ang. *prepaging*) i **stronicowanie na żądanie** (ang. *demand paging*). Jakie zachowania programów będą efektywnie obsługiwane przez jedną, a jakie przez drugą politykę?

Kiedy system korzysta z polityki ładowania, przydziału miejsca, zastępowania i usuwania?

polityka pobierania / sprowadzania (ang. *fetch*) – określa, kiedy strona powinna być wstawiona do pamięci fizycznej. Sytuacja: uruchamiamy nowy program, więc najpierw musimy go wczytać. Ale czy potrzebujemy od razu całego jego kodu i wszystkich danych w pamięci operacyjnej?

polityka przydziału / rozmieszczenia (ang. *placement*) – określa miejsce, w które sprowadzamy strony z pamięci drugorzędnej do RAM.

Przydatne, gdy mamy system tylko z segmentacją i nasze segmenty są różnej wielkości. Niezbyt istotne aktualnie, jeśli system ma stronicowanie, bo nie ma tego problemu. Możliwe alternatywy to first-fit, next-fit, best-fit itd. Omawiane wcześniej.

Przypomnienie:

- stronicowanie - pamięć fizyczna dzielona jest na bloki stałej długości zwane ramkami, pamięć logiczna dzielona jest na bloki stałej długości zwane stronami, rozmiary stron i ramek są identyczne.
  Profit: brak problemu fragmentacji zewnętrznej.
  Problem: narzut czasowy przy translacji adresu, niewielka fragmentacja wewnętrzna.
- segmentacja – dynamiczny podział pamięci na segmenty lub sekcje; referencja do miejsca w pamięci zawiera identyfikator segmentu i offset segmentu

Inne kwestie:

- systemy z niejednorodnym dostępem do pamięci (ang. *Non-Uniform Memory Architecture*) – do rozproszonej, współdzielonej pamięci komputera mogą pojawiać się odwołania z dowolnego procesora w komputerze. Jednak czas dostępu do określonej lokalizacji fizycznej zmienia się wraz z odległością pomiędzy procesorem a modułem pamięci. Stąd wydajność jest bardzo uzależniona od odległości między danymi a procesorem. Strategia: przypisujemy strony do tego modułu pamięci, który gwarantuje najlepszą wydajność.

- kolorowanie stron (ang. *page colouring*) - ułożenie danych w RAM może wpływać na liczbę chybień w pamięć cache.

  Idea: mamy sobie cache, który jest adresowany adresami fizycznymi. No i procesor robi tak, że sąsiednie strony w pamięci fizycznej są mapowane na osobne linie cache'a, co jest całkiem mądre. Ale ale, przecież jak mamy proces, to adresujemy wirtualnymi adresami, no i dwie sąsiednie strony w wirtualnej to nie sąsiednie w fizycznej. Czyli może być tak, że dwie sąsiednie wylądują w tej samej linii cache - ups, spacial locality zjebane, bo wybijają się nawzajem z cache. Ale page coloring to the rescue! Przypisujemy kolory stronom fizycznym i teraz jak przydzielamy dwie strony wirtualnej pamięci obok siebie, to mówimy alokatorowi, żeby przypadkiem nie dał nam tego samego koloru, bo będzie lipa.

- duże strony (ang. *huge pages*) - zwiększanie zasięgu TLB, co powoduje mniej chybień w pamięciożernych programach

polityka zastępowania (ang. *replacement*) – wybór strony w pamięci operacyjnej, która ma zostać przeniesiona na dysk, po to, by można było wstawić nową stronę. Wszystkie strategie zakładają, że strona, która ma zostać wymieniona, powinna być stroną, co do której istnieje najmniejsze prawdopodobieństwo wystąpienia odwołania w najbliższej przyszłości.

Chcemy zmniejszyć zbiór roboczy programu, potrzebujemy wyznaczyć ramki stron, których proces nie będzie już potrzebować. Będziemy wybierać strony-ofiary (ang. *victim*), a następnie wyrzucać je lub uspójniać z pamięcią drugorzędną.

