Wstęp

- System operacyjny, a system komputerowy
- Zadania i interfejs systemów operacyjnych
- Struktury systemów operacyjnych
- Działanie systemu komputerowego

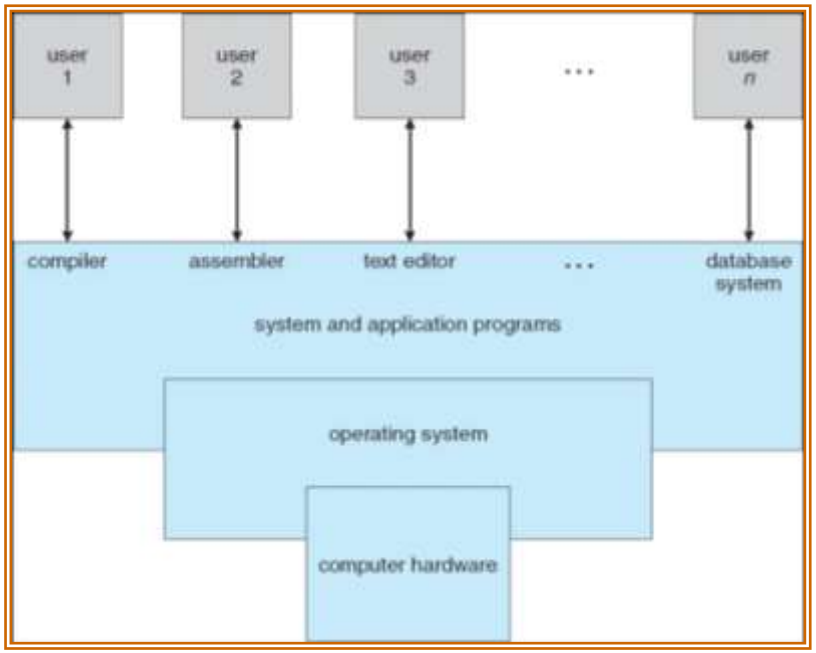
Data ostatniej modyfikacji: 20.09.2018

Co to jest system operacyjny?

- **Program pośredniczący** pomiędzy użytkownikiem systemu komputerowego i samym komputerem (sprzętem).
- **Dystrybutor zasobów** - przydziela zasoby systemu komputerowego i zarządza nimi.
- **Program sterujący** - kontroluje wykonanie programów użytkownika oraz pracę urządzeń wejścia/wyjścia.
- **Jądro** (*kernel*) – jedyny program działający przez cały czas (wszystkie inne programy to aplikacje bądź programy systemowe).
- **Podstawowe oprogramowanie systemu komputerowego**, które pozwala
  - wykonywać programy użytkownika i ułatwiać rozwiązywanie powstających problemów
  - uczynić system komputerowy wygodnym w używaniu
  - wykorzystać sprzęt jak najbardziej efektywnie
- Całość oprogramowania umożliwiającego:
  - zarządzanie sprzętowymi i programowymi zasobami systemu komputerowego
  - przekształcenie maszyny rzeczywistej w maszynę wirtualną o cechach wymaganych przez przyjęty tryb przetwarzania.

Składowe systemu komputerowego

- **Sprzęt** (*hardware*) – dostarcza podstawowych zasobów systemowi (procesor, pamięć, urządzenia wejścia/wyjścia).
- **System operacyjny** - zarządza i koordynuje wykorzystanie sprzętu przez różnorodne programy aplikacyjne użytkowników.
- **Programy aplikacyjne** - określają w jaki sposób należy użyć zasobów systemu dla rozwiązania zadań określonych przez użytkownika (kompilatory, systemy baz danych, gry, programy biurowe).
- **Użytkownicy** (ludzie, maszyny, inne komputery).



Tryby pracy systemu komputerowego

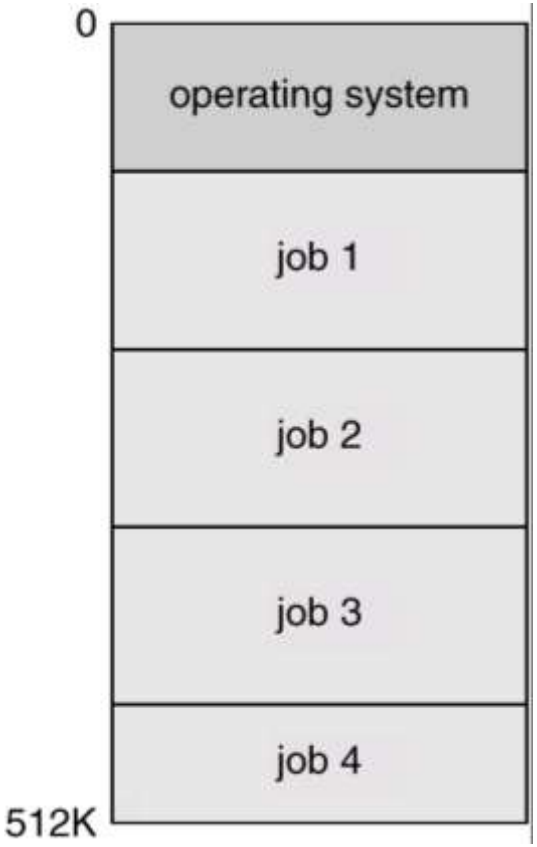
- Podstawowym czynnikiem, mającym wpływ na konstrukcję systemu operacyjnego, jest dominujący tryb pracy systemu komputerowego

- Główne tryby pracy:
  - **pośredni**, wsadowy (*off-line, batch*)
  - **bezpośredni**, interakcyjny (*on-line*)
  - **w czasie rzeczywistym** (*real-time*)

- W rzeczywistych systemach komputerowych występować mogą różne tryby pracy.

Wieloprogramowe systemy wsadowe

- Kilka zadań jest przechowywanych w pamięci operacyjnej
- Procesor jest przełączany pomiędzy tymi zadaniami przez planistę gdy wykonywane zadanie się zakończy, albo zamówi operację wejścia/wyjścia
- Wymagania wieloprogramowania (*multiprogramming*):
  - **Zarządzanie pamięcią** (memory management) – system musi przydzielać pamięć operacyjną kilku zadaniom chroniąc przed nieuprawnionym dostępem.
  - **Planowanie przydziału procesora** do zadań gotowych do wykonania (*CPU scheduling*) jest sterowane poleceniami (powłoka, *shell*).
  - Współdzielone **urządzenia dostępne za pomocą jednolitych procedur wejścia/wyjścia**.



Systemy z podziałem czasu i interakcyjne

- Systemy **wielozadaniowe** (*multitasking*) **z podziałem czasu** (*time-sharing*)
- Procesor jest przełączany pomiędzy kilkoma zadaniami, które są przechowywane w pamięci operacyjnej i na dysku (procesor może być bezpośrednio przydzielany jedynie do zadania będącego w pamięci operacyjnej). Przełączanie odbywa się w regularnych odstępach czasu, zazwyczaj na tyle często że zadania wydają się (użytkownikowi) wykonywać współbieżnie.
- Zadanie może być przesuwane („wytaczane”, ang. *swapped out*) z pamięci operacyjnej na dysk, a później przywracane z dysku do pamięci operacyjnej („wtaczane”, ang. *swapped in*).
- Dodatkowe własności systemów interakcyjnych:
  - Istnieje bezpośrednia komunikacja pomiędzy użytkownikiem i systemem; system po zakończeniu realizacji polecenia oczekuje następnego polecenia z klawiatury.
  - Czas odpowiedzi powinien być krótki (<1s) z małą wariancją.

Systemy czasu rzeczywistego

- Systemy czasu rzeczywistego (*Real Time* - RT) mają dobrze określone, stałe **ograniczenia czasowe** odpowiedzi na zdarzenia pochodzące z zewnątrz systemu.
- System **rygorystyczny** (*hard RT system*).
  - Ograniczenia czasowe muszą być **gwarantowane** (nieprzekraczalne). Zastosowanie w sterowaniu przemysłowym, robotyce itp..
  - Wymagania RT są sprzeczne z wymaganiami systemu z podziałem czasu ➔ systemy rygorystyczne nie są realizowanie w postaci systemu uniwersalnego (łączącego różne tryby pracy)
  - Sztywne gwarancje czasowe eliminują konstrukcje, które redukują koszty systemu przez wzrost/zmienność czasu realizacji (np. pamięć wirtualna)
- System **łagodny** (*soft RT system*)
  - Planowanie nastawione na dotrzymanie terminów, ale rzadkie przekraczanie ograniczeń czasowych jest dopuszczalne
  - Użyteczny w zastosowaniach (multimedia, *virtual reality*) wymagających zaawansowanych funkcji systemu operacyjnego.

Wstęp

1. System operacyjny, a system komputerowy
2. Zadania i interfejs systemów operacyjnych
3. Struktury systemów operacyjnych
4. Działanie systemu komputerowego

