Systemy operacyjne

2018/2019

Jarosław Koźlak

Prowadzący wykład

- **Prowadzący:**
  - dr hab. inż. Jarosław Koźlak

- **Kontakt:**
  - e-mail: kozlak@agh.edu.pl
  - www: http://home.agh.edu.pl/~kozlak
  - telefon: 12 3283308
  - pokój: : D17, 1 piętro, pokój 2.24

Organizacja zajęć i egzaminu

- **Laboratorium**
  - dopuszczalna jedna nieobecność nieusprawiedliwiona (ale nie zwalnia to z konsekwencji 0 punktów podczas ewentualnego sprawdzianu/kartkówki, w tym celu trzeba nieobecność usprawiedliwić)

- **Egzamin:**
  - Termin zerowy – ustny (losuje się zestawy z trzema pytaniami)
  - Pozostałe terminy – pisemne (krótkie odpowiedzi na kilkanaście pytań)
  - ocena 5.0 z laboratorium dopuszcza do zerowego terminu egzaminu

Tematyka wykładów

1. Wprowadzenie
   - Podstawowe pojęcia. Historia systemów operacyjnych.
   - Struktura systemów komputerowych i systemów operacyjnych.
   - Przykłady systemów operacyjnych: Unix. Linux. Windows.
2. Zarządzanie procesami
   - Procesy, planowanie przydziału procesora
   - Synchronizacja
   - Zakleszczenia
3. Zarządzanie pamięcią
   - Informacje wstępne, organizacja pamięci, rodzaje adresów, przestrzenie adresowe
   - Ciągły przydział pamięci. Algorytmy.
   - Stronicowanie,
   - Segmentacja
   - Pamięć wirtualna, zasada lokalności, stronicowanie na żądanie
   - Algorytmy zastępowania stron
4. Realizacja systemu plików
   - Organizacja systemu plików. Warstwy.
   - Sposoby przydziału miejsca na dysku plikom: przydział ciągły, listowy, indeksowy. I-wezły
   - Organizacja katalogów
5. Urządzenia i systemy wejścia-wyjścia
   1. Klasyfikacje urządzeń
   2. Odpytywanie, przerwania, bezpośredni dostep do pamięci
   3. Szeregowanie zamówień dyskowych

Literatura. Teoria

1. **Abraham Silberschatz, Peter B. Galvin, "Podstawy systemów operacyjnych", WNT, 2005 (wyd. 6 zmienione)**
   - **A. Silberschatz, P.B. Galvin, G. Gagne. „Operating System Concepts. Eighth Edition”, John Wiley & Sons Inc, 2009**
2. **William Stallings, „Systemy operacyjne”, 2018**
   - **William Stallings, „Operating Systems. Internals and Design Principles”. Ninth Edition, Pearson Prentice Hall, 2017**
3. Andrew S. Tanenbaum, "Modern Operating Systems", Third Edition, Prentice Hall International, 2008
4. Andrew S. Tanenbaum, Albert S. Woodhull, "Operating Systems. Design and Implementation" Prentice-Hall International, Inc. , 3rd ed, 2006
5. Thomas W. Doeppner, „Operating Systems in Depth”, Wiley, 2010
6. Thomas Anderson, Michael Dahlin, „Operating Systems. Principles & Practice”, Second Edition, 2011-2014

Literatura. Programowanie współbieżne

1. Ben-Ari “Podstawy programowania współbieżnego i rozproszonego”, Wydawnictwo Naukowo-Techniczne, Warszawa 1996.
2. Iszkowski, M. Maniecki “Programowanie współbieżne`”, Wydawnictwo Naukowo-Techniczne, Warszawa 1982.
3. Z. Weiss, T. Gruźlewski “Programowanie współbieżne i rozproszone w przykładach i zadaniach, Wydawnictwo Naukowo-Techniczne, Warszawa 1994.

Architektura systemu Unix/Linux

1. **Berny Goodheart, James Cox, "Unix System V Wersja 4 od środka", Wydawnictwo Naukowo-Techniczne, 2001 ("The magic garden explained. The internals of Unix System V Release 4. An open systems design", 1994)**

2. **Uresh Vahalia, „Jądro systemu Unix...”, WNT 2001**

3. **Daniel P. Bovet, Marco Cesati, „Understanding the Linux Kernel”, Third edition, O’Reilly, 2006**
   - Daniel P. Bovet, Marco Cesati, "Linux Kernel", Wydawnictwo RM 2001

4. **Robert Love, „Linux Kernel Development”, Novel Press, Third edition, 2010**

Programowanie w systemie Unix

1. **W. Richard Stevens, „Programowanie w środowisku systemu Unix”, WNT, 2002.**
   W. Richard Stevens, Stephen A. Rago, "Advanced programming in the Unix Environment", Third Edition, Addison Wesley Publishing Company, 2013
2. **Mark Mitchell, Jeffrey Oldham, Alex Samuel, "Linux. Programowanie dla zaawansowanych", Wydawnictwo ReadMe, 2002 (orig: "Advanced Linux Programming", New Riders Publishing, 2001)**
3. W. Richard Stevens, “Unix: Programowanie sieciowe”, t1-t2 WNT, 2000,2001
4. Kay A. Robbins, Steven Robbins, „Unix System Programming. Communication, concurrency and threads”, Prentice Hall, 2003
5. Marc J. Rochkind, „Advanced UNIX Programming”, Addison-Wesley, Second edition, 2004
6. **Michael Kerrisk, A Linux and Unix System Programming Handbook, no starch press. 2010.**

Windows

1. William R. Stanek, „Windows Server 2008. Inside Out,” Microsoft Press, 2008
2. Charles Petzold, „Programowanie Windows”, RM (orig: Microsoft Press),1999
3. Victor Toth , "Programowanie Windows 98/NT", Helion 1999
4. Jeffrey Richter, Programowanie aplikacji dla Microsoft Windows, Wydawnictwo RM, 2002
5. Mark E. Russinovich, David A. Solomon, Alex Ionescu, „Windows Internals” Part 1-2, Microsoft Press, 2012

Wybrane strony WWW

- Levenez E.: *UNIX History,* http://www.levenez.com/unix/
- Levenez E.: *Windows History,* http://www.levenez.com/windows/
- The Open Group: *UNIX History and Timeline*, http://www.unix.org/what_is_unix/history_timeline.html

- theForger's Win32 API Tutorial, http://www.winprog.org/tutorial/

- POSIX Threads Tutorial, http://math.arizona.edu/~swig/documentation/pthreads/
- POSIX Threads Programming, https://computing.llnl.gov/tutorials/pthreads/

- The Linux Kernel Archives, http://kernel.org/
- Linux Documentation Project, http://mirrors.kernel.org/LDP/

Regulamin laboratorium

Zaliczenie laboratorium uzyskuje się po zrealizowaniu następujących warunków:

- uczęszczanie na laboratoria i realizacja zadanych na nich zadań, bierze się pod uwagę:
  - obecność (dopuszczalna jedna nieobecność nieusprawiedliwiona),
  - przygotowanie do zajęć
  - realizacja zadań (dopuszczalny 1 tydzień spóźnienia)
- zaliczenie sprawdzianów (tematyka ostatnich ćwiczeń) i 2 x kolokwium (tematyka ostatnich kilku ćwiczeń + wybrane zagadnienia z wykładu)

Inne uwagi:

- podział na grupy, grupy zrównoważone

Plan laboratorium

1. g++/gcc. Make i gdb. Zarządzanie pamięcią. Biblioteki statyczne, współdzielone i dynamiczne. Przydatne funkcje. Pomiar czasu
2. Operacje na plikach
3. Tworzenie procesów. Środowisko procesu. Sterowanie procesami.
4. Sygnały
5. Potoki nazwane i nienazwane
6. Kolejki komunikatów (IPC & Posix)
7. Semafory i pamięć wspólna (IPC & Posix)
8. Wątki. (Podstawy, biblioteki)
9. Wątki (synchronizacja)
10. Sokety

Wprowadzenie

Struktura systemu komputerowego

System komputerowy złożony ze sprzętu, programów systemowych i aplikacji.

Struktura systemu komputerowego 2

- **Urządzenia fizyczne :** zintegrowane układy elektroniczne, kable danych, kable zasilające itd.
- **Mikroprogram:** prosty software, który bezpośrednio kontroluje urządzenia fizyczne, zwykle jest ulokowany w pamięci ROM
- **Język maszynowy:** obejmuje zbiór instrukcji, które mikroprogram potrafi wykonać, typowo 50-300 instrukcji dotyczących przesyłania danych, operacji arytmetycznych, porównywania wartości. Urządzenia WE/WY są obsługiwane przez ładowanie wartości do ***rejestrów urządzeń***
- **System operacyjny:** ma ukrywać złożoność operacji na sprzęcie i dostarczać wygodniejszy zbiór instrukcji. SO jest częścią oprogramowania, które jest uruchamiane w trybie jądra/trybie użytkownika
- **Powyżej systemu operacyjnego:** reszta oprogramowania i aplikacje, uruchamiane w trybie użytkownika.

Co to jest system operacyjny **?**

- **Popularne określenie zakresu systemu operacyjnego:**
  - To, co dostawca przysyła w odpowiedzi na nasze zamówienie „systemu operacyjnego”

- System operacyjny – przypomina rząd: Nie wykonuje sam żadnej użytecznej funkcji, ale ma za zadanie przygotować środowisko, w którym inne programy mogą wykonywać pożyteczne funkcje.

Różne spojrzenia na system operacyjny

- **System operacyjny jako Rozszerzona Maszyna**
  - architektura komputerów na poziomie języka maszynowego jest trudna do użycia w programach (szczególnie: operacje wejścia/wyjścia)
  - zadaniem systemu operacyjnego jest ukrycie tej złożoności i dostarczenie programiście bardziej przyjaznego interfejsu
  - system operacyjny udostępnia maszynę rozszerzoną /maszynę wirtualną, łatwiejszą do programowania

- **System Operacyjny jako Zarządca Zasobów**
  - różne rodzaje zasobów w systemie: procesory, pamięć, zegary, dyski, terminale, napędy taśmy magnetyczne, drukarki itd.
  - komputer ma za zadanie udostępniać zasoby użytkownikowi, kontrolować dostęp do nich i zapobiegać chaosowi i konfliktom między programami (i użytkownikami)

- **System operacyjny jako program sterujący:**
  - Nadzoruje działanie programów użytkownika, przeciwdziała błędom i zapobiega niewłaściwemu użyciu komputera.