- OPT – usuwamy stronę, do której odwołanie wystąpi najpóźniej (niemożliwy do zaimplementowania)
- FIFO – usuwamy stronę, która była najdłużej w pamięci
- LRU (ang. *least recently used*) – usuwamy stronę, której nie używaliśmy najdłużej
- NRU (ang. *not recently used*) – usuwamy stronę, która jest zmodyfikowana, ale do której nie było ostatnio dostępu (zostawiamy te, do których dostępujemy często)
- drugiej szansy - modyfikacja FIFO; usuwamy najstarszą nieużywaną stronę (albo najstarszą, jeśli wszystkie są używane)
- buforowanie stron – FIFO + wymieniana strona nie jest usuwana, ale przypisywana do jednej z dwóch list: stron wolnych, jeśli nie była zmodyfikowana, (i można tam wstawić strony) lub modyfikowanych
- zegarowy, wsclock, postarzanie stron

polityka usuwania / sprzątania (ang. *clean*) – służy do określenia, kiedy zmodyfikowana, ale dawno nieużywana strona powinna zostać zapisana (uspójniona) w pamięci drugorzędnej.
Most replacement algorithms simply return the target page as their result. This means that if target page is dirty (that is, contains data that have to be written to the stable storage before page can be reclaimed), I/O has to be initiated to send that page to the stable storage (to clean the page).

- **czyszczenie na żądanie**, leniwie (ang. *demand cleaning*) - uspójniamy stronę dopiero, gdy została wybrana do wymiany

- **czyszczenie wstępne**, gorliwie (ang. *precleaning*) - zbieramy strony w grupy i wyrzucamy na dysk, może już nie będą potrzebne

Lepsze podejście: buforowanie stron.

Porównaj stronicowanie wstępne i stronicowanie na żądanie. Jakie zachowania programów będą efektywnie obsługiwane przez jedną, a jakie przez drugą politykę?

- **stronicowanie na żądanie**, leniwie (ang. *demand paging*) – gdy wystąpi błąd strony, ładujemy ją z pamięci. Problem: gdy błędów jest wiele na krótkim odcinku czasu.

  Strona jest wstawiana do pamięci operacyjnej, tylko jeśli pojawia się odniesienie do lokalizacji w tej stronie. Gdy rozpoczyna się proces, pojawia się wiele błędów stron. Wraz z wstawieniem coraz większej liczby stron do pamięci operacyjnej, większość odwołań nawiązuje do adresów we właśnie wstawionych stronach (zasada lokalności). Stąd, po jakimś czasie liczba błędów stron powinna być mała.

- **stronicowanie wstępne**, gorliwie (ang. *prepaging*) - czytanie z wyprzedzeniem.

  Dyski działają wolno, wykonują się obroty talerzy, więc moglibyśmy robić coś podobnego co robimy w cache - jak ładujemy stronę, to załadujmy jeszcze kilka stron, które są tuż po niej (oczywiście zakładając, że są obok siebie na dysku, bo właśnie te "obroty talerza" chcemy zaoszczędzić)

  Problem: gdy nie korzystamy z tych stron. Profit: gdy startujemy program, bo z reguły właśnie wtedy potrzebujemy stron obok siebie.

L10.Z2. Czym różni się **odwzorowanie pamięci anonimowej** (ang. *anonymous mapping*) od **odwzorowania pliku na pamięć** (ang. *memory-mapped I/O*)? Rozważmy proces, który utworzył dziecko z użyciem wywołania `fork`. Opisz działanie mechanizmu **kopiowania przy zapisie** (ang. *copy-on-write*) dla odwzorowań **prywatnych**. Wszystkie procesy korzystające z pewnego **dzielonego** odwzorowania pliku na pamięć zakończyły swe działanie. Co w takim przypadku musi zrobić system? Strony których odwzorowań można przenieść do **przestrzeni wymiany** (ang. *swap space*)?

Czym różni się odwzorowanie pamięci anonimowej od odwzorowania pliku na pamięć?

Oba wykonujemy przy użyciu `mmap(addr, length, prot, flags, fd, offset)`. Mmap tworzy nowe odwzorowanie w wirtualnym przadrze wołającego procesu.

odwzorowanie pamięci anonimowej – nie jest wspierane przez żaden plik.

