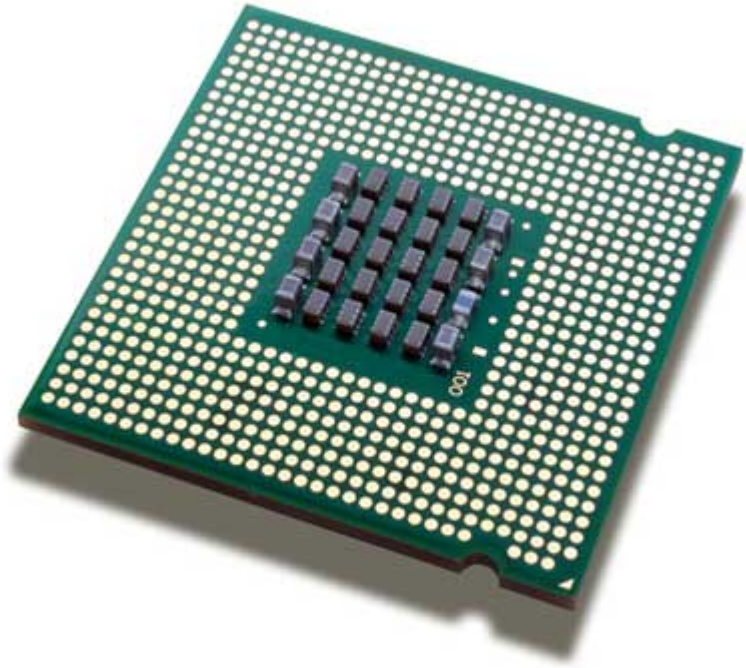
Wpływ architektury procesora na system operacyjny



5 listopada 2008

Radomir Mastalerz
Kamil Nowosad
Krzysztof Sachanowicz

AGENDA

- Klasyfikacja procesorów (32-bitowe a 64-bitowe, CISC a RISC itp).
- Potokowanie (pipelining).
- Klasyczne potokowanie procesora RISC.
- Wielordzeniowość.
- Wsparcie dla wielordzeniowości w Linuksie.
- Problemy związane z wielordzeniowością.
- Czy 64-bitowe adresowanie to dzisiaj konieczność?
- Przykłady 64 bitowych edycji systemów - dystrybucje Linuksowe (m. in. Ubuntu, openSuse), Windows XP Professional, Vista.

KLASYFIKACJA PROCESORÓW

32-bitowe a 64-bitowe

1. Co to znaczy, że procesor jest n-bitowy?

Tak naprawdę chodzi tu o kilka rzeczy:

Rozmiar rejestrów. N-bitowy procesor może przechowywać w rejestrach n-bitowe wartości całkowite, jego układ arytmetyczny obsługuje działania na wartościach tego rozmiaru itd.

Rozmiar szyny adresowej. N-bitowa szyna umożliwia zaadresowanie 2^N miejsc w pamięci.

Rozmiar szyny danych (o ile jest wydzielona). Jest to liczba bitów, które mogą być jednocześnie przesłane po magistrali.

Procesor nazywamy N-bitowym, kiedy spełnia pierwsze kryterium. Przykładowo Pentium Pro był 32-bitowy, ale używał 36-bitowej szyny adresowej i 64-bitowej szyny danych.

Co nam daje 64-bitowość?
Ogromną **przestrzeń adresową** – mamy do zaadresowania 2^{64} bajtów. To więcej, niż pojemność całej wyprodukowanej dotychczas pamięci. Zatem zapewne wystarczy nam do większości zastosowań i nie musimy się nią przejmować. Można też wtedy w pełni wykorzystać potencjał **plików mapowanych do pamięci** (mmap) – dzięki tak dużej przestrzeni adresowej, nie ma problemu z mapowaniem np. całej płyty DVD.

Wykonywanie kilku operacji arytmetycznych w jednym cyklu w takim dużym, 64-bitowym rejestrze. Można np. wrzucić do takiego rejestru dwie 32-bitowe liczby (cztery 16-bitowe itd.) i wykonać na każdej jakąś operację.

Po co nam operacje na tak dużych liczbach? Przykładowo w kryptografii. Często korzystamy z szyfrowanych połączeń internetowych, szyfrowanych systemów plików itp. W tego typu zastosowaniach wydajność może mieć spore znaczenie.

Poniższa tabelka przedstawia **porównanie kosztu (instrukcje i zasoby) wykonania instrukcji** wyciągania z pamięci, dodawania i zapisu do pamięci liczb 64-bitowych na architekturze 32- i 64-bitowej.

OPERACJA (2 liczby)	KOSZT	
	32-bit	64-bit
Ładowanie	4 rejestry 4 instrukcje	2 rejestry 2 instrukcje
Dodawanie	2 instrukcje	1 instruckja
Zapis	4 rejestry 4 instrukcje	2 rejestry 2 instrukcje
RAZEM	10 instrukcji, 4 rejestry i pole przeniesienia	5 instrukcji i 2 rejestry

(źródło: https://www.ibm.com/developerworks/library/pa-microdesign/)

Więcej na temat architektur 64-bitowych w dalszej części prezentacji.

2. Big-endian a little-endian

Są to dwa sposoby zapisu liczb o długości większej niż jeden bajt (ang. endianness).

W big-endian najbardziej znaczący bajt jest zapisywany **na początku**.
Wartość 0x01020304 będzie zapisana w następującej kolejności:

...	0x01	0x02	0x03	0x04	...

Tak jest np. w procesorach Motoroli.

W little-endian na początku jest zapisywany **najmniej znaczący** bajt, czyli dla wartości 0x01020304:

...	0x04	0x03	0x02	0x01	...

Tak jest np. w procesorach intelowskich.

Niektóre procesory, takie jak np. **PowerPC** potrafią działać **zarówno** w trybie **BE, jak i LE**.

Kolejność bajtów nie ma wpływu na wydajność, ani projekt systemu, jednak należy zawsze brać to pod uwagę podczas pisania np. pewnych „niskopoziomowych" operacji – np. na bitach. Ma to również znaczenie podczas **przesyłania danych przez internet** – przyjęło się, że dane przesyła się tam w porządku **big-endian** (standard **POSIX** daje nam gotowe funkcje konwertujące dane z porządku obowiązującego na komputerze klienta na sieciowy i odwrotnie - informacje na ten temat są dostępne na stronie manuala: **BYTEORDER(3)**).

3. CISC i RISC

CISC	RISC
Complex Instruction Set Computers	**Reduced** Instruction Set Computers
Złożone instrukcje procesora Często zajmujące wiele cykli zegara. Sprzęt daje nam szybkie i sprawne narzędzia do wykonywania pewnych czynności, jednak kosztem ogólnej szybkości przetwarzania instrukcji.	**Proste instrukcje procesora** Kod maszynowy jest dłuższy, ale daje większe pole do popisu kompilatorowi, (ew. programiście – jeżeli piszemy w asemblerze), wykonujące się zazwyczaj w jednym cyklu.
Niewyrównane adresowanie	**Tylko wyrównane adresowanie** Nie można trzymać np. czterobajtowej liczby całkowitej zaczynającej się w adresie niepodzielnym przez 4
Duża liczba trybów adresowania Możemy wykonywać wiele operacji bezpośrednio operując adresami argumentów w pamięci - bez potrzeby jawnego pobierania tych danych do rejestrów.	**Mała liczba trybów adresowania** Najpierw wrzucamy dane do rejstrów, a potem na nich wykonujemy operacje.
Mało rejestrów Np. w x86 8 rejestrów ogólnego przeznaczenia (np. typowo: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP)	**Dużo rejestrów** Np. MIPS ma 32 rejestry.
Przykłady procesorów: x86 (386, 486, Pentium, Pentium II, Athlon, ...) VAX PDP-11 Motorola 68000	**Przykłady procesorów:** procesory zastosowane w urządzeniach Apple'a (iPody, iPhone'y) IBM PowerPC SPARC Alpha

