Systemy operacyjne

Architektury. Popularne systemy operacyjne

2019/2020

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

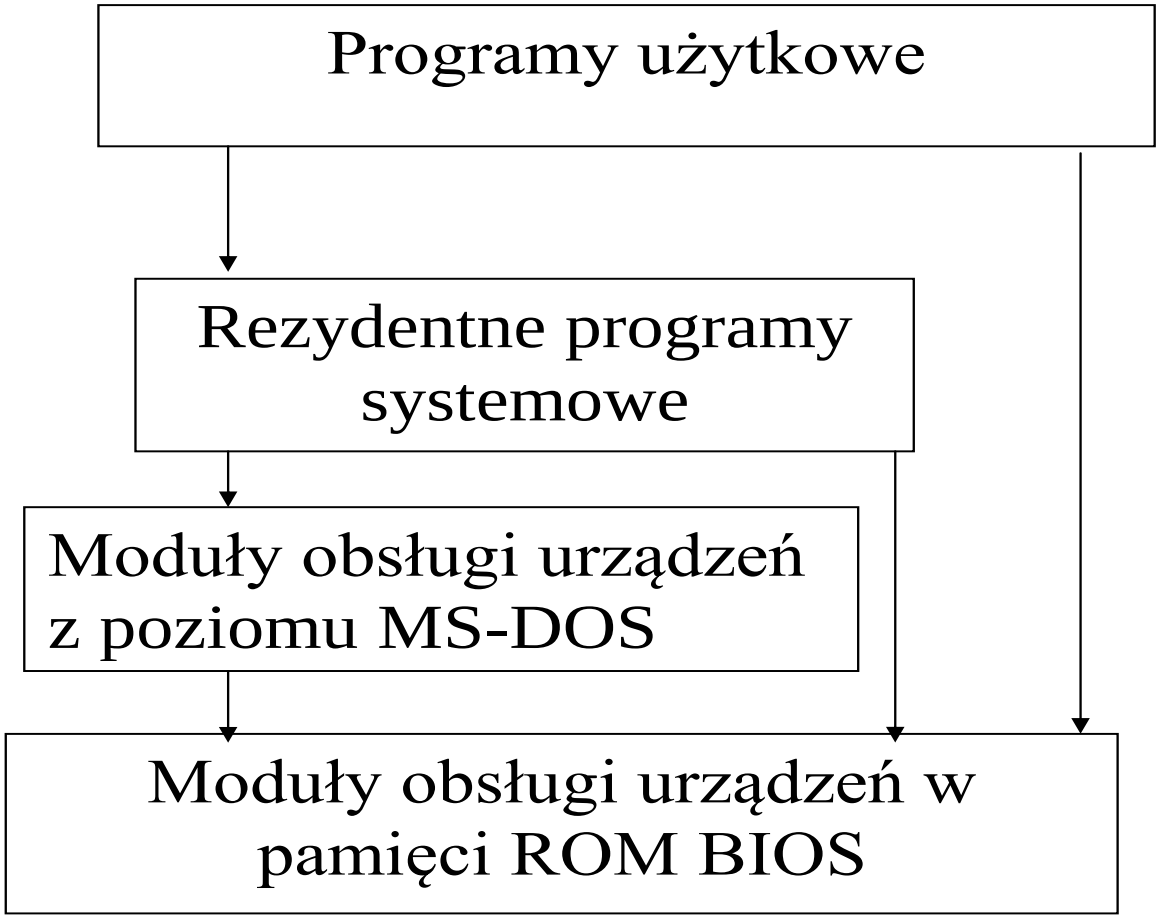
Struktura systemów operacyjnych: Prosta struktura