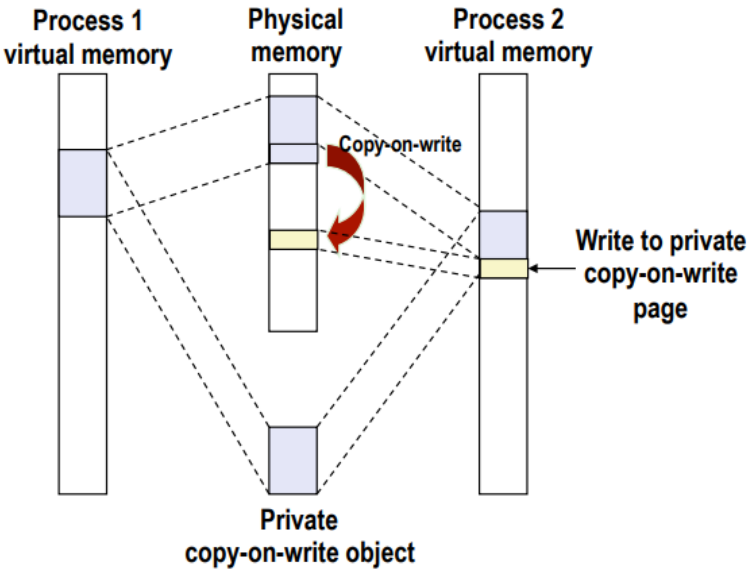
- zawartość jest inicjowana na 0, gdy raz coś tam zapiszemy, jest już jak każda normalna stronal
- przydzielamy z flagą `MAP_ANONYMOUS`
- argument `fd` jest ignorowany, choć niektóre implementacje wymagają ustawienia go na -1
- `offset` powinien być ustawiony na 0
- In this respect an anonymous mapping is similar to malloc, and is used in some malloc(3)implementations for certain allocations.

odwzorowanie pliku na pamięć – odwzorowujemy `length` bajtów pliku o deskryptorze `fd`, zaczynając od offsetu `offset`. Operacja ta powoduje, że do obszaru pliku można odnosić się jak do zwykłej tablicy bajtów w pamięci, eliminując potrzebę korzystania z dodatkowych wywołań systemowych typu `read` czy `write`.

Opisz działanie mechanizmu kopiowania przy zapisie dla odwzorowań prywatnych.

kopiowanie przy zapisie – strategia optymalizacyjna. Idea: gdy kilka procesów potrzebuje zasobów, które początkowo są nierozróżnialne, dajmy im wskaźniki do tego samego zasobu. Utrzymujmy ten stan, dopóki jeden z procesów nie spróbuje zmodyfikować swojej „kopii” zasobu – wtedy trzeba utworzyć faktyczną, prywatną kopię, żeby inne procesy nie widziały tych zmian. Zaleta: jeśli nikt nie zmodyfikuje zasobu, nigdy nie będzie trzeba robić kosztownych kopii.

Wykorzystywane przez sysopek m.in. w celu szybszego wykonywania `fork`. `fork` duplikuje proces wraz z jego pamięcią. Ponieważ kopiowanie pamięci jest czasochłonne, dlatego zamiast kopiowania system operacyjny - korzystając z mechanizmu stronicowania — początkowo odwzorowuje przestrzeń adresową procesu potomnego na pamięć fizyczną zarezerwowaną dla procesu nadrzędnego. Strony pamięci, które mogą być zmodyfikowane zarówno przez proces jak i jego potomka, otrzymują znacznik "kopiowane przy zapisie". Gdy jeden z procesów modyfikuje pamięć, kernel przechwytuje to wywołanie i kopiuje modyfikowane strony tak, aby zmiany dokonane przez jeden proces były niewidoczne dla drugiego. Od tej chwili proces nadrzędny i potomny zaczynają odwoływać się do fizycznie różnych stron.



odwzorowanie prywatne - mapowanie może być prywatne (flaga `MAP_SHARED`) lub dzielone (`MAP_PRIVATE`). Zmiany w prywatnym nie są widoczne dla odwzorowań tego samego pliku innych procesów. Wpisy w tablicy stron odwzorowania prywatnego są oznaczone jako read-only. Próba zapisu do prywatnej strony wywołuje protection fault.

Wszystkie procesy korzystające z pewnego dzielonego odwzorowania pliku na pamięć zakończyły swe działanie. Co w takim przypadku musi zrobić system? Strony których odwzorowań można przenieść do przestrzeni wymiany?

