AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Systemy operacyjne
Wykład 07

Wersja 2026

dr inż. Marek Wilkus **http://home.agh.edu.pl/~mwilkus**
Wydział Inżynierii Metali i Informatyki Przemysłowej
AGH Kraków

Na podstawie programu opracowanego przez dr inż. Krzysztofa Wilka

AGH

Pamięć wirtualna

- Pytanie: Czy proces rezerwuje pamięć i gospodaruje nią w sposób oszczędny?
  - Procesy często zawierają ogromne fragmenty kodu obsługujące sytuacje wyjątkowe.
  - Zadeklarowane tablice lub rozmiary list mają zwykle nadmiar przydzielonej pamięci.
  - Niektóre możliwości programu są niezwykle rzadko wykorzystywane.
- A gdyby tak w pamięci przebywała tylko ta część programu, która jest aktualnie wykonywana?

AGH

Pamięć wirtualna: zalety

- Program nie jest ograniczony wielkością pamięci fizycznej - można pisać ogromne programy bez specjalnych „sztuczek” programistycznych.
- Każdy program zajmuje w pamięci mniej miejsca niż program „kompletny”. Można więc w tym samym czasie wykonywać więcej zadań, polepszając wykorzystanie procesora.
- Maleje liczba operacji wejścia-wyjścia koniecznych do załadowania programów do pamięci oraz do realizacji wymiany - programy powinny więc wykonywać się szybciej.
- Nie ma konieczności robienia nakładek przy małej pamięci operacyjnej.

AGH

Pamięć wirtualna

- Pamięć wirtualna odseparowuje pamięć logiczną od jej fizycznej
  realizacji. Można ją zaimplementować jako:
  a) stronicowanie na żądanie
  b) segmentację na żądanie
- Procesy przebywają w pamięci pomocniczej (na dysku). Dla wykonania sprowadza się proces do pamięci, ale nie cały, tylko te strony, które są potrzebne (leniwa wymiana). Typowanie (zgadywanie) potrzebnych stron odbywa się podczas poprzedniego pobytu procesu w pamięci.
- Jeśli proces odwołuje się do strony, której nie ma w pamięci, to:
  - System sprawdza, czy odwołanie do pamięci było dozwolone czy nie (bit poprawności)
  - Jeśli było poprawne, sprowadza stronę do pamięci, modyfikuje tablicę stron i wznawia działanie procesu.

AGH

Stronicowanie na żądanie a wymiana

- Proces jest ciągiem stron,
- **Wymiana** dotyczy całego procesu (wszystkich jego stron),
- **Procedura stronicująca** dotyczy poszczególnych stron procesu,
- Procedura stronicująca zgaduje jakie strony będą w użyciu i tylko te ładuje do pamięci. Nigdy nie dokonuje się wymiana całego procesu.

AGH

Stronicowanie na żądanie a spowolnienie systemu (przykład)

p - prawdopodobieństwo braku strony
ECD - efektywny czas dostępu:

$$ECD=(1-p)*cd+p*cos$$

cd - czas dostępu do pamięci (10 do 200 ns)
cos - czas obsługi strony:
- obsługa przerwania „brak strony” (1 ÷ 100 μs)
- czytanie strony (duuużo μs)
- wznowienie procesu (1 ÷ 100 μs)

- Czynności obsługi braku strony:
  - Przejście do systemu operacyjnego,
  - Przechowanie kontekstu,...

Obsługa braku strony (c.d.)

- Określenie, że przerwanie to „brak strony”,
- Sprawdzenie poprawności odwołania do strony i jej położenia na dysku,
- Czytanie z dysku do niezajętej ramki:
  - opóźnienie (~8 ms)
  - szukanie sektora (~15 ms)
  - transfer danych (~1 ms)
- Zmiana przydziału procesora do innego procesu
- Przerwanie od dysku po skończonym transferze
- Przełączenie kontekstu
- Skorygowanie tablicy stron
- Czekanie na przydział procesora i wznowienie procesu

Razem: ok. 25 ms

Obsługa braku strony

25 ms = 25 000 μs = 25 000 000 ns

Efektywny czas dostępu:

ECD=(1-p)*cd+p*cos

ECD=(1-p)*100+p*25000000=

100-100*p+25000000*p=

100+24 000 900*p [ns]

- Efektywny czas dostępu jest więc proporcjonalny do prawdopodobieństwa nie znalezienia strony w pamięci.
- Przy prawdopodobieństwie p = 1%
  ECD wyniesie: **240 109** ns

(czyli 2 400 razy więcej niż dostęp bezpośredni do pamięci).