- 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

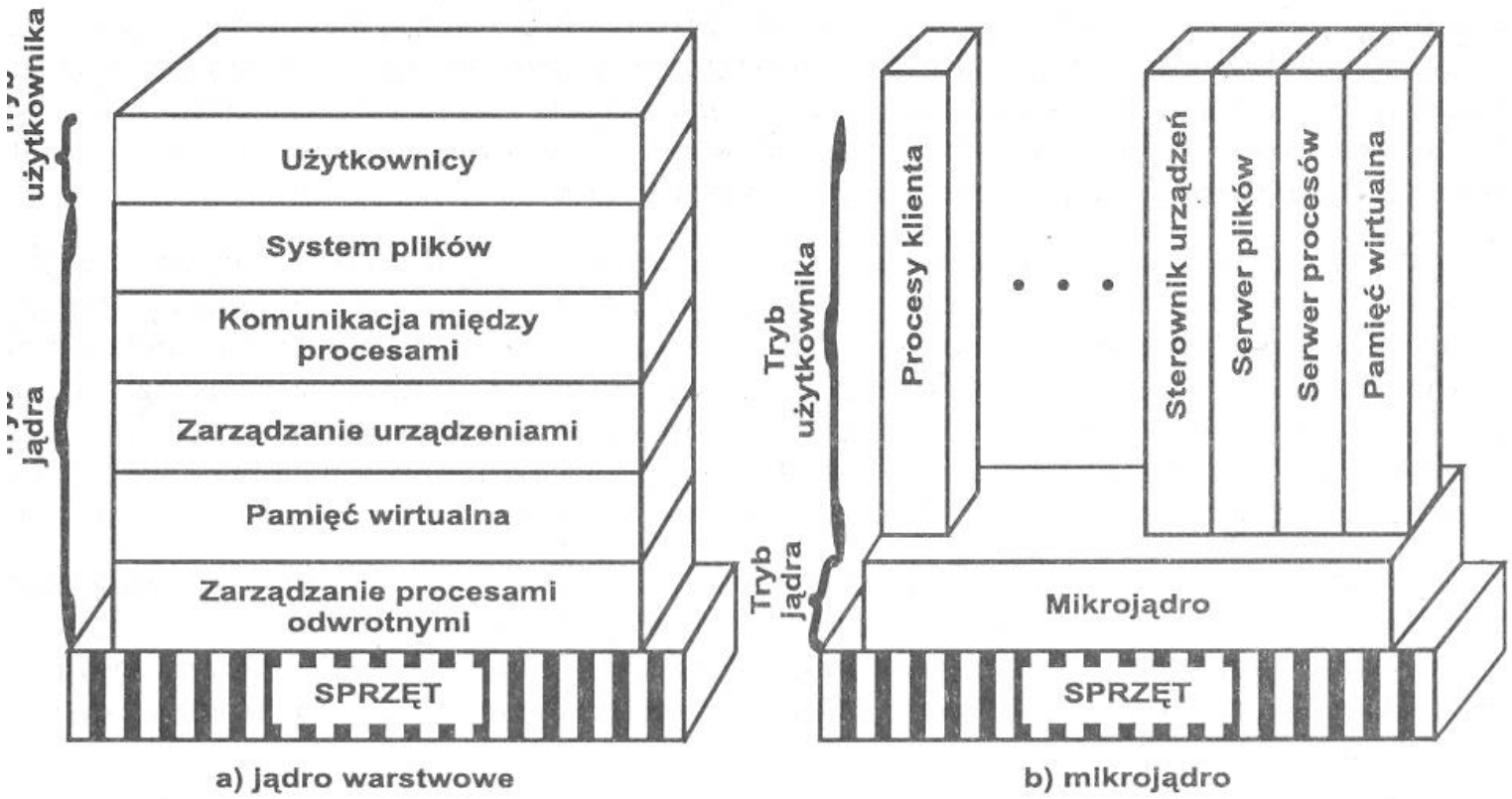
- Trend w nowoczesnych systemach operacyjnych:
  - przesunięcie kodu systemu do wyższych warstw i realizacja wielu funkcji systemowych jako procesy użytkownika
  - maksymalne "odchudzenie" samego systemu operacyjnego i pozostawienie minimalnego jądra

- 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”

