Implementacja Pamięci Wirtualnej

Piotr Anders
e-mail: p.anders@students.mimuw.edu.pl

Andzej Tomczyk
e-mail: a.tomczyk@students.mimuw.edu.pl

Janusz Dutkowski
e-mail: j.dutkowski@students.mimuw.edu.pl

16 grudnia 2002 roku

Spis treści

1 Wprowadzenie **4**
1.1 Pamięć wirtualna 4
1.2 Alternatywne rozwiązania 4
1.2.1 Nakładki 4
1.2.2 Ładowanie dynamiczne 4
1.3 Sposoby implementacji pamięci wirtualnej 5
1.3.1 Stronnicowanie na rządanie 5
1.3.2 Segmentacja na żądanie 7

2 Pamięć wirtualna w systemach MS Windows **7**
2.1 Wprowdzenie 8
2.2 Adresy Wirtualne 9
2.3 Pamięć zarezerwowana i używana 10
2.4 Translation Lookaside Buffers (TLBs) 10
2.5 Struktura zawartości Page-Table 11
2.6 Dzielenie stron między procesami 12
2.7 Virtual-Memory Manager (VMM) 13

3 FreeBSD 4.x **14**
3.1 Wstęp do zarządzania pamięcią w FreeBSD 14
3.2 Pamięć Wirtualna 16
3.2.1 Algorytm znajdowania strony 16
3.2.2 Obsługa błędu braku strony 16

4 AS/400 **18**
4.1 Wstęp do zarządzania pamięcią w AS/400 18
4.2 Pamięć Wirtualna 19
4.2.1 Algorytm znajdowania strony 19
4.2.2 Obsługa błędu braku strony 21

5 System Mach **21**
5.1 Wstęp do zarządzania pamięcią w Mach 21
5.2 Podstawowe komponenty w systemie Mach 22
5.3 Zarządzanie pamięcią w systemie Mach 22
5.3.1 Wirtualna Przestrzeń Adresowa Tasku 23
5.3.2 Zarządzanie stronami w pamięci fizycznej 24
5.3.3 Interakcja jądra, tasku użytkownika, i external pagera 25
5.3.4 Mapowanie Obiektu 25
5.3.5 Błąd braku strony 25
5.3.6 Wymiana stron 26
5.4 Zalety pamięci wirtualnej z external pagerem w systemie Mach 26

A Bibliografia 28

1 Wprowadzenie

Współczesne systemy operacyjne powinny umożliwiać jednoczesne wykonywanie wielu dużych programów. Procesy związane z programami (i dane, których potrzebują) często nie mieszczą się w całości w pamięci operacyjnej. Potrzebne są techniki umożliwiające wykonywanie procesów będących jedynie częściowo w pamięci operacyjnej, a cześciowo w pamięci pomocniczej (na dysku). Jednocześnie ważne jest, aby z punktu widzenia programisty pamięć była dużą jednolitą przestrzenią adresową, z której korzysta tylko jego program.

1.1 Pamięć wirtualna

- Programista dysponuje ogromną wirtualną przestrzenią adresową. Nie musi martwić się o rozmiar dostępnej pamięci fizycznej, ani o to jak jego program będzie w niej rozmieszczony (wiązanie adresów w czasie wykonywania). Pamięć logiczna jest oddzielona od fizycznej.
- Programy mogą wykonywać się szybciej, gdyż nie muszą w całości być ładowane do pamięci fizycznej.
- Pamięć wirtualną zapewnia system operacyjny. Nie ma (poza wyjątkami – np. Mach) ingerencji z poziomu użytkownika.
- Do implementacji pamięci wirtualnej potrzebne jest dodatkowe zaplecze sprzętowe.

1.2 Alternatywne rozwiązania

1.2.1 Nakładki

- Program może być większy niż dostępna pamięć fizyczna.
- Nie jest potrzebny żaden dodatkowy sprzęt ani wsparcie systemu operacyjnego.
- Programista musi sam zadbać o rozmieszczenie programu w pamięci fizycznej (potrzebne są specjalne algorytmy przemieszczenia i konsolidacji). Kody nakładek są przechowywane jako bezwzględne obrazy pamięci (wiązanie w czasie kompilacji).
- Potrzebny jest dodatkowy moduł obsługi nakładek.
- Programista jest obciążony dodatkową pracą. Musi dobrze znać strukturę programu.
- Nakładki stosowane były tam, gdzie brakowało środków sprzętowych dla innych technik.

1.2.2 Ładowanie dynamiczne

- Cały program może być większy niż dostępna pamięć fizyczna.
- Do pamięci ładowane są tylko wywoływane podprogramy.

- Kod programów jest przemieszczalny – wiązanie adresów następuje w czasie ładowania.
- Podprogram wykonywany może wywołać inny podprogram. Jeżeli tego podprogramu nie ma w pamięci to specjalny program ładujący musi go załadować i uaktualnić tablicę adresów.
- System może, ale nie musi wspierać ładowania dynamiczego dostarczając odpowiednie biblioteki. Zadanie tak czy inaczej spoczywa na programiście.

1.3 Sposoby implementacji pamięci wirtualnej

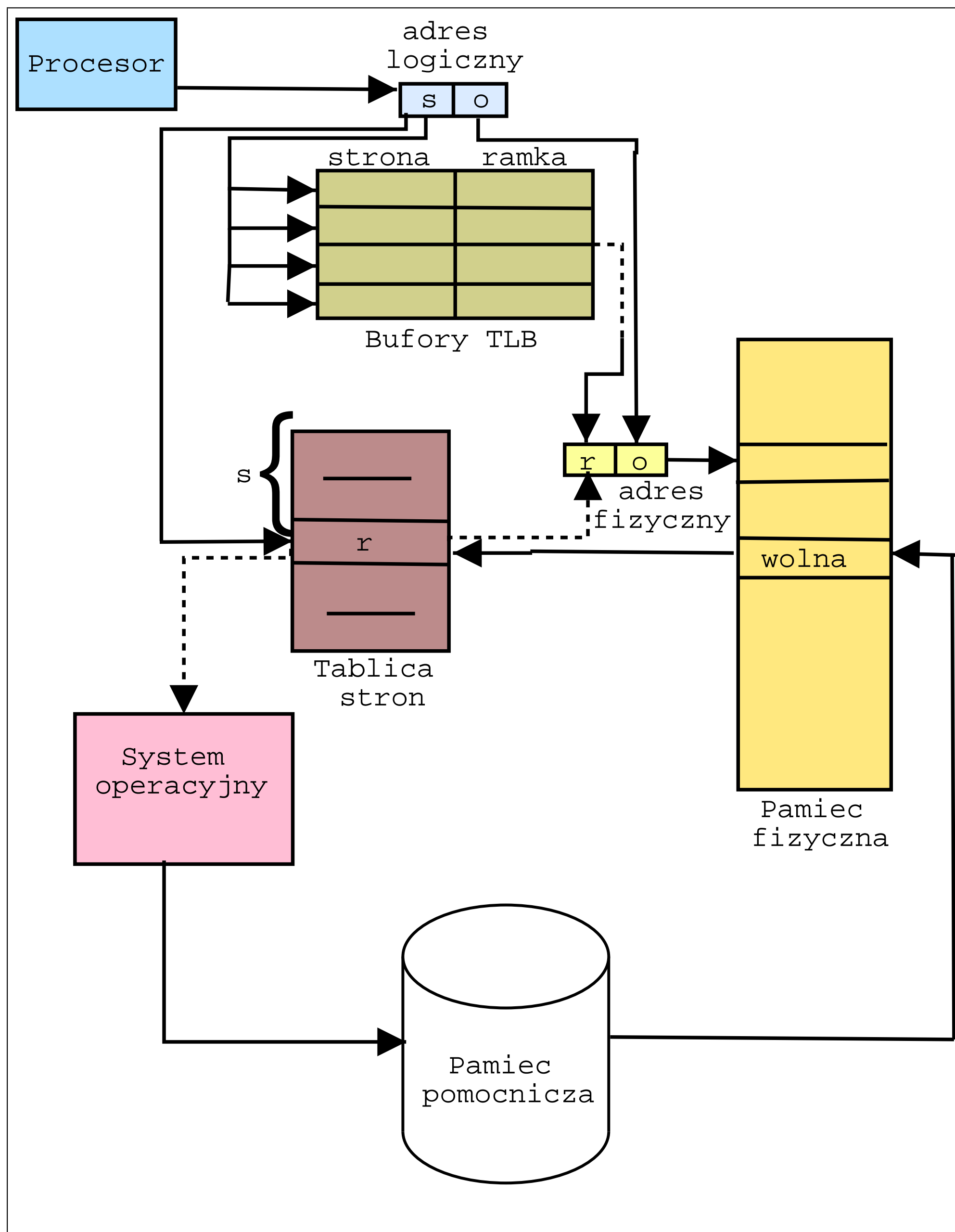
1.3.1 Stronnicowanie na rządanie

- Najwydajniejszy i najcześciej spotykany we współczesnych systemach sposób implementacji pamięci wirtualnej.
- Wymaga specjalnego sprzetu w postaci pamięci pomocnicznej (urządzenia wymiany) i tablicy stron różnie implementowanej w systemach przy pomocy:
  - rejestru bazowego tablicy stron – PTBR (page-table base register)
  - rejestrów asocjacyjnych – TLB (transaction look-aside buffers)
  - pamięci operacyjnej

