3. STRUKTURA SYSTEMU OPERACYJNEGO

Systemy operacyjne różnią się pod względem organizacji wewnętrznej.

Cele systemu muszą być określone przed rozpoczęciem projektowania.

Typ systemu determinuje wybór stosowanych algorytmów i strategii projektowej.

Metodyka analizy systemu operacyjnego:

- przegląd świadczonych usług;
- przegląd interfejsu, udostępnianego przez system użytkownikom i programistom;
- rozłożenie systemu na części i badania ich wzajemnych powiązań.

System operacyjny projektuje się, dzieląc go na mniejsze części.

3.1. Składniki systemu

❒ **Procesy**

Proces: program wykonywany: - zadanie wsadowe, program użytkownika, buforowanie wyjścia.

Proces korzysta z zasobów: -czas CPU,
-pamięć,
-pliki,
-urządzenia We/Wy.

Proces otrzymuje zasoby: -w chwili utworzenia,
-są przydzielane podczas jego działania.

Po zakończeniu procesu SO odzyskuje te zasoby, które nadają się do powtórnego użytku.

Wykonanie procesu przebiega sekwencyjnie, instrukcja po instrukcji.

Program składowany na dysku jest ***pasywnym*** elementem systemu.

☞ Program wykonywany może **rodzić wiele procesów**.

System komputerowy składa się ze zbioru procesów:

- procesy systemu operacyjnego realizują kod systemu,
- procesy użytkowe realizują kod użytkownika.

Wszystkie procesy mogą być multipleksowane przez CPU – współbieżność.

SO: -tworzy i usuwa procesy systemowe i użytkowe;
-wstrzymuje i wznawia procesy;
-dostarcza mechanizmów do synchronizacji i komunikacji;
-dostarcza mechanizmów obsługi zakleszczeń.

❒ **Pamięć operacyjna** to magazyn wspólnie eksploatowany przez CPU i urządzenia We/Wy.

Pamięć operacyjna jest jedyną pamięcią, którą CPU może adresować bezpośrednio.

Program wykonywany w PAO jest adresowany za pomocą adresów bezwzględnych.

Program zakończy działanie i zajmowany obszar PAO **oznaczany** jest jako **wolny**.

Wybór sposobu zarządzania pamięcią determinowany jest rozwiązaniami sprzętowymi.

SO: -przydziela i zwalniania obszary pamięci;
-ewidencjuje zajęte obszary i informuje, który proces je zajmuje;
-decyduje, które procesy będą ładowane do wolnych obszarów.

❒ **Pliki**

Informacja jest przechowywana na nośnikach różniących się budową i organizacją fizyczną.

Kontrolę nad każdym nośnikiem sprawują sterowniki.

SO tworzy jednolity, logiczny obraz magazynowanej informacji.

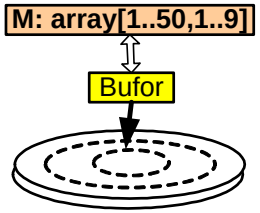
SO definiuje pliki jako jednostki ***logiczne*** informacji, niezależne od fizycznych właściwości urządzeń.

Plik to zbiór powiązanych ze sobą informacji, zdefiniowanych przez jego twórcę.

SO: -dostarcza niezbędne narzędzia do operacji plikowych,
-odwzorowuje pliki na obszary fizycznej pamięci dyskowej.

❒ **Pamięć zewnętrzna** *(dyskowa)*

Komponenty **SO** przechowywane są na dysku.



Pamięć zewnętrzna jest ciągle używana i musi być wydajna.

SO: -zarządza wolnymi obszarami (planuje i przydziela).

Wydajność SO zależy od podsystemu dyskowego ➔ od jakości stosowanych algorytmów.

❒ **Urządzenia We/Wy**

System operacyjny powinien **ukrywać** przed użytkownikiem (a nawet przed częścią samego siebie) szczegóły specyfiki sprzętowych urządzeń We/Wy.

Realizuje to ***podsystem We/Wy***.

➔ Cechy danego urządzenia zna tylko właściwy moduł sterujący.

Podsystem We/Wy to: -zarządzanie pamięcią (buforowanie, pamięć podręczna i spooling);
-ogólny interfejsy do modułów sterujących urządzeń;
-moduły sterujące (programy obsługi) urządzeń sprzętowych.

❒ **System ochrony**

Mechanizm nadzorujący dostęp programów, procesów, użytkowników do zasobów komputerowych.

Zawiera sposoby określania, **co** i **jak** ma podlegać ochronie
oraz **środki do wymuszenia** wprowadzonych ustaleń.

Dostarcza środki do rozróżniania między legalnym i nielegalnym użyciem zasobów.

-Sprzęt adresujący pamięć gwarantuje działanie procesu w obrębie *swojej przestrzeni* adresowej.

-Czasomierz zapewnia, że żaden proces nie przejmie na stałe kontroli nad CPU.

-**Nie** zezwala się użytkownikowi na bezpośredni dostęp do **rejestrów** urządzeń We/Wy.

Wyszukiwanie błędów w interfejsach między składowymi podsystemami zwiększa niezawodność SO.

❒ Interpreter poleceń

Interfejs między użytkownikiem a systemem operacyjnym.

Część systemów operacyjnych zawiera interpreter poleceń w swoim jądrze.

W MS-DOS i UNIX interpreter poleceń jest ***specjalnym programem***, wykonywanym przy rozpoczynaniu zadania lub gdy użytkownik rejestruje się w systemie.

Polecenia można przekazywać do SO za pomocą instrukcji sterujących *(control statements).*

Interpreter wiersza poleceń –program interpretujący instrukcje sterujące, jest automatycznie uruchamiany rozpoczęciem zadania lub rejestracją użytkownika.

Powłoka *(shell)* –program pobierający i wykonujący instrukcje.

❒ Usługi systemu operacyjnego

System operacyjny dostarcza usług programom i użytkownikom tych programów.

Wykonanie programu -system powinien załadować program do PAO i rozpocząć wykonywanie, następnie zakończyć w sposób normalny lub z przyczyn wyjątkowych.

Operacje We/Wy -program potrzebuje operacji We/Wy odnoszących się do pliku lub urządzenia. Środki do realizacji tych czynności dostępne są wyłącznie poprzez SO.
Manipulowanie plikami: zapis i odczyty, tworzenie i usuwanie poprzez nazwy.

Komunikacja -procesy wymagają wzajemnego kontaktu i wymiany informacji.

1. procesy działają w tym samym komputerze,
2. procesy wykonywane w różnych systemach komputerowych, powiązanych siecią.

Komunikacja przebiega z wykorzystaniem:

1. pamięci dzielonej,
2. techniki przekazywania komunikatów: SO przemieszcza informację między procesami.