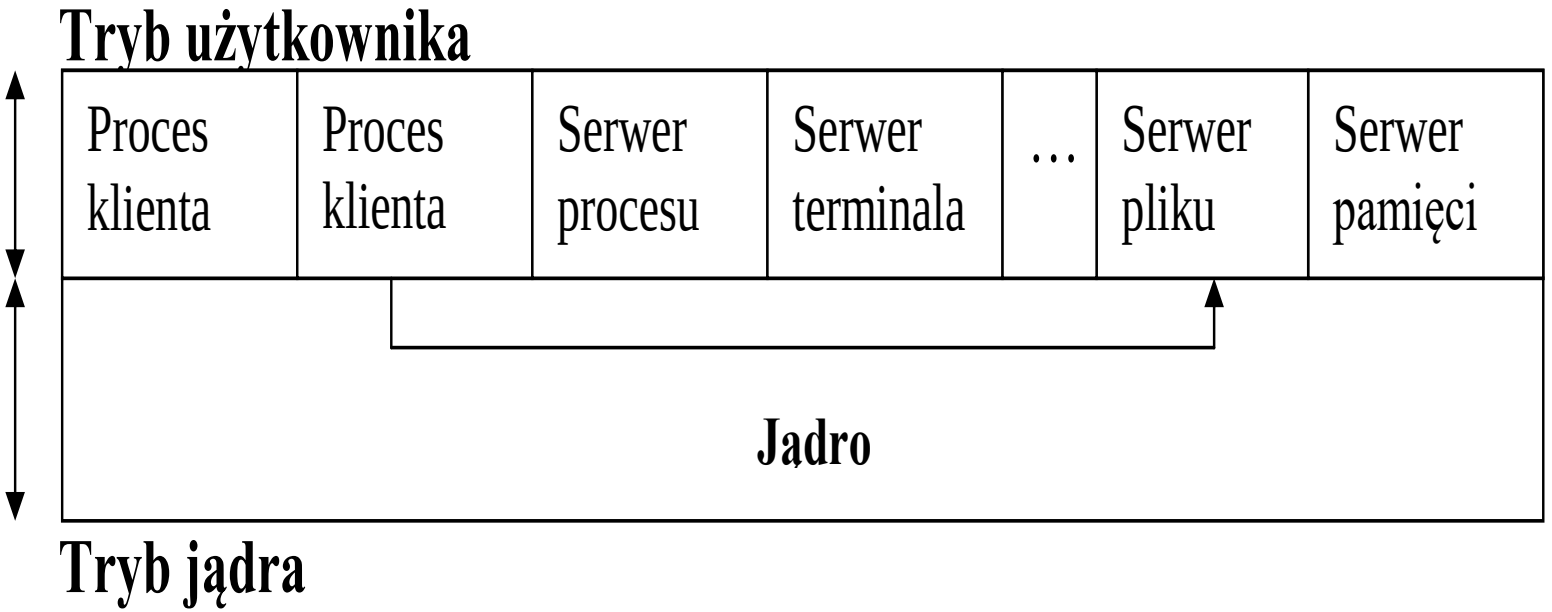
Wady rozwiązań opartych na mikrojądrze

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

Mikrojądro: Struktura klient-serwer

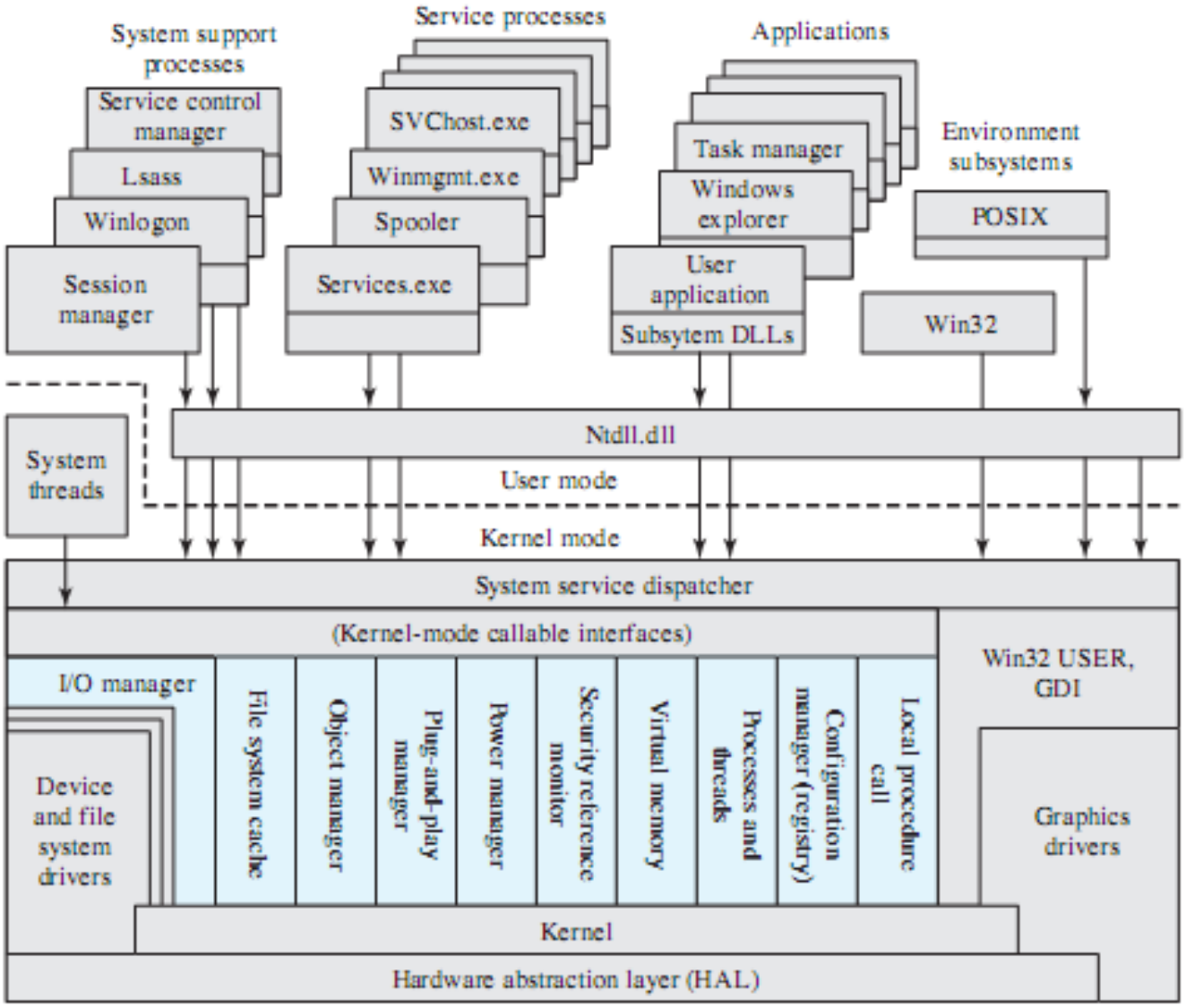
- **Struktura klient-serwer**
  - proces użytkownika (zwany **procesem klienta**) wysyła zlecenia do **procesu serwera**
  - **proces serwera** wykonuje zlecenie
  - **proces serwera** wysyła uzyskany wynik do **procesu klienta**