4. Jedno- i wielordzeniowe (o tym później)

PRZETWARZANIE POTOKOWE

(ang. pipelining)

Ogólny opis

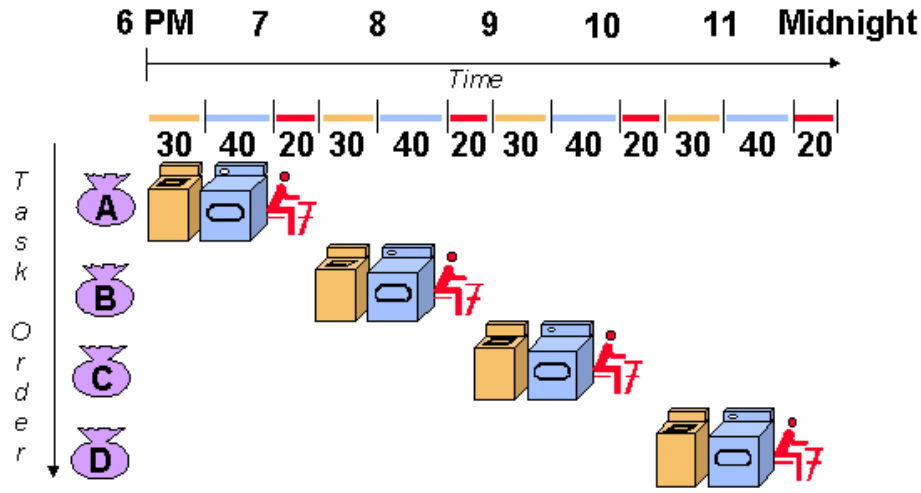
1. Wprowadzenie

Jest to kolejne z usprawnień wprowadzonych do procesora. Zauważono, że **pewne stadia przetwarzania instrukcji** (takie jak wyciąganie rozkazu z pamięci, dekodowanie go itd.) **można wykonywać równocześnie** dla różnych instrukcji (czyli podczas gdy oblicza się np. wynik operacji arytmetycznej, można już dekodować kolejny rozkaz).

2. Pralnia

Mechanizm **potokowania** można intuicyjnie wyjaśnić na przykładzie **pralni**.

Powiedzmy, że mamy **cztery porcje prania**. Każdą musimy **wyprać** (co trwa 30'), **wysuszyć** (40') i **poskładać** (20'). Oto dwa przykłady wykonania tej pracy:

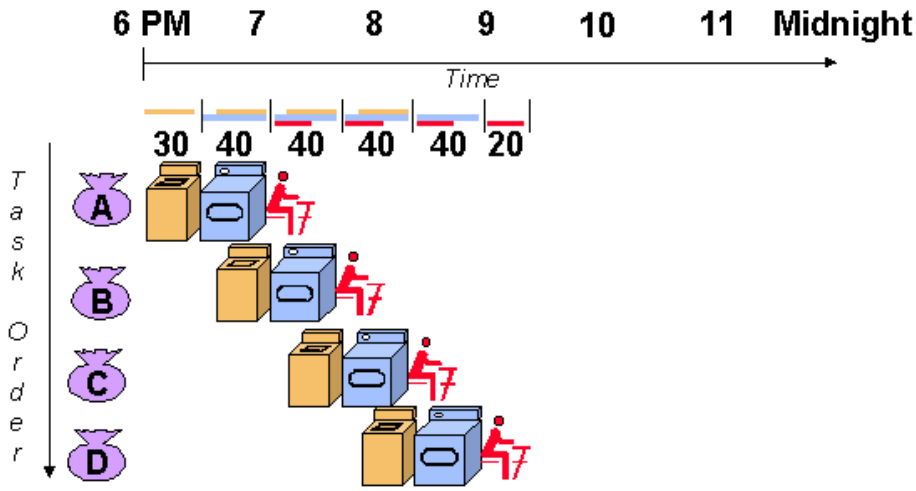


(źródło: http://www.ece.arizona.edu/~ece462/Lec03-pipe/)

Czyli możemy po prostu wykonać **kolejno** każdą z tych operacji **dla każdej porcji** prania.
Jak widać, **sumaryczny czas** wynosi **6h**.

3. Sprytne podejście:

Można jednak do tego zadania podejść inaczej:



(źródło: http://www.ece.arizona.edu/~ece462/Lec03-pipe/)

W momencie, kiedy **pierwsza** porcja prania **się wypierze,** możemy wrzucić ją **do suszarki** i jednocześnie wrzucić **kolejną do pralki**. Czyli, w odróżnieniu od pierwszego przykładu, **przez większość czasu działają oba urządzenia** (**i osoba** składająca pranie). W związku z tym, **wszystko** wykonuje się w **krótszym czasie**: dzięki takiemu podziałowi trwa to **3,5h**.

W przypadku procesora, **porcje prania** odpowiadają **instrukcjom**, a **pralka**, **suszarka** i **„składacz”** odpowiadają **podjednostkom** procesora odpowiedzialnym za poszczególne stadia przetwarzania instrukcji.

Klasyczne potokowanie RISC

1. Omówienie

Teraz przystąpimy do omówienia potokowania w procesorze na przykładzie **klasycznego potokowania RISC**. Było ono **zaimplementowane** w procesorach MIPS, SPARC, Motorola 88000, a potem też DLX.

W tym modelu, **przetworzenie** każdego **rozkazu** dzieli się na **5 części**:

Pobranie instrukcji z pamięci (ang. instruction fetch, **IF**)
Dekodowanie i pobranie wartości z rejestrów (ang. instruction decode, **ID**)
Wykonanie (ang. execute, **EX**)
Dostęp do pamięci (ang. memory access, **MEM**)
Zapis do rejestrów (ang. write back, **WB**)

2. Problemy pojawiające się w trakcie przetwarzania potokowego

Zależność danych: występuje, kiedy instrukcja wymaga danych, które są wynikiem instrukcji poprzedzającej, np.:

```
add $r3, $r2, $r1
add $r5, $r4, $r3
```

W tym przypadku pierwsza instrukcja **zapisuje wynik** dodawania zawartości rejestrów **$r1** i **$r2** do rejestru **$r3**, a druga **potrzebuje** tego **wyniku** do swojego obliczenia. W takiej sytuacji część **potoku** za pierwszą instrukcją zostaje **wstrzymana**.

Instrukcje warunkowe: występuje, kiedy w zależności od wyniku instrukcji, następuje instrukcja skoku, np.:

```
loop :
add $r3, $r2, $r1
sub $r6, $r5, $r4
beq $r3, $r6, loop
```
(w tej sytuacji, skok następuje wtedy, gdy wartości rejestrów $r3 i $r6 są równe)

Procesor nie jest w stanie **odgadnąć**, czy nastąpi **skok**, dopóki wartości obu tych rejestrów nie zostaną zapisane.

3. Sposoby na radzenie sobie z tymi problemami

Rozwiązanie pierwszego problemu mogłoby wyglądać następująco: załóżmy, że **przed lub po** parze **zależnych instrukcji** występują takie, które są od nich **niezależne**. Wtedy można **między** naszą **parę wstawić** takie instrukcje. Dzięki niezależności, **wynik wykonania** będzie **identyczny**, jednak **unikniemy opóźnień**.

```
add $r3, $r2, $r1
```

Inne instrukcje...

add $r5, $r4, $r3

Problemy związane ze **skokami** rozwiązuje się bardzo często za pomocą jakiegoś **mechanizmu przewidywania**, w którą **gałąź** skoczymy (ang. branch prediction) i **dodawania do potoku** kodu z tej właśnie gałęzi. **Jeżeli** wybór okaże się **błędny**, trzeba **wyrzucić** wszystkie instrukcje z tej gałęzi i zacząć **„napełniać potok”** instrukcjami z **drugiej**.