Wykrywanie błędów -SO jest powiadamiany o wystąpieniu błędu.
Błędy mogą wystąpić w sprzęcie lub mieć charakter programowy.
Każdy rodzaj błędu powinien być rozróżniony i obsłużony, gwarantując spójność obliczeń.

Przydzielanie zasobów: SO zarządza różnego rodzaju zasobami.
☞ Przydzielanie niektórych z nich (cykle procesora, PAO) wymaga odrębnego kodu.
Inne zasoby jak urządzenia We/Wy mogą mieć ogólniejszy kod.
Procedury planowania przydziału CPU uwzględniają:
-szybkość procesora,
-czekające zadania do wykonania,
-dostępne rejestry.

Rozliczanie -gromadzenie danych o użytkownikach i stopniu wykorzystania zasobów komputera.

Ochrona - współbieżnie procesy nie powinny zaburzać pracy innych procesów lub SO.
Gwarantuje nadzór nad dostępnymi zasobami systemu.
Zabezpiecza przed niepożądanymi czynnikami zewnętrznymi.
Chroni zewnętrzne urządzenia We/Wy (modemy) przed włamaniami.

3.2. Funkcje systemowe

Funkcje systemowe tworzą interfejs między wykonywanym programem a *SO*.

Standardowo można z nich korzystać za pomocą rozkazów w języku asemblera.

W niektórych w językach wyższego poziomu można:
-wywołać takie funkcje, które podczas wykonywania programu wykonują funkcje systemowe,
-funkcje systemowe są dołączane bezpośrednio do wykonywanego programu.

Języki C, PL/360, PERL, utworzono do pisania systemów operacyjnych; umożliwiają bezpośredni dostęp funkcji systemowych.

end, abort:	zakończenie, zaniechanie;
load, execute:	załadowanie, wykonanie;
waitfor time:	czekanie czasowe;
allocate, free memory :	przydział, zwolnienie pamięci;
create, terminale process:	utworzenie, zakończenie procesu;
waitfor event, signal event:	oczekiwanie, sygnalizacja zdarzenia;
getprocess, setprocess attributes:	pobranie, określenie atrybutów procesu.

Niech program w C++ wywoła funkcję: **fp** = **fopen**(**"E:\plik", "r"**) wówczas:
-nastąpi wywołanie funkcji **CreateFile** podsystemu Win32 z biblioteki *Kernel32.dll*;
-po inicjacji **CreateFile** zostanie wywołana funkcja **NtCreateFile** z biblioteki *Ntdll.dll;*
Funkcja z *Ntdll.dll* uruchamia instrukcję, powodującą przejście w tryb jądra;
Funkcję **NtCreateFile** wywołuje dyspozytor usług systemowych w programie *Ntoskrnl.exe.*

Rozważmy program czytający dane z jednego pliku i kopiujący do innego pliku.
Program otwiera plik **We** i tworzy plik **Wy** ***(2 funkcje systemowe)***.
Mogą wystąpić błędy: program wydaje komunikat ***(2 funkcje systemowe)*** i kończy awaryjnie.
Jeżeli plik **We** o podanej nazwie istnieje: zaniechanie pracy ***(funkcja systemowa)*** lub usunięcie istniejącego pliku (***funkcja systemowa***).
Czytanie z pliku **We** i pisanie do pliku **Wy** ***(2 funkcje systemowe)***.
Po skopiowaniu pliku, program może zamknąć oba pliki ***(funkcja systemowa)***,
wydać komunikat ***(funkcja systemowa)***
i zakończyć działanie ***(funkcja systemowa)***.

Nawet prosty program może intensywnie używać systemu operacyjnego.

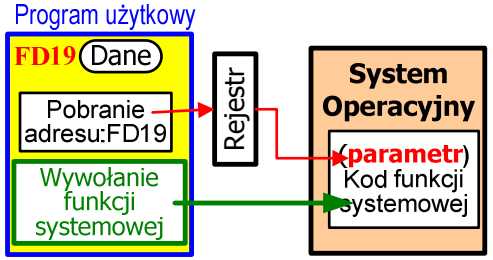
➡ **Kompilatory ukrywają przed użytkownikiem wywołania funkcji systemowych.**

Postać wywołań funkcji systemowych zależy od typu komputera.

Oprócz identyfikatora funkcji systemowej potrzebne są dodatkowe informacje, zależnie od SO i wywoływanej funkcji np.: adres i długość bufora w PAO, do którego będą przekazywane dane.

❒ Metody przekazywania parametrów do systemu operacyjnego:

1. poprzez rejestry CPU,
2. parametry przechowuje się w **bloku** lub **tablicy** w pamięci, adres bloku przekazuje się jako parametr za pośrednictwem rejestru.
3. parametry składuje się na **stosie** za pomocą programu, skąd są zdejmowane przez SO.



Rys. 3.2. Przekazanie parametru przez rejestr

☞ Metoda bloków lub stosów nie ograniczają liczby ani długości przekazywanych parametrów.

3.2.1. Nadzorowanie procesów i zadań

Jeżeli funkcja systemowa powoduje nagłe zakończenie programu lub w działaniu programu wykryto błąd *(error trap),* to wykonywany jest zrzut zawartości pamięci na dysk.

SO przekazuje sterowanie do interpretera poleceń, który czyta następne polecenie.

W systemie **interakcyjnym** zakłada się, że użytkownik wyda polecenie stosowne do błędu.

W systemie **wsadowym** interpreter przestaje wykonywać zadanie i przechodzi do następnego.

Istnieją systemy, gdzie stosowne instrukcje sterujące określają postępowania po wykryciu błędu.

● **Błąd** w **danych wejściowych zatrzymuje awaryjnie program.**

System określa kod błędu przyjmując, że zatrzymanie normalne ma kod błędu równy 0.

☞ Interpreter poleceń może kodem błędu sterować podejmowanie dalszych decyzji.

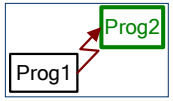
● **Proces aktywny może wymagać załadowania i wykonania innego programu.**

Interpreter poleceń uruchamia nowy program tak samo, jak gdyby zlecony przez użytkownika.

Gdzie przekazać sterowanie, gdy **nowo załadowany** program zakończy pracę ?

Jeśli po zakończeniu **nowego** programu sterowanie ma powrócić do poprzedniego programu, to należy przechować **obraz pamięci pierwszego** programu.

Gdy oba programy pracują **współbieżnie**, wówczas powstaje **nowy proces**, który trzeba uwzględnić w algorytmie wieloprogramowości.



● **Powinniśmy mieć wpływ** na **wykonanie nowego procesu**.

Należy ustalić wartości początkowych atrybutów procesu, w tym priorytetu, maksymalnego czasu wykonania itd.