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



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

Tradycyjne jądro UNIXa

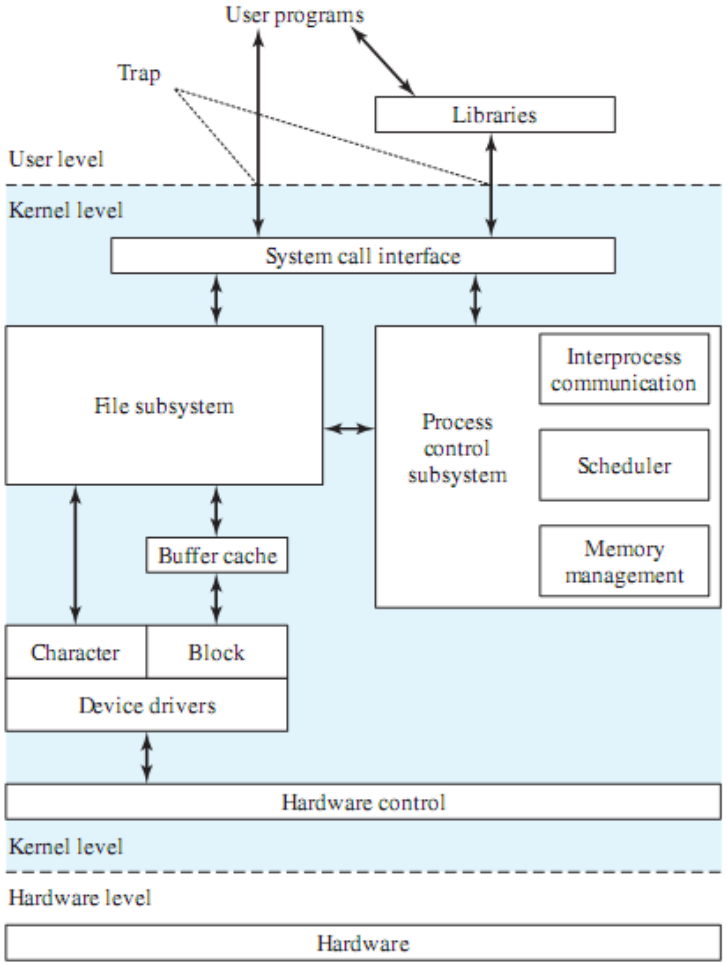


Figure 2.15 **Traditional UNIX Kernel**

System V Release 4 (SVR4)