Przy realizacji stronicowania na żądanie, trzeba rozwiązać kilka podstawowych problemów:

Wybór algorytmu wymiany stron Podstawowym problemem przy implementacji stronnicowania na rządanie jest wybór strony, która ma zostać odesłana do pamięci pomocniczej. Oceną poprawności algorytmu jest częstość błędu braku strony. Często wybór algorytmu zależy od dostępnego sprzętu. Znane algorytmy to:

- FIFO – anomalia Belady'ego
- Optymalny
- LRU
- FIFO z drugą szansą
- Dodatkowe bity odwołań
- LFU
- MFU



Rysunek 1: Stronicowanie

Przydział ramek

- minimalna liczba ramek – określona przez zbiór rozkazów komputera
- rówmy przydział
- proporcjonalny przydział
- globalny przydział
- lokalny przydział

Inne problemy

- szamotanie
- stronicowanie wstępne
- rozmiar strony – zależne bardziej od sprzętu. (problem wewnętrznej fragmentacji vs. marnowanie pamięci na tablice stron)

1.3.2 Segmentacja na żądanie

- Używane wobec braku odpowiedniego sprzętu np. OS/2 na Intel 80286.
- Przybliża stronnicowanie na żądanie.
- Segmenty opisane są przez deskryptory segmentów z bitami poprawności i odniesienia tak jak przy stronnicowaniu.
- OS/2 może upakowywać segmenty w pamięci.
- procesy moga informować system, które segmenty mogą być usuniete, a które muszą pozostać. (Zmienia się w ten sposób pozycja w kolejce do usunięcia z pamięci.)

2 Pamięć wirtualna w systemach MS Windows

Implementacja managera pamięci wirtualnej w systemach rodziny Windows 32 (czyli 95/98/2000) oraz Windows NT jest oparta na tych samych rozwiązaniach, będzie więc omawiane zarządzanie nią we wszystkich tych systemach na raz.

Kolejno szczegółowo rozwiniemy tematy:

- 32-bitowe adresy wirtualne
- Stronicowanie
- Pamięć zarezerwowana i użyta (commited)
- Obsługa błędów stron
- Dzielenie stron
- Optymizacja Copy-on-write
- Zarządzanie zbiorem stron procesu

2.1 Wprowdzenie

System Windows dzięki mechanizmom stronicowania pozwala każdemu procesowi na 32-bitowe liniowe adresowanie 4GB pamięci wirtualnej, z czego jednak górne 2GB są zarezerwowane dla systemu. Menadżer systemowy wirtualzując pamięć sprawia, że proces 'widzi' zawsze 2GB dostępnej pamięci niezależnie od tego ile pamięci fizycznej jest zainstalowane w komputerze. Robi to działający w tle osobny proces 'memory manager' zawierający kilka wątków stale zarządzających dostępnymi zasobami.

Windows w wersji 3.x był ograniczony do maksymalnej ilosci pamięci dostępnej dla wszystkich aplikacji, co było barierą dla dużych aplikacji. W nowych systemach ograniczenie jest praktycznie na granicy ilości wolnego miejsca na dysku przechowującym pliki wymiany (memory-backing swapfile (pamięcio- plecujące???))

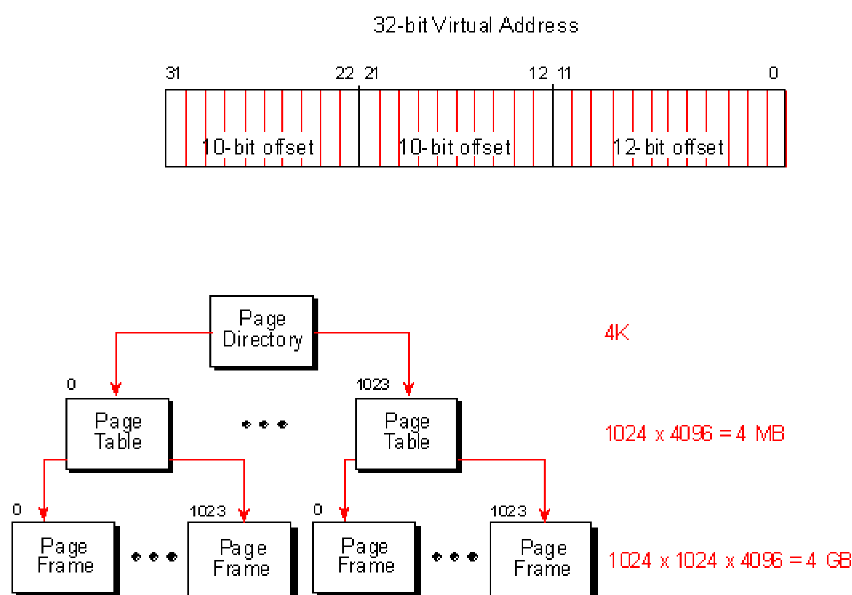
W wersjach 3.1 i wcześniejszych, segmenty pamięci były przesuwane w inne miejsce aby utworzyć większe partie ciągłej wolnej pamięci i by umieszczać wykonywalne bloki. W Windows 32 i NT nie ma takiej potrzeby z trzech powodów.

1. Segenty kodu nie muszą już być umiesczane w pierwszych 640K ponieważ system wymaga conajmniej 32-bitowej szyny adresowej więc może się dostać do dowolnej części fizycznej pamięci.
2. VMM (Virtual Memory Manager) sprawia, że dwa procesy mogą używać tego samego wirtualnego adresu dla dostępu do rożnych komórek pamieci fizycznej. Procesy więc mogą używać całej dostępnej pamięci nie bacząc na konflikt z innymi procesami.
3. Ciągłe bloki mogą być allokowane nieciągle w pamięci fizycznej.

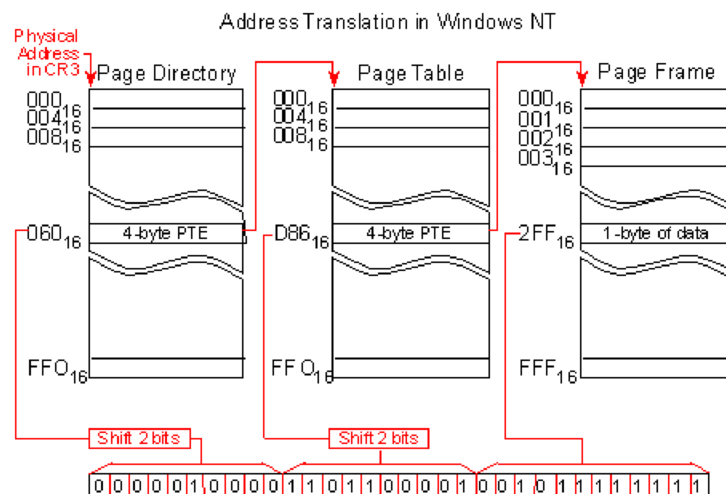
VMM dzieli pamięć na 4 kilowe (4096b) strony, niezależnie od tego co się w niej znajduje (kod, dane, zasoby, pliki itd.) Pamięć zapisana, ale chwilowo nie używana jest trzymana na dysku w plikach zwanych 'pagefiles', które zmieniają swoją wielkość dynamicznie w zależności od zapotrzebowania na pamięć.

2.2 Adresy Wirtualne

Dokładnie tak jak w Linux'ie omawianym na wykładach, adres wirtualny dzielony jest na trzy części. Dwie 10-o i jedną 12-o bitową. Pierwsza to offset wskazujący na 4-bajtową wartość w stronie pamięci nazywanej 'page directory', jedną i unikalną dla każdego procesu. Jest to jedna z 1024 (4K strona) wartości PDE (page-directory entries), która wskazuje na koleją stronę w pamięci nazywaną 'page table'. W tej identycznie drugie 10- bitów wsazuje na właściwą stronę w pamięci zwaną 'page frame', w której pozycję wyznacza ostatni 12-bitów wirtualnego adresu.