Można wcześniej zakończyć wykonywanie utworzonego procesu, jeśli okaże się niepoprawny lub niepotrzebny.

Można czekać na zakończenie nowych procesów: czekanie przez określony czas *(wait time)* lub na określone zdarzenie *(wait event).*

Procesy powinny sygnalizować występowanie zdarzeń.

● **Funkcje systemowe powinny sprawdzać poprawności programu.**

Funkcje systemowe wykonują zrzut zawartości pamięci *(dump),* lub tworzą ślad działania programu *(trace)* -rejestracja ciągu instrukcji w kolejności ich wykonywania.

W jednokrokowym trybie pracy CPU po wykonaniu każdego rozkazu wchodzi w tryb obsługi specjalnej pułapki, którą przechwytuje systemowy program diagnostyczny.

● **Profil czasowy informuje** o **czasie spędzanym** w **określonych obszarach pamięci.**

Sporządzenie profilu wymaga śledzenia programu lub regularnych przerwań zegarowych.

Wystąpienie przerwania zegarowego powoduje zapamiętanie stanu licznika rozkazów programu.

Częste przerwania pozwalają na statystyczną analizę czasu wykonywania fragmentów programu.

3.2.2. Działania na plikach

create file, delete file:	utworzenie pliku, usunięcie pliku
open, *close*, *read*, *write*:	otwarcie, zamknięcie, czytanie, pisanie
reposition:	zmiana położenia
getfile, *setfile attributes*:	pobranie, określenie atrybutów pliku

Operacje tworzenia, otwierania i usuwania plików.

Operacje czytania i pisania do pliku oraz zmiany punktu odniesienia w pliku.

Operacja zamknięcia pliku, informująca że plik nie będzie używany.

Wywołanie funkcji systemowej może wymagać podania nazwy pliku i jego atrybutów.

Jeśli system plików zawiera strukturę katalogów, to jest potrzebny zbiór operacji dla katalogów.

Dla plików i katalogów należy określać wartości atrybutów i czasami nadawać wartości początkowe.

Potrzeba **dwu** funkcji systemowych: pobrania atrybutu pliku i określenia atrybutu pliku.

```
                                    //  Windows
BOOL WriteFile(
   HANDLE                hFile,
   LPCVOID               pBuffer,
   DWORD                 nNumberOfBytesToWrite,
   LPDWORD               lpNumberOfBytesWritten,
   LPOVERLAPPED          lpOverlapped
   );
```

```
                                    // Windows
BOOL ReadFile(
   HANDLE                hFile,
   LPVOID                lpBuffer,
   DWORD                 nNumberOfBytesToRead,
   LPDWORD               lpNumberOfBytesRead,
   LPOVERLAPPED          lpOverlapped
  );
```

3.2.3. Zarządzanie urządzeniami

request, *release device*	zamówienie, zwolnienie urządzenia
read, *write*, *reposition*	czytanie, pisanie, zmiana położenia
get, set device attributes	pobranie, określenie atrybutów urządzenia
logically attach	logiczne przyłączanie urządzeń
detach devices	logiczne odłączanie urządzeń

Wykonywany program może potrzebować dodatkowych zasobów: większej pamięci, plików itd.

Jeżeli zasoby są dostępne, to będą mu przyznane i sterowanie powróci do programu użytkownika; w przeciwnym razie program musi **czekać** dopóki nie będzie wystarczającej ilości zasobów.

Pliki można rozumieć jako abstrakcyjne lub wirtualne urządzenia.

Wiele funkcji systemowych dla plików jest również potrzebnych w przypadku urządzeń.

Jeżeli system ma wielu użytkowników, to należy najpierw **Poprosić** o urządzenie, aby zapewnić sobie wyłączność jego użytkowania.

Po skończeniu użytkowania urządzenia należy je **Zwolnić**.

Te czynności podobne są do systemowego otwierania i zamykania plików.

Po zażądaniu przydziału urządzenia (i uzyskaniu go) do urządzenia można odnosić operacje czytania, pisania jak w operacjach na zwykłych plikach.

> Duże podobieństwo między urządzeniami We/Wy i plikami;
>
> Wiele SO łączy je w jedną strukturę plików-urządzeń.
>
> Urządzenia We/Wy są wówczas rozpoznawane jako pliki o specjalnych nazwach.

3.2.4. Informacja

get time or data, *set time or data*	określenie czasu lub daty
get, *set system data*	pobranie, określenie danych systemowych
getprocess file or device attributes	pobranie atrybutów procesu, pliku lub urządzenia
setprocess file or device attributes	określenie atrybutów procesu, pliku lub urządzenia

Wiele funkcji systemowych służy do przesyłania informacji między programem użytkownika a SO.

Funkcje systemowe przekazującą bieżący czas i datę lub liczbę bieżących użytkowników, numer wersji systemu operacyjnego, ilość wolnej pamięci lub miejsca na dysku itd.

SO przechowuje informacje o wszystkich swoich procesach oraz zawiera funkcje systemowe udostępniające te informacje (pobranie atrybutów procesu).

Dostępne są funkcje operujące na stanie informacji o procesach (określenie atrybutów procesu).

3.2.5. Komunikacja

create, delete communication connection	utworzenie, usunięcie połączenia komunikacyjnego
send, receive messages	nadawanie, odbieranie komunikatów
transfer status information	przekazanie informacji o stanie
attach or *detach remote devices*	przyłączanie lub odłączanie urządzeń zdalnych

Model pamięci dzielonej *(shared-memory model)*: procesy posługują się systemowymi funkcjami odwzorowania pamięci *(memory map)*, aby uzyskać dostęp do obszarów PAO należących do innych procesów.

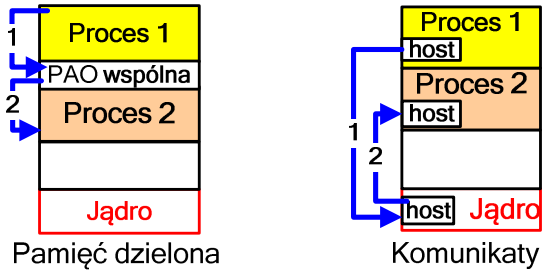
➜ SO zapobiega przenikaniu jednego procesu do pamięci innego.

Dzielenie pamięci wymaga, aby kilka procesów **zgodziło się** na usunięcie tego ograniczenia.

Wówczas procesy mogą wymieniać informację poprzez wspólnie użytkowane obszary.

-Procesy określają postać danych i ich miejsce w pamięci - **nie podlega to kontroli SO**.

-Procesy **muszą pilnować**, aby nie zapisywać jednocześnie tego samego miejsca.



Rys. 3.3. Komunikacja

Model przesyłania komunikatów *(message-passing model)*: *wymiana* informacji jest realizowana przez międzyprocesowe środki komunikacji, które dostarcza system operacyjny.