Cele systemu operacyjnego

- wygoda użytkownika (!)
- efektywne działanie systemu komputerowego (szczególnie istotne w rozbudowanych, wielodostępnych systemach z podziałem czasu)

- możliwe sprzeczności między powyższymi celami

- początkowo: przedkładano wydajność nad wygodę
- obecnie: najczęściej przedkłada się wygodę nad wydajność

Historia systemów operacyjnych: Proste systemy wsadowe

Cecha charakterystyczna: brak bezpośredniego nadzoru ze strony użytkownika podczas wykonywania jego zadania

Struktura systemu komputerowego:

- wielkie fizycznie maszyny obsługiwane przez konsolę
- urządzenia wejściowe: czytniki kart i przewijaki taśm
- urządzenia wyjściowe: drukarki wierszowe, przewijaki taśm, perforatory kart
- użytkownik przygotowywał zadanie (program, dane i informacje sterujące zapisane na kartach) i przekazywał je operatorowi
- wyniki były uzyskiwane po pewnym czasie (min., godz., dni)

Historia systemów operacyjnych: Proste systemy wsadowe 2

Struktura systemu operacyjnego:

- SO rezyduje na stałe w pamięci operacyjnej
- obowiązek SO : automatyczne przekazywanie sterowania od jednego zadania do następnego
- w celu przyspieszenia przetwarzania, zadania o podobnych wymaganiach grupowane razem i wykonywane w formie tzw. wsadu (batch), sortowanie dokonywane przez operatora
- jednostka centralna często pozostaje bezczynna w wyniku wolnej pracy mechanicznych urządzeń WE/WY

Historia systemów operacyjnych: Proste systemy wsadowe

Modyfikacje:

- wprowadzono dyski , co pozwoliło na zastosowanie spooling'u (*Simultaneous Peripheral Operation On-Line* – jednoczesna, bezpośrednia praca urządzeń)
- spooling pozwala na jednoczesne wykonywanie obliczeń jednego zadania i operacji WE/WY innego
- dysk – bufor, pozwala na czytanie z maksymalnym wyprzedzeniem z urządzeń WE i przechowywanie plików wyjściowych, aż urządzenia WY będą je mogły przyjąć
- zawartość karty nie była po odczytaniu z czytnika ładowana bezpośrednio do pamięci, ale przechowywana na dysku, a system operacyjny przechował tablicę opisującą rozmieszczenie obrazów kart na dysku
- podobnie – komunikaty wyjściowe były zapisywane do bufora systemowego i na dysku, a drukowane dopiero po zakończeniu zadania

Historia systemów operacyjnych: Wieloprogramowane systemy wsadowe

- Ze spoolingiem jest związana pula zadań (**job pool**): zadania, które zostały odczytane na dysk, gdzie czekają na wykonanie
- Istnieje możliwość **wyboru** przez system zadania do wykonania, spośród zadań przechowywanych na dysku.
- Wybór zadania jest związany z użyciem planowania zadań (szeregowania zadań, ang. job scheduling)
- Najważniejszy aspekt planowania zadań: **wieloprogramowanie**
- **Wieloprogramowanie:** w tym samym czasie system operacyjny przechowuje w pamięci kilka zadań – część z zadań zgromadzonych w **puli zadań**
- Przesłanki stosowania wieloprogramowania: jedno zadanie jednego użytkownika na ogół nie jest w stanie utrzymać cały czas w aktywności procesora lub urządzeń WE/WY

Historia systemów operacyjnych: Systemy z podziałem czasu

Wady systemów wsadowych:

- użytkownik nie może ingerować w zadanie podczas jego wykonywania się, musi zatem przygotować karty sterujące dla wszystkich możliwych zdarzeń.
- następne kroki wykonania zadania mogą zależeć od wcześniejszych wyników (np. kompilacja, uruchamianie...)
- trudności w testowaniu, programista nie może na bieżąco zmieniać programu w celu obserwacji jego zachowań

Wielozadaniowość i systemy z podziałem czasu

- **Wielozadaniowość (ang. multitasking):** procesor wykonuje na przemian wiele zadań, przełączenia między nimi występują tak często, że użytkownicy mogą mogą współdziałać ze swymi programami podczas ich wykonania.
- **System z podziałem czasu (ang. time-sharing systems)** – wielu użytkowników, każdy uzyskuje dostęp do procesora przez pewną małą porcję czasu; każdy użytkownik ma przynajmniej jeden proces w pamięci
- **Bezpośredni dostęp do systemu plików** (ang. on-line file system) – umożliwia wygodne korzystanie z danych i oprogramowania.
- **Plik (ang. file)** – zestaw powiązanych informacji, zdef. prze twórcę
- **Interakcyjny/bezpośredni (ang. hands on)** system komputerowy umożliwia bezpośredni dialog użytkownika z systemem
- Wejście (zazwyczaj): klawiatura,
- Wyjście (zazwyczaj): ekran (np. monitora)
- „instrukcje sterujące” przekazywane ze pośrednictwem klawiatury
- **Czas odpowiedzi** (ang. response time) powinien być krótki – max. **rzędu sekund**

Systemy wsadowe, a systemy interakcyjne

- systemy wsadowe są odpowiednie dla wielkich zadań, których wykonanie nie wymaga bezpośredniego dozoru
- zadanie interakcyjne składa się z wielu krótkich działań, a rezultaty poszczególnych poleceń mogą być nieprzewidywalne
- Rozbudowane mechanizmy: kolejkowanie, pamięć wirtualna itd.

Systemy operacyjne dla komputerów osobistych

- spadek cen sprzętu komputerowego -> stworzenie (znów) systemów komputerowych dla indywidualnych użytkowników (lata 70-te)
- Zmiany urządzeń WE/WY:
  - pulpity przełączników, czytniki kart -> klawiatury (wzór. na maszynach do pisania) i myszki
  - drukarki wierszowe i perforatory kart -> monitory ekranowe i małe, szybkie drukarki
- początkowo procesory są pozbawione cech potrzebnych do ochrony systemu operacyjnego przed programami użytkowymi, systemy operacyjne nie są wielostanowiskowe ani wielozadaniowe
- w rozwoju położono nacisk na maksimum wygody użytkowania i szybkość kontaktu z użytkownikiem
- z czasem –ze wzrostem możliwości sprzętu i podłączeniem mikrokomputerów do sieci- początkowo cechy dużych maszynach, są adoptowane do mikrokomputerów (ochrona plików, wielozadaniowość)

Systemy równoległe

- systemy wieloprocesorowe (systemy ściśle powiązane)
- **systemy wieloprocesorowe (ang. multiprocessor systems) –** pewna liczba procesorów współpracuje ze sobą dzieląc szynę, zegar i (czasami) pamięć i urządzenia zewnętrzne
- **takie systemy: systemy ściśle powiązane (tightly coupled)**
- **Zalety systemów wieloprocesorowych:**
  - **przyspieszenie pracy** (ale przy n procesorach < n)
  - **zwiększenie niezawodności**
  - **fault tolerant systems – systemy tolerujące awarie**
  - **właściwość łagodnej degradacji (ang. graceful degradation) --** system jest w stanie kontynuować pracę po awarii części sprzętu

Modele działania systemów wieloprocesorowych:

sprzętowe i programowe podwojenie funkcji

- – 2 identyczne procesory z lokalną pamięcią: procesor podstawowy i zapasowy, każdy proces ma 2 kopie na maszynie podstawowej i zapasowej, w ustalonych punktach kontrolnych stan informacji o każdym zadaniu jest kopiowany z maszyny podst. do zapasowej; kosztowne rozwiązanie

wieloprzetwarzanie symetryczne (ang. symmetric multiprocessing) –

- na każdym procesorze działa identyczna kopia systemu operacyjnego, które komunikują się w zależności od potrzeb:
- zaleta: równocześnie może pracować wiele procesów, bez pogarszania działania całego systemu
- wada: może dojść do sytuacji, że jedne procesory będą przeciążone, a inne – słabo obciążone, dlatego procesory mogą korzystać z pewnych wspólnych struktur danych; np. Solaris

wieloprzetwarzanie asymetryczne (ang. asymmetric multiprocessing) –

- każdy procesor ma przypisane inne zadanie, istnieje wyróżniony procesor główny, który zarządca systemem – najczęściej występuje w bardzo wielkich systemach, w których najwięcej czasu zajmują oper WE/WY -> zastosowanie procesora czołowego (front-end processor), który działa jak bufor między końcówką konwersacyjną i procesorem głównym;

Systemy rozproszone (ang. distributed systems)

inna nazwa: luźno połączone (ang. loosely coupled)

- wiele procesorów, procesory nie dzielą pamięci ani zegara
- każdy procesor ma własną pamięć lokalną
- komunikacja między procesorami przy użyciu linii komunikacyjnych
- procesory w takim systemie mogą być nazywane: stanowiska (sites), węzły (nodes),

Przyczyny tworzenia systemów rozproszonych:

- **podział zasobów –** użytkownik jednego stanowiska może korzystać z zasobów (np. plików, drukarek) dostępnych na innych
- **przyspieszanie obliczeń**
  - jeżeli obliczenie można rozłożyć na zbiór obliczeń cząstkowych, które mogą być wykonywane współbieżnie, to można je przydzielić do poszczególnych stanowisk
  - jeżeli stanowisko jest przeciążone zadaniami, to część z nich można przenieść do innego, mniej obciążonego – dzielenie obciążeń (ang. load sharing)
- **niezawodność --** w przypadku awarii jednego stanowiska, pozostałe mogą kontynuować pracę
- **komunikacja –** możliwość wymiany informacji, okna (X), poczta elektroniczna (ang. electronic mail),

Systemy czasu rzeczywistego (ang. real time systems)

- stosowane, gdzie istnieją surowe wymagania na czas wykonania operacji lub przepływu danych

