Wpływ architektury procesora na system operacyjny

Notatki do prezentacji

Krzysztof Kąs (rozdziały 5-7)
Paweł Bedyński (rozdziały 1-4)
Krzysztof Niemkiewicz (rozdziały 8-15)

14 listopada 2008

Spis treści

1 Wprowadzenie **5**
1.1 Gdzie szukać informacji 5
1.2 Rozszerzenia, instrukcje specjalne dla różnych architektur . . . 5

2 RISC vs CISC **8**
2.1 Cechy podstawowe 8
2.2 Czym jest x86 9

3 Pipelining **10**

4 Memory Barriers
na podstawie linux documentation **13**
4.1 Wprowadzenie 13
4.2 Założenia 14
4.3 Rodzaje barier 16
4.4 LINUX KERNEL założenia 17
4.5 LINUX KERNEL explicit 18
4.6 LINUX KERNEL implicit 19

5 Architektury 32-bitowe i 64-bitowe **21**
5.1 Czy 64-bitowe adresowanie to dzisiaj konieczność? 21
5.1.1 Przydzielanie pamięci dla aplikacji w systemie Windows 22
5.1.2 PAE - Physical Address Extension 22

5.1.3 AWE - Address Windowing Extensions 23
5.2 Adresowanie w trybie 64-bitowym 24
5.2.1 Segmentacja w trybie 64-bitowym 26
5.2.2 Wielkość stron 26
5.3 Tryby pracy procesora 64-bitowego 27
5.4 64-bitowe systemy operacyjne - przykłady 30
5.4.1 64-bitowe edycje systemu Windows 30
5.4.2 Linux 31
5.4.3 Mac OS X 31

6 Pamięć podręczna procesora 33
6.1 Spójność pamięci cache w procesorach Intela i AMD 33
6.2 MTRR - memory type range registers 33
6.3 Kolorowanie 33
6.3.1 Przykład: przyspieszanie schedulera w Linuksie 34
6.4 TLB - translation lookaside buffer 35
6.4.1 Chybienie w TLB 36
6.4.2 Zasięg bufora TLB 36

7 Architektura NUMA 37
7.1 NUMA a SMP 37
7.2 ccNUMA - cache coherent NUMA 38

8 VLIW - Very Long Instruction Word 39
8.0.1 Bardzo długie instrukcje 39
8.0.2 Zalety i wady 39

9 Wieloprocesorowość, wielordzeniowość 40
9.1 Wieloprocesorowosc, wielordzeniowosc 40
9.1.1 Wieloprocesorowość 40
9.1.2 Wielordzeniowość 40
9.2 Dlaczego wiele rdzeni? 41
9.2.1 Problemy z jednordzeniowymi procesorami 41
9.2.2 Wiele rdzeni 41
9.2.3 Problemy z wieloma procesorami logicznymi 42

10 Sposoby rozdzielania zadań dla maszyn wieloprocesorowych 42
10.1 Architektura asynchroniczna (1970-80) 42
10.1.1 AMP,ASMP,Master - Slave 42
10.1.2 Współczesne przykłady 42
10.2 SMP,BMP 43

10.2.1 SMP - Symmetric MultiProcessing . . . 43
10.2.2 Cache . . . 43
10.3 UMA . . . 43
10.4 NUMA, ccNUMA . . . 44

11 Pamięci podręczne(cache) 44
11.1 Utrzymywanie spójności dostępów do pamięci . . . 45
11.1.1 Źródła kłopotów . . . 45
11.1.2 Modele spójności - podobnie jak na BD . . . 45
11.1.3 Implementacja . . . 45

12 Problemy z współbieżnością 45
12.1 Przykre wspomnienia z PW . . . 45
12.1.1 Filozofia . . . 45
12.1.2 „Normalna“ semantyka . . . 46
12.2 Rozwiązania w linuxie . . . 46
12.2.1 Znane nam rozwiązania w linuxie . . . 46
12.3 Rozwiązania w Windows . . . 46
12.3.1 Porady dla twórców sterowników dla Windows . . . 46

13 Scheduler 47
13.1 Scheduler w linuxie - stary i nowy . . . 47
13.1.1 Stary(jądra 2.4.x) . . . 47
13.1.2 Nowy(jądra 2.6.x) . . . 47
13.1.3 Praktyczne porównanie . . . 48
13.2 Load balancing . . . 48
13.2.1 Procedura wyrównywania obciążenia procesorów . . . 48

14 Aplikacje które skutecznie wykorzystują wiele procesorów logicznych 48
14.1 Ogólne wymagania . . . 48
14.1.1 Prawo Amdahl'a . . . 48
14.1.2 Inne ograniczenia . . . 49
14.2 Grupy aplikacji przystosowanych do wielowątkowości... . . . 49
14.2.1 Aplikacje biznesowe . . . 49
14.2.2 Naukowe . . . 49
14.2.3 Inne . . . 49
14.3 ... i inne typy aplikacji które też chcą skorzystać z nowych możliwości . . . 50
14.3.1 Jeszcze inne :)... . . . 50

15 Podział architektur ze względu na liczbę potoków **50**
15.1 SISD,SIMD,MISD,MIMD 50
15.1.1 Podział architektur ze względu na liczbę potoków . . . 51
15.1.2 Obecnie ten podział traci na znaczeniu 51
15.2 Systemy rozproszone 51
15.2.1 Systemy rozproszone 51

1 Wprowadzenie

1.1 Gdzie szukać informacji

- linux-console: cat /proc/cpuinfo
  Podstawowe informacje o procesorze: taktowanie, flagi, wielkość cache'a.

- linux-console: ls -R /sys/devices/system/cpu/
  W tym katalogu system trzyma informacje o procesorze (procesorach).

- linux-documentation: np. cpu-freq/user-guide.txt
  Na stronach dokumentacji linuxa możemy znaleźć wiele plików README czy howto, przykładowo w user-quide.txt do cpu-freq znajdują się informacje o możliwościach zmiany taktowania procesora (tylko wskazane architektury wspierają tą funkcjonalność).

- linux-documentation: np. hot-plug.txt
  Ciekawć funkcjonalnośćić CPU o której można znaleźć informacje w dokumentacji linuxa jest HOT PLUG CPU czyli "wymrażanie"i budzenie procesorów w trakcie działania systemu. W dokumencie omówiono m.in scenariusz przechwytywania procesów ”związanych” z wymrażanym procesorem przez inne procesory.

1.2 Rozszerzenia, instrukcje specjalne dla różnych architektur

Szczegółowe omówienie architektury komputerów w tym architektury procesorów omawiane było na przedmiocie AKiPN. W tym miejscu wspomnieć warto o wpływie rozszerzeń najpopularniejszej architektury (x86) na systemy operacyjne.

MMX Pierwszym zbiorem rozszerzeń które nasze pokolenie pamięta było MMX.

> *MultiMedia eXtensions lub Matrix Math eXtensions to zestaw 57 instrukcji SIMD dla procesorów Pentium i zgodnych. Rozkazy MMX mogą realizować działania logiczne i arytmetyczne na liczbach całkowitych. Pierwotnie wprowadzone w 1997 przez Intela dla procesorów Pentium MMX, aktualnie dostępne również*

na procesory innych producentów - wraz z rozwojem procesorów i dodawaniem nowych rozszerzeń (np. SSE) zbiór rozkazów MMX powiększał się. Instrukcje te są wykorzystywane przez procesory od Intel Pentium MMX i AMD K6 wzwyż.[wikipedia]

MMX wprowadził 8 rejestrów 64 bitowych (mm0-mm7) wraz z zestawem instrukcji do wykonywania operacji arytmetycznych na całych wektorach jednocześnie. Zastosowania MMX to m.in przyśpieszenie dekodowania obrazów JPEG i PNG, filmów MPEG, a także wyświetlania grafiki trójwymiarowej. Niestety w tym zbiorze nowych instrukcji nie znalazły się żadne istotne z punktu widzenia samego systemu operacyjnego (a nie graficznego interface-u) rozwiązania.

SSE Zdecydowanie większym krokiem na przód w kierunku sprzętowego wspierania systemów operacyjnych przez procesor jest rodzina rozszerzeń SSE. Obecnie po SSE SSE2 SSE3 SSSE3 najnowszą wersją jest SSE4. Każda paczka to zestaw nowych instrukcji, które choć służą głównie do wykonywania operacji na wektorach (128bitowych, 16 nowych rejestrów xmm0-xmm16), to niekiedy zdarzają się "perełki"mające istotny wpływ na jądro systemu operacyjnego.

- SSE
  Już w pierwszej paczce SSE wprowadzono operacje PREFETCH i przeciwne do niej MOVNTQ MOVNTPS. Operacje te służą do dawania podpowiedzi (hint) procesorowi aby wskazany fragment pamieci umiescił (PREFETCH) lub nie umieszczał (MOVTNQ z rejestrow MMX, MOVNTPS z rejestrów SSE) w pamieci cache. Dodatkowo PREFETCH wystepuje w wersjach PREFETCH0 - PREFETCH2 oraz PREFETCHNTA, którymi możemy wskazać jak blisko (chodzi o poziom cache'a) dany obszar pamieci ma być umieszczony, bądź tez że określone dane będą używane jeszcze tylko raz.
  Wszystkie te operacje służą do przyśpieszenia wykonywania instrukcji, gdyż czas odczytu/zapisu z pamieci operacyjnej jest istotnie różny w stosunku do pamieci cache procesora (różnice sa wyraźne również pomiedzy poziomami cache'a). Mądrze wykorzystując te instrukcje programista może istotnie zwiększyć szybkość wykonywania określonego kodu, jednak równie dobrze może łatwo uzyskać efekt odwrotny (zaśmiecając ograniczony pojemnością cache nieistotnymi danymi)

- SSE3
  W trzeciej paczce SSE oprócz kolejnych instrukcji wspomagających

operacje wektorowe wprowadzono sprzętowe wsparcie dla synchronizacji. Operacje MONITOR i WAIT służą odpowiednio do zakładania "obserwatora" na wskazany obszar pamięci i "zawieszania się" na danym obszarze. Należy zauważyć iż każda architektura sama definiuje rozmiar tego obszaru (po obu stronach dodatkowo dokładane są 'pasy bezpieczeństwa'). Proces może wywołać instrukcje MONITOR na określonym fragmencie pamięci, a następnie zasnąć (WAIT) na nim. Procesor obudzi go gdy ktoś inny będzie chciał wykonać operacje zapisu/odczytu z obserwowanego obszaru. Oczywiście odgórne ustalanie wielkości rozmiaru pamięci używanej przez instrukcje MONITOR mogłoby prowadzić do niechcianego budzenia procesów, bądź też sytuacji odwrotnej. Instrukcje MONITOR i MWAIT są używane przez jądro linuxa
wiecej informacji tutaj

Przyszłość W latach 2009-2010 AMD wypuści na rynek procesory wspierające zestaw instrukcji SSE5 natomiast jednostki Intel'a będą wspierały AVX. Obydwa zestawy instrukcji coraz bardziej zbiżają tą rodzinę CPU do założeń architektury CISC, choć równocześnie obydwa mają w swym zestawie rozkazy (RISCOWE) postaci

$$w = y \pm (\pm x \cdot z)$$

Jedna z ciekawych nowości (oprócz planowanych już rejestrów 1024 bitowych) bedzie wprowadzenie sprzętowego wsparcia dla szyfrowania AES informacje tutaj

2 RISC vs CISC

2.1 Cechy podstawowe

Szczegółowe omówienie specyfikacji architektur RISC i CISC było tematem całego wykładu z AKiPN, dlatego tutaj przypomnimy tylko najważniejsze informacje.

CISC

- ZALETY:
  - ograniczona liczba instrukcji
  - duza liczba rejestrów - uniwersalne
  - redukcja trybów adresowania
  - wszystkie rozkazy wykonują się w jedym cyklu maszynowym (pipelining)
- WADY:
  - dużo trybów odwołań do pamięci
  - skomplikowane instrukcje
  - duża liczba odwołań do pamięci (koszt czasu)

RISC

- ZALETY:
  - duża liczba instrukcji 100 - ....
  - mała liczba rejestrów - specjalizowane
  - duża liczba trybów adresowania
  - złożone instrukcje wymagające kilku-kilkunastu cykli zegara
- WADY:
  brak (wg p. Peczarskiego :))

2.2 Czym jest x86

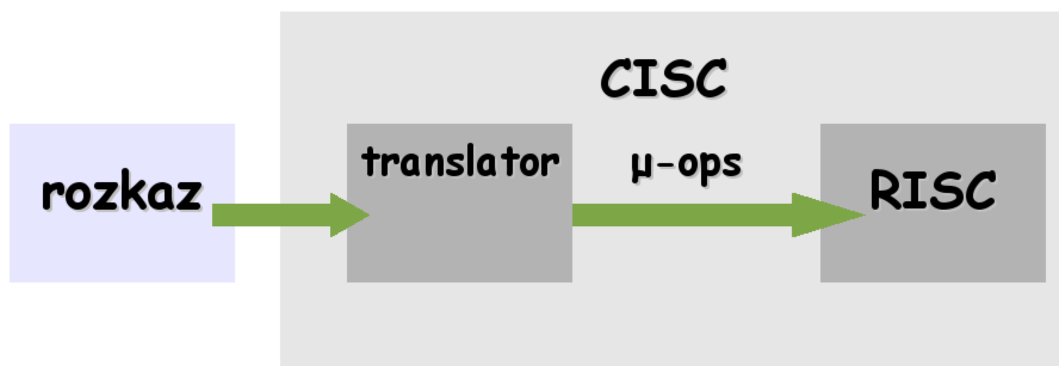
Osobnym tematem jest sklasyfikowanie najpopularniejszej obecnie architektury x86 do rodziny RISC albo CISC. Okazuje sie, że choć z początku procesory tej rodziny były typowymi CISCami, to z biegiem czasu coraz bardziej zaczynały przypominać RISCi. I tak możemy wyróżnić cechy x86 należące do obydwu - wydawać by się mogło - rozłącznych grup:
RISC

- krótki czas wykonywania rozkazu
- duża częstotliwość zegara sugeruje że operacje nie moga być óbszerne"

CISC

- duża liczba instrukcji (bardzo duża)
- skomplikowane instrukcje ... i tak sa rozbijane na mniejsze

Tak na prawdę x86 to z zewnątrz typowa architektura CISC, która wewnątrz przy użyciu translatora przekształca rozkazy na microoperacje wykonywane następnie przez moduł typowo RISCowy.



