Struktury systemu operacyjnego

Składowe systemu

- zarządzanie procesami,
- zarządzanie pamięcią główną,
- zarządzanie plikami,
- zarządzanie wejściem/wyjściem,
- zarządzanie pamięcią pomocniczą,
- praca w sieci,
- ochrona,
- system interpretacji poleceń.

Zarządzanie procesami

Za *proces* będziemy uważać program w trakcie wykonywania.

Różne procesy mogą realizować ten sam program.

Proces musi korzystać z pewnych zasobów: CPU, pamięć operacyjna, pliki, we/wy.

W związku z zarządzaniem procesami system odpowiada za:

- tworzenie i usuwanie procesów (również systemowych),
- wstrzymywanie procesów i ich wznawianie,
- dostarczenie mechanizmów:
  - synchronizacji procesów,

- komunikacji między procesami,
- postępowania z blokadami.

Zarządzanie pamięcią operacyjną

Pamięć — duża tablica słów i/lub bajtów, z których każde ma własny adres.

Służy do przechowywania szybko dostępnych danych współdzielonych przez CPU i urządzenia we/wy.

Pamięć operacyjna jest nietrwała.

System operacyjny odpowiada za:

- utrzymanie informacji o tym, które fragmenty pamięci są zajęte i przez kogo,
- decydowanie, które procesy należy załadować jeśli pojawia się wolna pamięć,
- przydzielanie i zwalnianie obszarów pamięci w miarę potrzeb.

Zarządzanie plikami

Plik — zbiór logicznie powiązanych informacji zdefiniowany przez twórcę.

Zazwyczaj zawiera programy (w postaci źródłowej albo wynikowej) lub dane.

Rozmaite formaty (struktury wewnętrzne) plików.

Do zadań systemu w zakresie zarządzania plikami należy:

- tworzenie i usuwanie plików,
- tworzenie i usuwanie katalogów,

- dostarczenie zestawu operacji do manipulowania plikami i katalogami,
- odwzorowywanie plików w pamięci pomocniczej,
- składowanie plików na trwałych nośnikach.

System zarządzania we/wy

System ten składa się z:

- podsystemu obsługi pamięci buforowanej i buforowania,
- ogólnego interfejsu do modułów sterujących urządzeń,
- modułów sterujących konkretnych urządzeń zewnętrznych.

Zarządzanie pamięcią pomocniczą

Ze względu na charakterystykę (wielkość i nietrwałość) pamięci głównej konieczne jest (zazwyczaj) stosowanie pamięci pomocniczej.

większość nowoczesnych systemów używa do tego celu dysków magnetycznych.

Typowe zadania systemu to:

- zarządzanie wolnymi obszarami,
- przydzielanie pamięci,
- szeregowanie operacji dyskowych.

Systemy rozproszone — praca sieciowa

Procesory wyposażone we własne zegary i lokalną pamięć operacyjną.

Połączenie przez sieć komunikacyjną.

Połączenia z wykorzystaniem protokołów (standardów) komunikacyjnych.

Dostęp do zasobów sieciowych: przyśpieszenie obliczeń, dostępność danych, niezawodność.

Realizacja np. przez uogólnienie dostępu do pliku.

Komunikacja — kontroler sterujący urządzeniem sieciowym.

Ochrona

Kontrola dostępu przez programy, procesy i użytkowników do zasobów systemowych i zasobów użytkowników.

Mechanizm ochronny musi:

- odróżniać dostęp niepowołany od powołanego,
- co ma podlegać ochronie i jakiej,
- dostarczać środki realizacji.

Błędy w interfejsach są podstawową przyczyną umożliwiającą obejście mechanizmów ochronnych.

Interpreter poleceń

Program czytający instrukcje (polecenia) użytkownika:

- shell,
- interpreter poleceń.

Podstawowa funkcja — pobrać i wykonać kolejne polecenie dotyczące np.: tworzenia procesów i zarządzania nimi, we/wy, administrowania pamięcią, dostępu do plików, ochrony, sieci, itp.

