Pamięć Wirtualna w Linuksie 2.4

Jarosław Bąbel

10 grudnia 2002

Spis treści

1 Segmentacja

W Linuksie używane są cztery segmenty (takie same 0 - 4GB): kod jądra, segment danych jądra, kod użytkownika, dane użytkownika. Linux nie bazuje na pojęciu segmentacji ze względu na trudności z tym związane (słabe wspomaganie ze strony sprzętu), ale pozwala na korzystanie z niej. Segmenty są zdefiniowane i opisane w odpowiedniej tablicy globalnej. Dodatkowo każdy proces ma swoją tablicę lokalną, ale z reguły taką jak globalna.

Rozważamy trzy rodzaje adresów:

1. logiczne - złożone z numeru segmentu i przesunięcia w segmencie
2. liniowe - w przestrzeni adresowej od 0 do 4GB (powstają z logicznych)
3. fizyczne (powstają z liniowych)

.

2 Odwzorowanie wirtualnej przestrzeni adresowej procesu

Wirtualna przestrzeń adresowa każdego procesu jest podzielona na dwie główne części:

1. 0 - 3GB - przestrzeń procesu
2. 3GB - 4GB - przestrzeń jądra

Przestrzeń jądra jest współdzielona przez wszystkie procesy. Dostęp do niej jest możliwy tylko w trybie jądra. Na początku tej części przestrzeni adresowej odwzorowana jest pamięć fizyczna.

Dla każdego procesu jądro utrzymuje strukturę opisującą przestrzeń adresową procesu. W tej strukturze znajduje się m.in. lista struktur opisujących ciągłe obszary pamięci (wirtualnej). Każdy ciągły obszar może mieć różne metody wykonywania zadań związanych z wirtualną pamięcią (głównie obsługa błędu braku strony), to pozwala jądru na poprawną obsługę wirtualnych regionów, które są używane w różny sposób. Np. dla obszaru związanego z kodem procesu obsługa błędu braku strony polega na wczytaniu odpowiedniej strony z pliku, dla obszaru przydzielonego przez malloc() obsługa błędu polega na ściągnięciu strony z pliku wymiany (lub jego pamięci podręcznej) lub przydzieleniu nowej strony.

Zawsze istnieją co najmniej trzy obszary:

1. kod i zainicjowane dane
2. niezainicjalizowane dane i sterta (bezpośrednio za poprzednim)
3. stos

Dodatkowo jest też obszar na biblioteki dzielone. Obszary są przechowywane w dwóch strukturach: lista i drzewo czerwono-czarne. Lista jest uporządkowana według adresów obszarów.

Drzewo pozwala na szybkie wyszukiwanie przynależnego adresowi obszaru co przyspiesza np. obsługę błędu braku strony.

Pamięć bezpośrednio za 3GB odwzorowuje w sposób ciągły pamięć fizyczną. Jeżeli pamięci fizycznej jest więcej niż 896MB, to pozostała część jest dostępna jako „HIGHMEM" (tylko na pamięć podręczną i pamięć procesów). Pozostałe 128MB jest wykorzystywane do przydzielania ciągłych obszarów wirtualnej pamięci dla jądra oraz odwzorowywania stron ze strefy „HIGHMEM".

3 Stronicowanie

Linux używa trzypoziomowego mechanizmu stronicowania nawet jeżeli w architekturze, na której pracuje brakuje odpowiedniego wspomagania. Jest to potrzebne do obsługi architektur pozwalających na adresowanie więcej niż 4GB pamięci np. 64 bitowych. Linux będzie używał stronicowania wyższego niż 1 rzędu nawet jeżeli sprzęt, na którym pracuje nie wspomaga stronicowania wcale. x86 obsługuje stronicowanie dwupoziomowe stąd środkowa tablica stron jest jednoelementowa. Przejście do innej architektury wymaga modyfikacji (uzupełnienia) kodu jądra.

Rozmiar strony (i ramki) na x86 wynosi 4kb. Pojedynczy katalog stron zajmuje jedną stronę (1024 adresy 32 bitowe). Adres liniowy składa się z dwóch dziesiątek bitów (indeks w głównej tablicy stron i indeks we wskazanej tablicy stron) i dodatkowych 12 bitów określających przesunięcie w ramce.