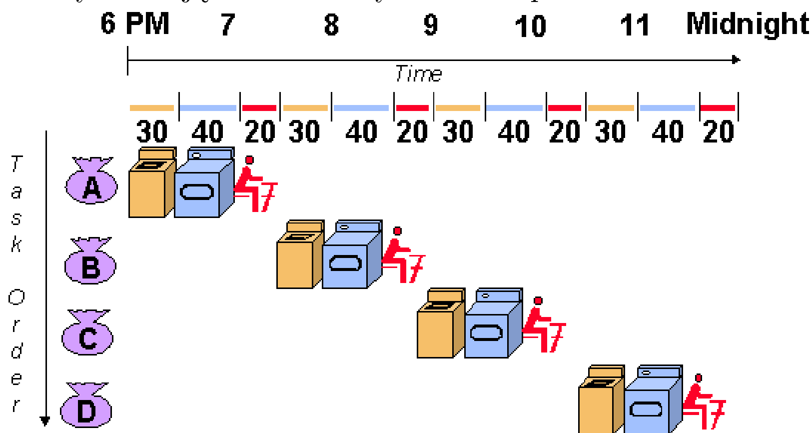
Tego typu architektura została nazwana PostRISC jednak wielu ludzi wciaż kłóci się o to, czym właściwie jest x86 odpowiednie forum tu (pl)

3 Pipelining

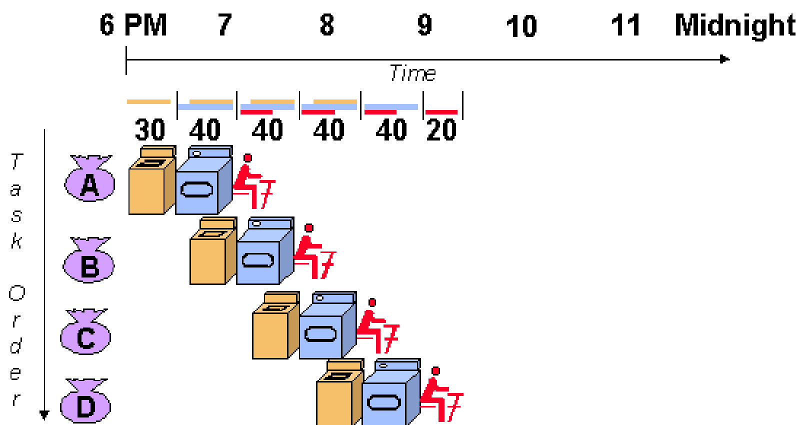
Wykonywnie instrukcji przez procesor można podzielić na następujące etapy (ich liczba może się różnić ze względu na architekturę, ale schemat generalnie przypomina poniższy podział):

1. pobranie rozkazu z pamięci operacyjnej
2. dekodowanie rozkazu, pobieranie danych z rejestrow
3. wykonanie instrukcji, lub obliczenie adresu
4. dostep do rejestru
5. zapis wyniku

Wyobrażając sobie te czynności na podstawie schematu działania pralni



możemy dojść do wniosku, że wykonywanie czynnosci (pranie suszenie składanie) sekwencyjnie jest wysoce nieefektywne. O wiele lepszy rezultat, a przy tym stopień wykorzystania maszyn/ludzi uzyskamy potokując zadania tzn. wykonując jednocześnie różne etapy różnych zadań



Współczesny procesor działa podobnie, wykonując w tym samym czasie kilka różnych instrukcji. Niestety jednak wykorzystując pipelining nie osiągamy przyśpieszenia gdzie nowy czas to 1/n starego (n to liczba potoków). Dzieje sie tak choćby z takiej przyczyny, że nie wszystkie zadania wykonują sie w takim samym czasie (choć w RISC wszystkie mają się wykonywać w jednym cyklu zegara). W takim wypadku najlepszym rezultatem byłby nadłuższy czas wykonywania jednego potoku, w przykładzie z pralnią - suszenie. Jednak i to nie jest możliwe. Zwróćmy uwagę na nastepujące przykłady

1. - add r3, r2, r1
   - add r5, r4, r3

   (add dodaje argument 2 do argumentu 3 i wynik umieszcza w argumencie 1, pozostałe operacje niezależne)

2. - loop:
   - add r3, r3 ,r1
   - sub r6, r5, r4
   - beq r3, r6, loop

   (beq sprawdza czy r3==r6 jesli tak to skok do loop)

W pierwszym przykładzie ze względu na 'zachodzące dane' procesor nie może dokonać spotokowania tych instrukcji jedna po drugiej, gdyż pierwsza instrukcja musi być 'w całości' wykonana aby jej wynik stał sie argumentem dla drugiej instrukcji. Drugi przykład nastręcza nawet więcej problemów, ponieważ w przypadku każdej pętli procesor nie wie jaki będzie rezultat sprawdzenia warunku stopu, a tym samym nie wie jaką instrukcje ma spotokować. (zauważmy ze musiałby miec taką informacje na kilka instrukcji do tyłu i w dodatku ją pamiętać).

Istnieją jednak mechanizmy dzięki którym procesor może poradzić sobie z wiekszością takich problemów.
Po pierwsze CPU może zmienić kolejność wykonywania instrukcji. W przykładzie 1 gdyby pomiędzy dwie instrukcje dodawania 'weszło' kilka instrukcji niezwiązanych z użytymi rejestrami. Procesor po wykonaniu pierwszego 'add' zająłby sie tymi nowymi rozkazami, a tym samym zanim rozpocząłby wykonywanie drugiej instrukcji 'add', rejestr r3 miałby już odpowednią wartość.
Po drugie architekci procesorów mogą zakładać 'dobrą wolę' programistów, którzy raczej nie piszą pętli po to, aby wykonała sie ona raz czy dwa. Sensowne jest zatem przypuszczenie że po ostatniej instrukcji z pętli wykona

sie pierwsza instrukcja z tej samej pętli. Dlatego gdy procesor będzie dochodził do końca pętli może 'puścić' w potok instrukcje z początku pętli, a w przypadku pomyłki (wyjścia z pętli) najwyżej troche instrukcji wykona się szeregowo.
Inne metody przewidywania skoku polegają na dynamicznym zapamiętywaniu przez procesor historii wyborów w każdym rozgałęzieniu (pętli). Współczesne algorytmy (algorytm Yeh'a) potrafią poprawnie przewidzieć skok w 90%

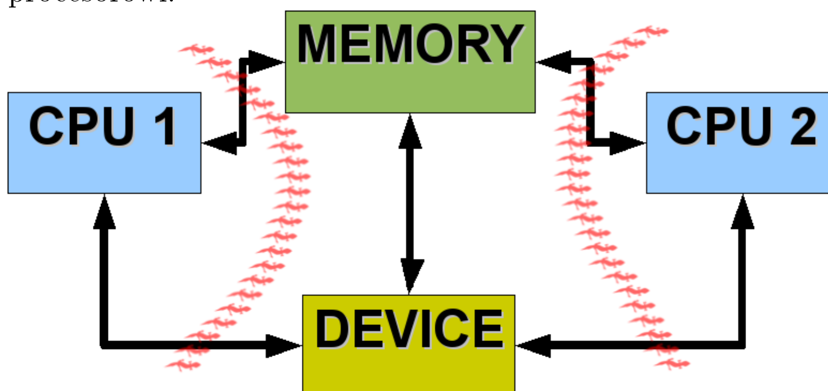
Superskalarność polega na współbieżnym przetwarzaniu rozkazów w równoległych potokach. Rozwiązanie to możliwe jest poprzez zduplikowanie (fizyczne) niektórych części procesora. W ten sposób możemy rozdzielać instrukcje na wiele pasm potoków, bądź też stworzyć specjalizowane układy (np tylko do obliczeń zmiennoprzecinkowych)

4 Memory Barriers

na podstawie linux documentation

4.1 Wprowadzenie

Rozpatrzmy model systemu gdzie każdy procesor generuje operacje odczytu zapisu do wspólnej pamięci. Dodatkowo procesor moze przeorganizować kolejność wykonywania instrukcji, jeśli tylko nie wpływa to na wynik wynkonania programu. Podobnie kompilator może zmienić porządek instrukcji dawanych procesorowi.



A zatem operacje na pamięci generowane przez procesory są widziane z perspektywy reszty systemu jako rozkazy przechodzące przez zaznaczone linie.

Rozpatrzmy nastepującą sekwencje zdarzeń:

```
CPU 1           CPU 2
=============== ===============
{ A == 1; B == 2 }
A = 3;          x = A;
B = 4;          y = B;
```

(oznacza stan zastany)

Zbiór możliwych kombinacji operacji LOAD STORE na wpółdzielonej pamięci wygląda następująco:

```
STORE A=3,      STORE B=4,      x=LOAD A->3,    y=LOAD B->4
STORE A=3,      STORE B=4,      y=LOAD B->4,    x=LOAD A->3
STORE A=3,      x=LOAD A->3,    STORE B=4,      y=LOAD B->4
STORE A=3,      x=LOAD A->3,    y=LOAD B->2,    STORE B=4
STORE A=3,      y=LOAD B->2,    STORE B=4,      x=LOAD A->3
STORE A=3,      y=LOAD B->2,    x=LOAD A->3,    STORE B=4
STORE B=4,      STORE A=3,      x=LOAD A->3,    y=LOAD B->4
STORE B=4, ...
...
```

A to może prowadzić do następujących wyników:

```
x == 1, y == 2
x == 1, y == 4
x == 3, y == 2
x == 3, y == 4
```

Rozpatrzmy następny przykład

```
CPU 1           CPU 2
=============== ===============
{ A == 1, B == 2, C = 3, P == &A, Q == &C }
B = 4;          Q = P;
P = &B          D = *Q;
```

Występuje tu oczywista zależność danych (data dependency), ponieważ wartość ładowana do D zależy od adresu pobranego z P przez CPU2. Na koniec wykonania tego zbioru instrukcji możemy mieć następujące wyniki:

```
(Q == &A) and (D == 1)
(Q == &B) and (D == 2)
(Q == &B) and (D == 4)
```

Zaznaczmy że CPU2 nigdy nie załaduje C do D bo CPU1 zaladuje P do Q zanim zacznie ładowanie *Q

4.2 Założenia

Należy rozróżnić następujące formy zależności danych:

1. brak zależności
2. overlaping - instrukcje odnoszą sie do tego samego fragmentu pamięci
3. data dependency - np gdy jedna instrukcja pobiera adres, pod który "sięga" druga instrukcja

Istnieją pewne minimalne założenia, których możemy "wymagać" od procesora

- Z perspektywy pojedynczego procesora jego instrukcje "zależne" wykonują się szeregowo

  ```
  Q = P; D = *Q;
  Procesor wykona nastepujące instrukcje
  Q = LOAD P, D = LOAD *Q
  ```

- 'zachodzace' instrukcje zapisu/odczytu ("wewnątrz" jednego procesora) będą szeregowane

```
 a = *X; *X = b;
CPU wykona: a = LOAD *X, STORE *X = b
a dla:
 *X = c; d = *X;
CPU wykona:  STORE *X = c, d = LOAD *X
```

Co musimy a czego nie możemy zakładać:

- Nie mozemy zakładac, że niezależne odczyty/zapisy wykonują się szeregowo tzn. dla X = *A; Y = *B; *D = Z; możemy mieć:

```
X = LOAD *A,  Y = LOAD *B,  STORE *D = Z
X = LOAD *A,  STORE *D = Z, Y = LOAD *B
Y = LOAD *B,  X = LOAD *A,  STORE *D = Z
Y = LOAD *B,  STORE *D = Z, X = LOAD *A
STORE *D = Z, X = LOAD *A,  Y = LOAD *B
STORE *D = Z, Y = LOAD *B,  X = LOAD *A
```

- Musimy zakładać, ze zależne (overlapping) zapisy/odczyty mogż być mergowane lub pomijane
  tzn. X = *A; Y = *(A + 4); możemy mieć:

```
      X = LOAD *A; Y = LOAD *(A + 4);
      Y = LOAD *(A + 4); X = LOAD *A;
      {X, Y} = LOAD {*A, *(A + 4) };
{} - oznacza jednoczesny dostęp do pamięci
```

  a dla *A = X; *A = Y; możemy mieć:

```
STORE *A = X; Y = LOAD *A;
STORE *A = Y = X;
```

Niezależne operacje na pamięci mogą być wykonywane efektywnie w losowej kolejności ale może to stwarzac problemy (np CPU-CPU lub I/O). Czasem jednak wymagane jest by kompilator i CPU były poinformowane o pewnych restrykcjach. Memory barriers nakładaja częściowy porządek na operacjach po dwóch stronach bariery. Powoduja że sekwencja zdarzeń na pamięci staje się dla innych cześci systemu imitacja sytuacji, w której to CPU wykonuje te operacje w takiej, a nie innej kolejności

4.3 Rodzaje barier

Rodzaje barier:

1. WMB - write (store) memory barrier
   Bariera na zapis daje gwarancje, że wszystkie operacje zapisu (STORE) przed blokadą zostaną faktycznie wykonane przed nią, natomiast wszystkie operacje po blokadzie zostaną wykonane po niej. Specyfikacja nie mówi nic o wpływie WMB na operacje odczytu (LOAD). Bariera WMB powinna być zwykle sparowana z RMB albo data-dependency barrier (chodzi o tzw. S(ymetric)M(ulti)P(processing) barrier pairing)

2. Data dependency barrier
   jest słabsza od pozostałych. Stosuje sie ją w przypadku wystąpienia zależności między danymi (tj np gdy pierwszy LOAD pobiera adres, pod który do sięga drugi LOAD), mamy wtedy pewność poprawnej kolejności wykonania tych instrukcji. Blokada ta nie specyfikuje żadnego wpływu kolejność operacji z innym typem zależności.