Usługi (serwisy) systemowe

System dostarcza pewnych usług programom i użytkownikom. Typowe klasy tych usług to:

- załadowanie do pamięci, wykonanie programu i jego zakończenie,
- pośrednictwo w realizacji operacji we/wy,
- operacje tworzenia, usuwania, czytania i zapisywania plików,
- mechanizmy komunikacji między procesami — zwykle poprzez pamięć wspólną (w tym samym systemie) albo wymianę komunikatów (w różnych systemach)
- mechanizmy nieustannego wykrywania błędów i poprawnej reakcji systemu w przypadku wykrycia błędu.

Dodatkowe funkcje umożliwiające zwiększenie efektywności systemu:

- przydział zasobów,
- rozliczenia (accounting) w celach statystycznych albo komercyjnych,
- ochrona — każdy dostęp do zasobów jest kontrolowany.

Funkcje (wywołania) systemowe — system calls

Interfejs miedzy wykonywanym programem a systemem operacyjnym.

Dostępne jako instrukcje z poziomu języka assemblera.

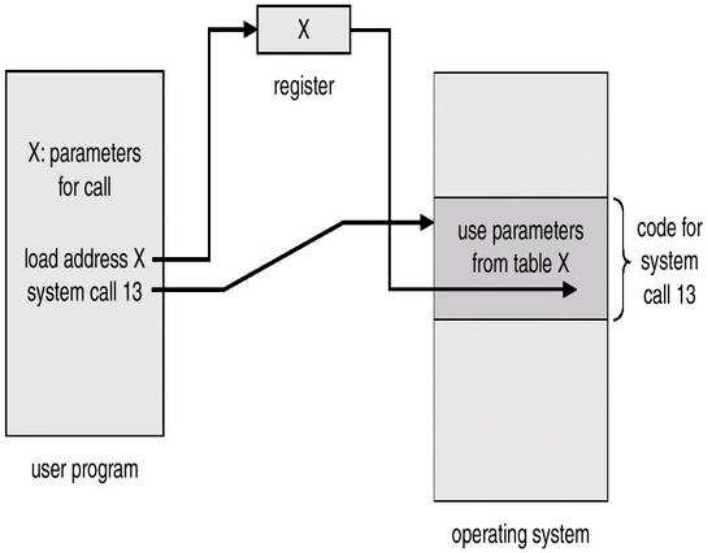
Niektóre języki wyższego poziomu pozwalają na bezpośrednie użycie funkcji systemowych, np.: C, PERL, niektóre implementacje Pascala.

Trzy metody przekazania parametrów funkcji systemowej:

- w rejestrach procesora,

- w tablicy w pamięci — adres tablicy w rejestrze procesora,
- na stosie systemowym — program wkłada, system zdejmuje.

Dwie ostatnie metody nie ograniczają liczby i długości parametrów.



Rodzaje funkcji systemowych

Przykładowe rodzaje i funkcje systemowe:

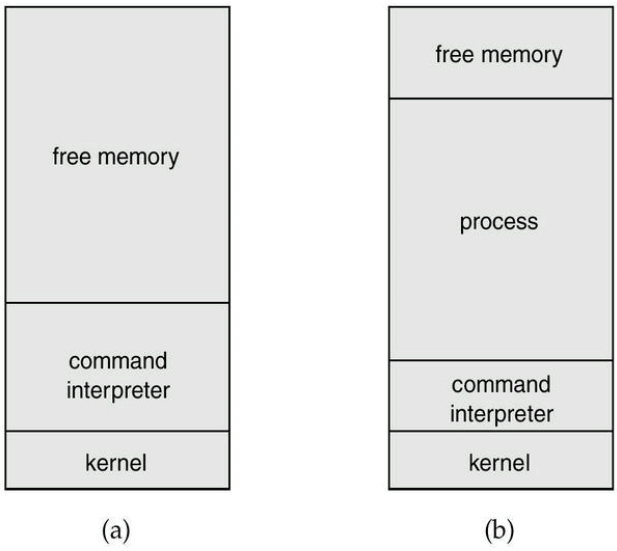
- Kontrola procesów: end, abort, load, execute, create process, terminate process, itd
- Pliki: create, delete, open, close, read, itd.
- Urządzenia: request device, release device, read, write, itd.
- Utrzymywanie informacji: get time, get date, get system data, get process attributes, itd.
- Komunikacja: create connection, send message, receive message, itd.