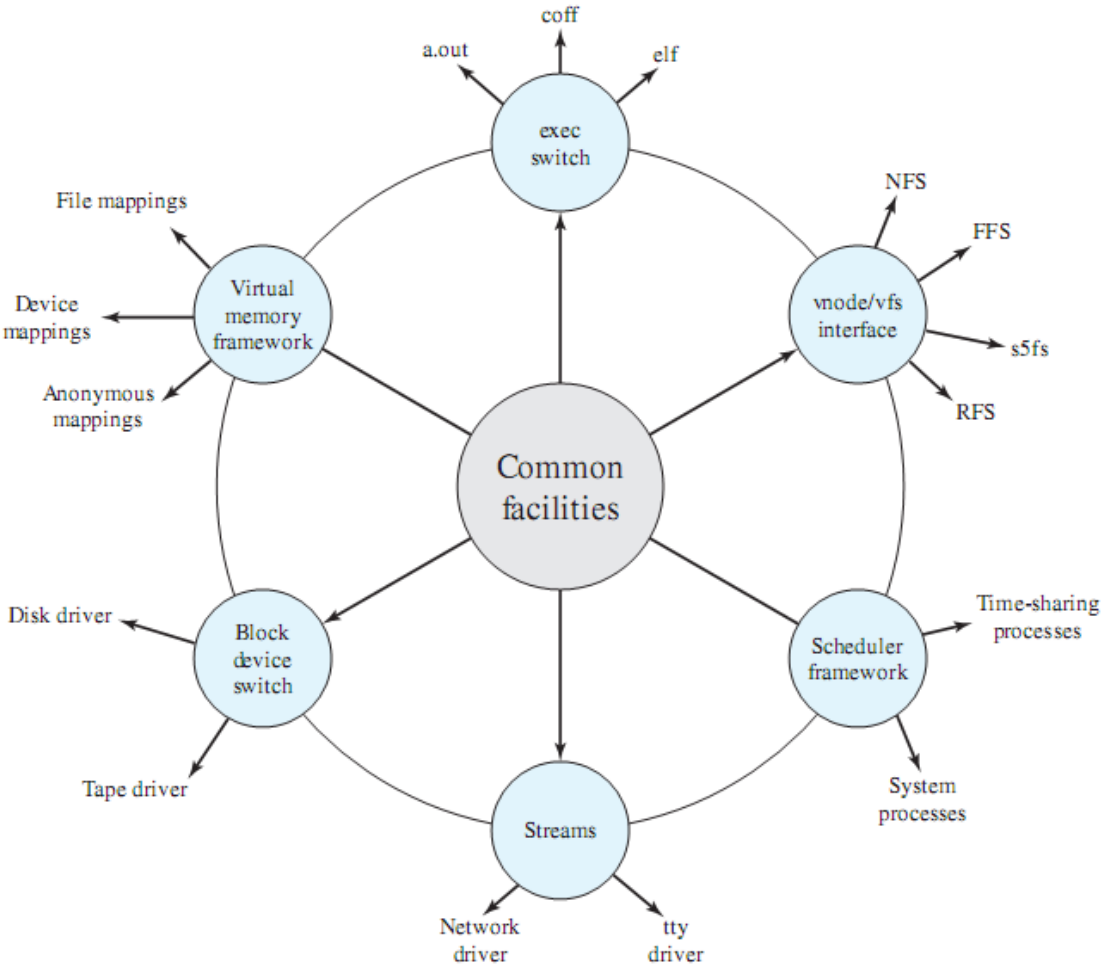