3. RBM - read (load) memory barrier
   Symetryczna do WMB. Ponadto implikuje data-dependency barrier

4. General memory barrier
   jednoczesna blokada WMB i RMB. nakłada cześciowy porządek zarówno na operacjach LOAD jak i STORE

a także pośrednie wywołania barier (definiowane przez specyfikacje, a nie konkretna implementacje!)

- operacja LOCK
  Wszystkie operacje za LOCK'iem będą za LOCK'iem, natomiast wszystkie operacje przed LOCKiem MOGĄ przeskoczyć ZA LOCK'a. LOCK jest prawie zawsze sparowany z UNLOCK'iem

- operacja UNLOCK
  Symetryczna do LOCK. Wszystkie operacje przed UNLOCK'iem bedą przed UNLOCK'a, natomiast wszystkie operacje za UNLOCK'iem MOGĄ przeskoczyć przed UNLOCK'a.

Memory barriers są wymagane tylko jeśli ma nastąpić interakcja między dwoma procesorami bądź procesorem a urządzeniem. Jeśli możemy zagwarantować, że takiej interakcji nie będzie, wtedy nie ma sensu stosować barier.
Powyższe założenia są "gwarantowanym minimum". Różne architektury moga specyfikować większe gwarancje, jednak w żadnym wypadku nie należy tych założeń traktować jako powszechnie obowiązujące.

4.4 LINUX KERNEL założenia

Czego nie możemy założyć o LINUX KERNEL MEMORY BARRIERS:

1. operacja dostępu do pamięci wykonana przed założeniem bariery będzie ukończona w chwili ukończenia operacji bariery (bariera może być postrzegana jako linia której niektóre instrukcje nie mogą przekraczać)
2. bariera na jednym CPU ma bezpośredni wpływ na drugi CPU lub inne urządzenia (co z pośrednim?)
3. jeden CPU będzie widział w prawidłowej kolejności efekty dostępu do pamięci innego CPU nawet jeśli ten drugi używa MEMORY BARRIER (odsyłam do "SMP BARRIER PAIRING")
4. inny hardware nie zmieni kolejności wykonywania instrukcji dostępu do pamięci

Anomalie

```
CPU 1           CPU 2
=============== ===============
{ A == 1, B == 2, C = 3, P == &A, Q == &C }
B = 4;
<write barrier>
P = &B
Q = P;
D = *Q;
```

W powyższym przykładzie w oczywisty sposob wystepuje data dependency i wydaje się że na końcu Q musi być albo &A lub &B oraz że:

```
(Q == &A) implies (D == 1)
(Q == &B) implies (D == 4)
```

Jednak wiedza CPU2 o P może być 'updateowana' PRZED wiedzą o B, co prowadzi do następującego przeplotu:

```
(Q == &B) and (D == 2) !?
```

Pomimo tego że może to sprawiać wrażenie błędu spójności, taka sytuacja faktycznie możę się zdarzyć i istnieją procesory, w których taki przeplot jest możliwy (DEC Alpha)
By rozwiązać ten problem musimy wprowadzić na CPU2 data-dependency barrier pomiędzy ładowanie adresu, a załadowanie danych.

```
CPU 1           CPU 2
=============== ===============
{ A == 1, B == 2, C = 3, P == &A, Q == &C }
B = 4;
<write barrier>
P = &B
                  Q = P;
                  <data dependency barrier>
                 D = *Q;
```

To wymusza wystąpienie jednej z dwóch pierwszych możliwości, blokując nieporządany przeplot.

4.5 LINUX KERNEL explicit

barrier()

- prosta blokada powodujaca że żadna operacja dostępu nie może przedostać się z jednej strony na drugą
- ta blokada jest osobna względem procesora więc....
- procesor może poźniej przeorganizowac kolejnosc wykonywania instrukcji ?! ...
- postawienie tej blokady na niektórych architekturach może wystarczyć, jednak nie możemy zakładać tego o wszystkich architekturach.

CPU memory barriers

```
TYPE            MANDATORY               SMP CONDITIONAL
=============== ======================= ===========================
GENERAL         mb()                    smp_mb()
WRITE           wmb()                   smp_wmb()
READ            rmb()                   smp_rmb()
DATA DEPENDENCY read_barrier_depends()  smp_read_barrier_depends()
```

Linux kernel ma osiem podstawowych funkcji - barier CPU
Bariery SPM na układach jednoprocesorowych są redukowane do bariery kompilatora ponieważ zakłada się, że CPU będzie wewnętrznie spójny i instruckje zachodzące (overlapping) będą wykonane w poprawnej kolejności.
Ponadto istnieją troche bardziej zaawansowane

- $set_mb(var, value)$
- $set_wbm(var, value)$

Przypisują one wartość do zmiennej a potem ustawiają co najmniej write-barrier (drugi przypadek).

```
smp_mb__before_atomic_dec();
smp_mb__after_atomic_dec();
smp_mb__before_atomic_inc();
smp_mb__after_atomic_inc();
```

Te funkcje służa do wykonywania atomowych opeacji inkrementacji i dekrementacji.
Przykład:

```
obj->dead = 1;
smp_mb__before_atomic_dec();
atomic_dec(&obj->ref_count);
```

Nałożenie tej bariery spowoduje że flaga "dead" zostanie ustawiona zanim zmieni się licznik.
wiecej informacji w:

Documentation\atomic_ops.txt

4.6 LINUX KERNEL implicit

Niektóre z funkcji jądra linuxa pośrednio wywołują memory barriers. Są nimi m.in. funkcje LOCK a takżę scheduler, funkcje alokujące pamięc.
schedule(), i podobne, zwalnianie i alokowanie pamięci nakładają Full Memory Barrier
dalej zajmiemy się funkcjami LOCK/UNCLOCK

Jądro Linuxa ma kilka konstrukcji typu LOCK/UNLOCK

- spin locks
- R/W spin locks
- mutexy
- semafory
- R/W sepathores

We wszystkich tych przypadkach występują warianty operacji "LOCK" i "UNLOCK", a te z kolei nakładają określone memory barriers:

1. implikacje operacji LOCK
   Wszystkie operacje po LOCK wykonają się faktycznie po LOCK, te przed mogą przeskoczyć za LOCK

2. implikacje operacji UNLOCK
   Wszystkie operacje przed UNCLOCK wykonają się faktycznie przed UNLOCK, te po mogą przeskoczyć przed UNLOCK

3. implikacje LOCK(1) ; LOCK(2)
   Kolejność 2 operacji LOCK nie zostanie zamieniona

4. implikajce LOCK ; UNLOCK
   Kolejność opreacje LOCK UNLOCK nie zostanie zmieniona

5. implikacje LOCK'a ktory sie nie powiodl
   Niektóre warianty operacji LOCK mogą się nie powieść. Może się to zdarzyć np. datego, że nie jest możliwe uzyskanie LOCK'a natychmiast, albo np. jakiś nieblokowany sygnał przyszedł podczas czekania na LOCK'a. Operacje LOCK które się nie udały nie nakładają żadnych memory barrier.

Dlatego z (1),(2) i (4) wynika że UNLOCK z następującym po nim LOCK są równoważne full barrier natomiast LOCK z następującym po nim UNLOCK już nie
Następstwem tego ze LOCK i UNLOCK mogą być barierami "w jedna strone" jest to, że efekty instrukcji spoza sekcji krytycznej mogą "przesiąknąć" do wnętrza sekcji krytycznej

Funkcje LOCK przykład

```
*A = a; *B = b;
LOCK
*C = c; *D = d;
UNLOCK
*E = e; *F = f;
```

Następująca kolejność zdarzeń jest możliwa:

```
LOCK, {*F,*A}, *E, {*C,*D}, *B, UNLOCK

{*F,*A} oznacza jednoczesny dostep.
```

Ale ŻADNA z poniższych już nie:

```
{*F,*A}, *B, LOCK, *C, *D, UNLOCK, *E
*A, *B, *C,  LOCK, *D,     UNLOCK, *E, *F
*A, *B,      LOCK, *C,     UNLOCK, *D, *E, *F
*B,          LOCK, *C, *D, UNLOCK, {*F,*A}, *E
```

5 Architektury 32-bitowe i 64-bitowe

5.1 Czy 64-bitowe adresowanie to dzisiaj konieczność?

Ilość pamięci RAM w obecnie sprzedawanych komputerach wynosi zazwyczaj 2 lub 4 GB. Jednak w serwerach jest ona znacznie większa. Taka duża przestrzeń adresowa jest bardzo przydatna w symulacjach komputerowych i w bazach danych, gdyż ograniczenie się do 4 GB pamięci RAM dostępnej dla takiego programu powoduje znaczny spadek wydajności. 32-bitowa architektura procesora ogranicza jednak do 4 GB ilość pamięci, z jakiej może korzystać aplikacja.

W poniższej tabeli przedstawiono obsługiwaną przez niektóre systemy Windows ilość pamięci wirtualnej i fizycznej.

system operacyjny	*pamięć dostępna dla jednej aplikacji*	*maksymalna obsługiwana wielkość pamięci RAM*
Windows XP Professional (32-bit)	*4 GB*	*4 GB*
Windows XP Professional (64-bit)	*16 TB*	*128 GB*
Windows Server 2003 Standard (32-bit)	*4 GB*	*4 GB*
Windows Server 2003 Datacenter (32-bit)	*4 GB*	*128 GB (przy wsparciu sprzętu)*
Windows Server 2003 Datacenter (64-bit)	*16 TB*	*512 GB*

Jak widać 64-bitowa architektura pozwala znacząco zwiększyć zarówno logiczną jak i fizyczną przestrzeń adresową.

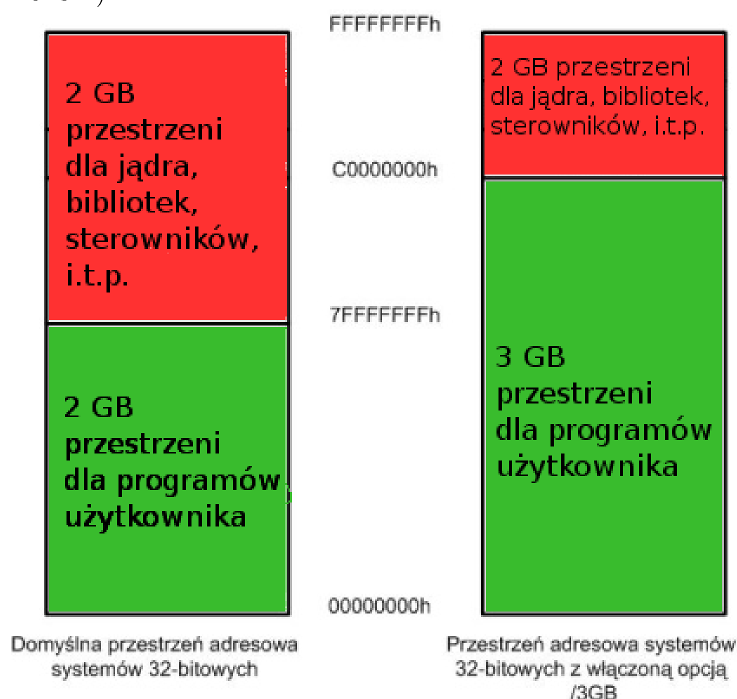
Przedostatni wiersz tabeli sugeruje jednak, że przy odpowiednim wsparciu systemu operacyjnego jest możliwe wykorzystanie pamięci RAM ponad 4 GB nawet przy zastosowaniu 32-bitowej architektury procesora.

5.1.1 Przydzielanie pamięci dla aplikacji w systemie Windows

Przestrzeń adresowa dostępna dla aplikacji w 32-bitowym systemie operacyjnym wynosi 4 GB.

W rzeczywistości jednak w 32-bitowym systemie Windows aplikacja ma na swoje potrzeby mniej niż 4 GB. Standardowo logiczna przestrzeń adresowa wielkości 4 GB jest bowiem podzielona na dwie równe części. Tylko dolną częścią może dysponować aplikacja użytkownika, gdyż górna cześć jest przeznaczona dla systemu operacyjnego.

Poprzez odpowiednią modyfikację pliku boot.ini można zmienić ten podział tak, by użytkownik miał do dyspozycji nieco więcej pamięci (dozwolone są wartości od 2 do 3 GB).