Komunikację rozpoczyna nawiązanie połączenia.

Musi być znana nazwa odbiorcy: inny proces w tej samej CPU lub proces w innym komputerze.

> **Każdy proces ma swoją nazwę procesu**, tłumaczoną na równoważny identyfikator, za pomocą którego SO odwołuje się do procesu.
>
> Tłumaczeń dokonują funkcje systemowe **pobrania nazwy sieciowej** *(get hosted)* i **pobrania nazwy procesu** *(get processed)*.

Uzyskane identyfikatory stają się parametrami funkcji systemowych **otwierania połączenia** *(open connection)* i **zamykania połączenia** *(close connection)*.

Proces odbiorcy udziela zgody na nawiązanie komunikacji za pomoc **funkcji akceptującej połączenie** *(accept connection)*.

Większość procesów realizujących połączenia stanowią tzw. **demony** *(daemons)*, będące programami systemowymi.

Wywołują one **funkcję czekania na połączenie** *(waitfor connection)* i **są budzone**, gdy połączenie zostanie nawiązane.

Źródło komunikacji zwane ***klientem*** i demon odbiorczy nazywany **serwerem**, wymieniają komunikaty za pomocą funkcji systemowych **Czytania** i **Pisania komunikatu**.

Wywołanie funkcji zamknięcia połączenia kończy komunikację.

☛ **Pamięć dzielona** zapewnia maksymalną szybkość i wygodę komunikacji, gdyż w obrębie jednego komputera komunikacja może przebiegać z szybkością działania PAO.
Pojawiają się problemy z zakresu ochrony i synchronizacji.

☛ **Przesyłanie komunikatów** jest przydatne w wymianie mniejszych ilości danych.
Jest łatwiejsze do zrealizowania w komunikacji międzykomputerowej niż metoda pamięci dzielonej.

3.3. Programy systemowe

Programy systemowe tworzą środowisko do opracowywania i wykonywania innych programów.

Niektóre są interfejsami użytkownika do funkcji systemowych.

Manipulowanie plikami: tworzenie, usuwanie, kopiowanie, przemianowywania, itp.

Informowanie o stanie systemu: pobierają z systemu datę, czas, ilość dostępnej pamięci lub miejsca na dysku, liczbę użytkowników.

Tworzenie i zmienianie zawartości plików: rozmaite odmiany edytorów.

Obsługa języków programowania: kompilatory, asemblery oraz interpretery.

Ładowanie i wykonywanie programów: programy ładowania kodu o adresach absolutnych i kodu przemieszczalnego, konsolidatorów oraz ładowaczy nakładek.

Komunikacja między różnymi systemami komputerowymi.

Interpreter poleceń: najważniejszy program systemowy; pobiera i wykonuje polecenia użytkownika.

◗ Dwa sposoby realizacji poleceń interpretera:

1. **Interpreter poleceń sam zawiera kod wykonujący polecenia.**

   Polecenie usunięcia pliku powoduje skok interpretera do części ***własnego*** kodu, która obsłuży pobranie parametrów polecenia i wywoła odpowiednią funkcję systemową.

   Rozmiar interpretera zależy od liczby poleceń - każde polecenie wymaga odrębnego kodu.

2. **Polecenia wykonywane** są **przez specjalne programy systemowe** - system UNIX.

   Interpreter ***„nie rozumie***" poleceń, służą tylko do zidentyfikowania pliku, który ma być załadowany do pamięci i wykonany.

   Polecenie **delete *Par*** odnajduje plik o nazwie ***delete***, ładuje go do pamięci i wykonuje z parametrem ***Par***.

Funkcja związana z poleceniem **delete** jest określona przez kod zawarty w pliku ***delete***.

-Łatwość dołączania nowych poleceń do systemu: utworzyć nowy plik z nazwą funkcji.

-Interpreter może być niewielki i nie trzeba zmieniać go przy dodawaniu nowych poleceń.

Kod wykonujący polecenie jest osobnym programem i SO musi zawierać mechanizm przekazywania parametrów od interpretera poleceń do programu systemowego.

Interpreter poleceń i program systemowy nie muszą przebywać w tym samym czasie w pamięci.

Załadowanie i wykonanie programu trwa dłużej niż zwyczajny skok do innego miejsca kodu bieżąco wykonywanego programu.

❒ **Interpreter poleceń** w **środowisku MS- DOS**

System MS-DOS jest jednozadaniowy z interpreterem poleceń wywoływanym na początku pracy komputera.

System wprowadza program do PAO, nawet kosztem części własnego kodu, aby zapewnić mu najwięcej przestrzeni.

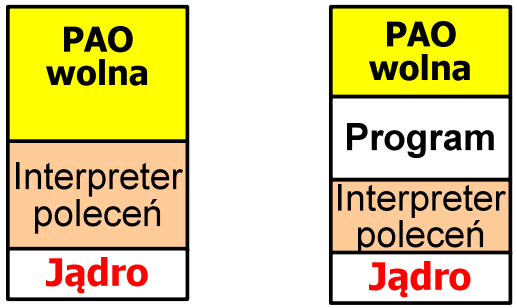
Program uruchomiony:

-wystąpi błąd i uaktywni się pułapka systemowa;

-program zatrzyma się, wykonując funkcję systemową.

W obu przypadkach kod błędu zostanie przechowany w pamięci SO.

Po wykonaniu tych czynności wznawia działanie mały fragment interpretera poleceń, który nie został zniszczony przez kod programu.



Rys. 3.4. MS-DOS

Najpierw **interpreter** powtórnie ładuje z dysku resztę swojego kodu, a po zakończeniu udostępnia ostatni kod błędu użytkownikowi lub następnemu programowi.

----------MS-DOS zawiera mechanizm pozwalający na uproszczone działania współbieżne----------

Program **TSR** „przechwytuje przerwanie", po czym kończy pracę za pomocą funkcji systemowej „zakończ i pozostań w pogotowiu" (**T***erminale and* **S***tay* **R***esident).*

Może on przechwytywać przerwania zegarowe wskutek umieszczenia adresu jednego z jego podprogramów na wykazie procedur obsługi przerwań wywoływanych po wyzerowaniu się czasomierza systemowego.

Procedura **TSR** będzie wykonywana przy każdym impulsie zegarowym.

Funkcja ***systemowa TASR*** powoduje **zarezerwowanie** przez MS-DOS **obszaru** na pomieszczenie programu TSR, dzięki czemu obszar ten **nie będzie zapisany** przez wprowadzany ponownie do pamięci **interpreter** poleceń.

3.4. Struktura systemu

System Operacyjny dzielony jest na małe części.

