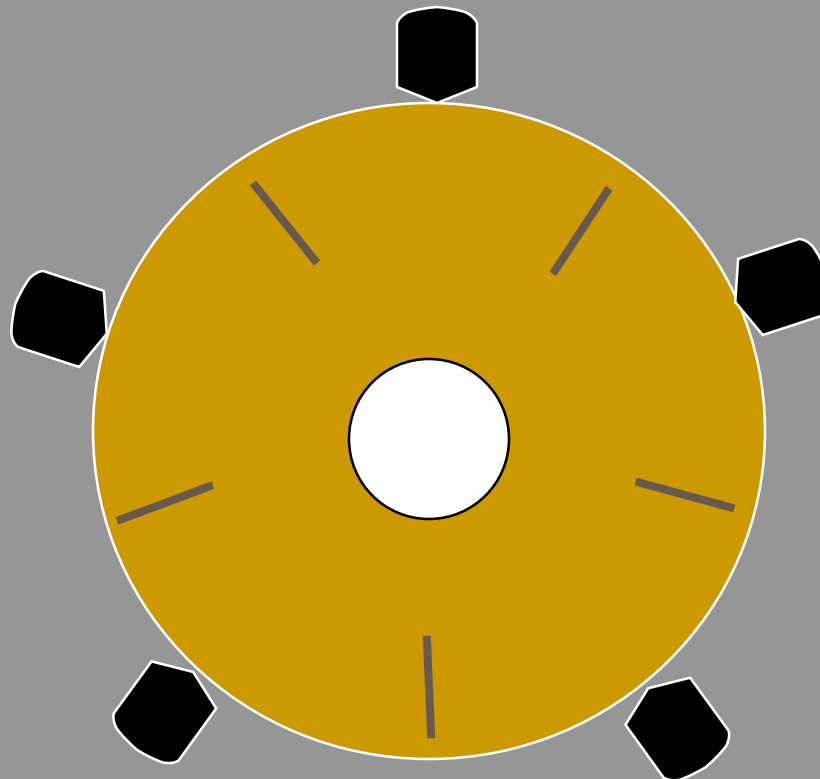


Problem filozofów



Kiedy myślący filozof poczuje głód,usiłuje podnieść najpierw lewą, a potem prawą pałeczkę. Po zakończonym jedzeniu odkłada pałeczki z powrotem na stół.

Blokada (pat, zakleszczenie)

Przypadek: jednocześnie każdy z filozofów podniósł jedną pałeczkę i czeka na drugą.

Skutek: umrą z głodu

Przykład komputerowy:

Jeden proces pisze na drukarce, a drugi na taśmie.

W czasie tych operacji pierwszy proces żąda taśmy, a drugi - drukarki. Obydwa zaczynają czekanie aż zasób się zwolni.

W chwili obecnej rzeczywiste systemy operacyjne nie rozwiązują jeszcze problemu zakleszczeń, gdyż są one stosunkowo rzadkie (i mogą być likwidowane przez operatora), ale w niedalekiej przyszłości ten problem stanie przed WAMI.



Blokada -warunki wystąpienia

Do zakleszczenia może dojść, jeśli spełnione są równocześnie warunki:


- wzajemne wykluczanie - co najmniej jeden zasób musi być niepodzielny (w danym czasie może go używać tylko jeden proces)
- Przetrzymywanie i oczekiwanie - przynajmniej jeden proces przetrzymuje jakiś zasób, ponieważ czeka na przydział dodatkowego innego zasobu, przetrzymywanego właśnie przez inny proces,
- Brak wywłaszczeń - zasób może być zwolniony jedynie z inicjatywy przetrzymującego, np. po zakończeniu procesu,
- Czekanie cykliczne: P_1 czeka na zasób przetrzymywany przez P_2 , P_2 czeka na oddanie przez P_3 ... P_n czeka na zwolnienie zasobu przez P_1








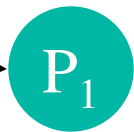
Graf przydziału zasobów

 - proces n

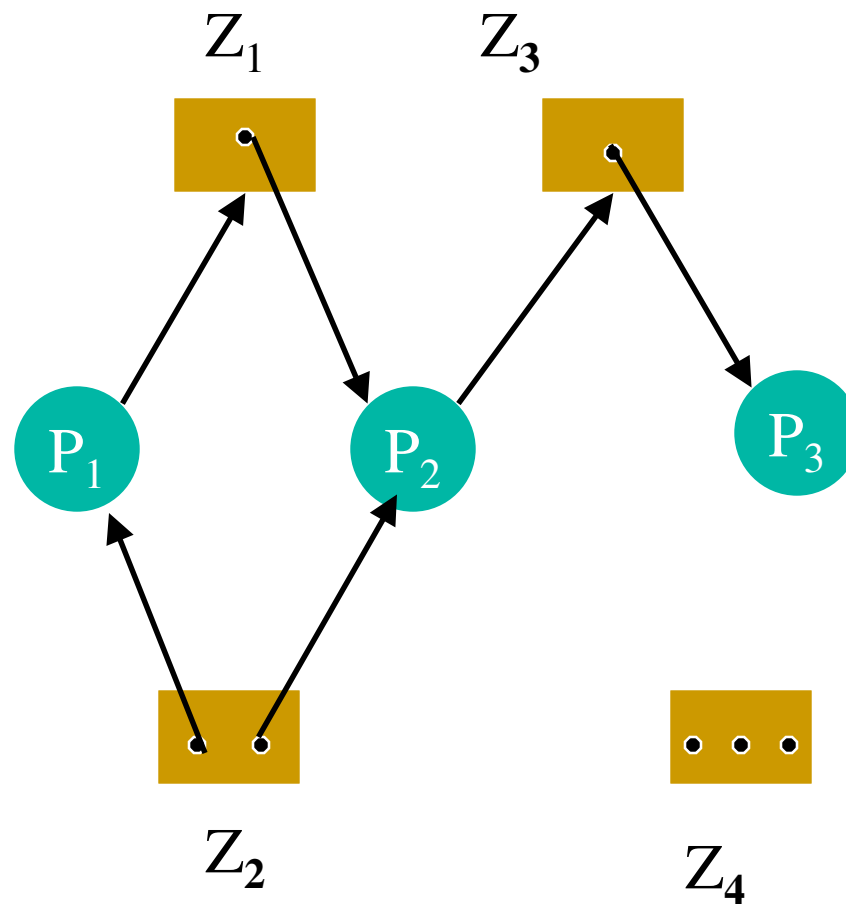
Z_n

 - zasób n , 3 egzemplarze

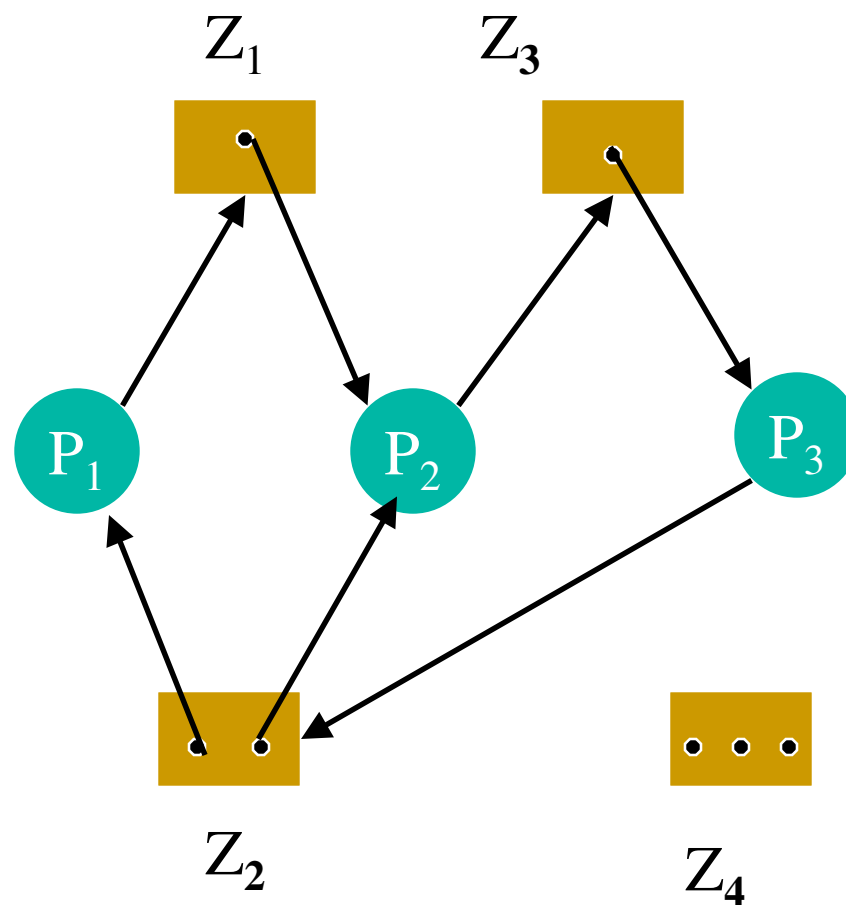
   Z_3 - proces P_3 żąda zasobu Z_3
(krawędź zamówienia)

   P_1 - proces P_1 trzyma egzemplarz zasobu Z_2
(krawędź przydziału)

Graf przydziału zasobów



Graf przydziału zasobów z zakleszczeniem



Warunki wystąpienia zakleszczenia

- Jeśli graf przydziału zasobów nie ma cykli, to nie ma zakleszczenia,
- Jeśli zasób Z_1 każdego typu ma tylko jeden egzemplarz, to istnienie cyklu jest warunkiem koniecznym i dostatecznym do wystąpienia blokady,
- Jeśli istnieje po kilka egzemplarzy zasobu, to istnienie cyklu jest jedynie warunkiem koniecznym wystąpienia blokady



Metody postępowania z zakleszczeniami

- Stosowanie protokołu gwarantującego, że system nigdy nie wejdzie w stan zakleszczenia,
- Pozwolenie na występowanie zakleszczeń i podjęcie stosownych działań po takim fakcie,
- Zlekceważenie problemu (np. UNIX) - założenie, że zakleszczenia nie pojawiają się.

Praktycznie w takich systemach zakleszczenia pojawiają się rzadko (np. raz do roku), więc raczej się nie inwestuje w kosztowne mechanizmy do unikania zakleszczeń lub ich likwidacji, a wykonuje się to ręcznie.

Zapobieganie zakleszczeniom

Trzeba zapewnić, aby nie wystąpił co najmniej jeden z warunków koniecznych:

- Wzajemne wykluczanie - nie można uniknąć tego warunku, gdyż pewne zasoby są z natury niepodzielne, np. drukarki,
- Przetrzymanywanie i oczekiwanie - trzeba zagwarantować, że gdy proces zamawia jakiś zasób, to nie przetrzymuje innych zasobów
 - proces otrzymuje wszystkie zasoby przed rozpoczęciem działania,
 - proces może zamówić zasób, jeśli wcześniej oddał wszystkie pozostałe

Wada- słabe wykorzystanie zasobów, możliwość głodzenia

Zapobieganie zakleszczeniom, cd.

- Brak wywłaszczeń - trzeba dopuścić, aby nie tylko proces mógł zwolnić zasoby
 - jeśli proces nie może dostać żadanego zasobu, to traci pozostałe i czeka na nie,
 - jeśli proces zamawia jakieś zasoby, to się sprawdza czy są dostępnejeśli tak, to się je przydziela,
jeśli zasób jest trzymany przez inny proces, który czeka na jakieś zasoby, to mu się zasoby odbiera i daje zamawiającemu,
w przeciwnym wypadku proces czeka i może utracić swoje zasoby.

Zapobieganie zakleszczeniom, cd.

- Czekanie cykliczne - aby nie wystąpiło, trzeba zasobom tego samego typu nadać liczbowe identyfikatory (np. taśmy - 2, drukarki - 5) i trzeba dbać, aby proces zamawiał zasoby w rosnącym porządku numeracji, ewentualnie aby zwalniał zasoby o numeracji wyższej od zamawianego.

Unikanie zakleszczeń

- Każdy proces deklaruje maksymalną liczbę zasobów danego typu
- Stan przydziału zasobów jest określony przez liczbę dostępnych i przydzielonych zasobów oraz przez maksymalne zapotrzebowanie na zasoby
- Algorytm unikania zakleszczeń sprawdza stan przydziału zasobów i blokuje niektóre przydziały, aby nie doszło do czekania cyklicznego

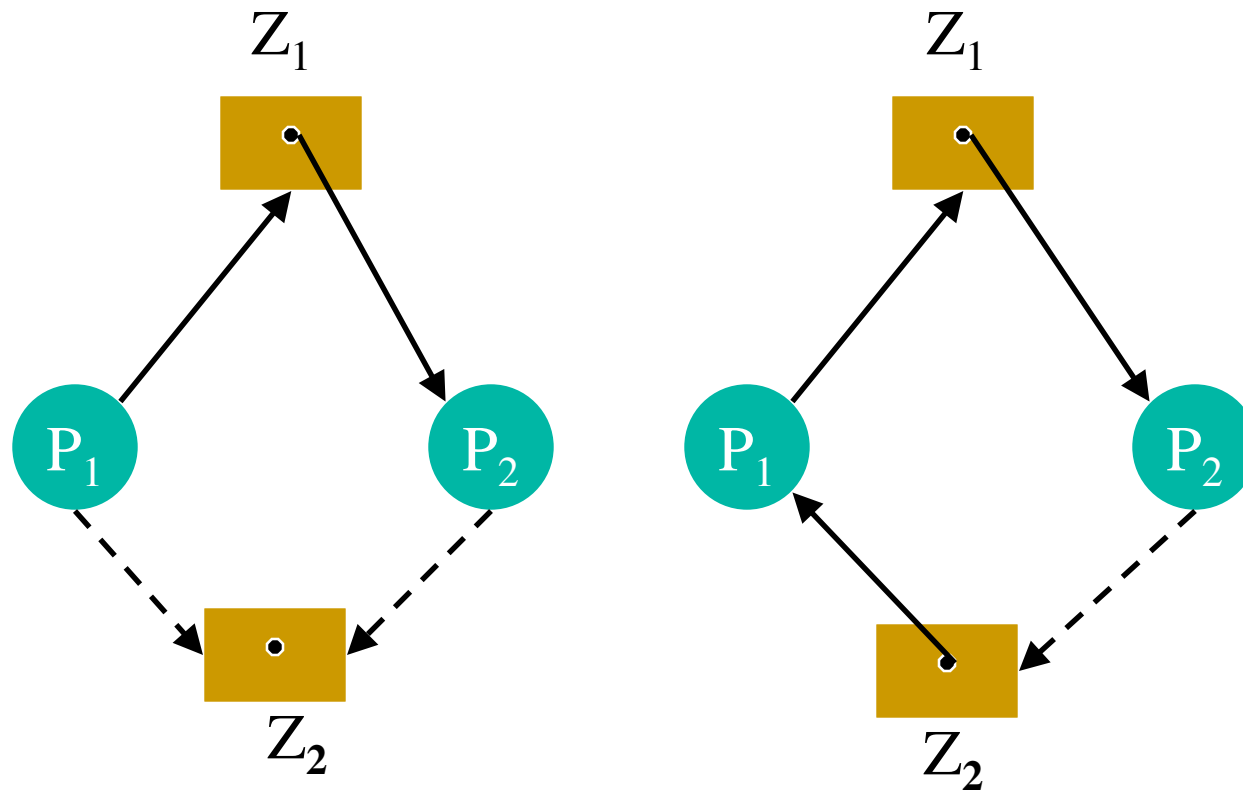
Jeśli porządek przydzielania zasobów procesom jest taki, że nie ma możliwości wystąpienia zakleszczeń, to system jest w **stanie bezpiecznym**

Stan bezpieczny

Jeżeli dla każdego procesu P_i (spośród $P_1..P_n$) jego zapotrzebowanie na zasoby może być zaspokojone przez aktualnie dostępne zasoby, oraz przez zasoby używane przez wszystkie procesy P_j , gdzie $j < i$, to taki ciąg procesów jest **bezpieczny**.

- Jeżeli P_i nie ma danej chwili dostępnych wszystkich zasobów, to poczeka aż procesy P_j skończą działanie i oddadzą swoje zasoby P_i
- Po otrzymaniu wszystkich zasobów proces P_i może dokończyć pracę i zwolnić zasoby
- Wtedy proces P_{i+1} może otrzymać zasoby (jeśli na nie czekał) itd.

Graf przydziału zasobów - analiza stanu bezpiecznego



Stan zagrożenia

Algorytm bankiera

dla wielo-egzemplarzowych zasobów

- Każdy proces wchodzący do systemu musi zadeklarować maksymalną liczbę używanych egzemplarzy każdego zasobu, a liczba ta musi być nie większa od całkowitych zasobów w systemie
- System decyduje, czy przydział tych zasobów pozostawi stan bezpieczny
- Jeśli tak, to przydziela. Jeśli nie - to proces musi poczekać na zwolnienie zasobów przez inne procesy
- Jeśli proces uzyska potrzebne zasoby, to musi je zwrócić w skończonym czasie.

Wykrywanie zakleszczeń

Jeśli nie unikamy zakleszczeń to w systemie muszą istnieć:

a) Algorytmy wykrywania zakleszczenia

b) Algorytmy likwidowania zakleszczenia

ad. a): Tworzy się graf oczekiwania i okresowo wykonuje algorytm poszukiwania cykli w tym grafie

ad. b): - Usunięcie wszystkich zakleszczonych procesów lub
- Usuwanie procesów pojedynczo aż do
wyzeliminowania cyklu zakleszczenia

Kryteria wyboru ofiary:

1. Priorytet
2. Czas wykonywania i pozostały do zakończenia,
3. Liczba i typ zasobów trzymanych przez proces,
4. Liczba procesów które trzeba będzie przerwać,
5. Proces interakcyjny czy wsadowy

Wywłaszczanie

- Wybierając **ofiara** do wywłaszczenia trzeba mieć na uwadze minimalizację kosztów - ile zasobów odzyskamy, a ile pracy procesu stracimy
- **Wycofanie** (o ile to możliwe) procesu do bezpiecznego punktu, od którego może wznowić działanie
- **Głodzenie** to w tym wypadku wywłaszczanie ciągle tego samego procesu. Algorytm powinien więc zapewniać, że proces będzie ofiarą tylko skończoną ilość razy.

Mieszane metody unikania impasu

- zapobieganie
- unikanie
- wykrywanie

Jeżeli podzielimy zasoby na różne klasy, np.

- a) zasoby wewnętrzne, zużywane przez system, np. bloki kontrolne procesów,
 - b) pamięć główna,
 - c) zasoby zadania, np. przydzielone urządzenia, pliki
 - d) wymienny obszar pamięci - obszar w pamięci pomocniczej, przeznaczony na poszczególne zadania użytkowników
- to do każdej z klas można stosować inne metody rozwiązania problemu zakleszczeń.

Mieszane metody unikania impasu

Ad a): poprzez porządkowanie zasobów - nie trzeba dokonywać wyborów między realizowanymi zadaniami

Ad b): ponieważ można wysłać obraz zadania do pamięci pomocniczej, pozwala to na wywłaszczenie procesów z pamięci głównej

Ad c): poprzez taktykę unikania zakleszczeń, bazując na informacjach z kart sterujących zadania

Ad d): można zastosować wstępny przydział pamięci, gdyż maksymalne zapotrzebowanie każdego procesu na pamięć jest znane.

System plików

Definicje:

- Plik jest logiczną jednostką magazynowania informacji w pamięci nieulotnej
- Plik jest nazwanym zbiorem powiązanych ze sobą informacji, zapisanym w pamięci pomocniczej
- Plik jest ciągiem bitów, bajtów, wierszy lub rekordów

Atrybuty pliku:

- nazwa (zgodna z regułami dla danego systemu operacyjnego)
- typ (jeżeli system wymaga rozróżnienia typów)
- położenie (wskaźnik do urządzenia i położenie na tym urządzeniu)
- rozmiar (w bajtach, słowach lub blokach)
- ochrona (prawa dostępu do pliku - np rwx)
- czas, data i identyfikator właściciela (czas utworzenia, modyfikacji, dostępu)



Operacje plikowe

Definicje:

- **Tworzenie pliku**
 - znalezienie miejsca w systemie plików
 - wpis do katalogu
- **Zapisywanie pliku** - podaje się nazwę (identyfikator) pliku i informację do zapisania. Istotne jest miejsce od którego piszemy (wskaźnik położenia)
- **Czytanie pliku** - podaje się nazwę pliku i bufor w pamięci. Można wykorzystać ten sam wskaźnik położenia
- **Zmiana pozycji w pliku** - modyfikacja wskaźnika położenia
- **Usuwanie pliku** - zwalnia się przestrzeń zajmowaną przez plik i likwiduje się wpis katalogowy
- **Skracanie pliku** - likwidowanie części albo całej zawartości pliku bez kasowania jego nazwy i atrybutów



Inne operacje

Definicje:

- **Dopisywanie** - dopisywanie nowych informacji na końcu istniejącego pliku
- **Przemianowanie pliku** - zmiana nazwy pliku, często tą samą komendą wykonuje się **przesuwanie pliku**, czyli zmianę jego położenia - do innego katalogu, na inny dysk
- **Otwieranie pliku** - stosowane w wielu systemach w celu uniknięcia wielokrotnego czytania informacji o pliku - dane z katalogu kopiowane są do tablicy otwartych plików
- **Zamykanie pliku** - kiedy plik przestaje być potrzebny, usuwa się wpis z tablicy otwartych plików
- **Otwieranie i zamykanie plików w systemach wieloużytkownikowych musi uwzględniać równoczesne korzystanie z pliku przez kilka procesów!**

Typy plików

System rozpoznaje typy plików poprzez:

- **Rozszerzenia** - w MSDOS niektóre typy plików określane przez rozszerzenia nazwy (*.com, *.exe...)
- **liczby magiczne** - oraz typowe fragmenty początku pliku - identyfikacja w systemie Unix (komenda **file**, plik **/etc/magic**)
- **atrybut twórcy** (w MAC OS) - czyli nazwę programu, przy pomocy którego utworzono plik

Typy dostępu do plików

- **dostęp sekwencyjny** - informacje w pliku są przetwarzane kolejno, rekord po rekordzie
- **dostęp bezpośredni** - umożliwia czytanie z zapisywanie bloków w dowolnej kolejności. Rekordy muszą być stałej długości. Używany jest tam, gdzie potrzebny jest szybki dostęp do wielkich ilości informacji, np w bazach danych.
- **dostęp indeksowy** (plik indeksowy w pamięci, lub na dysku)



Katalogi

- **jednopoziomowy** - ograniczeniem jest konieczność spełnienia warunku niepowtarzalności nazw
- **dwupoziomowy** - każdy użytkownik ma własny katalog macierzysty, a w nim pliki
- **wielopoziomowe drzewiaste**
- **acykliczne grafy** - do pliku można dojść wieloma drogami

Ochrona plików

Można kontrolować wiele operacji:

- **czytanie** pliku
- **pisanie** do pliku, lub zapisywanie go na nowo
- **wykonywanie** - załadowanie pliku do pamięci i wykonanie go
- **dopisywanie** danych na końcu pliku
- **usuwanie** pliku i zwalnianie obszaru przez niego zajętego
- **opisywanie** - wyprowadzenie nazwy i atrybutów pliku

klasy użytkowników pliku:

- **właściciel** - użytkownik, który utworzył dany plik
- **grupa** użytkowników, którzy wspólnie korzystają z pliku i potrzebują podobnego zakresu dostępu
- **wszyscy inni**

Przydział miejsca na dysku

- **przydział ciągły** - każdy plik zajmuje ciąg kolejnych bloków na dysku.
zalety: minimalna liczba operacji przeszukiwania dysku, łatwość implementacji dostępu sekwencyjnego i bezpośredniego.
wady: trudności ze znalezieniem wolnego miejsca (fragmentacja zewnętrzna)
- **przydział listowy** - istnieje lista powiązanych ze sobą bloków dyskowych, stanowiących dany plik. Bloki te mogą się znajdować w dowolnym miejscu na dysku.
zalety: brak fragmentacji zewnętrznej, nie trzeba deklarować długości pliku (plik może rosnać, dopóki są wolne bloki)
wady: trudność w implementacji dostępu bezpośredniego, zajęcie sporej przestrzeni przez wskaźniki, w przypadku błędu jednego wskaźnika można wejść w obszar innego pliku.

Przydział miejsca na dysku, cd

- **przydział indeksowy** - podobny jak przydział listowy, ale wskaźniki umieszczone w jednym miejscu - w tablicy indeksów
zalety: jak w przydziale listowym,
wady: wskaźniki bloku indeksowego zajmują zazwyczaj więcej miejsca niż wskaźniki przy przydziale listowym

Implementacje dla dużych plików

- **schemat listowy** - jeśli lista bloków jest dłuższa niż blok indeksowy, na ostatniej pozycji w bloku indeksowym podaje się adres bloku kontynuacji,
- **indeks wielopoziomowy** - pozycje bloku indeksowego wskazują na bloki indeksowe poziomu drugiego.
- **schemat mieszany** - pierwsze kilka, kilkanaście pozycji wskazuje bezpośrednio na bloki, a następne 2-3 na indeksy poziomu drugiego w indeksowaniu 2,3,4-poziomowym. Schemat ten zastosowany w systemie UNIX.

Przydział miejsca a wydajność

Przykład złego doboru metody przydziału: przy przydziale listowym i dostępie bezpośrednim, dostęp do bloku n wymaga n ostępów do dysku.

Metody poprawy wydajności

- **deklaracja typu dostępu przy tworzeniu pliku**
 - jeśli dostęp ma być bezpośredni, stosuje się przydział ciągły (wymaga podania wielkości pliku przy tworzeniu)
 - jeśli dostęp ma być sekwencyjny, stosuje się przydział listowy
- System musi mieć zaimplementowane obie metody przydziału
- **typ przydziału zależny od wielkości pliku**
 - dla małych, kilku-blokowych plików przydział ciągły,
 - dla dużych plików przydział np indeksowy
- **Stosowanie klastrów (gron) i różnych wielkościach i**
stosowanie możliwie największych (np 56 kB) dla dużych plików. Uzupełnianie do końca pliku małymi klastrami.

Zarządzanie wolną przestrzenią

- **mapa bitowa**- w wektorze bitowym każdy wolny blok jest reprezentowany przez 1 a zajęty - przez 0. łatwość znalezienia wolnego bloku istnieje dzięki rozkazom procesora pokazującym pozycję pierwszego niezerowego bitu w słowie. Metoda ta może być stosowana dla małych dysków
- **lista powiązana** -w pamięci przechowuje się położenie pierwszego wolnego bloku, a w nim - położenie następnego itd. Metoda mało wydajna - aby przejrzeć listę wolnych bloków, należy wszystkie przeczytać.
- **grupowanie** - w pierwszym wolnym bloku przechowywana jest lista n wolnych bloków. W n-tym wolnym bloku znajduje się lista następnych n bloków itd.
- **zliczanie** - przechowuje się adres pierwszego wolnego bloku i liczbę n następujących po nim wolnych bloków. I tak dla każdej grupy. N jest zazwyczaj > 1

Poprawa wydajności systemów dyskowych

- **pamięć podręczna**- przechowywanie całych ścieżek dysku w pamięci - prawdopodobnie będą z nich w niedługim czasie czytane dane. Wykorzystana do tego celu specjalna pamięć, lub nieużywana pamięć główna
- **wczesne zwalnianie** -usuwanie bloku z bufora natychmiast, gdy pojawia się zamówienia na następny (oszczędza pamięć)
- **czytanie z wyprzedzeniem** - z zamówionym blokiem czyta się kilka następnych, gdyż prawdopodobnie zaraz będą potrzebne
- **RAM-dysk** - wszystkie operacje dyskowe przeprowadza się w pamięci. Zawartość RAM-dysku jest pod kontrolą użytkownika. Wada - zawartość ginie po wyłączeniu zasilania, awarii.

Czynności operatorskie

- **sprawdzanie spójności**- po awarii systemu, czy np po wyłączeniu „z kontaktu”. Program **chkdsk** , **scandisk** (DOS, Windows), **fsck** (UNIX). Zazwyczaj uruchamiają się automatycznie (znacznik „czystości” systemu plików)
- **składowanie i odtwarzanie** - robienie kopii systemu plików na innym nośniku i odtwarzanie po awarii. **kopia zapasowa**, **Norton Ghost** (DOS, Windows), **tar**, **backup**, **restore** (Unix).
Składowanie przyrostowe - kopia całego systemu raz, a potem zapisywanie tylko zmienionych plików
Kopie „wieczyste” - co jakiś czas, taśmy pozostają w archiwum „na zawsze”

System plików w MS-DOS

- **katalog**- zawiera: nazwy plików (8 znaków), rozszerzenie (3 znaki), długość pliku, atrybuty (h s r a), datę utworzenia pliku, wskazanie pozycji FAT
- **FAT (file allocation table)** - tablica, której elementy odpowiadają kolejnym jednostkom alokacji (sektorom, blokom, klasterom). FAT jest umieszczony na początku dysku, w dwóch kopiach.

Przykład użycia: **Z1** zajmuje bloki: 7,8,11,3, **Z2** zajmuje blok 4, a **Z3** jest w blokach: 1,2,5,6,9

Nr: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 15 17 18 19 20
02 05 ff ff 06 09 08 11 ff 00 03 00 00 00 00 00 00 00 00 00

FAT 12 - każda pozycja miała 12 bitów, czyli mogła zawierać wartość 0-4096, co przy klasterach 8 kB dawało 32 MB

Kolejne wersje systemu - FAT 16 (do 2 GB) i FAT 32.

Bariera - czas przeszukiwania tablicy FAT.

System plików w Windows NT

- **tom (volume)**- podstawowa jednostka dyskowa - może to być część dysku, cały dysk lub kilka dysków razem
- **klaster (grono, cluster)** - podstawowa jednostka przydziału, jest to grupa sąsiadujących sektorów. Są znacznie mniejsze niż w systemach z FAT - 4 kB dla dużych dysków. Jako adresy dyskowe używane są **logiczne numery gron**.
- **plik** jest obiektem strukturalnym złożonym z atrybutów. Atrybuty pliku są strumieniami bitów. Jednym z atrybutów są też dane pliku.
- **główna tablica plików (MFT)** - przechowuje opisy plików, zawarte w jednym lub kilku rekordach dla każdego pliku
- **odsyłacz do pliku** - niepowtarzalny identyfikator pliku, składa się z 48-bitowego numeru pliku (pozycja w MFT) i 12-bitowego numeru kolejnego

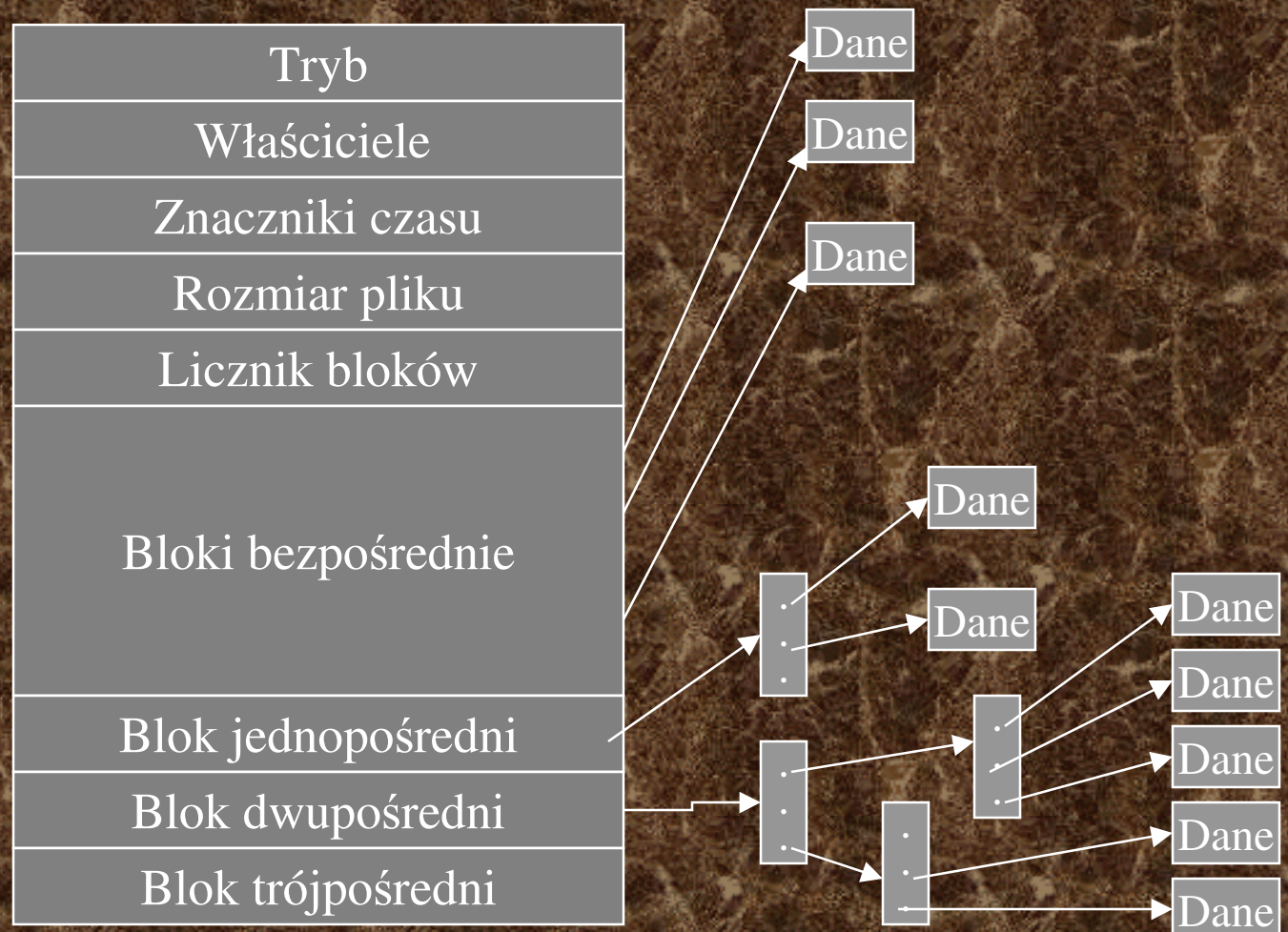
Windows NT - inne pliki systemowe

- **kopia MFT**- kopia pierwszych 16 pozycji MFT - do działań naprawczych,
- **plik dziennika** - zawiera wszystkie zmiany danych w systemie plików
- **plik tomu** - dane tomu, dane o wersji NTFS, który sformatował ten tom, bit informujący o konieczności chkdsk
- **tablica definicji atrybutów** - typy atrybutów i dopuszczalne dla nich operacje
- **katalog główny**
- **plik mapy bitów** - wskazuje zajęte i wolne grona
- **plik rozruchowy** - kod początkowy NT
- **plik złych gron**

Usuwanie skutków awarii jest stosunkowo proste, gdyż operacje dyskowe odbywają się na zasadzie **transakcji**

System plików w UNIX

- **katalog**- zawiera: nazwy plików (do 256 znaków), wskazanie Inode (węzła). Za wyjątkiem znacznika, nie różni się od pliku.
- **Inode** - zawiera pozostałe informacje o pliku:



System plików w UNIX, cd

- **superblok**- zawiera statyczne parametry systemu plików całkowity rozmiar systemu plików, rozmiary pełnych bloków danych i ich fragmentów, dane dotyczące przydziału miejsca
- **grupa cylindrów** - dla zminimalizowania ruchu głowic dysku, cylindry dysku (ta sama ścieżka na wszystkich głowicach) pogrupowano po kilka, zapisując w każdej grupie:
 - superblok,
 - blok cylindra (zawierający mapę wolnych bloków, mapę wolnych i-węzłów, dane statystyczne do strategii przydziału),
 - I-węzły
 - bloki danych (do końca grupy cylindrów)

System wejścia-wyjścia

Trzy rodzaje urządzeń wejścia-wyjścia:

- Urządzenia pamięci (dyski, taśmy)
- Urządzenia przesyłania danych (karty sieciowe, modemy)
- Urządzenia komunikacji z człowiekiem (klawiatury, myszy, monitory)

Różnice między urządzeniami we-wy

- **Urządzenie znakowe** - przesyła bajty (znaki) z osobna jeden za drugim (terminal)
Urządzenie blokowe - przesyła jednorazowo całe bloki (dysk)
- **Dostęp sekwencyjny** - dane przesyłane kolejno w sposób uporządkowany (modem)
Dostęp swobodny - można mieć dostęp do danych w różnych miejscach, niekoniecznie kolejno (CD-ROM)
- **Przesyłanie synchroniczne** - taktowane zegarem (taśma)
przesyłanie asynchroniczne - w nieokreślonych chwilach czasu, sterowane startem i stopem (klawiatura)
- **Urządzenie dzielone** - przez kilka procesów (dysk)
Wyłączne - tylko dla jednego użytkownika (taśma)
- **Szybkość działania** - od B/s do GB/s
- **Kierunek przesyłania** - czytanie, pisanie lub czytanie i pisanie

Sterowanie urządzeniami wejścia-wyjścia

Przekazywanie poleceń z procesora do sterownika:

Sterownik posiada rejestry do pamiętania danych i sygnałów sterujących. Procesor pisze i czyta do rejestrów w sterowniku.