Jak działają **metody przewidywania skoków**? Niektóre opierają się na pewnych **szacunkach „z życia”** (danych statystycznych), np. takich, że większość **krótkich skoków do tyłu** to skoki charakterystyczne dla **pętli**, a takie w ok. **90%** przypadków **są wykonywane** (zwykle pętla obraca się więcej niż jeden raz) – dlatego wtedy procesor może zdecydować, że **skacze**. W przeciwnym przypadku – **skoków do przodu**, jest to już ok. **50%** przypadków. Wtedy możemy się zdecydować **nie skoczyć**.

Istnieją dużo bardziej **wyszukane** metody przewidywania skoków (i zapewniają one statystycznie **dużą trafność**), jednak nie będziemy ich tutaj omawiać.

Sposoby na optymalizację mechanizmu potokowania

1. Superpotokowanie

Polega ono na **dodaniu dodatkowych stadiów** (kroków) do potoku. Dzięki temu, każdy **krok jest krótszy**, a więc cały **strumień** instrukcji **wykonuje się szybciej**. **Nowe kroki** powstają poprzez **podział** na części tych już istniejących. Na przykład, procesor MIPS R4000 ma **8 kroków** w potoku i dwa z dodatkowych powstały poprzez podział kroku **pobierania instrukcji** i kroku **zapisu do pamięci.**

Jak to wygląda w innych procesorach:
Oto przykładowy diagram przedstawiający potok w procesorze **Pentium 4:**

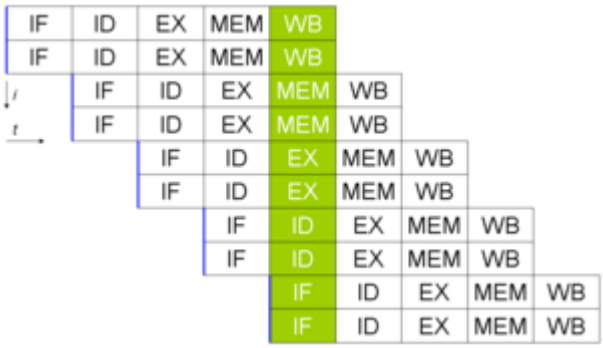
(źródło: wikipedia)

Jak łatwo policzyć, składa się z **20 kroków**.

2. Potokowanie superskalarne

Polega na tym, że **potok jest rozdzielony** na kilka **równoległych części**. Przykładowo, w niektórych procesorach RISC są **oddzielne moduły** do **obliczeń całkowitoliczbowych** i inne dla **zmiennoprzecinkowych**.

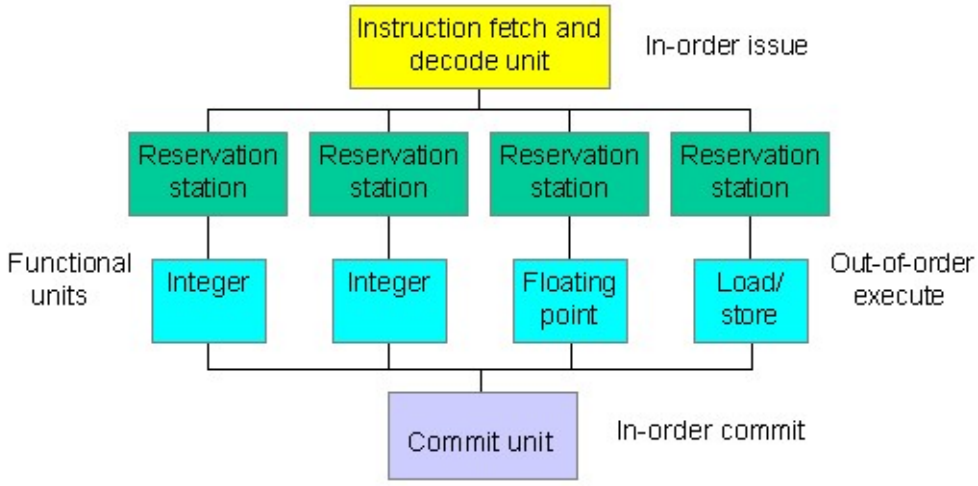
Potok można też rozdzielić w całości, co obrazuje poniższy diagram:



(źródło: wikipedia)

Jak widać w takiej sytuacji, jeżeli nie ma zależności danych, **w każdym cyklu mogą się zakończyć dwie instrukcje**.

Odmianą potokowania superskalarnego jest **potokowanie dynamiczne**.
Jest ono zobrazowane na poniższym diagramie:



(źródło: cse.stanford.edu/class/sophomore-college/projects-00/risc/pipelining/index.html)

Instrukcje wchodzące **do potoku**, przechodzą najpierw kolejno przez **jednostkę pobierającą i dekodującą**. Potem są wrzucane do **jednostek funkcjonalnych**, w których wykonują się **równolegle**, a następnie trafiają (znów **kolejno**) do **jednostki zatwierdzającej**. Dzięki temu, może się **jednocześnie wykonywać wiele instrukcji**, a jeżeli jakaś **musi poczekać** na wynik instrukcji wykonywanej wcześniej, to **blokuje jedynie swoją** jednostkę.

3. Podsumowanie

Potokowanie wpływa znacznie na czas wykonywania instrukcji, a co za tym idzie na wydajność. Dlatego też, **warto mieć je na uwadze** podczas **programowania niskopoziomowego**, zwłaszcza fragmentów **krytycznych dla wydajności** – czyli np. podczas tworzenia elementów **systemu operacyjnego**.

4. Co więcej

Ciekawe **artykuły**, **specyfikacje** i **zbiór odnośników** do stron, mówiących o programowaniu w

asemblerze, technikach optymalizacji i innych aspektach programowania niskopoziomowego przydatnych młodym **kernel hackerom** można znaleźć na stronie:

http://www.agner.org/optimize/

Wielordzeniowość (multi-core)

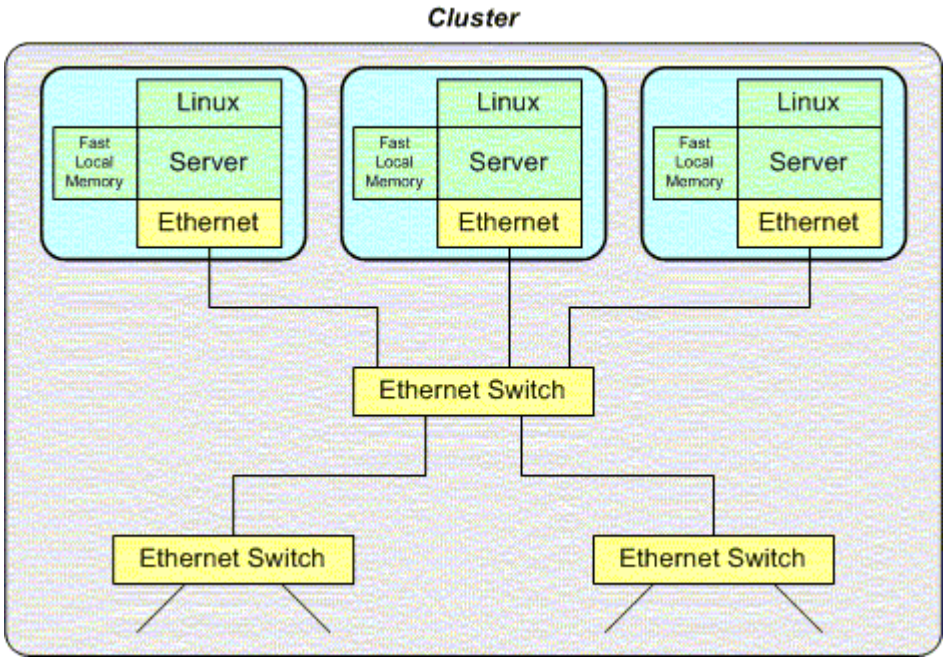
Wielordzeniowość wywodzi się z pojęcia architektury **SMP (*Symmetric Multiprocessing*).** Występuje ona, gdy dwa lub większa ilość identycznych pocesorów może komunikować się ze sobą za pomocą pamięci dzielonej. Każdy procesor ma równe uprawnienia w dostępie do pamięci. Występują dwa rodzaje SMP.