Wszystkie procesy współdzielą tablice stron jądra i mają swoje tablice stron dla przedziału 0...3GB-1 . Stąd tablice stron procesu są poprawne w trybie jądra co ułatwia przejście między trybem użytkownika, a trybem jądra (procesy mogłyby mieć zupełnie osobne odwzorowanie pamięci od pamięci jądra).

Ponieważ strony jądra zajmują uprzywilejowany segment, proces w trybie użytkownika nie może sięgnąć do adresów powyżej 3GB, te adresy są dostępne tylko w trybie jądra. Podczas pracy procesu odwołania do przestrzeni adresowej trybu użytkownika powodują, że jądro przydziela ramki i odwzorowuje je w przestrzeń adresową procesu dokonując odpowiednich zmian w tablicach stron procesu. Dla procesu ramka jest przydzielana logicznie (tzn. zmiany dokonywane są w strukturach, ale nie od razu przydziela się ramkę fizyczną), jądro dostaje ramkę fizyczną od razu.

Fizyczna strona przydzielona dla procesu posiada co najmniej dwa wirtualne odwzorowania: jedno w przestrzeni jądra (3GB+fizyczny adres strony) i jedno pod jakimś adresem mniejszym od 3GB w wirtualnej przestrzeni adresowej procesu. Takie strony mogą mieć jeszcze dodatkowe odwzorowania: np. kilka procesów uruchamiających ten sam program współdzieli kod programu przez odwzorowanie tych samych fizycznych stron we właściwych dla nich tablicach stron.

3.1 Najważniejsze informacje związane z ramką

1. Strona może należeć do pliku odwzorowanego do pamięci. Wtedy potrzebna jest informacja o pliku oraz przesunięciu.

2. Listy cykliczne wolnych stron i stron związanych z jednym plikiem.

3. Sąsiednie elementy w tablicach mieszających przyspieszających wyszukiwanie stron należących do plików.

4. Kolejka procesów oczekujących na zwolnienie strony (np. zakończenie operacji wejścia-wyjścia).

5. Licznik odniesień. Jest to liczba procesów korzystających z ramki. Pozwala stwierdzić, czy strona należy w ogóle do jakiegoś procesu. W przypadku stron dzielonych i mechanizmu kopiowania przy zapisie, licznik ten jest zmniejszany, gdy któryś proces dostaje nową kopię ramki w związku z zapisem.

6. Flagi. Binarne parametry strony.

   - blokada ramki
   - powodzenie operacji wejścia-wyjścia
   - bit odniesienia - zapalany podczas odwołania do strony, gaszony przy poszukiwaniu stron do usunięcia
   - aktualność ramki - zgodność z odpowiednikiem na dysku

3.2 Kopiowanie przy zapisie (ang. copy on write)

Jeżeli tworzona jest kopia działającego procesu w pamięci, to należące do niego strony nie są natychmiast kopiowane. Są wykorzystywane do czasu wykonania zmian przez któryś z procesów potomnych.

4 Alokator strefowy

Pamięć fizyczna jest odwzorowana w górny obszar pamięci (od 3GB 3GB+896MB). Ramki zajmowane przez kod jądra i jego statyczne struktury są zarezerwowane i nigdy nie będą przydzielane w innych celach. Pozostałe ramki są wykorzystywane jako wirtualna pamięć procesów, bufory, dodatkowa wirtualna pamięć jądra itd. w zależności od potrzeb.

Alokator strefowy zarządza pamięcią fizyczną.