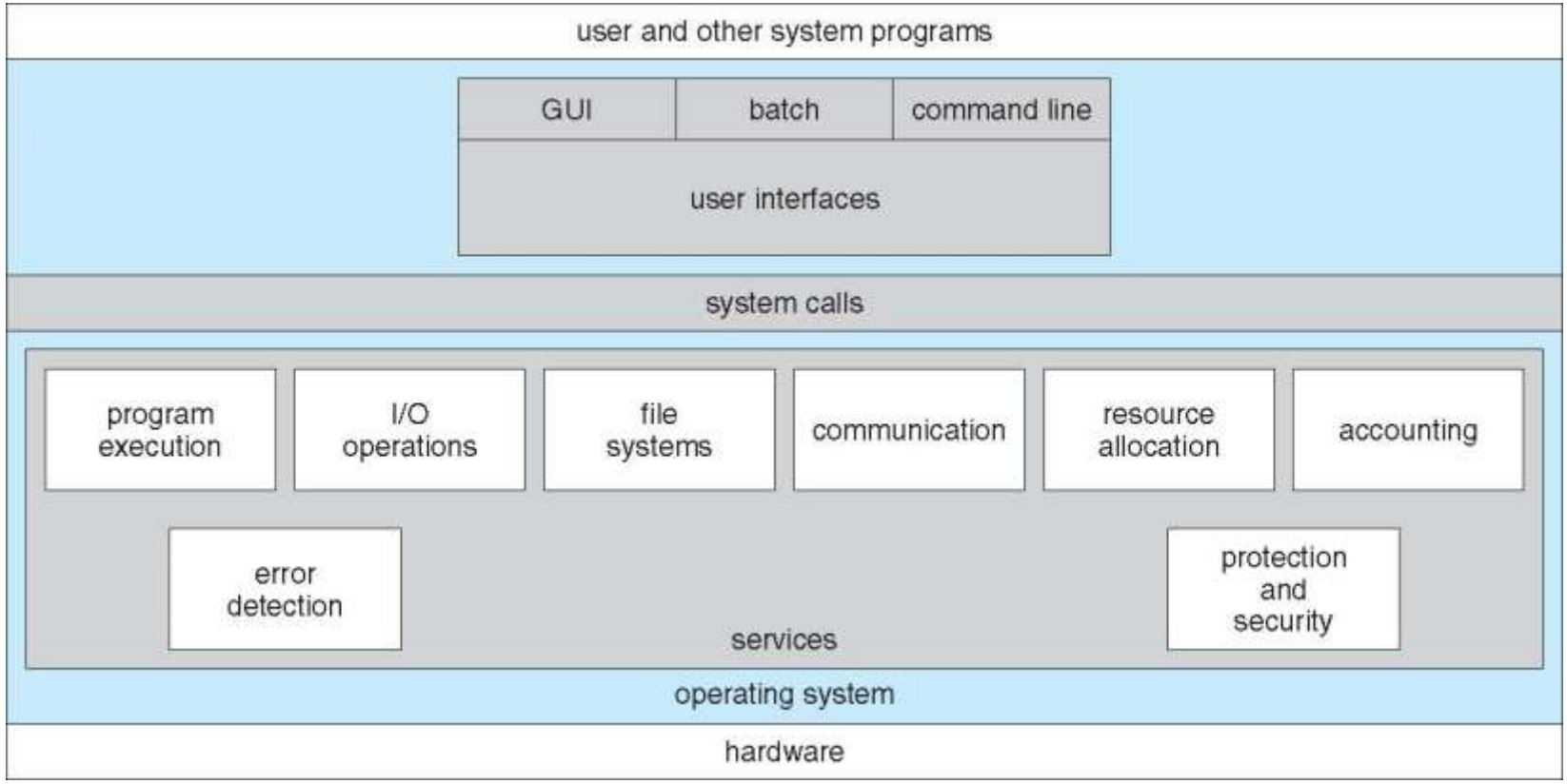
Zadania systemu operacyjnego

- Wykonywanie programów użytkownika w zadowalający ich sposób oraz ochrona danych użytkownika
  - Użytkownicy zazwyczaj chcą wygody, łatwości użytkowania i dobrej wydajności (ale niekoniecznie efektywnego wykorzystania zasobów).
  - Komputer współdzielony musi próbować zadowolić wszystkich użytkowników
  - Dla komputerów kieszonkowych (i innych o ograniczonych zasobach) ważna jest użyteczność i pobór energii
- Efektywne **zarządzanie** zasobami systemu komputerowego:
  - Przydział i odzyskiwanie zasobów
  - Planowanie dostępu do zasobów
  - Ochrona i autoryzacja dostępu do zasobów
  - Rozliczanie użytkowników z wykorzystania zasobów
  - Obsługa błędów

Zasoby systemu komputerowego

- Zasoby zarządzane przez system operacyjny
  - Procesor(y), rdzenie
  - Pamięci i inne urządzenia systemu komputerowego
  - Informacja przechowywana w systemie
- Zestaw zasobów zależy od rodzaju środowiska obliczeniowego, które system tworzy czy współtworzy
  - System jednostanowiskowy ogólnego przeznaczenia
  - Portal
  - Komputery sieciowe (cienki klient)
  - Obliczenia rozproszone (i sieciowe systemy operacyjne)
  - Obliczenia klient-serwer (asymetria relacji: klient-serwer)
  - Sieci partnerskie (P2P, *peer-to-peer*)
  - Serwer udostępniające maszyny wirtualne
  - Obliczenia w chmurze (wirtualizacja zasobów w sieci)
  - Systemy wbudowane (*real-time embedded systems*)

Podstawowe usługi systemu operacyjnego

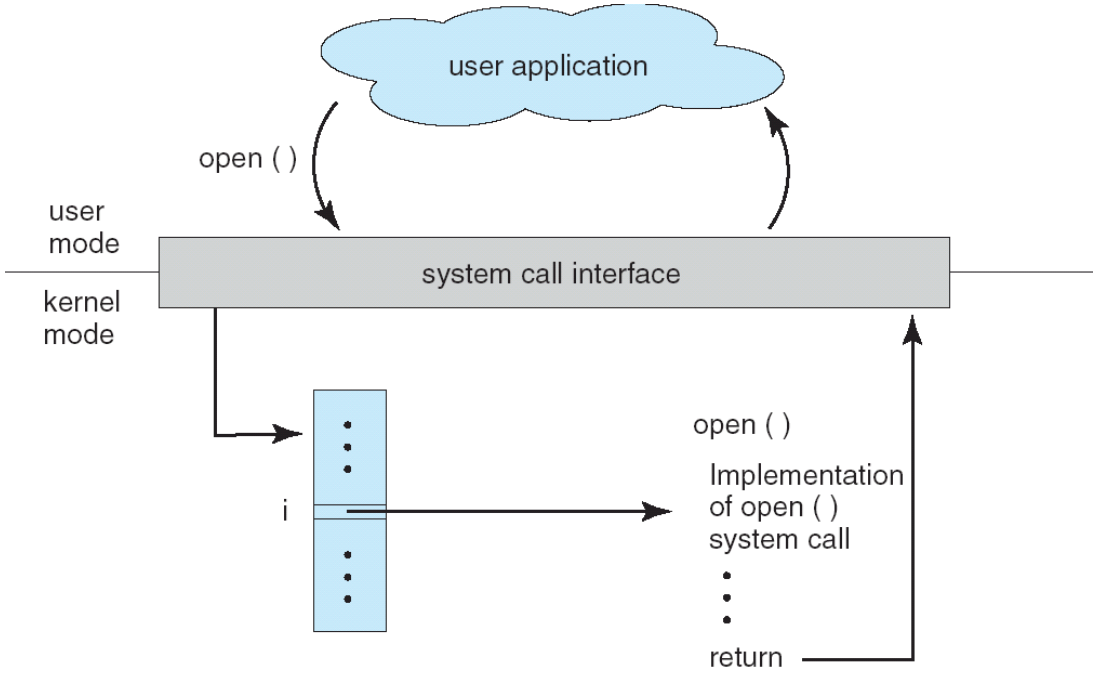


Współbieżność dostępu do zasobów

- **Wieloprogramowość** i praca z **podziałem czasu** wymagają bezpiecznego i efektywnego dzielenia zasobów systemu operacyjnego pomiędzy wiele różnych programów (procesów), użytkowników.
- Realizacja współbieżności dostępu do zasobów związana jest ze stosowaniem:
  - Przerwań
  - Środków systemowej komunikacji i synchronizacji
  - Architektur wielordzeniowych/wieloprocesorowych
- Stopień współbieżności zależy nie tylko od sprzętu (rdzenie/procesory), ale i własności systemu operacyjnego:
  - Symmetric Multiprocessing (**SMP**) – obciążenie zadaniami użytkownika i jądra rozłożone pomiędzy procesorami/rdzeniami.
  - Asymmetric Multiprocessing – układ master-slaves
- Dostęp do wspólnych zasobów odbywa się przez **zamawiane dostępu u systemu operacyjnego** za pomocą **funkcji systemowych**.
- Wywołanie systemowe (wywołanie funkcji systemowej) jest ogólną metodą zlecania systemowi wykonania operacji bezpośrednio niedostępnej.
- Funkcje systemowe tworzą interfejs (**API**) pomiędzy wykonującym się programem i systemem operacyjnym

Funkcja API a funkcja systemowa

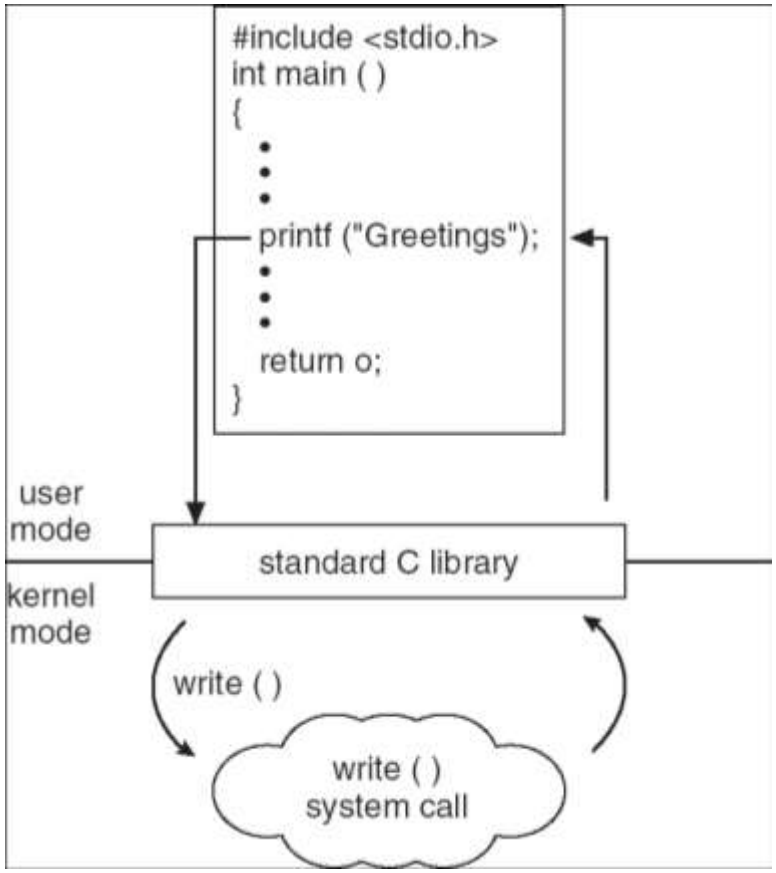
- Funkcje systemowe są dostępne bezpośrednio z asemblera.
- Języki wysokiego poziomu (np. C) pozwalają wywoływać funkcje systemowe za pomocą:
  - funkcji standardowych (specyficznych dla języka) lub
  - podprogramów bibliotecznych, tworzących interfejs jądra (Application Program Interface, API). Popularne API: Win32, POSIX API, JAVA API



Wywoływanie f. systemowej za pośrednictwem biblioteki języka C

Program w języku C wywołuje funkcję **printf**() ze standardowej biblioteki C. ➔

Funkcja ta powoduje zapisanie sformatowanego tekstu do lokalnego bufora strumienia standardowego wyjścia (*stdout*). Kiedy bufor jest zapełniony (lub wywołano funkcję opróżniania bufora: **fflush()**) zostaje wywołana funkcja systemowa (**write**() dla API POSIX) , powodująca przeniesienie zawartości bufora do urządzenia skojarzonego ze standardowym wyjściem.