- Procesor posiada specjalne rozkazy do pisania i czytania portów,
- Operacje we-wy odbywają się w pamięci - rejestry są odwzorowywane w przestrzeni adresowej procesora.

W komputerach IBM PC zastosowane są obydwie metody:

- Dla kontrolera grafiki - ekran odwzorowany w pamięci
- Dla portów szeregowych - rejestry we-wy i bufory

Port wejścia-wyjścia



- Stan (czytane przez procesor: zakończenie wykonywania polecenia, dostępność bajtu do czytania, błąd urządzenia)
- Sterowanie (zapisywane przez procesor: rozpoczęcie polecenia, zmiana trybu pracy urządzenia)
- Dane wejściowe (czytane przez procesor dane właściwe)
- Dane wyjściowe (zapisywane przez procesor dane dla urządzenia)

Układ FIFO - „magazyn pośredni” - buforuje dane, których komputer lub urządzenie nie może w danej chwili odebrać.

Odpytywanie

Do uzgadniania pomiędzy procesorem a urządzeniem w prostym schemacie producent-konsument wystarczą dwa bity:

- od strony procesora **bit gotowości polecenia** w rejestrze pleceń - sygnalizujący kompletne polecenie dla urządzenia
- od strony urządzenia **bit zajętości** (w rejestrze stanu), sygnalizujący że urządzenie jest zajęte pracą.

Kolejność działań przy uzgadnianiu:

- Procesor realizuje aktywne czekanie, dopóki bit zajętości jest ustawiony
- Procesor ustawia bit pisania i wpisuje bajt danych do rejestru danych wy.
- Procesor ustawia bit gotowości polecenia
- Sterownik ustawia bit zajętości po zauważeniu bitu gotowości polecenia
- Sterownik czyta rejestr poleceń, rozpoznaje polecenie pisania. Czyta bajt danych z rejestru i wykonuje na urządzeniu operację wejścia-wyjścia
- Sterownik czyści bit gotowości polecenia, bit błędu, a na końcu bit zajętości

I tak dla każdego bajtu danych



Przerwania

Jeśli urządzenie jest rzadko gotowe do działania, odpytywanie staje się nieefektywne (procesor większość czasu poświęca na aktywne czekanie).

Mechanizm przerwań:

- procesor ma końcówkę (nóżkę) badającą stan linii zgłaszania przerwań po wykonaniu każdego rozkazu.
- Jeśli procesor wykryje wystąpienie przerwania, to wykonuje operacje zachowania stanu bieżącego procesu i przechodzi do procedur obsługi przerwań.
- Po wykonaniu niezbędnych operacji procesor wraca do wykonywania przerwanego zadania
- W nowoczesnych architekturach komputerów możliwa zaawansowana obsługa przerwań:
 - opóźnianie obsługi przerwania podczas działań krytycznych,
 - maskowanie przerwań (dwie linie przerwań - maskowalna i niemaskowalna)
 - przerwania wielopoziomowe o różnym priorytecie



Bezpośredni dostęp do pamięci

- Jest używany w celu uniknięcia transmisji bajt-po-bajcie (zwanego programowanym wejściem-wyjściem) dla urządzeń transmitujących wielkie ilości danych (np dysk), co oszczędza wiele cykli procesora
- Wiele procedur związanych z transmisją jest wtedy wykonywana przez specjalizowany procesor - **sterownik bezpośredniego dostępu do pamięci (DMA controller)**
- Przed rozpoczęciem transmisji w trybie DMA, procesor zapisuje w pamięci blok sterujący DMA (wskaźnik do źródła, adres docelowy, liczba bajtów do przesłania), następnie przesyła do sterownika DMA adres tego bloku i przechodzi do wykonywania innych prac.
- Sterownik DMA wykonuje transmisję, przejmując w tym czasie sterownie szyną pamięci. Procesor nie ma wtedy dostępu do pamięci, ale może korzystać z cache i rejestrów.

Wejście-wyjście z blokowaniem

- Blokowanie uwalnia procesor od aktywnego czekania - proces przenoszony jest do kolejki procesów czekających. Po zakończeniu we-wy proces przechodzi do kolejki procesów gotowych.
- Niektóre procesy wymagają wejścia-wyjścia bez blokowania, np proces w którym sygnały z klawiatury lub myszy przeplatają się z przetwarzaniem i wyświetlaniem na ekranie, albo czytanie z dysku z dekompresją danych.
- W aplikacjach wielowątkowych można zablokować pewne wątki, a inne zostawić aktywne.

Podsystem wejścia-wyjścia w jądrze

- **Planowanie wejścia-wyjścia** ma na celu poprawę wydajności systemu, polepszenie wspólnego korzystania z urządzeń przez procesy i zmniejszenie średniego czasu oczekiwania.
- **Buforowanie** - dopasowanie prędkości „producenta” i „konsumenta” danych (podwójne buforowanie), dopasowanie urządzeń operujących na różnych wielkościach bloków danych (np pakiety sieci a bloki na dysku)
- **Przechowywanie podręczne** (caching) - zapamiętanie kopii danych w szybkiej pamięci podręcznej
- **Spooling** - użycie bufora do przechowywania danych przeznaczonych dla urządzenia, które nie dopuszcza przeplatania danych z różnych procesów (np drukarka)
- **Obsługa błędów**
- **Struktury danych jądra** - jądro musi przechowywać informacje o stanie używanych składowych wejścia-wyjścia

Podsystem wejścia-wyjścia w jądrze, cd

Podsystem nadzoruje:

- zarządzanie przestrzenią nazw plików i urządzeń,
- przebieg dostępu do plików i urządzeń,
- poprawność formalną operacji,
- przydzielanie miejsca w systemie plików,
- przydział urządzeń,
- buforowanie, caching oraz spooling,
- planowanie operacji wejścia-wyjścia
- dogłębne nadzory nad urządzeniami, obsługę błędów, czynności naprawcze po awarii.
- konfigurowanie i wprowadzanie w stan początkowy modułu sterującego

Poprawianie wydajności wejścia-wyjścia

Podsystem ma na celu:

- zmniejszać liczbę przełączeń kontekstu,
- zmniejszać liczbę kopiowań danych w pamięci podczas przekazywania od urządzenia do aplikacji,
- zmniejszać częstość przerwania poprzez przesyłanie dużych porcji informacji, przez optymalizację sterowników i stosowanie odpytywania, gdzie to korzystne,
- zwiększać współbieżność poprzez stosowanie sterowników pracujących w trybie DMA,
- realizować elementarne działania za pomocą sprzętu i pozwalać na ich współbieżne wykonywanie w sterownikach,
- równoważyć wydajność procesora, podsystemów pamięci, szyny i operacji wejścia-wyjścia

Elementy od których zależy wydajność dysku

Czytanie lub pisanie wymaga ustawienia głowicy nad określoną ścieżką i na początku konkretnego sektora. Potrzebny na to jest **czas dostępu** - suma czasu przeszukiwania i opóźnienia obrotowego

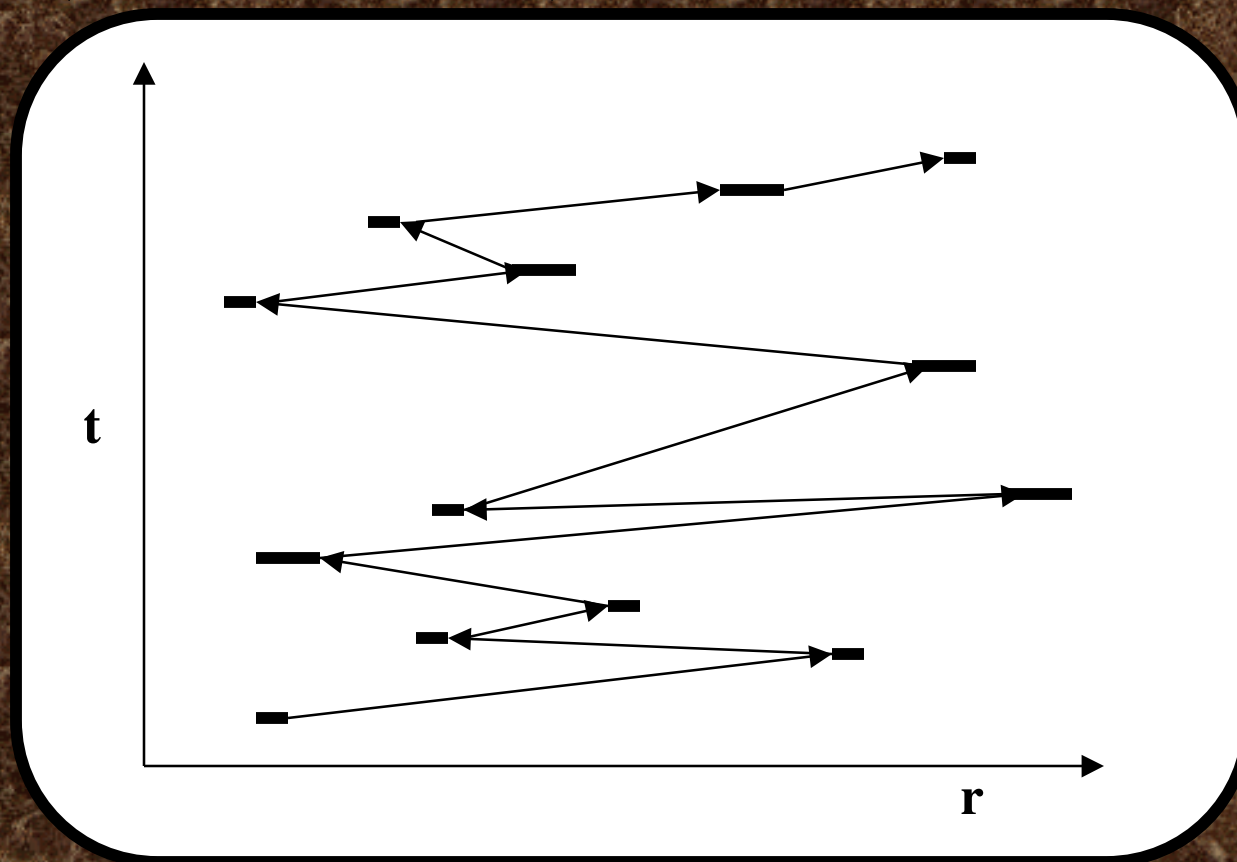
- czas przeszukiwania (seek time) - czas potrzebny na ustawienie głowicy nad ścieżką
- opóźnienie obrotowe (rotational latency) - czas potrzebny na obrót właściwego sektora pod głowicę,

Czas przeszukiwania ma duży wpływ na wydajność.

Możemy go optymalizować, ponieważ system operacyjny zarządza kolejką żądań dysku

Planowanie wejścia-wyjścia dla dysku

- FCFS - (first come, first served)



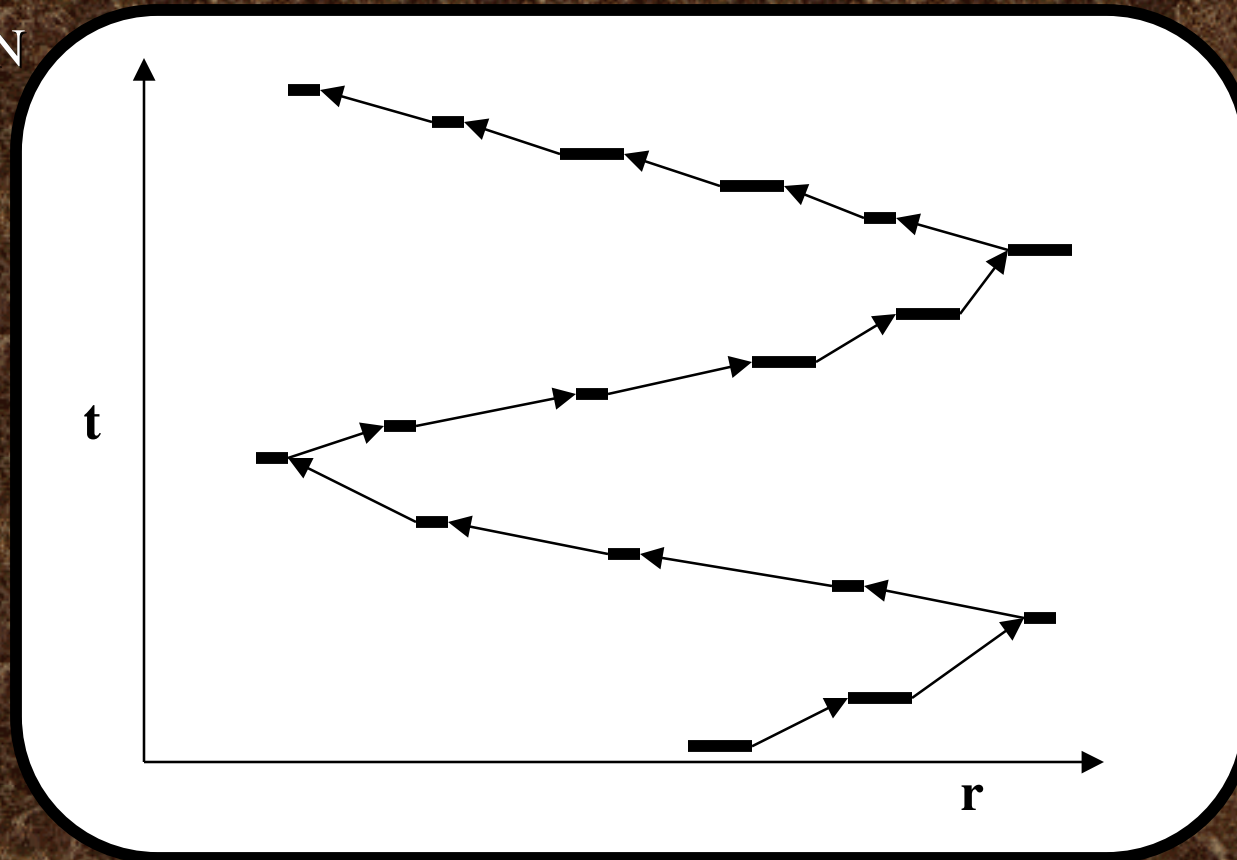
żądania wejścia-wyjścia realizowane w kolejności zgłaszania. Algorytm sprawiedliwy, ale mało wydajny. Przy dużej liczbie zgłoszeń planowanie zbliżone do losowego (najgorszego)

Planowanie wejścia-wyjścia dla dysku, cd

- PRI (*priority*) - krótkie zadania wsadowe i interakcyjne mają wyższy priorytet i pierwszeństwo w dostępie do dysku. Rozwiązanie nieefektywne dla baz danych.
- LIFO (*last in-first out*) - minimalizuje ruch głowicy (zadania korzystają z ograniczonego obszaru na dysku). Możliwość głodzenia procesów
- SSTF (*shortest seek time first*) - najmniejszy ruch głowicy od pozycji bieżącej. Zawsze minimalny czas przeszukiwania. Możliwe głodzenie.

Planowanie wejścia-wyjścia dla dysku, cd

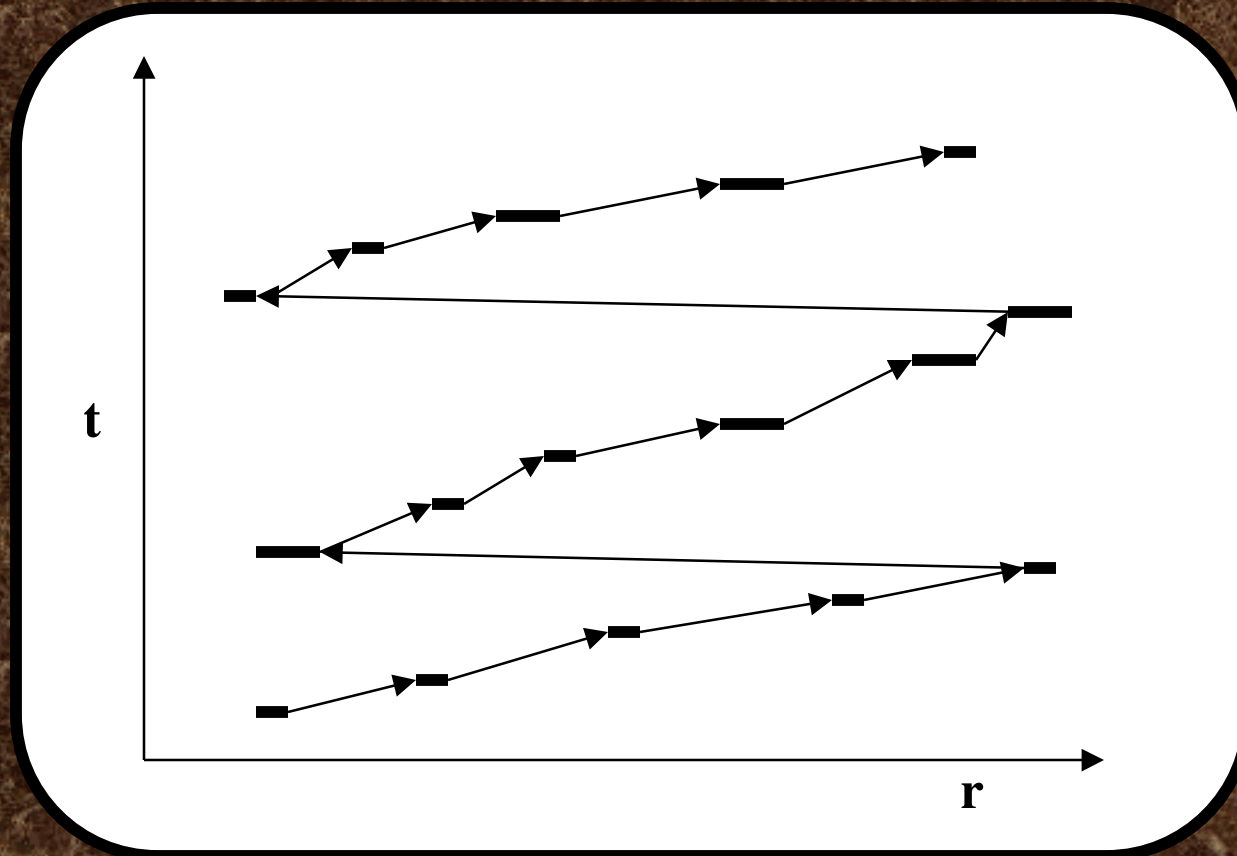
■ SCAN



głowica przesuwa się od jednej krawędzi dysku do drugiej, lub do ostatniego żądania. Potem zmienia się kierunek przesuwania. Działa jak winda - najpierw zamówienia „w górę”, potem „w dół”. Najszybciej po sobie obsługiwane żądania na skrajnych ścieżkach, a najpóźniej - po przeciwnej stronie dysku. Nie ma głodzenia

Planowanie wejścia-wyjścia dla dysku, cd

- C-SCAN (circular scan)



po obsłużeniu ostatniego zamówienia na końcu dysku, głowica wraca na początek i zaczyna od zera.

- N-step-SCAN - podział kolejki na podkolejki o długości N. Kolejne podkolejki obsługiwane metodą SCAN do końca.
- FSCAN - dwie kolejki, jedna realizowana, nowe żądania przychodzą do drugiej

Użytkowy interfejs we-wy drivery urządzeń

- Moduły sterujące - wewnętrznie dostosowane są do konkretnych urządzeń, a zewnętrznie udostępniają pewien standardowy interfejs
- standaryzacja pomaga producentom sprzętu tworzyć sterowniki do własnych urządzeń, widziane przez „obce” systemy operacyjne
- Systemy muszą umożliwiać instalację sterowników do nowego sprzętu

Zegary i czasomierze

Spełniają trzy podstawowe funkcje:

- podawanie bieżącego czasu,
- podawanie upływającego czasu,
- powodowanie wykonania określonej operacji w określonej chwili

czasomierz programowalny - służy do pomiaru upływającego czasu i powodowania wykonania operacji w zadanym czasie

- można go zaprogramować na określony czas, po którym generuje on przerwanie
- jest to też zegar systemowy do taktowania kwantów czasu (dla przydziału procesora)



Urządzenia sieciowe

- Sieciowe wejście-wyjście różni się znacznie od dyskowego, pod względem wydajności i adresowania
- Interfejs gniazda (socket) - aplikacje umożliwiają tworzenie gniazda, połączenie lokalnego gniazda ze zdalnym adresem, nasłuchiwanie, przesyłanie i odbieranie pakietów za pomocą połączenia
- funkcja **wybierz** zarządza gniazdami
- W systemach Windows NT/2000 - interfejs do kontaktowania się z kartą sieciową oraz interfejs do do protokołów sieciowych
- W systemie Unix - półdupleksowe potoki, pełnodupleksowe kolejki FIFO, pełnodupleksowe strumienie, kolejki komunikatów i gniazda

Gniazdo (socket)

Wg. Wikipedii:

Gniazdo - pojęcie abstrakcyjne reprezentujące dwukierunkowy punkt końcowy połączenia. Dwukierunkowość oznacza możliwość wysyłania i przyjmowania danych. Wykorzystywane jest przez aplikacje do komunikowania się przez sieć w ramach komunikacji międzyprocesowej.

Gniazdo posiada trzy główne właściwości:

1. typ gniazda identyfikujący protokół wymiany danych
2. lokalny adres (np. adres IP (protokół), IPX, czy Ethernet)
3. opcjonalny lokalny numer portu identyfikujący proces, który wymienia dane przez gniazdo (jeśli typ gniazda pozwala używać portów)

Gniazdo może posiadać (na czas trwania komunikacji) dwa dodatkowe atrybuty:

1. adres zdalny (np. adres IP (protokół), IPX, czy Ethernet)
2. opcjonalny numer portu identyfikujący zdalny proces (jeśli typ gniazda pozwala używać portów)

Adres IP wyznacza węzeł w sieci,
numer portu określa proces w węźle,
a **typ gniazda** determinuje sposób wymiany danych.

Jeśli gniazdo używa numerów portów to lokalny numer portu może zostać przydzielony automatycznie i nosi wtedy nazwę **efemerycznego** numeru portu. Lokalny numer portu może też zostać wymuszony przez wykonanie **przypisania** (*bind*) gniazdu numeru pożądanego przez twórcę aplikacji.

Wejście-wyjście w różnych systemach operacyjnych

MS-DOS

- Nazwy urządzeń zakończone dwukropkiem, np. A:, C:, PRN:
- Odwzorowane są na określone adresy portów za pomocą tablicy urządzeń.
- System operacyjny może przypisać każdemu urządzeniu dodatkowych funkcji, np. spooling dla drukarki

UNIX

- Nazwy urządzeń są pamiętane w przestrzeni nazw systemu plików (katalog /dev /devices)
- W strukturze katalogowej zamiast numeru I-węzła jest zapisany numer urządzenia w postaci pary <starszy,młodszy>.
- **Starszy** numer urządzenia identyfikuje moduł sterujący, który trzeba wywołać dla obsługi we-wy, a **młodszy** jest przekazywany do modułu sterującego jako indeks do tablicy urządzeń.
- Wpis w tablicy urządzeń zawiera adres portu lub adres sterownika odwzorowany w pamięci

Wejście-wyjście w różnych systemach operacyjnych

UNIX -cd

- **Strumień** - kanał komunikacyjny pomiędzy procesem użytkownika a urządzeniem
- Strumień składa się z:
 - głowy - interfejs z programem użytkownika,
 - zakończenia sterującego - interfejs z urządzeniem
 - modułów przetwarzających - filtrów na komunikatach płynących w strumieniu.
- Każdy ze składników strumienia zawiera co najmniej jedną parę kolejek: we i wy
- Kontrola przepływu w strumieniu realizowana jest za pomocą buforów wejścia-wyjścia
- We-wy na strumieniu jest asynchroniczne
- Sterownik jest zawieszony do chwili pojawienia się danych
- Gdy bufor jest pełny - utrata komunikatów

Wejście-wyjście w różnych systemach operacyjnych

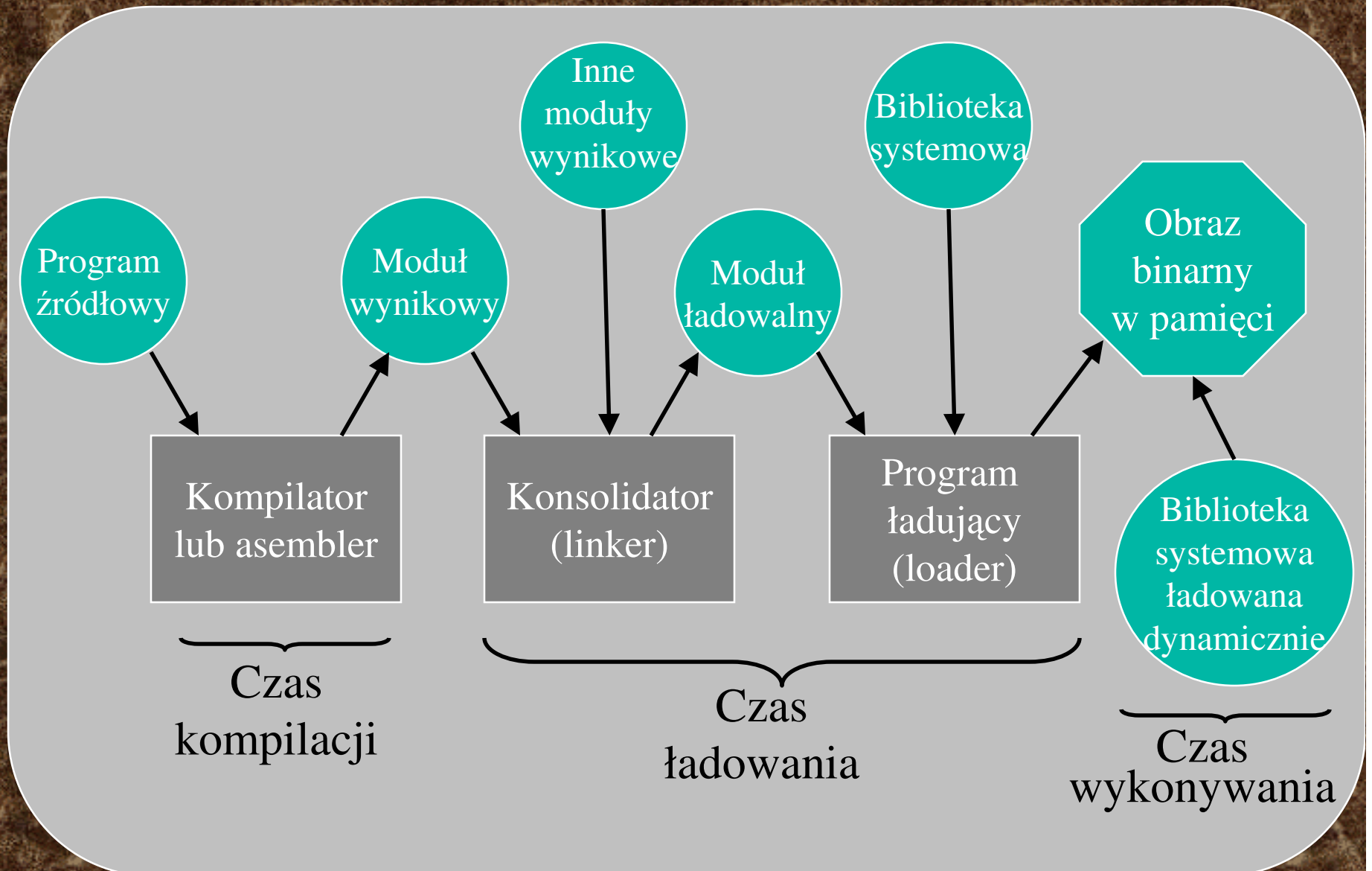
Windows NT/2000

- **Cache manager** - zarządza całym podsystemem wejścia-wyjścia
- Podsystem we-wy traktuje **File System Drivers** tak samo jak **Device Drivers**
- **Network Drivers** - sterowniki sieci
- **Hardware Device Drivers** - dostęp do rejestrów urządzeń wejścia-wyjścia
- Wejście-wyjście może być **asynchroniczne** i **synchroniczne**
- System ma wbudowane mechanizmy do sygnalizowania zakończenia asynchronicznego wejścia-wyjścia

Zarządzanie pamięcią

- Przed wykonaniem program musi być pobrany z dysku i załadowany do pamięci.
- Tam działa jako proces.
- Podczas wykonywania, proces pobiera rozkazy i dane z pamięci.
- Większość systemów pozwala procesowi użytkownika przebywać w dowolnej części pamięci fizycznej.
- System musi uwzględniać to, że program „nie wie” pod jakim adresem pamięci będzie umieszczony. System musi zawierać mechanizmy „tłumaczące” adresy w programie na rzeczywiste adresy w pamięci fizycznej.
- System musi też gospodarować wolną pamięcią, racjonalnie ładując kolejne programy w wolne miejsca pamięci

Od programu źródłowego do procesu



Powiązanie programu z adresami w pamięci

może być dokonane w czasie każdej z trzech faz

- **Czas kompilacji** - jeśli podczas kompilacji wiadomo pod jakim adresem pamięci program będzie przebywał, to tworzy się **kod bezwzględny**, np programy .com w DOS-ie. Aby zmienić położenie takiego programu w pamięci, trzeba go powtórnie przekompilować. Jeśli kod programu ma być **przemieszczalny**, to adresy muszą być określone w sposób względny, np w odległościach od początku modułu.
- **Czas ładowania** - jeśli podczas kompilacji nie określono rzeczywistych adresów pamięci, to program ładujący wylicza je na podstawie adresów względnych.
- **Czas wykonania** - jeśli proces może ulegać przemieszczeniu w pamięci podczas wykonywania, to muszą istnieć mechanizmy wyliczania w dowolnym momencie rzeczywistych adresów.

Dołączanie dynamiczne

- **Ładowanie dynamiczne** - podprogram nie jest wprowadzany do pamięci dopóty, dopóki nie zostanie wywołany. Do pamięci wprowadza się jedynie program główny, a potem sukcesywnie potrzebne moduły. Dzięki temu oszczędza się miejsce w pamięci nie ładując niepotrzebnie wielkich modułów, np. obsługi błędów.
- **Konsolidacja dynamiczna** - bez tej właściwości wszystkie programy muszą mieć dołączone kopie bibliotek, w tym systemowych. Powoduje to marnotrawstwo dysku i pamięci. Biblioteki dynamicznie linkowane są sprowadzane do pamięci w momencie ich wywołania i tam mogą służyć nawet kilku programom. Dodatkowa zaleta - aktualizacja biblioteki nie wymaga zazwyczaj przekompilowania programu.

Nakładki

- **Nakładki** są potrzebne jeśli proces jest większy niż ilość dostępnej pamięci.

- Przykład - dwuprzebiegowy asembler

mamy do dyspozycji 150 kB pamięci, a poszczególne elementy zadania mają wielkość:

Kod przebiegu 1: 70 kB

Kod przebiegu 2: 80 kB

Tablica symboli: 20 kB

wspólne podprogramy: 30 kB

Razem: 200 kB



Nakładki

Robimy dwie nakładki:

1. Tablica symboli, wspólne podprogramy, kod przebiegu 1 oraz moduł obsługi nakładek (10 kB) - razem 130 kB
2. Tablica symboli, wspólne podprogramy, kod przebiegu 2 oraz moduł obsługi nakładek - razem 140 kB

Obydwie nakładki są poniżej 150 kB.

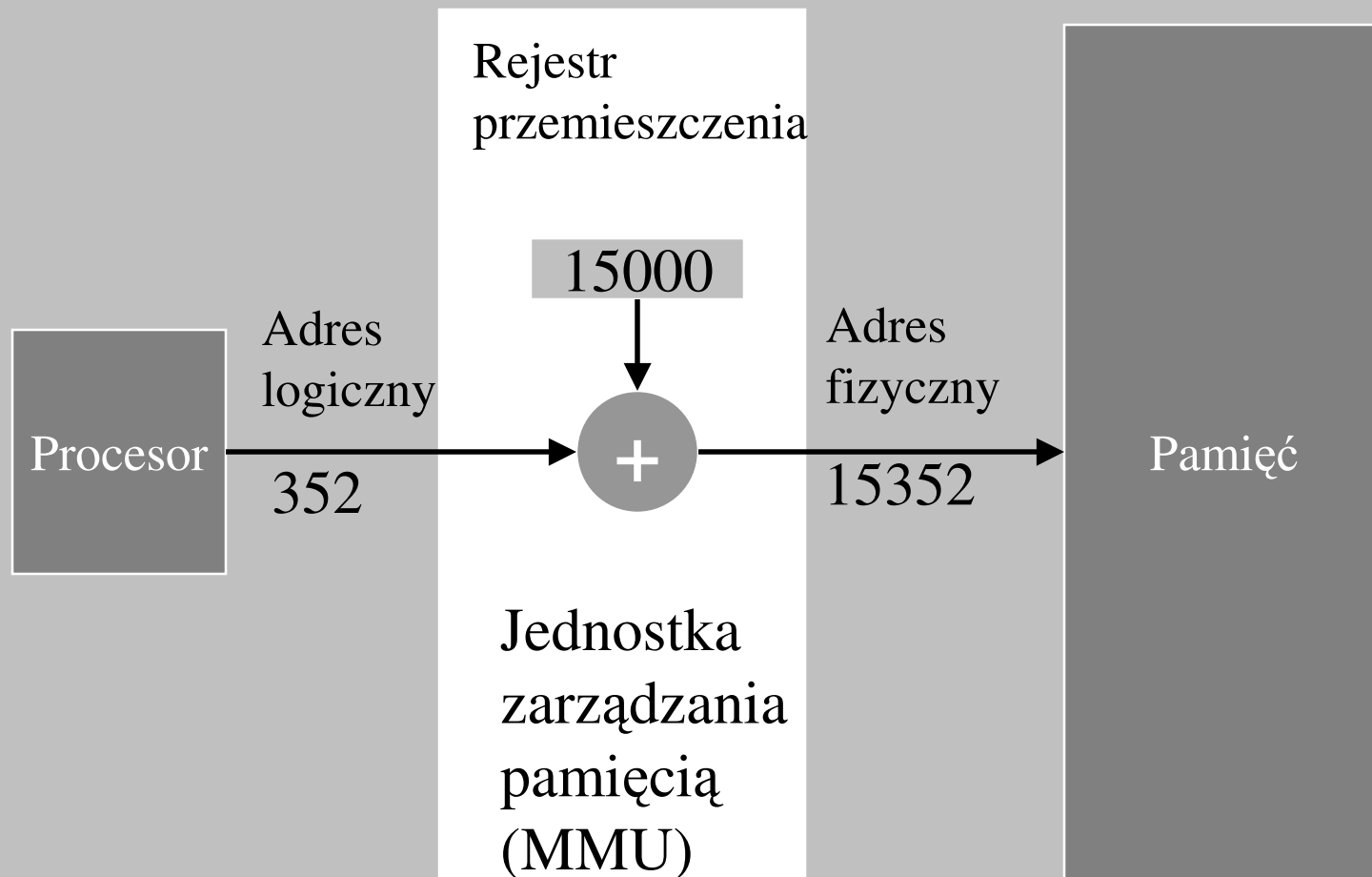
Kody nakładek 1 i 2 są przechowywane na dysku w postaci obrazów bezwzględnych pamięci i są czytane przez moduł obsługi nakładek, w zależności od potrzeb.

Program nakładkowy nie potrzebuje specjalnego wsparcia ze strony systemu operacyjnego, ale wymaga starannego programowania, ze znajomością rzeczy.