4.1 Strefy

Różne zakresy fizycznych stron mogą mieć różne właściwości z punktu widzenia jądra. Np. „DMA”, obszar pamięci, który pozwala zapisywać i odczytywać dane bezpośrednio do pamięci RAM bez udziału procesora działa tylko dla fizycznych adresów mniejszych niż 16MB. Dalej, niektóre systemy mogą mieć więcej fizycznej pamięci niż może być zaadresowane między 3GB, a 4GB. Te ramki nie są bezpośrednio dostępne dla jądra i muszą być potraktowane w inny sposób

(mogą być przeznaczone na strony użytkowników i pamięć podręczną - jest to tzw. „HIGHMEM", strefa, w której znajduje się pamięć fizyczna powyżej 896MB). Alokator strefowy obsługuje takie różnice dzieląc pamięć na kilka stref i traktując każdą z nich jak osobną jednostkę do allokacji. Każde żądanie przydzielenia strony jest związane z listą stref, w których można ją allokować. Lista jest uporządkowana od najbardziej do najmniej preferowanej strefy. Np. przy żądaniu przydzielenia strony dla użytkownika trzeba sprawdzić miejsce w „zwykłej" strefie, później w „HIGHMEM" i jeżeli to się nie powiedzie to w strefie „DMA". Prośba o przydział strony „DMA" może być spełniona tylko w strefie „DMA" i tylko ona znajdzie się na liście możliwych stref.

Każda fizyczna ramka w strefie ma swoją strukturę z opisem jej stanu.

4.2 System bliźniaków

Kod jądra może zażyczyć sobie „ciągłego" bloku stron (dokładnie ramek) o zadanej długości (nie za długi 2^n gdzie n jest co najwyżej 9, czyli bloki mogą mieć maksymalnie 512 stron, np. 2MB na x86 w jądrze 2.4.18). Może też zwracać (zwalniać) ramki blokami. Z pomocą przychodzi algorytm bliźniaków.

Wewnątrz każdej strefy do obsługi przydziału stron używany jest system bliźniaków. Jest on bardzo szybki dzięki prostym operacjom „bitowym" i odpowiednio do tego skonstruowanym strukturom (rozmiary bloków to potęgi dwójki). Istotą tego systemu jest minimalizacja fragmentacji zewnętrznej i duża wydajność.

Najwięcej przydziałów i zwolnień stron jest wykonywanych dla wirtualnej przestrzeni adresowej procesu. Ze względu na to, że najczęściej allokacje te występują podczas obsługi błędu braku strony, większość przydziałów stron dotyczy pojedynczej strony.

Pewne części jądra, szczególnie sterowniki urządzeń, mają specjalne potrzeby w stosunku do pamięci fizycznej. Mogą na przykład potrzebować czterech ciągłych ramek pamięci obsługującej DMA. System bliźniaków pozwala na szybką obsługę takich przydziałów.

System bliźniaków traktuje fizyczną pamięć jak zbiory 2^n stron zaczynających się na granicach adresów podzielnych przez 2^n. System łączy sąsiadujące bloki w pojedyncze bloki wyższego rzędu. Każdy blok rozmiaru 2^n jest „kumplem" (ang. buddy) drugiej połowy bloku wyższego rzędu (2^{n+1}), który go zawiera. Alokator trzyma listy wolnych bloków dwustronowych (zaczynających się na granicy dwóch stron) , czterostronowych (zaczynających się na granicy czterech stron) itd. W celu przydzielenia bloku pewnego rzędu (n) sprawdzamy najpierw listę wolnych bloków danego rzędu i (w przypadku niepowodzenia) kolejno wyższych rzędów. Po znalezieniu wolnego bloku następuje przydział. Jeżeli znaleziono blok wyższego rzędu niż wymagany to dzieli się go na dwa bloki o rozmiarze 2^{rzad-1}, pierwsza część jest dodawana do listy wolnych bloków, a druga część jest przydzielana, chyba, że jeszcze jest za duża to postępujemy rekurencyjnie.

Kiedy zwalniamy pamięć sprawdzamy, czy zwracany blok ma wolnego „kumpla" (na liście wolnych bloków danego rzędu). Jeżeli tak to sklejamy te dwa bloki usuwając „kumpla" z listy i zwalniając rekurencyjnie dwa razy większy blok.

5 Obsługa błędów strony

Na różnych architekturach występują różnice w „niskopoziomowej” obsłudze błędu strony. Podstawowe kwestie, które trzeba rozstrzygnąć to: czy błąd spowodował użytkownik, czy jądro; sprawdzenie praw dostępu do strony; sprawy stosu.