Przykłady funkcji API Win32 i POSIX

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

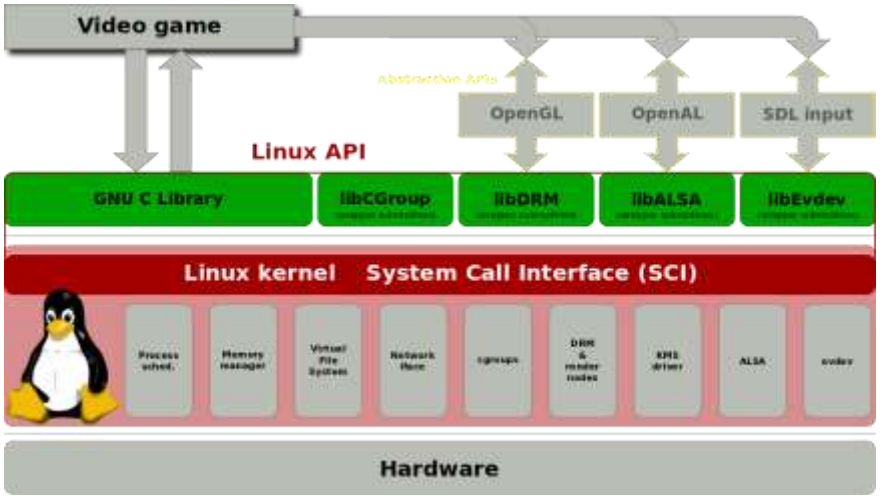
Wybrane standardy API systemów operacyjnych

- UNIX itp.:
  - SVID (System V Interface Definition)
  - 4.x BSD (Berkeley Software Distribution)
  - POSIX (Portable Operating System Interface) – IEEE, ISO, The Open Group. Stan na 2018 r.: **POSIX.1-2017** (IEEE Std 1003.1-2017)
    - **Base Definitions (XBD)**
    - **Shell and Utilities (XCU)**
    - **System Interfaces (XSH)**
    - **Rationale (XRAT)**

    Oddzielny składnik standardu: **POSIX Conformance Test Suite**
  - Single Unix Specification (SUS) – rodzina standardów, które musi spełniać system zawierający w nazwie UNIX: (93, 95, 98, 03). W skład wchodzą składniki standardu POSIX oraz **POSIX Certification Test Suite)** i specyfikacja interfejsu terminali **CURSES**. Łącznie: 1742 interfejsów.

- win16/win32/win64/winCE – API dla różnych wersji systemów operacyjnych Microsoft Windows (WinAPI)

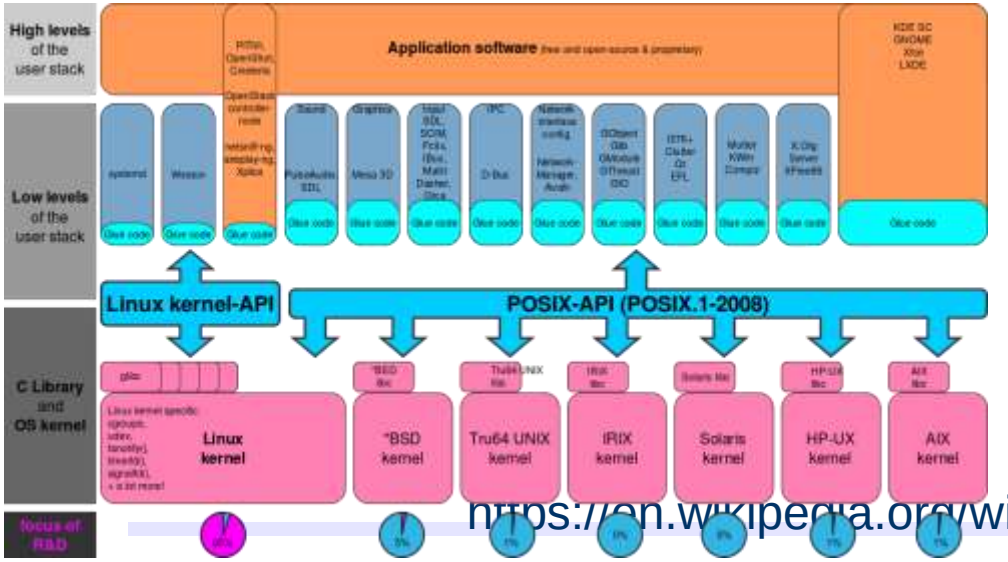
Interfejsy jądra systemu Linux

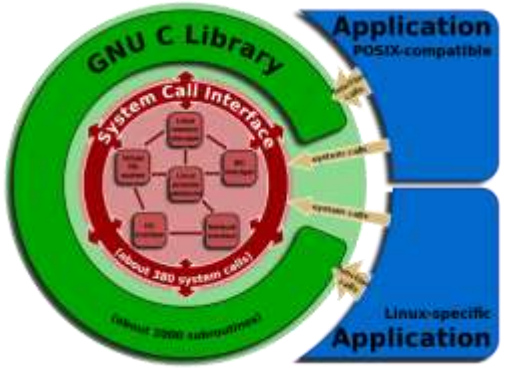


Linux API:

- Kernel internal API
- Kernel-user space API:
  - System Call Interface +
  - GNU C Library (glibc) wrapper

Linux API a POSIX API





https://en.wikipedia.org/wiki/Linux_kernel_interfaces

Wstęp

1. System operacyjny, a system komputerowy
2. Zadania i interfejs systemów operacyjnych
3. Struktury systemów operacyjnych
4. Działanie systemu komputerowego

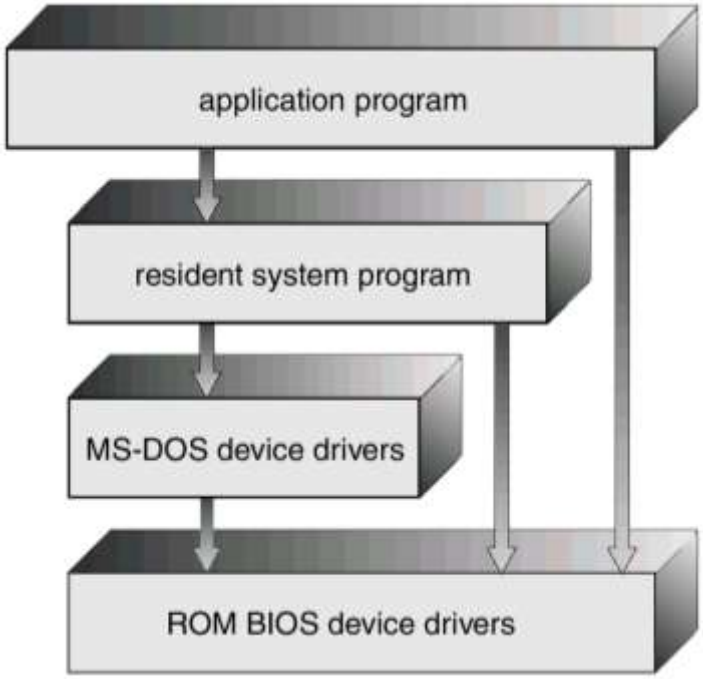
Struktury systemów operacyjnych

- Są znane różne podejścia do strukturyzacji ogromnego kodu składającego się na system operacyjny.
- Niektóre sposoby strukturyzacji jądra
  - prosta struktura
  - jądro monolityczne + programy systemowe
  - struktura warstwowa
  - mikrojądro + kod systemowy dla trybu użytkownika
  - podejście hybrydowe

Struktura prostego systemu operacyjnego

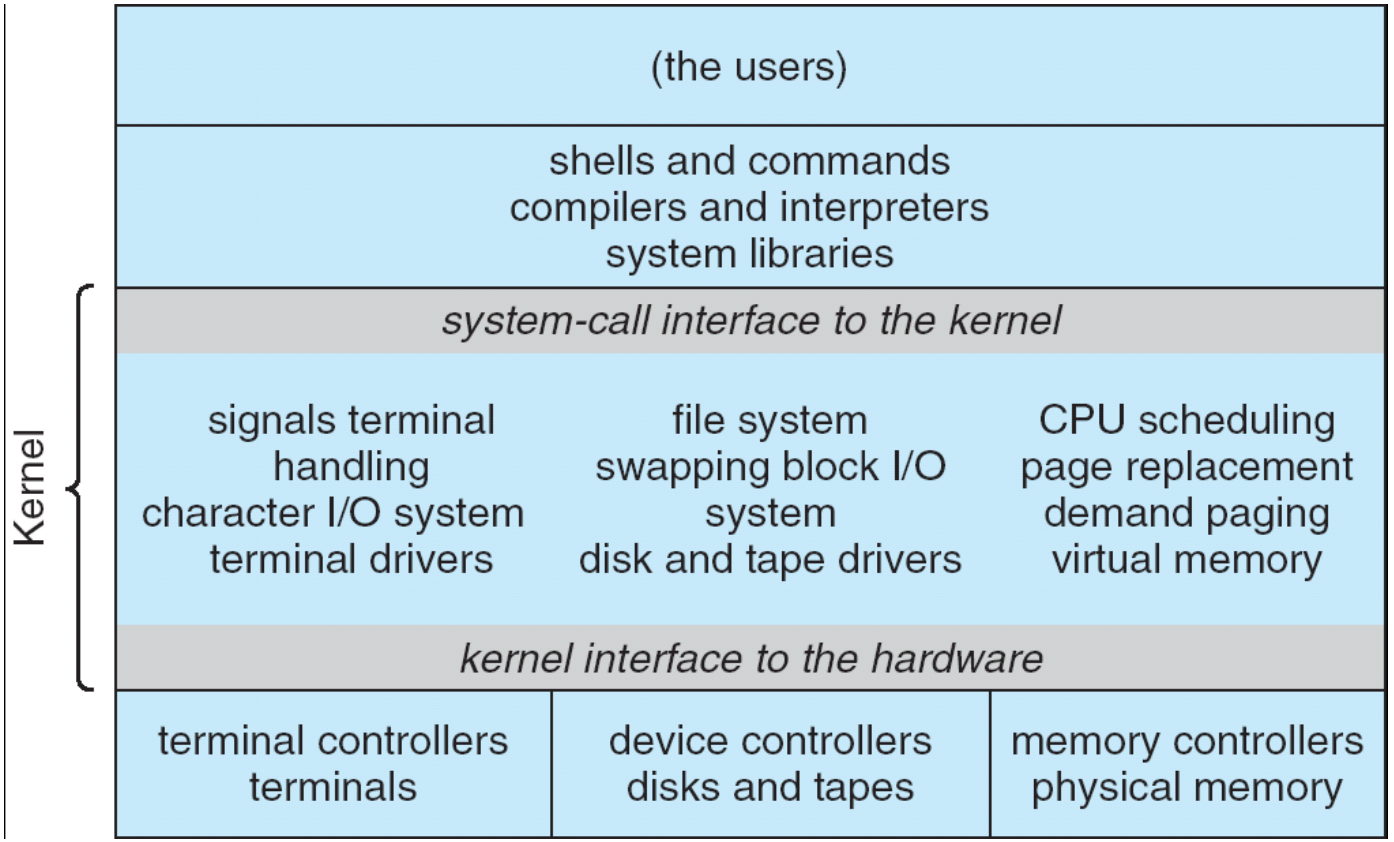
- System MS-DOS napisano tak, by osiągnąć maksymalną funkcjonalność przy oszczędności miejsca. Stąd:
  - nie zadbano o staranną modularyzację,
  - interfejsy i poziomy funkcjonalne nie są wyraźnie rozdzielone.