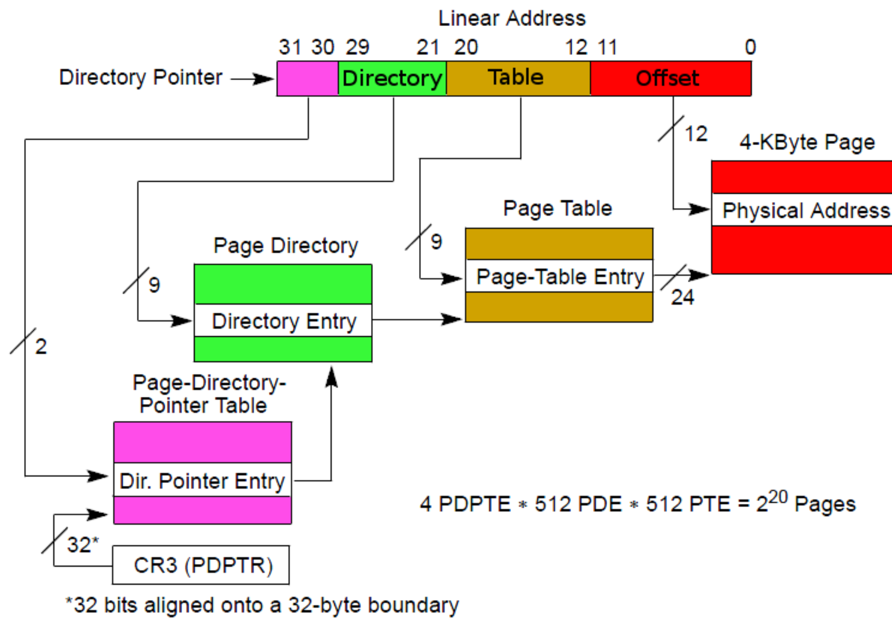
5.1.2 PAE - Physical Address Extension

Wracając do przedstawionej wcześniej tabeli wyjaśnię teraz, w jaki sposób przy odpowiednim wsparciu systemu operacyjnego jest możliwe wykorzystanie pamięci RAM ponad 4 GB nawet przy zastosowaniu 32-bitowej architektury procesora.

Jednym z mechanizmów to umożliwiających jest wprowadzone w procesorze Pentium Pro rozszerzenie adresu fizycznego. Odbywa się ono poprzez przejście procesora w specjalny 36-bitowy tryb adresowania. Tryb PAE (Physical Address Extension) umożliwia procesorom x86 dostęp do większej ilości pamięci fizycznej niż 4 GB. Wymaga on wsparcia systemu operacyjnego (np. Windows XP, Windows Vista, Windows Server, Linux 2.6).

Procesory wspierające ten tryb mają zewnętrzną magistralę adresową o co najmniej 36 wyprowadzeniach. Można dzięki temu zaadresować maksymalnie 64 GB pamięci RAM (36 bitów) na procesorze 32-bitowym. Wówczas każdy uruchomiony 32-bitowy program (który korzysta maksymalnie z 4 GB RAM) umieszcza się w tej większej 64 GB fizycznej przestrzeni adresowej.

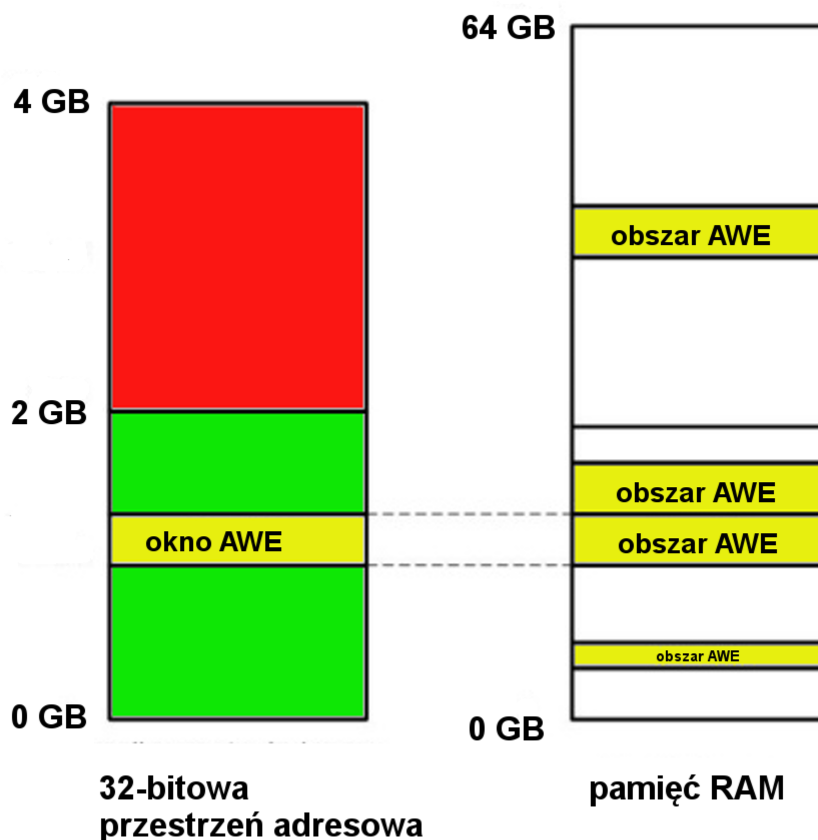
Trójpoziomowe stronicowanie w tym trybie jest przedstawione na rysunku:



5.1.3 AWE - Address Windowing Extensions

Inną metodą uzyskania dużej przestrzeni adresowej na procesorach 32-bitowych jest Address Windowing Extensions. Ma ona zastosowanie w przypadku dysponowania większą niż 4 GB ilością pamięci operacyjnej. Polega ona na tym, że poprzez "wąskie okienko" w standardowej przestrzeni adresowej dla aplikacji użytkownika można uzyskać dostęp do dodatkowego obszaru.

Na przykład serwer bazy danych, oprócz korzystania z przydzielonych mu przez system operacyjny 2 GB, może dodatkowo w innym obszarze zaalokować 6 GB pamięci RAM (dzięki temu nie musi tak często pobierać danych z dysku twardego). AWE jest bowiem zbiorem API, które pozwalają procesowi zaalokować niestronicowaną pamięć fizyczną i potem dynamicznie zamapować ją na swoją wirtualną przestrzeń adresową.



Dodatkowa pamięć nie jest dostępna bezpośrednio, lecz poprzez mapowane okno. Wiążą się z tym pewne ograniczenia:

- procesy nie mogą współdzielić stron pamięci
- nie można tej samej ramce w pamięci RAM przypisać kilku różnych adresów logicznych w ramach jednego procesu
- jeśli wiele aplikacji używa AWE, to mogą one wyczerpać całą dostępną pamięć fizyczną, uniemożliwiając kolejnym korzystanie z tego mechanizmu
- aplikacje używające AWE nie mogą być emulowane na innej platformie.

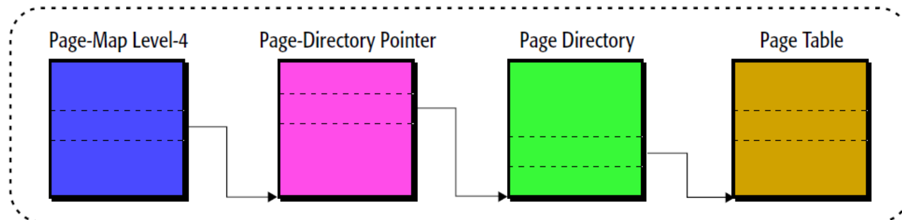
Alternatywą w Linuksie dla AWE jest funkcja systemowa mmap() (która mapuje pliki lub urządzenia do pamięci).

5.2 Adresowanie w trybie 64-bitowym

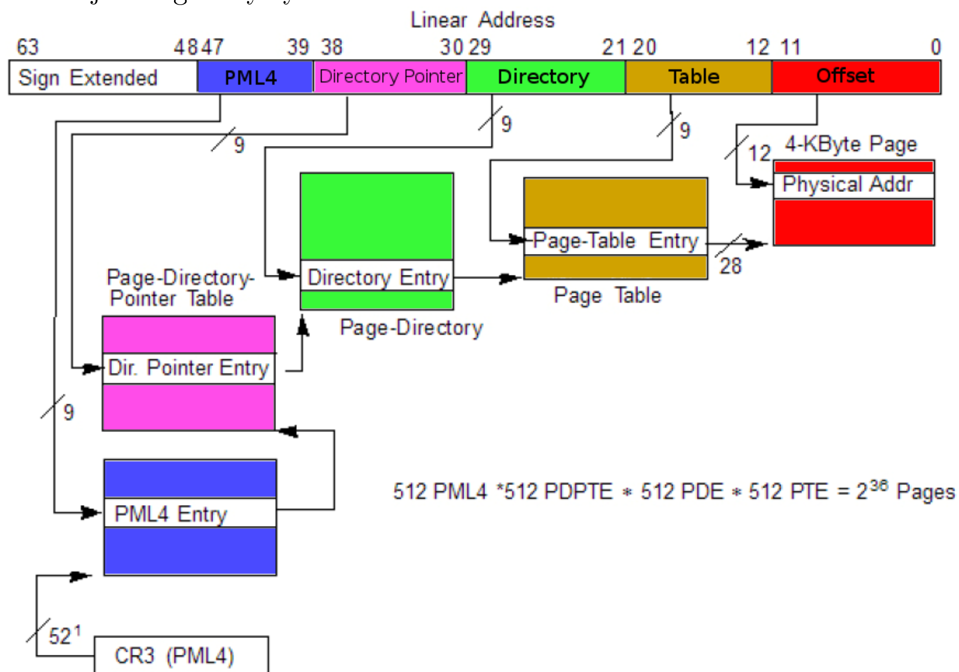
Przedstawione mechanizmy zwiększania przestrzeni adresowej w architekturze 32-bitowej komplikują mechanizm zarządzania pamięcią przez system operacyjny. Mają też pewne ograniczenia, które czynią je nieefektywnymi dla bardziej profesjonalnych

zastosowań. Dlatego też coraz częściej stosuje się 64-bitowe systemy operacyjne. Wymagają one do działania 64-bitowych procesorów (od niedawna tylko takie są w sprzedaży), na przykład IA-32e lub AMD64.

W trybie IA-32e, przy stronie 4 KB, stosuje się czteropoziomowe stronicowanie:

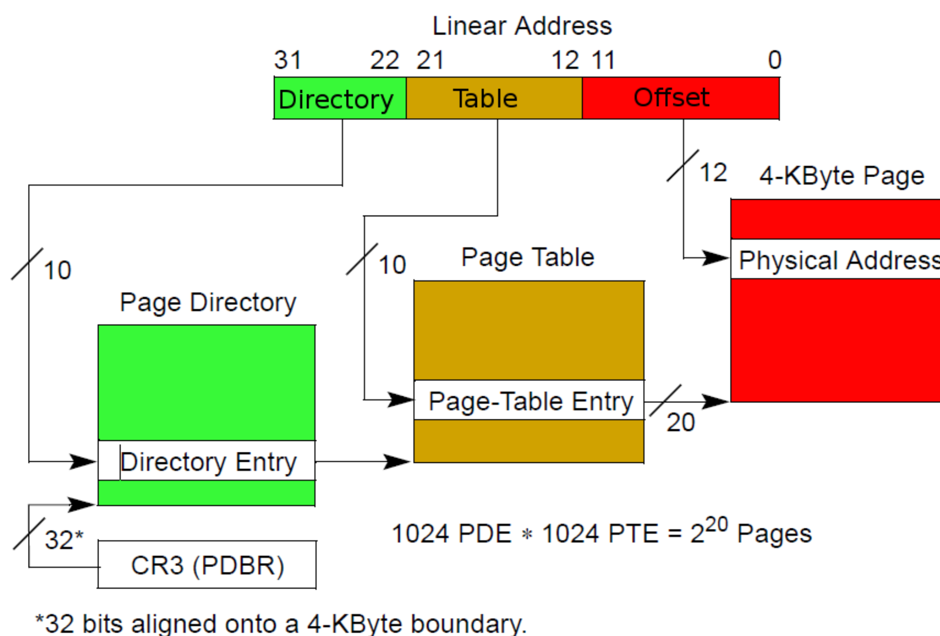


Bardziej szczegółowy rysunek:



Przestrzeń adresowa ma długość mniejszą niż 64 bity, gdyż nie jest nikomu potrzebna pamięć wielkości 2^{64} bajtów (pozostałe, nieużywane bity są zarezerwowane na przyszłość).

Dla porównania w trybie 32-bitowym, przy stronie 4 KB, stosuje się dwupoziomowe stronicowanie:



5.2.1 Segmentacja w trybie 64-bitowym

W trybie IA-32e segmentacja jest niemożliwa do zrealizowania przez system operacyjny (istnieje ”płaska” 64-bitowa liniowa przestrzeń adresowa). Procesor traktuje bowiem rejestry CS, DS, ES, SS jako zero. Natomiast rejestry FS, GS mogą być używane jako dodatkowe rejestry bazowe w obliczeniach liniowych adresów. Są one wykorzystywane w niektórych strukturach danych systemu operacyjnego.