- Przykład: sterownik w urządzeniu np. przy nadzorowaniu eksperymentów naukowych, obrazowaniu badań medycznych, sterowaniu procesami przemysłowymi , systemach wizualizacji
- komputer pozyskuje dane z czujników o musi je analizować i regulować działanie kontrolowanego obiektu, w zależności od jego stanu

- **Wymaganie systemu czasu rzeczywistego:** przetwarzanie danych **musi** się zakończyć przed upływem określonego czasu

- **W systemie z podziałem czasu:** szybkie uzyskanie odpowiedzi jest pożądane (ale nie jest konieczne)
- **W systemie wsadowym:** nie ma ograniczeń czasowych

Rodzaje systemów czasu rzeczywistego

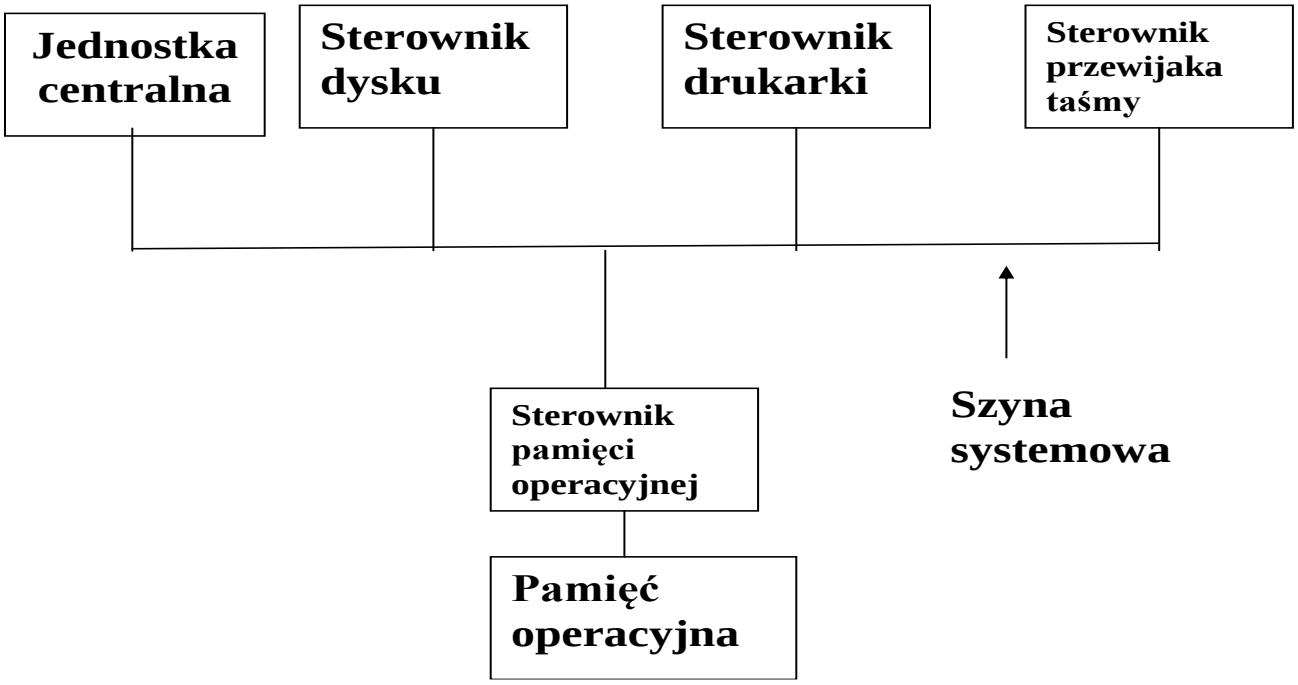
- **rygorystyczny system czasu rzeczywistego (ang. hard real-time system) –** gwarantuje terminowe wypełnianie krytycznych zadań, systemy o specjalnej konstrukcji np. dane w pamięci o krótkim czasie dostępu lub w pamięci ROM, z reguły brak pamięci wirtualnej.
- Żaden z istniejących, uniwersalnych systemów operacyjnych nie umożliwia działania w czasie rzeczywistym.

- **łagodny system czasu rzeczywistego (ang. soft real time system) –** krytyczne zadanie do obsługi w czasie rzeczywistym otrzymuje pierwszeństwo przed innymi zadaniami i zachowuje je aż do swojego zakończenia, opóźnienia muszą być ograniczone --- zadanie nie może czekać w nieskończoność na usługi jądra
- zastosowanie w przemyśle i robotyce jest ryzykowne, przydatne w technikach multimedialnych, kreowaniu sztucznej rzeczywistości, zaawansowanych projektach badawczych (wyprawy planetarne, badania podmorskie)
- Większość współczesnych systemów operacyjnych (w tym Unix) może spełniać te wymagania

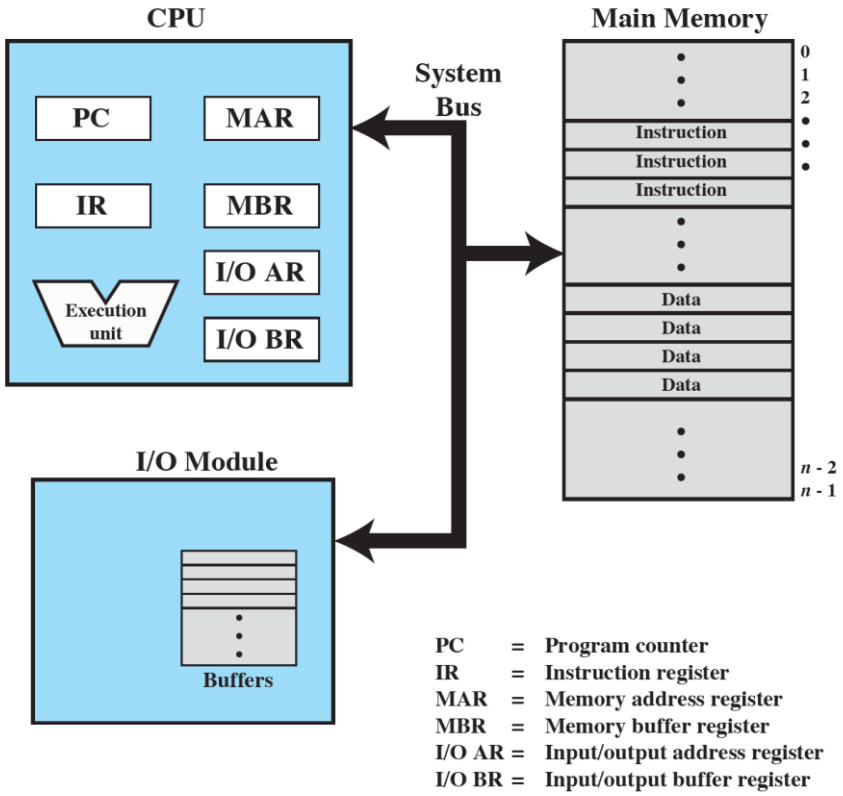
Struktura systemów komputerowych

Elementy składowe systemu komputerowego:

- jednostka centralna (ang. central processor unit - CPU)
- pewna liczba sterowników urządzeń (ang. device controllers)
- pamięć operacyjna
- szyna systemowej łączącej poszczególne elementy