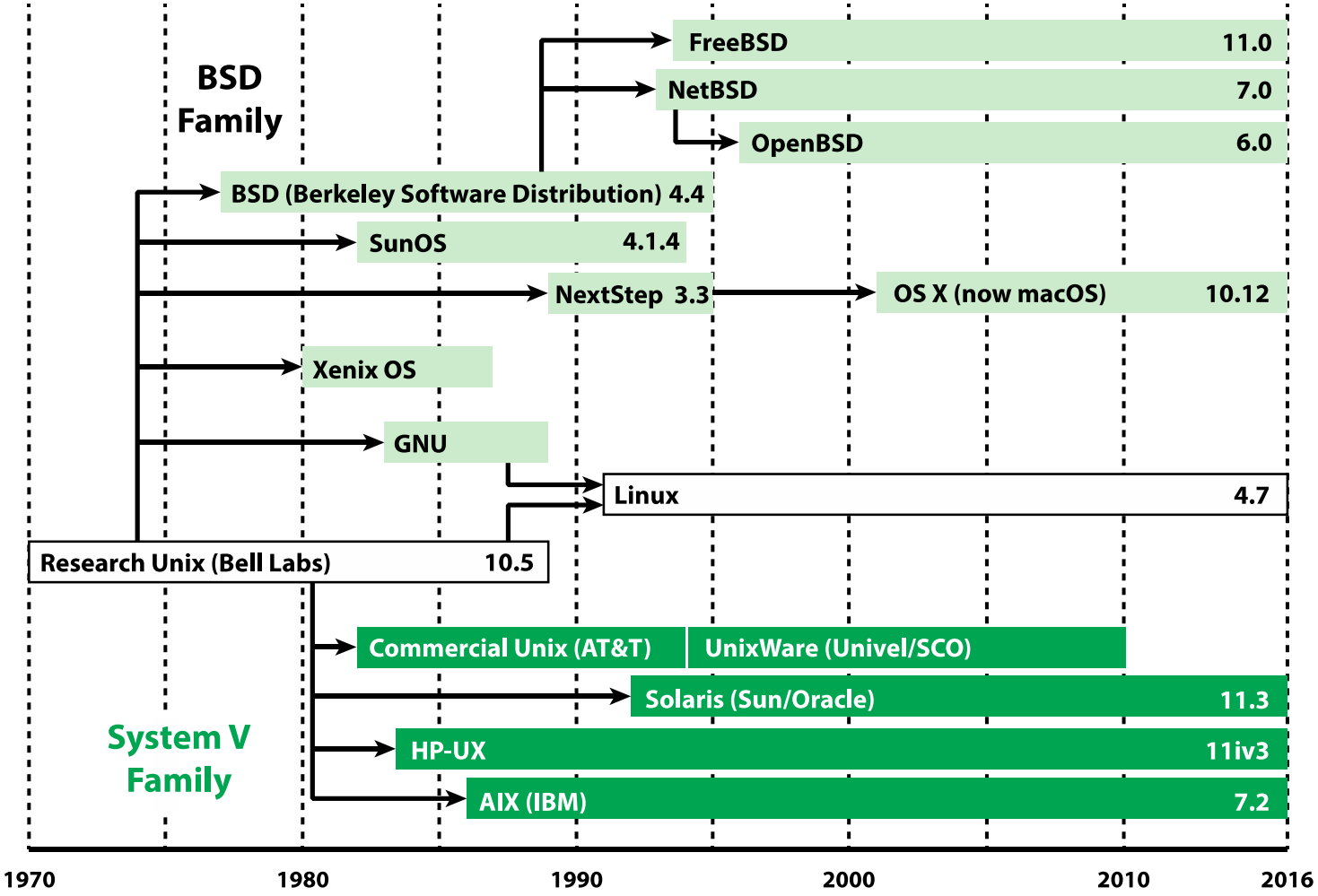