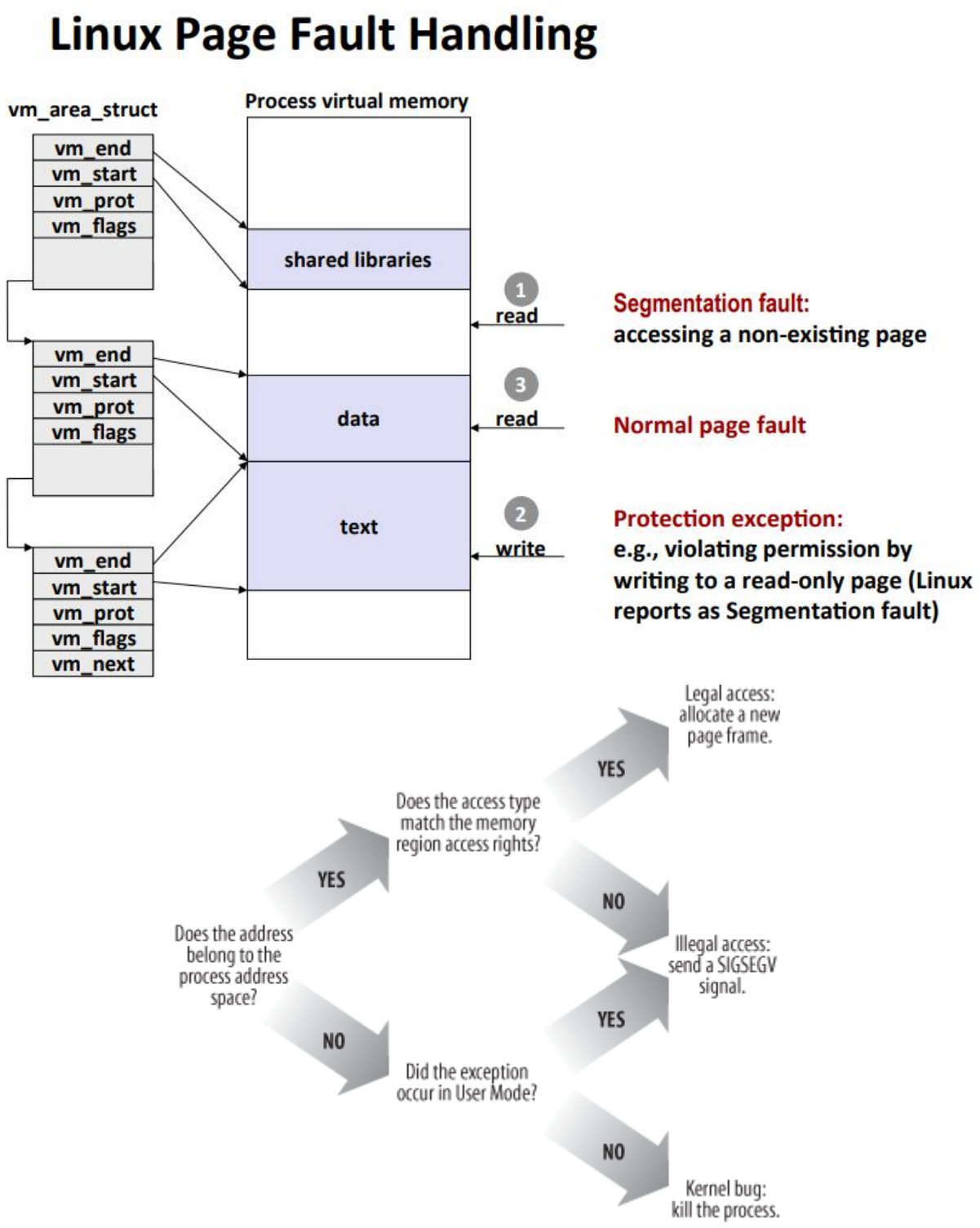
odwzorowanie dzielone – z flagą `MAP_SHARED`. Zmiany w tym odwzorowaniu są widoczne dla innych procesów odwzorowujących ten sam obszar i nanoszone (w przypadku mapowania pliku) do pliku.

przestrzeń wymiany – przestrzeń na dysku używana do tymczasowego przechowywania stron, gdy pamięć RAM jest już pełna. Na Linuxie jest na to osobna partycja. Dla standardowych 4-8GB RAM ma ona zazwyczaj rozmiar 4GB. Nie powinna być traktowana jako dodatkowy RAM, bo dostęp zajmuje więcej czasu. Przenosi się tam głównie strony z odwzorowań anonimowych, bo nie można ich ujednolicić z pamięcią.

System musi wyczyścić strony współdzielone, poczekać aż ta operacja się zakończy i wyrzucić te strony z pamięci operacyjnej.