Logiczna i fizyczna przestrzeń adresowa

- **Adres fizyczny** - jest to adres oglądany przez jednostkę pamięci (umieszczony w jej rejestrze adresowym)
- **Adres logiczny** - jest to adres wygenerowany przez procesor
- Odwzorowanie adresów logicznych na fizyczne realizowane jest sprzętowo przez jednostkę zarządzania pamięcią (MMU)
- W MMU przeliczanie adresu odbywa się najczęściej poprzez dodanie do adresu z procesora wartości **rejestru przemieszczenia**.
- Program użytkownika działa wyłącznie na logicznych adresach

Wyliczanie adresu fizycznego



Wymiana

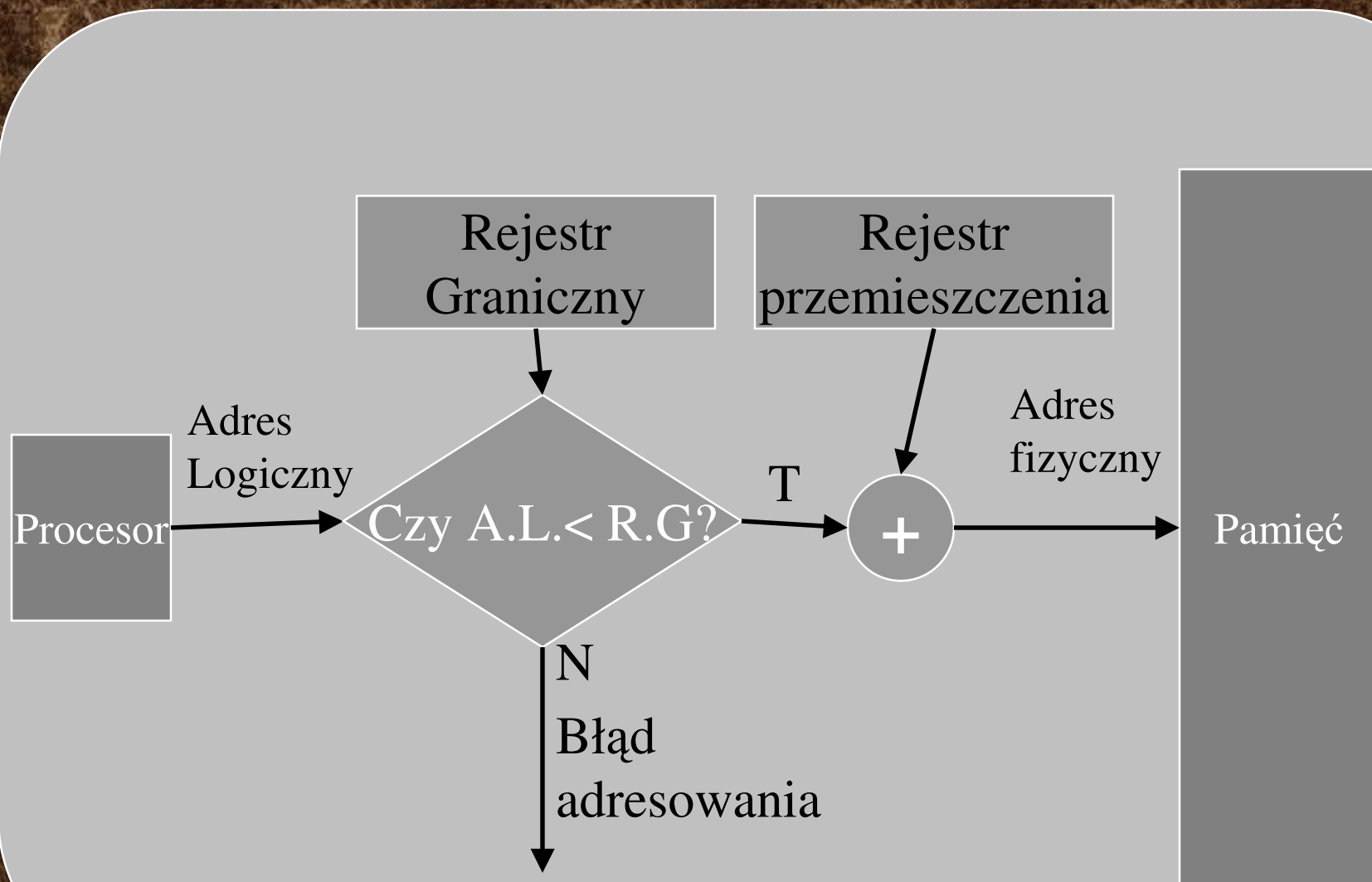
- Jest to tymczasowe odesłanie procesu do pamięci pomocniczej (na dysk) i przepisanie z powrotem w celu kontynuowania działania.
- Zazwyczaj po wymianie proces powraca na to samo miejsce w pamięci, chyba że są zapewnione mechanizmy przeliczania adresów.
- Warunkiem szybkiej wymiany jest używanie dysku o krótkim czasie dostępu i o pojemności zapewniającej pomieszczenie obrazów pamięci wszystkich użytkowników.
- W planowaniu priorytetowym stosuje się czasem wymianę poprzez **wytaczanie** procesu, gdy nadchodzi proces o wyższym priorytecie i **wtaczanie** z powrotem do pamięci, gdy procesy wysoko-priorytetowy skończył działanie

Przydział ciągły

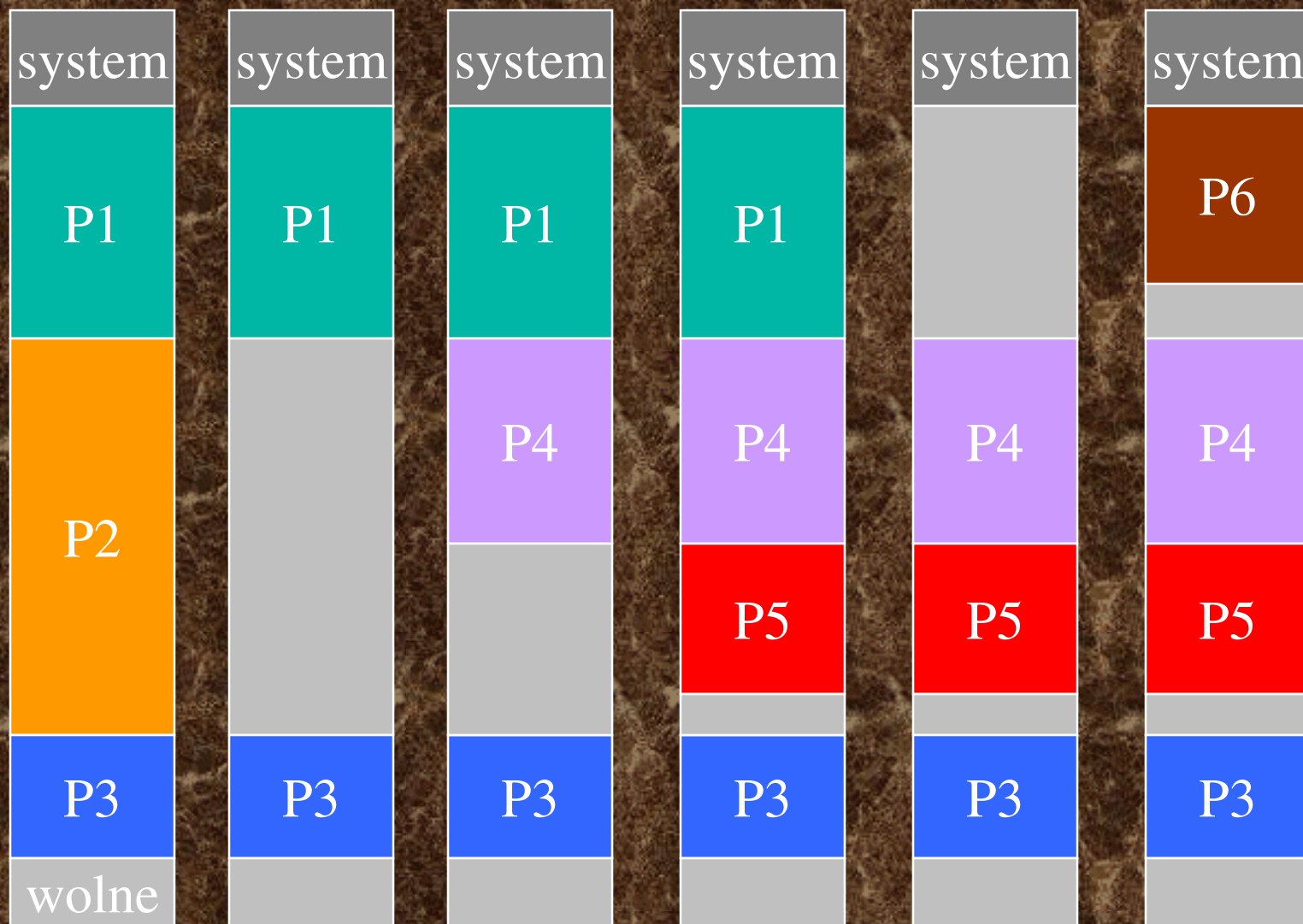
Pamięć operacyjna zajęta jest przez:

- System operacyjny - umieszczony zazwyczaj w tej części pamięci, gdzie wektor przerwań - jest to najczęściej **pamięć dolna**
- Procesy użytkownika, umieszczane zazwyczaj w **pamięci górnej**
- W celu ochrony obszaru pamięci zajętego przez system, a także procesów użytkownika przed wzajemną ingerencją, wykorzystuje się **rejestr przemieszczenia** i **rejestr graniczny**
- **rejestr przemieszczenia** - wartość najmniejszego adresu fizycznego dla danego procesu (offset)
- **rejestr graniczny** - maksymalny adres logiczny procesu

Kontrola zakresu adresów



Przydział pamięci przy wielu procesach



Kryteria wyboru wolnego obszaru pamięci

Dziura - ciągły obszar niezajętej pamięci

- **Pierwsze dopasowanie** - system przydziela pierwszą dziurę o wystarczającej wielkości. Szukanie rozpoczyna się od początku wykazu dziur lub od miejsca w którym zakończono ostatnie szukanie.
- **Najlepsze dopasowanie** - przydziela się najmniejszą z dostatecznie dużych dziur (najmniejsza pozostałość po przydziale)
- **Najgorsze dopasowanie** - przydziela się największą dziurę.
- Czasami duża pozostałość po takim przydziale jest bardziej przydatna niż małe fragmenty po najlepszym dopasowaniu. Ciekawe i oryginalne podejście do zagadnienia, ale praktyka wykazała, że dwie pozostałe metody są lepsze pod względem czasu działania i wykorzystania pamięci.

Fragmentacja

Dziura - ciągły obszar niezajętej pamięci

- **Fragmentacja zewnętrzna**- suma wolnych obszarów pamięci wystarcza na spełnienie zamówienia, ale nie tworzą one spójnego obszaru
- **Fragmentacja wewnętrzna**- jeśli po przydzieleniu pamięci do procesu pozostałby wolny obszar wielkości kilku bajtów, to przydziela się go też do procesu, ale stanowi on „nieużytek”- nie będzie wykorzystany (ale zmniejszy się tablica „dziur”)
- Fragmentację zewnętrzną można zmniejszyć poprzez takie upakowanie procesów, aby cała wolna pamięć znalazła się w jednym dużym bloku.

Jest to możliwe tylko wtedy, gdy ustalanie adresów jest wykonywane dynamicznie podczas działania procesu.

Przetaskowanie procesów nie można robić podczas operacji we/wy.



Stronicownie

pomaga w racjonalnym wykorzystaniu wolnych miejsc w pamięci

- Pamięć fizyczną dzieli się na bloki o stałej długości (**ramki**) o długości 2^n (512 B do 16 MB)
- Pamięć logiczna podzielona jest na **strony** o tym samym rozmiarze
- Wolną pamięć obrazuje lista wolnych ramek
- Proces o wielkości N stron jest ładowany w N ramek (niekoniecznie kolejnych)
- Tablica stron odwzorowuje adresy logiczne na fizyczne
- Eliminuje się fragmentację zewnętrzną, ale występuje fragmentacja wewnętrzna (zaokrąglenie w górę wielkości procesu do wielokrotności rozmiaru ramki)



Stronicownie

Każdy adres wygenerowany przez procesor dzieli się na dwie części: numer strony i odległość na stronie. Numer jest używany jako indeks w tablicy stron, która zawiera adresy bazowe wszystkich stron w pamięci operacyjnej.

Łącząc adres bazowy z odległością na stronie uzyskuje się fizyczny adres w pamięci.

Jeżeli rozmiar strony jest potęgą 2 oraz:

- rozmiar przestrzeni adresowej wynosi 2^m
- rozmiar strony wynosi 2^n ,
- to $m-n$ bardziej znaczących bitów adresu logicznego wskazuje nr strony ($=2^{(m-n)}$),
- n mniej znaczących bitów wskazuje odległość na stronie

Tablica stron

Jest przechowywana w pamięci operacyjnej. Jej położenie wskazuje **rejestr bazowy tablicy stron**.

Rozmiar tablicy stron jest przechowywany w **rejestrze długości tablicy stron**. Określa on na największy dopuszczalny adres.

Przy korzystaniu z tablicy stron, dostęp do pamięci wymaga dwukrotnego dostępu do pamięci -dlaczego?

W celu przyspieszenia dostępu do pamięci stosuje się rozwiązanie sprzętowe - małą, szybką pamięć podręczną zwaną **rejestrami asocjacyjnymi** lub **buforami translacji adresów stron**. Bufory te zawierają 8 do 2048 pozycji.

Jeśli dany numer strony nie znajduje się w buforach, to przeszukiwana jest cała tablica stron. Przy dobrze skonstruowanym algorytmie, w buforach translacji znajduje się 80 do 98 % potrzebnych numerów stron.

Efektywny czas dostępu

Jest to średni czas dostępu do każdego adresu pamięci.

Przykład:

Przeglądnięcie rejestrów asocjacyjnych trwa 20ns

Dostęp do pamięci trwa 100 ns

Współczynnik trafień - procent stron znalezionych w rejestrach asocjacyjnych.

Dostęp do stron „trafionych” = $20 + 100 = 120$ ns

Dostęp do stron nie występujących w rejestrach = $20 + 100 + 100 = 220$ ns

Dla współczynnika trafień = 80%:

E.C.D. = $0,8 * 120 + 0,2 * 220 = 140$ ns

Dla współczynnika trafień = 98%:

E.C.D. = $0.98 * 120 + 0.02 * 220 = 122$ ns

Ochrona pamięci

Bity ochrony - przypisane do każdej ramki.

Można w ten sposób zaznaczyć strony tylko do czytania, do czytania i pisania i do wykonywania.

Bit poprawności - jest ustawiony, jeśli dana strona jest w przestrzeni adresowej procesu (strona jest „legalna” dla procesu).

Jeśli strona jest poza przestrzenią adresową procesu, ma ustawiony znacznik „nielegalna”

Stronicowanie wielopoziomowe

Nowoczesne systemy zezwalają na stosowanie bardzo wielkich przestrzeni adresów logicznych (2^{32} do 2^{64})

W takim przypadku tablica stron może zawierać nawet milion wpisów. Jeśli każdy wpis to 4 B, rozmiar tablicy wyniesie 4 MB, na każdy proces. Tablice takie mogą być większe niż same procesy!

Celowy jest więc podział na mniejsze tablice.

Przy 32-bitowej przestrzeni adresowej:

20-bitowy adres strony i 12-bitową odległość na stronie można zastąpić przez:

10-bitowy adres strony, 10-bitową odległość na tej zewnętrznej stronie i 12-bitową odległość wewnętrzną

Stronicowanie wielopoziomowe

Każdy poziom jest zapisywany jako oddzielna tablica, więc przekształcenie adresu logicznego w fizyczny może wymagać czterech dostępów do pamięci.

czas dostępu wynosi wtedy np 520 ns.

Ale zastosowanie pamięci podręcznej bardzo go skraca.

Dla współczynnika trafień wynoszącego 98%:

Efektywny czas dostępu= $0,98 \cdot 120 + 0,02 \cdot 520 = 128$ ns.

Odwrócona tablica stron

Odwrócona tablica stron ma po jednej pozycji dla każdej ramki w pamięci fizycznej

Każda pozycja zawiera numer procesu posiadającego ramkę oraz adres wirtualny strony przechowywanej w ramce rzeczywistej pamięci.

W systemie istnieje tylko jedna tablica stron.

Ogranicza to zajętość pamięci, ale zwiększa czas przeszukiwania (trzeba przeszukać całą tablicę)

Stosowanie tablic haszowania ogranicza przeszukiwanie do co najwyżej kilku wpisów.



Strony dzielone

20 użytkowników korzysta równocześnie z edytora tekstu. W pamięci musi się znaleźć 20 bloków danych i 20 kopii kodu edytora.

- Jeśli kod programu nie modyfikuje sam siebie w czasie działania (jest wznawialny) to można zastosować mechanizm stron dzielonych.
- Wznawialny kod programu jest widziany przez wszystkie procesy pod tą samą lokacją.
- Każdy proces dysponuje więc własnym obszarem danych i jednym wspólnym kodem programu
- Mechanizm ten może być też stosowany przy innych intensywnie używanych programach (kompilatory, systemy okien, bazy danych)

W systemach z odwróconą tablicą stron mechanizm ten napotyka na trudności - dlaczego?.

Segmentacja

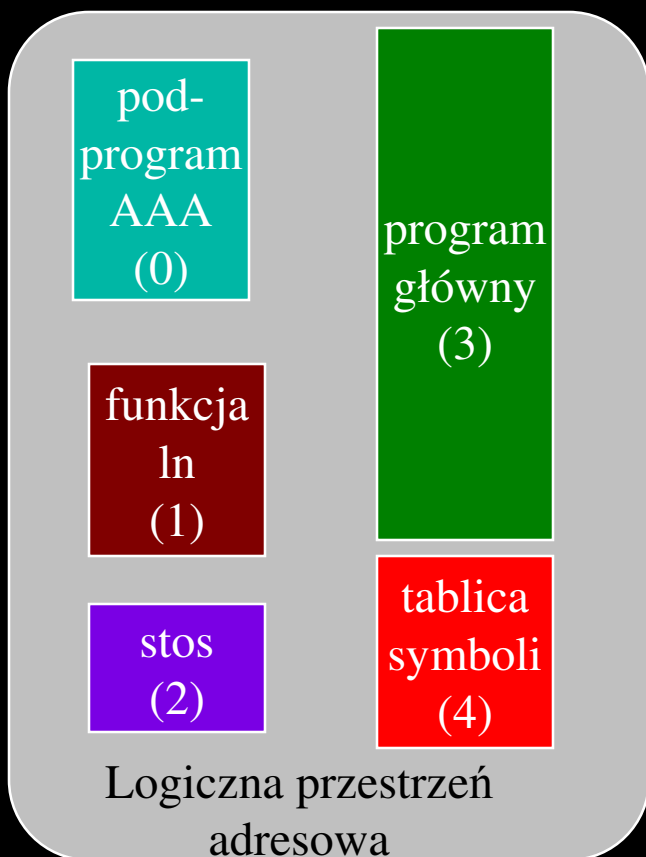
- Jest to mechanizm naśladujący postrzeganie pamięci tak jak użytkownik
- Przestrzeń adresów logicznych jest zbiorem segmentów. Każdy segment ma nazwę i długość. Użytkownik określa więc każdy adres poprzez nazwę segmentu i odległość.
- Kompilator automatycznie tworzy segmenty tworzące program wynikowy. Najczęściej oddzielnymi segmentami są:
 - program główny,
 - procedury,
 - funkcje,
 - zmienne lokalne,
 - zmienne globalne,
 - stos wywołań procedur (parametry i adresy powrotu)
 - bloki wspólne (common)...



Segmentacja

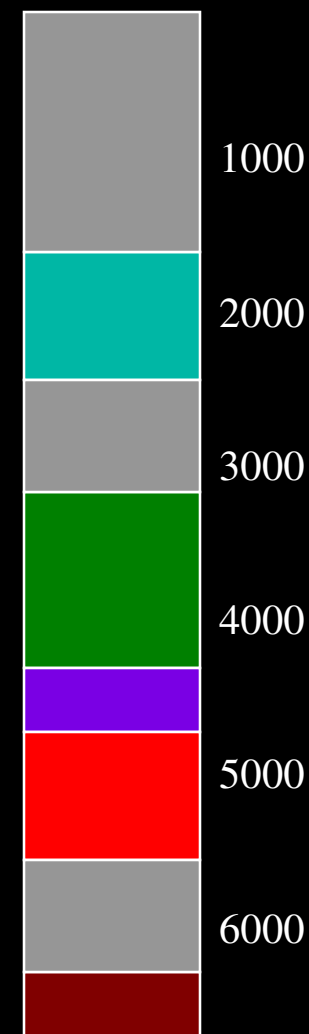
- Adres logiczny składa się z dwóch części:
numer segmentu, odległość w segmencie
- Tablica segmentów zawiera pary danych:
 - baza (fizyczny adres początku segmentu w pamięci)
 - granica (długość segmentu)
- Rejestr bazowy tablicy segmentów - adres tablicy segmentów w pamięci
- Ponieważ programy mogą mieć bardzo różniącą się liczbę segmentów, stosuje się też rejestr długości tablicy segmentów, który służy do sprawdzenia czy podany numer segmentu jest poprawny ($< RDTs$)

Segmentacja - przykład



	Baza	Granica
0	1400	1000
1	6300	400
2	4300	400
3	3200	1100
4	4700	1000

**Tablica
segmentów**



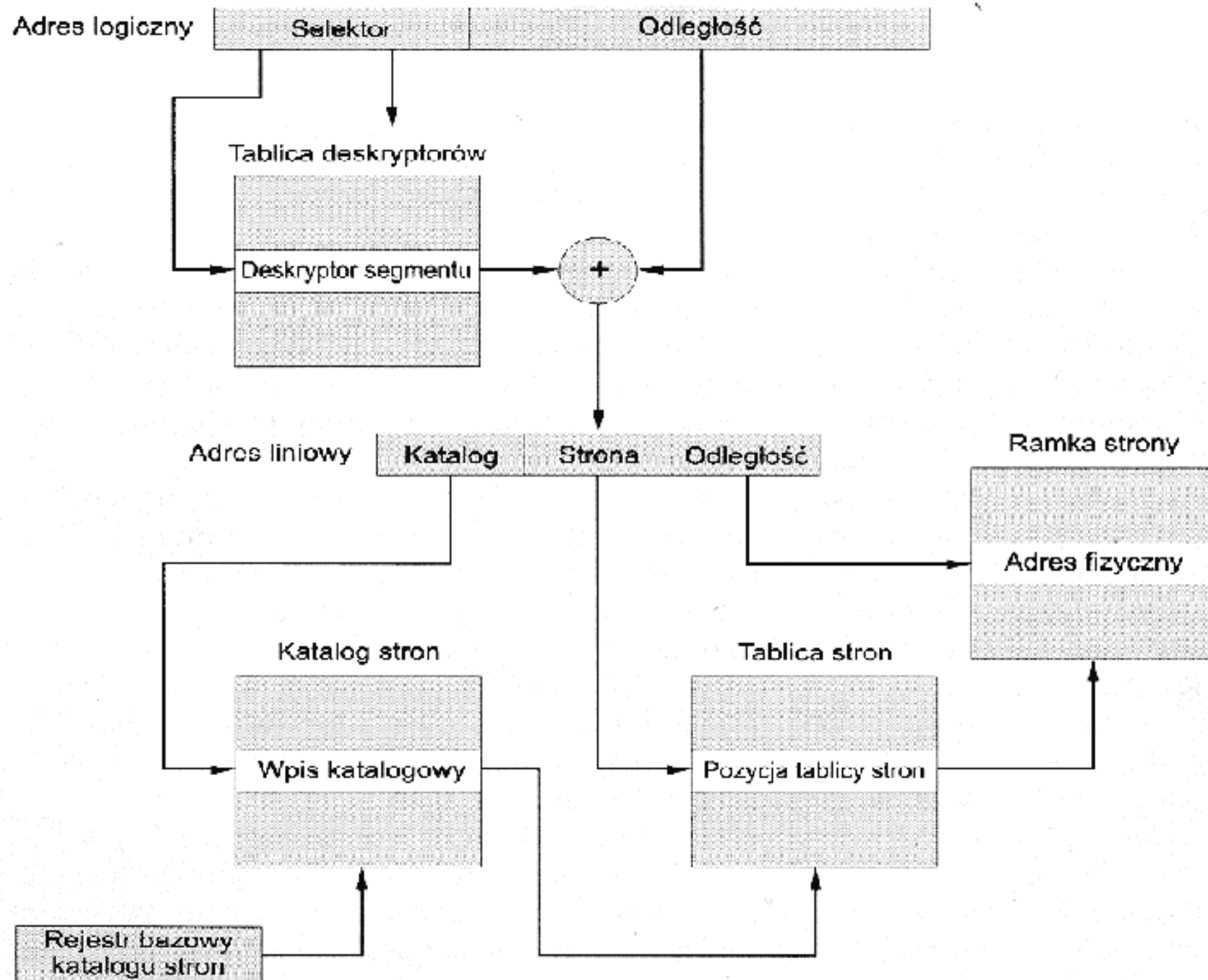
Pamięć fizyczna 7000

Segmenty dzielone

- Segmentacja ułatwia wspólne użytkowanie kodu programu, podprogramów lub niektórych danych
- W tablicach segmentów wszystkich współużytkowujących procesów obszary (segmenty) współdzielone mają takie same wpisy bazy i granicy (wskazują więc na ten sam obszar pamięci)
- We wszystkich procesach segmenty dzielone muszą mieć ten sam numer - dlaczego?
- Alokacja segmentów w pamięci odbywa się metodą pierwszego lub najlepszego dopasowania (segmenty mają różne długości)
- Ponieważ segmentacja jest algorytmem dynamicznego przemieszczania, można przesuwąć segmenty w celu lepszego upakowania

Segmentacja ze stronicowaniem

- W systemie MULTICS - pozycja w tablicy segmentów nie wskazuje adresu bazowego, ale adres tablicy stron dla tego segmentu
- Systemy oparte na Motoroli 68000 - prosta przestrzeń adresowa
- Systemy oparte na Intel ≥ 80386 - segmentacja ze stronicowaniem z dwupoziomowym schematem stronicowania
 - maksymalna liczba segmentów w procesie: 16K
 - każdy segment mniejszy niż 4 GB
 - rozmiar strony 4 kB
 - przestrzeń adresowa ma dwie strefy po co najwyżej 8K segmentów:
 - a) prywatne segmenty procesów przechowywane są w **tablicy lokalnych deskryptorów**,
 - b) wspólne segmenty procesów przechowywane są w **globalnej tablicy deskryptorów**



Segmentacja ze stronicowaniem (Intel)

- **selektor** jest 16-bitową liczbą:
 - 13 b - numer segmentu
 - 1 b - czy segment jest w lokalnej czy globalnej tablicy deskryptorów,
 - 2 b - ochrona
- każdy adres logiczny jest parą: selektor, odległość
- procesor ma 6 rejestrów segmentów (do adresowania 6 segmentów) oraz 6 rejestrów mikroprogramowych do przechowywania pozycji z lokalnej i globalnej tablicy deskryptorów,
- sprawdzanie adresu: rejestr wyboru wskazuje na pozycję w lokalnej lub globalnej tablicy; na podstawie adresu początku segmentu i jego długości tworzy się adres liniowy (sprawdzenie poprawności); jeśli poprawny, to do bazy dodaje się odległość.

Stronicowanie dwupoziomowe (Intel)

- liniowy adres jest 32-bitową liczbą:
 - 20 najstarszych bitów - numer strony
 - 10 b - wskaźnik do katalogu stron,
 - 10 b - wskaźnik do tablicy stron,
 - 12 najmłodszych bitów - odległość na stronie

Pamięć wirtualna

Pytanie: Czy proces rezerwuje pamięć i gospodaruje nią w sposób oszczędny?

- Procesy często zawierają ogromne fragmenty kodu obsługujące sytuacje wyjątkowe
- Zadeklarowane tablice lub rozmiary list mają zwykle nadmiar przydzielonej pamięci,
- Niektóre możliwości programu są niezwykle rzadko wykorzystywane.

A gdyby tak w pamięci przebywała tylko ta część programu, która jest aktualnie wykonywana?



Zalety:

- Program nie jest ograniczony wielkością pamięci fizycznej - można pisać ogromne programy bez specjalnych „sztuczek” programistycznych
- każdy program zajmuje w pamięci mniej miejsca niż program „kompletny”. Można więc w tym samym czasie wykonywać więcej zadań, polepszając wykorzystanie procesora
- maleje liczba operacji wejścia-wyjścia koniecznych do załadowania programów do pamięci oraz do realizacji wymiany - programy powinny więc wykonywać się szybciej
- nie ma konieczności robienia nakładek przy małej pamięci operacyjnej

Pamięć wirtualna

Pamięć wirtualna odseparowuje pamięć logiczną od jej fizycznej realizacji. Można ją zaimplementować jako:

- a) stronicowanie na żądanie
- b) segmentację na żądanie

Procesy przebywają w pamięci pomocniczej (na dysku). Dla wykonania sprowadza się proces do pamięci, ale nie cały, tylko te strony, które są potrzebne (leniwa wymiana). Typowanie (zgadywanie) potrzebnych stron odbywa się podczas poprzedniego pobytu procesu w pamięci.

Jeśli proces odwołuje się do strony, której nie ma w pamięci, to:

- system sprawdza, czy odwołanie do pamięci było dozwolone czy nie (bit poprawności)
- Jeśli było poprawne, sprowadza stronę do pamięci, modyfikuje tablicę stron i wznowia działanie procesu

Stronicowanie na żądanie a wymiana

- Proces jest ciągiem stron
- **Wymiana** dotyczy całego procesu (wszystkich stron)
- **Procedura stronicująca** dotyczy poszczególnych stron procesu
- **Procedura stronicująca** zgaduje, jakie strony będą w użyciu i tylko je ładuje do pamięci. Nigdy nie dokonuje się wymiana całego procesu

Czy stronicowanie n.ż. bardzo spowalnia komputer?

p - prawdopodobieństwo braku strony

ECD - efektywny czas dostępu:

$$ECD = (1-p) * cd + p * cos$$

cd - czas dostępu do pamięci (10 do 200 ns)

cos - czas obsługi strony:

- obsługa przerwania „brak strony” ($1 \div 100 \mu s$)
- czytanie strony (duuużo μs)
- wznowienie procesu ($1 \div 100 \mu s$)

Czynności obsługi braku strony:

- przejście do systemu operacyjnego,
- przechowanie kontekstu,

- określenie, że przerwanie to „brak strony”
- Sprawdzenie poprawności odwołania do strony i jej położenia na dysku,
- czytanie z dysku do niezajętej ramki:
 - opóźnienie (~8 ms)
 - szukanie sektora (~15 ms)
 - transfer danych (~1 ms)
- zmiana przydziału procesora do innego procesu
- przerwanie od dysku po skończonym transferze
- przełączenie kontekstu
- skorygowanie tablicy stron
- czekanie na przydział procesora i wznowienie procesu

Razem: ok. 25 ms

$$25 \text{ ms} = 25\,000 \mu\text{s} = 25\,000\,000 \text{ ns}$$

Efektywny czas dostępu:

$$ECD = (1-p) * cd + p * cos$$

$$ECD = (1-p) * 100 + p * 25\,000\,000 = 100 - 100 * p + 25\,000\,000 * p = \\ 100 + 24\,000\,900 * p \text{ [ns]}$$

Efektywny czas dostępu jest więc proporcjonalny do prawdopodobieństwa nie znalezienia strony w pamięci.

Przy prawdopodobieństwie $p = 1\%$ ECD wyniesie: **240 109 ns**
(czyli 2 400 razy więcej niż dostęp bezpośredni do pamięci)

Zastępowanie stron

Założenie, że tylko część stron każdego procesu jest potrzebna w pamięci może doprowadzić do **nadprzydziału** (nadmiar procesów w pamięci i absolutny brak wolnych ramek).

Aby nie blokować procesu potrzebującego kolejnej ramki, stosuje się **zastępowanie stron**.

- uruchamia się algorytm typowania ramki-ofiary
- stronę-ofiarę **zapisuje** się na dysku,
- aktualizuje się tablicę wolnych ramek,
- **wczytuje** się potrzebną stronę do zwolnionej ramki
- aktualizuje się tablicę stron
- wznowia się działanie procesu

Dwukrotne korzystanie z dysku bardzo wydłuża czas obsługi braku strony!



Zastępowanie stron, cd

Bit modyfikacji (zabrudzenia) jest ustawiany, jeśli na stronie zapisano choćby 1 bit. Jeśli nie zapisano, nie trzeba strony zrzucić na dysk.

Zastępowanie stron jest podstawą stronicowania na żądanie.

Praktycznie każdy proces wykonuje się z użyciem mniejszej ilości ramek niż by wynikało z wielkości procesu.

Mechanizm działa efektywnie przy dobrze opracowanych algorytmach przydziału ramek i algorytmach zastępowania stron.

Algorytmy zastępowania stron

- Algorytmy optymalizowane pod kątem minimalizacji **częstości braku stron**
- Algorytmy ocenia się na podstawie ich wykonania dla pewnego ciągu odniesień (odwołań) do pamięci i zsumowanie liczby braku stron w tym ciągu

Algorytm FIFO (first in, first out)

- O każdej ze stron zapamiętuje się informację, kiedy ona została sprowadzona do pamięci.
- Zastępuje się „najstarszą” stronę

Przykład:

Dla ciągu odniesień do stron: 1,2,3,4,1,2,5,1,2,3,4,5

...1,2,3,4,1,2,5,1,2,3,4,5

Dla 3 dostępnych ramek dla procesu:

1	4	5	
2	1	3	(9 braków stron)
3	2	4	

Dla 4 dostępnych ramek dla procesu:

1	5	4	
2	1	5	(10 braków stron)
3	2		
4	3		

Anomalia Bledy'ego ^^^^

Algorytm optymalny

Zastąp tę stronę, która najdłużej nie będzie używana

Dla ciągu odniesień:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
-	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-	-	1	1	3	3	3	3	3	3	3	3	3	3	3	3	3	1	1	1

9 braków stron, nie ma anomalii Bledy'ego

Algorytm optymalny jest trudny w realizacji, ponieważ wymaga wiedzy o przyszłym ciągu odniesień.

Jest on używany głównie do teoretycznych studiów porównawczych (o ile % dany algorytm jest gorszy od opt.)

Inne algorytmy

- Algorytm LRU (Latest Recently Used) - zastępowanie stron, które były najdawniej używane. Typowanie najstarszych poprzez:
 - **licznik** (w tablicy stron jest rejestr czasu ostatniego używania strony)
 - **stos** (przy każdym odwołaniu do strony, jej numer jest wyjmowany i umieszczany na szczycie stosu)
- Algorytmy bazujące na metodzie LRU
 - **z bitami odniesienia** (po odwołaniu do strony, znacznik przyjmuje wartość 1)
 - **dodatkowe bity odwołań** (co stały czas ustawianie kolejnych bitów rotacyjnie)
 - **druga szansa** (jeśli bit odniesienia=1 to zeruje się go, zmienia czas na bieżący i przegląda kolejne strony -FIFO. Jeśli bit=0 to się stronę wymienia)

Inne algorytmy, cd

- ulepszony algorytm drugiej szansy

wykorzystuje się dwa bity: bit odniesienia i bit modyfikacji, jako parę. Powstają cztery klasy stron:

(0,0) - nie używana ostatnio i nie zmieniana (najlepsza ofiara)

(0,1) - nie używana ostatnio, ale zmieniana (gorsza, bo trzeba ją zapisać)

(1,0) - Używana ostatnio, ale nie zmieniana (prawdopodobnie za chwilę będzie znów używana)

(1,1) - używana ostatnio i zmieniona (chyba będzie używana niedługo, a poza tym trzeba ją zapisać - najgorsza kandydatka na ofiarę)

Zastępujemy pierwszą napotkaną stronę z najniższej klasy

Inne algorytmy, cd

- algorytmy zliczające

Wprowadzamy licznik odwołań do strony

LFU (least frequently used) -wyrzucić najrzadziej używaną

MFU (most frequently used) - bo najrzadziej używana jest chyba niedawno wprowadzona do pamięci i będzie niedługo w użyciu

Obydwa te algorytmy niezbyt dobrze przybliżają optimum.

- algorytmy buforowania stron

Zanim się usunie ofiarę, wczytuje się potrzebną stronę do wolnej ramki.

Zaletą - można wcześniej uruchomić proces, zanim strona-ofiara zostanie zapisana na dysku. Zapis robi się w wolnej chwili.

Po zapisie opróżnioną ramkę dopisuje się do listy wolnych.

Przydział ramek

Każdemu procesowi system musi przydzielić pulę ramek pamięci potrzebnych do jego pracy.

Trzy najpopularniejsze algorytmy przydziału:

- **równy** (każdy proces dostaje tyle samo ramek)
np 50 ramek i 5 procesów, to każdy dostaje po 10
- **proporcjonalny** (liczba przydzielonych ramek proporcjonalna do wielkości procesu)
- **priorytetowy** (liczba przydzielonych ramek zależy tylko od priorytetu procesu, albo od priorytetu i wielkości)

Globalne i lokalne zastępowanie ramek

- **Zastępowanie lokalne** - proces może zastępować ramki wyłącznie z puli tych, które dostał przy przydziale
- **Zastępowanie globalne** - Proces może korzystać z puli wszystkich wolnych ramek, nawet jeżeli są wstępnie przydzielone innemu procesowi. Proces może zabrać ramkę drugiemu.

Praktyka wykazała, że zastępowanie globalne daje lepszą przepustowość systemu.

Szamotanie

Jeśli proces nie ma wystarczającej liczby ramek, to w pewnym momencie musi wymienić stronę, która będzie potrzebna w niedługim czasie. W konsekwencji, kolejne braki stron będą występowały bardzo często. Taki proces „szamocze się”, spędzając więcej czasu na stronicowaniu niż na wykonaniu. Zmniejsza się wykorzystanie procesora.

Scenariusz szamotania

- jeżeli wykorzystanie jednostki centralnej jest za małe, planista przydziału procesora **zwiększa wieloprogramowość**
- nowy proces **zabiera** ramki pozostałym procesom
- zaczyna brakować ramek
- strony ustawiają się w kolejce do urządzenia stronicującego, a jednocześnie zmniejsza się kolejka procesów gotowych



Szamotanie, cd

- wykorzystanie procesora maleje, bo procesy stoją w kolejce
- system zwiększa wieloprogramowość
- sytuacja staje się tragiczna - żaden proces nie pracuje, tylko wszystkie stronicują

Jak powstrzymać szamotanie lub do niego nie dopuścić?