Warstwowa struktura MS-DOS



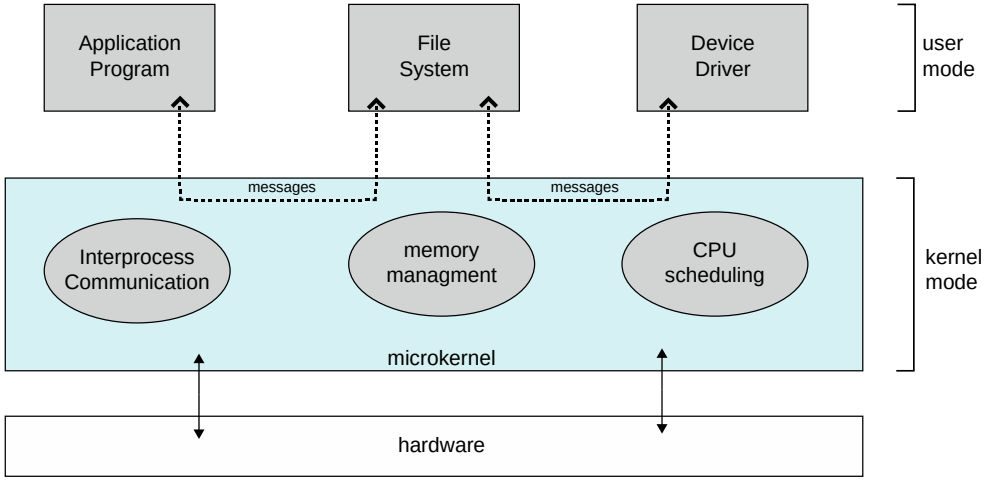
Struktura (monolitycznego) systemu UNIX

- Ograniczona strukturyzacja. Dwie rozdzielne części systemu:
- jądro
- programy systemowe (zwykle łączone potokowo)



System z mikrojądrem

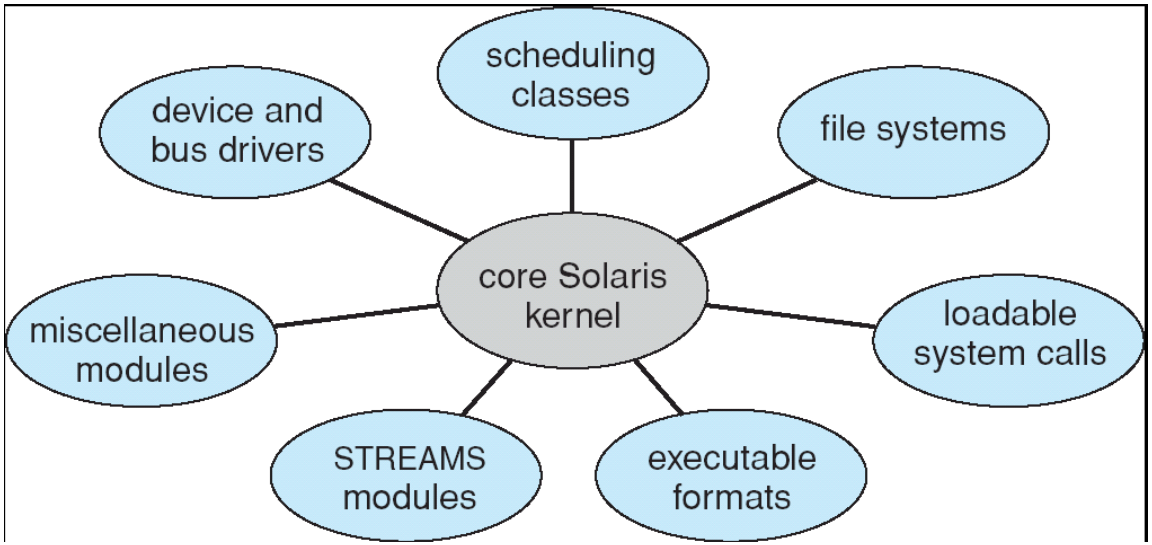
- Maksymalnie dużo kodu, które tradycyjnie było w jądrze, jest przeniesione do modułów wykonywanych w trybie nieuprzywilejowanym (user space)
- Komunikacja pomiędzy modułami: **kolejki komunikatów** (**message passing)**
- Cechy:
  - **+** („łatwa”) rozszerzalność funkcjonalności jądra
  - **+** („łatwe”) przenoszenie systemu na nowe architektury
  - **+** większa niezawodność i bezpieczeństwo (mniej kodu dla trybu uprzywilejowanego)
  - **-** narzut czasowy na przełączanie trybu procesora przy komunikacji z mikrojądrem



Systemy z modułami jądra

- Współczesne systemy operacyjne udostępniają mechanizm modułów jądra:
  - Podejście zorientowane obiektowo
  - Każdy z komponentów jest ładowany oddzielnie – stosownie do aktualnej potrzeby
  - Moduły komunikują się przez dobrze zdefiniowany interfejs