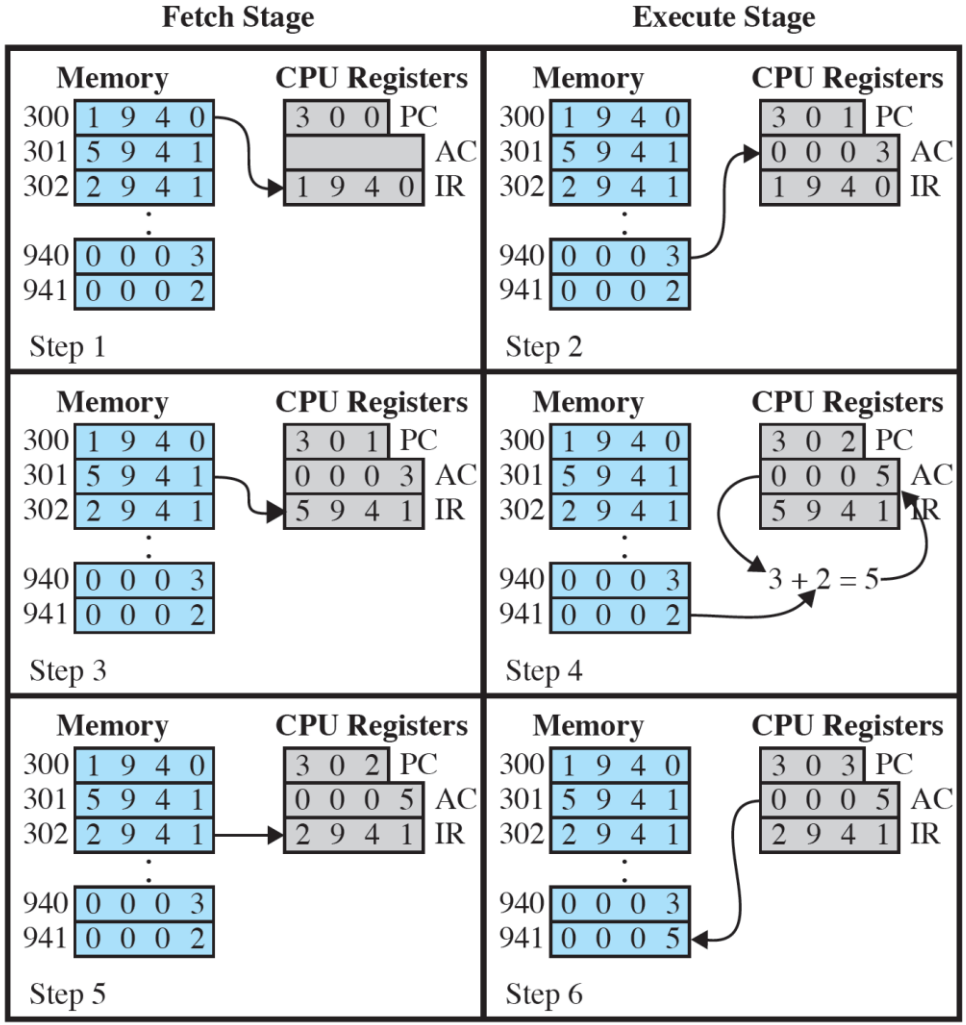


Elementy systemu komputerowego



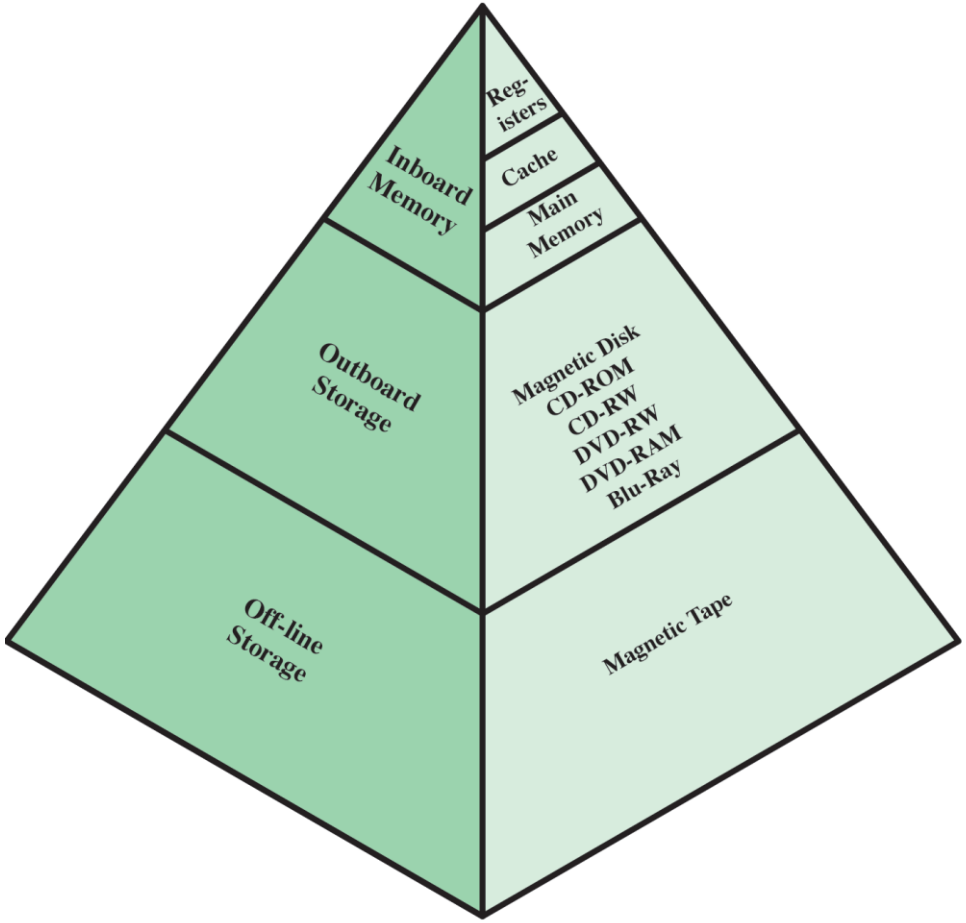
- PC – licznik rozkazów
- IR – rejestr instrukcji
- MAR – specyfikuje adresy w pamięci dla następnej operacji odczytu lub zapisu,
- MBR – zawiera dane zapisywane do pamięci lub odczytywane z pamięci
- I/OAR – określ konkretne urządzenie We/Wy
- I/OBR – używane do wymiany danych między modułem We/Wy i procesorem

Przykładowe wykonanie programu komputerowego



- Wykonanie programu, dodanie zawartości adresu 940 do zawartości adresu 941

Hierarchia pamięci



- Wraz z przemieszczaniem się w dół hierarchii
  - Maleje koszt każdego bitu
  - Wzrasta pojemność
  - Wzrasta czas dostępu
  - Wzrasta częstość dostępu do pamięci przez procesor

Elementy składowe systemu

- **zarządzanie procesami** - tworzenie i usuwanie procesów użytkowych i systemowych, wstrzymywanie i wznawianie procesów, mechanizmy synchronizacji procesów, mechanizmy komunikacji między procesami, mechanizmy obsługi zakleszczeń
- **zarządzanie pamięcią operacyjną** - utrzymywanie ewidencji aktualnie zajętych części pamięci, wybór procesów ładowanych do wolnych obszarów pamięci, stosowne do potrzeb przydzielanie i zwalnianie obszarów pamięci
- **zarządzanie plikami** - tworzenie i usuwanie plików i katalogów, dostarczanie operacji do manipulowania plikami i katalogami, odwzorowywanie plików na obszary pamięci pomocniczej, składowanie plików na stałych nośnikach pamięci
- **zarządzanie systemem WE/WY** - obsługa buforowania, pamięci podręcznej i spoolingu, obsługa ogólnego interfejsu do modułów sterujących urządzeń, obsługa modułów sterujących poszczególnych urządzeń
- **zarządzanie pamięcią pomocniczą** -zarządzanie obszarami wolnymi, przydzielanie pamięci, planowanie przydziału obszarów pamięci dyskowej

Elementy składowe systemu 2

- **praca sieciowa -** dostęp do sieci, zasobów dzielonych
- **system ochrony -** nadzorowanie dostępu do programów, procesów i plików
- **system interpretacji poleceń -** tworzenie procesów, zarządzanie procesami, obsługa WE/WY, administrowanie pamięcią operacyjną i pomocniczą, dostęp do plików, ochrony, sieci

Usługi systemu operacyjnego (1)

- **Usługi dla użytkownika:**
  - **wykonanie programu -** załadowanie do pamięci, uruchomienie, zakończenie
  - **operacje WE/WY -** na pliku lub urządzeniu
  - **manipulowanie systemem plików -** tworzenie, usuwanie, zapisywanie, odczytywanie plików
  - **komunikacja -** pamięć wspólna i komunikaty
  - **wykrywanie błędów**

Usługi systemu operacyjnego (2)

- **Usługi do optymalizacji działania systemu:**
  - **przydzielanie zasobów**
  - **rozliczanie -** przechowywanie danych o tym, jak użytkownicy korzystają z zasobów systemu - do wystawiania rachunków lub celów statystycznych
  - **ochrona -** nadzór nad dostępami do zasobów systemu, zabezpieczenie systemu przed niepożądanymi czynnikami zewnętrznymi - użytkownik musi uwierzytelnić w systemie swoją tożsamość

Struktura systemów operacyjnych:

- **Dekompozycja systemu operacyjnego:**
  - system operacyjny – struktura wielka i złożona
  - musi poprawnie działać i dawać się łatwo modyfikować
  - w tym celu – dekompozycja systemu na małe części,
    - każda część – fragment systemu z określonym wejściem, wyjściem i funkcjami

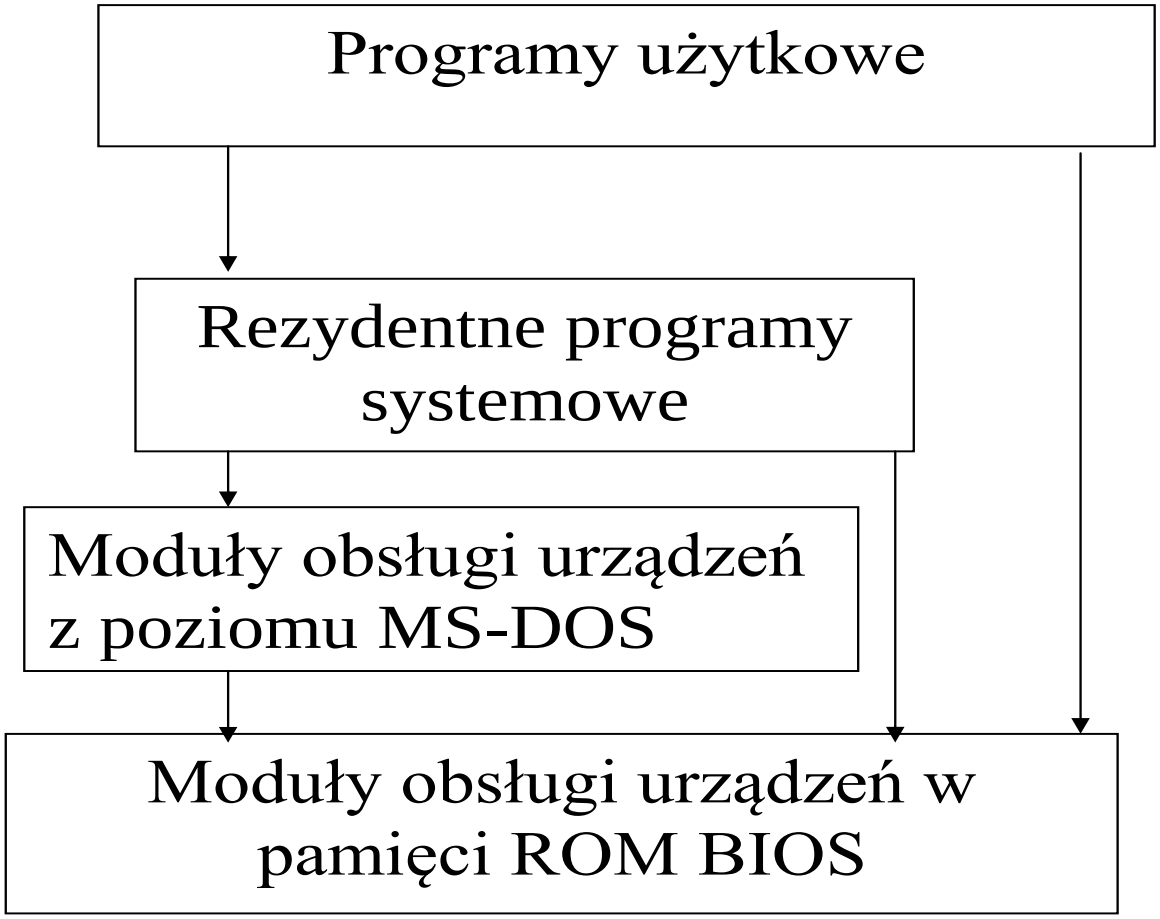
Struktura systemów operacyjnych: Prosta struktura