5.2.2 Wielkość stron

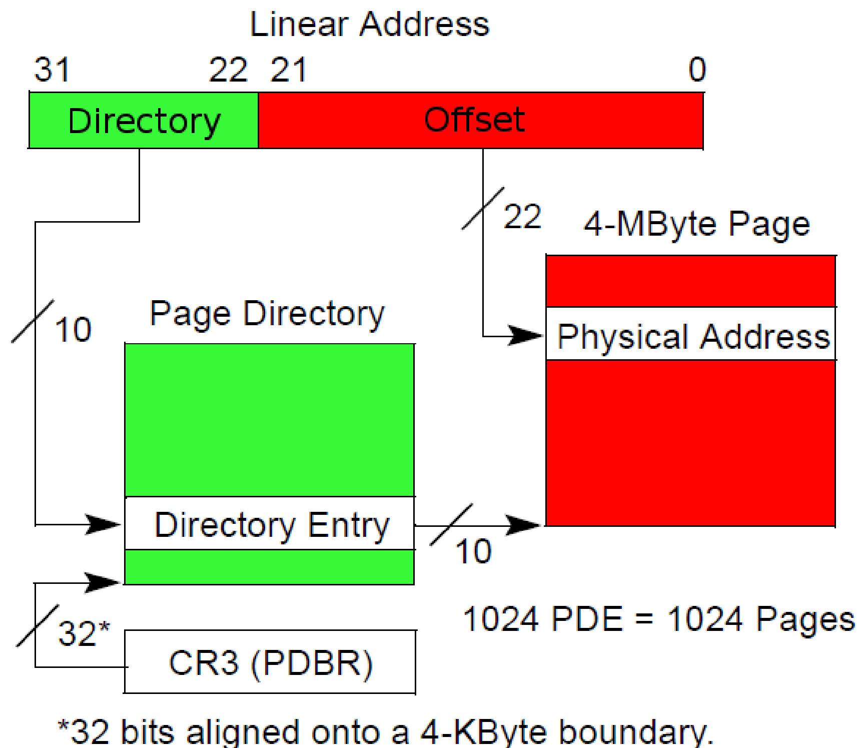
System operacyjny powinien dostosować wielkość stron do architektury procesora i wykonywanych zadań. Domyślnie wynosi ona 4 KB (Small Page Size). Możliwa jest jednak większa wartość:

- 4 MB dla x86
- 2 MB dla x64 i x86 w trybie PAE

Większy rozmiar strony zwiększa efektywność pamięci podręcznej procesora (co zostanie dalej opisane). System operacyjny może zwiększyć wielkość strony tylko gdy jest zainstalowana dostatecznie duża ilość pamięci RAM (na przykład Windows XP wymaga co najmniej 256 MB). Oprócz tego procesor musi zapewniać specjalny tryb adresowania.

Procesory Intela i AMD posiadają tryb PSE (Page Size Extensions), który pozwala na stosowanie dużych rozmiarów stron (2 lub 4 MB) i w konsekwencji

zmniejszenie liczby poziomów stronicowania o jeden. Na przykład 32-bitowy procesor bez PAE ma wówczas jednopoziomowe stronicowanie:



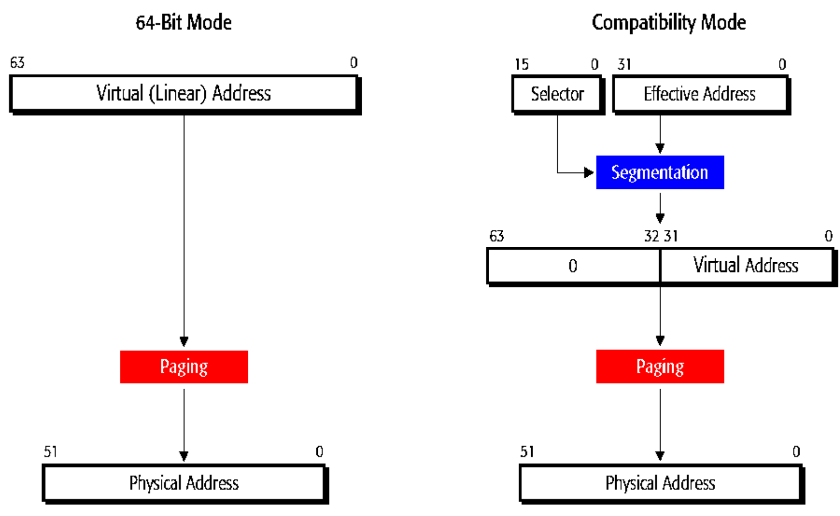
5.3 Tryby pracy procesora 64-bitowego

Procesory 64-bitowe (na przykład o architekturze IA-32e) pozwalają na pracę w dwóch trybach:

1. Natywnym (long mode) - w którym pamięć jest adresowana 64-bitowo. Jest on wykorzystywany przez 64-bitowe systemy operacyjne. Ten tryb wymaga od systemu operacyjnego pracy w trybie chronionym, zatem nie jest obsługiwany 16-bitowy tryb rzeczywisty. Posiada on dwa podtryby:

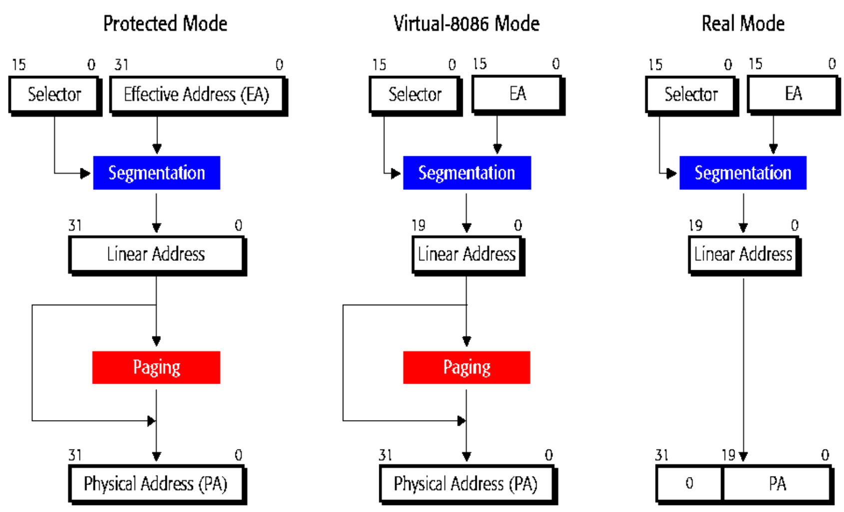
   - 64-Bit Mode - zapewnia pełne wsparcie dla 64-bitowego oprogramowania.
   - Compatibility Mode - pozwala na zaimplementowanie w 64-bitowym systemie operacyjnym binarnej kompatybilności z istniejącymi 16 i 32-bitowymi aplikacjami bez potrzeby rekompilacji.

   Podtryby trybu long mode są przedstawione na rysunku:



2. Kompatybilnym (legacy mode), który umożliwia uruchamianie programów 32-bitowych bez jakiejkolwiek modyfikacji. Jest on wykorzystywany przez 16-bitowe i 32-bitowe systemy operacyjne. W tym trybie procesor zachowuje się po prostu tak samo jak procesor 32-bitowy (a więc nie pozwala na uruchamianie aplikacji 64-bitowych). Jest więc możliwy tryb rzeczywisty systemu operacyjnego.

   Podtryby trybu legacy mode są przedstawione na rysunku:

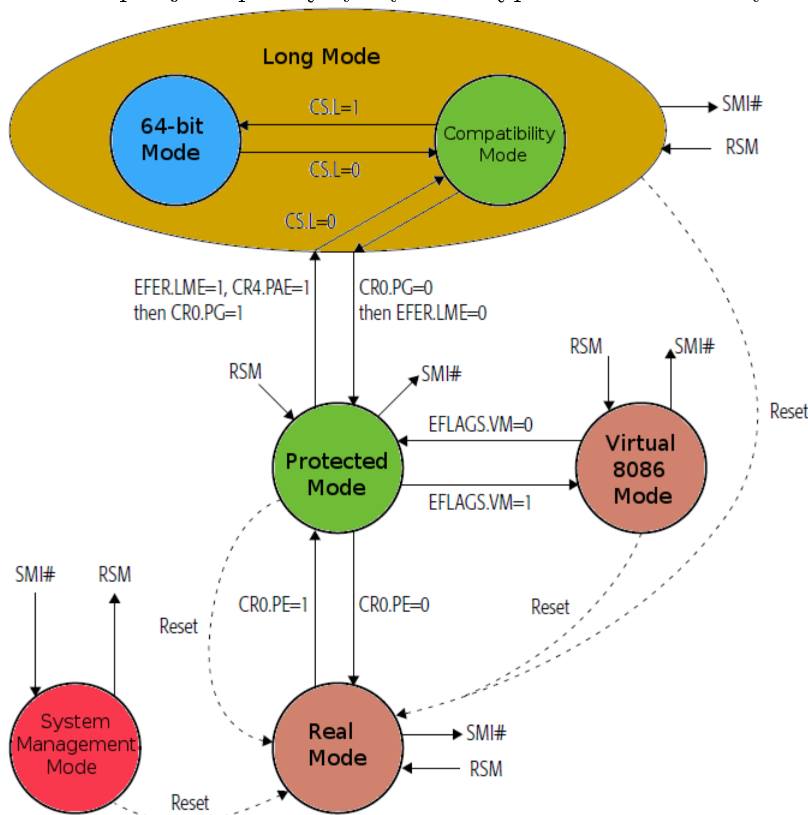


Podsumowanie trybów pracy procesora znajduje się w tabeli:

Operating Mode		Operating System Required	Application Recompile Required	Defaults: Address Size (bits)	Defaults: Operand Size (bits)	Register Extensions	Typical GPR Width (bits)
Long Mode	64-Bit Mode	64-bit OS	yes	64	32	yes	64
	Compatibility Mode		no	32		no	32
				16	16		16
Legacy Mode	Protected Mode	Legacy 32-bit OS	no	32	32	no	32
				16	16		
	Virtual-8086 Mode			16	16		16
	Real Mode	Legacy 16-bit OS					

System operacyjny w momencie startu pracuje w trybie 16-bitowym. Następnie poprzez odpowiednie wpisy do rejestrów systemowych przełącza się w tryb 32-bitowy, a potem (gdy procesor i system operacyjny są 64-bitowe) w tryb 64-bitowy.

Możliwe przejścia pomiędzy trybami są przedstawione na rysunku:



5.4 64-bitowe systemy operacyjne - przykłady

Aby wykorzystać architekturę 64-bitową, systemy operacyjne muszą być w znacznym stopniu zmodyfikowane w stosunku do swoich 32-bitowych wersji. Zazwyczaj działają na nich zarówno 32 jak i 64-bitowe aplikacje. Praktyka pokazuje jednak, że czasami występują z tym problemy.

5.4.1 64-bitowe edycje systemu Windows

Największym problemem w 64-bitowym Windowsie są niekompatybilne sterowniki urządzeń. Zazwyczaj jest niemożliwe uruchamianie ich tak jak inne aplikacje. Działają one bowiem bezpośrednio pomiędzy systemem operacyjnym a sprzętem. Nie można więc po prostu zastosować emulacji. Konieczne są zatem 64-bitowe sterowniki. W związku z tym logo "Certified for Vista" może uzyskać tylko sprzęt, do którego zostaną dostarczone sterowniki 32 i 64-bitowe. Dzięki temu, że wszystkie sterowniki muszą być cyfrowo podpisane, zwiększa się stabilność systemu.

Wszystko to sprawia, że wiele, zwłaszcza starszych i tańszych, urządzeń nie działa w 64-bitowym Windowsie.

Poza tym wiele starych sterowników używa 32-bitowych wskaźników. Na przykład 32-bitowe urządzenie PCI może chcieć mieć bezpośredni dostęp do 64-bitowej pamięci RAM w trybie DMA. Powoduje to problemy, bo system operacyjny może dać jedynie 64-bitowy wskaźnik, który nie mieści się w rejestrach tego urządzenia. W takim przypadku system operacyjny powinien odpowiednio na to zareagować, zmieniając odpowiednio sposób komunikacji ze sterownikiem.

64-bitowe wersje systemu Windows posiadają następujące ograniczenia pamięciowe:

- maksymalnie 16 TB logicznej przestrzeni adresowej (8 TB dla procesów jądra i 8 TB dla procesów użytkownika)

- nie więcej niż 128 GB (Windows XP) lub 1 TB (Windows Server 2003) pamięci RAM

64-bitowy Windows używa technologii Windows-on-Windows 64-bit (WOW64), która umożliwia uruchamianie 32-bitowych aplikacji. Ponieważ architektura x86-64 wspiera 32-bitowe instrukcje, to WOW64 może przełączać proces pomiędzy 32 i 64-bitowym trybem. Dzięki temu nie występuje spadek wydajności 32-bitowych programów. Nie działają jednak żadne programy 16-bitowe (np. DOS).

Nie jest też możliwe mieszanie w jednym procesie kodu 32 i 64-bitowego. Dlatego też 64-bitowe programy nie mogą ładować 32-bitowych bibliotek DLL, i odwrotnie: 32-bitowe programy nie mogą ładować 64-bitowych bibliotek DLL. Na przykład w 64-bitowym programie Internet Explorer nie działają żadne 32-bitowe dodatki.

64-bitowa edycja Windows Vista zawiera zarówno 32 jak i 64-bitowe biblioteki. Te pierwsze służą technologii WOW64. Muszą być one ładowane do pamięci przy uruchamianiu programu 32-bitowego. Dlatego też ta warstwa emulacji sprawia, że 64-bitowa Vista ma większe wymagania co do pamięci RAM.

5.4.2 Linux

Linux był jednym z pierwszych systemów operacyjnych, na których można było stosować architekturę 64-bitową. System ten posiada wsteczną kompatybilność umożliwiającą uruchamianie 32-bitowych programów. Dzięki temu jest możliwa 64-bitowa rekompilacja programów, które wykorzystują inne, 32-bitowe programy.

Kilka dystrybucji Linuksa jest rozpowszechniana z 64-bitowym jądrem i środowiskiem użytkownika. Niektóre są dostępne w wersji 32-bitowej i 64-bitowej na pojedynczej płycie DVD w celu automatycznego wyboru właściwego dla komputera oprogramowania podczas instalacji. Inne są dostępne w osobnych wersjach: 32-bitowej i 64-bitowej.