Loosely-coupled SMP = cluster

Wczesne systemy SMP oparte na Lunuksie były luźno związanymi systemami – procesory znajdowły się w samodzielnych komputerach, a komputery były połączone bardzo szybkim łączem (np. 10G Ethernet, światłowód). Taką architekturę nazywamy teraz klastrem.

Figure 3. Architektua klastra

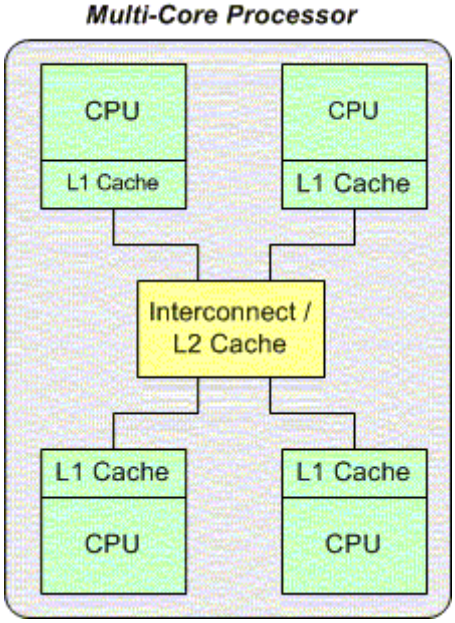
(źródło: http://www.ibm.com/developerworks/library/l-linux-smp/)



Tightly-coupled multiprocessing = CMP = multi-core

CMP (chip-level multiprocessing) odnosi się do właściwej wielordzeniowości. Jest to właściwie SMP w postaci klastra uproszczone i zeskalowane do poziomu płytki drukowanej

Figure 4. Tightly-coupled multiprocessing architecture



Jednostki obliczeniowe CPU są połączone wspólną magistralą do pamięci dzielonej cache poziomu 2. Każde CPU ma swoją własną pamięci cache poziomu 1. Bardzo małe fizyczne odległości między jednostkami obliczeniowymi pozwalają na zminimalizowanie czasów dostępu do pamięci. Taka architektura świetnie nadaje się do aplikacji wielowątkowych, które są rozdzielane na dostępne rdzenie. Taką własność nazywami **TLP (Thread-level parallelism)**. Obecnie w powszechnym zastosowaniu są procesory 2- i 4- rdzeniowe, ale są też serwery o nawet 100 jednostkach obliczeniowych.

Cechy procesorów wielordzeniowych:

1. Duża większa reakcyjność systemu. Program antywirusowy może działać na jednym rdzeniu i pozwalać użytkownikowi na komfortową pracę z innymi aplikacjami.
2. Procesor o dwóch rdzeniach 1GHz wytwarza znacznie mniej ciepła niż jeden rdzeń o taktowaniu 2 GHz (ilość wydzielanego ciepła nie rośnie liniowo ze wzrostem taktowania).
3. Mają dużo mniejszy narzut na komunikację między jednostkami obliczeniowymi niż systemy w architekturze klastra.

Table 1. Niektóre rocesory wielordzeniowe obłsugiwane przez Linuksa:

IBM	POWER4	SMP, dual CPU
IBM	POWER5	SMP, dual CPU, four simultaneous threads
AMD	AMD X2	SMP, dual CPU
Intel®	Xeon	SMP, dual or quad CPU
Intel	Core2 Duo	SMP, dual CPU
ARM	MPCore	SMP, up to four CPUs
IBM	Xenon	SMP, three Power PC CPUs
IBM	Cell Processor	Asymmetric multiprocessing (ASMP), nine CPUs

Intel HT (HyperThreading) jest czasami mylony z wielordzeniowością. Jest to system Intela polegający na wsparciu wielowątkowości. Jeden rdzeń procesora logicznie udaje dwa rdzenie – system operacyjny wybiera naraz dwa wątki do wykonania. Wątki te wykonują się na jednym rdzeniu w przeplocie, natomiast przełączanie między tymi wątkami jest bardzo szybkie (wspomagane sprzętowo) i powoduje zauważalny wzrost wydajności. Nie jest do techonologia, którą dzisiaj nazywamy wielordzeniowością.

Wielordzeniowość a jądro Linuksa

Procesor wielordzeniowy dało się odpalić już na jądrze w wersji 2.0 – jednak jego obsługa pozostawiała wiele do życzenia. Dopiero wersja 2.6 w pełni wykorzystuje wszystkie zalety wielordzeniowości – głównie dzięki poprawionemu schedulerowi.

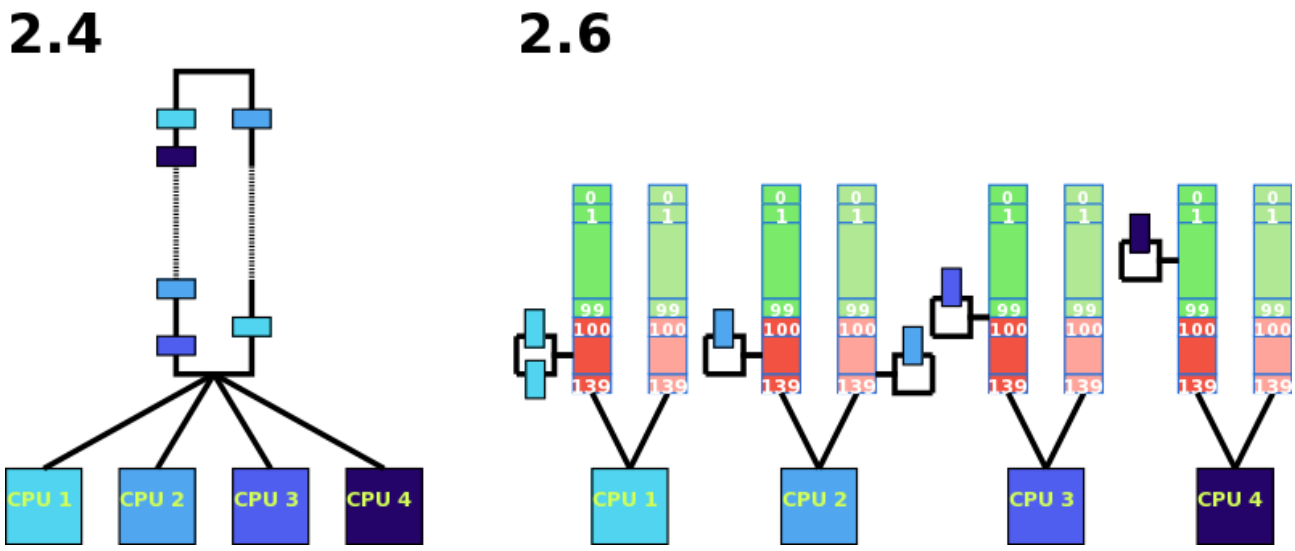
Podczas wykonywania kodu jądra przez jeden rdzeń następuje tzn. **„big lock”** - blokada wszystkich systemowych struktur danych. Pozostałe rdzenie w tym czasie nie mogą wykonywać kodu jądra. Taka sytuacja często powoduje znacznie zmniejszenie przepustowości systemu. Są plany podziały jądra na sektory, do których możliwy będzie równoległy dostęp.