Zastępowanie stron

- Założenie, że tylko część stron każdego procesu jest potrzebna w pamięci może doprowadzić do nadprzydziału (nadmiar procesów w pamięci i absolutny brak wolnych ramek).
- Aby nie blokować procesu potrzebującego kolejnej ramki, stosuje się **zastępowanie stron**.
  - uruchamia się algorytm typowania ramki-ofiary,
  - stronę-ofiarę **zapisuje** się na dysku,
  - aktualizuje się tablicę wolnych ramek,
  - **wczytuje** się potrzebną stronę do zwolnionej ramki,
  - aktualizuje się tablicę stron
  - wznawia się działanie procesu
- **Dwukrotne korzystanie z dysku bardzo wydłuża czas obsługi braku strony!**

Zastępowanie stron

- **Bit modyfikacji** (dirty bit, zabrudzenia) – jest ustawiany, jeżeli na stronie zapisano choćby jeden bit. Jeśli nic nie zmodyfikowano, nie trzeba takiej strony zrzucać na dysk.
- Zastępowanie stron jest podstawą stronicowania na żądanie.
- Praktycznie każdy proces wykonuje się z użyciem mniejszej ilości ramek niż by wynikało z wielkości procesu.
- Mechanizm działa efektywnie przy dobrze opracowanych algorytmach przydziału ramek i algorytmach zastępowania stron.

Algorytm zastępowania stron

- Algorytmy optymalizowane pod kątem minimalizacji częstości braku stron.
- Algorytmy ocenia się na podstawie ich wykonania dla pewnego ciągu odniesień (odwołań) do pamięci i zsumowanie liczby braku stron w tym ciągu.
- Algorytm FIFO (first in, first out)
  - O każdej ze stron zapamiętuje się informację, kiedy ona została sprowadzona do pamięci.
  - Zastępuje się „najstarszą” stronę.
- Przykład:
- Dla ciągu odniesień do stron: 1,2,3,4,1,2,5,1,2,3,4,5

...1,2,3,4,1,2,5,1,2,3,4,5

- Dla 3 dostępnych ramek dla procesu:

3 ramki

1 4 5
2 1 3 (9 braków stron)
3 2 4

- Dla 4 dostępnych ramek dla procesu:

4 ramki

1 5 4
2 1 5 (10 braków stron)
3 2
4 3

Anomalia Belady’ego ^^^^^^^^^

Algorytm optymalny

- Zastąp tę stronę, która najdłużej nie będzie używana.

Dla ciągu odniesień:

```
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7 7 7 2   2   2     2     2       7
- 0 0 0   0   4     0     0       0
- - 1 1   3   3     3     1       1
```

9 braków stron, nie ma anomalii Belady'ego
Algorytm optymalny jest trudny w realizacji, ponieważ wymaga wiedzy o przyszłym ciągu odniesień.
Jest on używany głównie do teoretycznych studiów porównawczych (o ile % dany algorytm jest gorszy od opt.)

Inne algorytmy

- Algorytm LRU (Least Recently Used) - zastępowanie stron, które były najdawniej używane. Typowanie najstarszych poprzez:
  - **licznik** (w tablicy stron jest rejestr czasu ostatniego używania strony),
  - **stos** (przy każdym odwołaniu do strony, jej numer jest wyjmowany i umieszczany na szczycie stosu.
- Algorytmy bazujące na metodzie LRU
  - **z bitami odniesienia** (po odwołaniu do strony, znacznik przyjmuje wartość 1),
  - **dodatkowe bity odwołań** (co stały czas ustawianie kolejnych bitów rotacyjnie),
  - **druga szansa** (jeśli bit odniesienia=1 to zeruje się go, zmienia czas na bieżący i przegląda kolejne strony -FIFO. Jeśli bit=0 to się stronę wymienia).