- systemy nie mające dobrze określonej struktury
- zwykle powstawały jako małe, proste i ograniczone SO, a potem się nadmiernie rozrastały (np. MS DOS)
- oryginalny Unix (początkowo był ograniczany przez sprzęt); dwie części:
  - jądro złożone z interfejsów i programów obsługi urządzeń
  - programy systemowe
- interfejsy Unixa:
  - funkcje systemowe definiują interfejs programisty
  - dostępne programy systemowe określają interfejs użytkownika
- dalszy podział jądra Unixa w kolejnych wersjach

System MS DOS

- interfejsy i poziomy funkcjonalne nie są wyraźnie wydzielone
- programy użytkowe mogą korzystać z podstawowych procedur WE/WY w celu bezpośredniego pisania na ekran lub dyski
- MS DOS nie jest odporny na błędne programy użytkowe
- Intel 8088 nie ma dualnego trybu pracy i ochrony sprzętowej

System MS-DOS 2



Struktura systemu Unix

Podejście warstwowe

- rozwój sprzętu komputerowego, który ułatwia podział SO na mniejsze, lepiej dobrane fragmenty
- ułatwienie modyfikacji systemu poprzez jego modularyzację
- Podejście warstwowe – sposób modularyzacji:
- dzielenie systemu operacyjnego na warstwy (poziomy)
- każda nowa warstwa jest zbudowana powyżej warstw niższych
- najniższa warstwa (0)– sprzęt
- najwyższa warstwa (N)– interfejs użytkownika
- dowolna warstwa pośrednia M:
  - zawiera dane i procedury, które mogą być wywołane z warstw wyższych (M+1)
  - może wywoływać operacje dotyczące warstw niższych
- podejście warstwowe ułatwia wyszukiwanie błędów i weryfikację systemu

Podejście warstwowe 2

- **Możliwe wady** realizacji warstwowych: mniejsza wydajność, obecnie ogranicza się liczbę warstw

Przykład systemu warstwowego: system THE

- *Warstwa 5: Programy użytkowe*
- *Warstwa 4: Buforowanie urządzeń wejścia i wyjścia*
- *Warstwa 3: Program obsługi kontroli operatora*
- *Warstwa 2: Zarządzanie pamięcią*
- *Warstwa 1: Planowanie przydziału procesora*
- *Warstwa 0: Sprzęt*

Maszyny wirtualne

- stosując techniki planowania przydziału procesora i pamięci wirtualnej system tworzy złudzenie, że proces pracuje na własnym wirtualnym procesorze, z własną wirtualną pamięcią
- inne zastosowania maszyn wirtualnych: emulacje innych systemów operacyjnych i architektur
- maszyna wirtualna jest trudna do realizacji (szczególnie np. dostęp do dysków)
- oprogramowanie maszyny wirtualnej może pracować w trybie monitora
- sama maszyna wirtualna może działać tylko w trybie użytkownika: realizuje się dwa wirtualne tryby pracy – użytkownika i monitora, z których każdy pracuje w fizycznym trybie użytkownika
- wirtualne operacje mogą trwać dłużej lub krócej niż rzeczywiste:
  - dłużej: są interpretowane
  - krócej: mogą być symulowane(odwołania do buforowego pliku dyskowego, do pamięci operacyjnej zamiast dyskowej)

Zalety koncepcji maszyny wirtualnej

- pełna ochrona różnorodnych zasobów systemowych
- każda maszyna wirtualna jest odizolowana od innych maszyn wirtualnych (bezpieczeństwo)
- umożliwia badania nad systemami operacyjnymi – łatwiej jest zmienić maszynę wirtualną niż cały system (konwersja danych, zmiana oprogramowania, sprzętu itd.)
- umożliwiają wykorzystanie programów użytkowych napisanych dla innych systemów operacyjnych

Koncepcja mikrojądra

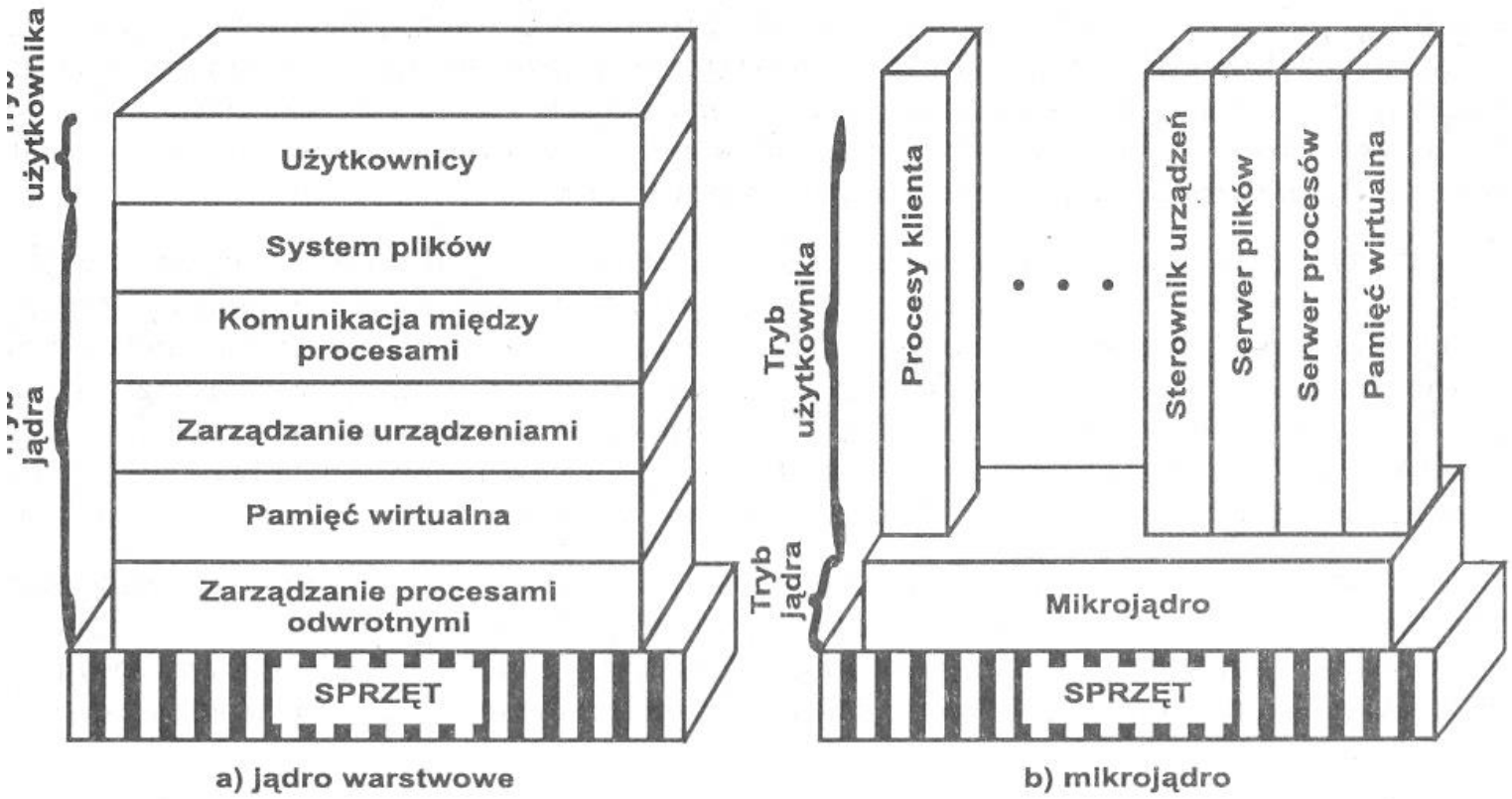
- Trend w nowoczesnych systemach operacyjnych:
  - przesunięcie kodu systemu do wyższych warstw i realizacja wielu funkcji systemowych jako procesy użytkownika
  - maksymalne "odchudzenie" samego systemu operacyjnego i pozostawienie minimalnego jądra

- Mikrojądro – niewielkie jądro systemu operacyjnego będące podstawą dla modularnych rozszerzeń
- Pojęcie spopularyzowane wskutek powstania systemu operacyjnego Mach
- Podejście prowadzi teoretycznie do znacznego wzrostu elastyczności i modularności systemu
- Przykłady architektur opartych na mikrojądrze: Mach, Chorus, L4
- Rozwiązanie zbliżone do architektury mikrojądra: Windows NT

Mikrojądro a systemy monolityczne

- W systemach monolitycznych: każda procedura mogła wywołać inną procedurę
  - Brak struktury: przyczyną niestabilności
- Stworzenie architektury warstwowej (funkcje zgrupowane hierarchicznie) nie rozwiązało problemu – nadal zmiany w jednej warstwie mogą mieć duży wpływ na inne warstwy, konsekwencje często trudne do wyśledzenia
- Architektura mikrojąda: usługi o mniejszym znaczeniu oraz aplikacje powinny być tworzone poza mikrojądrem i działać w trybie użytkownika
- Podsystemy zewnętrzne mogą obejmować: sterowniki urządzeń, systemy plików, menedżer pamięci wirtualnej, system wyświetlania okienek, usługi zabezpieczające

Mikrojądro a systemy warstwowe



Rysunek 4.10 Architektura jądra.

Zalety technologii wykorzystującej mikrojądro

- Jednolite interfejsy
- Rozszerzalność
- Elastyczność
- Przenośność między platformami
- Stabilność
- Obsługa systemów rozproszonych
- Obsługa systemów operacyjnych zorientowanych obiektowo (OOOS)

Zalety technologii wykorzystującej mikrojądro (2)

- **Jednolite interfejsy dla żądań generowanych przez procesy.** Procesy nie muszą odróżniać usług jądra od usług użytkownika, gdyż wszystkie usługi są udostępniane za pomocą przekazywania komunikatów.