Aby więc zaadresować całe 4GB zużywamy około 4MB na powyższą strukturę. Oczywiście odwołania do stron w tablicach i strony tablic nie są tworzone dopóki nie są potrzebne (aplikacja nie zgłosi zapotrzebowania na pamięć). Szczegółowy algorytm tłumaczenia adresu wirtualnego jest chyba jasny (i było omawiany na zajęciach). Takie rozwiązanie zapewnia nam na odseparowanie pamięci procesu (i uniknięcie kolizji), jednak utrudnia współdzielenie pamięci pomiędzy procesami.



2.3 Pamięć zarezerwowana i używana

Konsekwentnie do tych założeń, system musi przechowywać informacje na temat tego, które adresy są używane, a które nie. Istnieje jednak mechanizm rezerwowoania pamięci, co można by porównać do zaznaczenia jej jako uzywanej, chociaż w rzeczywistości tak nie jest, ale będzie nam potrzebna w przyszłości. Fukcja taka jest prezez API udostępniona programistom.

Jest to wykorzystywane na pryzkład przy: rezerwowaniu przestrzeni dla stosu; mapowania dużych plików (plik nie jest wczytywany do pamięci, ale jest rezerwowana pamięć tak jakby był. Dopiero na życzenie pamięć jest uzupełniana z pliku)

Podobnie dzieje się z kodem i blibliotekami DLL. Rezerwowana jest dla nich pamięć, a niekoniecznie cały plik jest wczytywany. Dzięki rezerwacji mamy ciągłość pamięci.

Także możliwe jest zarezerwowanie pamięci dla 'page tables', które jeszcze nie istnieją, gdyż pamięć nie jest jeszcze używana.

2.4 Translation Lookaside Buffers (TLBs)

Równolegle do opisanego powyżej schematu translacji wirtualnego adresu (który wydaje się być czasochłonny) Windows używa możliwosci współczesnych procesorów wyposażonych w Translation Lookaside Buffer. TLB jest to 64 kilowy bufor używany przez sprzęt do dostania się do fizycznego adresu. Windows używa TLB do bezpośredniego połączenia często używanych adresów wirtualnych z odpowiadającymi im stronami. Dzięki temu można dostąć się do nich bez odwiedzania 'page directory' i 'page talbe'.

Jako komponent sprzętowy działa niezależnie i współbierznie do programowego tłumaczenia co oszczędza czas i jest znacznie szybsze. Niestety jest też ograniczony i może sie tam znaleźć nie więcej niż 32 przekierowania.

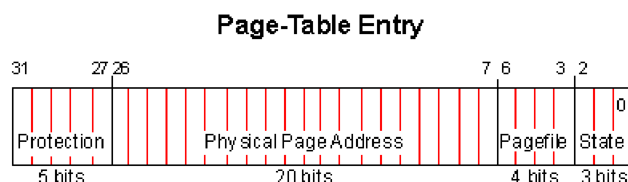
Każde wejście w TLB zawiera wirtualny adres i PTE odpowiedniej strony, a więc różne adresy na tej samej stronie tworzą jedną krotkę w TLB, co troche oszczędza cenne miejsce. Każde tłumaczenie programowe umieszcza nowy rekord w TLB, kiedy się przepełni usuwany

jest najdłużej nieuzywany.

Jednak przełączania pomiędzy procesami może spowodować, że adresy przechowywane w TLB są nieużyteczne dla kolejnych kontekstów, co znacznie obniża użyteczność tej metody. W związku z tym na platformach Intela bufor ten jest automatycznie czyszczony przy przełącznaiu kontekstu (sprzętowo). Lecz w Windows zamiast tego używany jest adres 36- bitowy. Dodatkowe 4 bity identyfikują proces do którego należy wpis w TLB

2.5 Struktura zawartości Page-Table

Jeżeli w zkresie zarezerwowanej pamięci zastanie użyte odwołanie (użycie pamięci), istnieje ona jako strona w pamięci fizycznej lub na dysku. Page-Table (jak i Page-Directory) zawierają informacje na temat jej lokalizacji, ochrony, stanu i ewentualnego składowania wg. schematu na rysunku.



Pierwsze 5 bitów oznaczające ochronę może zawierać znaczniki PAGE_NOACCESS, PAGE_READONLY i PAGE_READWRITE (dla aplikacji w podsystemie Win32).

Następne 20 to fizyczny adres strony (jeśli się tam znajduje). (20-bitów do dowolna 4K storna w 4GB przestrzeni adresowej). Jeśli strona jest przechowywana na dysku, to adres ten wskazuje na offset w odpowiednim pagefile-u.

Kolejne 4 reprezentują jeden z 16 możliwych plików wymiany, który przechowuje tę stronę.

Ostatnie 3 zawierają flagi stanu: pierwszy tranzycji (T- transition); drugi oznacza, że strona została zmieniona (zapis do pamięci) ale nie zachowana (D-dirty); a trzeci jej obecność w pamięci fizycznej. Możliwe stany to:

Table 1. Page-Table Entry Page States

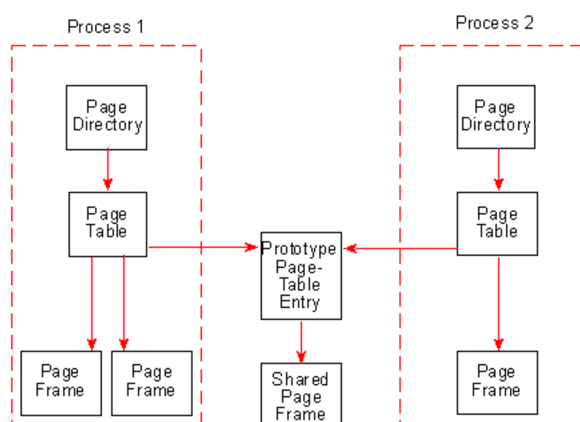
T	D	P	Page state
0	-	0	Invalid page
-	0	1	Valid page
-	1	1	Valid dirty page
1	0	0	Invalid page in transition
1	1	0	Invalid dirty page in transition

Jeśli strona wskazuje na kod lub mapuje plik, który jest na dysku, jej dane nie są zapisywane w żadnym z pagefile-i aby uniknąć nadmiarowości. W taki wypadku PTE z 4 bitów ochrony, 4 dla numeru pagefile-u i 20 adresu tworzy 28-bitowy odnośnik do systemowej struktury danych zawierającej nazwę właściwego pliku i lokalizację w nim dla odpowiedniej strony pamięci.

Błędy Stron
Kiedy system odwołuje się do strony nie istniejącej w pamięci procesor podnosi wyjąteg błędu ‘page-fault’. Wynik jest automatycznie przełączany na ‘Pager’-a, który ładuje stronę z dysku i oddaje sterowanie do orginalnych instrukcji. W ekstremalnym przypadku podczas jednago odwołania mogą zostać wygenerowan aż 3 pag-fault’y. Tu właśnie mocno może poprawić działanie TLB, dzięki którym możemy dwóch z nich uniknąć.

2.6 Dzielenie stron między procesami

Teoretycznie nie ma problemu, aby w tablicach dwóch procesów istniały identyczne PTE wskazujące na tę samą stronę (dzieloną). Gdyby jednak stan takiej strony się zmienił musiano by zmieniać wszystkie takie wejścia (np. kilkanaście :) i musiały by istnieci odwrotne referencje do wszystkich wejść dzielonych stron. Zamiast tego zastosowano mechanizm prototypów PTE, czyli specjalnych wejść stron tablic (prototype page-table entry) wskazujacych na dzieloną stronę, na które wskazują tablice procesów dzielących stronę.



Te prototypowane PTE są zaimpelmentowane jako globalne zasoby systemu w górnych przestrzeniach adresowych wszystkich procesów. Jest dla nich zarezerwowane maksymalnie 8MB dla wszystkich dzielonych stron. Są allokowane dynamicznie, więc miejsce nie marnuje się w przypadku nie istniejących stron. Zauważmy też, że przy odwołaniu do strony dzielonej mogą wystąpić aż 4 page-fault’y.