Inne algorytmy

- Ulepszony algorytm drugiej szansy
  wykorzystuje się dwa bity: bit odniesienia i bit modyfikacji, jako parę. Powstają cztery klasy stron:
  - (0,0) - nie używana ostatnio i nie zmieniana (najlepsza ofiara)
  - (0,1) - nie używana ostatnio, ale zmieniana (gorsza, bo trzeba ją zapisać)
  - (1,0) - Używana ostatnio, ale nie zmieniana (prawdopodobnie za chwilę będzie znów używana)
  - (1,1) - używana ostatnio i zmieniona (chyba będzie używana niedługo, a poza tym trzeba ją zapisać – najgorsza kandydatka na ofiarę).
- Zastępujemy pierwszą napotkaną stronę z najniższej klasy.

Inne algorytmy

- **Algorytmy zliczające**
  Wprowadzamy licznik odwołań do strony
  - LFU (least frequently used) - wyrzuć najrzadziej używaną
  - MFU (most frequently used) - bo najrzadziej używana jest chyba niedawno wprowadzona do pamięci i będzie niedługo w użyciu
- Obydwa te algorytmy niezbyt dobrze przybliżają optimum.

- **Algorytmy buforowania stron**
- Zanim się usunie ofiarę, wczytuje się potrzebną stronę do wolnej ramki.
- Zaleta: można wcześniej uruchomić proces, zanim strona-ofiara zostanie zapisana na dysku. Zapis robi się w wolnej chwili.
- Po zapisie opróżnioną ramkę dopisuje się do listy wolnych.

Przydział ramek

- Każdemu procesowi system musi przydzielić pulę ramek pamięci potrzebnych do jego pracy.
  Trzy najpopularniejsze algorytmy przydziału:

- **równy** (każdy proces dostaje tyle samo ramek) np 50 ramek i 5 procesów, to każdy dostaje po 10

- **proporcjonalny** (liczba przydzielonych ramek proporcjonalna do wielkości procesu)

- **priorytetowy** (liczba przydzielonych ramek zależy tylko od priorytetu procesu, albo od priorytetu i wielkości).

Globalne i lokalne zastępowanie ramek

- **Zastępowanie lokalne** - proces może zastępować ramki wyłącznie z puli tych, które dostał przy przydziale.
- **Zastępowanie globalne** - Proces może korzystać z puli wszystkich wolnych ramek, nawet jeżeli są wstępnie przydzielone innemu procesowi. Proces może zabrać ramkę drugiemu.

- **Praktyka wykazała, że zastępowanie globalne daje lepszą przepustowość systemu.**

Strona zerowa

- We współczesnych systemach proces może zarezerwować więcej pamięci niż zamawia w momencie jego tworzenia…
  - ...wczytanie pliku do tablicy/zmiennej,
  - ...przechwytywanie danych,
  - ...obliczenia wymagające zmiennej długości buforów.
- Zamówiona pamięć **nie jest przypisywana do procesu od razu.** Wskaźnik na nową stronę wskazuje na stronę zerową – specjalną stronę wypełnioną bajtami \0.
- Dzięki temu do odczytu program może zamówić nawet więcej pamięci niż jest w systemie i w urządzeniu stronicowania. Dopiero zapis wyzwala poszukiwanie nowej ramki w pamięci i przełączenie wskaźnika na stronę odpowiadającą tej ramce.

Szamotanie

- Jeśli proces nie ma wystarczającej liczby ramek, to w pewnym momencie musi wymienić stronę, która będzie potrzebna w niedługim czasie. W konsekwencji, kolejne braki stron będą występowały bardzo często. Taki proces „szamoce się”, spędzając więcej czasu na stronicowaniu niż na wykonaniu.
- Zmniejsza się wykorzystanie procesora.