Skalowalność, porównanie z jądrem 2.4

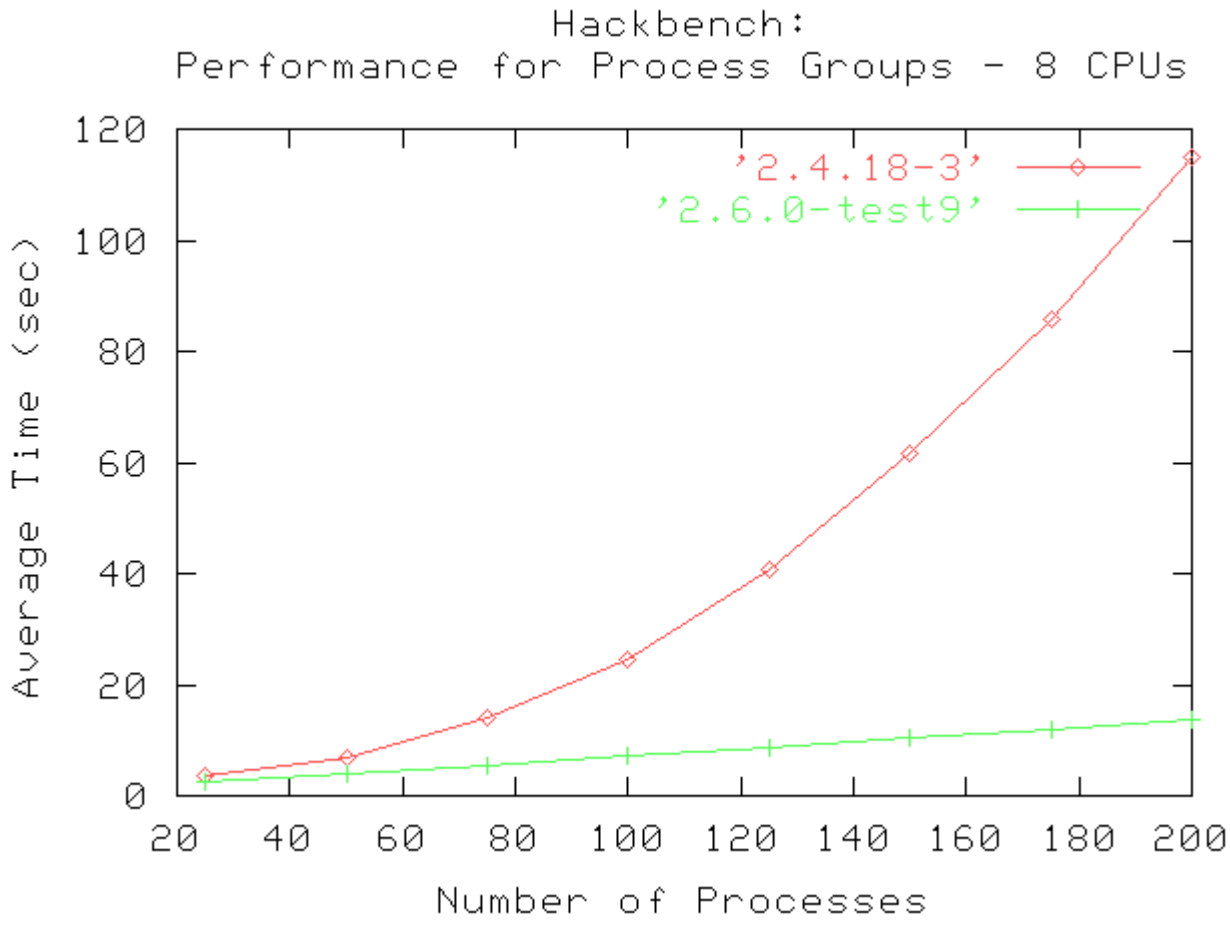
(źródło: notatki JMD)

W jądrze **2.4** była jedna **wspólna kolejka wszystkich procesów**. Wybór przez procesor kolejnego procesu wiązał się z założeniem blokady na kolejkę. Inne procesory musiały w tym czasie czekać, a algorytm wyboru kolejnego zadania działał w czasie liniowym. Przy większej liczbie procesorów, czas oczekiwania wydłużał się. Po zakończeniu epoki, jeden procesor wykonywał przeliczanie priorytetów, a inne procesory czekały na zakończenie tej operacji.

W jądrze **2.6** każdy **procesor ma własną kolejkę procesów**, a globalne szeregowanie polega na równoważeniu obciążenia procesorów (co 200 ms sprawdza się czy procesory są równomiernie obciążone i jeśli nie, to przenosi się procesy między kolejkami). Interakcja między procesorami jest niewielka, dlatego skalowalność jest dobra. Polepsza się wykorzystanie pamięci podręcznej procesora.



Na kolejnych diagramach pokazano różnice w zachowaniu algorytmu szeregowania w jądrze 2.4 i 2.6.



Problem dzielenia pamięci przez wątki

Wątek zakłada lock'a na zmiennej za każdym razem, gdy chce ją zmienić. Jeżeli jakaś zmiena wykorzystywana jest tylko na jednym rdzeniu, nie musimy jej za każdym razem blokować – zakładanie blokad powoduje nieuzasadniony narzut czasowy na wykonanie programy. Potrzeba istnienia zmiennych, które nie są dzielone przez rdzenie, została zrealizowana w jądrze 2.6. Możemy deklarować zmienne, które są dostępne tylko przez poszczególne rdzenie. Wykorzystywane jest do tego makro **`DEFINE_PER_CPU.`**

Poniższy przykład pochodzący z ./arch/i386/kernel/smp.c definiuje stan na każdym z rdzeni.

```
DEFINE_PER_CPU( int, cpu_state ) = { 0 };
```

Makro dla każdego rdzenia tworzy oddzielną instancją zmiennej. Do konkretnej instancji zmiennej odwołujemy się przy użyciu funkcji smp_processor_id(), która zwaraca numer rdzenia, na którym wykonuje się kod programu.

Przypisanie na zmienną cpu_state numer rdzenia, na którym wykonywany jest kod:

```
per_cpu( cpu_state, smp_processor_id() ) = CPU_ONLINE;
```

Żeby włączyć obsługą SMP kompilujemy jądro z opcją CONFIG_SMP. Nieużycie tej opcji spowoduje, że system będzie działał tylko na jednym rdzeniu.

CPU info

Czasami nie mamy pewności, czy system został poprawnie skonfigurowany. Informacje o aktywnych rdzeniach możemy odczytać z pliku */proc/cpuinfo*. Poniżej wynik komendy wylistowania pliku dla dwurdzeniowego procesora Xeon.

```
mtj@camus:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 15
model           : 6
model name      : Intel(R) Xeon(TM) CPU 3.73GHz
stepping        : 4
cpu MHz         : 3724.219
cache size      : 2048 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
pbe nx lm pni monitor ds_cpl est cid xtpr

bogomips        : 7389.18

...
```

```
processor       : 7
vendor_id       : GenuineIntel
cpu family      : 15
model           : 6
model name      : Intel(R) Xeon(TM) CPU 3.73GHz
stepping        : 4
cpu MHz         : 3724.219
cache size      : 2048 KB
physical id     : 1
siblings        : 4
core id         : 3
cpu cores       : 2
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
pbe nx lm pni monitor ds_cpl est cid xtpr

bogomips        : 7438.33
```

Obsługa SMP w Windows:

Jak to wygląda? Nie wiadomo. Brak materiałów na ten temat. Windows XP Proffesional i Vista obsługują procesory posiadające do 4 rdzeni (Linux teoretycznie do 16 rdzeni, jednak testy wykonywane były tylko dla procesorów maksymalnie 4-rdzeniowych).

Wykorzystanie wielordzeniowości

Na razie bardzo słabe, brakuje aplikacji korzystających z wielordzeniowości. Wydaje się, że wielordzeniowość będzie miała bardzo szerokie zastosowanie, przede wszystkich do obliczeń, które dają się sensownie zrównoleglić Główne obszary zastosowań SMP:

- multimedia
- renderowanie obrazów
- gry (nowy engine do Half-Life jest projektowany pod procesory wielordzeniowe)
- Real Time Systems
- aplikacje do obliczeń naukowych
- CAD/MAD

Oprogramowania na procesory wielordzeniowe

Trudności w wykorzystaniu potencjału tkwiącego w procesorach wielordzeniowych ma pomóc pokonać projekt Centrów Badawczych ds. Równoległego Przetwarzania. Potentaci na rynku informatycznym chcą na niego przeznaczyć 20 milionów dolarów.