64-bitowy Linux daje każdemu procesowi do 128 TB przestrzeni adresowej i potrafi zaadresować około 64 TB pamięci RAM w zależności od procesora i ograniczeń systemowych.

5.4.3 Mac OS X

W niektórych systemach operacyjnych jest możliwe uruchamiać jądro jako 32-bitowy proces, wspierając jednocześnie 64-bitowe procesy użytkownika. Dzięki temu użytkownicy mogą korzystać z większej przestrzeni adresowej i większej wydajności, bez utraty binarnej kompatybilności z istniejącymi 32-bitowymi sterownikami urządzeń. Wymaga to jedynie zmian w jądrze takiego systemu operacyjnego. W ten sposób działa właśnie system operacyjny Mac OS X.

W systemie Mac OS jest stosowana także inna technologia pozwalająca na uniezależnienie się od architektury procesora.

”Universal binary” to aplikacja, która może być uruchomiona natywnie zarówno na procesorach PowerPC jak i na procesorach Intela. ”Universal binary” zawiera zazwyczaj obie wersje skompilowanej aplikacji. System operacyjny wykonuje właściwą jej część w zależności od architektury procesora.

Rozmiar takiej aplikacji nie jest jednak dwukrotnie większy, gdyż niektóre elementy są współdzielone dla obu wersji. Poza tym do pamięci RAM ładowana jest tylko jedna wersja.

Format ”universal binary” daje też możliwość jednoczesnego dostarczenia aplikacji w wersjach: 32 i 64-bitowej. Tak więc ”universal binary” może zawierać nawet cztery wersje kodu wykonywalnego (32-bitowy PowerPC, 32-bitowy Intel, 64-bitowy PowerPC i 64-bitowy Intel).

6 Pamięć podręczna procesora

6.1 Spójność pamięci cache w procesorach Intela i AMD

Procesor nie zapewnia, że dane znajdujące się w jego pamięci podręcznej są zawsze spójne z pamięcią RAM. Dlatego też system operacyjny powinien rozpoznawać sytuacje, kiedy dochodzi do tej niespójności i w razie potrzeby usuwać niespójne dane z pamięci cache.

W celu zachowania spójności w hierarchii pamięci stosuje się między innymi instrukcję WBINVD. Zapisuje ona wszystkie zmienione linie we wszystkich poziomach pamięci podręcznej do pamięci operacyjnej, a następnie opróżnia całą pamięć podręczną.

6.2 MTRR - memory type range registers

Procesor pozwala na ustalenie regionów w pamięci, dla których korzystanie z pamięci podręcznej odbywa się w inny niż standardowy sposób. Odbywa się to poprzez odpowiednie wpisy do rejestrów MTRR (memory type range registers). Rejestry te pozwalają powiązać typ pamięci z zakresem przestrzeni adresowej. Dzięki temu procesor może optymalizować operacje na różnych typach pamięci, np. RAM, ROM, mapowane do pamięci urządzenia I/O.

Jednym z takich sposobów jest Strong Uncacheable (UC). Objęte nim lokacje w pamięci nie są nigdy sprowadzane do pamięci cache. Wszystkie odczyty i zapisy odbywają się poprzez szynę systemową. Jest to użyteczne w mapowanych do pamięci urządzeniach I/O.

Zazwyczaj MTRR są konfigurowane przez BIOS. W systemach wieloprocesorowych system operacyjny musi utrzymywać spójność MTRR (bo oczywiście różne procesory muszą używać tych samych wartości MTRR). Procesory AMD i Intela nie zapewniają bowiem sprzętowego wsparcia dla utrzymania tej spójności.

6.3 Kolorowanie

Kolorowanie oznacza alokowanie tych wolnych stron w pamięci operacyjnej, którym odpowiadają różne linie pamięci podręcznej procesora. Mechanizm ten zapewnia efektywniejsze zarządzanie pamięcią cache procesora dzięki zwiększeniu współczynnika trafień.

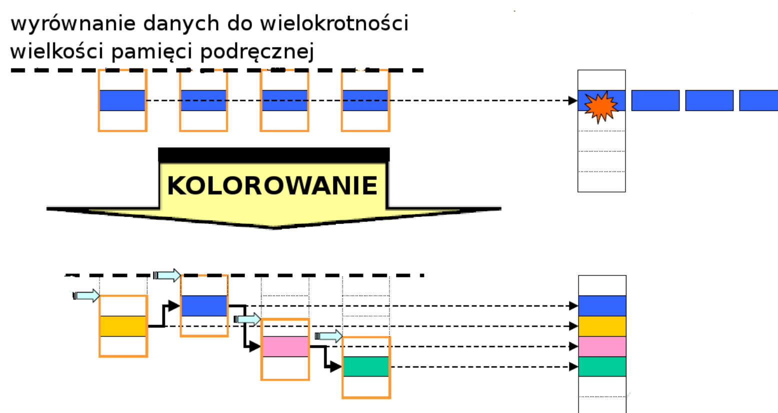
W niektórych zastosowaniach mamy bowiem do czynienia ze strukturami danych, których każdy element jest w ten sam sposób wyrównany w pamięci. Powoduje to, że wszystkie te elementy są odwzorowywane na tę samą linię pamięci cache. Wobec tego przy odwoływaniu się kolejno do tych elementów prawie za każdym razem następuje chybienie. Podsystem pamięci wirtualnej, który nie korzysta z kolorowania jest więc mniej przewidywalny pod względem wydajności.

Kolorowanie rozwiązuje ten problem poprzez sztuczną zmianę wyrównania danych w pamięci.

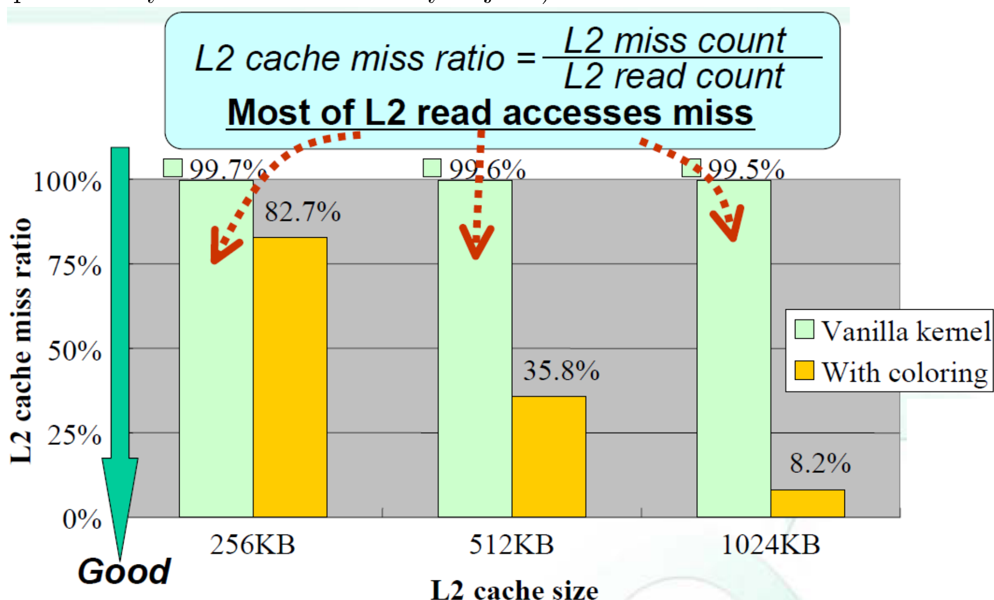
Mechanizm ten jest zazwyczaj wprowadzany przez niskopoziomowy kod dynamicznej alokacji pamięci w systemie operacyjnym, podczas mapowania pamięci wirtualnej na pamięć fizyczną. Co prawda kod wprowadzający obsługę kolorowania znacznie komplikuje podsystem alokowania pamięci wirtualnej, ale rezultat jest wart tego wysiłku.

6.3.1 Przykład: przyspieszanie schedulera w Linuksie

Można zmniejszyć czas przejścia przez kolejkę procesów gotowych, stosując kolorowanie:



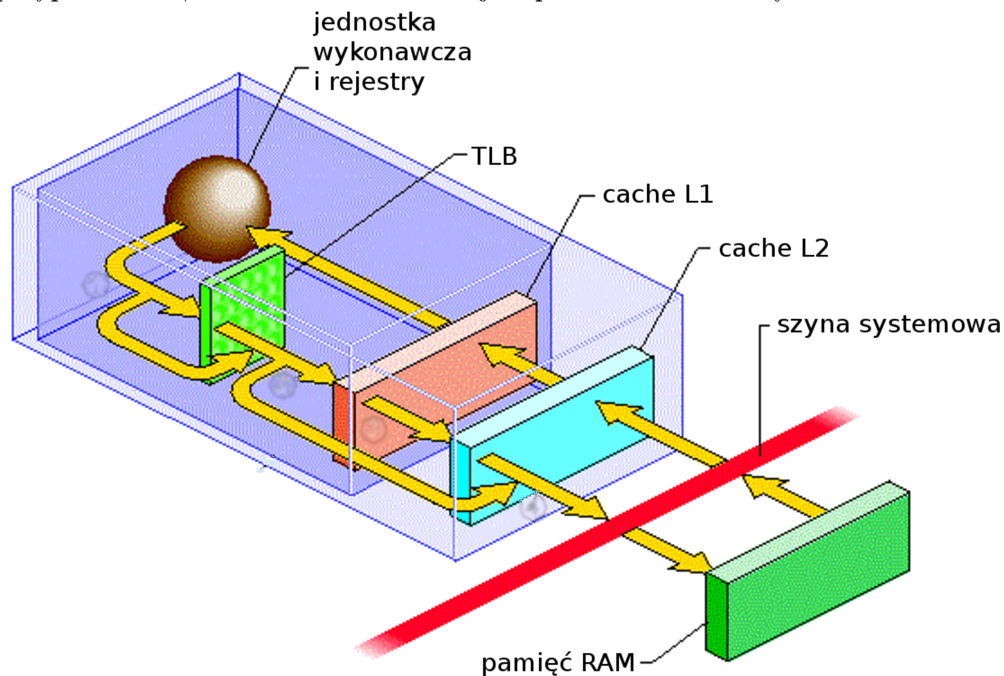
Różnica w wydajności jest szczególnie widoczna w przypadku Linuksa 2.4 (test przeprowadzony w laboratorium firmy Fujitsu):



Widać z tego wykresu, że kolorowanie jest bardziej opłacalne w przypadku dużych pamięci cache L2.

6.4 TLB - translation lookaside buffer

Dla przypomnienia, działanie bufora TLB jest przedstawione na rysunku:



TLB jest niedostępny dla procesów z priorytetem większym niż 0. Tylko system operacyjny może wywołać procedurę o priorytecie 0, która może opróżnić część lub całość TLB.

W celu zachowania spójności TLB, za każdym razem, kiedy katalog lub tablica stron ulega zmianie, system operacyjny musi natychmiast unieważnić odpowiedni wpis w TLB (poprzez instrukcję INVLPG), aby został on uaktualniony przy następnym odwołaniu do tego katalogu/tablicy stron.

Konieczność modyfikacji TLB zachodzi między innymi podczas przełączania kontekstu. Bez tego w buforze TLB zostałyby bowiem niepoprawne adresy wirtualne (gdyż te same logiczne adresy różnych procesów nie odpowiadają tym samym fizycznym adresom).

System operacyjny może radzić sobie z tym problemem na dwa sposoby:

- przez opróżnianie całej zawartości TLB przy zmianie kontekstu
- przez zapisywanie identyfikatorów przestrzeni adresowej (ASID - address space identifiers) razem z każdym wpisem w TLB. Jeśli identyfikatory ASID nie zgadzają się, to sygnalizowane jest chybienie. Dzięki temu zapewniona jest taka ochrona przestrzeni adresowej, która umożliwia jednoczesne korzystanie z TLB przez wiele procesów.

6.4.1 Chybienie w TLB

W zależności od architektury procesora, chybienie w TLB może być w różny sposób obsługiwane:

- W sprzętowym zarządzaniu TLB (np. w procesorach IA32), procesor samodzielnie przechodzi po tablicach stron aby sprawdzić, czy istnieje ważny wpis dla danego adresu logicznego. Jeśli tak, to ten wpis jest przenoszony do TLB. W przeciwnym razie procesor sygnalizuje systemowi operacyjnemu wyjątek braku strony. Wówczas system operacyjny sprowadza żądane dane do pamięci RAM, koryguje odpowiednio tablicę stron, a następnie wznawia proces.
- W programowym zarządzaniu TLB (np. w procesorach IA64), procesor od razu generuje wyjątek "TLB miss". System operacyjny przechodzi po tablicach stron i przeprowadza konieczną translację programowo. Następnie ładuje uzyskany wynik do TLB i wznawia program. Taki mechanizm daje systemowi operacyjnemu większe możliwości wyboru algorytmu zarządzania pamięcią. Podobnie jak w poprzednim przypadku, brak strony w pamięci RAM powoduje wyjątek braku strony (który też musi być obsłużony przez system operacyjny).

6.4.2 Zasięg bufora TLB

Jak było wcześniej wspomniane, procesor umożliwia korzystanie zarówno ze stron rozmiaru 4 KB jak i większych - rozmiaru 2 MB lub 4 MB. Wadą dużych stron jest oczywiście większa fragmentacja pamięci. Jednak ich stosowanie zwiększa rozmiar pamięci, przy której adresowaniu nie występuje chybienie w TLB. Dlatego też, zwiększając n-krotnie rozmiar strony, zwiększamy n-krotnie zasięg TLB, w rezultacie osiągając wyższy współczynnik trafień w TLB.