Przykłady sterowania procesami i zadaniami

MS-DOS — system jednozadaniowy, interpreter poleceń wywoływany na początku pracy komputera.

Wykonanie programu polega na załadowaniu go do pamięci (nawet kosztem kodu interpretera) i przekazaniu sterowania do początku programu.

Kod zakończenia przechowywany jest przez system a działanie wznawia pozostały w pamięci fragment interpretera poleceń, który ładuje ponownie cały swój kod.



System UNIX — wielozadaniowy.

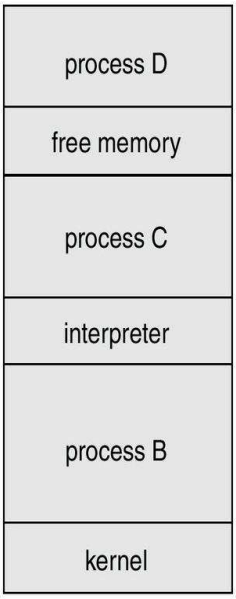
Uruchomienie programu przebiega tu w kilku krokach:

1. Po zarejestrowaniu się w systemie uruchamiany jest w imieniu użytkownika interpreter poleceń (shell).

2. W celu zapoczątkowania nowego procesu wykonuje on funkcję systemową `fork`.

3. Następnie za pomocą funkcji systemowej `exec` ładowany jest do pamięci nowy program.

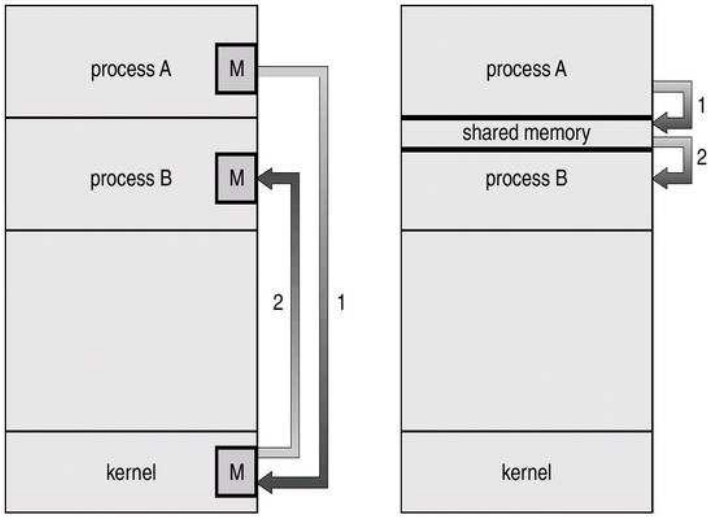
4. Shell albo czeka na zakończenie wykonania programu, albo inicjuje to wykonanie i jest gotów do odbioru kolejnych poleceń użytkownika.



Modele komunikacji między procesami

Dwa standardowe sposoby komunikacji :

- poprzez wymianę informacji w pamięci wspólnej — mapowanie pamięci (zgoda na dostęp dwóch procesów do tej samej pamięci), system nie kontroluje sposobu wymiany informacji, konieczna jest synchronizacja dostępu;
- poprzez przekazywanie komunikatów — korzystamy z pośrednictwa systemu, nawiązanie połączenia, adresacja, demony komunikacyjne, klienci i serwer.



Programy systemowe

Tworzą wygodne środowisko do opracowywania i wykonywania innych programów.

- manipulowanie plikami,
- informacje o stanie systemu,
- manipulowanie zawartością plików (edycja),
- wspomaganie języków programowania — translatory, interpretery, assemblery,
- ładowanie i wykonanie programu,
- komunikacja — komunikaty, poczta, pliki, zdalne wykorzystanie (remote login).