- **Rozszerzalność** – architektura mikrojądra sprzyja rozszerzalności, gdyż pozwala dodawać nowe usługi oraz obsługiwać zróżnicowane istniejące usługi oferujące analogiczne funkcje (np. różne usługi dla różnych systemów organizacji plików), dodanie nowej funkcji wymaga dodania nowego serwera lub modyfikacji istniejącego, nie trzeba generować nowego jądra

- **Elastyczność** – można zarówno dodawać nowe usługi i rozbudowywać system jak też usuwać pewne usługi, by uzyskać mniejszą i wydajniejszą realizację

- **Przenośność między platformami -** całość lub dużą część kodu zależnego od architektury sprzętowej znajduje się w mikrojądrze, łatwość przenoszenia na inną architekturę, gdyż trzeba zmieniać tylko dobrze określone, nieliczne moduły

Zalety technologii wykorzystującej mikrojądro (3)

- **Stabilność** – niewielkie mikrojądro może być dokładnie przetestowane, tylko mikrojądro powinno działać w trybie uprzywilejowanym, gdzie wystąpienie błędu jest krytyczne, niewielka liczba interfejsów programowych aplikacji zwiększa szansę wygenerowania dobrego jakościowo kodu znajdującego się poza jądrem

- **Obsługa systemów rozproszonych -** komunikaty skierowane do serwera usługi mogą dotyczyć zarówno serwerów zlokalizowanych na tym samym komputerze jak i na innych

- **Obsługa systemów obiektowych** – rozwiązania obiektowe wymuszają większą dyscyplinę podczas projektowania mikrojądra oraz modularnych rozszerzeń. Użyteczne może być także zastosowanie komponentów – elementów z wyraźnie zdefiniowanymi interfejsami, które można łączyć budując programy z gotowych „klocków”

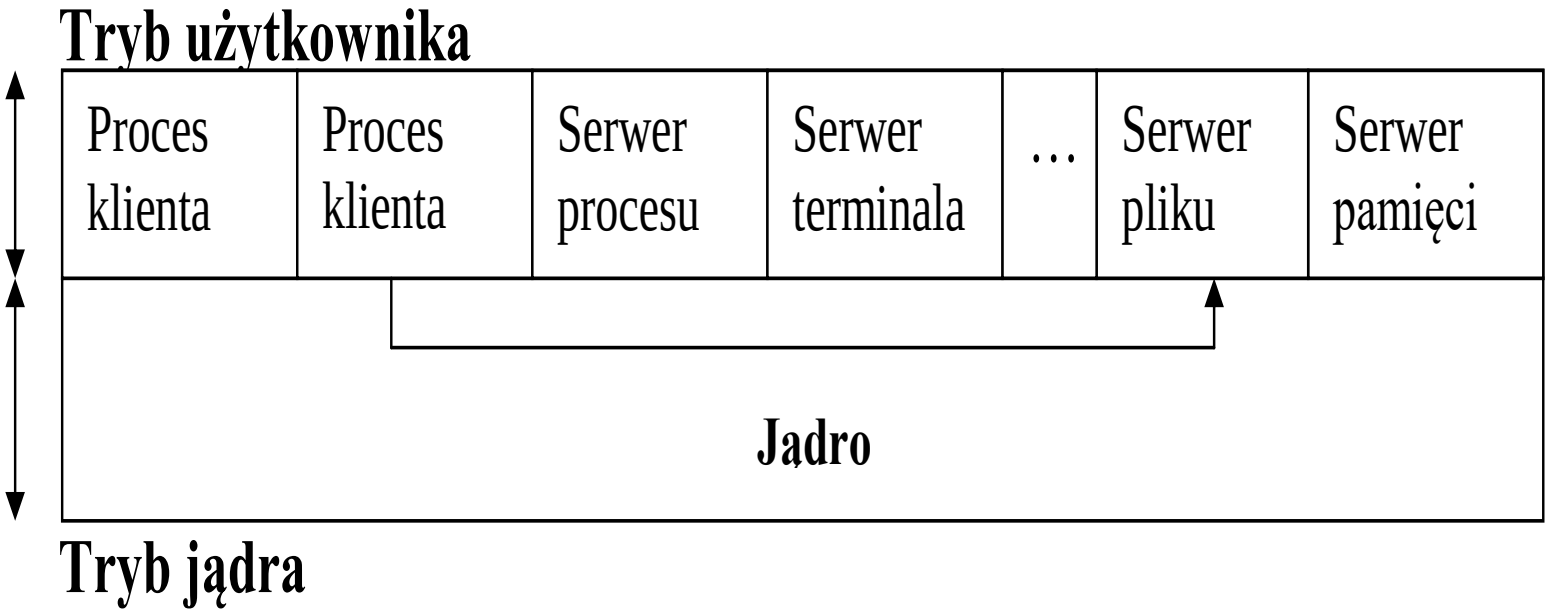
Wady rozwiązań opartych na mikrojądrze

- Wydajność
  - Komunikacja
  - Synchronizacja w przypaku synchronizacji działań dokonywanych przez serwery poziomu użytkownika

Mikrojądro: Struktura klient-serwer

- **Struktura klient-serwer**
  - proces użytkownika (zwany **procesem klienta**) wysyła zlecenia do **procesu serwera**
  - **proces serwera** wykonuje zlecenie
  - **proces serwera** wysyła uzyskany wynik do **procesu klienta**

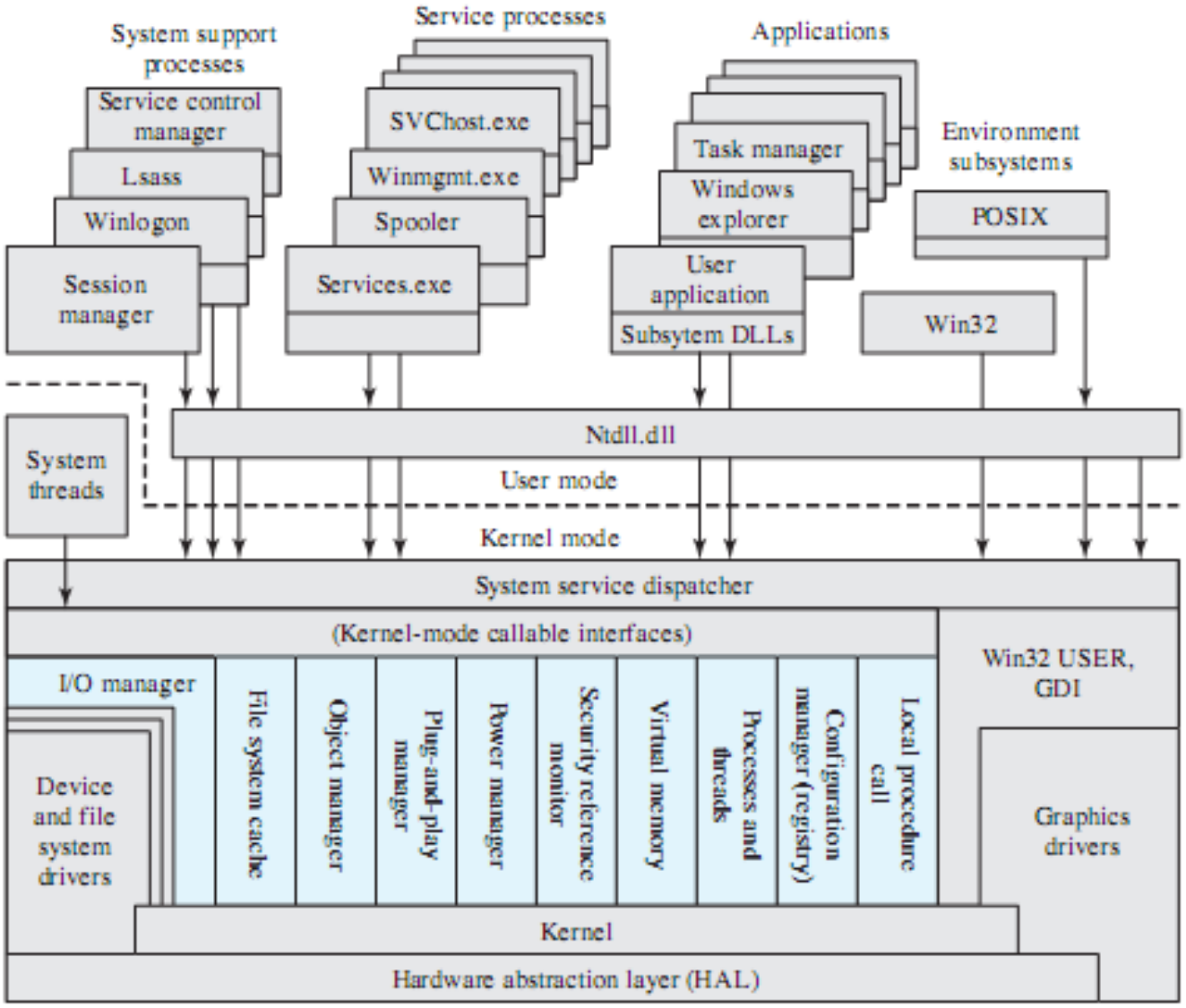
Mikrojądro: Struktura klient-serwer 2



Mikrojądro: Struktura klient-serwer 3

- **jądro -** obsługuje komunikację między klientem i serwerem
- podział systemu na małe części, z których każda zajmuje się tylko konkretną, dobrze określoną dziedziną
- serwery działają w trybie użytkownika, zatem nie mają bezpośredniego dostępu do pamięci
- łatwa adaptacja modelu klient-serwer dla potrzeb systemów rozproszonych
- Umieszczenia serwerów w trybie użytkownika lub jądra:
  - uruchomienie pewnych krytycznych serwerów (np. sterowników urządzeń we/wy) w trybie jądra
  - umieszczenie minimalnej ilości mechanizmów w jądrze, ale pozostawienie modułów określających strategię działań w serwerach w trybie użytkownika

SYSTEM WINDOWS NT



Lsass = local security authentication server
POSIX = portable operating system interface
GDI = graphics device interface
DLL = dynamic link libraries

Colored area indicates Executive

Moduły systemu Windows
Warstwa abstrakcji sprzętu (Hardware Abstraction Layer, HAL)

- Ma bezpośredni dostęp do sprzętu i zapewnia warstwie jądra interfejs niezależny od maszyny
- Umożliwia pisanie programowania jądra w sposób w sposób niezależny od maszyny, sprzyja przenośności systemu operacyjnego na różne platformy sprzętowe
- Procedury HAL mogą być wywoływane z bazowego systemu operacyjnego (jądro), lub przez sterowniki urządzeń (moduł Menadżer I/O)

Windows. Moduł jądra:

- Odpowiada za szeregowanie zadań, harmonogram realizacji wątków, obsługę sytuacji wyjątkowych i synchronizację pracy wieloprocesorowej
- Jądro kieruje wątki do wykonania na dostępnym procesorze
- Każdy wątek ma przydzielany priorytet, wątki o wyższych priorytetach mogą wywłaszczać wątki o niższych priorytetach
- moduł jądra nie jest stronicowany (nonpageable), strony nie są usuwane z pamięci do obszaru wymiany (pliku tymczasowego PAGEFILE.SYS)
- kod modułu nie jest wywłaszczalny, natomiast pozostałe oprogramowanie np. stosowane w modułach wykonawczych Windows jest wywłaszczalne (preemptive)
- Moduł jądra zarządza dwoma klasami obiektów:
  - obiektami dyspozytora
  - obiektami sterującymi
- Moduł jądra może być wykonywany równocześnie na wszystkich procesorach komputerów wieloprocesorowych i sam synchronizuje dostęp do swoich krytycznych miejsc w pamięci.