Figure 2.16 **Modern UNIX Kernel**

Ewolucja systemu Unix



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

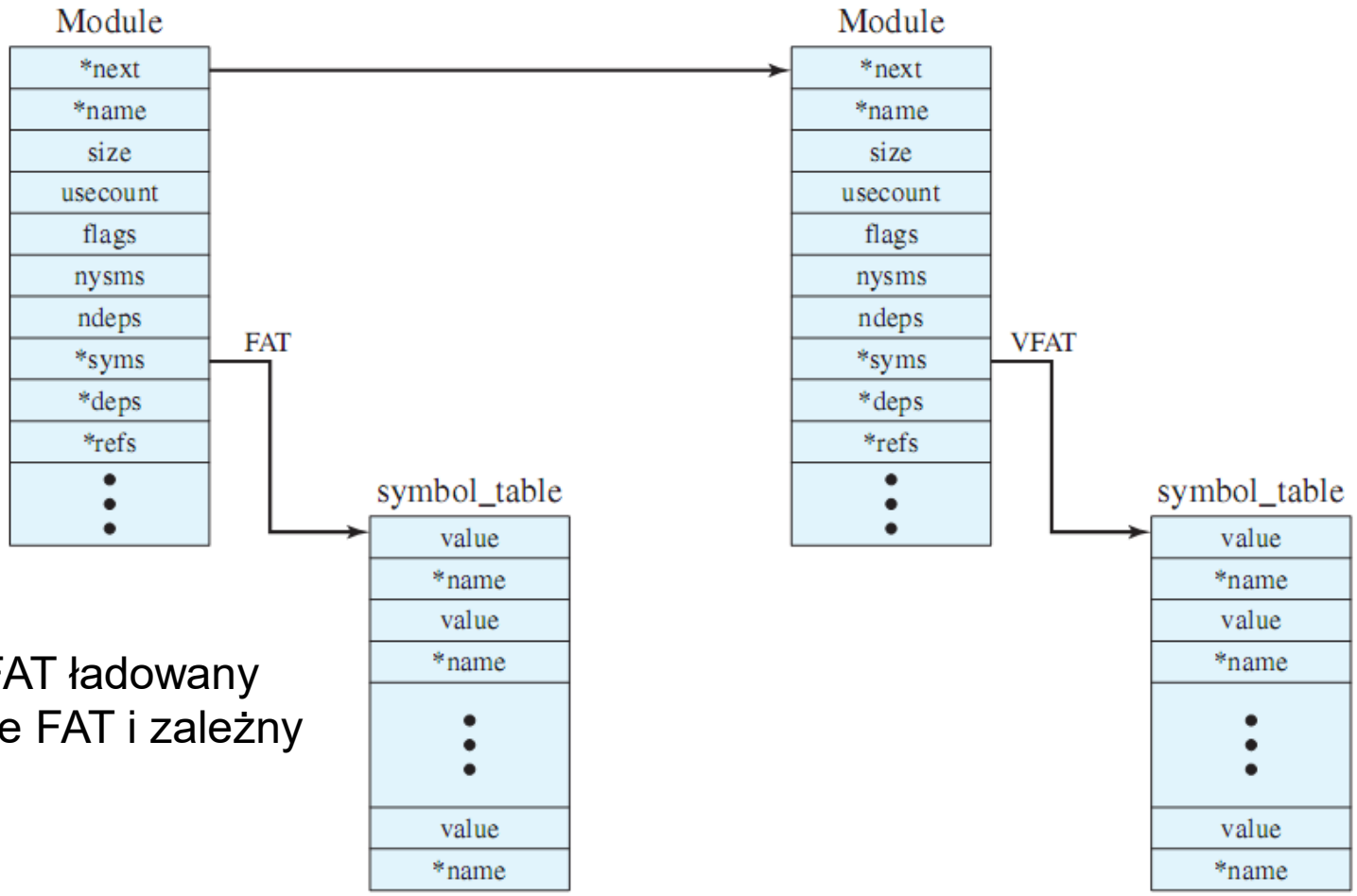