Optymizacja Copy-on-Write
Normalnie wszystkie strony mają ochronę PAGE_READWRITE. To ułatwia życia na przykład debuggerom, które mogą w kodzie wpisywać swoje znaczniki. Jeśili jednak debuggowany kod

(z wstawionymi break point'ami) jest równocześnie uruchamiany przez inny process może dojść do kolizji. Z drugiej strony trzymanie kopii kodu było by nadmiarowością. Rozwiązaniem jest Copy-on-write. Na podobnej zasadzie co prototypowane PTE strony kodu (zwykle dzielone) mają dodatkową cechę, że mogą być w razie konieczności skopiowane. Odbywa się to gdy wystąpi zapis do pamięci.

2.7 Virtual-Memory Manager (VMM)

VMM jest osobnym procesem odpowiedzialnym za stronicowanie i zarządzanie pamięcią fizyczną i plikami wymiany. Śledzi odwołania do wszystkich stron i ustala on ich aktywny zbiór dla każdego procesu. Jest to wykonywalny komponent systemu wykonywany wyłącznie w trybie jądra. Z oczywistych względów jego kod znajduje się w małej sekcji pamięci nazywanej 'nonpaged pool', która nigdy nie jest wyrzucana z pamięci na dysk.

Page-Frame Database
VMM używa prywatnej struktury danych do zapamiętywania statusu fizycznych stron pamięci zwanej page-frame database. Zawiera dane dotyczącej każdej storny w systemie i ma takie możliwe stany:

Valid Strona jest używana przez aktywny proces. Jej PTE jes zaznaczone jako istniejąca.

Modified Strona została zmieniona ale nie zapisana na dysk.

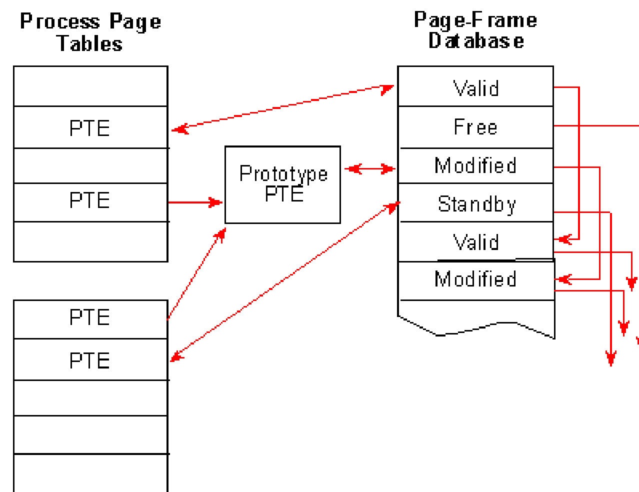
Standby Strona została usunięta z aktualnego zbioru stron procesu.

Free Nie posiada odpowiadającego jej PTE i jest możliwa do użycia. Wcześniej jednak musi być wyzerowana, chyba że ma być read-only.

Zeroed Strona jest wyzerowana i gotowa do natychmiastowego użycia.

Bad Strona generuje błąd sprzętowy i nie może być używana.

Wszystkie strony tego samego typu są połączone w listy. Co umożliwia szybkie wyszukanie np. kolejnych wolnych stron. Rekody w tej bazie są połączone z odpowiadającymi im PTE (tak aby można było szybko wprowadzać zmiany) co widać na poniższym rysunku.



Zarządzanie zbiorem stron procesu
Co pewien czas VMM minimalizuje liczbę używanych stron. W związku z czym zwalnia te, które są w stanie tranzycji i są Modified lub Stand By przenoszone są do Free (po uprzednim zapisaniu zmian i zmianie stanu).

Inna zaś wątek musi decydować, które strony należy wymieniać. Algorytm ten bazuje na ‘przewidywaniu’, która strona nie będzie najwcześniej potrzebna. Robi to na podstawie badania częstości i ostatniego używania poszczególnych stron. Komponent ten nazywa się workin-set manager.

Kiedy proces jest uruchamiany VMM przypisuje mu zbiór stron minimalny dla jego potrzeb (najmniejsza liczba stron wypełniająca potrzeby procesu bez głodzenia innego procesu). W trakcie jego działanie co pewien czas testuje rozmiar zbioru ‘kradnąc’ dobre strony (valid). Jeśli proces działa dalej bez generowania page-fault’a dla tej strony to usuwana jest ona ze zbioru i udostępniana systemowi.

Sama ‘kradzież’ odbywa się w dwóch etapach. Najpierw working- set manager zmienia PTE strony na złą w tranzycji, a potem dopiero zmieniany jest zapis w page-frame database na Modified albo Stand By (w zależności od tego czy była zmieniana).

3 FreeBSD 4.x

3.1 Wstęp do zarządzania pamięcią w FreeBSD

Pamięć w FreeBSD jest widoczna z punktu procesu użytkownika jako jednolita, wyłączna i ciągła przestrzeń adresowa.

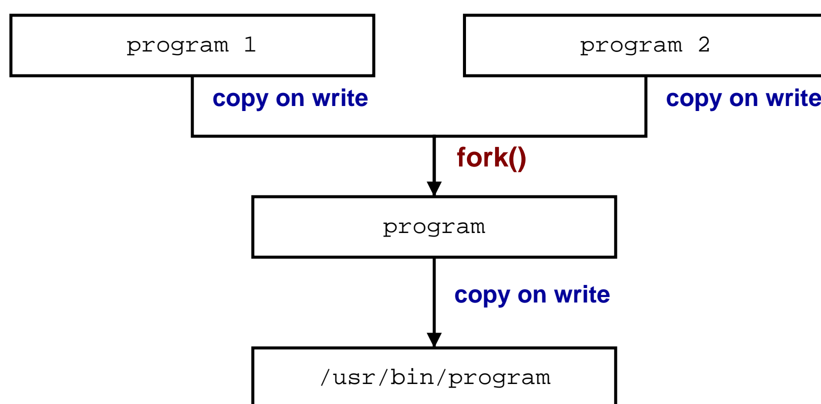
Z punktu podsystemu zarządzania pamięcią jest ona reprezentowana jako zbiór obiektów (`vm_object`). Każdy obiekt składa się ze zbioru stron (`vm_page`), oraz każdy obiekt ma przypisane jak jest wspierany z punktu widzenia systemu wymiany stron:

- nie wspierany (*unbacked*)

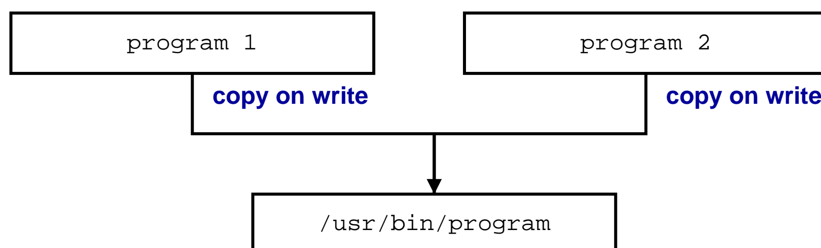
- wspierany przez przestrzeń swap (*swap-backed*)
- wspierany przez system plików (*file-backed*)
- wspierany przez fizyczne urządzenie (*physical device-backed*)

Informacja ta potrzebna jest w sytuacji, gdy strona z danego obiektu ma zostać wymieniona w pamięci. Pozwala ona określić gdzie dana strona ma zostać zachowana i skąd w razie zapotrzebowania pobrana.

Dodatkowo powyższe obiekty mogą 'zakrywać' jeden drugiego umożliwiając stosowanie optymalizacji typu 'copy-on-write'. Oznacza to, że jeśli w danym obiekcie nie ma szukanej strony, to można sięgnąć do obiektu poniżej niego i tam poszukać, a następnie np. przekopiować ją do najwyższego. Aby było to możliwe obiekty tworzą strukturę stosu, a właściwie odwróconego drzewa. Obiekt w jego korzeniu jest obiektem pierwotnym np. kodem programu zmapowanym (file-backed) z dysku. Powyżej zaś znajdują się obiekty stworzone poprzez modyfikacje danych, forkowanie procesów itp...



Oczywiście taki model może prowadzić do gromadzenia się dużej ilości stron marnowanych w obiektach już zakrytych. FreeBSD stosuje więc dodatkowo optymalizację zwaną 'All Shadowed Case'. Polega ona na wykryciu sytuacji, że wszystkie strony danego obiektu zostały już zduplikowane i usunięciu danego obiektu. Na przykład w powyższym przykładzie, jeśli obiekt 'program 2' zduplikował już wszystkie strony obiektu 'program', to możemy 'podpiąć' bezpośrednio do korzenia, a obiekt 'program 1' złączyć z 'program'. Usuwamy w ten sposób całkowicie obiekt 'program'.