Scenariusz szamotania

- Jeżeli wykorzystanie jednostki centralnej jest za małe, planista przydziału procesora **zwiększa wieloprogramowość**.
- Nowy proces zabiera ramki pozostałym procesom.
- Zaczyna brakować ramek.
- Procesy ustawiają się w kolejce do urządzenia stronicującego, a jednocześnie zmniejsza się kolejka procesów gotowych...

Szamotanie

- Wykorzystanie procesora maleje, bo procesy stoją w kolejce,
- System zwiększa wieloprogramowość,
- Sytuacja staje się tragiczna - żaden proces nie pracuje, tylko wszystkie stronicują.

- Jak powstrzymać szamotanie lub do niego nie dopuścić?
  - Stosować metodę lokalnego lub priorytetowego przydziału stron,
  - Stosować mechanizmy zmniejszenia wieloprogramowości przy szamotaniu,
  - Zapewnić dostawę wystarczającej ilości ramek.
- **Mierzenie częstości braków stron** - pomiar szamotania.
- Jeśli proces przekracza górną granicę częstości - dostaje wolną ramkę.
- Jeśli przekracza dolną granicę - odbiera mu się ramkę.

Problem operacji I/O

- Proces zamawia operację we-wy i ustawia się w kolejce do urządzenia,
- Procesor przydziela się innym procesom,
- Występują braki stron,
- W wyniku algorytmu globalnego zastępowania stron zostaje wymieniona strona zawierająca bufor we-wy procesu czekającego na operację we-wy,
- Zaczyna się operacja we-wy i nadpisuje dane innego procesu!

- Zapobieganie:
  - Zakaz wykonywania operacji we-wy wprost do pamięci użytkownika - ale kopiowanie czasochłonne,
  - Blokowanie stron w pamięci - strony czekające lub realizujące operację we-wy nie mogą być zastępowane.

W praktyce

- Windows - tradycyjnie przestrzeń stronicowania jest implementowana jako plik, stąd mogą pojawić się kłopoty z jego fragmentacją. Możliwe jest implementowanie jako partycja, lecz jest to rzadko stosowane (nie można łatwo powiększyć wówczas tej przestrzeni).
- Unix, Linux - Przestrzeń stronicowania implementowana jako partycja.
- W wielu systemach przestrzeń stronicowania służy również do odprowadzenia stanu RAM podczas procedury zapisu stanu systemu (Suspend to disk, „hibernacja”).

- Uwaga: W GNU/Linuksie tradycyjnie przestrzeń stronicowania nadal nazywa się „swap” czyli wymiana. To już jednak **nie jest wymiana, to jest stronicowanie**.

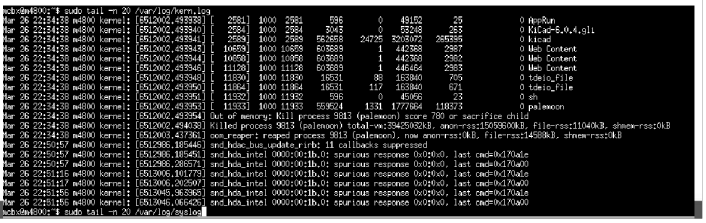
Implementacja

- Windows:
  - Plik wymiany/stronicowania,
  - Plik na osobnej partycji.
    Administratorzy często stosują rozwiązanie pośrednie: Plik na osobnej partycji - dzięki temu zachowujemy elastyczność doboru miejsca (na taki dysk można coś awaryjnie „wgrać”), ale likwidujemy problemy z fragmentacją.

- Unix:
  - Plik stronicowania,
  - Partycja stronicowania,
  - ZRAM, ZSWAP.