Większość użytkowników widzi system poprzez zbiór programów systemowych (nie funkcji).

Najważniejszy program systemowy — interpreter poleceń.

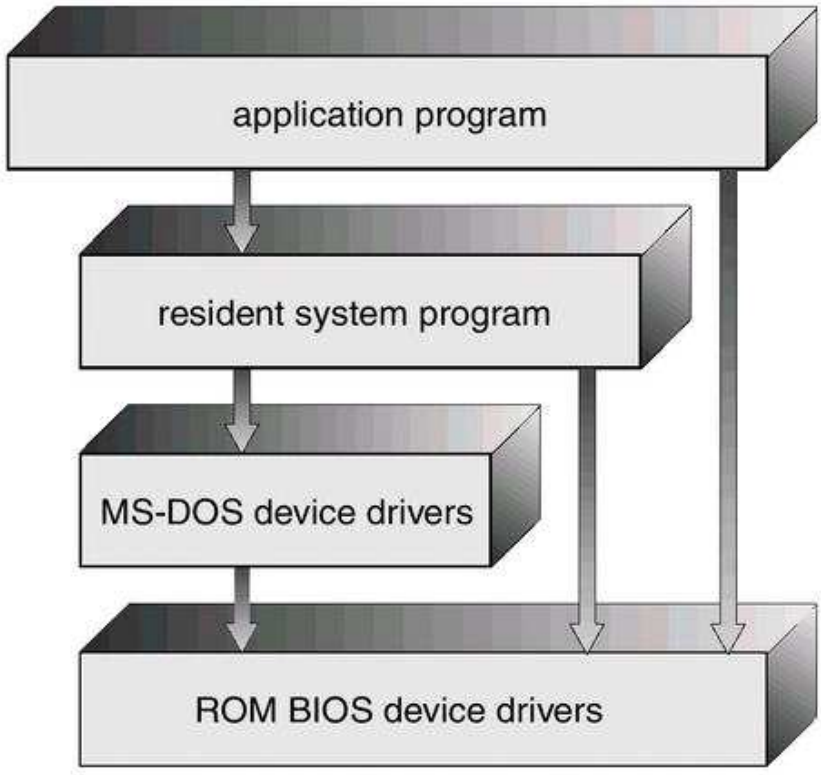
Polecenia wbudowane w interpreter albo wywołanie programu systemowego (UNIX).

Struktura systemu

MS-DOS napisany z myślą o maksymalnej funkcjonalności w jak najmniejszej objętości.

- bez podziału na moduły,
- przy istnieniu pewnej struktury interfejsy i poziomy funkcjonalności nie są dobrze wyodrębnione.

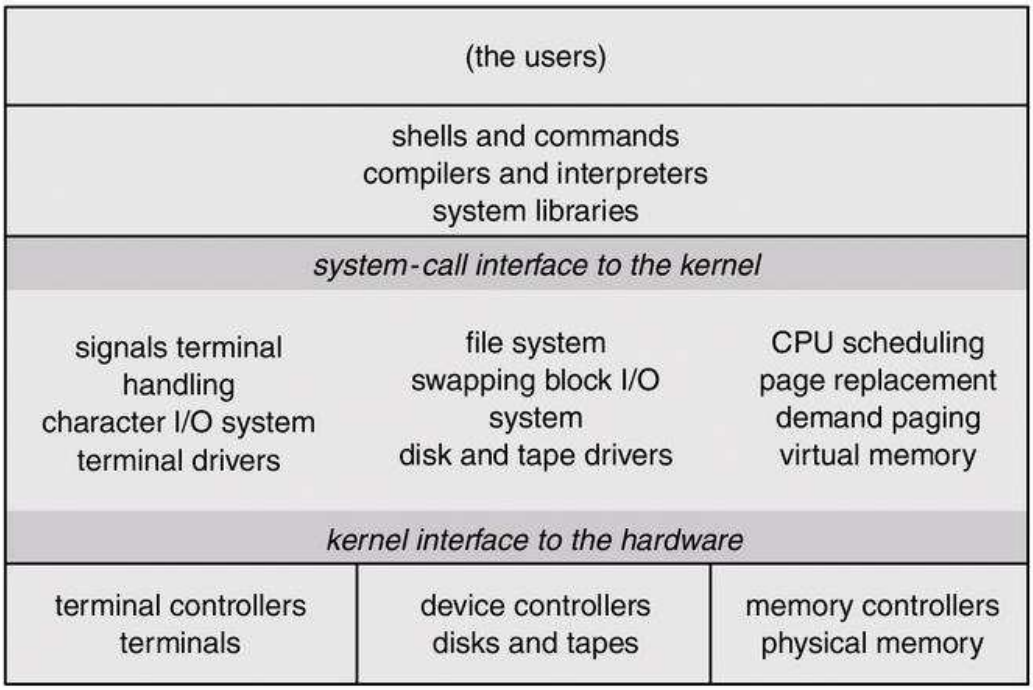
Ograniczenia sprzętowe — brak dualnego trybu pracy i ochrony sprzętowej.