L10.Z3. Na podstawie slajdów do wykładu „CSAPP3e: Virtual Memory Systems” wyjaśnij, w jaki sposób system Linux obsługuje błąd strony. Kiedy jądro wyśle procesowi sygnał `SIGSEGV` z kodem «`SEGV_MAPERR`» lub «`SEGV_ACCERR`»? W jakiej sytuacji wystąpi **pomniejszy** (ang. *minor*) błąd strony lub **poważny** (ang. *major*) błąd strony? W jakiej sytuacji wystąpi błąd strony przy zapisie, mimo że pole «`vm_prot`» pozwala na zapis do **obiektu wspierającego** (ang. *backing object*)?

Na podstawie slajdów do wykładu „CSAPP3e: Virtual Memory Systems” wyjaśnij, w jaki sposób system Linux obsługuje błąd strony.



Kiedy jądro wyśle procesowi sygnał `SIGSEGV` z kodem «`SEGV_MAPERR`» lub «`SEGV_ACCERR`»?

`SEGV_ACCERR` – próba dostępu do strony nieprawidłowymi uprawnieniami, np. twój kod próbuje zapisać do strony, która jest read-only.

`SEGV_MAPERR` – próba dostępu do strony, która nie jest odwzorowana w przestrzeni adresowej, np. dereferencja null pointera.

W jakiej sytuacji wystąpi pomniejszy błąd strony lub poważny błąd strony?

Gdy nie znaleziono strony w tablicy stron procesu, dochodzi do powstania błędu.

pomniejszy błąd strony (ang. *minor page fault*) - strona może fizycznie być przechowywana w pamięci, ale nie ma jej w tablicy stron procesu. Strona mogła być np. załadowana do pamięci z dysku przez inny proces. W tym przypadku nie ma potrzeby, by sięgać do dysku ponownie. Wystarczy odpowiednio zmapować stronę w tablicy stron.

poważny błąd strony (ang. *major page fault*) - gdy zachodzi konieczność sprowadzenia strony z dysku

błąd segmentacji (ang. *segmentation fault*) - w programie nastąpiła próba dostępu do nieprawidłowego adresu i nie ma potrzeby dodawania mapowania do TLB - sysopek zabija taki proces

W jakiej sytuacji wystąpi błąd strony przy zapisie, mimo że pole «vm_prot» pozwala na zapis do obiektu wspierającego?

Gdy np. strona jest w swap space, a proces chce sobie coś do niej zapisać. W RAM-ie jej fizycznie nie ma, więc page fault.
Ew. może przy copy-on-write.

L10.Z4. Rozważmy system ze stałym rozmiarem **zbioru rezydentnego**. Mamy pamięć fizyczną składającą się z 4 **ramek** oraz pamięć wirtualną złożoną z 8 **stron**. Startujemy z pustą pamięcią fizyczną. Jedyny wykonujący się proces generuje następujący ciąg dostępów do stron:

0, 1, 7, 2, 3, 2, 7, 1, 0, 3

Zaprezentuj działanie **algorytmów zastępowania stron** FIFO i LRU. Który z nich generuje najmniej **błędów stron**? Załóż, że zastąpienie ramki zachodzi dopiero w momencie zapełnienia całej pamięci fizycznej.

Definicje

stronicowanie - pamięć fizyczna dzielona jest na bloki stałej długości zwane **ramkami**, pamięć logiczna dzielona jest na bloki stałej długości zwane **stronami**, rozmiary stron i ramek są identyczne.

algorytmy zastępowania stron – ich zadaniem jest wybór strony do wymiany

błąd strony – sytuacja, w której nie znaleziono strony w tablicy stron procesu

zbiór roboczy (ang. *working set*) - to zbiór stron, których proces potrzebował w chwili t do działania w ciągu Δ ostatnich tyknięć wirtualnego zegara. Podzbiór rezydentnego w delcie czasu.

zbiór rezydentny (ang. *resident set*) - to zbiór wszystkich stron procesu rezydujących w pamięci operacyjnej w chwili t wirtualnego zegara procesu. Sysopek zarządza zbiorem rezydentnym i stara się przybliżać nim zbiór roboczy. (Zbiór rezydentny to to co proces trzyma w RAM-ie, wszystkie strony które sobie zadeklarował, jakby zamrozić całkowicie proces, i spojrzeć na cały RAM jaki on używał, to to jest zbiór rezydentny).

szamotanie (ang. *trashing*) - to stan procesu, w którym spędza on więcej czasu na oczekiwaniu na brakujące strony pamięci, niż na faktycznym wykonywaniu obliczeń, co znacząco spowalnia jego działanie.