Zdaniem Microsoftu i Intela, które postanowiły zaangażować się w projekt **Centrów Badawczych ds. Równoległego Przetwarzania (CBRP)**, przyszłość komputerów należy właśnie do procesorów wielordzeniowych. Dzięki CBRP rozwojowi powinien ulec rynek aplikacji, tworzonych specjalnie na potrzeby procesorów wielordzeniowych.

Firmy Microsoft i Intel postanowiły przeznaczyć na wspomniany projekt łącznie 20 milionów dolarów. **Utworzone zostaną dwa centra - jedno na Uniwersytecie Kalifornijskim w Berkeley, drugie na Uniwersytecie Illinois w Urbana-Champaign**. Obie uczelnie będą miały także wkład własny w wysokości odpowiednio 8 i 7 milionów dolarów. Intel poinformował, że czas trwania projektu wyznaczony został na 5 lat.

Architektury 64-bitowe a systemy operacyjne

Trochę historii

64-bitowe CPU były obecne w superkomputerach od lat 60-tych, a w przypadku platform RISC-owych początek przypada na lata 90-te. Od 2003 roku pojawiły się w segmencie komputerów osobistych, w postaci architektur x86-64 i 64-bitowej PowerPC.

- 1961: superkomputer IBM 7030 Stretch, 64-bitowe słowa danych i 32- lub 64-bitowe słowa instrukcji.
- 1976: superkomputer Cray-1, słowa 64-bitowe.
- 1994: Intel ogłasza plany 64-bitowej architektury IA-64 (Itanium) jako następcy 32-bitowej IA-32.
- 1999: Intel udostępnia zestaw instrukcji architektury IA-64. AMD ujawnia swój zestaw 64-bitowych rozszerzeń do IA-32, nazwanych x86-64 (zmienione później na AMD64).
- 2003: AMD wprowadza procesory Opteron i Athlon 64, bazowane na architekturze AMD64. Wychodzą dystrybucje Linuksa oraz system FreeBSD wspierające AMD64. Intel oświadcza, że procesory Itanium będą jego jedynymi procesorami 64-bitowymi.
- 2004: Intel, reagując na sukces rynkowy AMD, ogłasza, że rozwijał odpowiednik AMD64 nazwany IA-32e (później zmienione na EM64T). Intel wypuszcza procesory Xeon i Pentium 4 wspierające nowe instrukcje.

Wady i zalety architektur 64-bitowych

- Systemy operacyjne rezerwują na swój użytek część przestrzeni adresowej procesu, zostawiając jedynie 2 do 3.8 GB dla użytkownika, nawet jeśli komputer ma 4GB RAMu.
- Pliki mapowane do pamięci są mniej użyteczne na platformach o małej (32-bitowej) przestrzeni adresowej.
- Niektóre programy, np. kryptograficzne mogą odnieść dużą korzyść z obecności rejestrów 64-bitowych (jeśli są skompilowane na 64 bity) i wykonywać się 3-5 razy szybciej na 64-bitach w porównaniu do 32-bitów.
- Największą wadą architektur 64-bitowych w porównaniu z 32-bitowymi jest to, że dane zajmują więcej miejsca w pamięci.
- Obecnie znaczna część oprogramowania komercyjnego jest skompilowana na 32-bity.

Przykładowa architektura 64-bitowa: x86-64

x86-64 jest nadzbiorem architektury x86. Procesory x86-64 mogą uruchamiać istniejące programy 32-bitowe z pełną szybkością, jak również wspierają programy korzystające z większej przestrzeni adresowej i innych nowych usprawnień.

Właściwości

- 64-bitowe rejestry: arytmetyka 64-bitowa i inne operacje na danych 64-bitowych wykonywane są bezpośrednio, elementy na stosie są 8-bajtowe.

- Dodatkowe rejestry: liczba rejestrów ogólnego przeznaczenia wzrosła z ośmiu (e{a,b,c,d}x, e{b,s}p, e{s,d}i) w x86 do 16-u. AMD64 ciągle ma jednak mniej rejestrów niż wiele procesorów RISC-owych (zazwyczaj 32–64 rejestry) oraz IA-64 (128 rejestrów).
- Większa wirtualna przestrzeń adresowa.
- Większa fizyczna przestrzeń adresowa.
- Adresowanie względem rejestru wskaźnika bieżącej instrukcji (RIP).
- Bit NX (No-Execute).
- Rezygnacja ze starych rozszerzeń: część rozszerzeń wspomagających zadania systemu operacyjnego nieużywanych w nowoczesnych systemach operacyjnych jest niedostępna na AMD64. Dotyczy to segmentacji, mechanizmu sprzętowego przełączania zadań (Task Register i Task State Segment), tryb emulacji procesora 8086 (Virtual-8086).

Tryby pracy

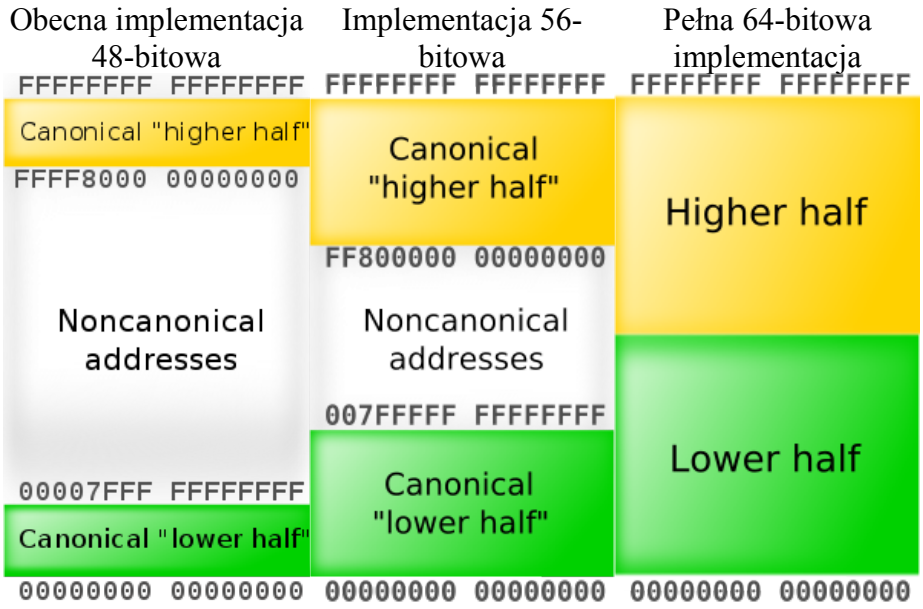
Tryb w jakim pracuje program		Wymagany OS	Konieczność rekompilacji programu	Domyślny rozmiar adresu	Domyślny rozmiar operandu	Rozszerzone rejestry	Typowy rozmiar GPR
Long	64-bitowy	64-bitowy	Tak	64	32	Tak	64
	Kompatybilności		Nie	32	32	Nie	32
				16	16		16
Legacy	Chroniony	16- lub 32-bitowy	Nie	32	32	Nie	32
				16	16		16
	Virtual 8086						
	Rzeczywisty	16-bitowy		16	16		16

(źródło: http://en.wikipedia.org/wiki/X86-64)