Podczas błędu braku strony jądro patrzy, czy pożądanej strony nie ma w pamięci podręcznej stron. Jeżeli znajdzie to przenosi ją na listę stron aktywnych (jeśli jeszcze jej tam nie ma) i natychmiast obsługuje błąd.

Jeżeli jesteśmy w trakcie obsługi przerwania lub nie ma kontekstu użytkownika, to znaczy, że błąd spowodowało jądro. Następuje wtedy próba naprawienia odwołania i w przypadku niepowodzenia system pada (oops).

Jeżeli błąd jest zgłaszany „z procesu użytkownika” (mamy kontekst) to sprawdzamy poprawność adresu w tym kontekście. Jeżeli coś się nie zgadza to zabijamy proces.

Niezależnie od architektury po stwierdzeniu poprawności odwołania próbujemy obsłużyć błąd. Są cztery przypadki:

1. Strony nie ma w pamięci i nigdy nie było: najpierw patrzymy, czy strony nie da się dzielić z jakąś inną (jeżeli da się i wymagamy pisania, to strona jest kopiowana), jeżeli nie da się dzielić i chcemy pisać to przydzielana jest nowa, czysta ramka, jeżeli chcemy czytać to podstawiana jest ramka wypełniona zerami (ZERO_PAGE - jedna na cały system).

2. Strony nie ma w pamięci, ale była wcześniej wykorzystywana: ściągamy ją z pliku wymiany (lub jego pamięci podręcznej jeżeli tam jest).

3. Ramka jest w pamięci: chcemy na niej pisać i jeżeli mamy odpowiednie prawa, to zaznaczamy stronę jako „brudną” i „odmładzamy” ją, jeżeli chcemy czytać to odwzorowujemy stronę i „odmładzamy”.

4. Może się zdarzyć, że chcemy pisać na stronie tylko do odczytu. Jest to możliwe w przypadku kiedy tworzymy proces używając techniki kopiowania przy zapisie (ang. Copy-On-Write), jeżeli strona jest współdzielona. Jeżeli na pewno nie jest to błąd, to przydzielamy nową stronę i kopiujemy starą.

W między czasie może wystąpić inny błąd np. błąd szyny. Wtedy kończymy proces, bo i tak w kółko występowałby błąd braku strony.

6 Postarzanie stron (ang. page aging)

Na różnych architekturach bit „brudnej strony” jest ustawiany sprzętowo lub nie.

Każda ramka ma przypisany wiek - pewną liczbę większą lub równą zero. Podczas przeglądania pamięci (przeglądamy fizyczne ramki) w poszukiwaniu stron do usunięcia zwiększamy wiek strony jeżeli był do niej dostęp i zmniejszamy jeżeli nie było. Po osiągnięciu wieku zerowego ramka staje się kandydatem do usunięcia z pamięci fizycznej.

Jądro przechowuje listy stron: aktywnych i nieaktywnych. Struktury te pozwalają na przybliżone posortowanie stron według najdłuższego okresu nieużywania (Linux przybliża algorytm LRU).

Proces postarzania stron (zmniejszania ich wieku w związku z brakiem odwołań) odbywa się na liście stron aktywnych. Strony, które osiągną zerowy wiek są przenoszone na listy stron nieaktywnych.

Strony, które nie są jeszcze kandydatami do usunięcia znajdują się na liście stron aktywnych, w tablicach stron procesów lub jednocześnie w obu. Jeżeli podczas przeglądania pamięci znajdziemy ramkę odwzorowaną przez jakiś proces, ale nie używaną, to przenosimy ją na listę stron aktywnych, aby umożliwić zwolnienie miejsca.

Strony nieaktywne (o wieku równym zero) są podzielone na dwie listy: nieaktywne „brudne” (ang. inactive dirty) i nieaktywne „czyste” (ang. inactive clean). Te drugie mogą być w każdej chwili udostępnione jako wolna pamięć, ponieważ są zgodne ze swoim odpowiednikiem dyskowym (plik wymiany, plik z kodem itd.), czyli „czyste”, dodatkowo dane na nich zawarte są aktualne, więc nowe odwołanie do nich skutkuje w natychmiastowym odwzorowaniu. Nieaktywne strony „brudne” są wymiatane na dysk po zebraniu pewnej ich ilości (w celu usprawnienia operacji dyskowych - zapisywanie pojedynczych stron często przerywa strumieniowy odczyt drastycznie zmniejszając wydajność) i przenoszone na listę nieaktywnych „czystych”.