Podstawowe komponenty modułu wykonawczego(1)

- **Menadżer konfiguracji** – komponent odpowiedzialny za implementacje i zarządzanie rejestrem systemowym
- **Menadżer procesu i wątku** – komponent odpowiedzialny za tworzenie i usuwanie (przerywanie działania) procesów i wątków. Komponent ten wykorzystuje wsparcie dla tego typu działań zaimplementowane w jądrze Windows. Moduł wykonawczy uzupełnia niskopoziomowe obiekty jądra dodatkową semantyką i funkcjami.
- **Monitor bezpieczeństwa** – egzekwuje politykę bezpieczeństwa na lokalnym komputerze. Chroni on zasoby systemu operacyjnego poprzez ochronę i audytowanie działających obiektów.
- **Menadżer I/O** – komponent implementuje niezależne od sprzętu wejście/wyjście oraz jest odpowiedzialny za dyspozycję sterownikami urządzeń.
- **Menadżer Plug and Play (PnP)** – określa wymagany sterownik dla konkretnego urządzenia po czym ładuje ten sterownik.
- **Menadżer zasilania** – koordynuje zdarzenia związane z zasilaniem oraz generuje notyfikacje zarządzania zasilaniem I/O przeznaczone dla sterowników urządzeń. Kiedy system jest w stanie idle, odpowiednio skonfigurowany menadżer zasilania może zredukować pobór mocy poprzez uśpienie CPU. Nad zmianami zasilania urządzeń zewnętrznych czuwają sterowniki tych urządzeń. Są one jednak koordynowane przez menadżera zasilania.

Podstawowe komponenty modułu wykonawczego(2)

- **Funkcje Windows Management Instrumentation** – umożliwiają sterownikom urządzeń publikowanie informacji wydajnościowych oraz konfiguracyjnych oraz otrzymywanie poleceń z usługi WMI – trybu użytkownika. Konsument informacji WMI może rezydować na maszynie lokalnej lub być podłączony poprzez sieć na maszynie zdalnej.
- **Menadżer pamięci podręcznej (cache)** – poprawia wydajności wejścia/wyjścia plikowego, poprzez przechowywanie danych dopiero co odczytanych w pamięci głównej dla szybkiego dostępu do nich oraz poprzez opóźnianie zapisu – gromadzenie danych w pamięci przeznaczonych do zapisu na dysku.
- **Menadżer pamięci wirtualnej** – komponent implementujący pamięć wirtualną, czyli schematu zarządzania pamięcią dostarczającego dużej, prywatnej przestrzeni adresowej dla każdego procesu. Przestrzeń ta może być rozszerzana poza pamięć fizyczną, jaka jest aktualnie dostępna.
- **Menedżer Obiektów** (Object Manager)
- **Wywołanie Procedury Lokalnej** (Local Procedure Call)

Podsystemy środowiskowe. Środowisko Win 32

- główny podsystem systemu -- Win32
- wykonuje aplikacje Win32 i zarządza wszystkimi funkcjami klawiatury, myszki i ekranu
- każdy proces systemu Win32 ma własną kolejkę wejściową
    - zarządca okien przydziela wszystkie operacje wejścia do kolejek wejściowych odpowiednich procesów
  - stosowana jest wielozadaniowość z wywłaszczaniem
  - sprawdza ważność obiektów przed użyciem – zapobiega wykorzystywaniu nieważnych lub błędnych uchwytów

SYSTEM UNIX

Tradycyjne jądro UNIXa

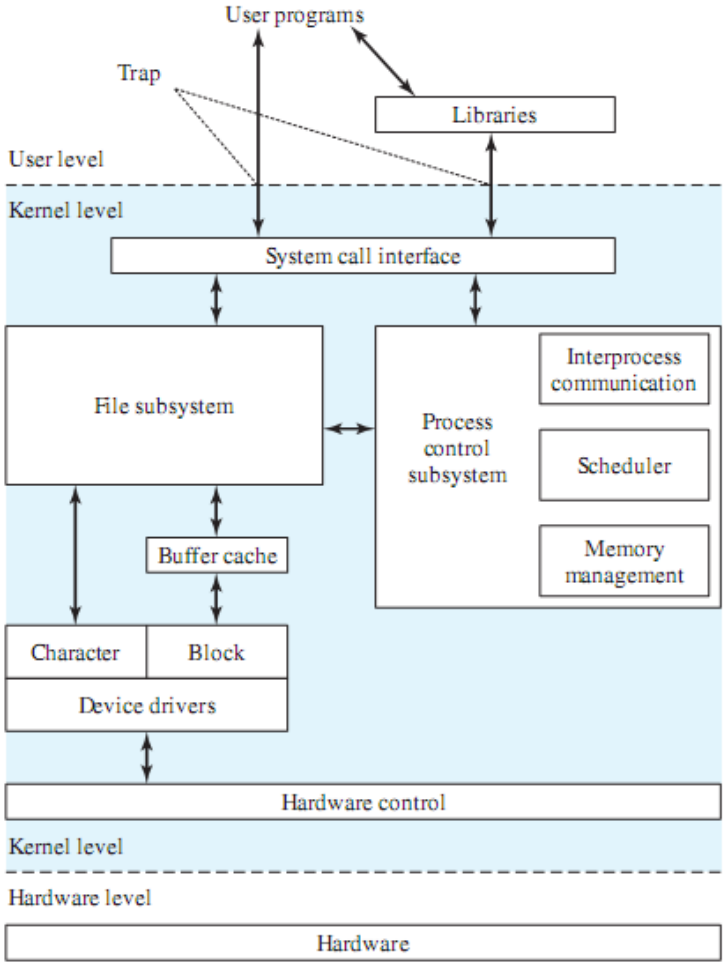


Figure 2.15 **Traditional UNIX Kernel**

System V Release 4 (SVR4)