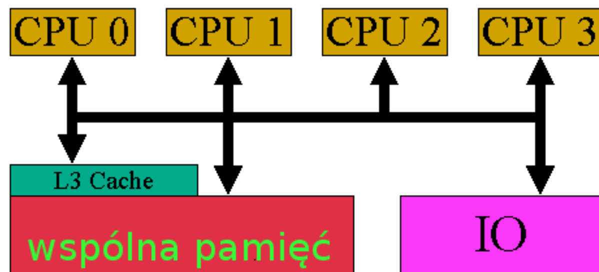
W niektórych architekturach stosuje się różne wielkości stron dla różnych typów aplikacji (np. bazy danych są bardziej wydajne przy większych rozmiarach stron). Wymaga to oczywiście programowego zarządzania TLB przez system operacyjny.

Jednak w przypadku pecetów stosuje się zazwyczaj stałą wielkość stron, umożliwiającą sprzętowe zarządzanie buforem TLB (które jest w tym przypadku szybsze). Na przykład Windows XP dla procesorów zgodnych z IA32 używa zawsze stron wielkości 4 KB.

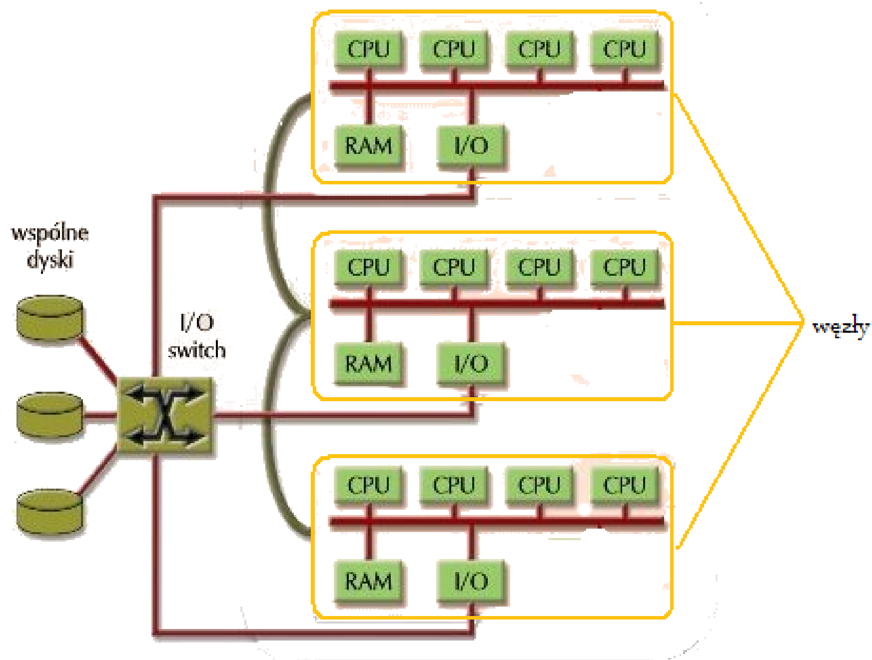
7 Architektura NUMA

7.1 NUMA a SMP

Komputery mające kilka lub kilkanaście procesorów zazwyczaj mają architekturę SMP. Charakteryzuje się ona pełną symetrią tych procesorów ze względu na dostęp do pamięci (UMA, czyli Uniform Memory Access):



Alternatywą jest NUMA (Non-Uniform Memory Access lub Non-Uniform Memory Architecture, czasami też DSM - Distributed Shared Memory). Jest to sposób organizacji pamięci w systemach wieloprocesorowych, w którym dla każdego procesora czas dostępu do pamięci zależy do miejsca, gdzie ta pamięć się znajduje. Zazwyczaj procesory są grupowane w węzły mające wspólną pamięć:



Każdy węzeł ma dostęp nie tylko do swojej lokalnej pamięci, ale też do pamięci innych węzłów i pamięci dzielonej przez wszystkie węzły (dostęp do odległej pamięci jest jednak wolniejszy).

W porównaniu z architekturą SMP, NUMA sprawdza się o wiele lepiej w ob-

liczeniach rozproszonych. Ponadto jest ona bardziej skalowalna. Można bowiem zainstalować więcej procesorów dzięki mniejszemu obciążeniu szyny pamięci.

Oczywiście czasami więcej niż jeden procesor może potrzebować tych samych danych. Dlatego też NUMA posiada dodatkowy sprzęt lub odpowiedni system operacyjny, umożliwiający migrację danych pomiędzy węzłami.

7.2 ccNUMA - cache coherent NUMA

Właściwie wszystkie stosowane obecnie realizacje NUMA zapewniają spójność pamięci podręcznej procesorów, więc są nazywane ccNUMA.

W architekturze NUMA, utrzymanie spójności pamięci podręcznych procesorów jest trudne w przypadku, gdy wiele pamięci podręcznych przechowuje dane z tego samego adresu. W celu rozwiązania tego problemu niezbędne jest wsparcie systemu operacyjnego. Z tego powodu ccNUMA działa wolno gdy wiele węzłów chce mieć częsty dostęp do tego samego obszaru pamięci.

Wsparcie systemu operacyjnego dla NUMA polega na redukcji częstości tego typu dostępu przez przydzielanie procesorów i alokowanie pamięci w odpowiedni dla tej architektury sposób. Poza tym system operacyjny powinien unikać stosowania algorytmów szeregowania i blokowania, które wymagają takiego wspólnego dostępu do pamięci.

Aby uzyskać wysoką wydajność, stronicowanie systemu operacyjnego powinno zapewniać mechanizm migracji stron. Powinien on automatycznie przenosić stronę do tego węzła, który często jej używa.

8 VLIW - Very Long Instruction Word

8.0.1 Bardzo długie instrukcje

- W jednej instrukcji kodujemy wiele atomowych operacji które procesor wykona równolegle

  Procesor w tej architekturze zawiera kilka niezależnych jednostek obliczeniowych. Przy dekodowaniu długiej instrukcji, każda jednostka otrzymuje jeden, krótki rozkaz.

- Przykład (arch. SHARC - Super Harvard Architecture Single-Chip Computer)

  ```
  f12=f0*f4, f8=f8+f12, f0=dm(i0,m3), f4=pm(i8,m9);
  ```

  fx - rejestry tej maszyny, liczba rejestrow musi byc dość duża żeby dało się wykonać wiele obliczeń naraz

Własności:

- Zbiór możliwych instrukcji zależy od liczby jednostek wykonawczych

  Co znaczy, że kompilator musi być tego świadom, programy są nieprzenośne.

- Rozwiązywanie problemów związanych z zrównoleglaniem kodu, wyborem które instrukcje będą wykonywane jako następne (if-y) itp jest przerzucone na kompilator

  Kompilator ma większe możliowości stosowania różnego rodzaju heurystyk dotyczących struktury kodu jako, że łatwiej jest takie mechanizmy umieścić w programie niż w wprost w elektronice. Poza tym kompilator działa offline, przed wykonaniem programu, można więc optymaliować kod maszynowy kosztem dłuższej kompilacji.

8.0.2 Zalety i wady

- Uproszczony hardware
- Zapewne lepsze przewidywania co do wyboru gałęzi kodu
- Jeśli mamy dobry kompilator to powinniśmy mieć lepsze rezulaty niż dla potokowania
- Niezgodność z innymi architekturami, złożone kompilatory

  Stosuje się różne obejścia tego problemu, np programy które w locie tłumaczą kod maszynowy z innych architektur na VLIW

- Czasem instrukcje są w znacznej mierze puste

  No bo nie wszystko da się zrównoleglić

9 Wieloprocesorowość, wielordzeniowość

9.1 Wieloprocesorowosc, wielordzeniowosc

9.1.1 Wieloprocesorowość

- Komputery z wieloma procesorami, każdy z nich na osobnej płytce z krzemu, z własną pamięcią podręczną , chłodzeniem itp
- Głównie superkomputery o naprawdę dużej liczbie procesorów i specjalnej budowie
- Specjalne SO, które muszą skutecznie zarządzać tak duża mocą obliczeniową
- Wieloprocesorowość jest raczej rzadko spotykana w komputerach klasy PC

9.1.2 Wielordzeniowość

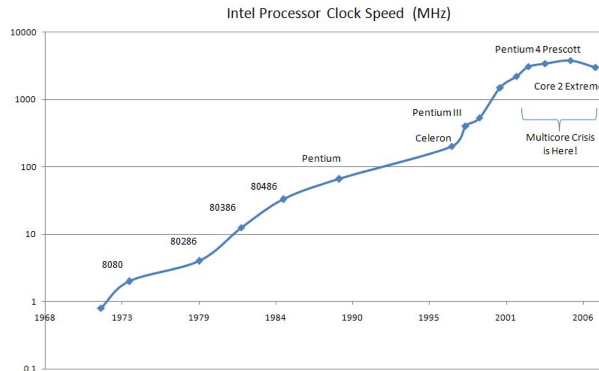
- Dość nowe rozwiązanie
- Projektowane dla komputerów klasy PC

  Chociaż obecnie buduje się też mainframe'y składając procesory wielordzeniowe, powstaje coś podobnego do NUMA
- Obecnie powszechnie stosowane

  Praktycznie każdy nowy komputer ma kilka rdzeni, wyścig „kto ma większe taktowanie" zamienił się w „ kto ma więcej rdzeni"
- Popularne SO muszą je obsługiwać aby pozostać popularnymi

  Bo nikomu nie będzie potrzebny system który wykorzystuje jeden rdzeń z czterech. Linux jak przystosowany do wieloprocesorowości, nie ma problemów z obługą wielu rdzeni. Windows również je obsługuje, chociaż poszczególne wersje tego systemu mają sztywne ograniczenia na liczbę procesorów fizycznych (ograniczenie to jest dość sztuczne bo np można mieć 2x2 rdzenie a nie można 4 osobnych procesorów. Takie problemy licencyjne występują też w różnego rodzaju oprogramowaniu).

9.2 Dlaczego wiele rdzeni?



Prędkość pojedyńczego procesora/rdzenia w funkcji czasu

Widzimy, że rozwój jednordzeniowych procesorów uległ zatrzymaniu w ostatnich latach.

9.2.1 Problemy z jednordzeniowymi procesorami

- Prawo Moore'a

  W wersji technicznej mówi, że przy stałej cenie procesora liczba tranzystorów które można w nim umieścić rośnie wykładniczo.

  W wersji popularnej, że prędkość procesora rośnie wykładniczo. Przy czym prędkość nawet pojedyńczego rdzenia to nie tylko taktowanie, liczą się też „sztuczki" np. potokowanie.

- Granice możliwości technicznych

  Okazuje się, że pojedyńczy szybki rdzeń wydziela za dużo ciepła, poza tym wąskim gardłem jest komunikacja z otoczeniem. Sporo czasu zajmuje też przełączanie kontekstu gdy mamy wiele wątków.

9.2.2 Wiele rdzeni

- Mniejsze wydzielanie ciepła i zużycie energii
- Teoretycznie n-razy szybszy procesor przy tej samej prędkości rdzenia
- Lepsze od kilku procesorów bo wspólny cache, nie zajmują też tyle miejsca na płycie głownej co ważne przy laptopach

Procesor logiczny - pojedyńczy rdzeń lub procesor jednordzeniowy

9.2.3 Problemy z wieloma procesorami logicznymi

- Mało programów równoległych
- „Often it's cheaper to buy proper computer than to rewrite software"
- Zapychanie pasma dostępu do pamięci i urządzeń (np karty sieciowej)

10 Sposoby rozdzielania zadań dla maszyn wieloprocesorowych

10.1 Architektura asynchroniczna (1970-80)

10.1.1 AMP,ASMP,Master - Slave

- Każdy procesor jest traktowany niezależnie
- Procesory mogą być różnego przeznaczenia, różnie taktowane
- Często mają dostęp do różnych urządzeń preferyjnych
- Procesy muszą specyfikować na którym procesorze mają pracować

  Nie da się ich przenosić bo inne procesory mogą nie spełniać wymagań procesu.
- Łatwo o wąskie gardło, nierównomierne rozłożenie obciążenia
- Architektura Master - Slave

  Może być rozumiana na różne sposoby:

  1. jeden główny procesor z SO, pozostałe procesory do szybkich obliczeń
  2. jeden głowny procesor do obliczeń, pozostałe do komunikacji z otoczeniem, przesyłania zadań itp

  Tego typu rozwiązania stosuje się gdy procesorów ma być kilka, w superkomputerach czasem też dodaje się wolniejsze procesory które odpowiadaja za komunikację i nadzór nad szybszymi.

10.1.2 Współczesne przykłady

- Procesor Cell(PS3 ale nie tylko...)

  Procesor glówny i 8 jednostek obliczeniowych PPE. Są plany budowy superkomputerów zbudowany z tych precesorów. Taka maszyna miałaby duże możliwości obliczeń numerycznych. Użycie procesorów produkowanych masowo obniża jej cenę.

- GPU

  Tu też występuje jeden główny procesor i poboczne jednostki - shader units, specjalne programy które na nich działają nazywają się shaderami. Ostatnio są próby użycia GPU i jego jednostek wektorowych do łamania haseł RSA i WPA(„bezpieczne” sieci bezprzewodowe).

- PC

  Właściwie cały komputer można potraktować jako architekturę asynchroniczną - jeden główny CPU i wiele procesorów w kartach, kontrolerach itp.

10.2 SMP,BMP

10.2.1 SMP - Symmetric MultiProcessing

- Wszystkie procesory traktowane równorzędnie

  Procesory powinny być tej samej architektury i podobnej szybkości.