- stosować metodę lokalnego lub priorytetowego przydziału stron
- stosować mechanizmy zmniejszenia wieloprogramowości przy szamotaniu
- zapewnić dostawę wystarczającej ilości ramek.

Mierzenie częstości braków stron - pomiar szamotania

Jeśli proces przekracza górną granicę częstości - dostaje wolną ramkę

Jeśli przekracza dolną granicę - odbiera mu się ramkę.

Problem operacji we-wy

- proces zamawia operację we-wy i ustawia się w kolejce do urządzenia
- procesor przydziela się innym procesom
- występują braki stron
- w wyniku algorytmu globalnego zastępowania stron zostaje wymieniona strona zawierająca bufor we-wy procesu czekającego na operację we-wy
- zaczyna się operacja we-wy i nadpisuje dane innego procesu!

Zapobieganie:

- zakaz wykonywania operacji we-wy wprost do pamięci użytkownika - ale kopiowanie czasochłonne
- blokowanie stron w pamięci - strony czekające lub realizujące operację we-wy nie mogą być zastępowane

Planowanie przydziału procesora

W pamięci operacyjnej znajduje się kilka procesów jednocześnie.

Kiedy jakiś proces musi czekać, system operacyjny odbiera mu procesor i oddaje do dyspozycji innego procesu.

Planowanie przydziału procesora jest podstawową funkcją każdego systemu operacyjnego.



Fazy procesora i wejścia-wyjścia

załaduj
przechowaj
dodaj
przechowaj
czytaj z pliku

Faza procesora

Czekaj na urządzenie
wejścia-wyjścia

Faza wejścia-wyjścia

przechowaj
dodaj
zwiększ indeks
pisz do pliku

Faza procesora

Czekaj na urządzenie
wejścia-wyjścia

Faza wejścia-wyjścia

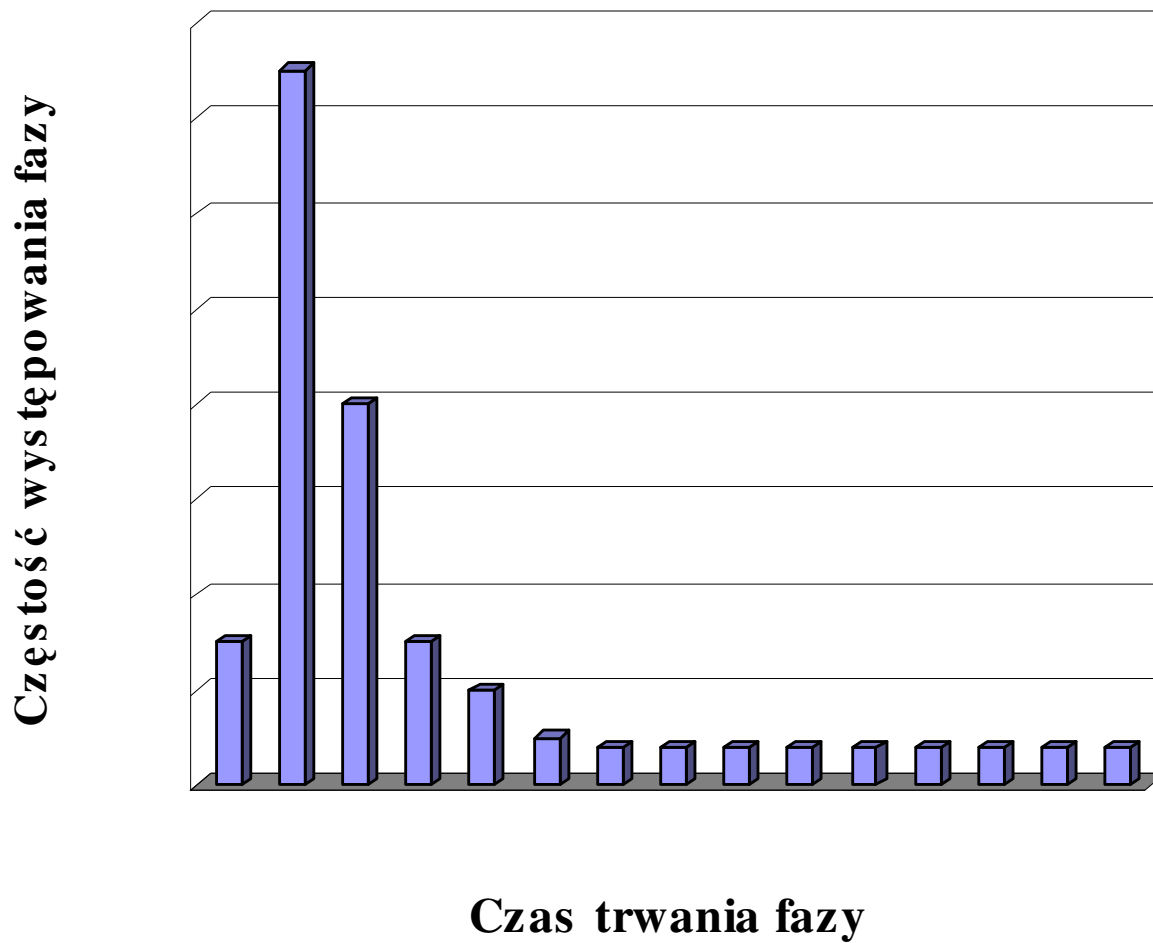
załaduj
przechowaj
dodaj
przechowaj
czytaj z pliku

Faza procesora

Czekaj na urządzenie
wejścia-wyjścia

Faza wejścia-wyjścia

Czasy faz procesora



Planowanie przydziału procesora

Proces ograniczony przez wejście-wyjście
ma zazwyczaj dużo krótkich faz procesora.

Proces ograniczony przez procesor może mieć
mało, lecz bardzo długich faz procesora.



Planowanie przydziału procesora

Decyzje o zmianie przydziału procesora podejmowane są, gdy:

1. proces przeszedł od stanu aktywności do czekania, np. na zakończenie potomka, lub zamówił we-wy,
2. proces przeszedł od stanu aktywności do gotowości, np. wskutek wystąpienia przerwania,
3. proces przeszedł od stanu czekania do gotowości, np. po zakończeniu we-wy,
4. proces kończy działanie.

Jeśli planowanie odbywa się tylko w przypadkach 1 i 4, to mamy do czynienia z **planowaniem niewywłaszczeniowym**, w przeciwnym wypadku planowanie jest **wywłaszczeniowe**.

Planowanie przydziału procesora

Bez wywłaszczeń - proces, który otrzymał procesor, odda go dopiero przy przejściu do stanu oczekiwania lub po zakończeniu. Nie wymaga zegara. Stosowane w systemach Windows.

Planowanie wywłaszczające jest **ryzykowne**.

Należy brać pod uwagę fakt, że proces w momencie wywłaszczenia może być w trakcie wykonywania funkcji systemowej.

Zabezpieczenia:

- system czeka z przełączeniem kontekstu do zakończenia f.s.
- przerwania są blokowane przy przejściu do ryzykownych fragmentów kodu jądra,
- nie wywłaszcza procesu, gdy wewnętrzne struktury jądra są niespójne.

Ekspedytor (dispatcher)

Jest to moduł, który przekazuje procesor do dyspozycji procesu wybranego przez planistę krótkoterminowego. Do jego zadań należy:

- przełączanie kontekstu,
- przełączanie procesu do trybu użytkownika,
- wykonanie skoku do odpowiedniego adresu w programie w celu wznowienia działania programu

Opóźnienie ekspedycji - czas, jaki ekspedytor zużywa na wstrzymanie jednego procesu i uaktywnienie innego.

Czas ten powinien być możliwie najkrótszy - ekspedytor powinien więc być jak najszybszy.

Kryteria planowania

- Wykorzystanie procesora - w realnych systemach od 40% (słabe) do 90% (intensywne),
- Przepustowość - mierzona ilością procesów kończonych w jednostce czasu (dla długich - kilka na godzinę, dla krótkich - kilka na sekundę),
- czas cyklu przetwarzania - od nadejścia procesu do systemu, do jego zakończenia (suma czasów oczekiwania na wejście do pamięci, w kolejce procesów gotowych, okresów aktywności i operacji wejścia-wyjścia)
- Czas oczekiwania - suma czasów w których procesor czeka w kolejce p. gotowych,
- Czas odpowiedzi - w systemach interakcyjnych - czas od wysłania żądania do rozpoczęcia odpowiedzi.

Planowanie optymalne

- **Maksymalne** wykorzystanie procesora,
- **maksymalna** przepustowość,
- **minimalny** czas cyklu przetwarzania,
- **minimalny** czas oczekiwania,
- **minimalny** czas odpowiedzi.

Planowanie metodą FCFS (pierwszy zgłoszony - pierwszy obsłużony)

Zgłaszają się trzy procesy:

1. **P1** o czasie trwania fazy: 24 ms
2. **P2** o czasie trwania fazy: 3 ms
3. **P3** o czasie trwania fazy: 3 ms

Diagram Gantta dla planowania FCFS:



Średni czas oczekiwania wynosi:

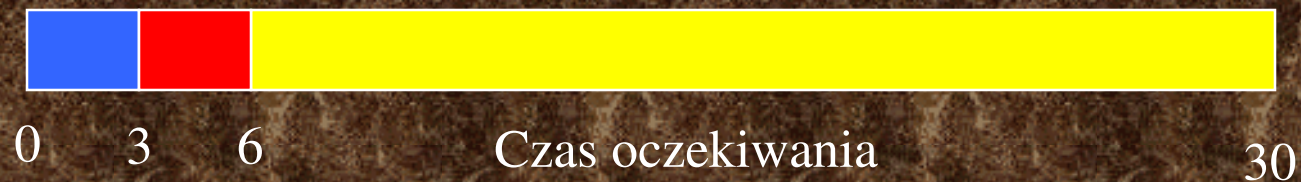
$$(0+24+27)/3 = 17 \text{ ms}$$

Planowanie metodą FCFS

A gdyby procesy zgłosiły się w kolejności:

1. P2
2. P3
3. P1

Diagram Gantta dla planowania FCFS:



Średni czas oczekiwania wyniósłby wtedy:

$$(0+3+6)/3 = 3 \text{ ms}$$

Planowanie metodą FCFS

- Pierwszy przykład pokazuje, że planowanie metodą FCFS nie jest optymalne
- Średni czas oczekiwania bardzo zależy od kolejności zgłoszenia procesów
- Efekt konwoju - małe procesy czekają na zwolnienie procesora przez jeden wielki proces
- Algorytm FCFS jest niewywłaszczający
- Algorytm ten jest nieużyteczny w systemach z podziałem czasu, w których procesy powinny dostawać procesor w regularnych odstępach czasu

Planowanie metodą SJF (najpierw najkrótsze zadanie)

Zgłaszają się cztery procesy:

1. **P1** o czasie trwania fazy: 6 ms
2. **P2** o czasie trwania fazy: 8 ms
3. **P3** o czasie trwania fazy: 7 ms
4. **P4** o czasie trwania fazy: 3 ms

Diagram Gantta dla planowania SJF:



Średni czas oczekiwania wynosi:

$$(3+16+9+0)/4 = 7 \text{ ms}$$

Dla metody FCFS wynosiłby:

$$(0+6+14+21)/4 = 10,25 \text{ ms}$$

Planowanie metodą SJF

- Można udowodnić, że planowanie tą metodą jest optymalne
- Umieszczenie krótkiego procesu przed długim w większym stopniu zmniejsza czas oczekiwania krótkiego procesu niż zwiększa czas oczekiwania procesu długiego
- Algorytm SJF jest często używany przy planowaniu długoterminowym
- Problem - nie można przewidzieć długości następnej fazy procesora, można ją tylko szacować, najczęściej za pomocą średniej wykładniczej z poprzednich faz procesora:

Planowanie metodą SJF

$$f_{n+1} = \alpha t_n + (1-\alpha) f_n$$

gdzie:

α – współczynnik wagi (0÷1)

t_n – długość n-tej fazy procesora,

f_n - informacje o poprzednich fazach

gdy $\alpha = 0$ - bierzemy pod uwagę tylko dawniejszą historię,

a gdy $\alpha = 1$ - bierzemy pod uwagę tylko ostatnie fazy.

Najczęściej $\alpha = 0,5$

Planowanie metodą SJF

Algorytm SJF może być wywłaszczający lub niewywłaszczający.

Gdy do kolejki dochodzi nowy proces, który posiada fazę procesora krótszą niż pozostały czas w bieżącym procesie, algorytm wywłaszczający odbiera procesor bieżącemu procesowi i przekazuje go krótszemu.

Metoda ta nazywa się SRTF (shortest remaining time first) czyli „najpierw najkrótszy pozostały czas”.

Algorytm niewywłaszczający pozwala na dokończenie fazy procesora.

Planowanie priorytetowe

Mechanizm bardzo podobny do SJF, ale kryterium szeregowania jest priorytet procesu.

Najpierw wykonywane są procesy o ważniejszym priorytecie.

Priorytety mogą być definiowane wewnętrznie, na podstawie pewnych cech procesu (np. wielkość pamięci, limity czasu, zapotrzebowanie na urządzenia we-wy itd..)

Priorytety definiowane zewnętrznie mogą np. zależeć od ważności użytkownika, jego firmy czy też od innych politycznych uwarunkowań.

Planowanie priorytetowe

Wady:

Procesy o niskim (mało ważnym) priorytecie mogą nigdy nie dostać procesora (głodzenie, nieskończone blokowanie)

Przykład: w 1973 r. w wycofywanym z eksploatacji na MIT komputerze IBM 7094 wykryto „zagłodzony” proces o niskim priorytecie, który został zapuszczony do wykonania w 1967 roku ;-(((

Rozwiązaniem problemu jest „postarzanie” czyli podnoszenie priorytetu procesów oczekujących zbyt długo.

Przykład: przydział mieszkania dla dziadka w Alternatywy 4

Planowanie rotacyjne (Round-Robin - RR)

- Zaprojektowane specjalnie do systemów z podziałem czasu.
- Każdy proces otrzymuje kwant czasu (10-100ms), po upływie którego jest wywłaszczany i umieszczany na końcu kolejki zadań gotowych.
- Kolejny proces do wykonania jest wybierany zgodnie z algorytmem FCFS
- Jeżeli jest n procesów gotowych a kwant czasu wynosi q , to każdy proces czeka nie dłużej niż $(n-1)*q$ jednostek czasu.

Planowanie metodą RR, kwant czasu = 25 ms

- 1. **P1** o czasie trwania fazy: 24 ms
- 2. **P2** o czasie trwania fazy: 3 ms
- 3. **P3** o czasie trwania fazy: 3 ms

Diagram Gantta dla $q=25$ ms:



Średni czas oczekiwania wynosi:

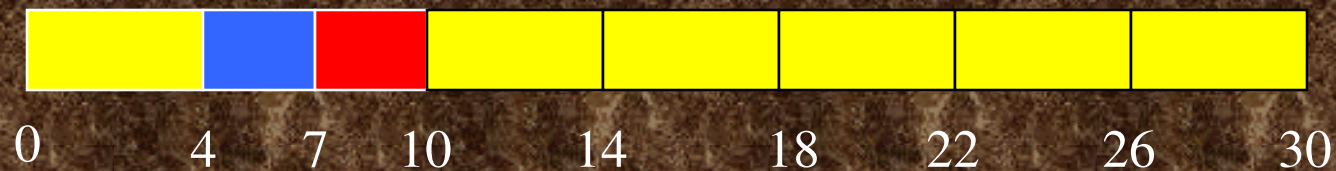
$$(0+24+27)/3 = 17 \text{ ms}$$

Przełączeń kontekstu: 2

Planowanie metodą RR, kwant czasu = 4 ms

- 1. **P1** o czasie trwania fazy: 24 ms
- 2. **P2** o czasie trwania fazy: 3 ms
- 3. **P3** o czasie trwania fazy: 3 ms

Diagram Gantta dla $q=4$ ms:



Średni czas oczekiwania wynosi:

$$(17)/3 = 5,66 \text{ ms}$$

Przełączeń kontekstu: 7

Planowanie metodą RR, kwant czasu = 1 ms

- 1. **P1** o czasie trwania fazy: 24 ms
- 2. **P2** o czasie trwania fazy: 3 ms
- 3. **P3** o czasie trwania fazy: 3 ms

Diagram Gantt dla $q=1$ ms:



Przełączeń kontekstu: 29

Planowanie rotacyjne, cd.

Wydajność algorytmu bardzo zależy od kwantu czasu q

- Gdy q jest duże, to algorytm RR przechodzi w FCFS
- Gdy q jest bardzo małe, to znaczna część czasu procesora poświęcona jest na przełączanie kontekstu

Dobra zasada:

80% faz procesora powinno być krótszych niż jeden kwant czasu.

Wielopoziomowe planowanie kolejek

Kolejka procesów gotowych jest podzielona na oddzielne podkolejki, na przykład:

- kolejka procesów pierwszoplanowych (foreground),
- kolejka procesów drugoplanowych (background).

Przykładowe proponowane algorytmy planowania:

- procesy pierwszoplanowe: RR
- procesy drugoplanowe: FCFS

Procesy pierwszoplanowe mają absolutny priorytet nad drugoplanowymi.

Aby nie „zagłodzić” procesów 2-planowych, stosuje się podział czasu na kolejki, przykładowo:

- kolejka pierwszoplanowa - 80% czasu procesora,
- kolejka drugoplanowa - pozostałe 20%

Kolejki wielopoziomowe ze sprzężeniem zwrotnym

Mechanizm ten pozwala na przesuwanie procesów pomiędzy kolejkami

Proces, który używa za dużo procesora, można „karnie” przenieść do kolejki o niższym priorytecie i przez to dać szerszy dostęp do procesora innym procesom.

Dzięki temu procesy ograniczone przez we-wy i procesy interakcyjne mogą pozostać w kolejkach o wyższych priorytetach.

Długo oczekujące procesy z kolejki niskopriorytetowej mogą być przeniesione do ważniejszej - działa mechanizm postarzania procesów (przeciwdziała ich głodzeniu).

Planowanie ze sprzężeniem zwrotnym jest najbardziej złożonym algorytmem planowania przydziału procesora.



Kolejki wielopoziomowe ze sprzężeniem zwrotnym - przykład

- Kolejka trzypoziomowa: K0, K1, K2
- Proces wchodzący trafia do kolejki k0 i dostaje kwant czasu 8 ms.
- Jeśli nie zakończy się w tym czasie, jest wyrzucany na koniec niższej kolejki K1.
- Gdy kolejka K0 się opróżni i przyjdzie czas wykonywania naszego procesu, dostaje on kwant czasu 16 ms.
- Jeśli i w tym czasie proces nie skończy działania, jest wyrzucany na koniec kolejki K2, obsługiwanej w porządku FCFS (oczywiście pod warunkiem, że kolejki K0 i K1 są puste).
- Tak więc najszybciej wykonywane są procesy do 8 ms, nieco wolniej procesy od 8 do $8+16=24$ ms, a najdłużej czekają procesy długie (są obsługiwane w cyklach procesora nie wykorzystanych przez kolejki 1 i 2)

Planowanie wieloprocessorowe.

- Planowanie heterogeniczne - dla systemów sieciowych, rozproszonych, o różnych procesorach - bardzo trudne w realizacji
- Planowanie homogeniczne - dla procesorów tego samego typu. Nie stosuje się oddzielnych kolejek dla każdego procesora - możliwość przestojów niektórych procesorów.

Wieloprzetwarzanie symetryczne - każdy procesor sam wybiera procesy ze wspólnej kolejki - działanie musi być bardzo starannie zaprogramowane, aby uniknąć np. dublowania.

Wieloprzetwarzanie asymetryczne - jeden procesor jest nadrzędny (master) i on planuje przydział procesów, a pozostałe (slave) wykonują przydzielone im zadania.

Planowanie w systemie Solaris

- Planowanie w wielopoziomowych kolejkach ze sprzężeniem zwrotnym,
- 4 klasy procesów: **real time**, **system**, **time sharing**, **interactive**
- Priorytet globalny i priorytety w obrębie klas
- Proces potomny dziedziczy klasę i priorytet
- Klasa domyślna - **time sharing**
- Im większy priorytet, tym mniejszy kwant czasu
- Klasa **system** - procesy jądra
- Klasa **interactive** - wyższy priorytet mają aplikacje graficzne X11
- Wątki o tym samym priorytecie planowane są algorytmem RR

Planowanie w systemie Windows 2000

- Planowanie priorytetowe z wywłaszczeniami
- 32 kolejki procesów, 6 klas priorytetów, 7 relatywnych priorytetów w obrębie klas
- Priorytet domyślny w klasie - **normal**
- Wątek jest wykonywany aż zostanie wywłaszczony przez proces o wyższym priorytecie, zużyje kwant czasu, wykonuje operację we-wy lub inne blokujące wywołanie systemowe, bądź też zakończy się.
- Proces wybrany na ekranie ma zwiększany trzykrotnie kwant czasu

Synchronizacja procesów

Proces producenta - zmodyfikowany (licznik)

repeat

...

produkuj jednostka w nast_p

...

while licznik =n **do** nic_nie_rob;

bufor [we] := nast_p;

we=we+1 **mod** n;

licznik:=licznik+1;

until false;

Synchronizacja procesów, cd

Zmodyfikowany proces konsumenta

repeat

while licznik=0 **do** nic_nie_rob;

nast_k := bufor [wy];

wy=wy+1 **mod** n;

licznik:=licznik-1;

...

konsumuj jednostka z nast_k

...

until false;

Synchronizacja procesów, cd

Wartość licznika wynosi 5.

Po tym czasie producent wyprodukował 1 jednostkę, a konsument skonsumował również 1 jednostkę.

Pytanie: ile wynosi licznik?

Synchronizacja procesów, cd

- | | |
|----------------------------|---------|
| 1. P: rejestr1:=licznik | (r1=5) |
| 2. P: rejestr1:=rejestr1+1 | (r1=6) |
| 3. K: rejestr2:=licznik | (r2=5) |
| 4. K: rejestr2:=rejestr2-1 | (r2=4) |
| 5. P: licznik:=rejestr1 | (l. =6) |
| 6. K: licznik:=rejestr2 | (l. =4) |

Ile wynosi licznik?

4

- | | |
|--------------------------|---------|
| 5'. K: licznik:=rejestr2 | (l. =4) |
| 6'. P: licznik:=rejestr1 | (l. =6) |

Ile wynosi licznik?

6

Bez synchronizacji procesów możemy nigdy nie uzyskać 5

Szkodliwa rywalizacja (race condition)

Jeżeli kilka procesów współbieżnie wykorzystuje i modyfikuje te same dane, to wynik działań może zależeć od kolejności w jakiej następował dostęp do danych. Następuje wtedy **szkodliwa rywalizacja**.

Sekcja krytyczna

Każdy ze współpracujących procesów posiada fragment kodu w którym następuje zmiana wspólnych danych. Jest to **sekcja krytyczna** procesu.

Jedno z zadań synchronizacji - jeśli jeden z procesów znajduje się w swojej sekcji krytycznej, inne nie mogą w tym czasie wejść do swoich krytycznych sekcji.

Każdy proces musi prosić (w sekcji wejściowej) o pozwolenie na wejście do swojej sekcji krytycznej.



Proces z sekcją krytyczną

repeat

Sekcja wejściowa

Sekcja krytyczna

Sekcja wyjściowa

until false;

Warunki poprawnego działania s. k.

- **Wzajemne wykluczanie:** jeśli proces działa w swej sekcji krytycznej, to żaden inny proces nie działa w swojej.
- **Postęp:** tylko procesy nie wykonujące swoich reszt mogą kandydować do wejścia do sekcji krytycznych i wybór ten nie może być odwlekany w nieskończoność.
- **Ograniczone czekanie:** Musi istnieć graniczna ilość wejść innych procesów do ich sekcji krytycznych po tym, gdy dany proces zgłosił chęć wejścia do swojej sekcji krytycznej i zanim uzyskał na to pozwolenie.

Przykładowy algorytm synchronizacji

- **Wspólne zmienne:**
var znacznik: array [0..1] of integer;
numer: 0..1;
- Na początku znacznik [0]=znacznik[1]=0
- Odpowiedni fragment procesu i (proces konkurencyjny ma nr j):

repeat

znacznik[i]:=1;

numer:=j;

while (znacznik[j]=1 and numer = j) do nic_nie rob;

sekcja krytyczna

znacznik[i]:=0;

reszta

until false;

Rozkazy niepodzielne

- Są to sprzętowe rozkazy składające się z kilku kroków, ale wykonywane nieprzerwanie, np.:

```
function Testuj_i_Ustal (var znak:boolean):boolean;  
  begin  
    Testuj_i_Ustal:=znak; }  
    znak:=true;          }  
  end;
```

Zastosowanie:

```
repeat  
  while Testuj_i_Ustal (wspolna) do nic_nie_rob;  
    sekcja krytyczna  
  wspolna:=false;  
  reszta  
until false;
```



Semafor

- Są to sprzętowe zmienne całkowite, do których dostęp jest za pomocą tylko dwóch niepodzielnych operacji:
- czekaj (S): $\text{while } S \leq 0 \text{ do nic_nie_rob;}$
 $S := S - 1;$ }
- sygnalizuj (S): $S := S + 1;$

Zastosowanie:

wspolna:typu semafor

repeat

czekaj (wspolna);

sekcja krytyczna

sygnalizuj(wspolna);

reszta

until false;

Przykład: Instrukcja S2 w procesie P2 musi być wykonana po zakończeniu wykonywania instrukcji S1 w procesie P1:

S1;

sygnalizuj (synch);

czekaj (synch);

S2;

Semaforey z blokowaniem procesu

(unika się „wirowania” procesu przed semaforem)

- Proces, zamiast aktywnie czekać, jest umieszczany w kolejce związanej z danym semaforem i „usypiany”
- Operacja **sygnalizuj**, wykonana przez inny proces, „budzi” proces oczekujący i umieszcza go w kolejce procesów gotowych do wykonania.

Implementacja:

type semaphore=record

 wartosc: integer;

 L:list of process;

end;

Semaforey z blokowaniem procesu

```
czekaj(S): S.wartosc:=S.wartosc-1;  
           if S.wartosc <0 then begin  
               dolacz dany proces do S.L;  
               blokuj;  
           end;
```

```
sygnalizuj (S): S.wartosc:=S.wartosc+1;  
               if S.wartosc <=0 then begin  
                   usun jakis proces P z S.L;  
                   obudz (p);  
               end;
```

Jeśli $wartosc < 0$ to $abs(wartosc)$ - liczba procesów czekających na ten semafor

Semaforey w systemach wieloprocesorowych

- W systemach jednoprocessorowych niepodzielność operacji „czekaj” i „sygnalizuj” można zapewnić poprzez blokadę przerwań na czas wykonywania ich rozkazów.
- W środowisku wieloprocesorowym nie ma możliwości blokowania przerwań z innych procesorów - w takim przypadku wykorzystuje się rozwiązania z sekcji krytycznych - operacje „czekaj” i „sygnalizuj” są sekcjami krytycznymi. Ponieważ ich kody są małe (kilkanaście rozkazów), to zajmowane są rzadko i przypadki aktywnego czekania nie występują często i trwają krótko.

Problem czytelników i pisarzy

(synchronizacja wielu procesów zapisujących i czytających te same dane)

- Problem 1: żaden z czytelników nie powinien czekać, chyba że pisarz w tym momencie pisze
- Problem 2: Jeśli pisarz czeka na dostęp do dzieła, to żaden nowy czytelnik nie rozpocznie czytania

Rozwiązanie problemu:

Procesy dzielą następujące zmienne:

```
var wyklucz, pis: semaphore;  
liczba_czyt:integer;
```

Semafor **pis** jest wspólny dla procesów czytelników i pisarzy; obydwa semaforey przyjmują wartość początkową 1 a liczba_czyt - 0.

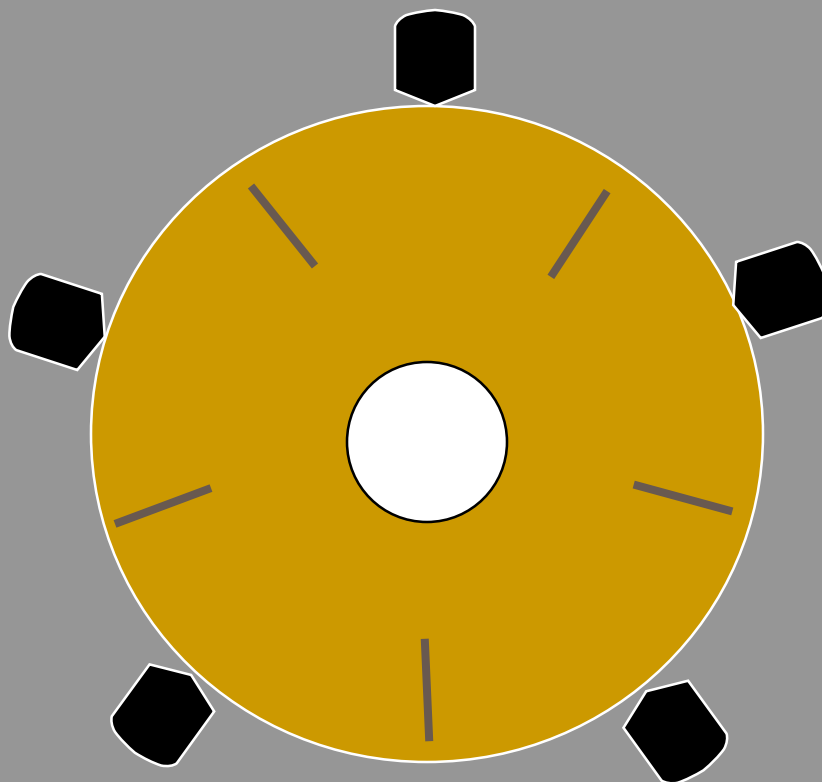
Semafor **pis** organizuje wykluczanie kilku pisarzy, a także jest zmieniany przez pierwszego i ostatniego czytelnika

Problem czytelników i pisarzy

- Proces pisarza:
czekaj (pis);
...
Pisanie
...
Sygnalizuj (pis);
- Proces czytelnika:
czekaj (wyklucz); #tylko 1 proces może działać w tej sekcji
liczba_czyt=liczba_czyt+1;
if liczba_czyt = 1 then czekaj (pis); #bo może być wewnątrz pisarz
sygnalizuj (wyklucz); #mogą wchodzić inni czytelnicy
...
czytanie
czekaj (wyklucz); #znów sekcja wyłączna
liczba_czyt:=liczba_czyt - 1;
if liczba_czyt =0 then sygnalizuj (pis); #może ew. wejść pisarz
sygnalizuj (wyklucz);

Problem filozofów

Zob.: http://student.uci.agh.edu.pl/~nowakow/projekt_sysopy/strona/projekt_pliki/wstep.htm



Kiedy myślący filozof poczuje głód,usiłuje podnieść najpierw lewą, a potem prawą pałeczkę. Po zakończonym jedzeniu odkłada pałeczki z powrotem na stół.

Problem filozofów - program

var paleczka: array [0..4] of semaphore;

repeat

czekaj (paleczka [i]);

czekaj (paleczka [i+1 mod 5]);

...

jedzenie

sygnalizuj (paleczka [i]);

sygnalizuj (paleczka [i+1 mod 5]);

...

myslenie

until false

Problem filozofów - blokada

Co będzie, jeśli każdy z filozofów w tym samym czasie poczuje głód i podniesie lewą pałeczkę?

Zapobieganie:

- zostawić jedno miejsce wolne przy stole,
- Pozwolić filozofowi na podniesienie pałeczek jak obydwie są dostępne (sprawdzanie i podnoszenie w sekcji krytycznej)
- rozwiązanie asymetryczne - nieparzysty filozof podnosi najpierw lewą pałeczkę, a parzysty - prawą

Region krytyczny

Jest to konstrukcja służąca do synchronizacji w języku wyższego poziomu.

Składnia:

region V when B do S;

gdzie:

V: zmienna używana wspólnie przez wiele procesów

type T: integer;

var V shared T;

Podczas wykonywania instrukcji S żaden inny proces nie ma prawa dostępu do zmiennej V.

Jeśli warunek B jest **true**, to proces może wykonać instrukcję S; w przeciwnym wypadku musi czekać na zmianę B oraz na opuszczenie sekcji krytycznej przez inne procesy.

Region krytyczny- implementacja

```
var bufor: shared record
    magazyn: array [0..n-1] of jednostka
    licznik, we, wy: integer;
end;

region bufor when licznik < n do begin
    magazyn[we] := nast_p;
    we := we + 1 mod n;
    licznik := licznik + 1;
end;

region bufor when licznik > 0 do begin
    nast_k := magazyn[wy];
    wy := wy + 1 mod n;
    licznik := licznik - 1;
end;
```


Monitor

Podstawową wadą semafora jest to, że nie jest to mechanizm strukturalny, przez co trudno jest analizować programy współbieżne i ogarnąć wszystkie możliwe przeploty rozkazów procesora.

Monitor stanowi połączenie modułu programistycznego z sekcją krytyczną i jest po prostu modułem zawierającym deklaracje stałych, zmiennych, funkcji i procedur. Wszystkie te obiekty, z wyjątkiem jawnie wskazanych funkcji i procedur są lokalne w monitorze i nie są widoczne na zewnątrz niego. Wskazane funkcje i procedury (tzw. eksportowane) są widoczne na zewnątrz monitora. Mogą je wywoływać procesy i za ich pośrednictwem manipulować danymi ukrytymi w monitorze. Monitor zawiera też kod, który służy do jego inicjacji, na przykład do ustawienia wartości początkowych zmiennych deklarowanych w monitorze.

Monitor

Jednocześnie co najwyżej jeden proces może być w trakcie wykonania kodu znajdującego się w monitorze. Jeśli jakiś proces wywoła procedurę eksportowaną przez monitor i rozpocznie jej wykonanie, to do czasu powrotu z tej procedury żaden inny proces nie może rozpocząć wykonania tej ani żadnej innej procedury/ funkcji monitora. O procesie, który wywołał funkcję/procedurę monitora i nie zakończył jeszcze jej wykonania, będziemy mówić, że *znajduje się w monitorze*. Zatem jednocześnie w monitorze może przebywać co najwyżej jeden proces.

Taka definicja narzuca naturalny sposób korzystania ze zmiennych współdzielonych przez procesy. Po prostu należy umieścić je w monitorze i dostęp do nich realizować za pomocą eksportowanych procedur i funkcji. Programista korzystający w taki właśnie sposób ze zmiennej dzielonej nie musi myśleć o zapewnianiu wyłączności w dostępie do niej - robi to za niego automatycznie sam monitor.

Transakcje

Transakcją jest zbiór operacji stanowiących logicznie spójną funkcję. Jest to na przykład ciąg operacji czytania lub pisania zakończonych operacją zatwierdzenia lub zaniechania. Transakcja zaniechana nie powinna pozostawić śladów w danych, które zdążyła już zmienić.

Wycofanie transakcji powinno zapewnić odtworzenie danych sprzed transakcji.

Spójność danych i ich sprawne odzyskiwanie po np. awarii systemu jest najważniejszą sprawą w bazach danych różnego rodzaju.

Jest to też rodzaj synchronizacji danych.

Transakcje - szeregowanie

Szeregowanie transakcji polega na takim zaplanowaniu wzajemnego przeplatania się rozkazów z kilku transakcji, aby nie występowały konflikty pisania/czytania tych samych danych

T1:
czytaj (a)
pisz (a)
czytaj (b)
pisz (b)

T2:

czytaj (a)
pisz (a)
czytaj (b)
pisz (b)



T1:
czytaj (a)
pisz (a)

czytaj (b)
pisz (b)

T2:

czytaj (a)
pisz (a)

czytaj (b)
pisz (b)

Transakcje - protokół blokowania

Z każdym obiektem danych kojarzy się **zamek**, od którego zależy dostęp do danych, np.

- Jeżeli transakcja dostaje dostęp do obiektu danych w **trybie wspólnym**, to może czytać ten obiekt, ale nie może go zapisywać
- Jeśli dostaje obiekt w **trybie wyłącznym**, to wolno jej zarówno czytać jak i zapisywać ten obiekt.

Każda transakcja musi zamawiać zamek blokujący obiekt w takim trybie, aby mogła wykonać zaplanowaną operację.

Odzyskiwanie za pomocą rejestru

Rejestr to zapis w pamięci trwałej, określający wszystkie zmiany w danych wykonywane podczas transakcji.

Każdy rekord w rejestrze (logu) zawiera następujące dane:

- nazwa transakcji,
- nazwa jednostki danych,
- stara wartość,
- nowa wartość,
- inne dane dotyczące transakcji, np. zaniechanie

Zanim rozpocznie się wykonywanie transakcji, w rejestrze zapisuje się rekord informujący o **rozpoczęciu** transakcji.

Każdy zapis (przez transakcję) poprzedzony jest zapisem odpowiedniego rekordu w rejestrze.

Gdy dochodzi do zatwierdzenia transakcji, w rejestrze zapisuje się rekord **zatwierdzenie**.