Linux: Modularne jądro monolityczne

- 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

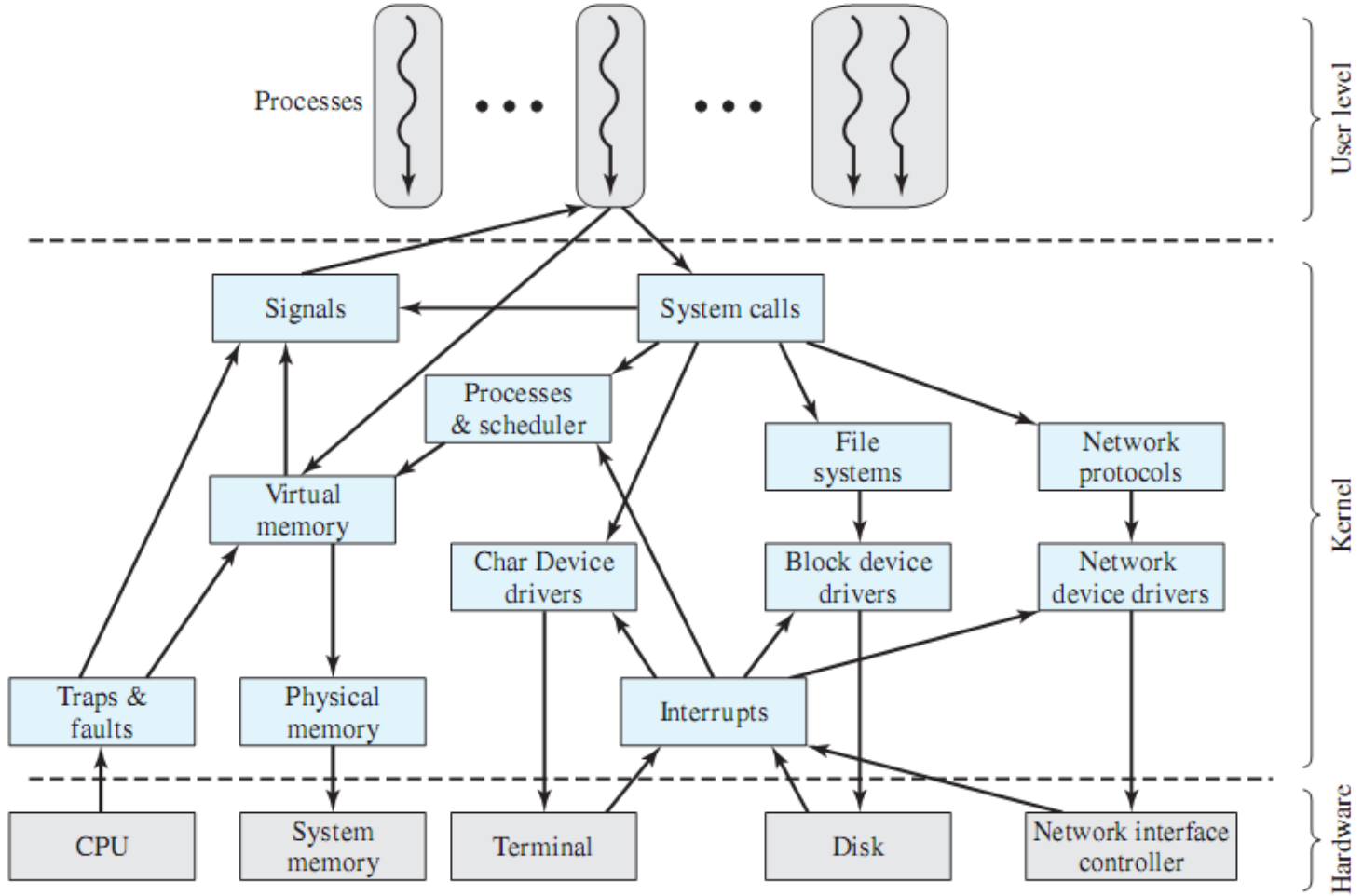


Figure 2.18 **Linux Kernel Components**