Należy zaznaczyć, że proces nie jest związany koniecznie z jednym vm_object. Proces związany jest vm_map, na który może składać się dowolnie dużo vm_object (i innych vm_map). Pozwala to np. na proste mapowanie plików.

Tak zbudowane struktury pamięci wirtualnej pozwalają na dynamiczne usuwanie i odtwarzanie sprzętowych tablic stron. Wszystkie dane potrzebne do zaadresowania strony zawarte są w hierarchii vm_map.

Dodatkowo należy zauważyć, że optymalizacje związane ze stronami to nie tylko 'Copy-On-Write' ale także strony 'Zero-Fill', czyli strony puste, wypełnione zerami, przyłączane dopiero przy odwołaniu.

Fizyczne zarządzanie stronami oparte jest na strukturach (vm_page). Każda odpowiada dokładnie jednej stronie fizycznej i na odwrót. Strona znajduje się zazwyczaj w którejś z następujących kolejek:

- vm_page_queue_free - strony wolne
- vm_page_queue_zero - strony wolne wyzerowane (gotowe do przydzielenia przy żądaniu 'Zero-Fill')
- vm_page_queue_active - strony aktywnie używane
- vm_page_queue_inactive - strony nieaktywne, ale zmienione i nie zapisane na dysk
- vm_page_queue_cache - strony nieaktywne gotowe do natychmiastowego użycia

Strona może być także w stanie 'Wired'. Oznacza to że strona ta nie powinna podlegać systemowi wymiany i nie znajduje się w żadnej z powyższych kolejek. Strony takie to często np. strony zawierające tablice stron.

3.2 Pamięć Wirtualna

3.2.1 Algorytm znajdowania strony

Samo adresowanie w procesie przebiega analogicznie jak np. w Linuksie przy użyciu tablic stron stworzonych z hierarchii vm_map opisanej powyżej.

3.2.2 Obsługa błędu braku strony

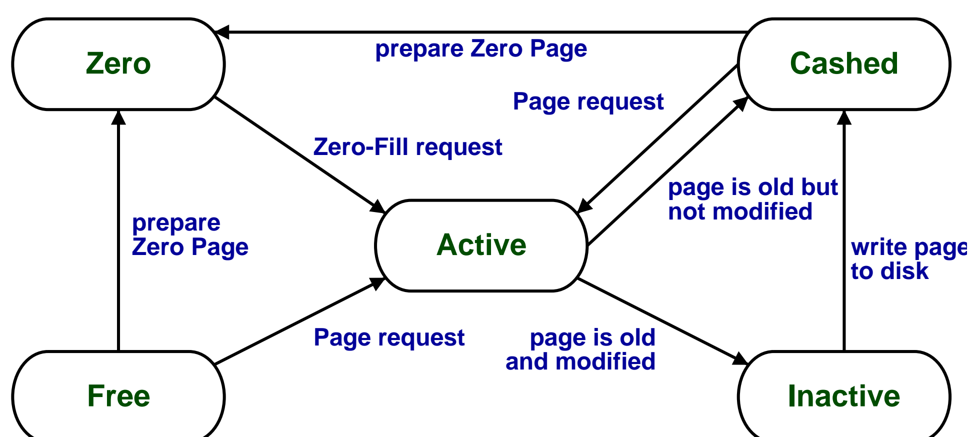
W przypadku błędu braku strony sprawdzane jest najpierw, czy szukana strona nie znajduje się w którejś z kolejek vm_page_queue_inactive lub vm_page_queue_cache. Jeśli tak, to jest przerzucana do kolejki vm_page_queue_active.

Jeśli szukana strona nie jest zbuforowana, to system pobiera stronę z którejś z kolejek vm_page_queue_free lub vm_page_queue_cache (z której to zależy od obciążenia i ilości stron wolnych). Następnie strona ta ładowana jest szukaną stroną z dysku i wstawiana do kolejki vm_page_queue_active.

Aby powyższe było możliwe system dba o odpowiednie zbalansowanie kolejek. Strony aktywne są regularnie przeglądane i postarzane przy użyciu bitu referencji i licznika ‘age’ (Jest oczywiście to odmiana algorytmu LRU). W zależności od obciążenia systemu, strony odpowiednio ‘stare’ przekładane są do odpowiedniej kolejki `vm_page_queue_inactive` lub `vm_page_queue_cache`.

Jeżeli też długość kolejki `vm_page_queue_inactive` osiągnie pewną wartość krytyczną (też zależną od obciążenia systemu) to strony zrzucane są na dysk i przekładane do kolejki `vm_page_queue_cache`.

System dba też w wolnym czasie procesora by mieć w kolejce `vm_page_queue_zero` co najmniej kilka stron wyzerowanych.



Powyższe rozwiązanie oparte na kolejkach gwarantuje zachowanie wysokiej wydajności w szczególności przy wysokim obciążeniu systemu.

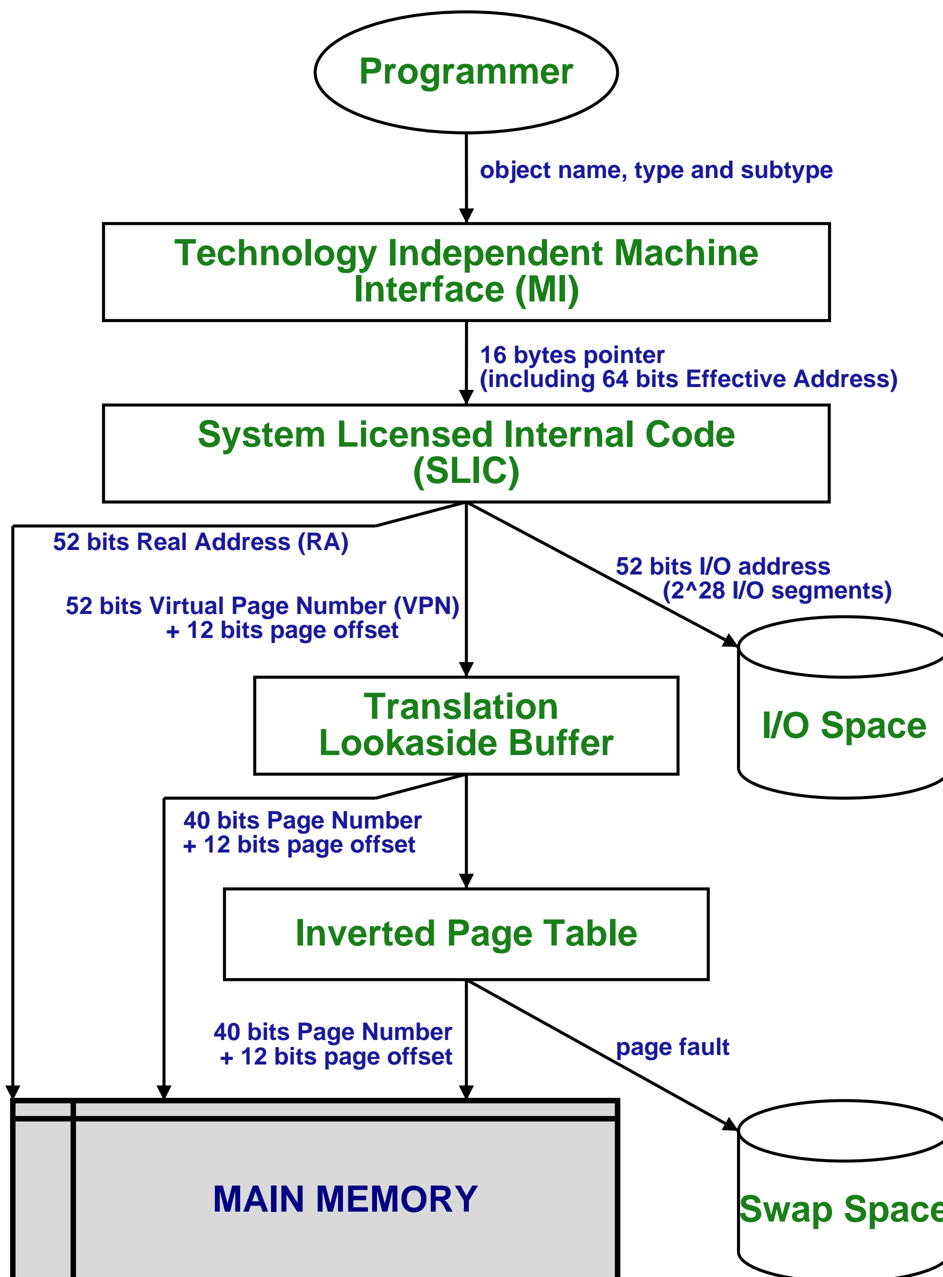
4 AS/400

4.1 Wstęp do zarządzania pamięcią w AS/400

Zarządzanie pamięcią w AS/400 jest konceptem wielopoziomowym. Z punktu widzenia programisty aplikacji, system operuje na obiektach. Nie jest przy tym wyróżnione w jakikolwiek sposób umiejscowienie tych obiektów - pamięć czy dysk. Każdy obiekt rozmieszczony jest zazwyczaj w dwóch lub więcej segmentach (jeden segment na przestrzeń funkcjonalną, pozostałe na przestrzeń danych). Segment składa się z maksymalnie 4096 stron po 4096 bajtów każda.