Odzyskiwanie za pomocą rejestru

Tworzenie rejestru jest pamięcio- i czasochłonne, ale dla bardzo ważnych danych nie jest to cena wygórowana.

Przy rekonstrukcji danych na podstawie rejestru korzysta się z dwóch procedur:

- **wycofaj** - odtwarza wszystkie dane uaktualnione przez transakcję T, nadając im stare wartości,
- **przywróć** - nadaje nowe wartości wszystkim danym uaktualnionym przez transakcję T.

Transakcja musi być wycofana, jeśli w rejestrze znajduje się rekord **rozpoczęcie**, a nie ma rekordu **zatwierdzenie**.

Transakcja musi być przywrócona, jeśli w rejestrze jest rekord **rozpoczęcie** oraz rekord **zatwierdzenie** dla danej transakcji.

Punkty kontrolne

Odtwarzanie za pomocą rejestru ma pewne wady:

- Proces przeglądania rejestru jest czasochłonny,
- większość transakcji zapisanych w rejestrze odbyła się pomyślnie przed awarią, odtwarzanie ich z rejestru jest dublowaniem pracy.

Dla przyspieszenia ewentualnego odtwarzania, system organizuje co jakiś czas tzw. **punkty kontrolne**, w których:

- wszystkie rekordy pozostające w tej chwili w pamięci operacyjnej są zapisane w pamięci trwałej (na dysku),
- wszystkie zmienione dane, pozostające w pamięci ulotnej, muszą być zapisane w pamięci trwałej,
- w rejestrze transakcji zapisuje się rekord **punkt kontrolny**

Po awarii przegląda się rejestr od końca. Po napotkaniu rekordu **punkt kontrolny**, przywracanie rozpoczyna się od pierwszej transakcji po nim.

Synchronizacja w systemie Solaris

Ze względu na implementację procesów czasu rzeczywistego, wielowątkowość i obsługę wielu procesorów, synchronizacja za pomocą sekcji krytycznych nie znalazła zastosowania.

Zastosowano **zamki adaptacyjne**.

Zamek rozpoczyna działalność jak standardowy semafor. Jeśli dane są już w użyciu, to zamek wykonuje jedną z dwu czynności:

- jeśli zamek jest utrzymywany przez wątek aktualnie wykonywany, to inny wątek ubiegający się o zamek będzie czekać (gdyż aktywny wątek niedługo się zakończy),
- jeśli zamek jest utrzymywany przez wątek nieaktywny, to wątek żądający zamka blokuje się i usypia, gdyż czekanie na zamek będzie dłuższe.

Synchronizacja w systemie Solaris

Zamki adaptacyjne stosuje się, gdy dostęp do danych odbywa się za pomocą krótkich fragmentów kodu (zamknięcie na czas wykonywania co najwyżej kilkuset rozkazów).

W przypadku dłuższych segmentów kodu stosuje się **zmienne warunkowe**.

Jeśli zamek jest zablokowany, to wątek wykonuje operację **czekaj** i usypia. Wątek zwalniający zamek sygnalizuje to następnemu z kolejki uśpionych co tamtego budzi.

Blokowanie zasobów w celu pisania lub czytania jest wydajniejsze niż używanie semaforów, ponieważ dane mogą być czytane przez kilka wątków równocześnie, a semafony dają tylko indywidualny dostęp do danych.

Systemy operacyjne

Studia niestacjonarne,
Informatyka stosowana, II rok

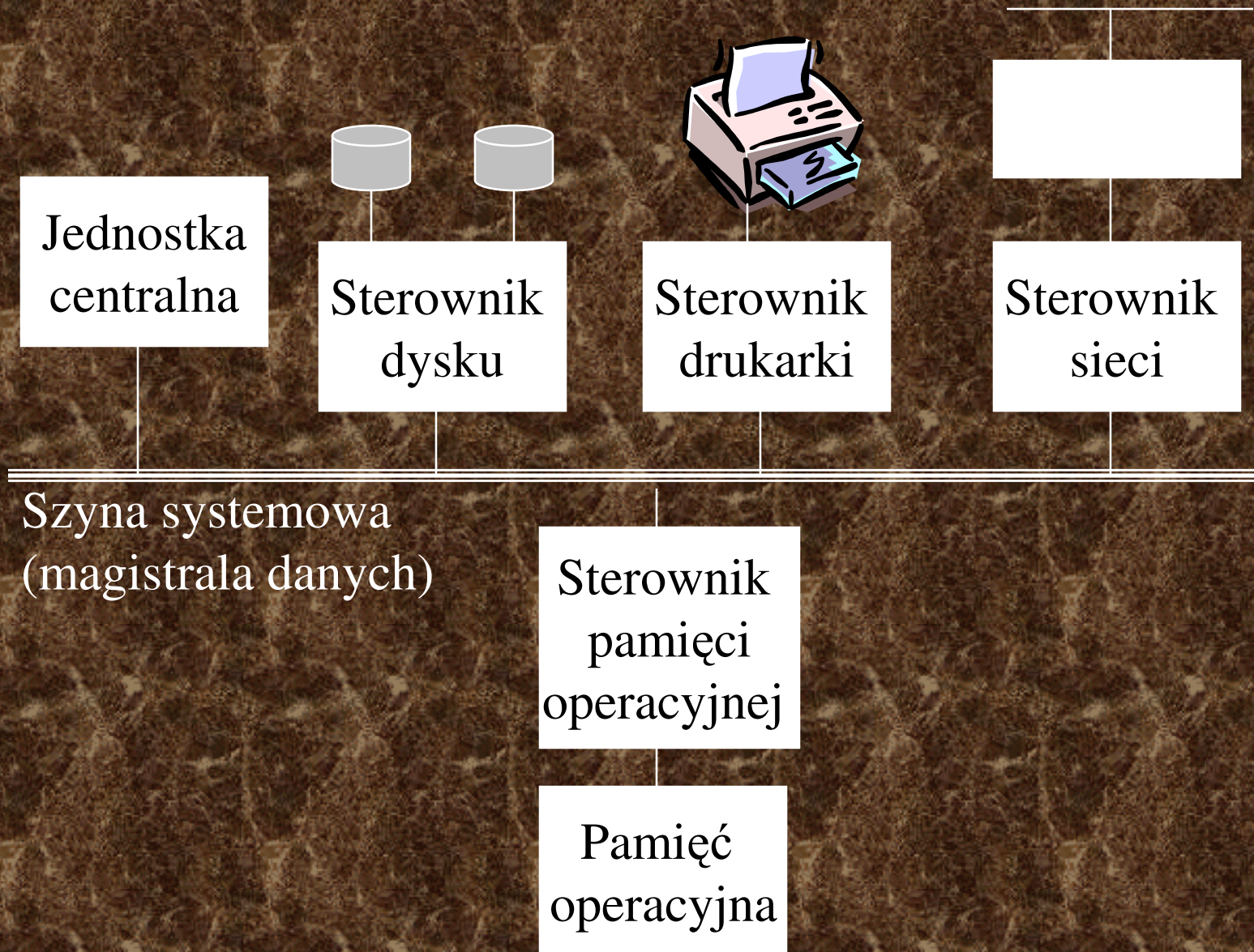
Krzysztof Wilk

*Katedra Informatyki Stosowanej
i Modelowania*

wilk@agh.edu.pl

Konsultacje: poniedziałki 12-13
oraz piątki przed i po zajęciach;
B-4, pok. 207

Budowa systemu komputerowego



Przerwania

- **Przerwanie** jest sygnałem pochodzącym od sprzętu lub oprogramowania i sygnalizuje wystąpienie zdarzenia
- sygnał przerwania sprzętowego pochodzi z zewnętrznego układu obsługującego przerwania sprzętowe; przerwania te służą do komunikacji z urządzeniami zewnętrznymi, np. z klawiaturą, napędami dysków itp.
- Sygnały przerwań od sprzętu wysyłane są do procesora najczęściej za pośrednictwem szyny systemowej

Przerwania

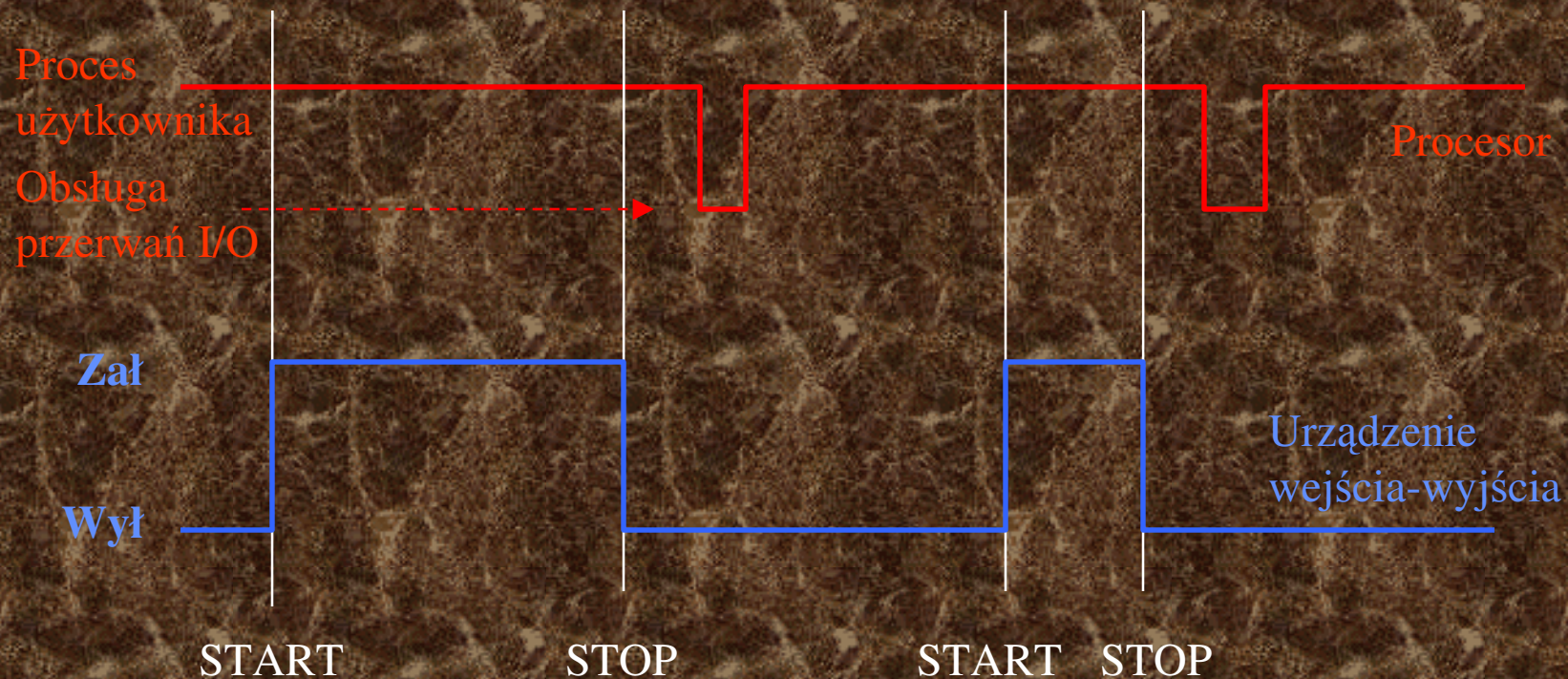
- **Przerwanie** polega na tym, że urządzenie wymagające obsługi procesora, np. wejścia/wyjścia (klawiatura), moduł DMA (Direct Memory Access), posiada specjalne połączenie (pojedynczy przewód) łączące niniejsze urządzenie z procesorem. Linia ta nosi nazwę linii przerwania. W momencie kiedy urządzenie chce być obsługiwane przez procesor wystawia umówiony sygnał na linii przerwania – z reguły jest to stan wysoki (wyzwalanie poziomem sygnału) lub też zmiana stanu z 0 na 1 (wyzwalanie zboczem).

Zdarzenia powodujące przerwanie:

- Zakończenie operacji wejścia-wyjścia
- Dzielenie przez zero,
- Niedozwolony dostęp do pamięci,
- Zapotrzebowanie na usługę systemu,
- itd., itp..

Każdemu przerwaniu odpowiada procedura obsługi.

Wykres czasowy przerwań procesu wykonującego operację wejścia-wyjścia



Wektor przerwań

Aby przyspieszyć operację obsługi przerwań stosuje się tablicę wskaźników do procedur obsługujących poszczególne przerwania.

Indeksy tej tablicy odpowiadają numerom urządzeń „generujących” przerwania, a elementami tablicy są adresy procedur obsługujących przerwania.

Przy przejściu do obsługi przerwania należy zapamiętać adres przerwanego rozkazu, a także np. zawartości rejestrów, jeżeli obsługa przerwania zmienia je.

W nowych systemach adres powrotny przechowywany jest na stosie systemowym. Podczas obsługi jednego przerwania inne są wyłączone, lub ustalone są **priorytety przerw** (przerwania maskowane).

Wyjątki

Przerwania wewnętrzne, nazywane wyjątkami (ang. exceptions) – zgłaszane przez procesor dla sygnalizowania sytuacji wyjątkowych (np. dzielenie przez zero); dzielą się na trzy grupy:

- faults (niepowodzenia) – sytuacje, w których aktualnie wykonywana instrukcja powoduje błąd; gdy procesor powraca do wykonywania przerwanego kodu wykonuje jeszcze raz tę samą instrukcję, która wywołała wyjątek;



Wyjątki

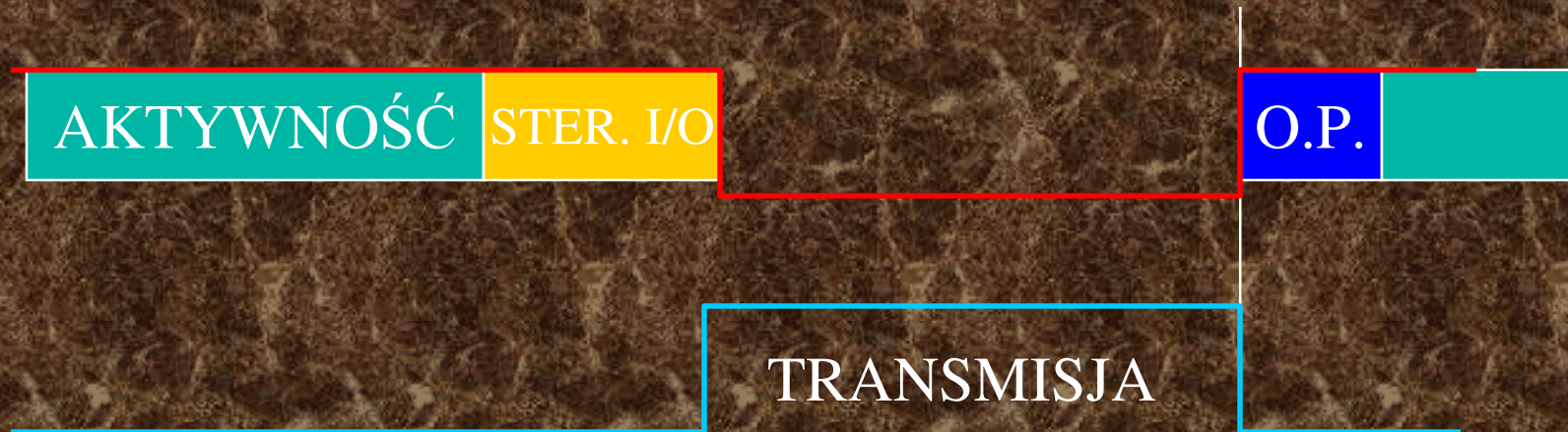
- traps (pułapki) – sytuacja, która nie jest błędem, jej wystąpienie ma na celu wykonanie określonego kodu; wykorzystywane przede wszystkim w debuggerach; gdy procesor powraca do wykonywania przerwane kodu, wykonuje następną instrukcję, po tej która wywołała wyjątek;
- aborts (nienaprawialne) – błędy, których nie można naprawić.

Przerwania programowe

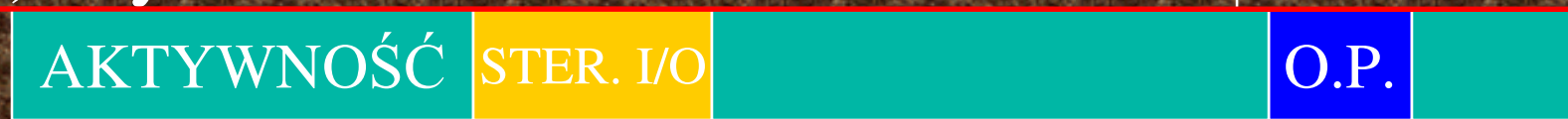
- z kodu programu wywoływana jest procedura obsługi przerwania; najczęściej wykorzystywane do komunikacji z systemem operacyjnym, który w procedurze obsługi przerwania (np. w DOS 21h, Windows 2fh, Linux x86 przerwanie 80h) umieszcza kod wywołujący odpowiednie funkcje systemowe w zależności od zawartości rejestrów ustawionych przez program wywołujący, lub do komunikacji z oprogramowaniem wbudowanym jak procedury BIOS lub firmware.

Obsługa wejścia-wyjścia

a) synchroniczna:



b) asynchroniczna



Asynchroniczne wejście-wyjście

W czasie wykonywania operacji wejścia-wyjścia jednostka centralna systemu może być użyta do obliczeń lub do rozpoczynania operacji wejścia-wyjścia z innych urządzeń. Ponieważ operacje I/O są powolne w porównaniu z szybkością procesora, może on w międzyczasie obsłużyć kilka innych zadań.

Bezpośredni dostęp do pamięci (DMA)

W przypadku wolnych urządzeń I/O, obsługa przesyłania danych z bufora urządzenia do pamięci, nie angażuje zbytnio procesora.

Dla urządzeń szybkich (dysk, sieć) wygodniej jest przesyłać cały blok danych bezpośrednio do pamięci, bez angażowania procesora. Umożliwia to mechanizm Direct Memory Access, realizowany sprzętowo.

Uwaga! Kradnie cykle pamięci!



Dwa tryby pracy procesora

Procesor rozróżnia dwa tryby:

- Tryb użytkownika (z ograniczeniami)
- Tryb monitora, nadzorcy, systemu, uprzywilejowany. Wykonuje potencjalnie niebezpieczne operacje. Są to tzw. **operacje uprzywilejowane**.



Operacje wejścia/wyjścia nie są bezpośrednio dostępne dla użytkownika (musi o nie prosić system operacyjny).

Użytkownik ma dostęp tylko do pamięci przydzielonej swojemu programowi.

Nie wolno też dopuścić do tego, aby system stracił kontrolę nad procesorem, np. przez nieskończoną pętlę w programie użytkownika.

Proces

Proces jest programem, który jest aktualnie wykonywany.

Jest to jednostka pracy w systemie.

System składa się ze zbioru procesów, z których część to procesy systemu operacyjnego, a pozostałe są procesami użytkowymi.

Zarządzanie procesami przez OS

- Tworzenie i usuwanie procesów użytkowych i systemowych,
- wstrzymywanie i wznowianie procesów,
- dostarczanie mechanizmów synchronizacji procesów,
- dostarczanie mechanizmów komunikacji procesów,
- dostarczanie mechanizmów obsługi zakleszczeń.



Zarządzanie pamięcią przez OS

- Ewidencja aktualnie zajętych obszarów pamięci, informacja o użytkownikach danych obszarów,
- decydowanie o tym, które procesy mają być załadowane do zwolnionych obszarów pamięci,
- przydzielanie i zwalnianie obszarów pamięci stosownie do potrzeb.

Zarządzanie plikami przez OS

- Tworzenie i usuwanie plików,
- tworzenie i usuwanie katalogów,
- dostarczanie elementarnych informacji do manipulowania plikami i katalogami,
- odwzorowanie plików na obszary pamięci pomocniczej,
- składowanie plików na trwałych nośnikach pamięci.

Inne funkcje systemu operacyjnego

- Zarządzanie systemem wejścia-wyjścia (buforowanie, pamięć, spooling, interfejs, moduły sterujące),
- zarządzanie pamięcią pomocniczą (dyskową),
- praca sieciowa,
- system ochrony,
- system interpretacji poleceń (powłoka).

Usługi systemu operacyjnego

- Wykonanie programu,
 - operacje wejścia-wyjścia,
 - manipulowanie systemem plików,
 - komunikacja między procesami,
 - wykrywanie błędów.
-
- przydzielanie zasobów,
 - rozliczanie
 - ochrona

Funkcje systemowe

Tworzą interfejs pomiędzy wykonywanym programem a systemem operacyjnym.

Poprzez f.s. program użytkownika „daje zlecenia” systemowi operacyjnemu.

F.S. - Nadzorowanie procesów

- Załadowanie lub wykonanie programu,
- zakończenie lub zaniechanie procesu,
- utworzenie lub zakończenie procesu (potomnego),
- pobranie lub ustawienie parametrów procesu,
- czekanie czasowe,
- oczekiwanie na zdarzenie lub sygnalizacja zdarzenia,
- przydział i zwolnienie pamięci.

F.S. - Operacje na plikach

- Utworzenie lub usunięcie pliku,
- otwarcie lub zamknięcie pliku,
- czytanie, pisanie lub zmiana położenia,
- pobranie lub ustawienie atrybutów pliku.

F.S. - Operacje na urządzeniach

- Zamówienie lub zwolnienie urządzenia,
- czytanie, pisanie lub zmiana położenia,
- pobranie lub ustawienie atrybutów urządzenia,
- logiczne przyłączanie lub odłączanie urządzeń.

F.S. - Utrzymywanie informacji

- Pobranie lub ustawienie daty/czasu,
- pobranie lub ustawienie danych systemowych,
- pobranie atrybutów procesu, pliku lub urządzenia,
- ustawienie atrybutów procesu, pliku lub urządzenia.

F.S. - Komunikacja

- Utworzenie, usunięcie połączenia komunikacyjnego,
- nadawanie, odbieranie komunikatów,
- przekazywanie informacji o stanie,
- przyłączanie i odłączanie urządzeń zdalnych.

Programy systemowe

- Manipulowanie plikami,
- informowanie o stanie systemu,
- tworzenie i zmienianie zawartości plików,
- translatory języków programowania,
- ładowanie i wykonywanie programów,
- komunikacja.

Struktura systemu - UNIX

Użytkownicy		
Powłoki i polecenia Kompilatory i interpretery Biblioteki systemowe		
Interfejs funkcji systemowych jądra		
Sygnały Obsługa terminali System znakowego wejścia-wyjścia Moduły sterujące terminali	System plików Wymiana System blokowego wejścia-wyjścia Moduły sterujące dysków i taśm	Planowanie przydziału procesora Zstępowanie stron Stronicowanie na żądanie Pamięć wirtualna
Interfejs między jądrem a sprzętem		
Sterowniki terminali Terminale	Sterowniki urządzeń Dyski i taśmy	Sterowniki pamięci Pamięć operacyjna

Proces

Proces jest wykonywanym programem.

Wykonanie procesu musi przebiegać w sposób sekwencyjny (w dowolnej chwili na zamówienie naszego procesu może być wykonany co najwyżej jeden rozkaz programu).

Proces - elementy

- kod programu (sekcja tekstu),
- bieżąca czynność (wskazana przez licznik rozkazów),
- zawartość rejestrów procesora,
- stos procesu,
- sekcja danych.

Stan procesu

- **nowy** - proces został utworzony.
- **aktywny** - są wykonywane instrukcje,
- **oczekujący** - czeka na wystąpienie zdarzenia, np. zakończenie operacji wejścia-wyjścia,
- **gotowy** - czeka na przydział procesora,
- **zakończony** - zakończył działanie.

Blok kontrolny procesu

Numer procesu

Stan procesu

Licznik rozkazów

Rejestry

Adresy pamięci

Wykaz otwartych
plików

...

Blok kontrolny procesu

- Stan procesu (nowy, gotowy, aktywny...),
- licznik rozkazów - adres następnego rozkazu do wykonania,
- rejestry procesora,
- informacje do planowania przydziału procesora (priorytet procesu, wskaźniki do kolejek),



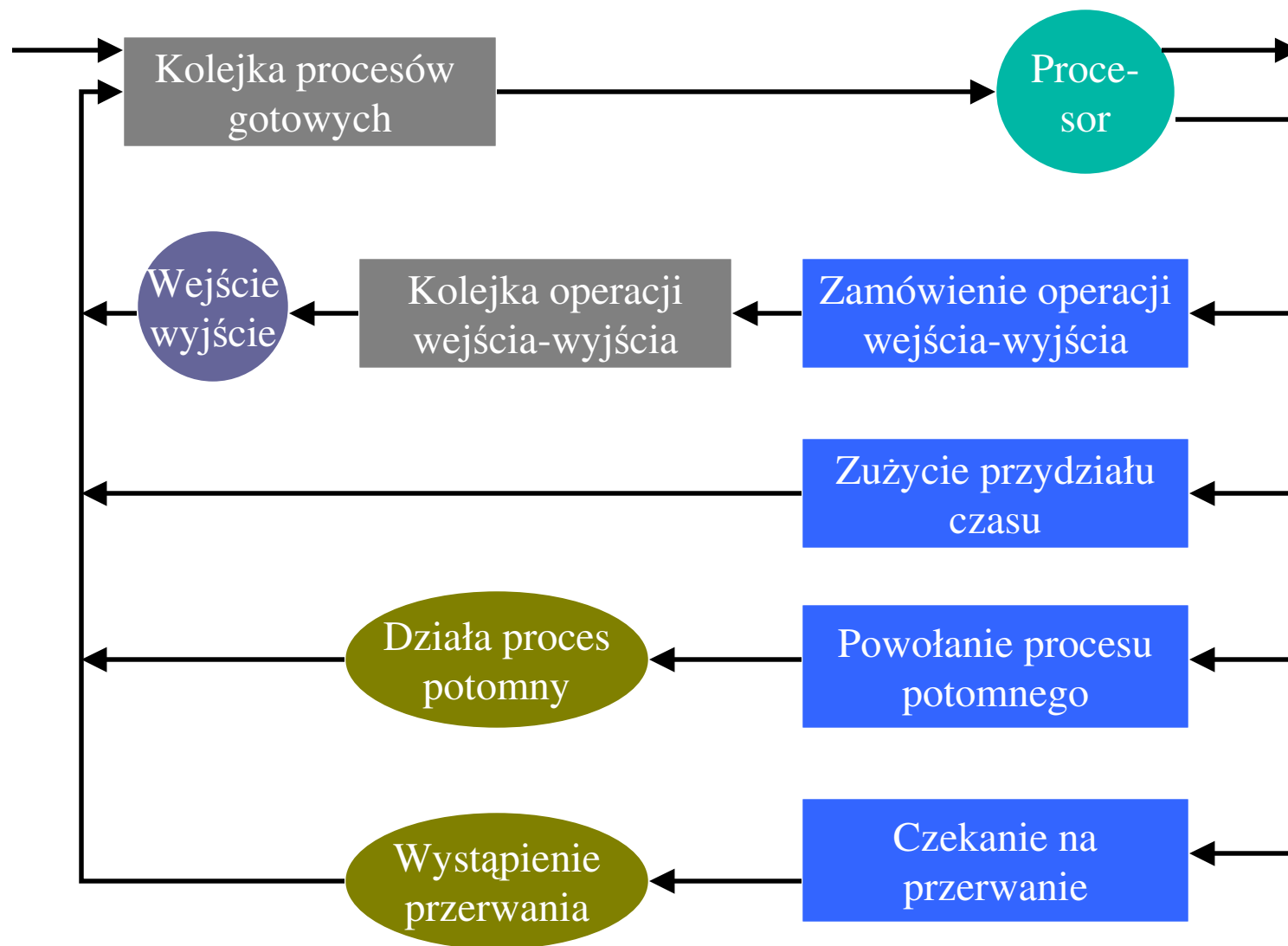
Blok kontrolny procesu, cd.

- zarządzanie pamięcią (granice pamięci, tablice stron, tablice segmentów...)
- informacje do rozliczeń (zużyty czas procesora, czas całkowity, konta...),
- informacje o stanie wejścia-wyjścia (urządzenia przydzielone do procesu, wykaz otwartych plików itd.).

Kolejki planowania procesów

- Kolejka zadań (job queue) - tworzą ją procesy wchodzące do systemu.
- Kolejka procesów gotowych (ready queue) - procesy gotowe do działania, umieszczone w pamięci,
- Kolejki do urządzeń (device queue) - procesy czekające na konkretne urządzenie.

Diagram kolejek



Planiści

- Planista długoterminowy (planista zadań) - wybiera procesy do kolejki procesów gotowych, do pamięci. Jest on wywoływany stosunkowo rzadko (sekundy) i nie musi być szybki.
- Planista krótkoterminowy (planista przydziału procesora) - wybiera proces z puli procesów gotowych i przydziela mu procesor. Jest on wywoływany b. często (co ms) i musi być b. szybki.



Planiści

Procesy możemy podzielić na:

- Ograniczone przez wejście-wyjście (więcej czasu zajmują operacje we-wy niż korzystanie z procesora)
- Ograniczone przez procesor (potrzebują znacznie więcej czasu procesora niż dla operacji we-wy)

Zadaniem planisty długoterminowego jest dobór optymalnej mieszanki zadań ograniczonych przez procesor i przez we-wy.

Planista średnioterminowy

Występuje w niektórych systemach z podziałem czasu. Jego zadaniem jest, w koniecznych przypadkach, zmniejszanie stopnia wieloprogramowości poprzez wysyłanie części zadań chwilowo na dysk (swapping). Pomaga to w doborze lepszego zestawu procesów w danej chwili, lub dla zwolnienia obszaru pamięci.

Przełączanie kontekstu

Podczas przejścia procesora z wykonywania jednego procesu do drugiego należy przechować stan starego procesu i załadować przechowany stan nowego.

Z punktu widzenia systemu są to działania nieproduktywne, tak jak przygotowanie czy sprząatanie stanowiska pracy, ale są niezbędne przy wieloprogramowości.

Mechanizm wątków pozwala na redukcję czasu przełączania kontekstu.



Tworzenie procesu

Proces macierzysty tworzy potomne za pomocą funkcji systemowej.

Nowy proces też może tworzyć potomne - powstaje wtedy drzewo procesów.

Proces macierzysty i potomek mogą dzielić w całości, w części, lub wcale nie dzielić ze sobą zasobów.

Proces macierzysty i potomek działają równolegle, lub też p. m. czeka, aż potomek zakończy działanie.



Tworzenie procesu, cd.

Proces potomny może być kopią procesu macierzystego, lub otrzymać zupełnie nowy program.

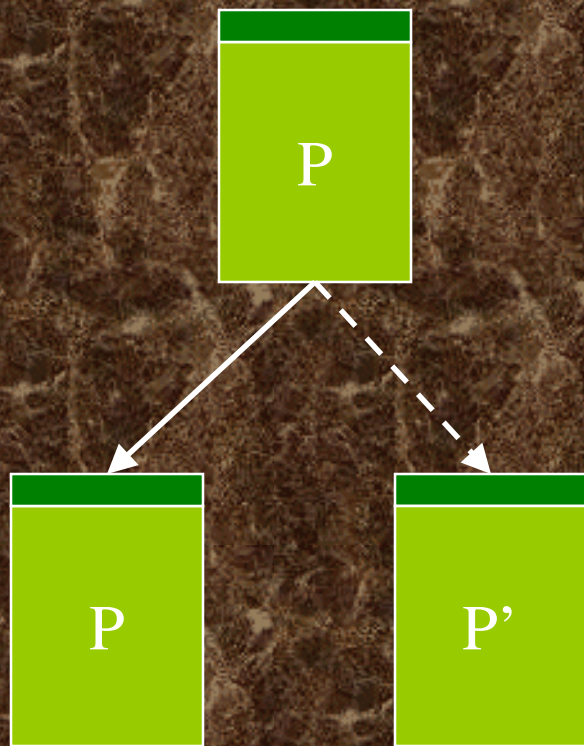
W systemie UNIX:

Nowy proces tworzy się funkcją systemową **fork**. Potomek zawiera kopię przestrzeni adresowej przodka - daje to możliwość komunikacji pomiędzy procesami.

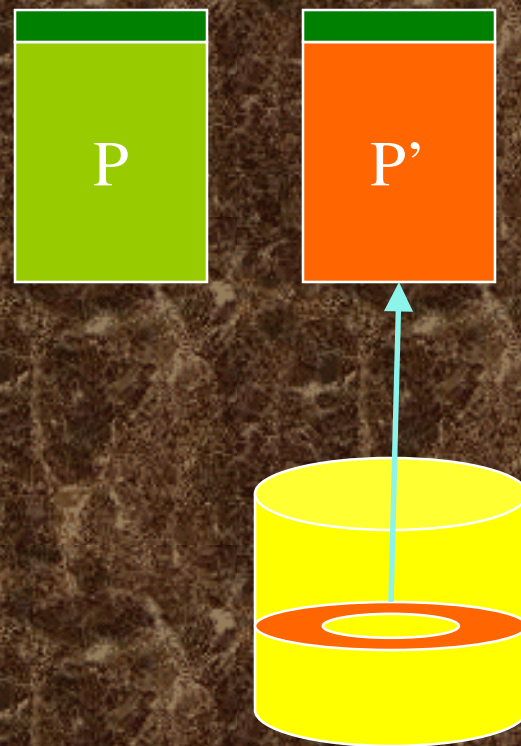
Funkcja systemowa **execve** ładuje nowy program do przestrzeni adresowej procesu (niszcząc poprzednią zawartość) i rozpoczyna jego wykonywanie.

Proces macierzysty albo tworzy nowych potomków, albo czeka na zakończenie procesu potomnego.

fork



execve



Tworzenie procesu, cd.

W systemie VMS:

Tworzy się nowy proces, umieszcza w nim nowy program rozpoczyna jego wykonywanie.

W systemie Windows NT:

Występują obydwa mechanizmy - albo tworzona jest kopia przestrzeni adresowej przodka, albo ładowany jest nowy program.

Kończenie procesu

Po wykonaniu ostatniej instrukcji proces prosi system operacyjny o usunięcie (f.s. **exit**).

System:

- przekazuje wyniki działania potomka do procesu macierzystego (wykonującego f.s. **wait**)
- odbiera potomkowi wszystkie zasoby (pamięć, otwarte pliki, bufory)

Kończenie procesu, cd

Proces macierzysty może spowodować „awaryjne” zakończenie potomka w przypadku gdy:

- potomek nadużył któregoś z przydzielonych zasobów,
- Wykonywane przez potomka zdanie stało się zbędne,
- proces macierzysty kończy się, a system nie zezwala na działanie „sieroty”.

Procesy współpracujące

Procesy są współpracujące, jeżeli „nasz” proces może wpływać na inne procesy, a inne procesy mogą wpływać na niego.

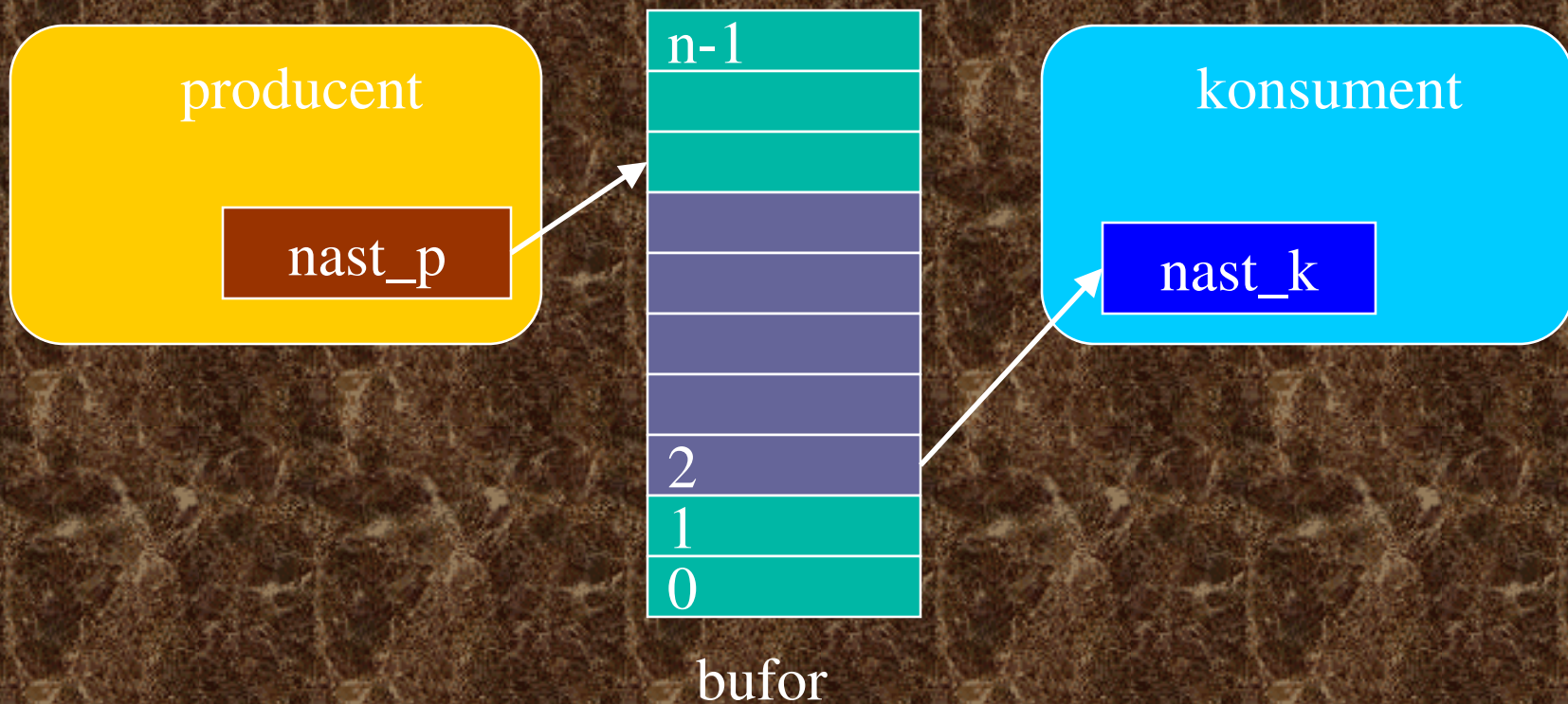
Zalety:

- Dzielenie informacji - kilka procesów może korzystać z danych np. z jednego pliku,
- Przyspieszenie obliczeń - w systemach wieloprocessorowych istnieje możliwość podziału zadania na mniejsze podzadania, wykonywane równolegle

Procesy współpracujące, cd.