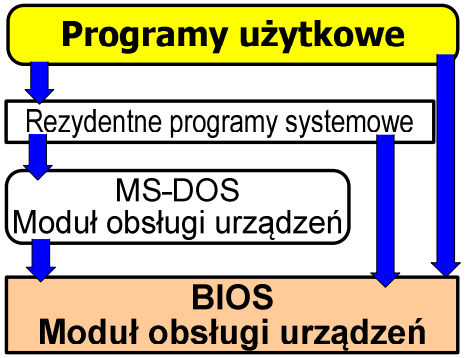
Każda część to fragmentem systemu, ze ***starannie*** określonym wejściem, wyjściem i funkcjami.

❒ **Struktura prosta**

Wiele systemów nie ma dobrze określonej struktury.

Powstały jako proste i ograniczone SO, które rozrastając się, przekraczały pierwotne założenia.

Przykładem takiego systemu operacyjnego jest MS-DOS.



Rys. 3.5. Struktura MS-DOS

Napisany pod kątem osiągnięcia maksymalnej funkcjonalności przy oszczędności miejsca, gdyż sprzęt wymuszał ograniczenia.

W systemie 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.

Błędne programy użytkowe mogą załamać cały system.

Możliwości MS-DOS ograniczał sprzęt.

Mikroprocesor Intel 8088, nie miał dualnego trybu pracy, ani ochrony sprzętowej.

❒ **Struktura warstwowa**

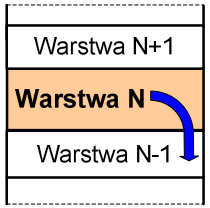
Zstępująca metoda projektowania umożliwia zdefiniowanie ogólnych funkcji i cech systemu oraz wyodrębnić jego części składowe.

Zasada ukrywania informacji daje swobodę w realizacji procedur niskopoziomowych, pod warunkiem, że zewnętrzne interfejsy z procedurami pozostaną niezmienione.

Modularyzację systemu uzyskano przez podział SO na warstwy, przy czym każda następna warstwa jest zbudowana powyżej niższych warstw.

Najwyższą warstwą jest interfejs z użytkownikiem.

Najniższą warstwę (0) stanowi sprzęt.



Rys. 3.6. Warstwy

Warstwa systemu operacyjnego jest implementacją abstrakcyjnego obiektu.

Zawiera struktury danych, procedury, mogące być wywołane z wyższych warstw.

Warstwa N-ta może wywołać operacje dotyczące niższych warstw.

Każda warstwa używa funkcji i korzysta z usług tylko niżej położonych.

To podejście upraszcza implementację, wyszukiwanie błędów i weryfikację systemu.

Pierwsza warstwa może być poprawiana bez troski o resztę systemu, ponieważ - z definicji - do realizacji swoich funkcji używa tylko **podstawowego sprzętu**.

Po uruchomieniu pierwszej warstwy, przystępuje się do opracowania drugiego poziomu itd.

Jeśli podczas uruchamiania kolejnej warstwy zostanie wykryty błąd, musi on pochodzić z danej warstwy, ponieważ warstwy niższe już sprawdzono.

- Implementacja każdej warstwy opiera się na operacjach dostarczanych przez warstwy **niższe**.
- Warstwa **nie musi** nic wiedzieć o implementacji tych operacji; wie tylko, co operacje te robią.

Każda warstwa ukrywa istnienie pewnych struktur danych, operacji i sprzętu przed warstwami wyższych poziomów.

Warstwy zastosowano pierwszy raz w SO **THE** z Technische Hogeschool w Eindhoven.

Koncepcje warstw mogą być różnorodne.

Warstwy systemu THE -1968

5: Programy użytkowe
4: Buforowanie urządzeń We/Wy
3: Program obsługi konsoli operatora
2: Zarządzanie pamięcią
1: Planowanie przydziału procesora
0: Sprzęt

W systemie Venus niższe poziomy (2 - 4), przeznaczone do planowania przydziału procesora i zarządzania pamięcią, napisano jako mikroprogramy.

Uzyskano szybsze działanie i większą przejrzystość interfejsu między warstwami.

Warstwy systemu Venus

6: Programy użytkowe
5: Obsługa i planowanie przydziału urządzeń
4: Pamięć wirtualna
3: Kanał We/Wy
2: Planowanie przydziału procesora
1: Interpreter instrukcji
0: Sprzęt

Jak efektywnie zdefiniować lokalizację warstw ?

- Obsługa **pamięci dyskowej** powinna być na **niższym poziomie** niż procedury zarządzania PAO.

  Procedury PAO korzystają z pamięci dyskowej w ramach koncepcji pamięci wirtualnej

zarządzanie PAO
zarządzanie **HDD**

- Obsługa **pamięci dyskowej** powinna być **powyżej programu** planującego przydział CPU.

  Program obsługi pamięci dyskowej może być **zmuszony do czekania** na operację We/Wy, a w tym czasie CPU może otrzymać nowy przydział.

zarządzanie **HDD**
przydział CPU

- W dużym systemie program **planujący przydział CPU** może mieć **więcej informacji** o wszystkich aktywnych procesach, niż daje się pomieścić w PAO.

  Powstaje potrzeba wymiany tych informacji między pamięciami, a to wymaga umieszczenia programu obsługi **pamięci dyskowej poniżej** programu planującego przydział CPU (korzystanie z pamięci wirtualnej).

przydział CPU
zarządzanie **HDD**

Realizacje warstwowe bywają mniej wydajne od innych.

Program użytkowy wykonując operację We/Wy wywołuje funkcję systemową,

która prowadzi do warstwy We/Wy,

a ta wywołuje warstwę zarządzania pamięcią

i poprzez warstwę planowania przydziału CPU dociera do sprzętu.

W każdej warstwie mogą występować zmiany parametrów, przenoszenie danych itd.

Każda warstwa *zwiększa* koszt odwołania do systemu, zatem łącznie wykonanie funkcji systemowej trwa dłużej niż w systemie nie podzielonym na warstwy.

System UNIX złożony jest z warstw.

Składa się z dwu odrębnych części: **jądra** i **programów systemowych**.

Jądro dzieli się na ciąg interfejsów i programów obsługi urządzeń, które dodawano i rozszerzano w trakcie rozbudowy systemu.

To, co znajduje się poniżej interfejsu funkcji systemowych a powyżej sprzętu, stanowi jądro.

Jądro -poprzez funkcje systemowe - udostępnia system plików, planowanie przydziału procesora, zarządzanie pamięcią itp.

Programy **systemowe** korzystają z udostępnianych przez jądro funkcji systemowych w celu wykonywania działań takich jak kompilowanie programów lub operowanie plikami.

Interfejs programisty definiuje funkcje systemowe.

Interfejs użytkownika ogólnie dostępne programy systemowe.