W trakcie postarzania wiek stron jest zmniejszany dwukrotnie. Zmniejsza to obciążenie procesora i pozwala na szybsze reagowanie na zmiany obciążenia systemu.

Postarzanie stron odbywa się ciągle „w tle”. Dzięki temu na bieżąco wiadomo, które strony są mało używane i w przypadku gwałtownego wzrostu obciążenia wiadomo co usunąć z pamięci.

7 Postępowanie przy dużym obciążeniu

Podczas gwałtownego zmniejszania się rezerw wolnej pamięci aktywność kontroli wieku stron wzrasta. Jądro w pierwszej kolejności próbuje „uwolnić” pamięć z buforów alokatora płytowego. Jeżeli nadal brakuje pamięci to następuje sprawdzenie stron z listy aktywnych i przesunięcie najmniej używanych na listę nieaktywnych. Następnie lista nieaktywnych stron jest przeglądana w poszukiwaniu stron, które można zwolnić (jednocześnie weryfikowana jest synchronizacja z dyskiem), najlepiej jeśli strona nie ma użytkownika. Jeżeli strona różni się od swojego odpowiednika dyskowego, ale nie jest bardzo potrzebna to zapisuje się ją na dysk. Jeżeli na listach stron większość stron ma użytkowników (tzn. jest odwzorowana w pamięci jakichś procesów i nie można ich bezpośrednio usunąć) to rozpoczyna się proces odbierania stron procesom.

Jako ostatnie źródło pamięci jądro próbuje zmniejszyć bufory związane z systemem plików. Jeżeli to nic nie da to unicestwiany jest jeden z aktywnych procesów (trwają prace nad „inteligentnym” wyborem procesu do usunięcia tzn. wybraniu procesu, od którego nie zależy praca systemu operacyjnego).

8 Pamięć podręczna stron i pliku wymiany (ang. page cache, swap cache)

Celem pamięci podręcznej jest utrzymywanie w pamięci jak największej ilości przydatnych danych, w ten sposób żeby błędy braku strony były obsługiwane szybko. Większość pamięci procesu użytkownika znajduje się w pamięci podręcznej stron lub pliku wymiany (ang. swap cache). Przeważnie strona użytkownika jest dodawana do pamięci podręcznej podczas pierwszego odwzorowania w przestrzeń adresową procesu i pozostaje tam do momentu odwołania do użycia przez inny proces lub przez jądro.

Pamięć podręczna jest „warstwą" systemu znajdującą się pomiędzy jądrem i kodem obsługującym dyskowe operacje wejścia wyjścia. Strony wymiatane z pamięci procesu nie są natychmiast zapisywane na dysk, ale dodawane do pamięci podręcznej.

Każda strona kodu (lub pliku odwzorowywanego do pamięci) jest związana z pamięcią podręczną i-węzłów (ang. inode cache), dzięki czemu plik dyskowy może posłużyć jako zewnętrzny magazyn. Strony anonimowe, nie związane z definicji z jakimś plikiem (np. przydzielane przez malloc()) magazynuje się w systemowym pliku wymiany (ang. swap file) i czasowo przechowuje w pamięci podręcznej pliku wymiany. Dopóki nie zostaną wycofane z odwzorowania pamięci procesu, w którym rozpoczęły swoje istnienie strony te nie są dodawane do pamięci podręcznej pliku wymiany i nie mają zarezerwowanej przestrzeni w pliku wymiany . W przeciwieństwie do stron odwzorowywanych z plików, które po utworzeniu znajdują się w pamięci podręcznej stron. Stąd widać potrzebę istnienia dwóch rodzajów pamięci podręcznych stron.