GPR – General Purpose Register

Szczegóły wirtualnej przestrzeni adresowej

Dostępne adresy wirtualne



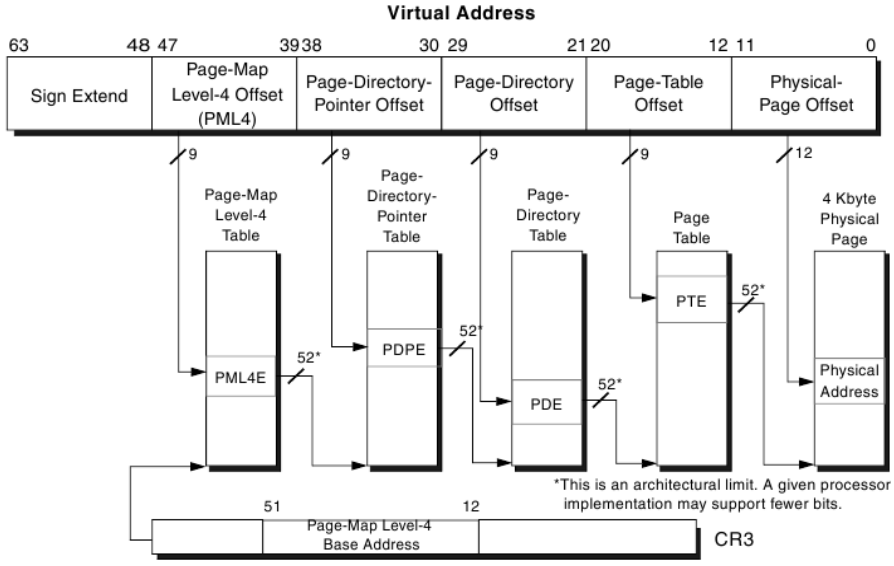
(źródło: http://en.wikipedia.org/wiki/X86-64)

- W obecnej implementacji jedynie 48 najmniej znaczących bitów adresu wirtualnego jest

faktycznie używanych w procesie translacji.

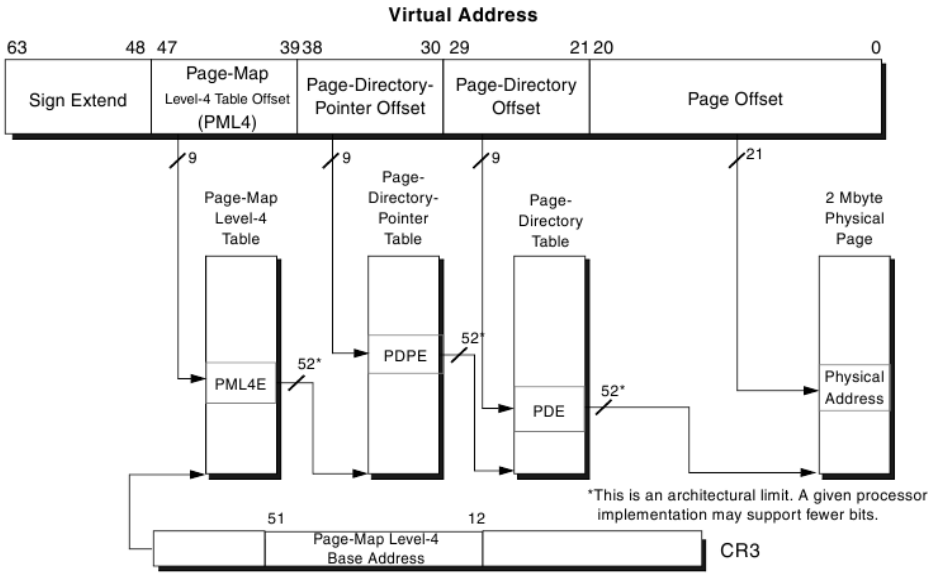
- Bity 48-63 adresu wirtualnego muszą być kopiami bitu 47 (jak bit znaku), lub procesor wygeneruje wyjątek.
- Adresy zgodne z tą regułą określane są jako kanoniczne. Zakres kanonicznych adresów obejmuje 0 - 00007FFF FFFFFFFF i FFFF8000 00000000 - FFFFFFFF FFFFFFFF, co daje 256 TB wirtualnej przestrzeni adresowej.

Stronicowanie



(źródło: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf)

strony 4kB, tryb long



(źródło: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf)

strony 2MB, tryb long

64-bitowe systemy operacyjne

Windows

64-bitowe edycje Windows XP Professional i Windows Server 2003 SP1 zostały wydane w marcu 2005, Windows Vista w styczniu 2007.

- 8 TB wirtualnej przestrzeni adresowej dla procesu.
- 8 TB wirtualnej przestrzeni adresowej dla jądra.
- Możliwość użycia do 128 GB (Windows XP) lub 1 TB (Windows Server 2003) pamięci RAM.
- Model LLP64: typy „int” i „long” są 32-bitowe, podczas gdy wskaźniki są 64-bitowe.
- Sterowniki muszą być 64-bitowe.
- Możliwość uruchamiania istniejących aplikacji 32-bitowych i bibliotek. 32-bitowy program może korzystać z 4 GB wirtualnej przestrzeni adresowej jeśli jest zlinkowany z opcją "large address aware".
- Nie da się uruchomić 16-bitowych aplikacji (Windows i DOS) pod Windows x86-64.

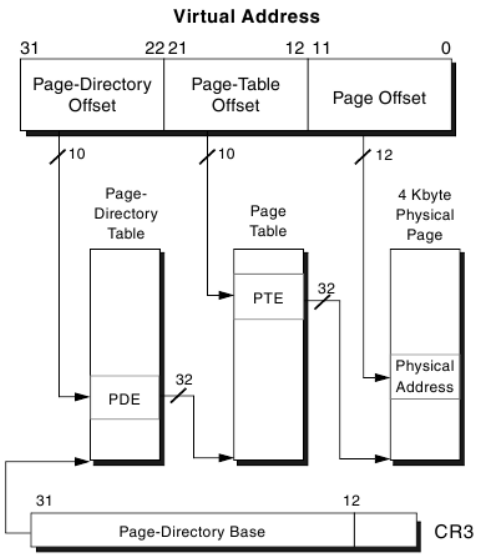
Linux

Linux był pierwszym systemem operacyjnym używającym trybu long architektury x86-64, począwszy od kernela 2.4, jeszcze przed wydaniem procesorów.

- Kompatybilność wsteczna z programami 32-bitowymi.
- Niektóre dystrybucje, jak SUSE, Mandriva, Debian, umieszczają wersje 32- i 64-bitowe na jednej płycie DVD. Inne są dostępne w dwóch wersjach skompilowanych oddzielnie dla obu architektur.
- Do 128 TB wirtualnej przestrzeni adresowej dla procesu.
- Może używać do 64 TB pamięci operacyjnej.
- Model LP64: „int” - 32 bity, „long” - 64 bity.

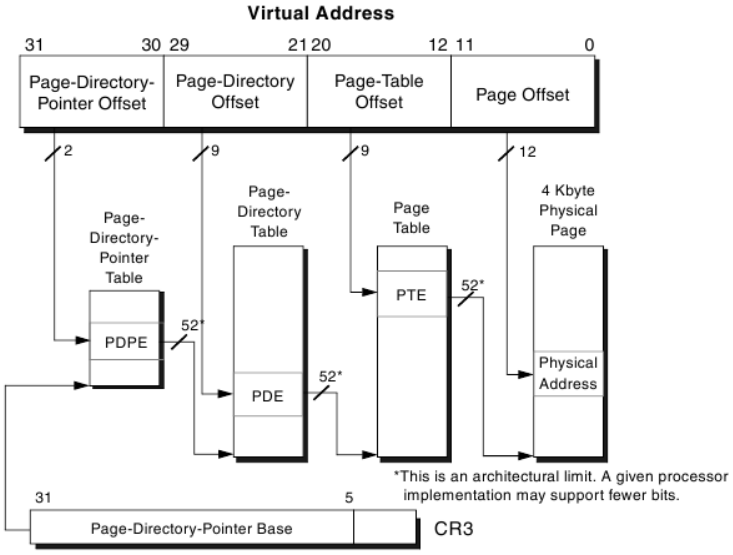
>4GB RAM, x86

Physical Address Extension



(źródło: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf)

strony 4kB, bez PAE



(źródło: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf)