Oba interfejsy stanowią kontekst, udostępniany przez jądro.

Rys. 3.7. Struktura systemu UNIX

Projektuje się systemy złożone z niewielu, lecz bardziej funkcjonalnych warstw.

Wykorzystano zalety modularnego programowania, ograniczając problemy z definiowaniem wzajemnych zależności między warstwami.

System **OS/2** został zrealizowany z większym uwzględnieniem warstwowości niż Windows 95.

Wprowadzono wielozadaniowość, podwójny tryb operacji.

Użytkownik nie może bezpośrednio korzystać z udogodnień niskiego poziomu.

Umożliwia to zwiększenie kontroli nad sprzętem i lepsze rozeznanie co do zasobów eksploatowanych przez programy użytkowników.

Rys. 3.8. Struktura systemu OS/2

Pierwsze wersje systemu Windows NT miały klasycznie warstwową organizację i okazały się mało wydajne w porównaniu z systemem Windows 95.

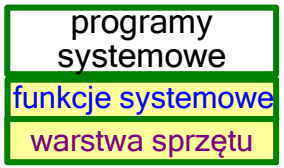
W Windows NT 4.0 przesunięto warstwy z przestrzeni użytkownika do przestrzeni jądra, integrując ze sobą.

3.5. Maszyna wirtualna

Na najniższym poziomie -w warstwowym systemie operacyjnym- znajduje się sprzęt.

W **jądrze** systemu użyto **rozkazów sprzętowych** do utworzenia zbioru **funkcji systemowych**.

Powyżej jądra systemu -w programach systemowych- można używać ***funkcji systemowych*** i **rozkazów sprzętowych,** i pod pewnymi względami, nie odróżniać **funkcji** od **rozkazów**.



Sprzęt i **funkcje systemowe** - **w programach systemowych** - **traktuje się tak, jak gdyby należały one** do **tego samego poziomu**.

Pewne systemy umożliwiają wywoływanie programów systemowych przez programy użytkowe.

Programy systemowe są o szczebel wyżej niż inne procedury.

Programy użytkowe mogą to **wszystko**, co **znajduje się** w **hierarchii poniżej nich, traktować** jako **część maszyny.**

Koncepcja maszyny wirtualnej (**VM**) jest wynikiem tak pojmowanej warstwowości.

Stosując przydział procesora, pamięć wirtualną, system operacyjny może tworzyć złudzenie, że wiele procesów pracuje na własnych CPU z własną pamięcią.

Maszyna wirtualna nie dostarcza dodatkowych funkcji, tworzy jedynie interfejs identyczny z podstawowym (symulowanym) **sprzętem.**

Każdy proces otrzymuje (wirtualną) **kopię komputera będącego podstawą systemu.**

Zasoby fizycznego komputera są dzielone w celu utworzenia maszyn wirtualnych.

Planowanie przydziału CPU daje wrażenie, że każdy użytkownik ma własny procesor.

Spooling i system plików pozwalają utworzyć wirtualne czytniki i wirtualne drukarki.

Zwykły terminal do pracy z podziałem czasu przejmuje rolę wirtualnej konsoli operatorskiej.

Pewnym problemem maszyny wirtualnej jest **realizacja systemu dysków.**

Niech maszyna z dwoma napędami dysków ma symulować **5** maszyn wirtualnych.

Oprogramowanie **VM** zajmuje dużo obszaru dysku na obsługę pamięci wirtualnej i spoolingu.

Stosuje się dyski wirtualne (minidysk), które są identyczne z fizycznymi z wyjątkiem rozmiaru.

System implementuje minidysk, przydzielając mu taką liczbę ścieżek dysku fizycznego, na jaką minidysk zgłosi zapotrzebowanie.

Użytkownik otrzymuje własne maszyny wirtualne i może traktować je jak maszyny bazową.

❒ **Realizacja**

Fizyczna maszyna bazowa ma dwa tryby pracy: **użytkownika** i **monitora.**

☞ Oprogramowanie **VM** może pracować w trybie monitora ➜ ponieważ jest ono SO.

☛ Sama **V**irtualna **M**aszyna może działać **tylko** w trybie **użytkownika**.

VM powinna mieć **dwa tryby pracy** - użytkownika i monitora, z których **każdy** pracuje w **fizycznym** trybie **użytkownika**.

Czynności, które zmieniają tryb użytkownika na tryb monitora w rzeczywistej maszynie (wywołanie funkcji systemowej), muszą również powodować przejście w **VM** od **wirtualnego** trybu użytkownika do **wirtualnego** trybu monitora.

Kiedy **wirtualny** monitor **VM** przejmie sterowanie, może zmienić zawartości rejestrów i licznika rozkazów **VM**, aby zasymulować efekt wywołania funkcji systemowej.

Może wznowić działanie **VM**, zaznaczając, że jest ona w trybie **wirtualnego** monitora.

Jeśli VM zacznie czytać z jej wirtualnego czytnika, to wykona **uprzywilejowany** rozkaz We/Wy.

☛ **Pamiętaj**: **V**irtualna **M**aszyna pracuje w **fizycznym** trybie **użytkownika**.

Rozkaz We/Wy spowoduje przejście do **wirtualnego** monitora **VM**.

Wirtualny monitor **VM** musi zasymulować wynik wykonania rozkazu We/We.

Najpierw odnajduje plik buforowy implementujący dany wirtualny czytnik.

Następnie tłumaczy czytanie z **wirtualnego** czytnika na **czytanie** utożsamionego z nim buforowego **pliku dyskowego**, i przesyła wirtualny „obraz karty" do wirtualnej pamięci **VM**.

Na koniec wznawia działanie maszyny wirtualnej.

Stan **V**irtualnej **M**aszyny uległ takiej samej zmianie, jak gdyby wykonano rozkaz We/We za pomocą prawdziwego czytnika w rzeczywistej maszynie pracującej w rzeczywistym trybie **monitora.**

Czas rozróżnia działanie maszyny wirtualnej i **rzeczywistej**.

Wirtualne operacje We/Wy są symulowane przy użyciu buforowego pliku dyskowego albo interpretowane.

Procesor często musi pracować wieloprogramowo dla wielu maszyn wirtualnych.

W skrajnym przypadku może być konieczne symulowanie wszystkich rozkazów.

Maszyny wirtualne IBM realizowane na sprzęcie IBM są wydajne, gdyż ***zwykłe*** **rozkazy** mogą być wykonywane wprost **za pomocą sprzętu.**

Symulacja dotyczy tylko rozkazów **uprzywilejowanych**, które z tego powodu działają wolniej.

❒ **Korzyści**

-W środowisku **VM** istnieje pełna ochrona różnorodnych zasobów systemowych.