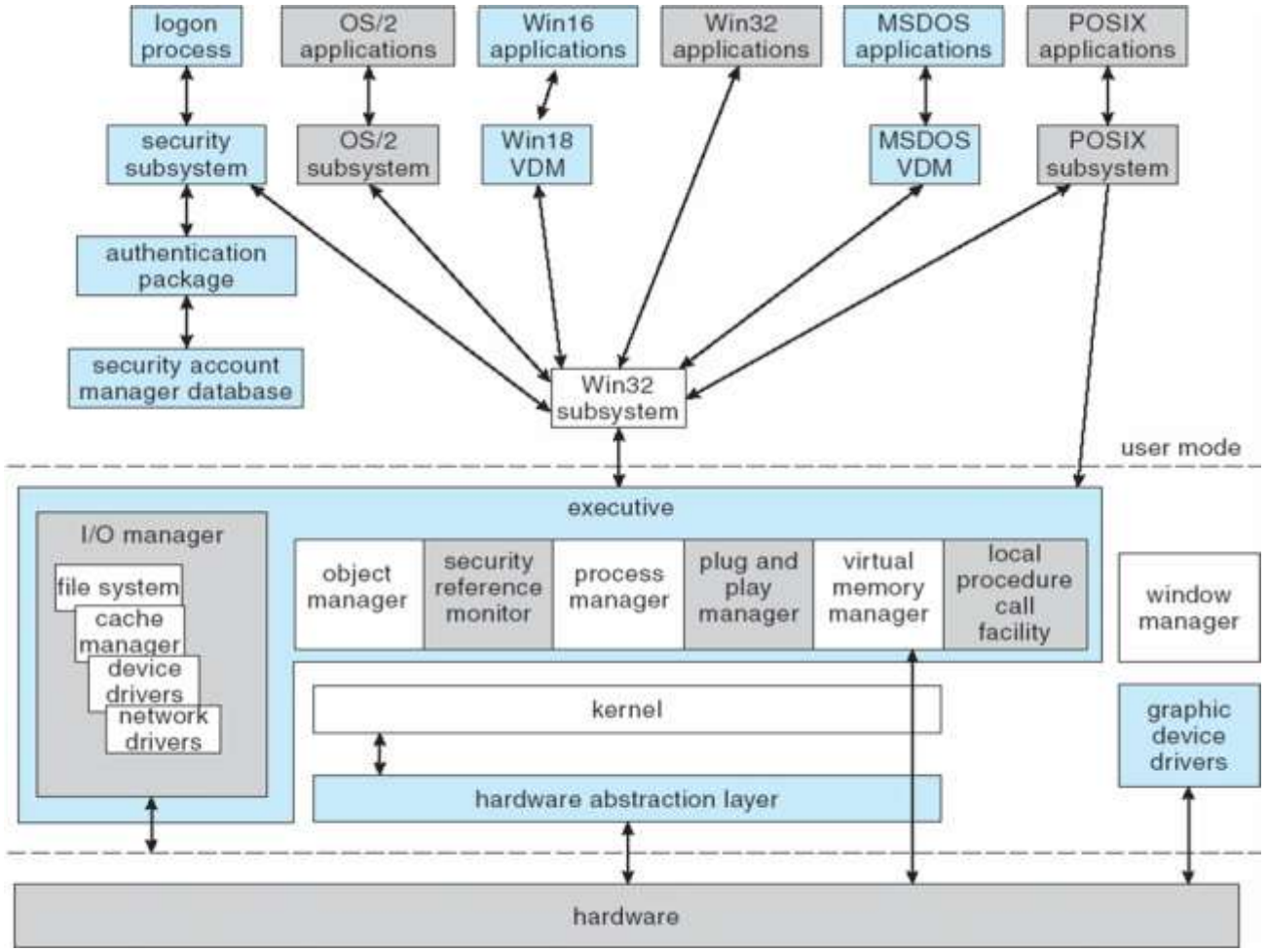
System Solaris



Systemy hybrydowe

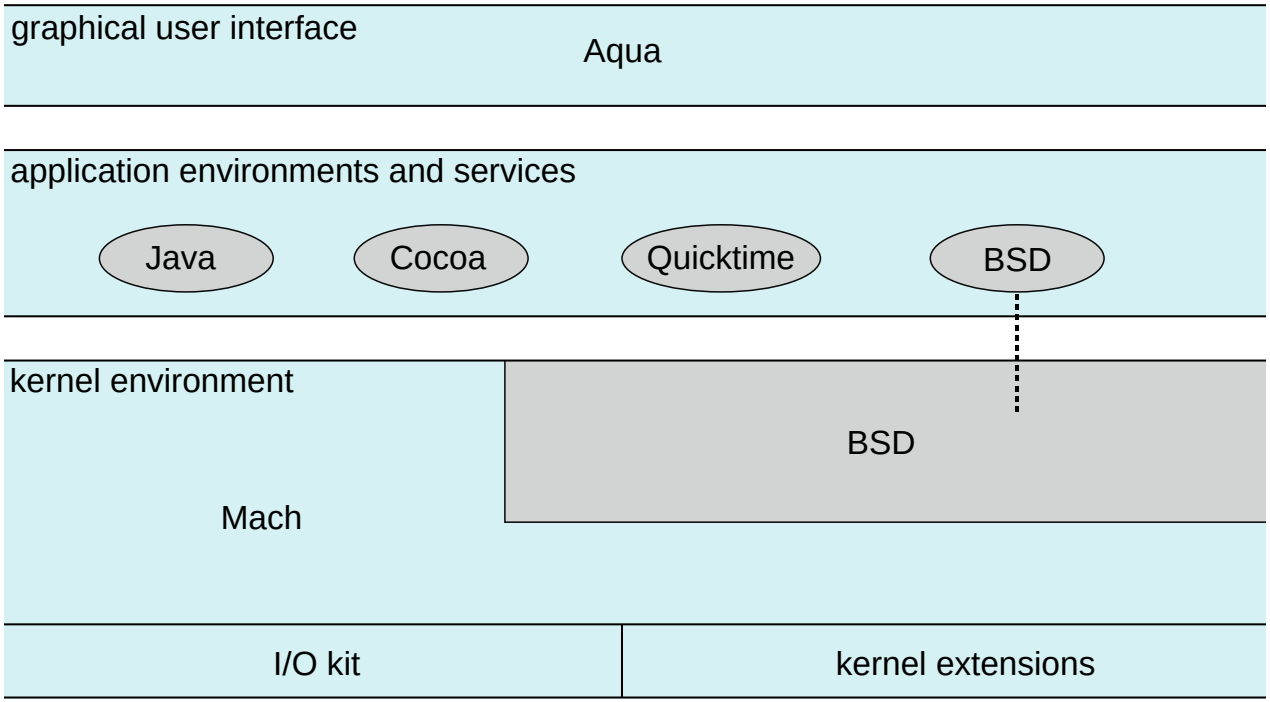
- Spotykane systemy operacyjnego zazwyczaj łączą kilka czystych modeli struktur aby rozwiązać problemy wydajności, bezpieczeństwa, użyteczności jak najlepiej.
  - Jądro Linuxa działa w trybie uprzywilejowanym; zwykle jest modularne z modułami ładowanymi statycznie bądź dynamicznie. Są też moduły, które udostępniają wybrane funkcjonalności jądra w trybie użytkownika *(userspace)*
  - Windows – struktura warstwowa: mikrokernel, egzekutor, moduły realizujące różne środowiska wykonawcze (np. win32, POSIX).
  - Apple Mac OS X hybrida, warstwowa. **Aqua** UI ze środowiskiem programistycznym **Cocoa.** W warstwie niższej mikrojądro Mach oraz jądro BSD i dynamicznie ładowane moduły (**kernel extensions**)

Struktura systemu Windows 7,…



Z: Silberschatz, Galvin, Gagne, Operating System Concepts, 9th ed., 2013

Struktura systemu Mac OS X

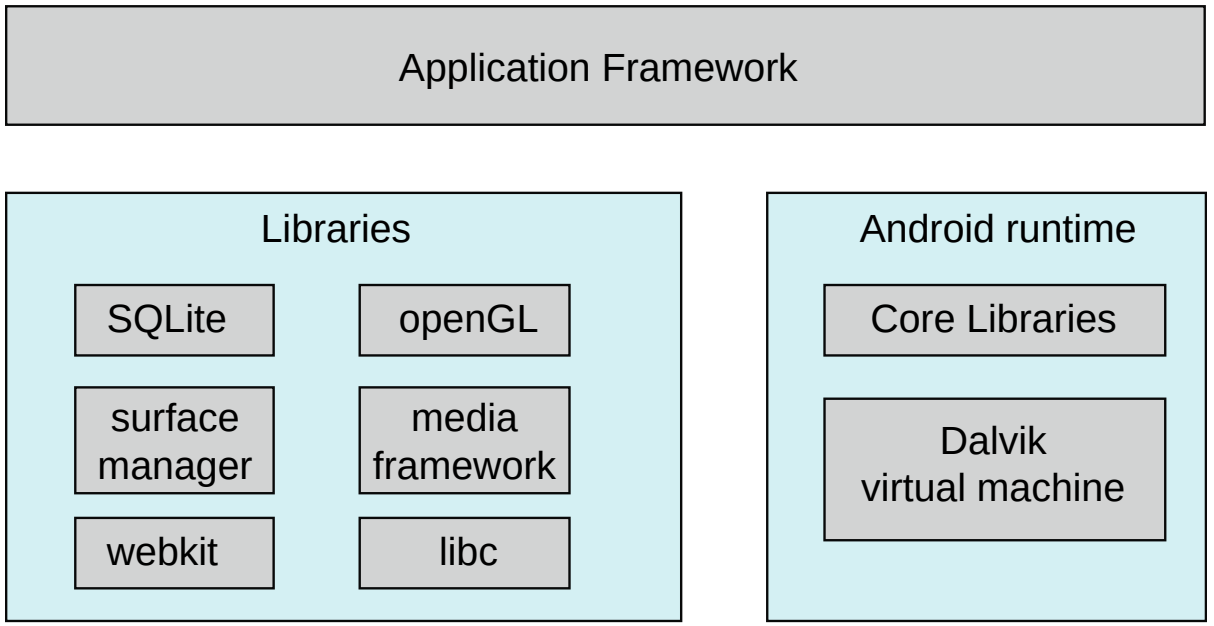


Hybrydowe jądro XNU

- Mikrojądro Mach 3: szeregowanie procesów (również RT), wątki, pamięć wirtualna
- Kod jądra BSD UNIX (POSIX API): model procesu i wątków, mechanizmy ochrony, systemy plików (w tym HFS/HFS+), IPC, protokoły sieciowe, gniazda, NFS,…

Architektura Android OS (Google)

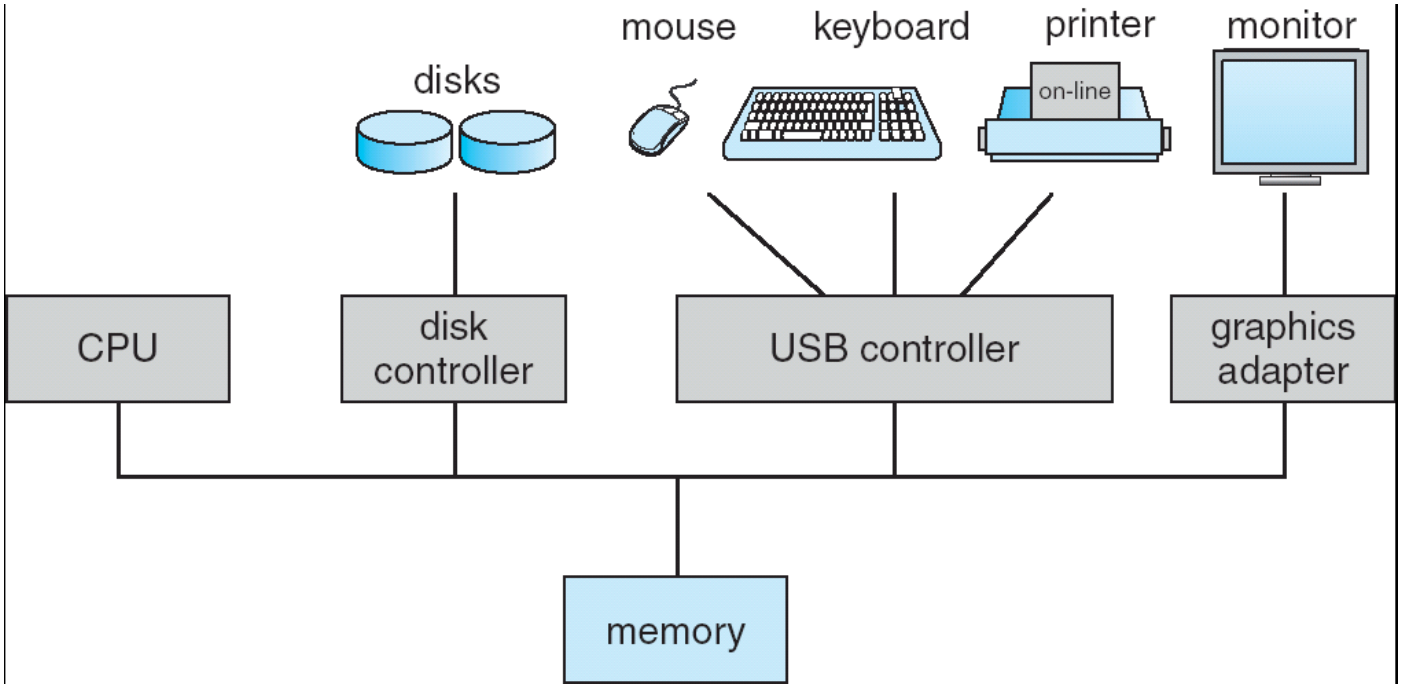
- System definiowany przez Open Handset Alliance (z Google). Open Source
- Zmodyfikowane jądro GNU/LINUX: procesy, pamięć, zarządzanie urządzeniami i mocą pobieraną przez system komputerowy
- Środowisko wykonania zawiera maszynę wirtualną Dalvik (wykonuje programy Java + Android API) oraz biblioteki (webkit, Sqlite, OpenGL, okrojone libc)