Oryginalny (pierwotny) UNIX — ograniczenia sprzętowe, sztywny podział na programy systemowe i jądro, warstwy.

Funkcje systemowe definiują interfejs programisty.

Programy systemowe definiują interfejs użytkownika.



Podejście warstwowe

Ulepszenie sprzętu pozwala na podział systemu na lepiej dobrane fragmenty.

Jednym ze sposobów modularyzacji jest podział na warstwy.

System podzielony jest na N warstw (poziomów): warstwa 0 - sprzęt, warstwa N - interfejs użytkownika.

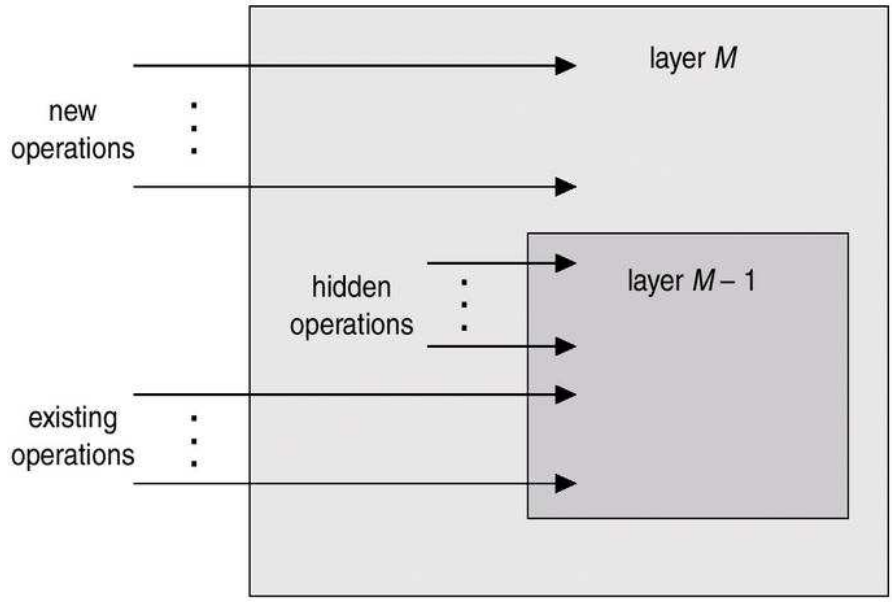
Warstwa zawiera struktury danych i operacje do manipulacji na nich.