Listy opisane w rozdziale „Postarzanie stron..." są częścią pamięci podręcznej stron. Dodatkowo w jej skład wchodzą tablice mieszające i inne struktury związane z plikami odwzorowywanymi w pamięci.

Strony dodawane do pamięci podręcznej stron najpierw umieszczane są na liście stron nieaktywnych.

9 Przydział pamięci dla jądra

9.1 Alokator bootowania

Odpowiada za rezerwowanie stron dla jądra na statyczne dane i kod podczas bootowania.

9.2 Wykorzystanie możliwości fizycznej pamięci podręcznej - alokator płytowy (ang. slab allocator)

Większość żądań o przydział pamięci jądra dotyczy małych, często używanych (czyli też tworzonych i usuwanych) struktur danych. Standardowy alokator operuje tylko stronami pamięci i nie wykorzystuje do końca możliwości sprzętu. Jeżeli allokacja wymaga przeglądania wielu elementów list, czy drzew to zawartość sprzętowej pamięci podręcznej jest zamazywana i nie spełnia swojej roli.

Podstawowa idea alokatora płytowego polega na unikaniu allokowania i usuwania często używanych obiektów pewnych typów (czyli zmniejszenie wykorzystania systemu bliźniaków). Osiąga się to przez przechowywanie zbioru często używanych obiektów w formie „upakowanej” w strony (co dodatkowo oszczędza pamięć przez zmniejszenie fragmentacji wewnętrznej).

Alokator płytowy przechowuje dane uporządkowane w strukturę trzypoziomową:

1. pamięć podręczna (ang. cache)
2. płyta (ang. slab)
3. obiekt

. Może być wiele pamięci podręcznych (dla konkretnych obiektów i ogólnego przeznaczenia). Każda pamięć podręczna składa się z płyt, które są blokami pamięci złożonymi z jednej lub więcej stron pamięci. Każda płyta zawiera pewną liczbę obiektów przedzielonych „dziurami” w ten sposób żeby trafiały do różnych linii fizycznej pamięci podręcznej. Struktura opisująca płytę może znajdować się na płycie lub poza nią. Płyty dzielą się na: zapełnione, częściowo zapełnione i puste. Jeżeli są częściowo wypełnione płyty to używa się ich w celu zminimalizowania fragmentacji.

Obiekty mające to samo przesunięcie względem różnych płyt z dużym prawdopodobieństwem będą odwzorowane do jednego wiersza fizycznej pamięci podręcznej co jest niekorzystne. Alokator płytowy stara się zredukować to przez tzw. kolorowanie płyt. Każdej płycie nadawany jest kolor, który określa przesunięcie obiektów w ramach płyty. Zmniejsza to prawdopodobieństwo wzajemnego zamazywania się w sprzętowej pamięci podręcznej obiektów położonych na różnych płytach.

9.3 Ciągłe obszary pamięci wirtualnej dla jądra

Jądro może mieć przydzielaną pamięć w formie następujących po sobie stron. Odwzorowywane fizyczne ramki nie muszą następować po sobie. Przydział w takiej formie odbywa się w przestrzeni wirtualnej powyżej 3GB, w odległości 8MB od odwzorowania pierwszych 896MB pamięci fizycznej. Ten bufor jest utrzymywany na wypadek przekroczenia granicy pamięci fizycznej. Podobnie przydzielone obszary są rozdzielane „dziurami” o rozmiarze jednej strony.

9.4 Odwzorowanie dowolnej fizycznej strony w przestrzeń wirtualną jądra

Jądro może odwzorować dowolną ramkę fizyczną w swoją przestrzeń adresową. Używane jest do tego ostatnie 128MB pamięci wirtualnej jądra. Ta technika pozwala m. in. na odwzorowywanie stron ze strefy „HIGHMEM” (które inaczej nie mogą być wysłane do pliku wymiany).

10 NUMA

Niejednorodny dostęp do pamięci (ang. Non-Uniform Memory Access) jest to sposób organizacji pamięci, w którym czas dostępu do różnych obszarów pamięci może być różny (zależny od jej fizycznego położenia). Wiele procesorów i wiele szyn - mniejsza konkurencja o dostęp do pamięci, lepsze wykorzystanie maszyn wieloprocesorowych.