Wstęp

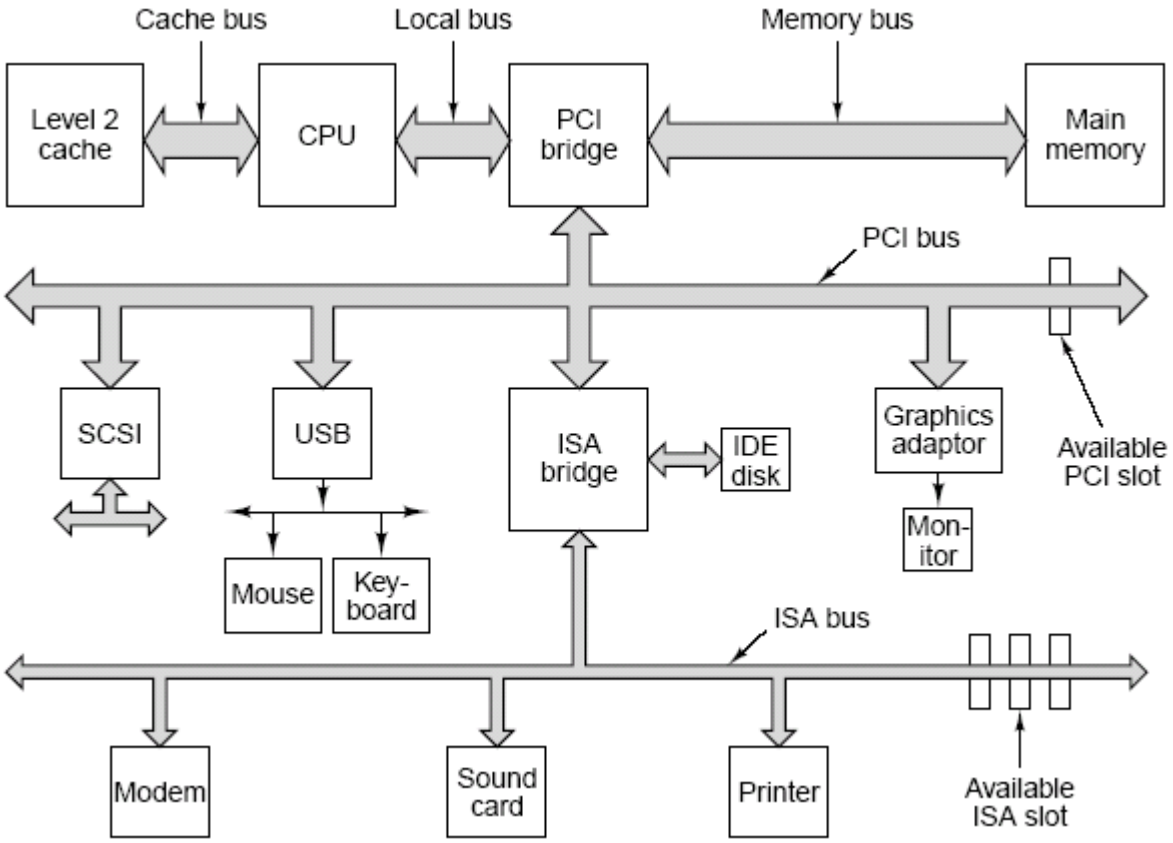
1. System operacyjny, a system komputerowy
2. Zadania i interfejs systemów operacyjnych
3. Struktury systemów operacyjnych
4. Działanie systemu komputerowego

Architektura systemu komputerowego



Jednostka centralna (*CPU*) oraz szereg sprzętowych sterowników urządzeń (*device controller)* są połączone magistralą, umożliwiającą kontakt ze wspólną pamięcią

Przykład: architektura ” klasycznego PC”



Działanie systemu komputerowego

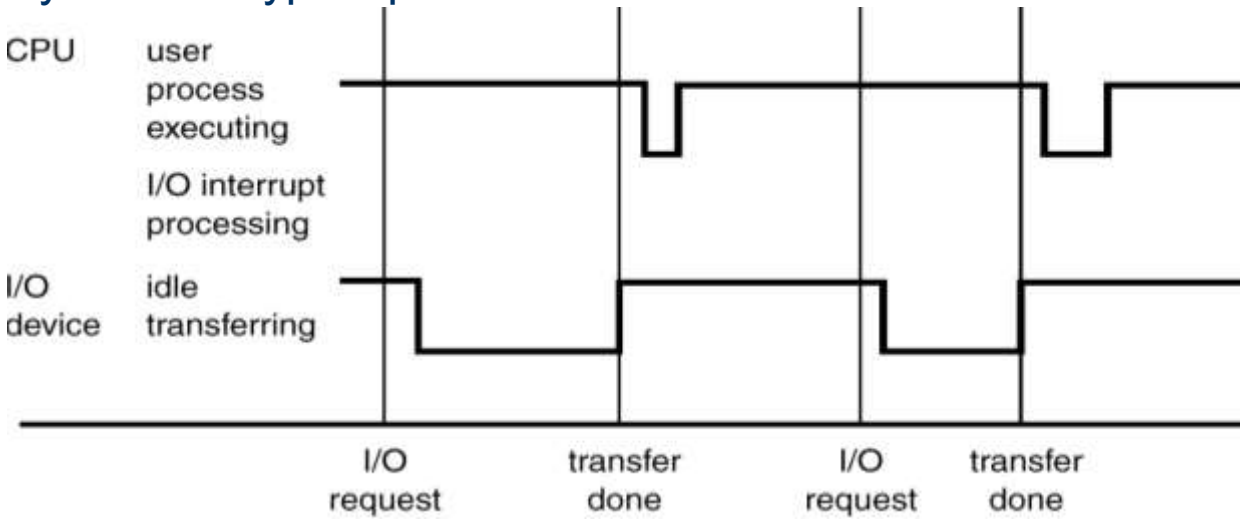
- Urządzenia wejścia/wyjścia i procesor mogą pracować współbieżnie (stąd problemy: komunikacji i synchronizacji), współzawodnicząc w dostępie do pamięci
- Każdy sterownik urządzenia (device controller) zarządza urządzeniami określonego typu.
- Każdy sterownik urządzenia ma lokalny bufor. Procesor kopiuje dane z/do pamięci operacyjnej do/z lokalnych buforów sterowników.
- Operacje wejścia/wyjścia - pomiędzy urządzeniem a lokalnym buforem sterownika urządzenia.
- Sterownik urządzenia powiadamia procesor o zakończeniu operacji wejścia/wyjścia za pomocą przerwania (interrupt).

Przerwania

- Przerwanie powoduje przekazanie sterowania do procedury obsługi za pomocą wektora przerwań (interrupt vector), który zawiera adresy procedur obsługi przerwań (*interrupt service routines, ISR*).
- Architektura przerwań musi dbać o zachowanie adresu przerwanej instrukcji procesora.
- Zwykle podczas obsługi jednego przerwania inne przerwania są wyłączone (disabled), aby zapobiec utracie przerwania (lost interrupt). Istnieją też schematy priorytetowe: przerwanie o wyższym priorytecie może przerwać obsługę aktualnie obsługiwanego przerwania, ale inne przerwania są maskowane.
- Pułapka (trap), czyli wyjątek (*exception*) jest rodzajem przerwania generowanym programowo dla sygnalizacji błędu (np. dzielenia przez zero) bądź żądania realizacji zamówienia, wymagającego obsłużenia przez system operacyjny.
- System operacyjny jest sterowany przerwaniami (*interrupt driven*).

Obsługa przerwań

- System operacyjny zachowuje stan procesora, tzn. rejestry, licznik rozkazów.
- System ustala który typ przerwania miał miejsce:
  - wymagające odpytywania (polling) przerwania od urządzeń wejścia/wyjścia
  - z wektorowego systemu przerwań
- Oddzielne fragmenty kodu określają jaka akcja powinna nastąpić w każdym z w/w typów przerwań.