strony 4kB, PAE

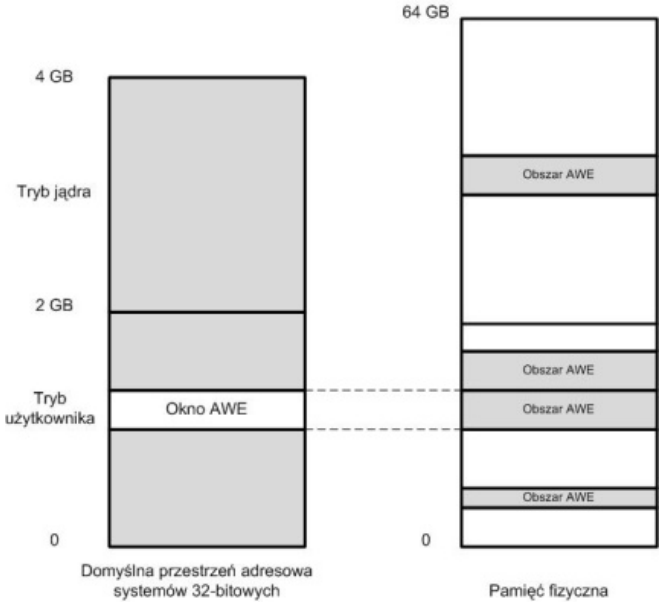
- Obecnie architektura x86 używa 36 bitów z 52 dostępnych.
- Bit NX.
- Linux kernel ma pełne wsparcie dla PAE począwszy od wersji 2.6, dając dostęp do 64 GB pamięci.
- PAE jest obsługiwane na następujących 32-bitowych wersjach Windows:

Wersja	Maksymalna pamięć fizyczna
Windows 2000 Advanced Server	8 GB
Windows 2000 Datacenter Server	32 GB
Windows XP	4 GB
Windows Server 2003 Enterprise Edition	32 GB
Windows Server 2003 R2 (or SP1) Enterprise Edition	64 GB
Windows Server 2003 Datacenter Edition	64 GB
Windows Server 2003 Standard Edition	4 GB
Windows Vista	4 GB
Windows Server 2008 Enterprise or Datacenter Edition	64 GB
Windows Server 2008 inne edycje	4 GB

(źródło: http://en.wikipedia.org/wiki/Physical_Address_Extension)

- Windows XP SP2 i późniejsze domyślnie pracują w trybie PAE celem używania bitu NX (na procesorach wyposażonych w NX).
- Biurkowe wersje Windows XP i Vista mają 4 GB ograniczenie na pamięć fizyczną ze względu na kompatybilność sterowników.

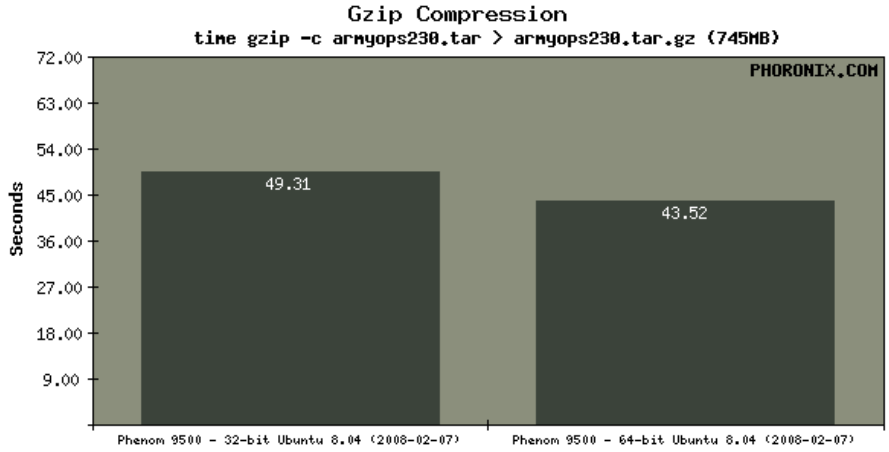
Address Windowing Extensions



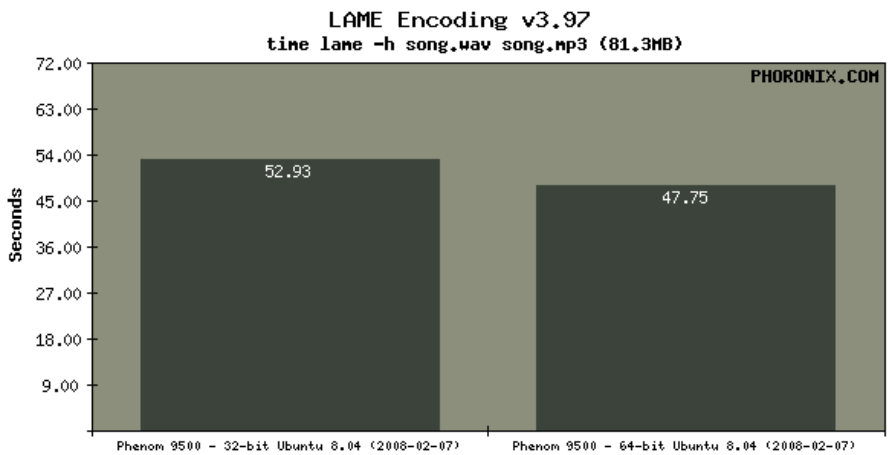
(źródło: http://www.microsoft.com/poland/technet/article/art0092_05.mspx)

- Windows API.
- Program musi mieć przywilej Lock Pages in Memory żeby używać AWE.
- Pod Linuksem można użyć mmap() z flagą MAP_ANONYMOUS.

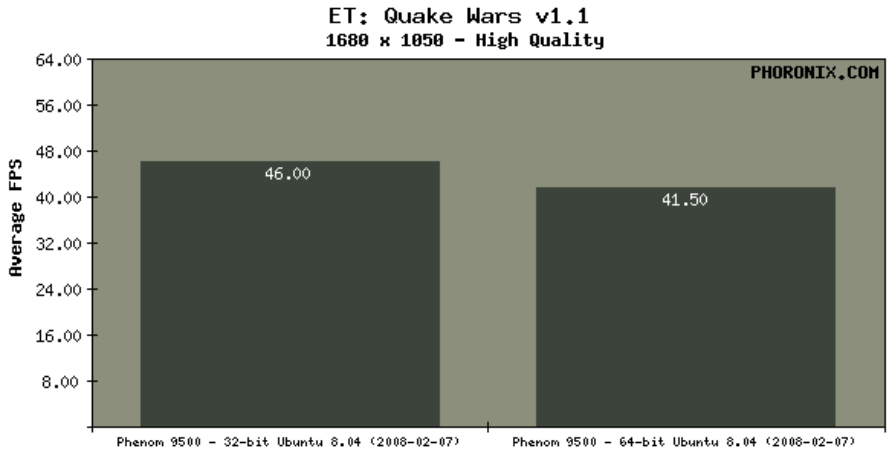
Benchmarki



(źródło: http://www.phoronix.com/scan.php?page=article&item=998&num=3)



(źródło: http://www.phoronix.com/scan.php?page=article&item=998&num=3)



(źródło: http://www.phoronix.com/scan.php?page=article&item=998&num=3)

Źródła:

- http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf
- http://www.microsoft.com/poland/technet/article/art0092_05.mspx
- http://en.wikipedia.org/wiki/64-bit
- http://en.wikipedia.org/wiki/Physical_Address_Extension
- http://en.wikipedia.org/wiki/X86-64
- http://www.phoronix.com/scan.php?page=article&item=998&num=3
- http://www.ece.arizona.edu/~ece462/Lec03-pipe/
- http://www.ibm.com/developerworks/library/l-linux-smp/
- Wikipedia
- notatki z przedmiotu 'architektura komputera i programowanie niskopoziomowe'
- notatki p. Janiny Mincer-Daszkiewicz z przedmiotu 'Systemy operacyjne'