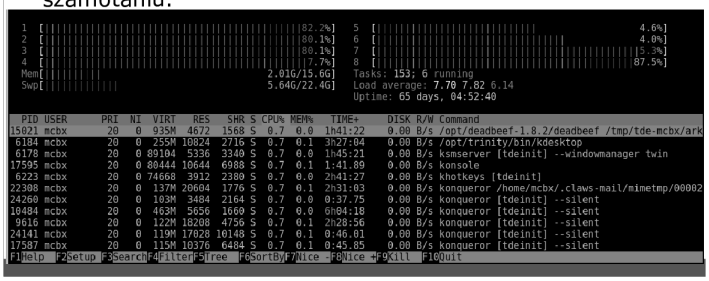
Gdy pamięci rzeczywiście już brakuje

- OOM Killer - część jądra odpowiedzialna za usunięcie z pamięci programu, który nadużył pamięci.
  - Wyznaczane jest, który to proces lub wątek za pomocą obliczania dla każdego z nich kryterium wskazującego, czy usunięcie tego wątku spowoduje największą korzyść dla systemu.
  - Wątek lub proces jest zabijany - preferowane są pojedyncze wątki procesów wielowątkowych.



Post factum

- Procesy, których strony zostały zapisane na dysk, zostają uruchomione.
- Większość procesów „utknie” w kolejce I/O do partycji stronicowania.
- Planowanie średnioterminowe zapobiega wówczas szamotaniu.



Problemy przy wirtualizacji

- Współczesne systemy dysponują różnymi aspektami wirtualizacji - od uruchamiania programów w osobnych enklawach, do pełnej wirtualizacji kilku kopii systemów.
- Każda z tych opcji spełnia jakąś funkcję i umożliwia osiągnięcie różnych celów:
  - Separowanie programów,
  - Dzielenie mocnych serwerów na słabsze wydajnościowo jednostki,
  - Uruchamianie programów kompatybilnych z innymi systemami lub wersjami systemów,
  - Zapewnienie bezpieczeństwa ...

Problemy przy wirtualizacji

- Czyli podczas wirtualizacji:
  - Strony współdzielone:
    - + mniejsze użycie RAM,
    - - bezpieczeństwo
  - Stronicowanie do wspólnej przestrzeni:
    - + Wydajniejsza praca
    - - Możliwy exploit przy skompromitowaniu systemu.
  - Zabezpieczenie przed szamotaniem na poziomie nadzorcy:
    - + Pewność
    - - Łatwa droga do „przeskoczenia” z jednej maszyny do drugiej
  - Zabezpieczenie przed szamotaniem na poziomie poszczególnych maszyn wirtualnych:
    - + Bezpieczeństwo
    - - Nieoptymalne gospodarowanie zasobami.

Jak odseparować systemy?

- PROBLEM:
  Mamy fizycznie wielki serwer, mający dziesiątki rdzeni, setki GB RAMu i TB przestrzeni dyskowej. Chcemy podzielić go na x logicznych serwerów realizujących różne funkcje (kupują różni klienci, różne poziomy złożonej aplikacji, różne wersje systemu itp.).
- Pytania:
  - Jak istotne jest dla nas bezpieczeństwo takiej separacji? (przy wielu użytkownikach więcej niż przy jednym)
  - Czy każda instancja musi uruchamiać swój system operacyjny? I czy to mogą być różne systemy? (np. celem zapewnienia zgodności)
  - Czy każda instancja będzie miała zupełnie inną konfigurację?

Podstawowe techniki

0. Separacja na poziomie użytkowników tego samego systemu.
  - Mogą mieć dostęp do różnych fragmentów systemu plików, różne uprawnienia i przydziały,
  - Mogą mieć przyznawane i blokowane uprawnienia do urządzeń.
  - Działa jeden system operacyjny, w jednej wersji…
  - ...chociaż użytkownicy mogą mieć nieco inne wersje narzędzi.
  - Łatwe rozwiązanie oferowane przez każdy współczesny system.
  - Problem: Nie jest to rozwiązanie najbezpieczniejsze (możliwe przełączanie, użytkownik na hasło).