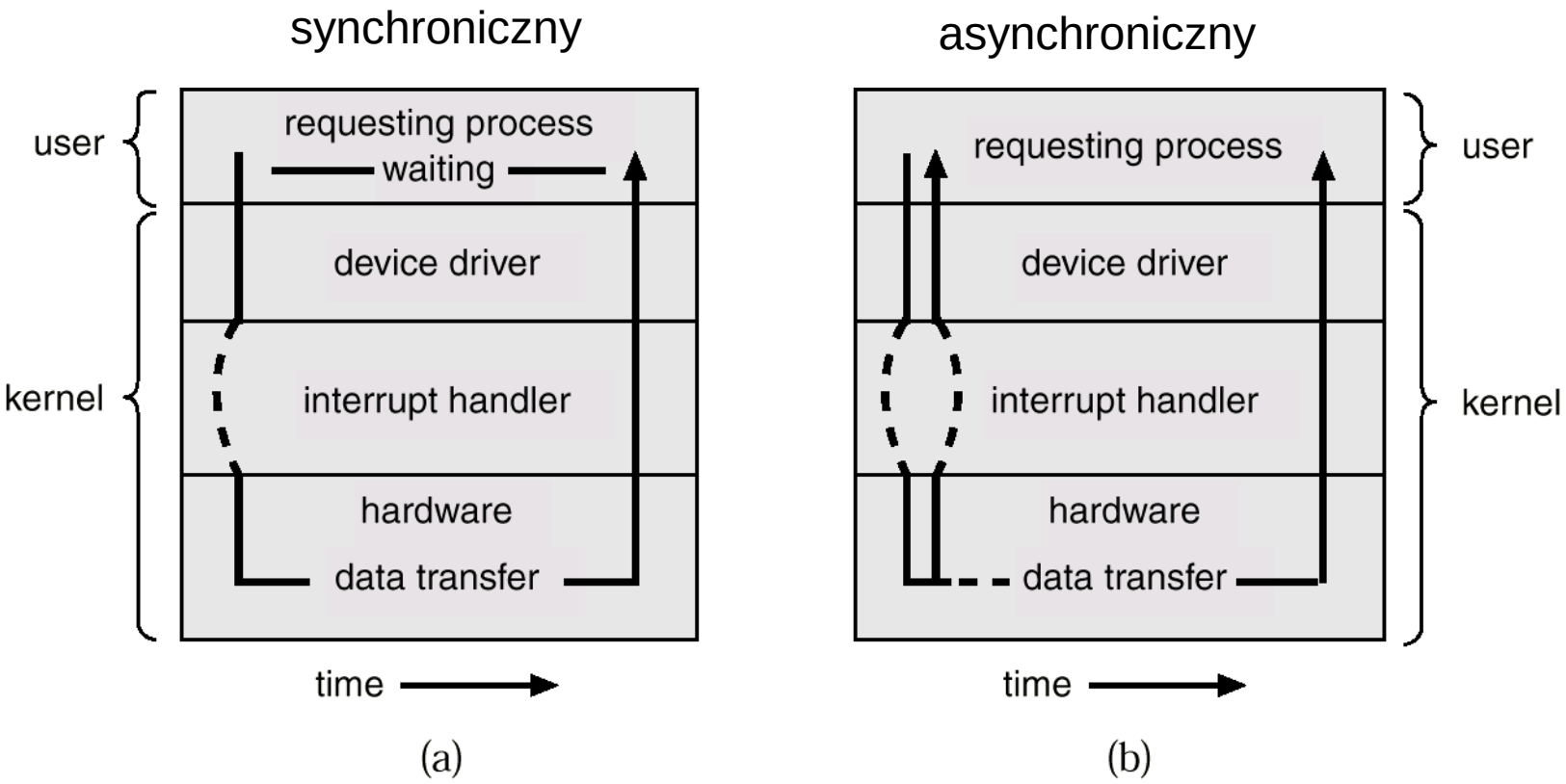


Przebieg zdarzeń przy obsłudze przerwań

Obsługa wejścia/wyjścia

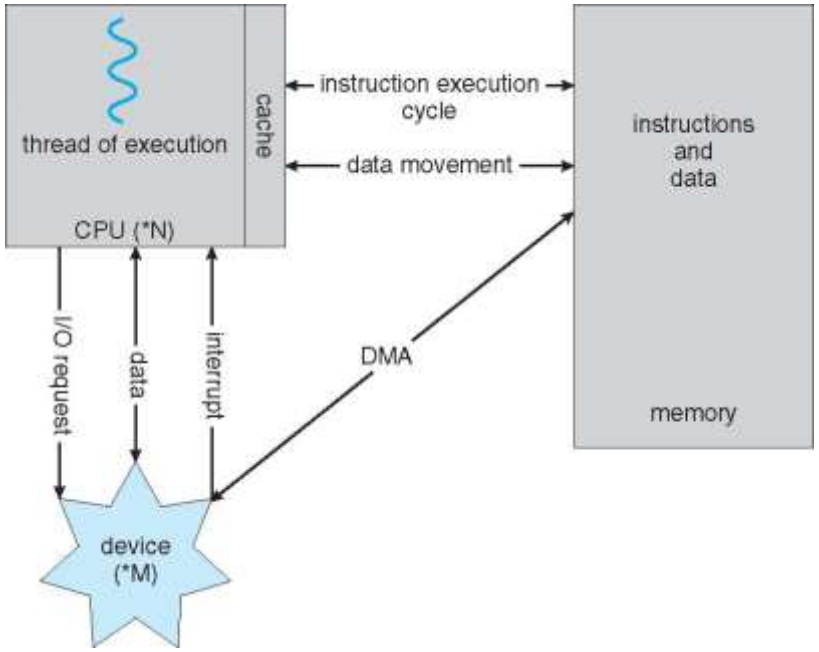
- **Synchroniczna operacja wejścia-wyjścia**. Po rozpoczęciu operacji wejścia/wyjścia sterowanie wraca do zadania-zleceniodawcy dopiero po zakończeniu tej operacji.
  - Rozkaz wait powoduje bezczynność procesora do czasu przyjścia przerwania
  - Pętla czekania (wait loop) jest powtarzana aż nadejdzie sygnał przerwania.
  - Jeśli system zawsze czeka na koniec operacji wejścia/ wyjścia - nie jest możliwe współbieżne wykonywanie wielu takich operacji i ew. obliczeń za pomocą procesora.
  - Operacja synchroniczna może być również realizowana w trybie nieblokującym.
  - Sterowanie wraca do zadania-zleceniodawcy natychmiast, ale jeśli zamawiana operacja nie mogła być wykonana bezzwłocznie – zwracany jest kod błędu.
- **Asynchroniczna operacja wejścia-wyjścia**. Po rozpoczęciu operacji wejścia/wyjścia sterowanie wraca do zadania-zleceniodawcy natychmiast - bez czekania na zakończenie tej operacji. Zadanie-zleceniodawca jest zazwyczaj powiadamiane asynchronicznie o zakończeniu realizacji zamówionej operacji (pomyślnym bądź nie) i może (za pomocą oddzielnej funkcji systemowej) pobrać kod zakończenia (błędu).

Dwa sposoby obsługi wejścia/wyjścia



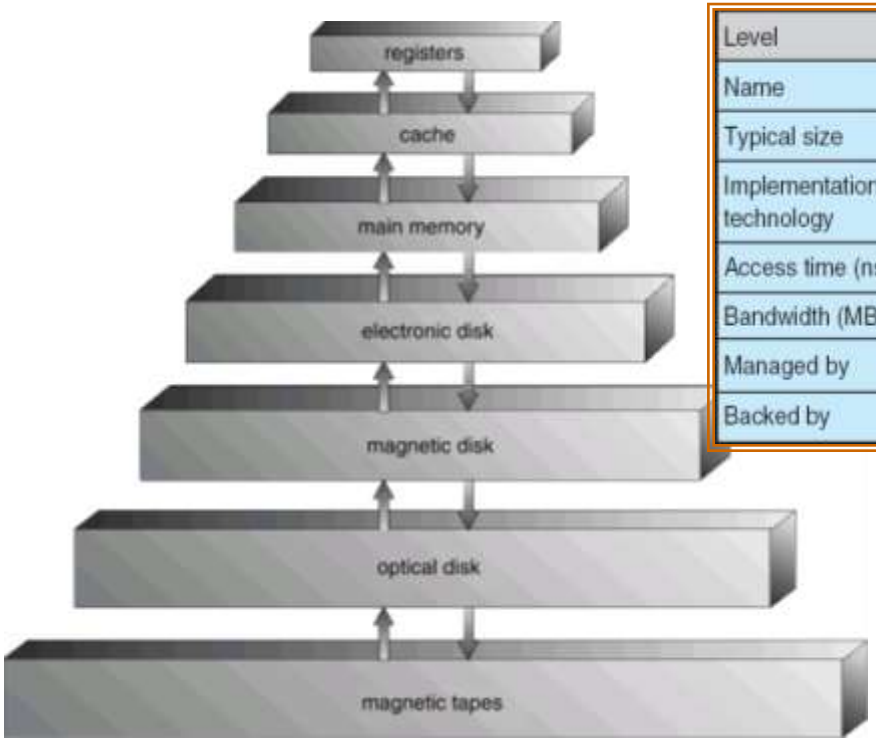
Bezpośredni dostęp do pamięci - DMA

- DMA umożliwia efektywne wykorzystanie urządzenia wejścia/wyjścia o szybkości transmisji zbliżonej do szybkości transmisji pamięci operacyjnej.
- Jednostka centralna ustawia w rejestrach DMA adresy źródła i przeznaczenia oraz długość transmisji. Sterownik DMA kopiuje bloki danych z bufora urządzenia do pamięci operacyjnej - bez udziału procesora
- DMA wysyła jednostce centralnej przerwanie po zakończeniu transmisji (jedno przerwanie na blok, a nie dla każdego bajta).
- . Uwaga: DMA i jednostka centralna współzawodniczą w dostępie do pamięci operacyjnej → zysk bywa różny



Struktura pamięci

- Pamięć operacyjna - jedyna duża i szybka (ale ulotna) pamięć, do której procesor może mieć bezpośredni dostęp.
- Pamięć pomocnicza - nieulotne rozszerzenie pamięci operacyjnej o dużej pojemności.

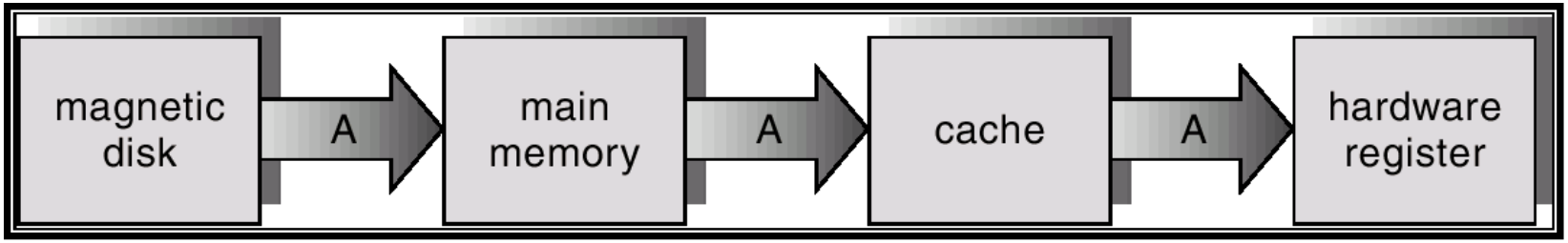


Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

← Hierarchia pamięci

Hierarchia pamięci – c.d.

Droga danej **A** z dysku magnetycznego do rejestru jednostki centralnej (CPU).



Wielopoziomowe stosowanie idei pamięci podręcznej *(caching)* – kopiowanie informacji do szybszego podsystemu jedynie w razie potrzeby. Problemy konstrukcyjne:

- wybór rozmiaru pamięci podręcznej i rozmiaru bloku
- wybór algorytmów zarządzania (funkcja odwzorowująca, algorytm zastępowania)
- wybór reguł zapisu, zapewniający zgodność pamięci podręcznej (*cache coherency*) oraz efektywność

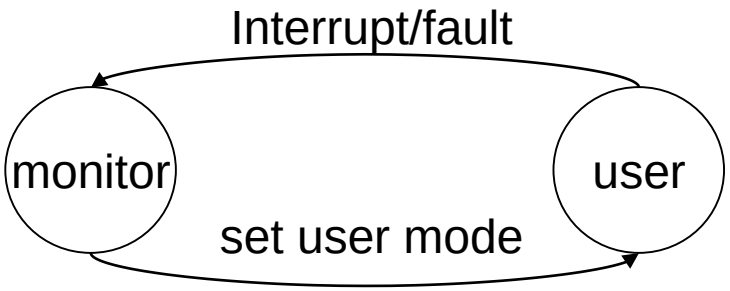
Ochrona sprzętowa

- Dualny tryb operacji (tryby: użytkownika i monitora).
- Ochrona wejścia/wyjścia
- Ochrona pamięci
- Ochrona jednostki centralnej (procesora)

Dualny tryb operacji

1. **Tryb użytkownika** (*user mode*) - gdy wykonywany jest kod na rzecz użytkownika.
2. **Tryb monitora** (nadzorcy, systemu, *kernel mode*) – wykonywany jest kod na rzecz systemu operacyjnego.

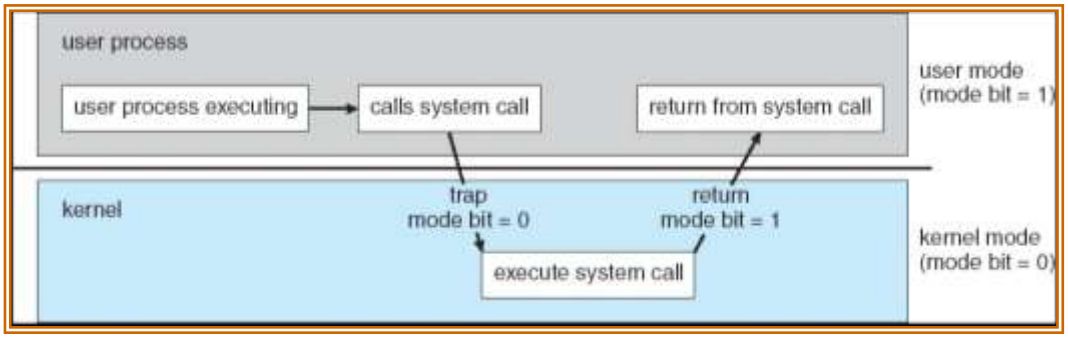
- **Bit trybu** jednostki centralnej wskazuje aktualny tryb pracy
- Przy wystąpieniu przerwania czy pułapki **sprzęt wymusza przejście do trybu monitora**



- **Rozkazy uprzywilejowane** *CPU* są dostępne jedynie w trybie monitora.