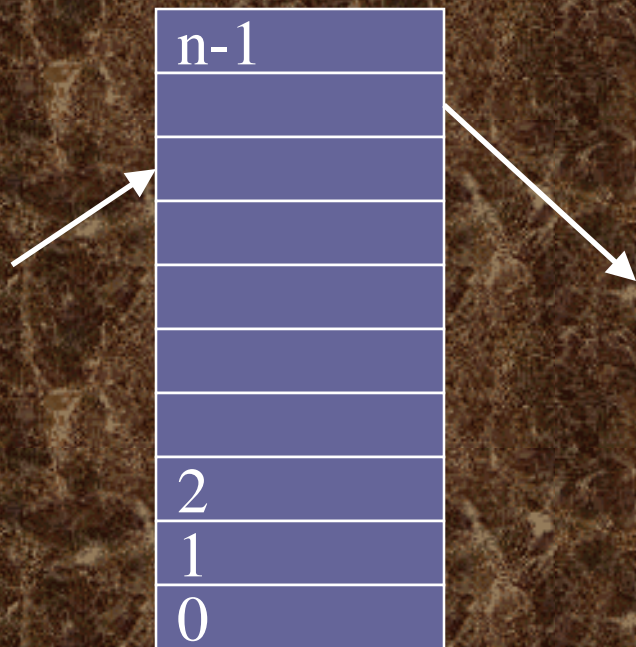
- Modularność - można konstruować system w sposób modularny
- Wygoda - jeden użytkownik może w tym samym czasie wykonywać kilka zadań, np. edycję, kompilację, drukowanie.

Współpraca, problem producenta i konsumenta



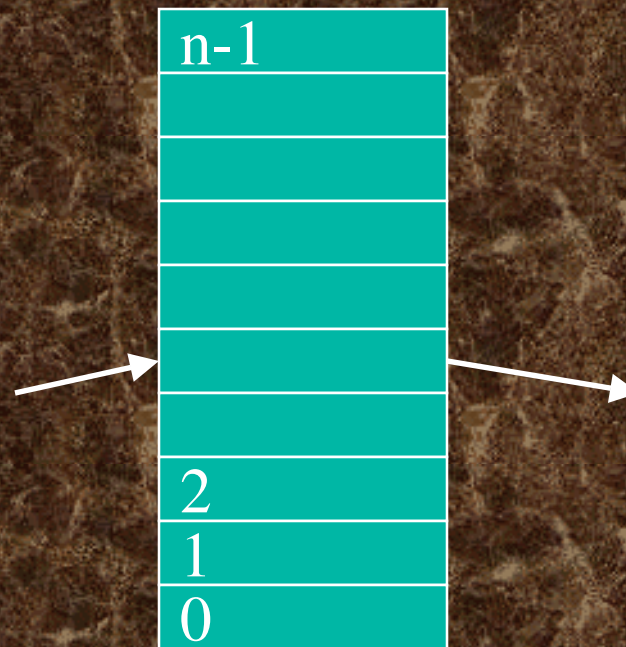
Bufor ograniczony

$$we+1 \bmod n = wy$$



Bufor
pełny

$$we = wy$$



Bufor
pusty

Bufor ograniczony - wspólne zmienne

var n;

type jednostka =.....;

var bufor array [0..n-1] **of** jednostka;

we,wy: 0..n-1

we- następne wolne miejsce w buforze,

wy - pierwsze zajęte miejsce w buforze

Proces producenta

repeat

...

produkuj jednostka w nast_p

...

while $we+1 \bmod n = wy$ **do** nic_nie_rob;

bufor [we] := nast_p;

$we = we+1 \bmod n$;

until false;

Proces konsumenta

repeat

while we = wy **do** nic_nie_rob;

nast_k := bufor [wy];

wy=wy+1 **mod** n;

...

konsumuj jednostka z nast_k

...

until false;

Komunikacja międzyprocesowa

System komunikatów -występują dwie operacje:

- nadaj komunikat
- odbierz komunikat

W celu realizacji komunikacji procesy muszą:

- ustanowić łączy komunikacyjne,
- nadawać i odbierać komunikaty.

Komunikacja za pomocą pamięci dzielonej lub szyny systemowej

Komunikacja międzyprocesowa - podstawowe pytania:

- Jak ustanawia się połączenie?
- Czy jedno łączy na więcej niż dwa procesy?
- Ile łączy pomiędzy parą procesów?
- Pojemność łącza? Obszar buforowy?
- Jaki rozmiar komunikatów? Stałej czy zmiennej długości?
- Czy łączy jedno czy dwukierunkowe?
- Komunikacja bezpośrednia czy pośrednia?



Komunikacja bezpośrednia

Dwie operacje elementarne:

nadaj (IDP1, komunikat)

odbierz(IDP2, komunikat)

gdzie IDP1 i IDP2 są identyfikatorami procesów 1 i 2.

Własności łącza:

- ustanawiane automatycznie pomiędzy parą procesów, wystarczy aby procesy znały swoje identyfikatory

Komunikacja bezpośrednia, cd.

- łączy dokładnie dwóch procesów,
- między parą procesów dokładnie jedno łączy,
- łączy zazwyczaj dwukierunkowe, dopuszczalne jednokierunkowe.

K. B. może służyć do przesyłania produktu w problemie producenta-konsumenta:

repeat

...

wytwarzaj jednostka w nast_p

...

nadaj (konsument, nast_p);

until false;

repeat

odbierz (producent, nast_k);

...

konsumuj jednostka z nast_k

...

until false;

Komunikacja pośrednia

- Komunikaty są nadawane i odbierane za pomocą skrzynek pocztowych (portów). Procesy mogą się z sobą skomunikować jeżeli mają wspólną skrzynkę pocztową.
- Łącze jest ustanawiane jedynie wtedy, gdy procesy dzielą jakąś skrzynkę,
- łącze może być związane z więcej niż dwoma procesami,
- każda para procesów może mieć kilka łączy, poprzez różne skrzynki pocztowe,
- łącze może być jedno lub dwukierunkowe.

Komunikacja pośrednia

problem trzech procesów

Trzy procesy P1, P2 i P3 dzielą jedną skrzynkę pocztową. Proces P1 wysyła komunikat, natomiast P2 i P3 próbują go odebrać - powstaje konflikt.

Jak tego uniknąć?

- Zezwalać jedynie na łącza pomiędzy dwoma procesami,
- zezwalać co najwyżej jednemu procesowi na wykonanie w danej chwili operacji odbioru,
- dopuścić, aby system wybrał proces do którego dotrze komunikat (albo P2 albo P3, a nie oba). System powinien poinformować nadawcę o wyborze.

Buforowanie

Kolejka komunikatów:

- pojemność zerowa - łącze nie dopuszcza aby czekał w nim jakikolwiek komunikat - nadawca czeka aż odbiorca odbierze,
- pojemność ograniczona - w kolejce może pozostawać tyle komunikatów, na ile zaprojektowano kolejkę. W przypadku kolejki pełnej nadawca musi czekać.
- Pojemność nieograniczona - kolejka ma potencjalnie nieskończoną długość. Nadawca nigdy nie czeka.

Sytuacje awaryjne

- Zakończenie procesu - system musi rozwiązać problemy:
 - gdy proces czeka na komunikaty z zakończonego,
 - gdy nadaje komunikaty do zakończonego,
- Utrata komunikatów
 - system wykrywa to i ponownie nadaje komunikat,
 - proces nadawczy wykrywa i ew. powtarza komunikat,
 - system wykrywa i powiadamia proces nadawczy.
- Zniekształcenie komunikatów - kontrola poprawności przez sumy kontrolne, sprawdzanie parzystości itd.

Wątki

Wątek jest podstawową jednostką wykorzystania procesora.
Jest to część składowa procesu wielowątkowego.

Wątek składa się z:

- licznika rozkazów,
- zbioru rejestrów,
- obszaru stosu

Takie elementy jak:

- sekcja kodu,
- sekcja danych,
- zasoby systemu (otwarte pliki, sygnały)

są wspólne dla kilku równorzędnych wątków.



Wątki

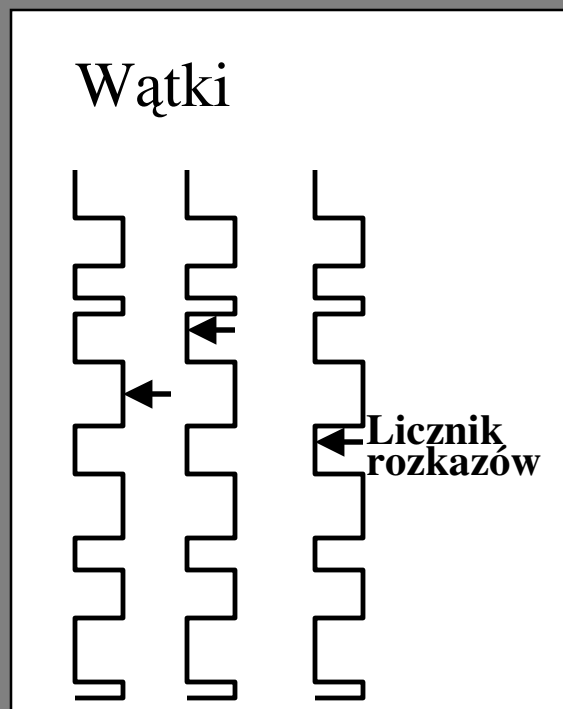
Zalety:

- Przełączanie między wątkami i tworzenie nowych wątków nie wymaga dużej aktywności procesora
- Przy przełączaniu nie trzeba wykonywać prac związanych z zarządzaniem pamięcią

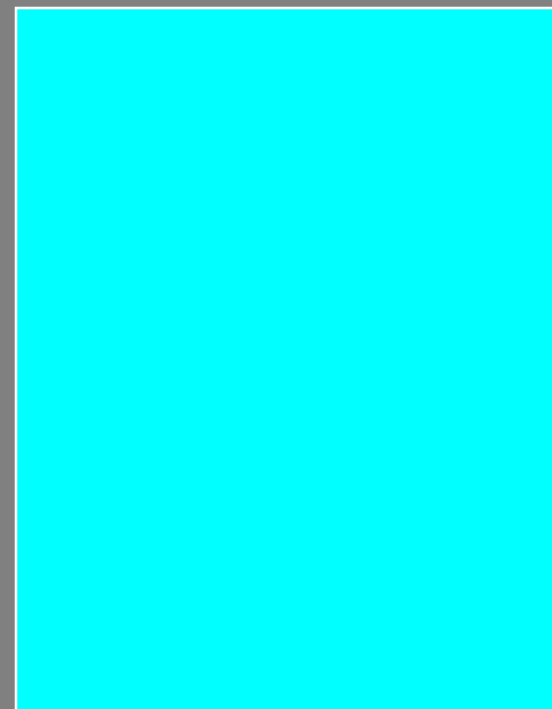
Wątki poziomu użytkownika

Przełączanie tych wątków nie wymaga wzywania systemu operacyjnego, może więc być szybkie. Niestety, przy odwołaniu do systemu, wszystkie wątki danego zadania muszą czekać na zakończenie funkcji systemowej.

Wątki



Segment tekstu



Segment danych

Wątki

Działanie wątków przypomina działanie procesów. Mogą być w stanach: gotowości, zablokowania, aktywności, kończenia.

Wątek może tworzyć wątki potomne, może się zablokować do czasu wykonania wywołania systemowego.

Jeśli jeden wątek jest zablokowany, może działać inny wątek.

Wątki jednego zadania są do siebie zależne - mogą np. nadpisywać stosy innych wątków.

Ale z drugiej strony - producent i konsument mogą być wątkami jednego zadania, a wspólny obszar danych znacznie zwiększy wydajność procesu.

Wątki

- Obsługiwane przez jądro (Mach, OS2, Windows)
- Wykonywane na poziomie użytkownika
- Mieszane (Solaris 2)

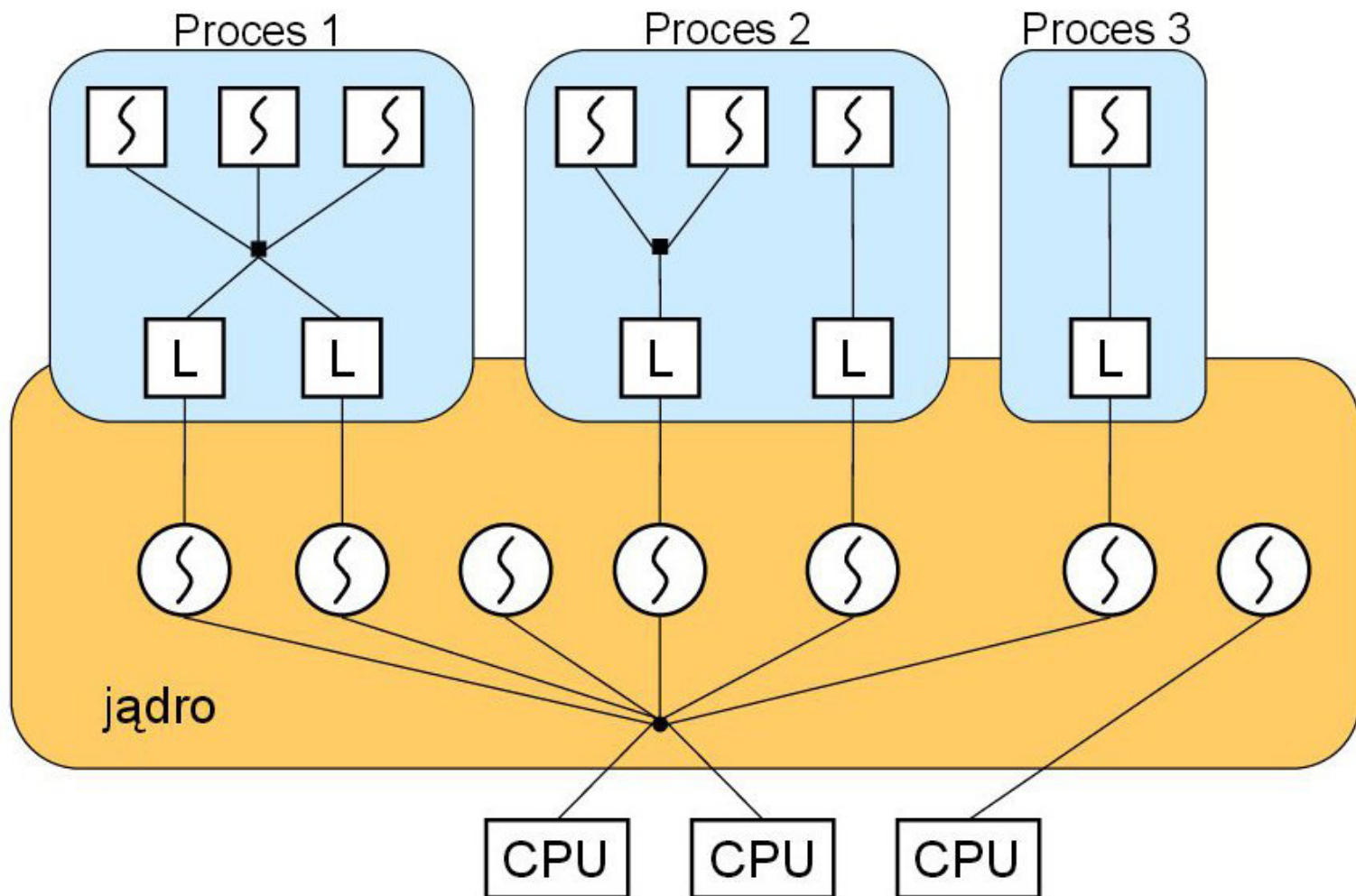
Wątki poziomu użytkownika są najszybsze w przełączaniu, ale jądro systemu nie uwzględnia ich na poziomie przydziału procesora. Tak więc proces o jednym wątku i o 1000 wątków dostają taki sam kwant czasu.

W systemie Solaris istnieje również pośredni poziom wątków, zwanych procesami lekkimi.

Każde zadanie ma przynajmniej jeden proces lekki do którego podłączone są wątki poziomu użytkownika.

Wątki a procesy lekkie

Procesy lekkie



Wątki poziomu jądra

- posiadają małą strukturę danych i stos,
- podlegają planowaniu,
- przełączanie wątków nie wymaga informacji o pamięci,
- przełączanie jest stosunkowo szybkie.

Procesy lekkie

- posiadają blok kontrolny procesu,
- potrzebne informacje o pamięci,
- przełączanie kontekstu dość wolne.

Wątki użytkownika

- posiadają stos i licznik rozkazów,
- szybkie przełączanie, gdyż jądro nie jest angażowane.

Systemy operacyjne

- Wykłady: 18 godzin, B-4, s. 314a
- Laboratoria/ćwiczenia: 18/3 godziny, s. 101, B-4

Systemy operacyjne

Warunki zaliczenia przedmiotu

- Zaliczenie z ćwiczeń laboratoryjnych i audytoryjnych (prowadzący ćwiczenia podaje szczegółowe warunki zaliczenia)
- Egzamin z materiału wykładów - pisemny

Program wykładów

- Zadania i właściwości systemów operacyjnych,
- Jądro systemu, struktura i funkcje,
- Procesy, ich współpraca i współistnienie,
- Organizacja i adresowanie pamięci,
- Organizacja systemu plików,

Program wykładów, c.d.

- Urządzenia wejścia/wyjścia,
- Obsługa sieci i aplikacje sieciowe,
- Systemy rozproszone,
- Systemy czasu rzeczywistego.

Program ćwiczeń i laboratoriów

- System Windows, charakterystyczne procesy, rejestr systemu, konfiguracja,
- System UNIX - analiza i sterowanie procesami, prace na systemie plików, analiza i strojenie systemu,
- Skrypty - zapoznanie się ze składnią skryptów pod różne powłoki, samodzielne rozwiązywanie zadań
- PERL

Strona WWW przedmiotu:

<http://www.metal.agh.edu.pl/~wilk/dydaktyka.html>

Literatura:

- ◆ Notatki z wykładów,
- ◆ Silberschatz A., Galvin P.B.: Podstawy systemów operacyjnych, WNT 2000.
- ◆ Nemeth E., Snyder G., Hein T., Whaley B.: Unix i Linux Przewodnik administratora systemów, Helion 2011
- ◆ Lister A.M., Eager R.D.: Wprowadzenie do systemów operacyjnych, WNT 1994.
- ◆ Stevens R.W.: Programowanie w środowisku systemu UNIX, WNT 2002.
- ◆ Bach M.J.: Budowa systemu operacyjnego UNIX, WNT 1995.
- ◆ Petersen R.: Arkana Linux, Wydawnictwo RM 1997.
- ◆ Inne dostępne książki o systemach DOS, Unix, Windows,
- ◆ Strony WWW.

Abraham Silberschatz
Peter B. Galvin

PODSTAWY SYSTEMÓW OPERACYJNYCH

wydanie piąte

MS-DOS

Windows NT

Linux

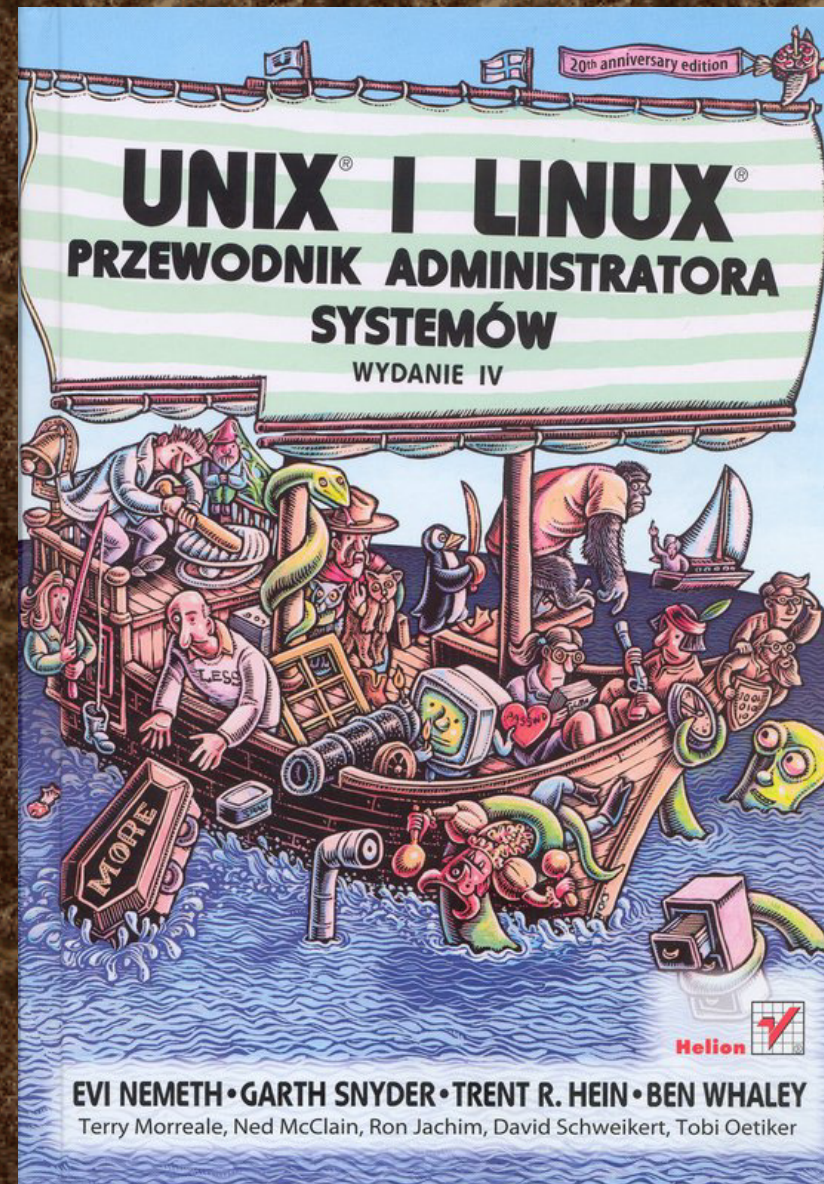
MacOS

UNIX

Solaris

MULTICS

WYDAWNICTWA
NAUKOWO-
TECHNICZNE

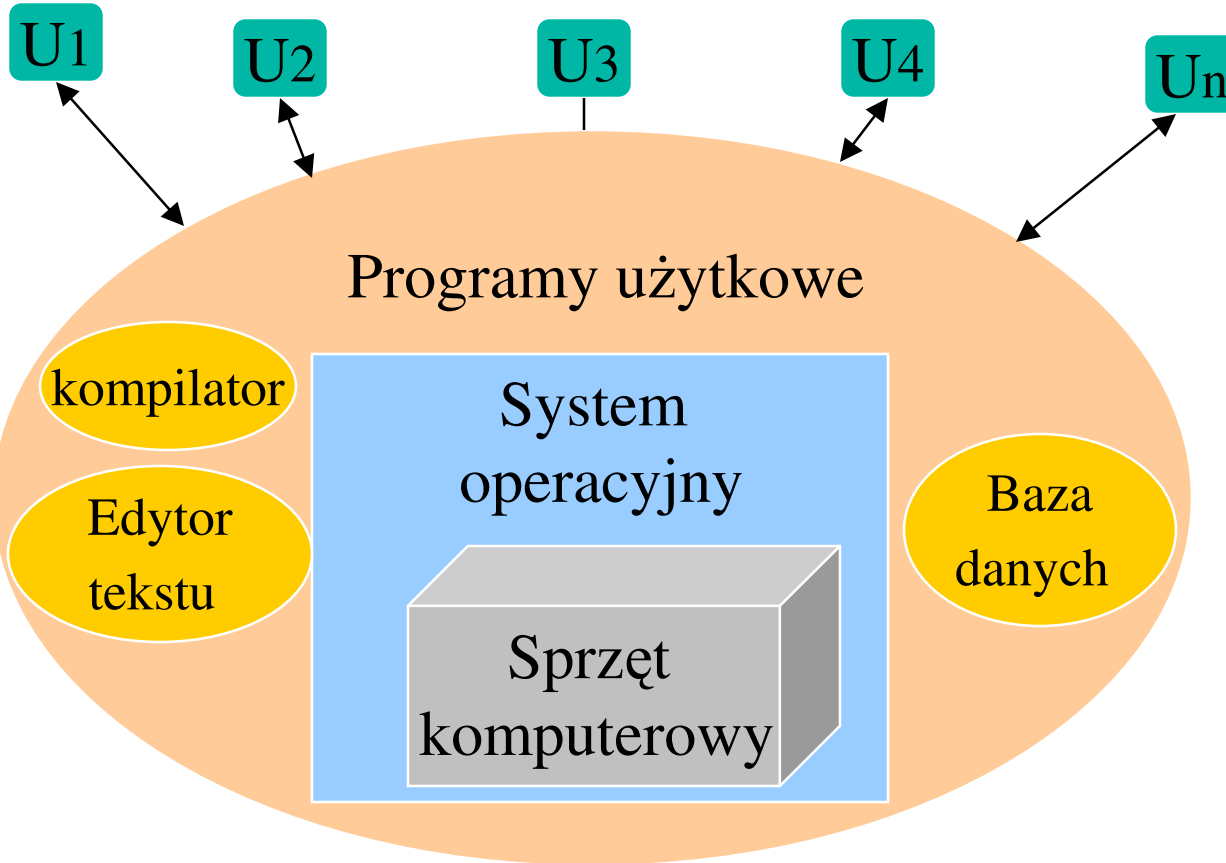


System operacyjny

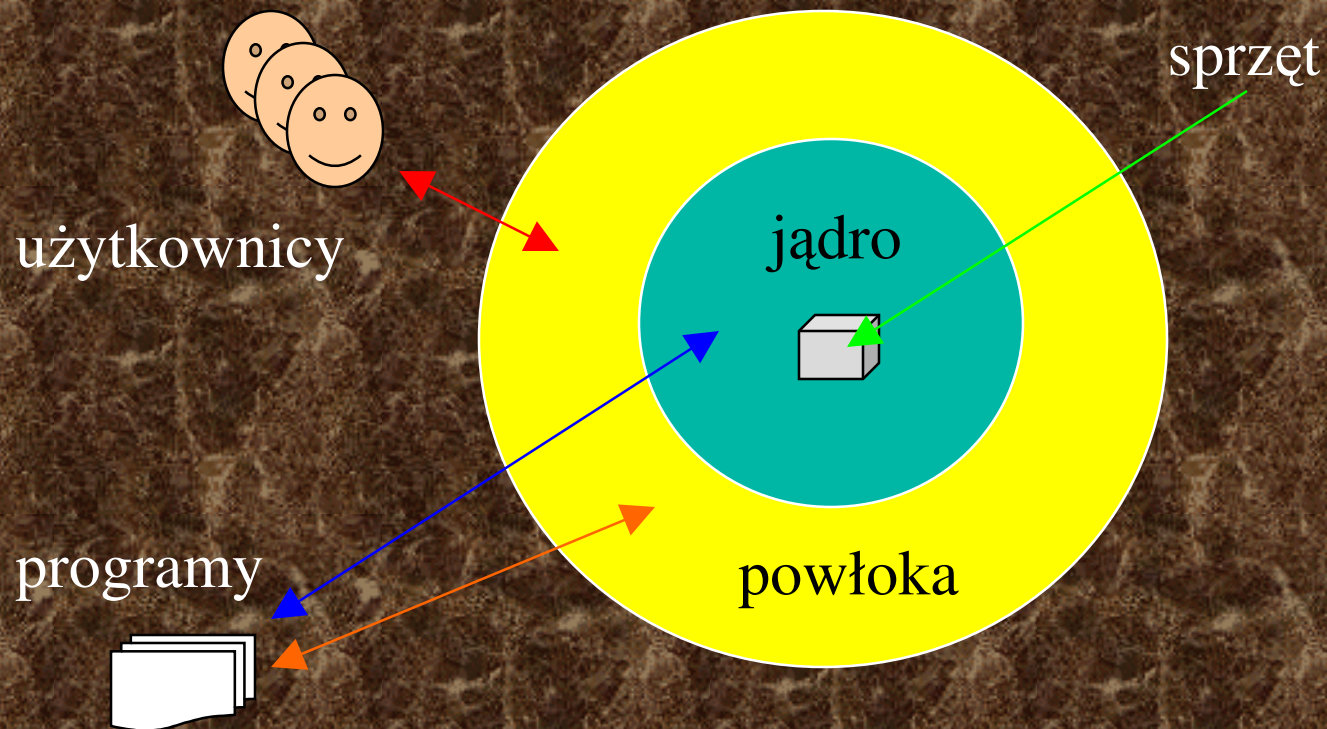
Definicja (wg A. S. i P. B. G.):

System operacyjny jest programem, który działa jako pośrednik pomiędzy użytkownikiem komputera a sprzętem komputerowym. Zadaniem systemu operacyjnego jest tworzenie środowiska w którym użytkownik może wykonywać programy.

Użytkownicy



Warstwowa budowa systemu operacyjnego



Składniki systemu

- jądro - komunikuje się z komputerem przez sterowniki urządzeń i wykonuje kolejowanie zadań, obsługę pamięci
- powłoka - stanowi interpreter poleceń systemu (komunikacja z użytkownikiem)
- programy - polecenia systemowe nie zawarte w jądrze, programy narzędziowe, programy użytkowe

Głównym celem systemu operacyjnego jest to, aby był system komputerowy był wygodny w użyciu.

Drugim celem jest wydajna eksploatacja sprzętu komputerowego.

Pytanie: Czy komputer mógłby się obyć bez systemu operacyjnego?

Odpowiedź: Tak, ale...

- Program użytkowy musiałby zawierać wszelkie procedury obsługi pamięci, urządzeń wejścia i wyjścia, dysków itd.
- Program musiałby sprawdzać czy urządzenia są gotowe, czy nie są aktualnie wykorzystywane przez inne programy,
- Program musiałby znać np. organizację danych na dysku, protokół komunikacji sieciowej itd

- Po przeniesieniu na inny, nawet bardzo podobny komputer, trzeba by w programie zmienić większość procedur obsługi sprzętu
- Istniałyby ogromne problemy ze standaryzacją i kompatybilnością

Przykład takiego oprogramowania użytkowego to na przykład tzw. *firmware* sprzętu elektronicznego zawierającego procesory.

Systemy operacyjne:

- jednozadaniowe (np. DOS)
- wielozadaniowe (np. UNIX)
- niewielozadaniowe ;-) (Windows)

Systemy MS Windows

- na komputery IBM PC
- ciągle rozwijane (ale wymagają coraz silniejszych komputerów)
- interfejs graficzny ułatwia pracę
- systemy wielozadaniowe, ale bez wielodostępu chociaż...
- świadczą pewne usługi poprzez sieć

Systemy UNIX

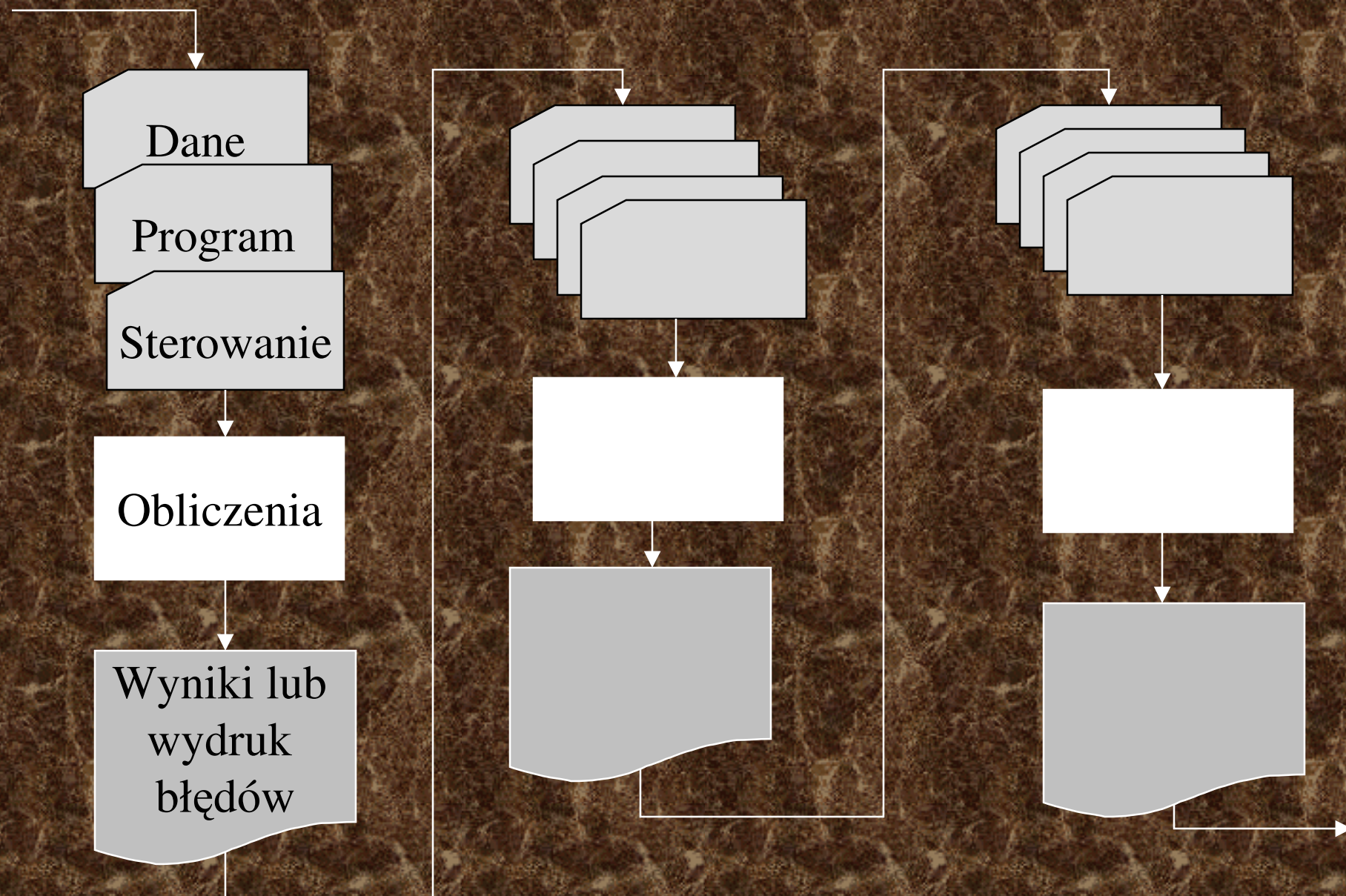
- Na wszystkie typy komputerów, od prostych PC do superkomputerów
- systemy od początku wielodostępne i wielozadaniowe
- łatwe w konfiguracji (pliki tekstowe)
- przeznaczone głównie do pracy zdalnej
- jasno określone prawa użytkowników

Wielozadaniowość i wielodostęp

- **jednozadaniowość** - kolejne zadanie wykonywane po zakończeniu poprzedniego
- **wielozadaniowość** - wykonywanie wielu zadań w „tym samym” czasie.
W rzeczywistości zadania są wykonywane kolejno w przydzielonych im przedziałach czasowych (chyba że jest kilka procesorów)
- **wielodostęp** - w tym samym czasie z jednego komputera korzysta wielu użytkowników

Ewolucja systemów operacyjnych

Systemy wsadowe



Systemy wsadowe

- Wykonywane są kolejno zadania obejmujące wczytywanie programu i danych, obliczenia i wydruk wyników
- następne zadanie wykonywane po zakończeniu poprzedniego
- kolejność zadań ustawia operator
- zadania o podobnych wymaganiach grupowane są w tzw wsad (batch)

Systemy wsadowe

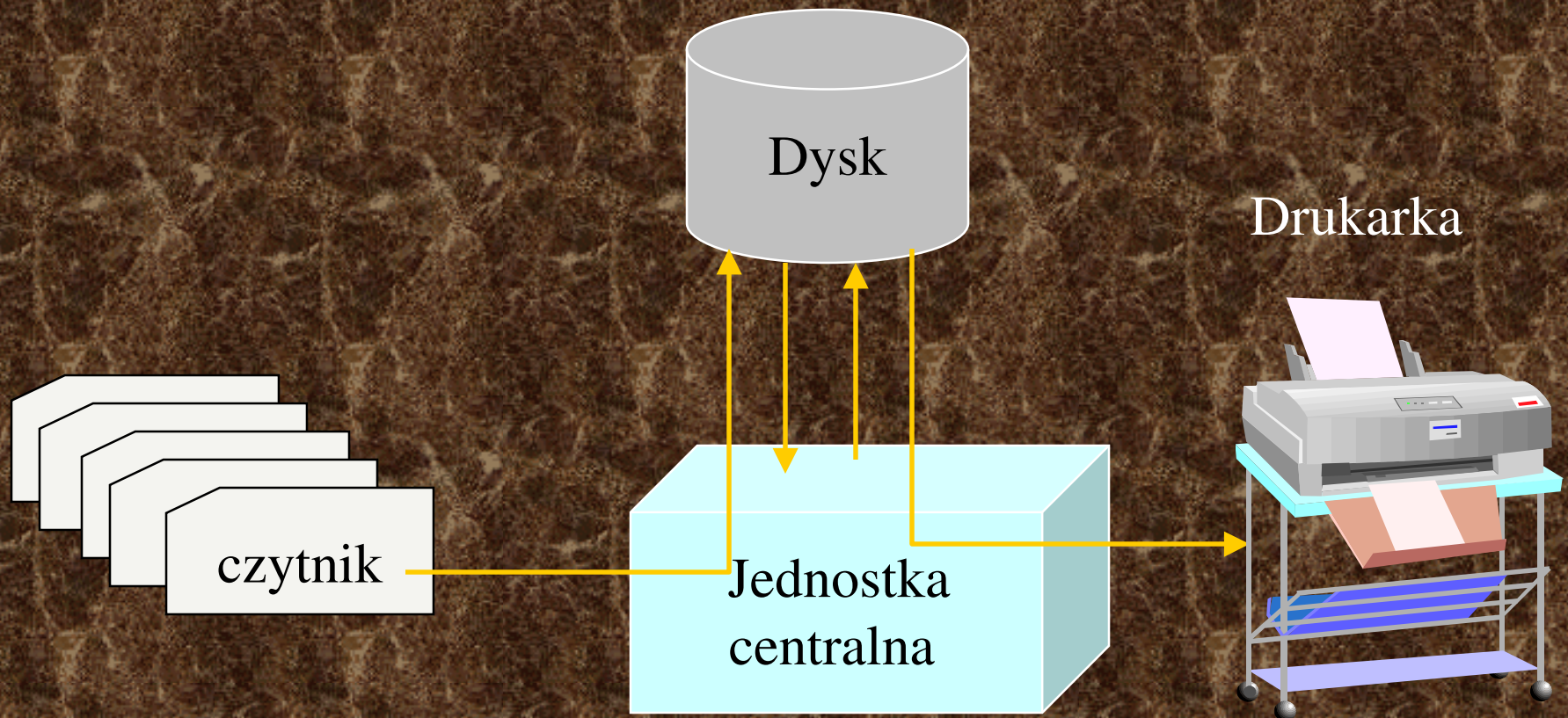
Zalety: bardzo prosty system operacyjny (tylko automatyczne przekazywanie sterowania od jednego zadania do drugiego)

Wady:

- brak nadzoru użytkownika podczas wykonywania zadania (niemożność podjęcia często prostych decyzji),
- tylko jedno zadanie w tym samym czasie,
- małe wykorzystanie jednostki centralnej - bezczynność podczas wczytywania i wydruku danych (obliczenia - kilka tys operacji /s, wczytywanie - kilka, kilkanaście kart na sekundę).