- Procesy mogą swobodnie migrować między procesorami
- SO powinien zapewniać równomierne obciążenie procesorów
- Obecnie najczęściej stosowane
- W linuxie od 1.2.33

10.2.2 Cache

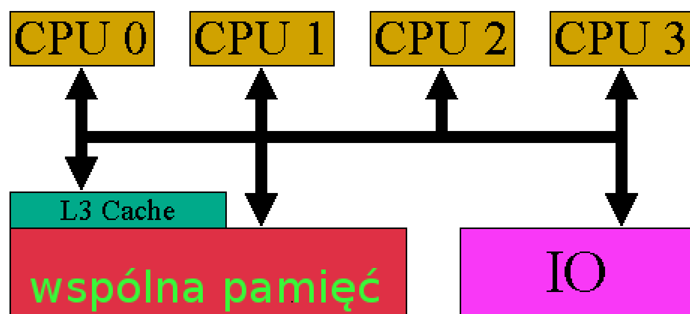
- Poważnym problemem technicznym jest synchronizacja cache
- Zazwyczaj zapewniana sprzętowo, ale powoduje to znaczne spowolnienie tych dostępów do pamięci(szczególnie zapisów, które wymagają synchronizacji)
- BMP - Bounded MultiProcessing

  Ogólnie działa jak SMP ale procesy mają możliwość związania się z danych procesorem logicznym, czy ich grupą. Służy to do grupowania procesów korzystających ze wspólnej pamięci na jednym procesorze logicznym lub na kilku rdzeniach tego samego procesora - rdzenie w znacznej mierze współdzielą cache.

  Zarówno Linux jak i Windowsy obsługują BMP.

10.3 UMA

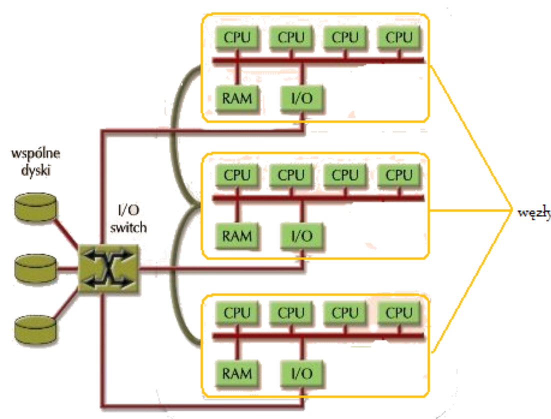
UMA, czyli Uniform Memory Access



Jak nazwa wskazuje, wszystkie procesory mają jednakowy dostęp do pamięci.

10.4 NUMA, ccNUMA

Non-Uniform Memory Access lub Non-Uniform Memory Architecture, czasami też DSM - Distributed Shared Memory



Procesory z jednego węzła mają szybki dostęp do pamięci wewnątrz danego węzła a wolniejszy dostęp do pamięci innych węzłów. Nie jest to ostre ograniczenie ale ze względu na wydajność aplikacje lub SO muszą odpowiednio rozmieszczać prcesy tak, żeby odwołania pomiędzy węzłami były możliwie rzadkie. Architektura NUMA jest łatwiejsza do budowy, narzuca mniejsze wymagania na sprzęt, zwłaszcza przy wielu procesorach.

Obecnie w praktyce ccNUMA- Cache Concurrent NUMA(czyli ze sprzętową synchronizacją pamięci podręcznych).

11 Pamięci podręczne(cache)

Ta część prezentacji powstała jako bardzo skrócona wersja jednego z wykładów z SO. W trakcie prezentacji nie wchodziłem zbytnio w szczegóły, wszystkich zainteresowanych odsyłam do wykładów nr 14 i 15 na stronie `http://students.mimuw.edu.pl/SO`.

11.1 Utrzymywanie spójności dostępów do pamięci

11.1.1 Źródła kłopotów

- lokalne pamięci podręczne
- buforowanie operacji pisania

11.1.2 Modele spójności - podobnie jak na BD

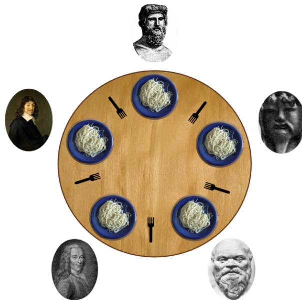
- Spójność ścisła
- Spójność sekwencyjna
- Spójność słaba, na wyjściu, na wejściu

11.1.3 Implementacja

- Algorytm scentralizowanego zarządcy
- Dynamiczny algorytm zarządcy rozproszonego
- Jawne blokowanie zasobów
- Linda

12 Problemy z współbieżnością

12.1 Przykre wspomnienia z PW



12.1.1 Filozofia

- Problem pięciu filozofów (raczej rzadko spotykany w SO)
- Wyłączność dostępu do zasobów
- Problem czytelników i pisarzy
- Zapewnienie „normalnej“ semantyki programów na wielu procesorach

12.1.2 „Normalna“ semantyka

x = 5;
x = x + 1;
if (x!=6)printf(”Spokojnie, to tylko PW”);

12.2 Rozwiązania w linuxie

12.2.1 Znane nam rozwiązania w linuxie

Dokładny opis wszystkich tych mechanizmów był podany na ćwiczeniach więc tu je tylko wymienię.

- `atomic_t`, `atomic_read(v)`,`atomic_set(v, i)`,`atomic_dec_and_test(v)` ...
- blokowanie przerwań
- `spinlock_t`
- `rw_lock_t` - spinlock dla problemu czytelników/pisarzy
- semafory(< 2.6.24), mutexy(> 2.6.16) - „my“(w sensie laboratorium) jesteśmy akurat pomiędzy

12.3 Rozwiązania w Windows

12.3.1 Porady dla twórców sterowników dla Windows

Korzystałem z dokumentu dostępnego przez stronę :
`http://www.microsoft.com/whdc/driver/kernel/MP_issues.mspx`
Jak się okazuje, w Windows mamy analogiczne mechanizmy jak w linuxie, oczywiście pod nieco innymi nazwami.

- Różne poziomy `IRQL`

  `IRQL` - „poziom przerwania”. Opisuje jak istotny (z punktu widzenia SO jest kod w danym momencie wykonywany. Im wyższy tym więcej przerwań jest blokowanych tak aby np scheduler nie przeszkadzał, przerywając kluczowe operacje. Poza tym ograniczeniem (tzn swego rodzaju liniowym porządkiem na przerwaniach) jądro Windows jest w pełni wywłaszczalne.

- słowo kluczowe `volatile` - mało skuteczne, semantyka nie jest dokładnie wyspecyfikowana

  `volatile` jest modyfikatorem dla zmiennej który mówi kompilatorowi, że może ona się zmieniać niezależnie od jego wiedzy i za każdym razem powinna być od nowa wczytywana z pamięci. Przy czym specyfikacja C nie mówi dokładnie czego kompilatorowi nie wolno itp..

- operacje atomowe, analogicznie jak w linuxie
- `InterlockedXxx`(szybkie) i `ExInterlockedXxx`

  `InterlockedXxx` - krótkie operacje arytmetyczne i podobne, albo wykonywane wewnątrz spinlocka albo poprzez operacje atomowe. Gwarantowane jest, że wykonają się atomowo na każdej architekturze.

  `ExInterlockedXxx` - bardziej skomplikowane operacje, np na listach, chronione spinlockami
- jawne Spinlocki
- `Mutex, fast mutex, and executive resource`

  Executive resource - mutex dla problemu czytelników/pisarzy

13 Scheduler

13.1 Scheduler w linuxie - stary i nowy

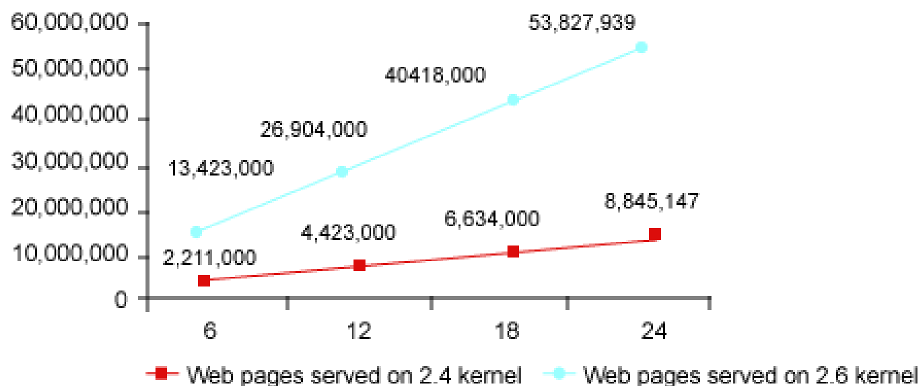
13.1.1 Stary(jądra 2.4.x)

- analogiczny jak dla jednego procesora
- jest jedna globalna kolejka procesów gotowych(wtedy był jeszcze Big Kernel Lock)
- wolny procesor bierze najlepszy proces z kolejki
- brak jakichkolwiek rozwiązań ustalających optymalne rozłożenie procesów

13.1.2 Nowy(jądra 2.6.x)

- osobna kolejka dla kazdego procesora
- Load balancing co 200ms(albo 1ms jeśli procesor wykonuje pusty kod `idle`)
- jak dobierac procesy miedzy procesorami?
- pamiętajmy o potrzebie synchronizacji cache

13.1.3 Praktyczne porównanie



Serwer apache postawiony na czterordzeniowej maszynie i testowany na liczbę wyświetlonych stron dla jądra 2.4 i 2.6 . Widać, że sama zmiana jądra daje kilkukrotne przyspieszenie.

13.2 Load balancing

13.2.1 Procedura wyrównywania obciążenia procesorów

Każdy procesor wykonuje ją niezależnie co 0.2 s.

- wyszukanie najbardziej zatłoczonej kolejki
- wybór kolejki expired lub active
- wybór procesu o najmniejszej wartości priorytetu który nie jest wykonywany i nie ma danych w cache'u (taki proces łatwo przenieść)
- przeniesienie procesu
- i tak dalej aż te dwie kolejki nie bedą zbalansowane

14 Aplikacje które skutecznie wykorzystują wiele procesorów logicznych

14.1 Ogólne wymagania

14.1.1 Prawo Amdahl'a

$$W = \frac{1}{(1-P)+\frac{P}{S}}$$

- W - czas wykonania programu z wieloma procesorami w stosunku do czasu na pojedynczym procesorze

- P - część programu którą można wykonać równolegle
- S - liczba procesorów logicznych do dyspozycji
- Zakładamy, że nasze procesory logiczne są takiej wydajności jak ten jeden pierwotny

Prawo to pokazuje, że nawet dla bardzo dużej liczby procesorów nie da się przyspieszyć programów powyżej granicy wyznaczonej przez tą część programu która musi być wykonana szeregowo. Z drugiej strony w dużej aplikacji praktycznie zawsze da się wydzielić część obliczeń do osobnego wątku.

14.1.2 Inne ograniczenia

- Dostęp do szyny danych
- Dostęp do pamięci
- Dostęp do struktur jądra
- Dostęp do dysku, sieci itd.
- W praktyce dobrze jeśli dla dwóch rdzeni mamy 60% szybsze wykonanie

14.2 Grupy aplikacji przystosowanych do wielowątkowości...

14.2.1 Aplikacje biznesowe

- serwery WWW i serwery aplikacji, np Apache,Tomcat, JBoss...
- SAPy
- Bazy danych

14.2.2 Naukowe

- SETI, fizyka cząstek elementarnych
- alg. genetyczne (AI), Blast

14.2.3 Inne

- Łamanie haseł, zwłaszcza brute-force
- Maszyny wirtualne, najlepiej po prostu „oddac“ jeden rdzen na maszyne wirtualna
- ...

14.3 ... i inne typy aplikacji które też chcą skorzystać z nowych możliwości

14.3.1 Jeszcze inne :)...

- Bioshock, UT3 , Call Of Duty 4, Company of Heroes ,Crysis ....

Zgodnie, z tym co napisałem wcześniej, jeśli mamy dużą aplikację i chcemy ją przyspieszyć to możemy wydzielić jej część do wątków pobocznych. Utrudnia to pisanie aplikacji i debugowanie ale pozwala wykorzystać wiele rdzeni czy procesorów. Silnik do Unreal Tournament 3 został dostosowany do wielordzeniowości i gry które na jego podstawie zostały napisane (i wiele innych) wykorzystują całą moc nowoczesnych CPU.

15 Podział architektur ze względu na liczbę potoków

15.1 SISD,SIMD,MISD,MIMD

Powtórka z AKiPN.

15.1.1 Podział architektur ze względu na liczbę potoków

- SISD(Single Instruction, Single Data) - komputery z jednym procesorem logicznym
- MISD(Multiple Instruction, Single Data) - rzadko spotykane (podobno w NASA używają takiej architektury jako, że potrzebują mieć pewność, że nie nastąpiła pomyłka)
- SIMD(Single Instruction, Multiple Data) - jednostki wektorowe, GPU
- MIMD(Multiple Instruction, Multiple Data) - komputery z wieloma procesorami, systemy rozproszone

15.1.2 Obecnie ten podział traci na znaczeniu

- Prawie każdy komputer jest MIMD
- Prawie każdy komputer zawiera jakieś elementy typu SIMD (np GPU)

15.2 Systemy rozproszone

Tak na koniec. Ciąg dalszy na SO, wykłady 12-15.

15.2.1 Systemy rozproszone

- Sieciowy system operacyjny - współdzielone pliki
- Rozproszony system operacyjny (wiele komputerów) - komunikaty
- Rozproszony system operacyjny (wieloprocesorowy) - współdzielona pamięć
- Ciąg dalszy nastąpi...