Przy odwołaniu kolejne warstwy systemu tłumaczą obiekt kolejno na 16 bajtowy wskaźnik systemowy zawierający m.in. 64 bitowy adres efektywny (Effective Address). Następnie albo na 52 bitowy rzeczywisty adres w pamięci (Real Address), albo 52 bitowy numer strony wirtualnej i 12 bitowe przesunięcie, albo na 52 bitowy adres przestrzeni I/O. Rozróżnienie zależne jest od 12 najważniejszych bitów (jeśli $800 to RA, jeśli $801 to I/O).

Jeżeli adresem wynikowym jest numer strony wirtualnej, to następuje odwołanie do odwróconej tablicy stron. W niej wyszukany zostaje właściwy adres strony lub zostaje zgłoszony błąd braku strony i strona zostaje sprowadzona z przestrzeni wymiany.



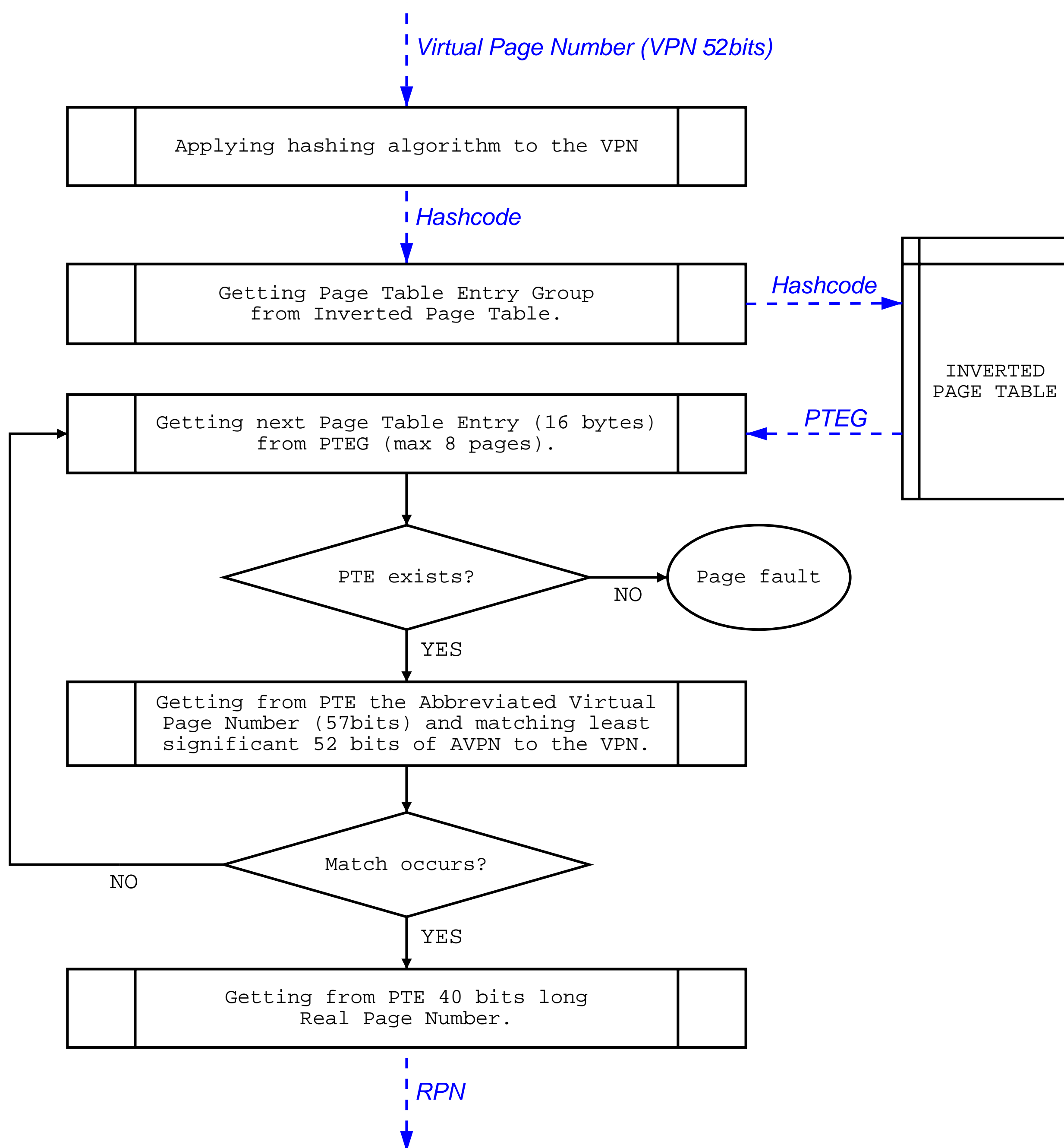
4.2 Pamięć Wirtualna

4.2.1 Algorytm znajdowania strony

Mając numer wirtualnej strony (VPN) system aplikuje do niego funkcję haszującą. Otrzymany w ten sposób klucz jest indeksem w odwróconej tablicy stron związanej z procesem (Inverted Page Table). Pod tym indeksem znajduje się tablica (Page Table Entry Group) zawierająca 16 bajtowe wpisy (Page Table Entry) dla ośmiu stron. Następnie system sprawdza, czy któraś ze znalezionych 8 stron jest tą szukaną.

Dokonuje się tego poprzez porównanie 52 najmniej znaczących bitów z 57 bitowego skróconego numeru strony wirtualnej (AVPN). AVPN to pierwsze 57 bitów PTE. Jeśli porównanie się powiodło i bit ważności PTE ustawiony jest na 1 to PTE zawiera 40 bitowy rzeczywisty numer strony (Real Page Number) będący lokalizacją szukanej strony w pamięci.

Jeśli żaden z 8 PTE nie jest poprawny, to najpierw sprawdzane jest, czy do procesu nie jest dołączona drugorzędna tablica stron (Secondary Page Table). Jeśli jest to powyższy algorytm jest dla niej powtarzany, a jeśli nie to zgłaszany jest błąd braku strony.



4.2.2 Obsługa błędu braku strony

W przypadku błędu braku strony, strona musi zostać pobrana z dysku. Zazwyczaj oznacza to, że któraś z istniejących stron musi zostać zwolniona. Algorytm stosowany do wyboru strony zwalnianej to odmiana LRU zwana Rozszerzony Algorytm Drugiej Szansy (Enhanced Second Chance Algorithm (Silberschatz and Galvin, p. 323)).

Algorytm ten opiera się na wykorzystaniu zawartych w PTE flag: Bit Odwołania (R) i Bit Zmiany (C). Bit odwołania ustawiany jest przy każdym odwołaniu się do strony, a bit zmiany przy każdej jej zmianie. Algorytm stara się znaleźć i usunąć stronę dla której R=0 i C=0, co oznacza, że strona ta nie była ostatnio ani zmieniana ani nawet nikt się do niej nie odwoływał.

Podczas tego wyszukiwania tworzona jest lista stron dla których R=0 ale C=1. Jak lista ta osiągnie pewną długość krytyczną inicjowany jest zapis tych stron na dysk i zmiana ich flag C na 0. Pomaga to zagwarantować, że w systemie znajdą się strony z R=0 i C=0.