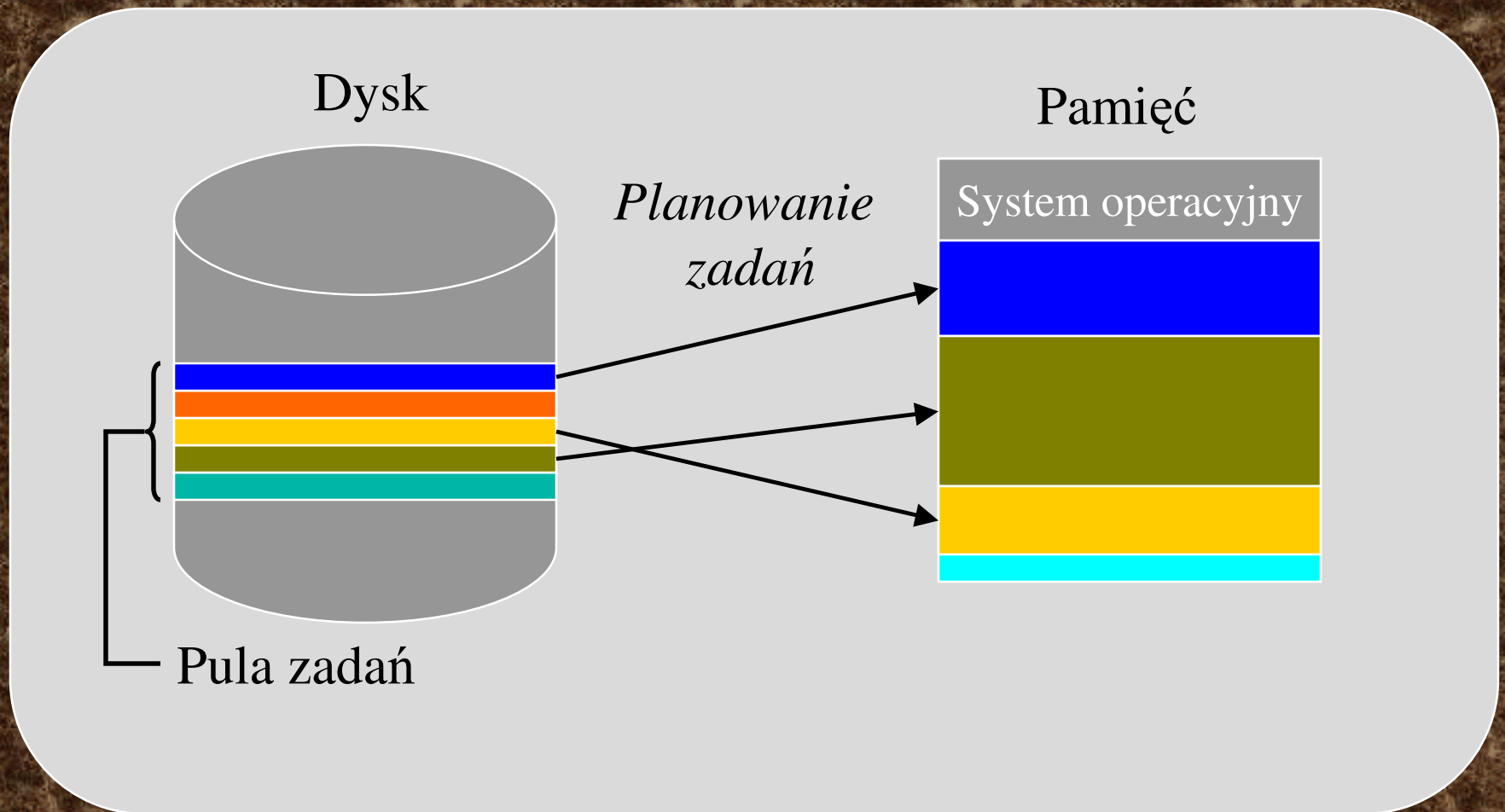
Spool(ing)

simultaneous peripheral operation on-line

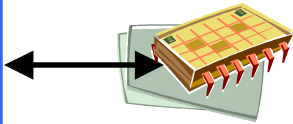
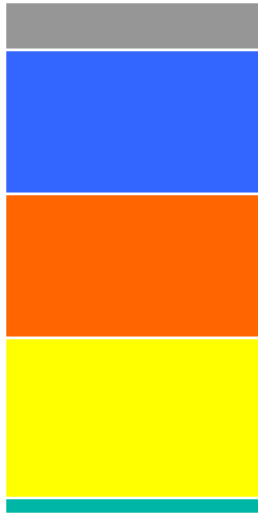


Spooling umożliwia wykonywanie w tym samym czasie operacji wejścia/wyjścia jednego zadania i obliczeń z innych zadań. Kosztem zajęcia niewielkiej części dysku stało się możliwe znacznie lepsze wykorzystanie jednostki centralnej i urządzeń peryferyjnych.

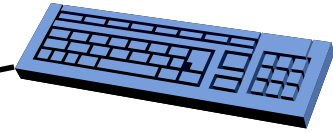
Wieloprogramowy system wsadowy



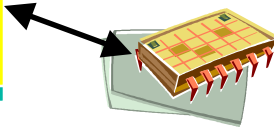
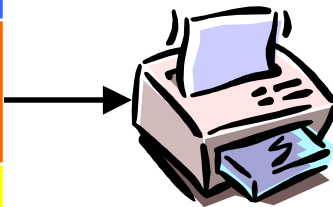
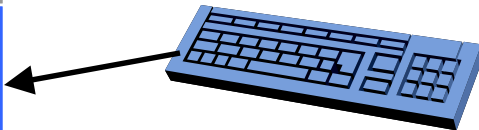
1



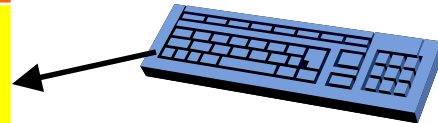
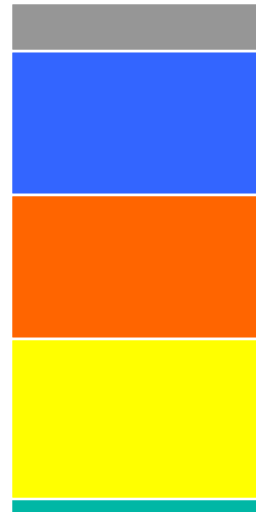
2



3



4



Wieloprogramowy system wsadowy

- W tym samym czasie system operacyjny przechowuje w pamięci kilka zadań
- Gdy aktualnie wykonywane zadanie oczekuje na usługę lub zakończenie operacji (np. I/O), wykonywane jest następne zadanie
- System powraca do wykonywania poprzedniego zadania, gdy zakończyło ono oczekiwanie, a następne są zajęte oczekiwaniem

Wieloprogramowy system wsadowy

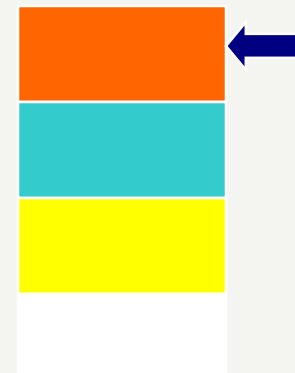
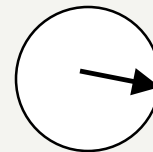
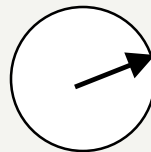
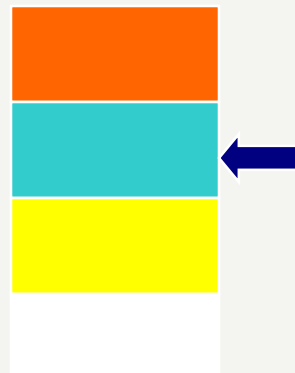
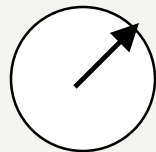
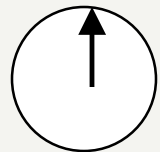
Zalety:

- lepsze wykorzystanie procesora i urządzeń wejścia/wyjścia
- Szybsze wykonywanie puli programów

Wady:

- Wymaga skomplikowanego systemu operacyjnego (planowanie zadań, przydział procesora)
- Zadanie bez operacji I/O może na długo zablokować inne zadania

Systemy z podziałem czasu



Procesor wykonuje na przemian wiele różnych zadań.

Przełączanie następuje tak często, że użytkownicy mogą współdziałać z każdym programem podczas jego wykonywania.

Każdy z użytkowników odnosi wrażenie, że dysponuje własnym komputerem, choć w rzeczywistości jeden komputer jest dzielony pomiędzy wielu użytkowników.

Systemy operacyjne z podziałem czasu, oprócz planowania zadań, muszą mieć mechanizmy zarządzania pamięcią, dostępu do systemu plików, a także mechanizmy synchronizowania zadań i komunikacji między nimi.

Systemy równoległe

- Wiele procesorów współpracuje ze sobą, dzieląc szynę komputera, zegar, pamięć i urządzenia zewnętrzne.
- Jeden zasilacz i obudowa - oszczędność w konstrukcji
- W przypadku awarii jednego procesora system działa dalej - niezawodność
- Stosowane jest wieloprzetwarzanie symetryczne lub asymetryczne

Systemy rozproszone

- Procesory nie współdzielą szyny komputera, zegara, ani pamięci, ale komunikują się ze sobą za pomocą linii komunikacyjnych (np. sieci)
- Procesory mogą się znacznie różnić od siebie

Po co?

- Podział zasobów (przy jednym węźle drukarka, przy drugim duży dysk)
- Przyspieszenie obliczeń (jeżeli można, obliczenia rozbija się na działania współbieżne)
- Niezawodność (w przypadku awarii jednego stanowiska pozostałe mogą kontynuować pracę)
- Komunikacja międzyludzka (poczta, chat, WWW)

Systemy czasu rzeczywistego

- Stosowane tam, gdzie istnieją surowe wymagania na czas wykonania operacji lub przepływu danych
- Są to np. systemy nadzorowania eksperymentu, badań medycznych, sterowanie procesami w przemyśle itd..
- Są to też sterowniki wtrysku w samochodzie, programator pralki, sterowniki w rakietach czy innej broni...
- Rygorystyczne systemy czasu rzeczywistego - minimalizacja czasu operacji, konkurencyjne programy i podział czasu- wykluczone!
- Łagodne systemy - możliwe do aplikacji w standardowych warunkach (zadania czasu rzeczywistego dostają pierwszeństwo przed innymi aż do zakończenia)

Historia popularnych systemów operacyjnych

Część I -MS-DOS

na podstawie:

http://www.republika.pl/p_bogus/software/msdos/msdos.htm

Historia MS-DOS

08.1980

QDOS 0.1

napisany przez Tima Pattersona w **Seattle Computer Products**

12.1980

86-DOS 0.3

napisany w Seattle Computer Products (poprawiona, przemianowana wersja QDOS'a), **MS wykupuje prawa** do systemu na zasadach *non-exclusiv*

02.1981

MS-DOS 1.0

uruchomiony na prototypie IBM PC

07.1981

86-DOS 1.0

MS wykupuje wszystkie prawa do systemu od SCP za \$50 000

Historia MS-DOS, c.d.

08.1981

PC-DOS 1.00

wprowadzony wraz z IBM PC (odpowiada 86-DOS):

- obsługa jednostronnych, 8-sektorowych napędów dysków elastycznych 160kB;
- zawiera: COMMAND.COM - interpreter plików wsadowych oraz poleceń: DIR, REN, DEL, COPY, TYPE, PAUSE, CHKDSK, DATE, TIME, FORMAT, SYS, ponadto EDLIN, DEBUG (assembler/debugger), FILCOM (program do porównywania plików);
- wewnętrzna wersja 1.05 poprawione kilka błędów

Historia MS-DOS, c.d.

03.1983

PC-DOS 2.0

wprowadzony wraz z IBM PC-XT, usprawnienia:

- hierarchiczna struktura katalogów i podkatalogów na wzór UNIX'a,
- koncepcje standardowego urządzenia I/O, strumienia danych (z możliwością przekierowywania jego wejścia/wyjścia), przetwarzania potokowego - również wzorowane na UNIX'ie,
- instalowalne sterowniki urządzeń,
- poprawione zarządzanie pamięcią operacyjną,
- nowy sposób realizacji dostępu systemu do plików poprzez handle (ze względów na kompatybilność pozostawiono również dotychczasowe FCB),

- polecenie PRINT działające w tle i realizujące wydruk kolejki plików (spooler),
- sterownik ekranu i klawiatury ANSI.SYS,
- nowe polecenia systemowe: BACKUP, CD, MD, PATH, RD, RESTORE, TREE, GOTO, IF, ECHO, CLS, FOR, SHIF, LABEL,
- mini-assembler DEBUG,
- ulepszony interpretator BASICa,
- obsługa twardych dysków (do 10MB),
- obsługa dwustronnych, 9-sektorowych napędów dysków elastycznych 360kB

Historia MS-DOS, c.d.

08.1984

PC-DOS 3.0

wprowadzone wraz z IBM PC-AT, usprawnienia:

- 16-bitowy FAT,
- możliwość uruchamiania programów przez podanie ścieżki dostępu,
- obsługa dysku stałego o pojemności 20 MB,
- obsługa napędów 5.25" 1.2MB,
- możliwość symulacji dysku w pamięci operacyjnej (RAMDISK),
- tzw. polecenia narodowe (COUNTRY, KEYBxx i SELECT),
- nowe polecenia: ATTRIB, GRAFTABL, LABEL i SHARE,
- kilka dodatkowych programów narzędziowych,
- obsługa sieci IBM.

Historia MS-DOS, c.d.

03.1985

PC-DOS 3.10

poprawione błędy, dodana generalna obsługa sieci LAN i współużytkowania plików, nowe polecenia: JOIN i SUBST, zmienione polecenia: LABEL i TREE.

01.1986

PC-DOS 3.20

- obsługa napędów 3.5" 720kB,
- obsługa laptopów IBM PC Convertible,
- zabezpieczenie przed przypadkowym formatowaniem dysku stałego,
- możliwość kontroli urządzeń niestandardowych i logicznych,
- polecenia systemowe REPLACE i XCOPY,
- ulepszone polecenia: COMMAND, ATTRIB, FORMAT, GRAPHICS, SELECT, SHELL,
- wersje znacjonalizowane MS- i PC-DOS

Historia MS-DOS, c.d.

04.1987

PC-DOS 3.30

wprowadzony wraz z modelem IBM PS/2, usprawnienia:

- obsługa napędów 3.5" 1.44MB,
- obsługa kilku partycji (do 24 dysków logicznych),
- przełączanie stron kodowych (CHCP, NLSFUNS),
- usprawniona obsługa języków narodowych (KEYB, KEYBOARD.SYS, zrezygnowano z KEYBxx),
- nowe polecenia systemowe: APPEND, CALL, FASTOPEN,
- programy sterujące: DISPLAY.SYS i PRINTER.SYS,
- zmodyfikowane polecenia: ATTRIB, BACKUP, DATE, TIME, FDISK, RESTORE, XCOPY,
- obsługa zegara CMOS w AT

Historia MS-DOS, c.d.

08.1988

PC-DOS 4.00

- obsługa dysków większych niż 32MB,
- obsługa EMS (Expanded Memory), realizowana przez programy XMA2EMS.SYS i XMAEM.SYS,
- programy VDISK.SYS i DEBUG.COM,
- rozszerzona obsługa sieci,
- „graficzne” środowisko systemu - DOSSHELL,
- nowe polecenia: MEM, INSTALL, SWITCHES,
- zmodyfikowane polecenia: APPEND, CHKDSK, ERASE, FASTOPEN, FDISK, FORMAT, GRAPHICS, MODE, REM, REPLACE SELECT, SYS.

Historia MS-DOS, c.d.

06.1991

MS-DOS 5.00

- obsługa XMS (Extended Memory) i HMA (High Memory Area) poprzez sterownik HIMEM.SYS i EMM386.EXE,
- możliwość ładowania sterowników (device drivers) i programów rezydentnych (TSR - Terminate (and) Stay Resident) do UMB (Upper Memory Blocks) - obszaru powyżej 640kB,
- obsługa do 8 twardych dysków,
- dysk logiczny może być większy niż 32MB,
- język interpretator QBASIC,
- pełnoekranowy edytor tekstu EDIT,
- system informacji pomocniczych HELP i opcja "/" w każdym poleceniu,
- zwiększenie bezpieczeństwa systemu (UNFORMAT, UNDELETE, MIRRIR),

Historia MS-DOS, c.d.

03.1993

MS-DOS 6.00

- kompresja dysku (DOUBLESPEACE),
- wybór konfiguracji przy starcie,
- optymalizacja wykorzystania pamięci (MEMMAKER),
- przerobiony program SMARTDRIVE (cache także CD-ROM'u),
- przesyłanie danych złączem szeregowym/równoległym (INTERLINK),

08.1995

MS-DOS 7.00

- element Windows 95

Historia popularnych systemów operacyjnych

Część II -MS-WINDOWS

na podstawie:

http://www.republika.pl/p_bogus/software/win/win16.htm

Historia MS-WINDOWS

06 1985 **Microsoft Windows 1.0** (egzemplarze testowe).

11 1987 **Windows 2.03** (DOS 3.0, min. 512 kB RAM)

05 1990 **Windows 3.0** (DOS 3.1, min 640+256 kB RAM)

04 1992 **Windows 3.1** (DOS 5.0, min 640 +2048 kB RAM)

11 1993 **Windows for Workgroups 3.11**

04 1994 **Windows NT 3.1**

08 1995 **Windows 95**

1997 **Windows NT 4.0**

06 1998 **Windows 98**

Historia MS-WINDOWS

02 2000 **Windows 2000**

09 2000 **Windows ME**

10 2001 **Windows XP** (300 MHz, 128 MB RAM, 1,5 GB na HDD)

05 2003 **Windows 2003 server**

2005 Windows Vista, wersja Beta (do tej pory Longhorn)

01 2007 Windows Vista, wersja rynkowa

05 2007 Windows 7, wersja Milestone 1

10 2009 Windows 7, wersja finalna

10 2012 Windows 8 (wersja 64b: 1GHz, 2GB RAM, 16 GB na HDD)

10 2014 Windows Server 2012 R2

07 2015 Windows 10

Historia popularnych systemów operacyjnych

Część III -UNIX

na podstawie:

[http://pl.wikipedia.org/wiki/Historia_systemu_
operacyjnego_Unix/Kalendarium](http://pl.wikipedia.org/wiki/Historia_systemu_operacyjnego_Unix/Kalendarium)



Dennis
Ritchie



Ken
Thompson



Linus
Torvalds

Historia systemu UNIX

1966 - rozpoczęcie prac na **Multics**, zespół naukowców z Bell Labs, MIT i General Electric, wśród tych pierwszych Ken Thompson

1969 - AT&T porzuca projekt Multicsa

pierwsza wersja **Unix** napisana w assemblerze w ośrodku Bell Labs firmy AT&T (stan New Jersey, USA) przez Dennisa Ritchie i Kena Thompsona

1971 - port Unix-a na nowe komputery PDP-11 firmy DEC

1972 - **Unix Second Edition**, pojawia się potok (ang. pipe)

1973 - **Unix Fourth Edition**, pierwsza napisana w C, co zaowocowało przenośnością systemu

1975 - **Unix Sixth Edition** - inaczej Wersja Szósta - rozprowadzana nieodpłatnie w uczelniach dla zastosowań akademickich, dała początek rozszerzeniom BSD

Historia systemu UNIX, c.d.

1976 - John Lions na Uniwersytecie Nowej Południowej Walii pisze komentarz do kodu Unixa. tzw. **Lions Book** zawiera pełny kod v.6

1977 - pod koniec roku ukazują się pierwsze taśmy **1BSD**,
rozszerzenia Unixa z Uniwersytetu Kalifornijskiego Berkeley

1978 - w połowie roku ukazują się taśmy **2BSD**

1979 wydanie **3BSD**

Unix 7th edition - the last true Unix - zawiera C, UUCP i powłokę Bourne'a.

Przeniesiony na nowe komputery VAX kernel liczył 40 Kb.

1980 - w Santa Cruz Operation (SCO) na licencji od AT&T i na zamówienie MS powstaje **Xenix**, pierwszy Unix dla PC

1982 - Silicon Graphics prezentuje **IRIX**

Historia systemu UNIX, c.d.

1983 - rozpoczęcie projektu **GNU** (GNU's Not Unix)

System V - pierwsza komercyjna wersja Unixa AT&T

1984 - pojawia się idea open system

Sun udostępnia specyfikację **NFS**

4.2BSD - wprowadzenie rozszerzeń TCP/IP

DEC wydaje pierwszą wersję **Ultrix**

1986 - **4.3BSD** - wprowadzenie named, daemona DNS

Hewlett-Packard prezentuje **HP-UX**, własną wersję Unixa

1988 - publikacja standardu POSIX.1. Powstają konkurencyjne UNIX International (związek AT&T i Sun Microsystems) i Open Software Foundation (OSF, pozostali sprzedawcy Unixów), organizacje standaryzujące Unixa

Historia systemu UNIX, c.d.

1989 - **SVR4 - 4** wydanie System V; unifikacja Systemu V, BSD oraz Xenixa

SCO UNIX przedstawia **System V/386**, pierwszy Unix spoza AT&T, który ma prawo być sprzedawany pod tą nazwą

1990 - luty, IBM wydaje **AIX** (Advanced Interactive eXecutive)

1991 - **Solaris 2** (oparty na SysV) wydany przez Sun Microsystems
wrzesień powstanie jądra **Linux**, wersja 0.01

1992 - 22 grudnia, Novell kupuje UNIX System Laboratories od AT&T

1993 - **4.4BSD**

Novell przekazuje prawa do marki Unix i Single UNIX Specification dla X/Open

20 kwietnia - ukazuje się **NetBSD 0.8**

Historia systemu UNIX, c.d.

grudzień - premiera **FreeBSD 1.0** (na bazie 4.3BSD Net/2)

koniec roku - **SVR4.2MP**, ostatnia wersja Systemu V z Unix
System Laboratories (USL)

1994 - **4.4BSD-Lite**

powstają firmy Red Hat Linux oraz Caldera

13 marca - **Linux 1.0**

czerwiec - ugoda między Novellem a UCB w sprawie kodu BSD.

26 października 1994 - **wydanie NetBSD 1.0**

listopad - **FreeBSD 2.0** (oparte na kodzie 4.4BSD Lite)

1995 - Novell sprzedaje SCO prawa do UnixWare

7 marca - **Linux 1.2**

Historia systemu UNIX, c.d.

1996 - powstaje Open Group po zjednoczeniu OSF z X/Open.

9 czerwca - **Linux 2.0**

1997 - druga wersja Single UNIX Specification, rozszerzona o tryb czasu rzeczywistego, wątki, 64-bitowość. Dostępna na www.

1999 - 26 stycznia - ukazuje się **Linux 2.2**

2000 - 2 sierpnia - Caldera, dystrybutor Linuxa przejmuje SCO

2001 - trzecia wersja Single UNIX Specification, powstaje IEEE POSIX

4 stycznia - Linux 2.4

2003 - 19 stycznia - prezentacja FreeBSD 5.0

2006 – IRIX 6.5.30

2008 – Open Solaris 2008.05

2009 – FreeBSD 8.0

2010 – sierpień – informacja o zaprzestaniu rozwoju systemu OpenSolaris ze względu na politykę firmy Oracle.

Historia systemu UNIX, c.d.

1996 - 9 czerwca - **Linux 2.0**

1999 - 26 stycznia - ukazuje się **Linux 2.2**

2000 - 2 sierpnia - Caldera, dystrybutor Linuxa przejmuje SCO

2001 - 4 stycznia - Linux 2.4

2003 - 19 stycznia - prezentacja FreeBSD 5.0

2006 – IRIX 6.5.30

2008 – Open Solaris 2008.05

Android 1.0 (Apple Pie)

2009 – FreeBSD 8.0

2010 – sierpień – informacja o zaprzestaniu rozwoju systemu OpenSolaris ze względu na politykę firmy Oracle.

03 2016 - FreeBSD 10.3

10 2015 – Oracle Solaris 11.3

08 2016 - Android 7.0 (Nougat)

05 2016 - Linux 4.6

Systemy rozproszone

Wg Wikipedii:

System rozproszony to zbiór niezależnych urządzeń (komputerów) połączonych w jedną, spójną logicznie całość. Połączenie najczęściej realizowane jest przez sieć komputerową. Urządzenia są wyposażone w oprogramowanie umożliwiające współdzielenie zasobów systemowych.

Jedną z podstawowych cech systemu rozproszonego jest jego *transparentność*, inaczej *przezroczystość*, która stwarza na użytkownikach systemu rozproszonego wrażenie pojedynczego i zintegrowanego systemu.



Systemy rozproszone - cechy

współdzielenie zasobów - wielu użytkowników systemu może korzystać z danego zasobu (np. drukarek, plików, usług, itp.)

otwartość - podatność na rozszerzenia, możliwość rozbudowy systemu zarówno pod względem sprzętowym, jak i oprogramowania

współbieżność - zdolność do przetwarzania wielu zadań jednocześnie

skalowalność - zachowanie podobnej wydajności systemu przy zwiększeniu skali systemu (np. liczby procesów, komputerów, itp.)

odporność na błędy - zdolność działania systemu mimo pojawiania się błędów (np. poprzez utrzymywanie nadmiarowego sprzętu)

transparentność, przeźroczystość - postrzeganie systemu przez użytkownika jako całości, a nie poszczególnych składowych.

Otwartość

- usługi muszą być zgodne ze standardowymi regułami opisującymi ich składnię i semantykę (np. protokoły sieciowe)
- specyfikacja interfejsu musi być kompletna i neutralna

Programy od różnych dostawców MUSZĄ współpracować ze sobą, o ile spełniają warunek zgodności interfejsów

Przenośność – aplikacja stworzona dla jednego systemu może być uruchomiona w innym bez potrzeby dokonania jakichkolwiek zmian.

Przezroczystość

- a) dostępu
- b) położenia
- c) wędrówki
- d) przemieszczania
- e) zwelokrotniania
- f) współbieżności
- g) awarii
- h) trwałości

Ad a) – poprzez ujednolicenie metod dostępu do danych
i ukrywanie różnic w ich reprezentacji



Przezroczystość

Ad b) – użytkownicy nie mogą określić położenia zasobu,
np. na podstawie jego nazwy

Ad c) – można przenosić zasoby między serwerami bez zmiany
odwoływania się do nich

Ad d) – możliwość przenoszenia zasobów nawet podczas ich
używania

Ad e) – użytkownik nie zauważa faktu zwielokrotniania zasobów

Ad f) – możliwość współbieżnego przetwarzania nie
powodującego utraty spójności

Ad g) – niezauważalne zastępowanie uszkodzonych węzłów

Ad h) – maskowanie sposobu przechowywania zasobu (pamięć
lub dysk)

Skalowalność

- pod względem rozmiaru (możliwość dodawania nowych zasobów i użytkowników)
- geograficzna (rozzrucenie zasobów i użytkowników po całym świecie)
- administracyjna (skuteczna, mimo że rozrzucona administracja systemem)

Skalowalność

1. Ukrywanie opóźnień komunikacji

- komunikacja asynchroniczna
- część obliczeń po stronie klienta

2. Rozpraszanie (np DNS)

3. Zwielokrotnianie

- równoważenie obciążenia
- zwiększenie dostępności
- zwiększenie niezawodności
- caching

Problem spójności danych

Realizacje sprzętowe

Systemy

- wieloprocesory (pamięć dzielona)
- multikomputery (pamięć odrębna)

Architektura

- szyna
- przełącznik

Systemy homogeniczne

Sieci systemowe – grupa komputerów homogenicznych połączonych siecią

- architektura połączeń – szyna lub przełącznik
- topologia połączeń – siatki i hiperkostki

Realizacje:

- procesory o masywnej równoległości (specjalna sieć)
- klastry, grupy stacji roboczych (sieć standardowa)

Oprogramowanie

1. Systemy operacyjne dla komp. rozproszonych:
 - a) ściśle powiązane (zarządzanie wszystkimi globalnymi zasobami przez system) – w wieloprocessorach, komp. homogenicznych
 - b) luźno powiązane (zbiór współpracujących komputerów z lokalnymi S.O.) – sieciowe systemy operacyjne
2. Oprogramowanie warstwy pośredniej

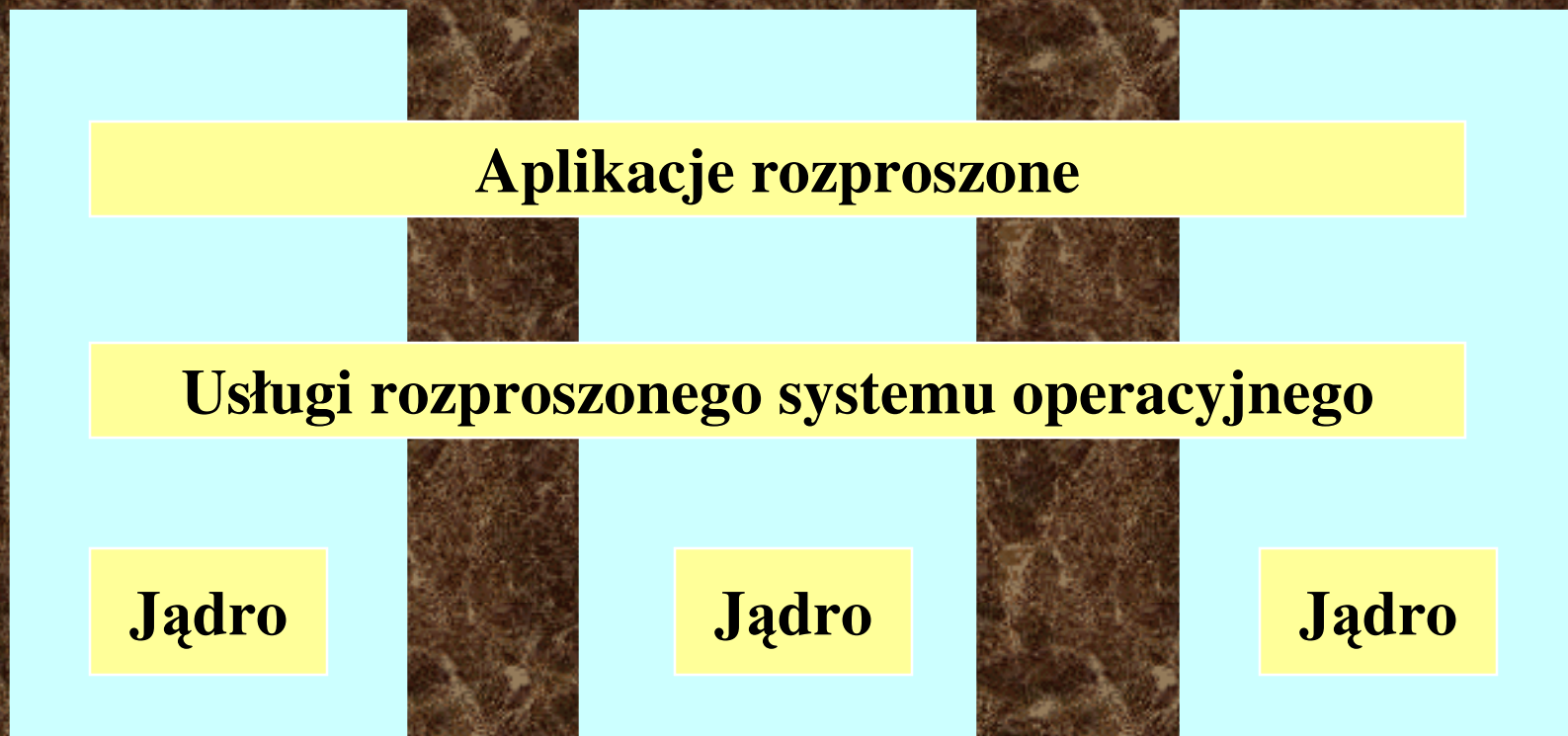
Oprogramowanie - cele

Ad 1a: Ukrywanie zasobów sprzętowych i zarządzanie nimi

Ad 1b: Oferowanie lokalnych usług klientom zdalnym

Ad 2: Zapewnianie przezroczystości rozproszenia

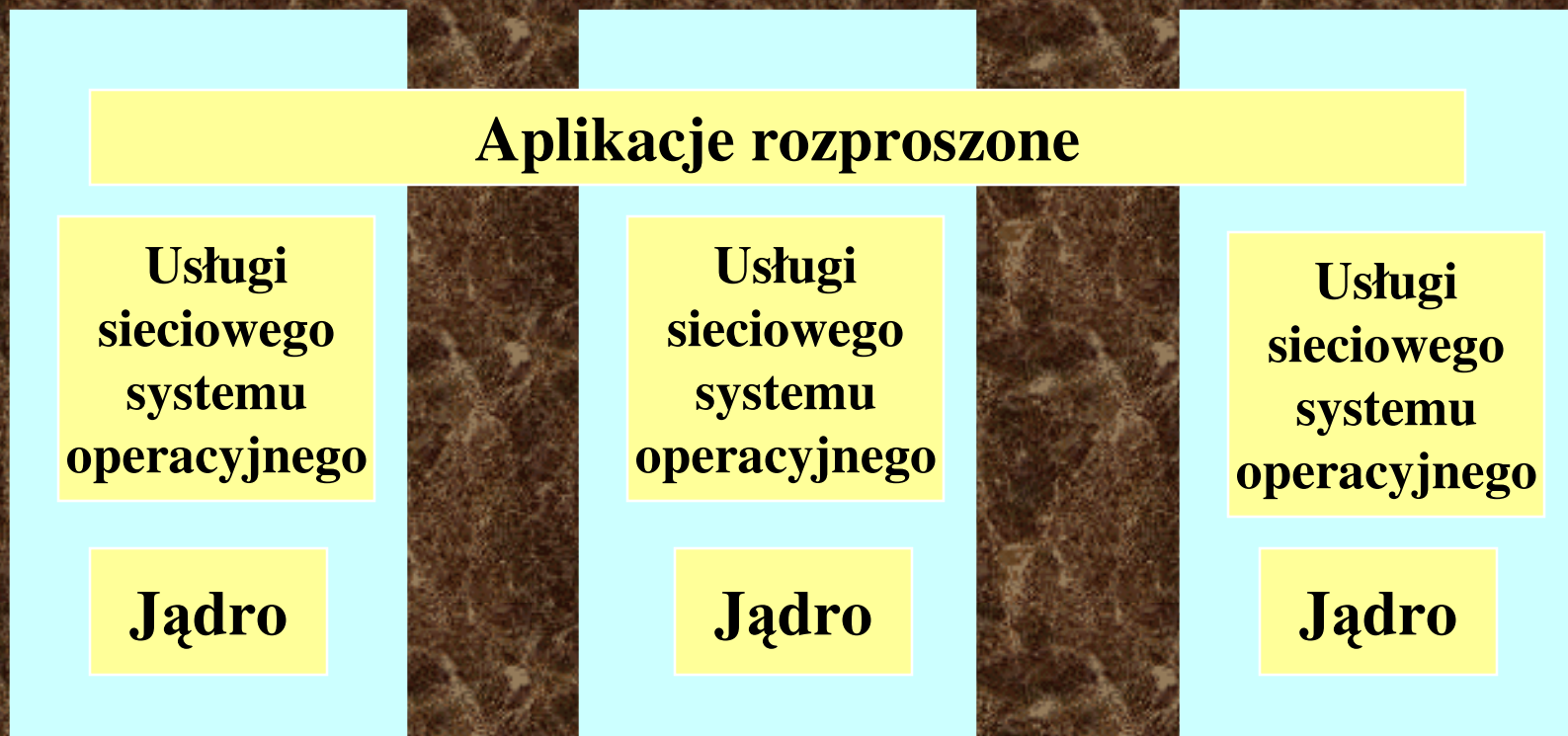
Wielokomputerowy S.O.