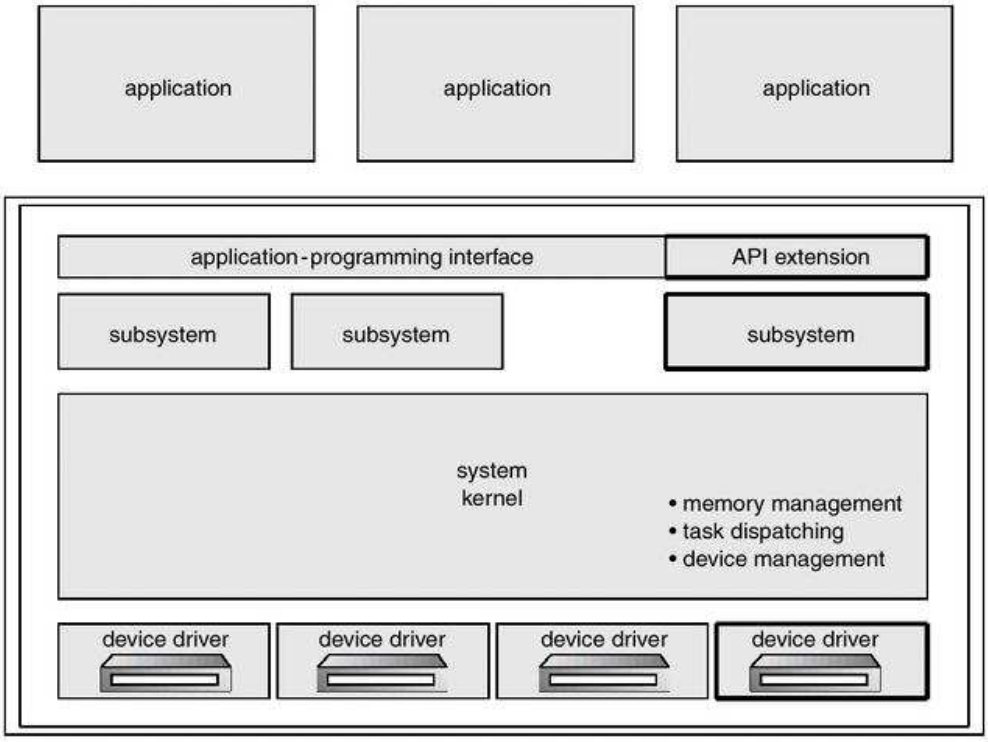
Każda warstwa używa wyłącznie operacji dostarczonych przez warstwę bezpośrednio niższą.

Każda warstwa może mieć własne operacje ukryte — używane wyłącznie do swoich celów.

Budowa ściśle warstwowa bywa mniej wydajna od innych. Każda warstwa dodaje własny narzut na koszt odwołania dos systemu — zmniejszenie

liczby bardziej funkcjonalnych warstw (np. OS/2).





System z mikrojądrem

Co się tylko da przenosimy do programów systemowych albo nawet do przestrzeni użytkownika.

Komunikacja między modułami użytkownika poprzez wymianę komunikatów.

Zyski: łatwiejsze rozszerzenie i przenoszenie na inną architekturę, większa niezawodność i bezpieczeństwo.

Maszyny wirtualne

Traktujemy wszystko co jest poniżej programu użytkowego jak sprzęt.

Każdy proces otrzymuje wirtualną kopię komputera będącego podstawą systemu.

Każdy ma złudzenie pracy na swoim własnym procesorze ze swoją własną kopią pamięci.

Zasoby są dzielone dla utworzenia maszyn wirtualnych — PS, pamięć wirtualna, spooling.

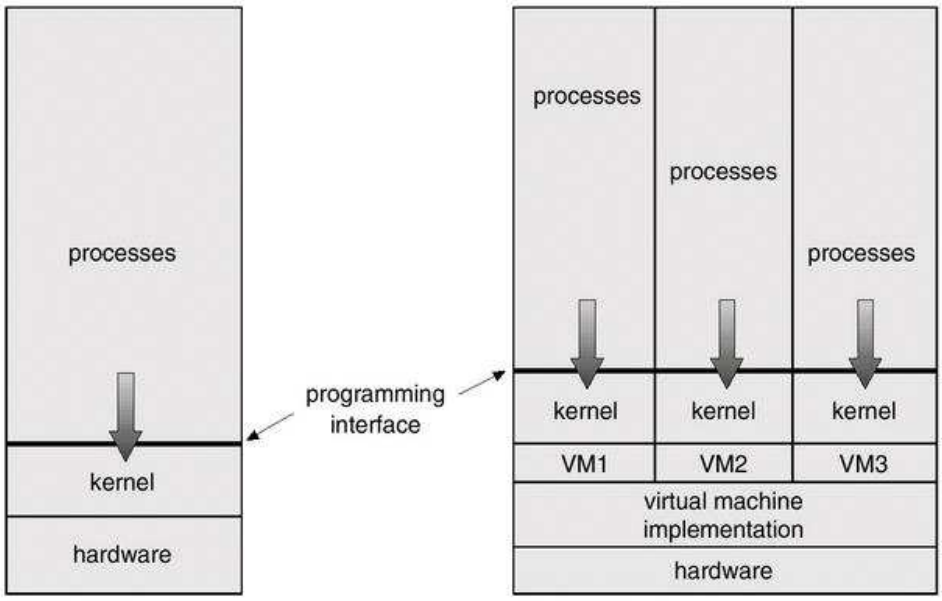
Pewien problem stanowi wirtualizacja dysków — minidyski.