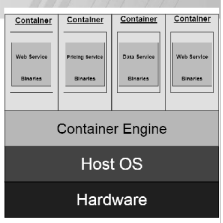
Podstawowe techniki

1. Jedno jądro systemu, wiele zestawów narzędzi (np. przez chroot).
   - Bezpieczniejsze niż poprzednie,
   - Można uruchamiać „jakby” wiele systemów - jądro jest jednak to samo,
   - Każdy „logiczny serwer” ma swój system plików, katalog główny /, zestaw narzędzi, konfigurację - można to zmieniać dopóki nie zmienia się ustawień jądra.
   - Jedno jądro systemu kontroluje całość,
   - Tylko niektóre systemy umożliwiają zrealizowanie tego sposobu.
   - Bezpieczniej, ale wciąż problemy bezpieczeństwa z jądrem systemu → kłopoty z bezpieczeństwem całości.

Podstawowe techniki

2. Konteneryzacja
   - Każdy program ma swój „kontener” definiujący program, jego biblioteki, ustawienia.
   - System uruchamia kontener w odrębnej przestrzeni, „montując” swój system plików (lub jego fragment) o ile to potrzebne do jakiegoś podkatalogu.
   - Kontener może zawierać np. gotowy, skonfigurowany serwer WWW, serwer pocztowy, czy programy obsługi aplikacji.
   - Większe bezpieczeństwo,
   - ...kosztem zmniejszenia wydajności…
   - ...oraz użycia pamięci i I/O (kopie bibliotek!).

Konteneryzacja



- Stanowi rozwinięcie podejścia „jedno jądro - wiele zestawów narzędzi”. Tym razem jest to „jedno jądro - wiele przestrzeni użytkownika” lub, uogólniając, wirtualizacja na poziomie procesów (części systemu operacyjnego).
- W przeciwieństwie do maszyn wirtualnych, nie wymagają systemu na każdy kontener.
- Większość systemów kontenerowych oferuje jakieś API, które ułatwia portatywność aplikacji w kontenerach.
- Dodatkowo oferują możliwość zarządzania i administracji aplikacjami w sposób łatwiejszy niż przy pakietach (odpada „dependency hell”).
- Przykłady: Docker, containerd, FreeBSD jails

Podstawowe techniki

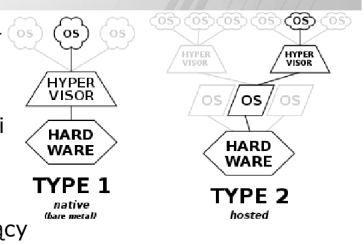
3. Wirtualizacja
   - Jako podstawowy system operacyjny działa hypervisor - program nadzorujący maszyny wirtualne.
   - Każda maszyna uruchamia na swojej (pliko)partycji swoją kopie systemu operacyjnego, z reguły w swojej zarezerwowanej części RAMu.
   - Wysokie bezpieczeństwo (hypervisory są bezpieczne).
   - Pełna możliwość konfiguracji i uruchamiania innych systemów, a nawet emulowania innych architektur (z różną efektywnością).
   - ...kosztem rezerwowania RAMu i przestrzeni dyskowej oraz zmniejszeniem wydajności.

Wirtualizacja

- Umożliwia podział dużego systemu komputerowego na mniejsze, każdy z niezależnym systemem operacyjnym.
- Separacja pamięci jest pod względem bezpieczeństwa wystarczająca dla większości przypadków.
- Dysk podzielony jest na partycje lub obrazy partycji (plikopartycje)
  - W przypadku dysków magnetycznych to rozwiązanie bez prealokacji przestrzeni wprowadza kłopotliwą fragmentację!
- Całością steruje hypervisor - program działający na najwyższych uprawnieniach kontrolujący maszyny wirtualne.

Hypervisor - czym jest?

- Hypervisor typu 1 - Działają bezpośrednio na fizycznej maszynie, nieraz będąc dedykowanymi systemami operacyjnymi, lub modułami zminimalizowanego jądra istniejącego systemu (w przypadku gdy chcemy szybko zaadaptować istniejący system. Np. Xen, Microsoft Hyper-V.