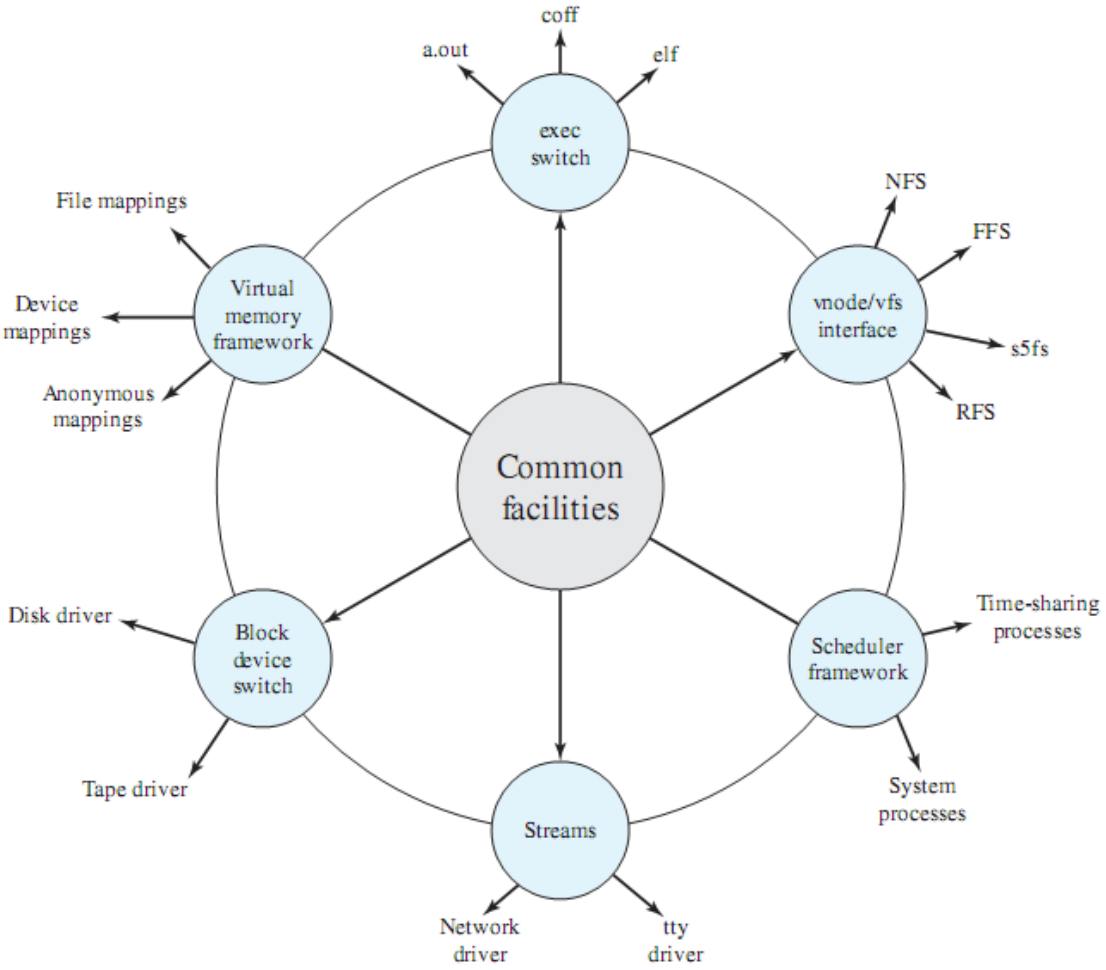
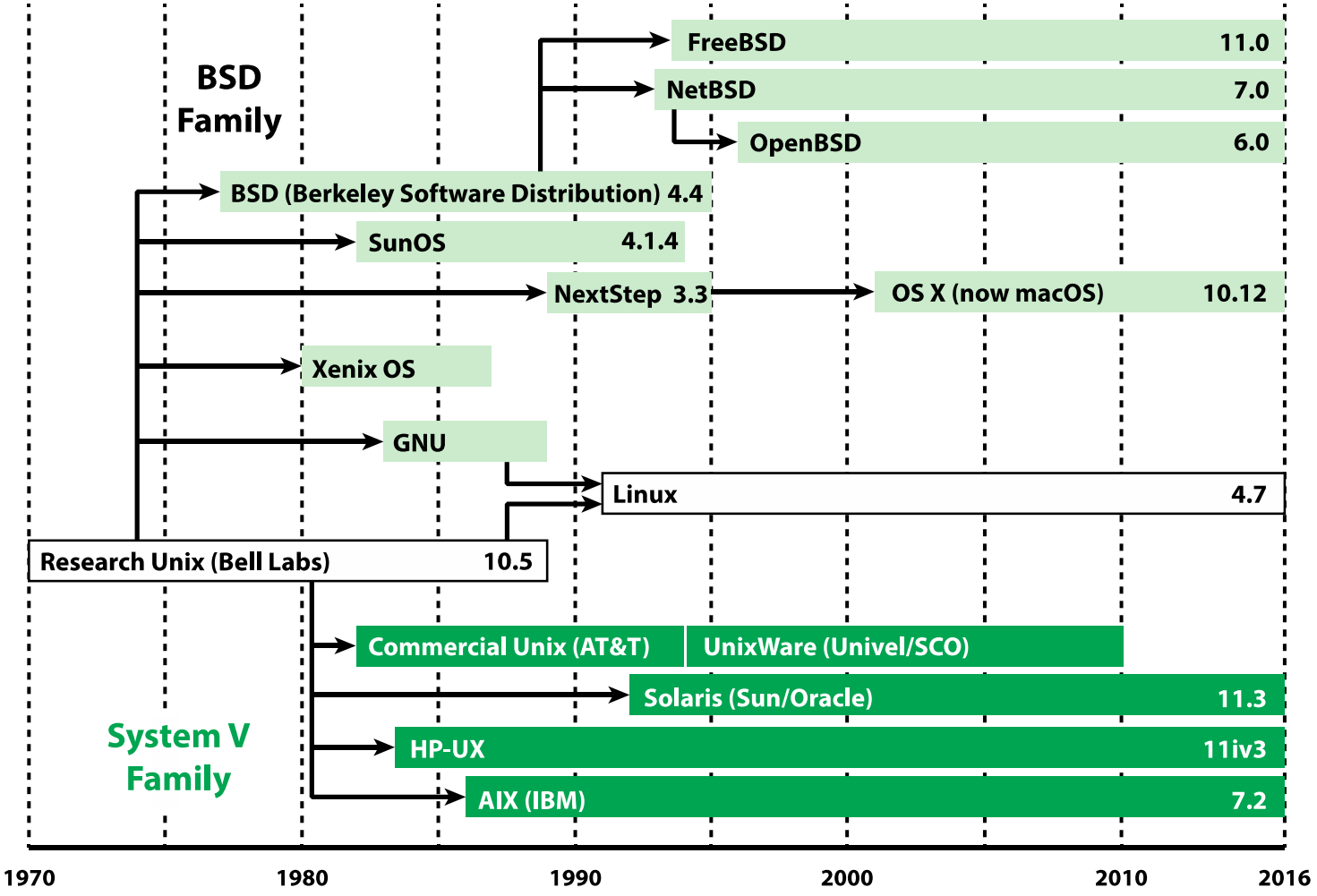


Figure 2.16 **Modern UNIX Kernel**

Uproszczona ewolucja systemu Unix



SYSTEM LINUX

Jądro Uniksa a jądro Linuksa

- Linux wspiera dynamiczne ładowanie modułów jądra
  - Choć jądro Linuksa jest monolityczne, możliwe jest ładowanie kodu jądra na żądanie
- Linux wspiera symetryczną wieloprocesorowość (SMP). Obecnie komercyjne wersje Uniksa wspierają SMP, ale tradycyjne implementacje nie wspierały
- Jądro Linuksa jest wywłaszczalne. Z komercyjnych realizacji Uniksowych, np. Solaris i Irix mają wywłaszczalne jądra, ale tradycyjne implementacje nie były wywłaszczalne
- Linux ma specyficzne podejście do wielowątkowości. Nie ma rozróżnienia wątków i zwykłych procesów. Dla jądra wszystkie procesy są takie same, niektóre jedynie współdzielą zasoby

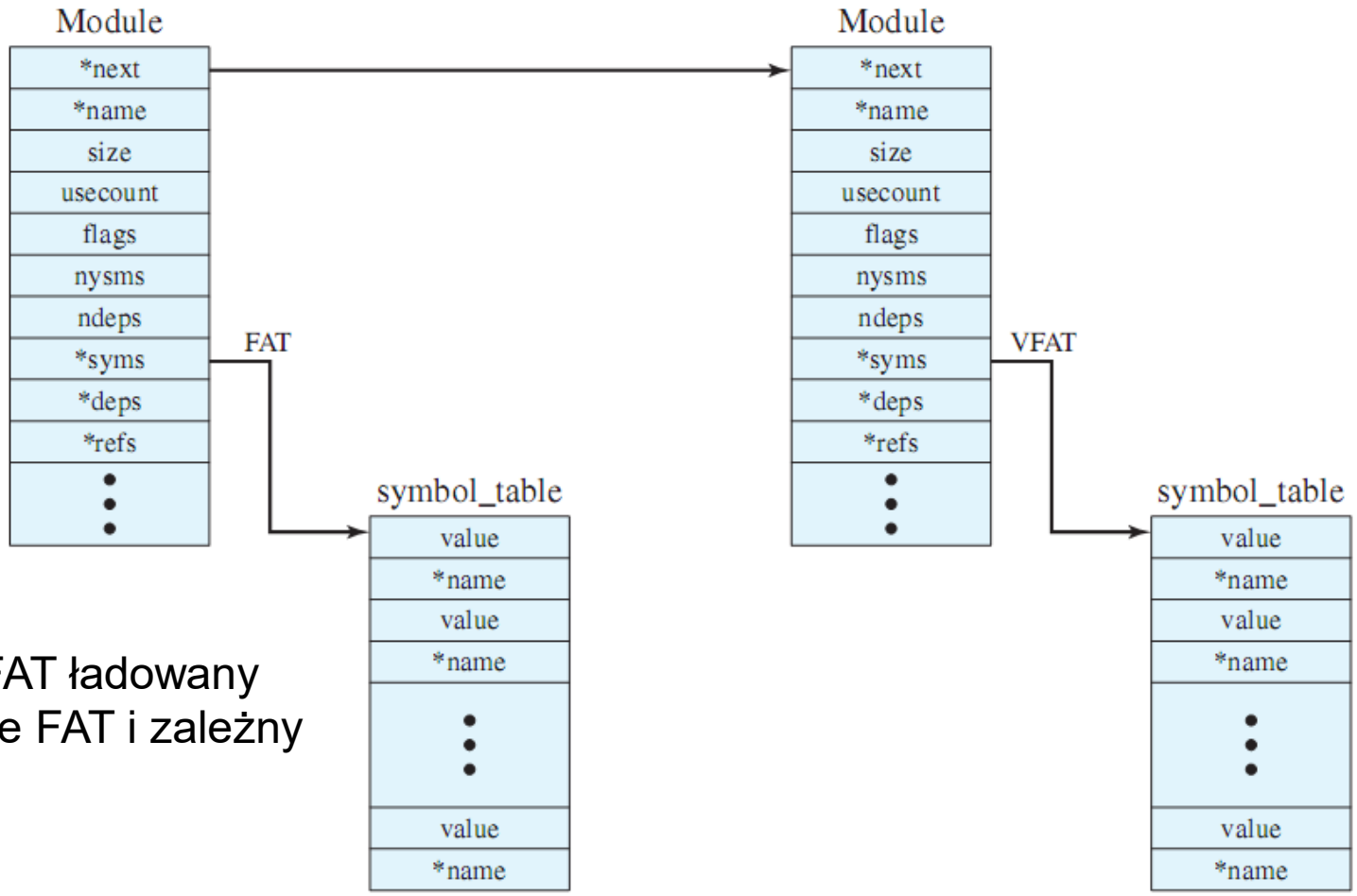
Linux: Modularne jądro monolityczne

- Jądro monolityczne, jest strukturyzowane jako zbiór modułów

- Ładowalne moduły (Loadable modules)
  - Plik obiektowy który może być linkowany i odlinkowywany w czasie wykonania

- Charakterystyki:
  - Dynamiczne linkowanie
  - Stos modułów

Moduły jądra w systemie Linux



Moduł VFAT ładowany po module FAT i zależny od niego

Figure 2.17 **Example List of Linux Kernel Modules**

Moduły jądra w systemie Linux (2)

Każdy moduł jest definiowany przez dwie tablice:

- tablice modułu
- tablicę symboli

Tablica modułu zawiera następujące elementy:

- *next: wskaźnik do następnego modułu. Wszystkie modulł są zorganizowane w listę powiązaną
- *name: wskaźnik do nazwy modułu.
- size: Rozmiar modułu w stronach pamięci,
- usecount: Licznik użycia pamięci
  - Licznik jest zwiększany, gdy operacja dotycząca funkcji modułu jest rozpoczynana oraz dekrementowana, gdy operacja ta się kończy
- flags: Flagi modułu
- nsyms: Liczba eksportowych sygnałów.
- ndeps: Liczba modułów, do których następują referencje.
- *syms: Wskaźnik do tablicy symboli modułu
- *deps: Wskaźnik do listy modułów, do których następują referencje w danym module
- *refs: Wskaźnik do listy modułów, których używa ten moduł.

Tablica symboli definiuje te symbole kontrolowane przez moduł, które są używane gdzie indziej

,

Komponenty jądra Linuxa