Użytkownicy poszczególnych maszyn mogą wykonywać na nich dowolny system operacyjny (np. CMS w systemie VM firmy IBM).

Zalety i wady:

- pełna ochrona zasobów systemowych — każda maszyna jest izolowana,
- brak możliwości współdzielenia zasobów —- współużytkowanie minidysku albo sieć maszyn wirtualnych,
- dobre środowisko do badań nad systemami operacyjnymi,

- trudna i skomplikowana implementacja.



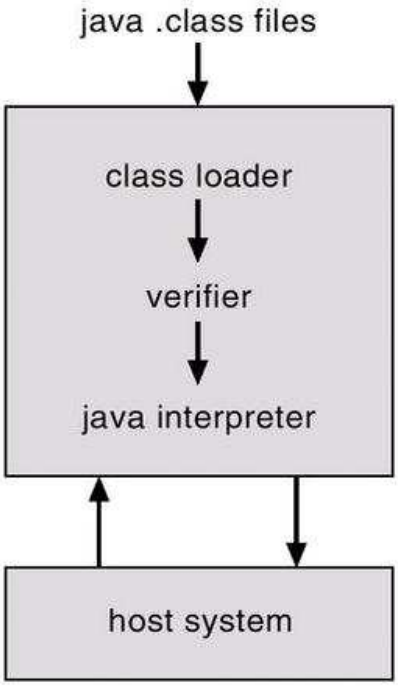
Java i inne

Skompilowane programy w Javie są niezależne od platformy sprzętowej.

Są wykonywane przez maszynę virtualną Javy (JVM).

JVM — kompletne środowisko programistyczne zapewniające np. kontrolę bezpieczeństwa.

Inne języki programowania też korzystają z koncepcji maszyny virtualnej (np. Prolog).



Założenia projektowe i implementacja

Cele użytkownika — wygodny w użyciu, łatwy do nauczenia, solidny, bezpieczny i szybki.

Cele systemu — łatwy do zaprojektowania, zaimplementowania, utrzymania, elastyczny, solidny, bezbłędny i efektywny.

Mechanizmy określają jak coś należy zrobić, polityka mówi co trzeba zrobić.

Rozdzielenie mechanizmów i polityki pozwala na ewentualne zmiany polityki.

Tradycyjnie implementowane w językach niskiego poziomu — ostatnio coraz częściej w językach wysokiego poziomu (cały praktycznie Linux napisany jest w C).

Kodowanie w języku wysokiego poziomu ma liczne zalety: łatwiej się pisze, kod jest mniejszy, łatwiej usuwa się błędy.

Znacznie łatwiej się go przenosi na inną architekturę, jeśli napisany jest w języku wysokiego poziomu.

Generowanie - SYSGEN

Systemy zwykle projektuje sie dla pewnej klasy sprzętu — konieczność dostosowania systemu do konkretnej architektury.

Możliwość zmiany pewnych parametrów wewnętrznych systemu umożliwia lepszą efektywność systemu.

Bootowanie — ładowanie programu jądra systemu.

Program ładujący — kod zapamiętany w pamięci ROM, który potrafi zlokalizować obraz jądra, załadować je i uruchomić system.