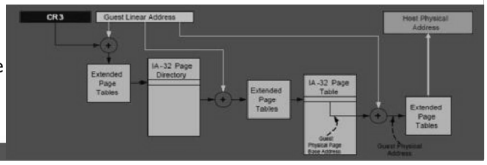
- Hypervisor typu 2 - Działają w ramach systemu operacyjnego działającego na fizycznej maszynie. Maszyny wirtualne startują więc jako procesy w tym systemie. Przykładowo: VirtualBox, Vware, QEMu

Wirtualizacja jest kłopotliwa

- Powiedzmy, że mamy 8-rdzeniowy procesor, 5 maszyn, każda otrzymuje 1 rdzeń.
- Ale MMU jest jedna!
- Pojawiają się problemy z wydajnością dostępu do pamięci:
  - Albo symulujemy MMU dla każdej maszyny na poziomie hypervisora - co zwiększa jego zapotrzebowanie na zasoby. Dodatkowo ułatwia ataki umożliwiające wpłynięcie na pracę innej maszyny.
  - Niektóre architektury procesorów mogą „przywiązać" rdzeń do segmentu pamięci. Wciąż pozostaje problem co ma zastąpić MMU w przeliczaniu adresów logicznych na fizyczne (dla VM) które są logicznymi (dla hypervisora) i znowu są przeliczane na fizyczne (przez MMU).

Sprzętowo wspomagana wirtualizacja

- Występuje w niektórych architekturach procesorów (>AMD Opteron, >Intel i3).
- Każdy rdzeń (lub ich grupa stanowiąca VM) otrzymuje swój zakres w pamięci. Strony są odpowiednio mapowane (Shadow pages - wówczas obsługa przerwaniami!).
- Procesor takiego rdzenia uruchamiany jest „od zera" w trybie rzeczywistym i pozwala mu się pracować w tym zakresie pamięci („unrestricted guest"). Może tam zastartować zupełnie od zera system operacyjny.
- Taki rdzeń z pamięcią korzysta ze stałego segmentu systemowego MMU i bloku buforów translacji TLB (<=Nehalem) lub wykorzystuje MMU i wolne TLB w czasie gdy nikt inny tego nie robi. Nad przydziałem sprawuje opiekę hypervisor (> ok. VPro).



Gdy potrzebna jest tylko zgodność API

- Niektóre systemy posiadają programy udostępniające API innego systemu - np. WINE w Uniksach, CrossOver w Mac'u umożliwiają uruchamianie aplikacji Windowsowych, a WSL w Windowsie - programów linuksowych. Win32s w 16-bitowych Windowsach udostępniał 32-bitowe API umożliwiające uruchamianie 32-bitowego oprogramowania w 16-bitowym systemie.
  - Kompletność tych interfejsów jest jednak różna.

Thunking

- Technika umożliwia „wstrzyknięcie" dodatkowego kodu do wywołanej funkcji – najczęściej na początek lub koniec.
- Dzięki temu te same funkcje mogą być zmodyfikowane dla przypadków szczególnych, a nawet można dodać nowe sposoby ich wywołania.
- Można również zmodyfikować zachowanie i wywołanie funkcji dla zachowania zgodności.
- Zastosowanie: Modyfikacja i tłumaczenie wywołań API, zapewnienie modularności (np. Sterowniki niektórych urządzeń w Windows wywołują modyfikowany podprogram obsługi charakterystyczny dla urządzenia).

Thunking – skąd się to wzięło?

- Wołamy funkcję int funkcja(int zmienna) :

  wynik = funkcja (zmienna + 2)

- Zauważmy, że zanim uruchomi się funkcja, uruchamia się "zmienna + 2".
- Tu możemy poszerzyć działanie funkcji. Nawet, gdy funkcja to wołanie binarnego "bloba" z API systemu.

Shimming

- Jest to przechwytywanie rozkazów API i ich modyfikacja/adaptacja.
- W przeciwieństwie do thunkingu nie modyfikuje się tu kodu sprzed wywoływania funkcji, jednakże można dla danej funkcji API wykonać zupełnie nowy, inny kod.
- Implementowane jako „podkładka" - moduł pośredniczący między programem a systemem, nie jako moduł nadrzędny, jak w thunkingu.

Dziękuję za uwagę