-Każda maszyna wirtualna jest całkowicie odizolowana od innych maszyn wirtualnych,

Problem: Nie ma bezpośredniej możliwości wspólnego użytkowania zasobów.

Dwa podejścia dzielenia zasobów:

❶ Wspólne użytkowanie **minidysku** (struktura dysku fizycznego ale realizacja programowa).

❷ Sieć VM, z których każda wysyła informacje przez wirtualną sieć komunikacyjną.
Sieć wzorowana jest na fizycznych sieciach, ale realizowana programowo.

Maszyna wirtualna jest znakomita do **badań nad systemami operacyjnymi.**

Programiści systemowi dostają własną maszynę wirtualną i na niej, zamiast na maszynie fizycznej, realizują własne strategie rozwojowe systemu.

Maszyny wirtualne są sposobem rozwiązywania zagadnień zgodności systemów.

Istnieją tysiące programów dostępnych w systemie MS-DOS na procesorach firmy Intel.

Aby były dostępne dla komputerów firmy Sun Microsystems wystarczy utworzyć wirtualną maszynę Intel nadbudową nad rdzennym procesorem.

Jeśli CPU ten jest wystarczająco szybka, to program systemu MS-DOS działa szybko, chociaż każdy jego rozkaz jest tłumaczony.

3.6. Koncepcja projektowanie systemu

Projekt Systemu Operacyjnego w dużym stopniu zależy od wyboru sprzętu i typu systemu.

Należy określić: -**cel użytkownika**
-**cel systemu**
oraz **stworzyć specyfikację systemu**.

Użytkownik wymaga aby był: -wygodny,
-łatwy w użyciu i nauce,
-niezawodny,
-bezpieczny i szybki.

Projektant chciałby aby był: -łatwy do zaprojektowania, realizacji i pielęgnowania,
-elastyczny, niezawodny,
-wolny od błędów i wydajny.

Sformułowanie specyfikacji i **projektowanie systemu operacyjnego jest pracą twórczą**.

Istnieją pewne ogólne zasady.

Inżynieria oprogramowania zawiera zbiór zasad, z których można skorzystać projektując SO.

Ważną zasadą projektowania jest **oddzielenie polityki** od **mechanizmu**.

Mechanizmy określają, jak czegoś dokonać.

Mechanizmem ochrony CPU jest **czasomierz**.

Polityka decyduje, co ma być zrobione.

Polityką są **ustawienia** czasomierza dla poszczególnych użytkowników.

Odseparowanie polityki od mechanizmu daje elastyczność projektowania.

☛ Zmiana polityki może pociągać zmiany leżącego u jej podłoża mechanizmu.

☛ Przy mechanizmie ogólnym zmiana polityki wymaga jedynie przedefiniowania parametrów.

Podjęto decyzję **polityczną:** pierwszeństwo mają programy intensywnie korzystającym z **We/Wy** nad programami używającymi intensywnie **CPU**.

Jeżeli stosowny mechanizm został wydzielony i uniezależniony od polityki, to w innym systemie komputerowym można łatwo zrealizować politykę przeciwną.

Jeżeli **mikrojądro** jest podstawą SO to rozdział między mechanizmem a polityką przebiega w sposób skrajny, realizując podstawowy zbiór elementarnych działań składowych.

Składowe te są niemal całkowicie niezależne od sposobu ich implementacji.

Umożliwiają dołączanie zaawansowanych mechanizmów i polityk w formie modułów jądra utworzonych przez użytkowników lub wprost za pomocą programów użytkowych.

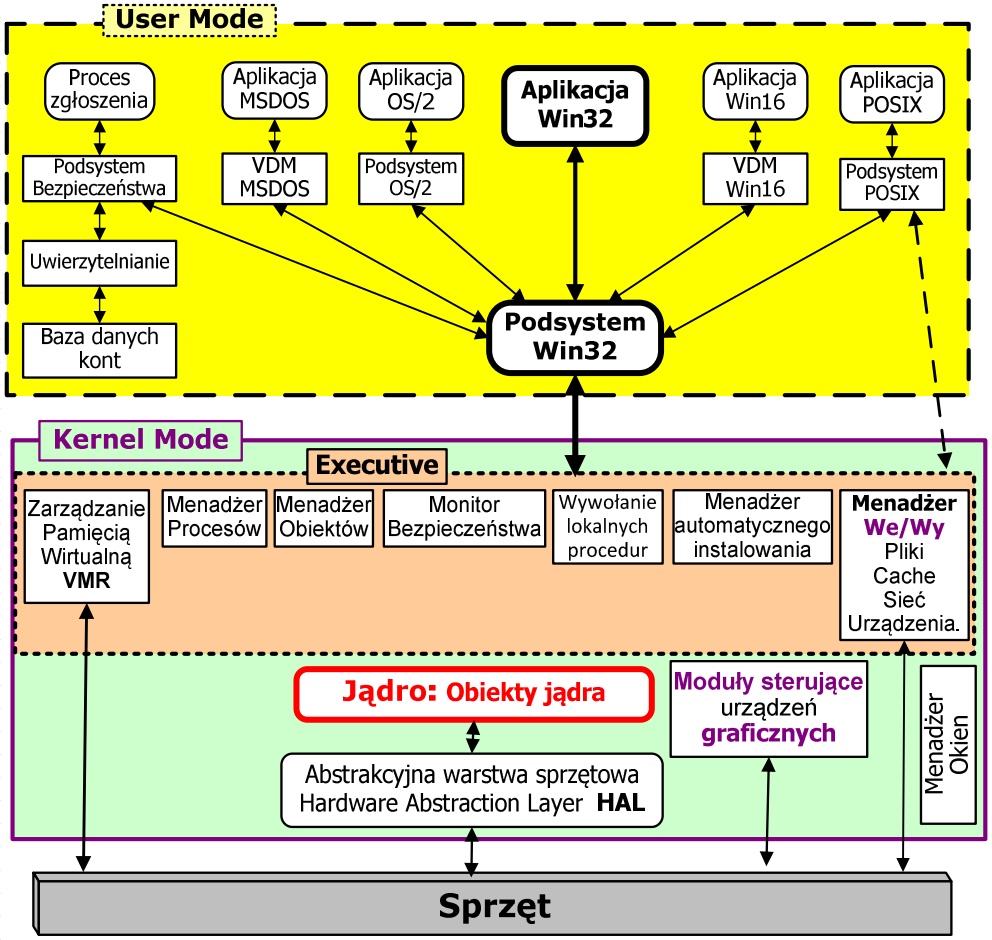
W systemie Apple Macintosh zakodowano zarówno mechanizmy, jak i sposoby ich użycia.

Wszystkie aplikacje mają podobne interfejsy, gdyż sam interfejs jest wbudowany w jądro.