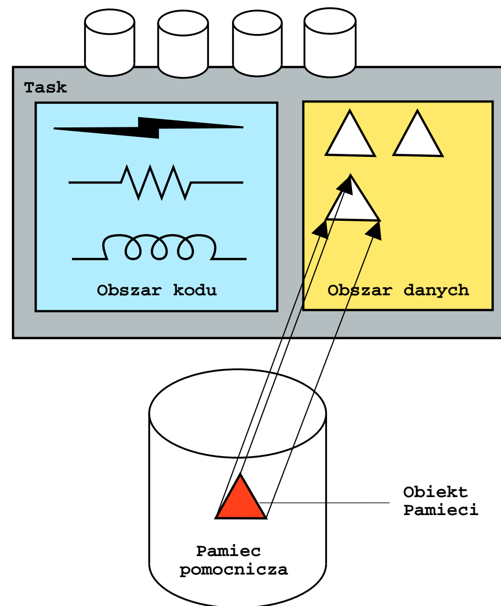
Po obsłużeniu błędu braku strony wszystkie bity odwołania (R) ustawiane są na 0.

5 System Mach

5.1 Wstęp do zarządzania pamięcią w Mach

Podstawową abstrakcją w systemie Mach jest obiekt pamięci (memory object). Może on reprezentować plik, pipe'a lub inne dane mapowane do pamięci wirtualnej. Obiekty pamięci mogą być zarządzane przez managery z poziomu użytkownika. Pozwala to na testowanie i stosowanie własnych algorytmamów zarządzania pamięcią, co nie jest możliwe w tradycyjnych systemach, gdzie pager jest częścią jądra.

5.2 Podstawowe komponenty w systemie Mach



task – środowisko wykonywania dla wątków. Jest podstawową jednostką allokującą zasoby. Zawiera stronicowaną przestrzeń wirtualną i chroniony dostęp do zasobów systemowych przez porty. Task może zawierać jeden lub więcej wątków.

wątek – podstawowa jednostka wykonująca. W Mach nie ma procesów. Proces można przyrównać w Mach do tasku z jednym wątkiem. Wszystkie wątki w tasku dzielą jego zasoby (pamięć, porty, ...)

port – kanał komunikacyjny implementowany jako kolejka komunikatów. Porty są chronione prawami – task musi mieć prawa danego portu, żeby móc do niego pisać.

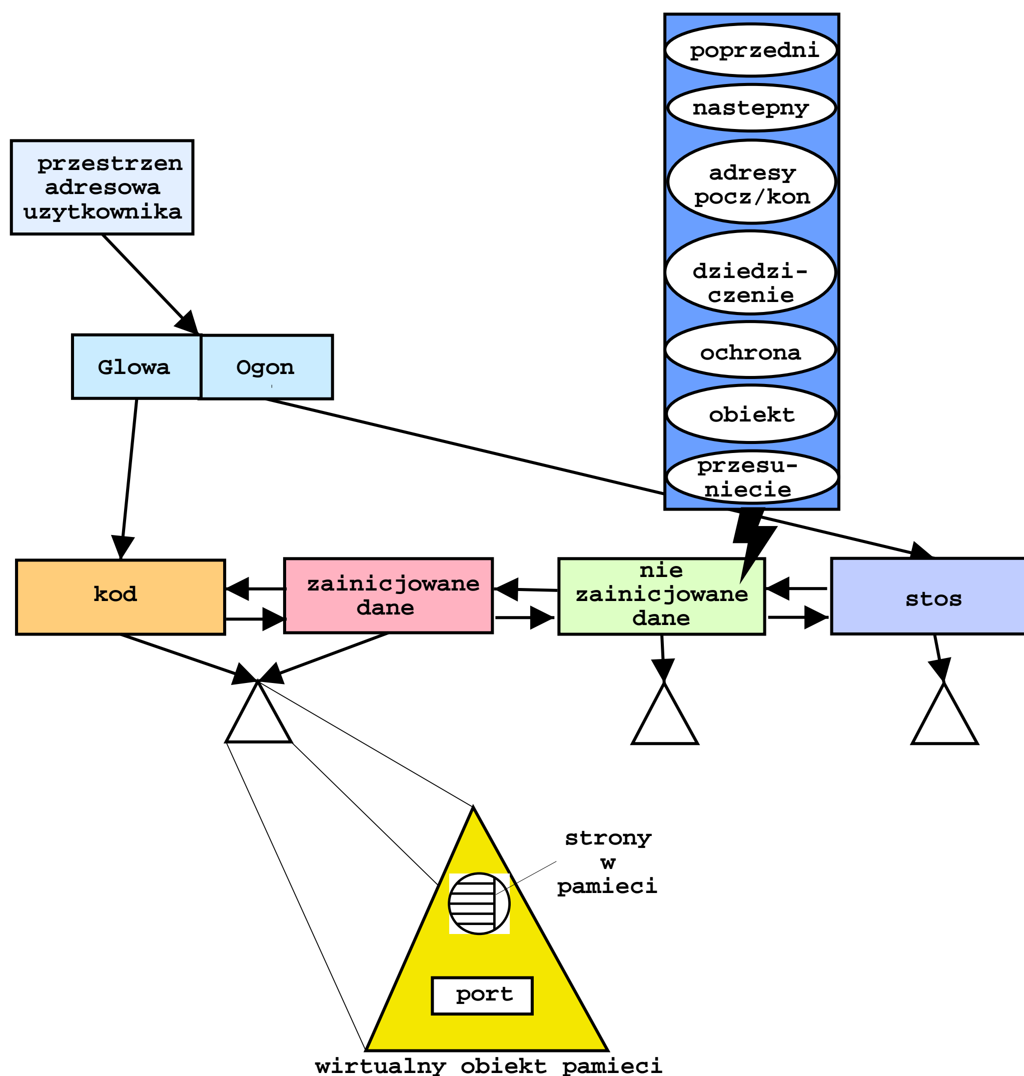
komunikat – utypowiony zbiór danych dowolnego rozmiaru używany do komunikacji między wątkami. W ten sposób taski mogą przekazywać innym taskom prawama dostępu do portów.

obiekt pamięci (memory object) – zbiór danych, który taski mogą mapować w swojej przestrzeni adresowej. Obiekt taki może np. reprezentować plik. Obiektami pamięci może zarządzać external memory manager z poziomu użytkownika.

5.3 Zarządzanie pamięcią w systemie Mach

Mach pozwala użytkownikowi na tworzenie obiektów pamięci, które są zarządzane przez external pager (task z poziomu użytkownika). Kiedy task chce uzyskać dostęp do zasobu tworzy nowy obiekt używając external pager i zleca jądru odwzorowanie go w swojej przestrzeni adresowej.

5.3.1 Wirtualna Przestrzeń Adresowa Tasku



Mapa adresów Taskowi przypisana jest podwójnie wiązana lista, której węzły opisują odwzorowanie z ciągłego obszaru adresów wirtualnych tasku w ciągły obszar obiektu pamięci. Przestrzeń adresowa nie jest kompresowana – może mieć wielkość powyżej 4 GB. Na podstawie mapy adresów tworzona jest tablica stron, która zawiera jedynie allokowane regiony. Wyjątkową cechą systemu jest to, że tablica stron nie musi przechowywać wszystkich odwzorowań adresów wirtualnych w fizyczne. Mogą one być odtworzone w każdej chwili na podstawie mapy adresów i innych struktur niezależnych od sprzętu. Takie rozwiązanie może zmniejszyć liczbę zajętych ramek i usprawnić działanie systemu. Mapowanie nowych adresów można odwlekać do momentu kiedy jest to niezbędne.

5.3.2 Zarządzanie stronami w pamięci fizycznej

- Pamięć fizyczna dla obiektu pamięci jest allokowana dopiero w momencie odwołania się do niego przez wątek.

- Pamięć fizyczna w Mach jest traktowana głównie jako cache dla obiektów pamięci.