Realizacja funkcji systemowych

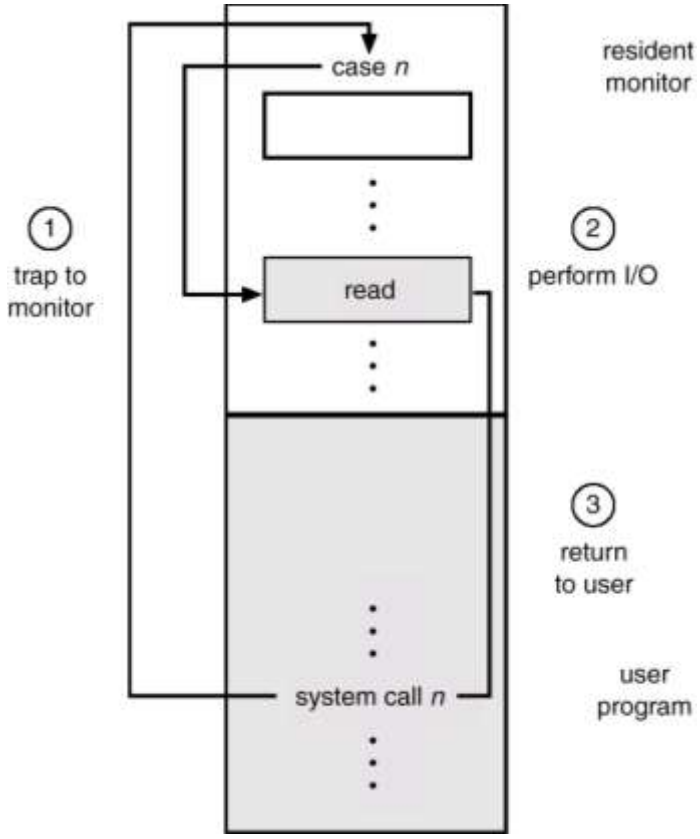
- Wywołanie systemowe zazwyczaj wyzwalane jest przez wykonanie specjalnego rozkazu przerwania programowego (pułapki, ang. trap), zmieniające tryb pracy CPU (na tryb monitora)
- Z wektora przerwań pobierany jest adres odpowiedniej procedury obsługi (ISR).
- Procedura ISR pobiera argumenty wywołania funkcji systemowej: (rejestry CPU | stos | blok pamięci wskazywany przez rejestry).
- Jeśli parametry wywołania są poprawne i dopuszczalne; to wywoływany jest kod jądra, który realizuje zamówienie. W przeciwnym przypadku ISR ustawia kod błędu i powraca.
- Procedura ISR przekazuje do wołającego parametry wyjścia funkcji, po czym zmieniany jest tryb pracy CPU i występuje powrót.



Uwaga: w realnych systemach modele realizacji funkcji systemowych mogą być bardziej złożone, np. po to by zwiększyć stopnień współbieżności wykonywania funkcji systemowych i kodu użytkownika (typowo wywoływany jest planista)

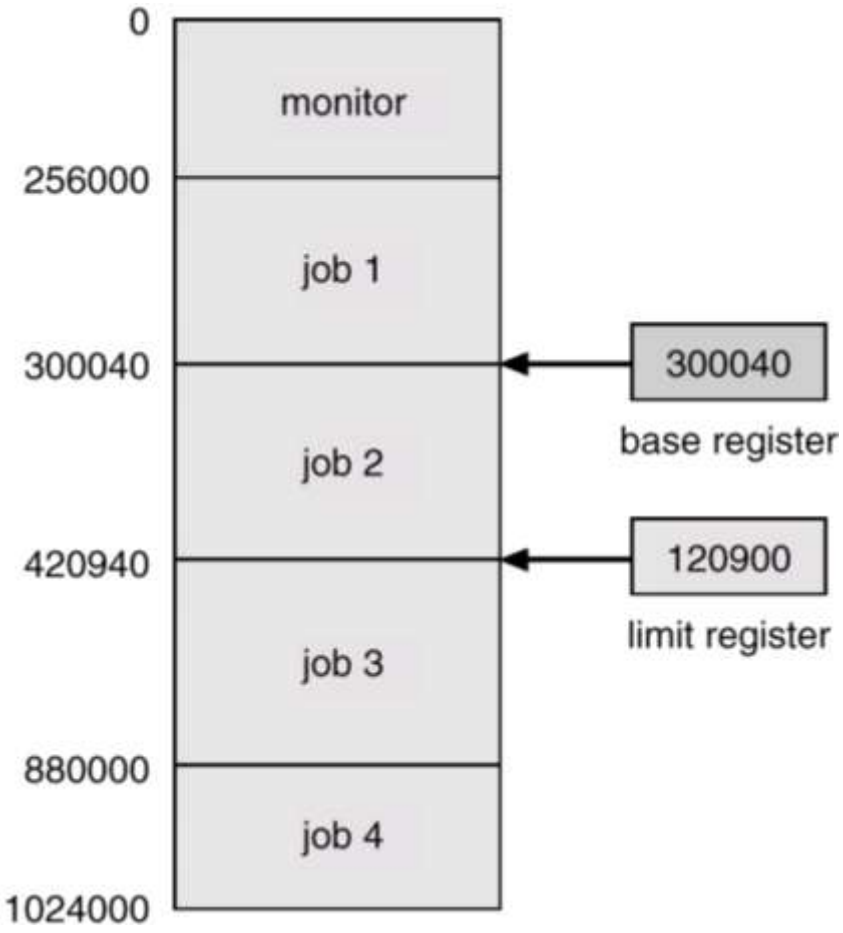
Ochrona wejścia/wyjścia

- Wszystkie instrukcje wejścia/wyjścia są uprzywilejowane
- Dla pełnej ochrony wejścia/wyjścia trzeba mieć pewność, że program użytkownika nigdy nie przejmie kontroli nad komputerem w trybie monitora (np. oznacza to konieczność ochrony wektora przerwań)
- Programy użytkownika mają dostęp do instrukcji wejścia/wyjścia jedynie za pośrednictwem funkcji systemowych (w których dokonuje się przełączanie trybu pracy CPU, sprawdzanie uprawnień i praw dostępu)



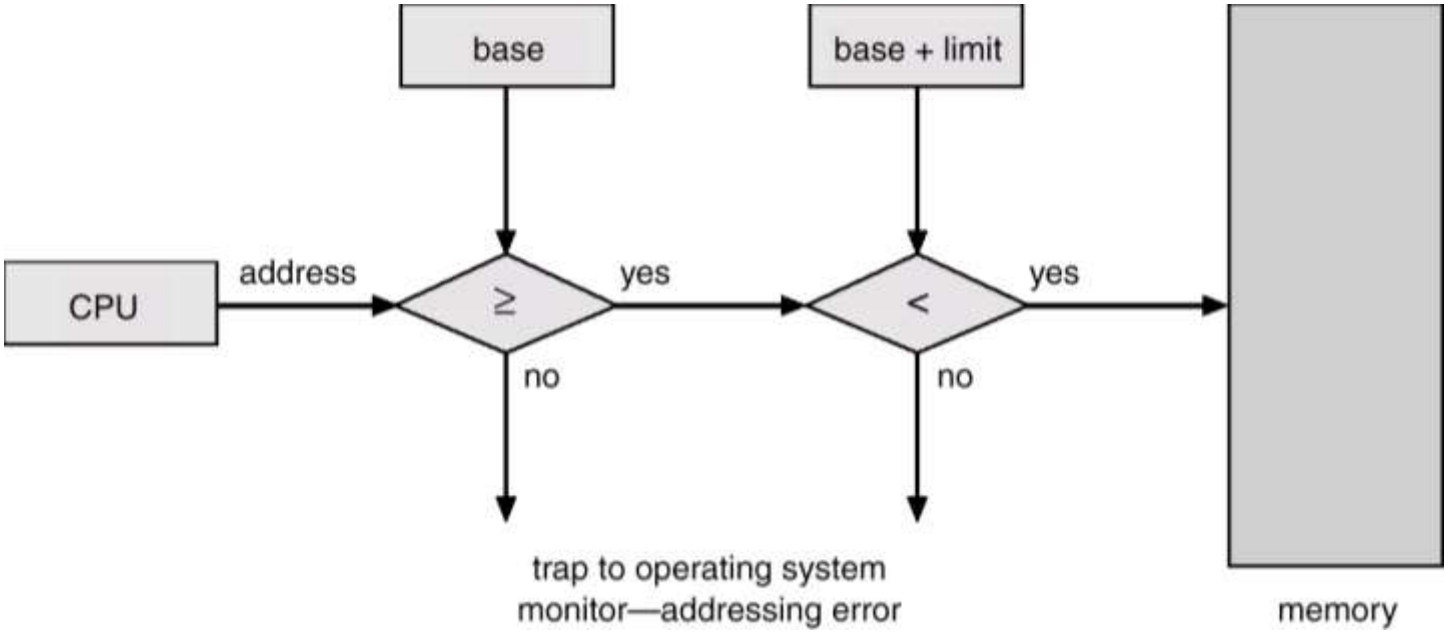
Ochrona sprzętowa pamięci

- Konieczna jest ochrona przynajmniej wektora przerwań i procedur obsługi przerwań.
- Dla uzyskania ochrony pamięci potrz
  zakres poprawnych adresów:
  - rejestr bazowy – przechowuje n
    fizyczny pamięci.
  - rejestr graniczny – zawiera rozr
- Pamięć poza dopuszczalnym zakres
  przed dostępem.



Ochrona sprzętowa pamięci

- W trybie monitora (uprzywilejowanym) system ma nieograniczony dostęp do pamięci monitora (systemu) jak i użytkownika
- Instrukcje zapisu do rejestru bazowego i granicznego są **uprzywilejowane**
- Błąd adresowania wyzwala pułapkę (ang. trap), obsługiwaną przez odpowiednią systemową procedurę ISR o adresie z wektora przerwań.



Ochrona jednostki centralnej

- **Czasomierz** (*timer*) – generuje okresowo przerwania dla zapewnienia, że system operacyjny może przejąć sterowanie. Sposób generacji za pomocą zegara stałookresowego i licznika:
  - Przy każdym tyknięciu zegara następuje zmniejszenie zawartości licznika (wartość początkową ustawia system za pomocą uprzywilejowanej instrukcji).
  - Przy wyzerowaniu licznika generowane jest przerwanie.
- Czasomierz wykorzystywany jest powszechnie do realizacji pracy z podziałem czasu.
- Czasomierz jest też używany do wyznaczania bieżącego czasu.
- Rozkazy modyfikujące działanie czasomierza są zastrzeżone do użytku monitora (**uprzywilejowane**).