3.7. Struktura Systemu WINDOWS

POSIX (*Portale Opreating system based on Unix*) standard przenośnego systemu operacyjnego, przeznaczonego do obsługi środowiska komputerowego (początek IEEE P1003.1 w 1986 r.).

System Windows XP zgodny jest na poziomie kodu źródłowego dla aplikacji standardu **POSIX**.



Warstwa **HAL** –oprogramowanie, ukrywające szczegóły sprzętowe przed górnymi warstwami SO.

Moduły sterujące **We/Wy** oraz moduły obsługi urządzeń **graficznych** mogą bezpośrednio kontaktować się ze sprzętem ze względu na wydajność.

Jądro: -planuje procesy,
-synchronizuje CPU na niskim poziomie,
-obsługuje przerwania i sytuacje wyjątkowe,
-działania naprawcze po awarii zasilania.

VDM **V**irtual **D**OS **M**achine -zawiera jednostkę wykonywania rozkazów I486 oraz procedury emulujące funkcje ROM BIOS systemu MS-DOS.

System Windows udostępnia Interfejs Programowania Aplikacji **Win32 API**.

API -***A**pplication **P**rogramming **I**nterface* to zestawu funkcji, używanych w celu komunikowania się z systemem operacyjnym.

Plik nagłówkowy **<windows.h>** zawiera ich deklaracje.

- Dostęp do Obiektów Jądra:
  wywołanie funkcji **Create**NazwaObiektu (zwraca uchwyt)
- Wspólne korzystanie z obiektów:
  dziedziczenie uchwytu, nadanie nazwy obiektowi; DuplicateHandle;
- Zarządzanie procesami:
  klasy priorytetów procesu;
- Komunikacja międzyprocesowa:
  wspólne używanie Obiektów Jądra, przekazywanie komunikatu.
- Zarządzanie pamięcią:
  pamięć wirtualna, pliki odwzorowane w pamięci, sterty, lokalna pamięć wątku.

3.7.1. Obiekty jądra

Obiekty jądra powstają w wyniku wywołania funkcji zaczynających się od **Create*nazwaObiektu*.**

Każdy **ObiektJądra** jest **blokiem pamięci** używanym bezpośrednio **tylko** przez **jądro**.

Blok ten to struktura danych, której składowe przechowują informacje o Obiekcie.

Pewne składowe jak: deskrytor_bezpieczeństwa, licznik_dostępów występują we wszystkich rodzajach Obiektów.

Aplikacja nie może dostać się do struktur **O**biekt**J**ądra i bezpośrednio zmienić ich zawartości.

Każdy **O**biekt**J**ądra jest dostępny za pomocą specjalnych funkcji, przewidzianych dla Niego.

Wywołana funkcja do utworzenia konkretnego **O**biekt**J**ądra zwraca **uchwyt** jednoznacznie identyfikujący nowy Obiekt.

☛ Niektórym Obiektom Jądra można opcyjnie nadawać unikalne **nazwy**.

Uchwyt może być używany tylko przez **wątek** działający w ramach tego samego procesu.

Standardowe użycie danego uchwytu w wywołaniu funkcji z poziomu drugiego procesu zakończy się niepowodzeniem.

ObiektyJądra są własnością jądra, a nie procesu,

☞więc inny **proces** też może mieć do nich dostęp

Uchwyty Obiektów**J**ądra **nie są wartościami ogólnosystemowymi,** są związane z określonymi procesami, co wyklucza ich **proste** współużytkowanie.

Metody dzielenia Obiektów z innymi procesami: -dziedziczenia uchwytów,
-nadawania im nazw,
-tworzenia duplikatów.

Jeśli proces wywołuje funkcję tworzącą jakiś **O**biekt**J**ądra, a następnie proces ten kończy swoje działanie, ObiektJądra **niekoniecznie** musi zostać zniszczony.

Jeśli **O**biekt**J**ądra utworzony przez jeden proces używany jest przez inny proces, jądro nie usunie tego Obiektu, dopóki drugi proces będzie z niego korzystać.

➔ **O**biekt**J**ądra może żyć dłużej niż proces, który go utworzył.

Jądro wie, ile procesów używa konkretnego **O**biekt**J**ądra.

Każdy Obiekt wyposażony jest w **licznik użyć**, który po utworzeniu **O**biektu ustawiany jest na 1.

Po każdym uzyskaniu dostępu do **O**biektu przez **inny proces** jego licznik użyć jest zwiększany o 1.

Po zakończeniu działania **procesu** jądro automatycznie zmniejsza licznik użyć.

Jeśli licznik użyć ObiektuJądra zejdzie do **0**, jądro zniszczy ten Obiekt.

Mechanizm gwarantuje usuwanie Obiektów, do których nie odwołuje się już żaden proces.

Obiekty jądra mogą być chronione za pomocą deskryptora bezpieczeństwa.

Niemal wszystkie funkcje tworzące **O**biekty**J**ądra, mają wśród argumentów wskaźnik do struktury **SECURITY_ATTRIBUTES**.

Deskryptor bezpieczeństwa opisuje, kto utworzył Obiekt i kto może mieć dostęp do niego.

Wykorzystywane są zwykle w aplikacjach serwerowych.

Dla aplikacji uruchamianej po stronie klienta, można zignorować tę możliwość.

W większość aplikacji w miejscu SECURITY_ATTRIBUTES podaje się NULL.

Powoduje to utworzenie **O**biektu**J**ądra ze **standardowym** zabezpieczeniem, co oznacza, że wszyscy członkowie grupy administratorów oraz **twórca O**biektu będą mieli pełny dostęp do niego, **a pozostali żadnego dostępu**.

W trakcie inicjalizacji procesu system alokuje **TabliceUchwytów** dla tego procesu, która jest przeznaczona **wyłącznie** na **O**biekty**J**ądra.

W momencie inicjalizacji procesu jego **T**ablica**U**chwytów jest pusta.

Wątek procesu wywołuje funkcję tworzącą **O**biekt**J**ądra.

Jądro alokuje blok pamięci na ten Obiekt, inicjalizuje go, a następnie przegląda Tablicę procesu z uchwytami, szukając wolnego miejsca.

Wartość uchwytu jest zazwyczaj indeksem **T**ablicy**P**rocesu z uchwytami, pod którym zapisana jest informacja **O**biekcie.

W Windows zwracana wartość oznacza liczbę bajtów od początku tablicy uchwytów, a nie sam indeks.

Obiekty **USER** albo **GDI** *(Graphics Device Interface*) **nie są O**biektami**J**ądra.

Czy ikona jest obiektem **USER**, czy też **O**biektem**J**ądra?

Żadna funkcja tworząca obiekty USER lub **GDI nie ma parametru SECURITY_ATTRIBUTES.**