Zaprezentuj działanie algorytmu zastępowania stron FIFO.
0: 1: 7: 2: (2, 7, 1, 0) (new <- old)
3: (2, 7, 1, **3**)
2: hit
7: hit
1: hit
0: (2, 7, **0**, 3)
3: hit

Zaprezentuj działanie algorytmu zastępowania stron LRU.
0: 1: 7: 2: (2, 7, 1, 0)
3: (2, 7, 1, **3**)
2: hit
7: hit
1: hit
0: (2, 7, 1, **0**)
3: (**3**, 7, 1, 0)

Który z nich generuje najmniej błędów stron?
Pewnie zależy od zestawu żądań…wg książki LRU jest bliski OPT-owi.

L10.Z5. Wyjaśnij działanie algorytmu **drugiej szansy**. Następnie rozważ przykład z obrazu 3-15(b) w podręczniku Tanenbaum. Niech bity R stron, począwszy od B a kończąc na A, wynoszą odpowiednio:

1, 1, 0, 1, 1, 0, 1, 1

Która ze stron zostanie zastąpiona w trakcie obsługi następnego błędu strony? Jak będzie działał algorytm, jeśli wszystkie bity są ustawione na 1?

Wyjaśnij działanie algorytmu drugiej szansy

algorytm drugiej szansy – prosta modyfikacja algorytmu FIFO, która zapewnia uniknięcie problemu wyrzucenia często używanej strony. Polega na zbadaniu bitu referenced najstarszej strony. Jeżeli ma on wartość 0, oznacza to, że strona jest nie tylko stara, ale i nieużywana, zatem zostaje zastąpiona natychmiast. Jeśli bit R ma wartość 1, jest on zerowany, a strona zostaje umieszczona na końcu listy stron (jakby właśnie została dodana do pamięci). Bit R oznacza jakby liczbę szans, które pozostały danej stronie.

Która ze stron zostanie zastąpiona w trakcie obsługi następnego błędu strony?

D, czyli pierwsza napotkana z bitem R ustawionym na 0.

Jak będzie działał algorytm, jeśli wszystkie bity są ustawione na 1?

Jeśli do wszystkich stron były odwołania, algorytm degeneruje się do klasycznego FIFO. Sysopek będzie po kolei przenosił strony na koniec listy, zerując bit R za każdym razem, kiedy strona zostanie dodana na koniec listy. Ostatecznie powróci do strony B, która ma teraz wyzerowany bit R. W tym momencie B jest usunięta z pamięci.

L10.Z6. Rozważmy komputer podobny do MERA-400 posiadający tylko 4 ramki o rozmiarze 4KiB. W pierwszym takcie zegara systemowego bity R ramek są ustawione zgodnie z bitami w ciągu 0111. W kolejnych taktach zegarowych zarejestrowano wartości bitów: 1011, 1010, 1101, 0010, 1010, 1100 i 0001. Używamy algorytmu **postarzania stron** z 8-bitowymi licznikami. Oblicz wartości liczników dla ostatniego taktu zegara systemowego. Następnie proces wywołuje błąd strony – wybierz ramkę ofiarę. Jaka będzie wartość licznika dla nowo wstawionej ramki?

algorytm postarzania stron – stronom zbioru rezydentnego dajemy rejestr przesuwny. Bity dostępu wsuwamy od lewej co cykl wirtualnego zegara. Usuwamy stronę o najmniejszej wartości.

Oblicz wartości liczników dla ostatniego taktu zegara systemowego.

Strona / bity	0111	1011	1101	0010	1010	1100	0001
0	0…	10…	111…	0111…	10111…	110111…	0110111
1	1…	01…	101…	0101…	00101…	100101…	0100101
2	1…	11…	011…	1011…	11011…	011011…	**0011011**
3	1…	11…	111…	0111…	00111…	000111…	1000111

Następnie proces wywołuje błąd strony – wybierz ramkę ofiarę. Jaka będzie wartość licznika dla nowo wstawionej ramki?

Kiedy wystąpi błąd braku strony, z pamięci jest usuwana strona, której licznik ma najmniejszą wartość, tutaj: wytłuszczona ramka. Zastępujemy ją ramką z licznikiem o wartości 1000000.