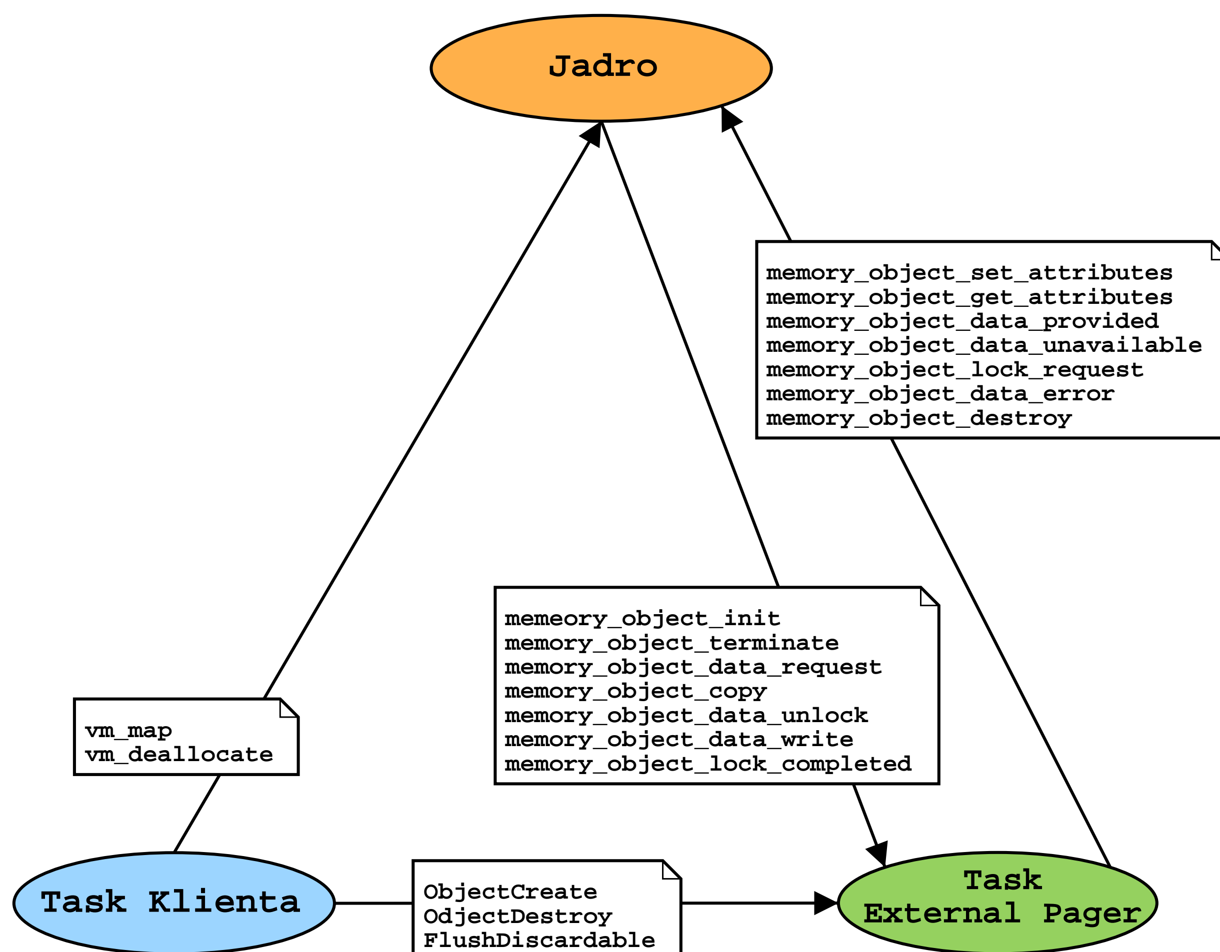
- Mach trzyma dane o stronach obecnych w pamięci fizycznej, dla wszystkich zamapowanych objektów w tablicy indeksowanej przez fizyczny numer strony. Jednocześnie każda pozycja w tablicy może być powiązana w kilka list ułatwiających różne operacje na pamięci:

  1. lista obiektu pamięci – łączy wszystkie strony danego obiektu, co usprawnia dealokację i wirtualne kopiowanie

  2. kolejka allokacyjna – osobna dla wolnych i zajętych stron; używana przez pageout daemon.

  3. tablica haszująca indeksowana przez obiekt/offset – umożliwia szybkie sprawdzenie fizycznej strony w czasie page-fault.

5.3.3 Interakcja jądra, tasku użytkownika, i external pagera



5.3.4 Mapowanie Obiektu

Task mapuje obiekt pamięci wywołując `vm_map`. Task może podać adres dla obiektu, albo pozostawić wybór jądru. Każdy wpis w mapie tasku zawiera informacje o atrybutach dziedziczenia i ochrony regionu pamięci, którego dotyczy. Atrybuty te są wspólne dla wszystkich stron w regionie. Jądro może allokować wiele wpisów dla jednego obiektu pamięci jeżeli atrybuty stron obiektu są różne.

5.3.5 Błąd braku strony

1. Wątek odwołuje się do strony obiektu pamięci, której nie ma w pamięci operacyjnej.

2. Występuje błąd braku strony i przejście do jądra. Wątek zostaje uśpiony.

3. Jądro wysyła komunikat `memory_object_data_request` do portu obiektu pamięci.

4. External pager związany z obiektem przekazuje stronę w `memory_object_data_provided`, albo zwraca bład – `memory_object_data_error`.

5.3.6 Wymiana stron

- Za wymianę stron odpowiedzialny jest wewnętrzny wątek jądra - pageout deamon.
- Strony do usunięcia z pamięci są wybierane na podstawie algorytmu FIFO drugiej szansy, który przybliża algorytm LRU.
- Strony odsyłane są do odpowiedniego pagera (domyślnego lub tego z poziomu użytkownika).

Domyślny pager Mach używa domyślnego pagera do obsługi własnych stron i stron, do których użytkownik nie ma przypisanego zarządcy. Jest on także używany kiedy external pager z poziomu użytkownika nie wykona żądanej wymiany stron. W systemie Mach 3.0 domyślny manager może korzystać z plików w standardowym systemie plików lub ze specjalnej partycji na dysku.

Stronnicowanie z poziomu urzytkownika External pager z poziomu użytkownika może mieć lepszą strategię wymiany stron dla danego obiektu niż paging daemon.

Oto opis jednego ze sposobów zarządzania wymianą stron z poziomu użytkownika:

- Task użytkownika i external pager mają wspólną tablicę atrybutów stron danego obiektu w pamięci dzielonej, która jest osobnym obiektem pamięci mapowanym przez oba taski.
- Task użytkownika zapisuje tam informację czy strona może zostać wyrzucona czy nie tzn. czy musi być zapisana w pamięci pomocniczej podczas page-out.
- External pager czyta te informacje i odpowiednio zarządza stronami obiektu. Może w ten sposób uniknąć zapisu na dysk nieistotnych stron. Może również poprawić domyślną strategię wymiany stron. Robi to na dwa sposoby:

  User initiated Task klienta ustawia atrybuty stron i wysyła FlushDiscardable do pagera. Pager wysyła `pager_flush_request` do jądra, żeby zwolnić strony.

  Pager initiated Pager wysyła `pager_flush_request` do jądra, kiedy dostaje od niego `memory_object_data_write`. Jądro wysyła ten komunikat kiedy zaczyna brakować pamięci. Jest to więc dobry moment by zwolnić niepotrzebne strony.

5.4 Zalety pamięci wirtualnej z external pagerem w systemie Mach

- Możliwość monitorowania obiektu pamięci i zwalniania pamięci fizycznej zanim zrobi to jądro. Jest to szczególnie użyteczne kiedy wykonujemy programy, które korzystają z pamięci w specyficzny, znany sposób.

- Możliwość kontrolowania dostępu do mapowanego pliku.
- Możliwość kontrolowania kolejności operacji wykonywanych w pamięci pomocniczej. Np. w systemach zarządzania bazą danych tranzakcje muszą być zapisane w pliku z logami zanim zmienią dane w bazie.
- Możliwości eksperymentowania z różnymi algorytmami pamięci bez ingerencji w jądro systemu.

A Bibliografia

1. A.Silbershatz P.B.Galvin G.Gagne: *Podstawy Systemów Operacyjnych*

2. A.Silbershatz P.B.Galvin G.Gagne: *Operating System Concepts sixth edition* – Appendix B: The Mach System

   http://www.bell-labs.com/topic/books/os-book/mach.pdf

3. R.Raschid, A.Tevanian, M.Young, D.Golub, R.Baron, D.Black, W.Bolosky, J.Chew: *Machine-Independent Virtual Memory Managment for Paged Uniprocessor and Multiprocessor Architectures*

   http://www-2.cs.cmu.edu/afs/cs/project/mach/public/www/doc/publications.html

4. I.Subramanian: *Managing Discardable Pages with an External Pager*

   http://www-2.cs.cmu.edu/afs/cs/project/mach/public/www/doc/publications.html

5. *Design Elements of the FreeBSD VM System*

   http://www.freebsd.org/doc/en_us.ISO8859-1/articles/vm-design/

6. *The FreeBSD VM System*

   http://www.kr.freebsd.org/handbook/handbook336.shtml

7. *AS/400 Memory Management*

   http://www.varietysoftworks.com/jbaldwin/Education/single-level_store.html

8. *Teraspace Storage*

   http://www-1.ibm.com/servers/eserver/iseries/whpapr/teraspace.html