Jądro musi znać wzajemne odległości między procesorami a pamięcią. Optymalne wykorzystanie systemu osiąga się wtedy, gdy pamięć procesu znajduje się w tym samym węźle co procesor(y) na, którym(ch) jest wykonywany. Węzeł w najprostszym przypadku to urządzenia i procesory połączone jedną szyną. NUMA jest wspomagana przez jądra serii 2.5.

11 MMU-less

Jednostka zarządzająca pamięcią (ang. Memory Management Unit) jest odpowiedzialna za translację adresów wirtualnych na fizyczne. Pamięć wirtualna jest najczęściej implementowana przy użyciu MMU. Stąd Linuks na architekturach nie posiadających MMU nie posiada pamięci wirtualnej. Każdy proces musi być umieszczony w pamięci w miejscu, w którym można go uruchomić. Dodatkowo pamięć ta musi być ciągła i nie może się powiększać, brakuje też ochrony pamięci (te problemy oczywiście usuwa pamięć wirtualna). Rozwiązaniem jest przydzielanie pamięci ze wspólnej puli systemowej przez alokator jądra oraz zwracanie nie używanej pamięci do puli.

12 PAE - Physical Address Extension

Linuks umożliwia wykorzystanie pamięci fizycznej do 64GB nawet na procesorze i386. W tym celu używa się pełnego stronicowania trzypoziomowego i tablic stron z wpisami 64 bitowymi. Pojedynczy proces nadal może zaadresować tylko 4GB (ze względu na 32 bitową architekturę), ale fizyczna pamięć odwzorowywana w jego wirtualną przestrzeń może pochodzić ze strefy „HIGHMEM”. W ten sposób np. 10 procesów pracujących współbieżnie może używać 10GB fizycznej pamięci. PAE umożliwia też przechowywanie w strefie „HIGHMEM” pamięci podręcznej stron. Zamiast logicznych adresów do ramki odnosimy się przez wskaźnik do struktury ją opisującej.

13 Architektury obsługiwane przez Linuksa

Compaq Alpha AXP, Sun SPARC i UltraSPARC, Motorola 68000, PowerPc, PowerPC64, ARM, Hitachi SuperH, IBM S/390, MIPS, HP PA-RISC, Intel IA-64, DEC VAX, AMD x86-64, CRIS. Dodatkowo architektury bez MMU (z reguły małe przenośne urządzenia).

14 Materiały

- http://lse.sourceforge.net/numa - O architekturach NUMA.
- http://www.linuxdevices.com - Linux jako system wbudowany (również o MMU-less)
- http://www.uclinux.org - Linux na maszynach bez MMU
- http://home.earthlink.net/˜ jknapka/linux-mm/vmoutline.html - Opis pamięci wirtualnej w Linuksie (obecnie w trakcie tworzenia)
- http://www.linux-mm.org - Strona poświęcona zarządzaniu pamięci w Linuksie i nie tylko, dużo odniesień
- http://www.surriel.com/lectures/mm.html - „Za mało, za wolno”, slajdy do prezentacji o pamięci w ogóle i w Linuksie (2.2, 2.4, 2.5)
- http://www.surriel.com/lectures/linux24-vm.html - Artykuł przedstawiający wady i zalety systemów zarządzania pamięcią w Linuksach serii 2.2 i 2.4
- http://cne.gmu.edu/modules/vm - Pamięć w Linuksie i nie tylko, ciekawy interfejs
- http://www.csn.ul.ie/˜ mel/projects/vm/ - Dokumenty o pamięci wirutalnej w Linuksie podane na wykładzie
- http://linuxcompressed.sourceforge.net/vm24/ - Krótki przegląd działania wirtualnej pamięci w Linuksie 2.4
- http://www.symonds.net/˜ abhi - Strona autora podręcznika (podany na wykładzie, ponad 300 stron) o pamięci w Linuksie 2.4. Podręcznik zawiera szczegółowe opisy funkcji i struktur związanych z pamięcią wirtualną. Niestety jeszcze nie jest skończony.