Rozproszona pamięć dzielona

- Stronicowana rozproszona pamięć dzielona jako forma komunikacji
- Problemy z efektywnością:
 - zwielokrotniania stron do odczytu,
 - zwielokrotnianie wszystkich stron,
 - rezygnacja ze ścisłej spójności
 - fałszywe dzielenie (gdy 2 procesy odwołują się do różnych zmiennych, ale na tej samej stronie)

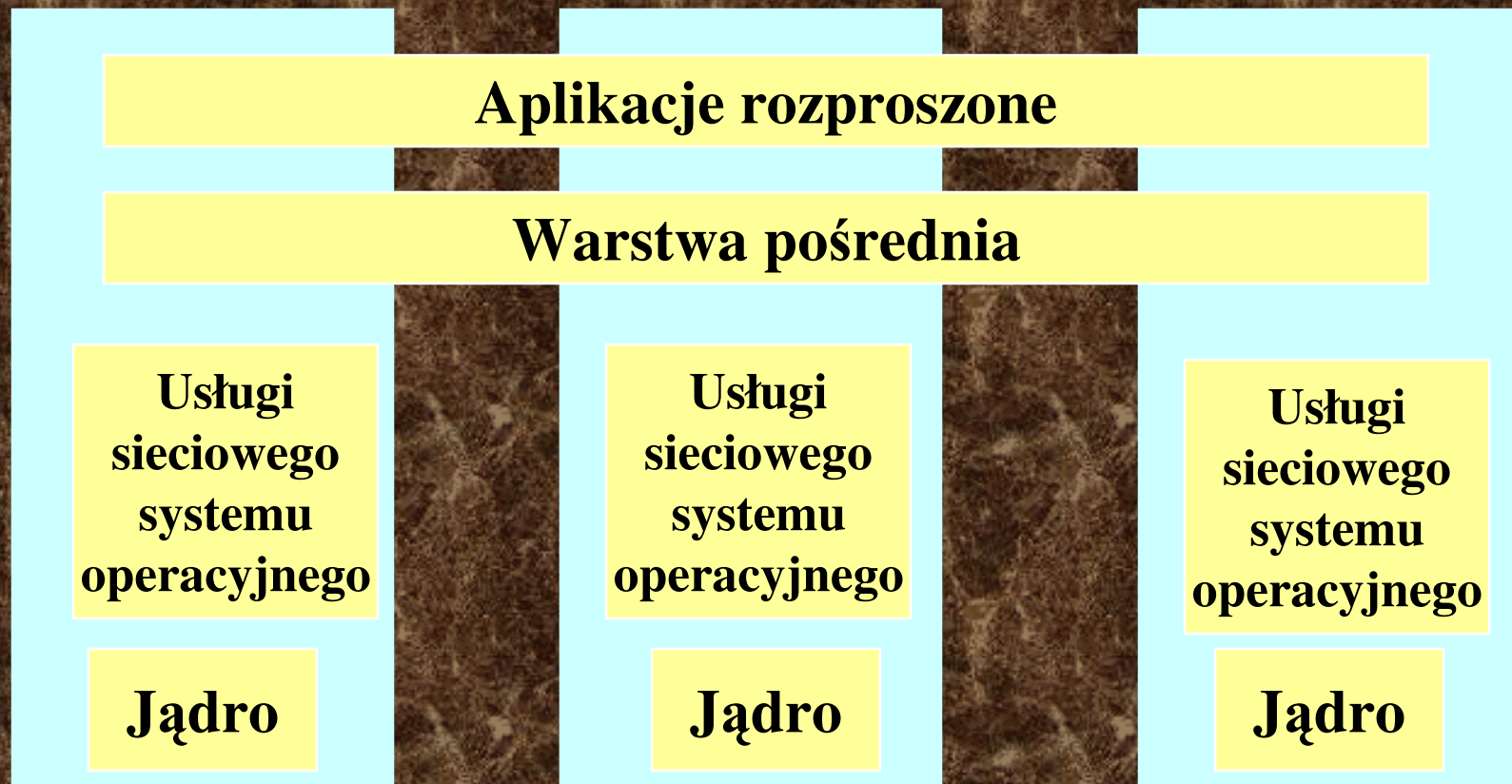
Sieciowy S.O.



Usługi sieciowego S.O.

- Praca zdalna (np. rlogin)
- Kopiowanie plików (np. rcp)
- Sieciowy system plików
 - serwer
 - klient

Oprogramowanie warstwy pośredniej



Wzorce, założenia do warstwy pośredniej

- Wszystko jest plikiem (z Unixa) - komunikacja jako zapis/odczyt pliku
- Zdalne wywołania procedur (RPC) – procedury zdalne jak lokalne (ukrywanie komunikacji)
- Obiekty rozproszone – obiekt na jednej maszynie, interfejs do niego na wielu
- Model dokumentów rozproszonych (WWW)

Usługi warstwy pośredniej

- Komunikacja – RPC, zdalne obiekty, przezroczysty dostęp do rozproszonych plików, baz danych, dokumenty WWW
- Nazewnictwo – lokalizacja zasobów - skalowalność
- Trwałość – pliki, bazy danych, rozproszona pamięć dzielona
- Transakcje rozproszone – atomowość, dane na wielu maszynach, maskowanie awarii
- Bezpieczeństwo

Otwartość warstwy pośredniej

- Nadbudowa nad systemem – uniezależnienie od systemu
- Zależność aplikacji od warstwy pośredniej
- Niekompletność interfejsów warstwy pośredniej – konieczność odwoływania się bezpośrednio do systemu
- Zgodność warstwy pośredniej ze standardem, ale nieprzenośność aplikacji

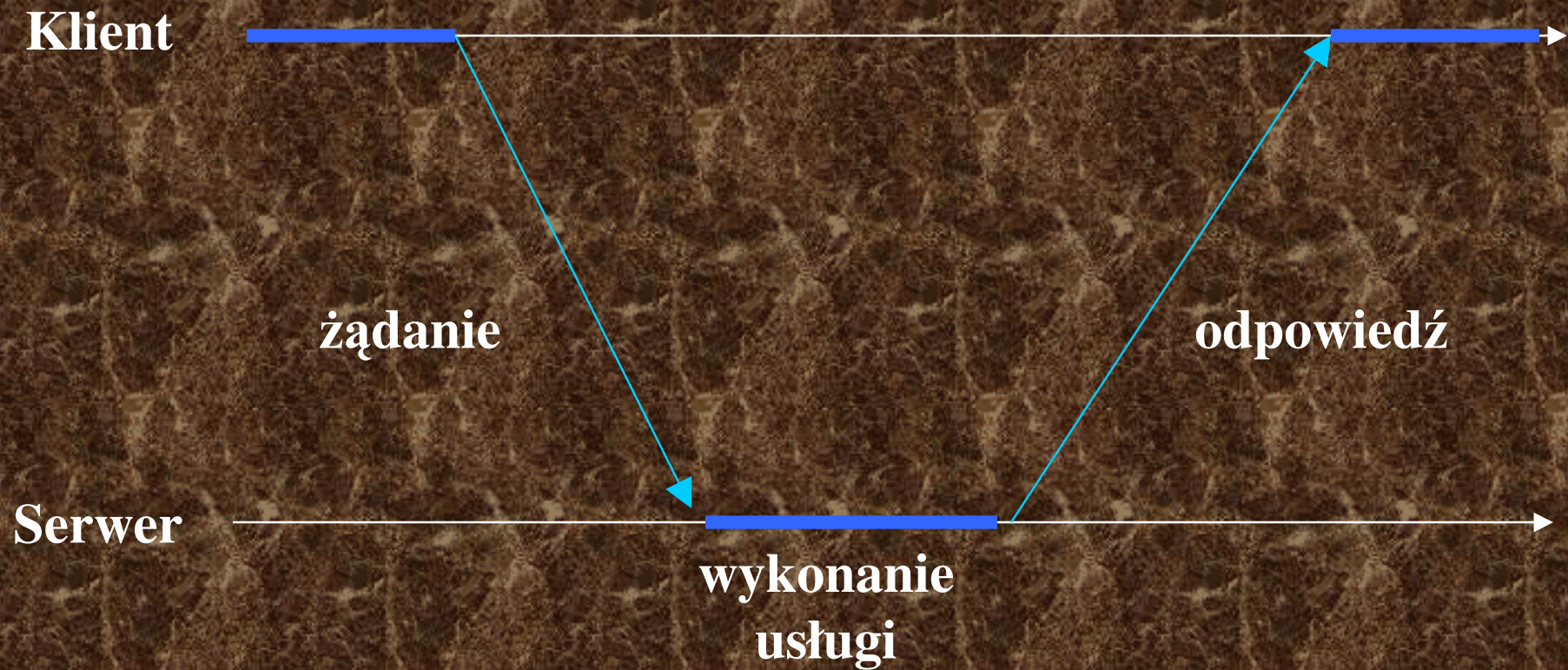
Przykłady warstw pośrednich

- gniazda (ang. *sockets*)
- RPC (*Remote Procedure Call*)
- DCE (*Distributed Computing Environment*)
- CORBA (*Common Object Request Broker Architecture*)
- DCOM (*Distributed Component Object Model*)
- RMI (*Remote Method Invocation*)

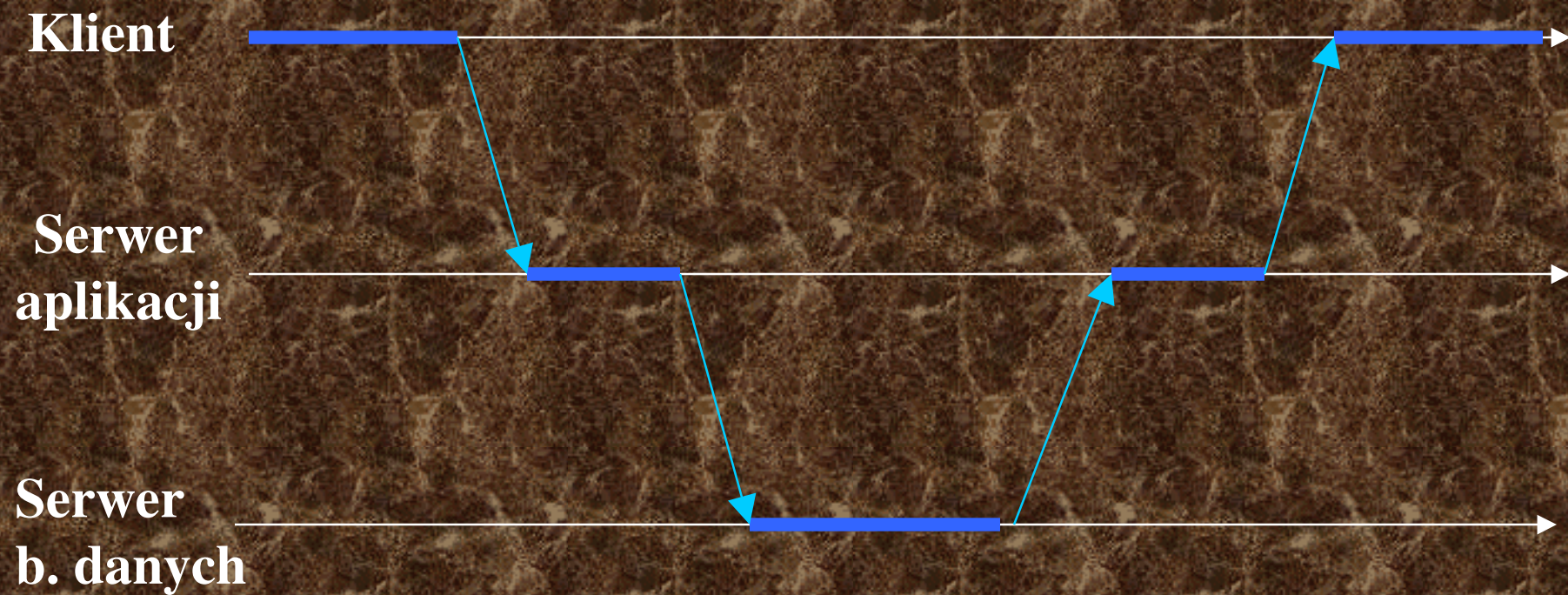
Porównanie systemów

	Rozproszony S.O.		Sieciowy system operacyjny	Warstwa pośrednia
	Wieloproc.	Wielokomp.		
Przezroczystość	b. duża	duża	mała	duża
Jeden SO?	tak	tak	nie	nie
Kopie S.O.	1	n	n	n
Komunikacja	pamięć dzielona	komunikaty	pliki	zależna od modelu
Zarządzanie zasobami	globalne, centralne	globalne, rozproszone	lokalne	lokalne
Skalowalność	nie	umiarkowana	tak	zmienna
Otwartość	zamknięty	zamknięty	otwarty	otwarty

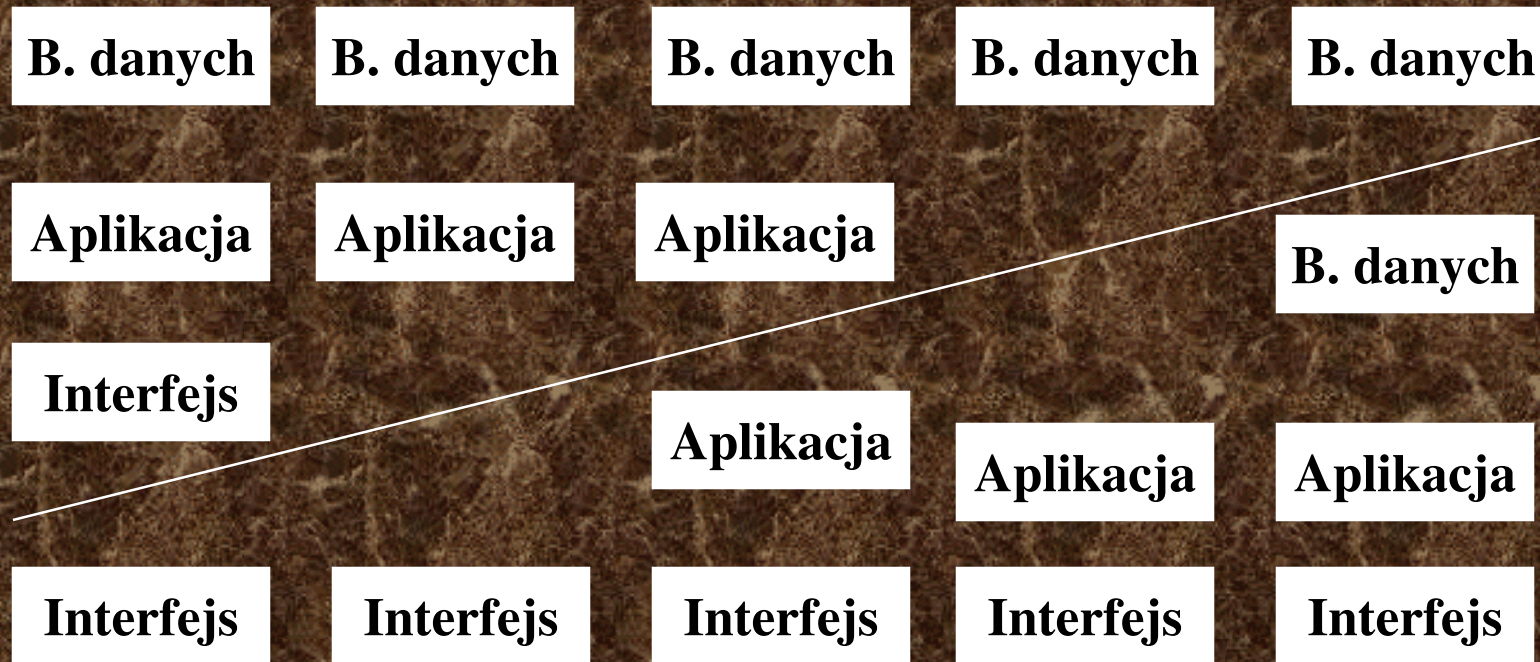
Model klient-serwer



Model trójwarstwowy



S e r w e r



K l i e n t



Systemy czasu rzeczywistego

Definicja IEEE:

System czasu rzeczywistego (real time) to system, którego poprawność działania zależy nie tylko od poprawności logicznych rezultatów, lecz również od czasu, w jakim te rezultaty są osiągnane (czasu reakcji).

Systemy czasu rzeczywistego

- Tryb przetwarzania w czasie rzeczywistym jest takim trybem, w którym programy przetwarzające dane napływające z zewnątrz są zawsze gotowe, a wynik ich działania jest dostępny nie później niż po zadanym czasie. Moment nadejścia kolejnych danych może być losowy (asynchroniczny) lub ściśle określony (synchroniczny)*
- System czasu rzeczywistego jest systemem interaktywnym, który utrzymuje ciągły związek z asynchronicznym środowiskiem, np. środowiskiem, które zmienia się bez względu na system, w sposób niezależny *
- Oprogramowanie czasu rzeczywistego odnosi się do systemu lub trybu działania, w którym przetwarzanie jest przeprowadzane na bieżąco, w czasie wystąpienia zewnętrznego zdarzenia, w celu użycia rezultatów przetwarzania do kontrolowania lub monitorowania zewnętrznego procesu *

Systemy czasu rzeczywistego

- System czasu rzeczywistego odpowiada w sposób przewidywalny (w określonym czasie) na bodźce zewnętrzne napływające w sposób nieprzewidywalny *
- System mikrokomputerowy działa w czasie rzeczywistym, jeżeli wypracowane przez ten system decyzje są realizowane w tempie obsługiwanego procesu. Inaczej mówiąc, system działa w czasie rzeczywistym, jeżeli czas reakcji systemu jest niezauważalny przez proces (decyzja jest wypracowana we właściwym czasie) **

* - Lal K., Rak T., Orkisz K : “RTLinux – system czasu rzeczywistego”, HELION, 2003.

** - Plaza R., Wróbel E.: „Systemy czasu rzeczywistego”, Wydawnictwo Naukowo – Techniczne, Warszawa 1988.

Systemy czasu rzeczywistego - przykłady

- Sekwencja awaryjnego wyłączania silnika rakietowego
- System zbierania danych (np. informacji o opadzie deszczu).
- System kontrolny ABS w samochodzie.
- System dostarczania paliwa do silników samolotu.
- Odtwarzanie plików mpeg.
- Kontroler serwomechanizmu.
- Systemy podtrzymywania życia w szpitalu.

Czy systemy czasu rzeczywistego muszą być szybkie?

- **real time** oznacza nie "szybki", lecz "przewidywalny"
- **gwarantowany pesymistyczny czas reakcji** nie oznacza szybkiego czasu reakcji, a jedynie czas reakcji z góry określony.
- system czasu rzeczywistego może wydawać się wolniejszy od "zwykłego" systemu operacyjnego. Wynika to z faktu, że techniki stosowane do przyspieszania pracy systemu operacyjnego (pamięć podręczna, wielopotokowe procesory, etc.) wprowadzają element indeterminizmu. Indeterminizm jest niedopuszczalny w przypadku systemu cz. rz., gdyż, uniemożliwia zapewnienie przewidywalności systemu.

Systemy czasu rzeczywistego - podział

■ hard (rygorystyczne)

- gwarantują terminowe wypełnianie krytycznych zadań. Wymaga to ograniczenia wszystkich opóźnień w systemie.
- Pamięć pomocnicza jest na ogół bardzo mała albo nie występuje wcale. Wszystkie dane są przechowywane w pamięci o krótkim czasie dostępu lub w pamięci, z której można je tylko pobierać (ROM).
- Prawie nie spotyka się w systemach czasu rzeczywistego pamięci wirtualnej.
- Dlatego rygorystyczne systemy czasu rzeczywistego pozostają w konflikcie z działaniem systemów z podziałem czasu i nie wolno ich ze sobą mieszać.



Systemy czasu rzeczywistego - podział

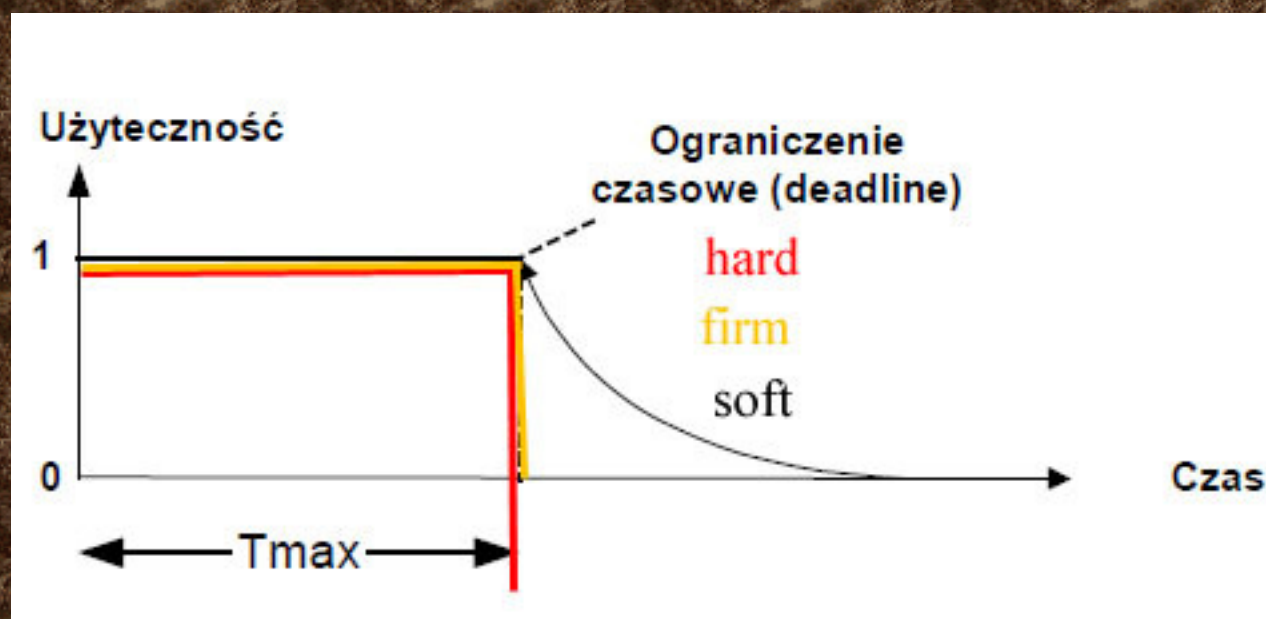
■ soft (łagodne)

- krytyczne zadanie do obsługi w czasie rzeczywistym otrzymuje pierwszeństwo przed innymi zadaniami i zachowuje je aż do swojego zakończenia.
- opóźnienia muszą być ograniczone - zadanie czasu rzeczywistego nie może w nieskończoność czekać na usługi jądra.
- łagodne wymagania dot. czasu rzeczywistego umożliwia godzenie ich z systemami innych rodzajów.
- zastosowanie w technikach multimedialnych, kreowaniu sztucznej rzeczywistości itd
- znajdują one swoje miejsce wszędzie tam, gdzie istnieje potrzeba systemów o bardziej rozbudowanych możliwościach

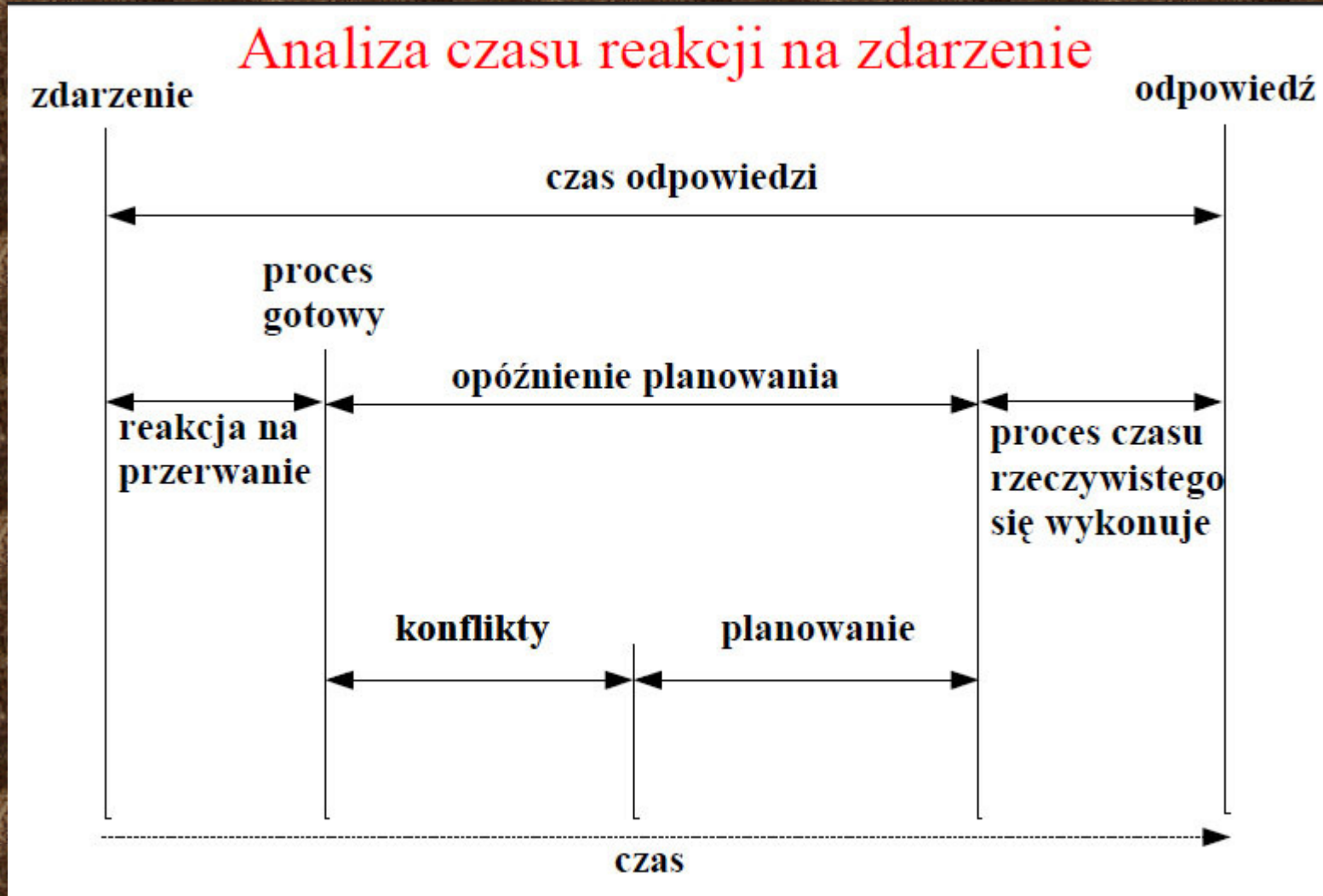
■ firm (mocne)

- wymagania pośrednie pomiędzy hard a soft
- nie wykonanie zadania w terminie skutkuje nieprzydatnością wyników, ale nie zagraża katastrofą itp.

Użyteczność odpowiedzi



Proces RT w systemie standardowym

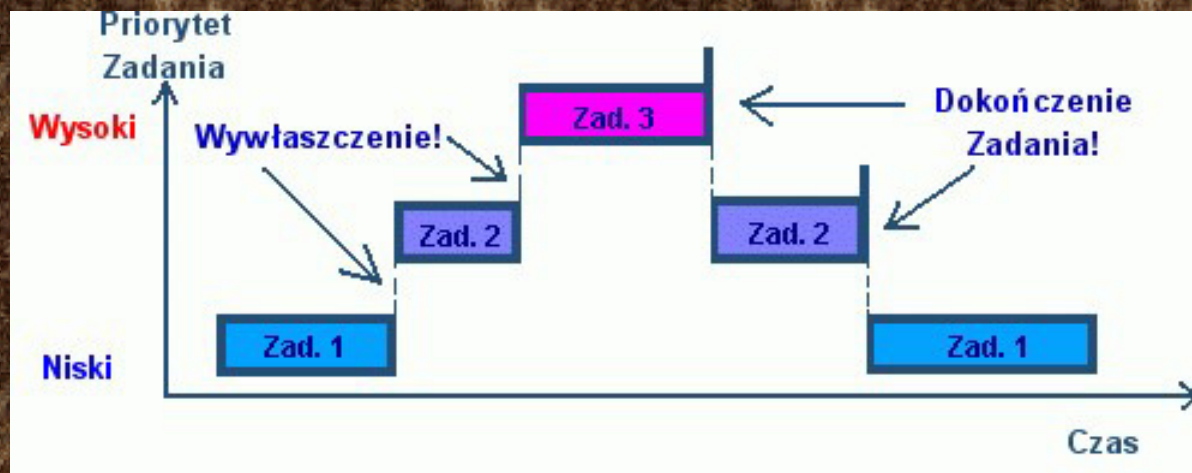


Przykładowe czasy opóźnień dla różnych systemów operacyjnych (Pentium 100MHz)

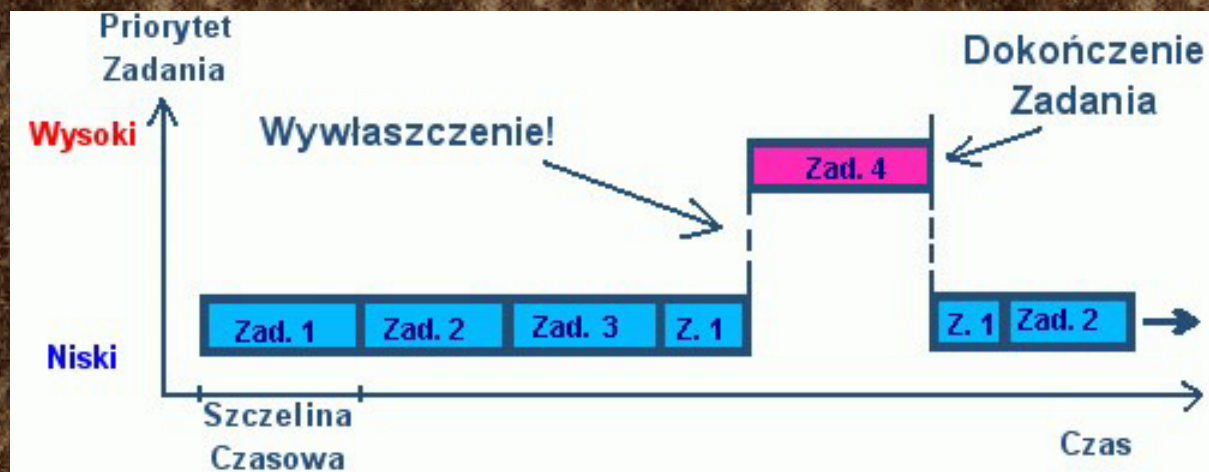
System	Tryb pracy	Opóźnienie
Windows 98/ 2000/ XP	in real time	100 μ s – 100ms
Linux	soft real-time	1ms
Linux IEEE 1003.1d	hard real-time	10 μ s – 100 μ s
Linux RT	hard real-time	1 μ s – 10 μ s
Jądro RTOS	hard real-time	1 μ s – 10 μ s

Szeregowanie zadań

- priorytetowe



- z podziałem czasu

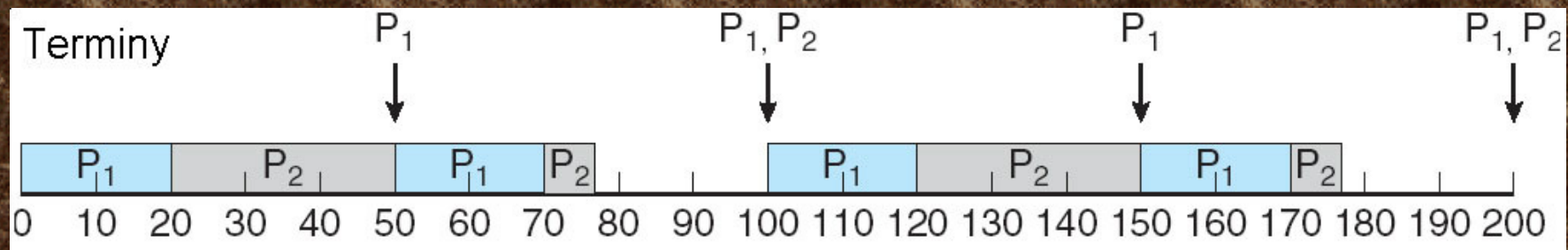


Planowanie procesów

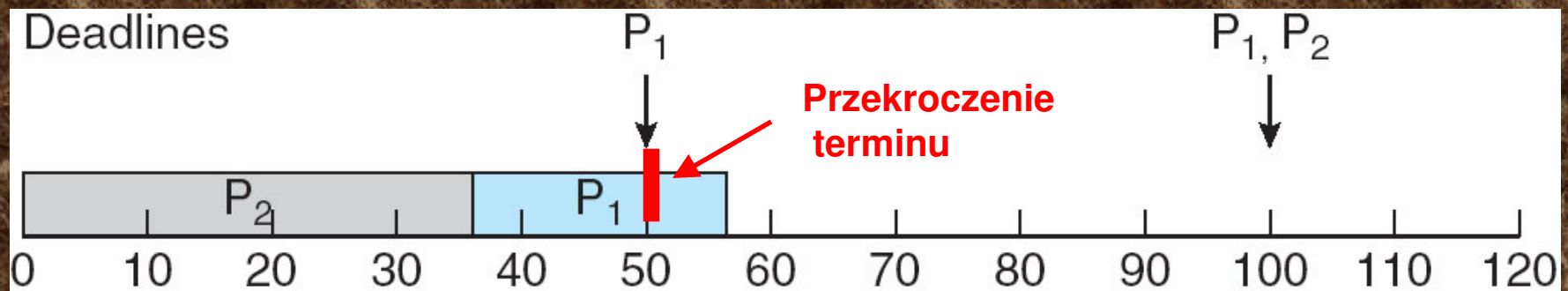
- Przyjmujemy periodyczny (okresowy) model procesów.
- Każdy proces może być opisany przez następujące parametry:
 - okres p (period) tzn. czas pomiędzy kolejnymi zdarzeniami wymagającymi obsługi przez proces
 - termin d (deadline) w którym zdarzenie musi być obsłużone (od momentu zajścia zdarzenia)
 - czas t (time) potrzebny procesowi na obsługę zdarzenia
- Zachodzi relacja $0 \leq t \leq d \leq p$
- Stopień wykorzystania procesora jest równy $u = t/p$.
- Warunek konieczny wykonywalności szeregowania: suma stopni wykorzystania procesora $\sum u \leq 1$.
- Proces oznajmia swoje parametry t, d, p planiście. Planista albo podejmuje się wykonania procesu gwarantując dotrzymania terminu albo odrzuca proces.

Rate-monotonic scheduling

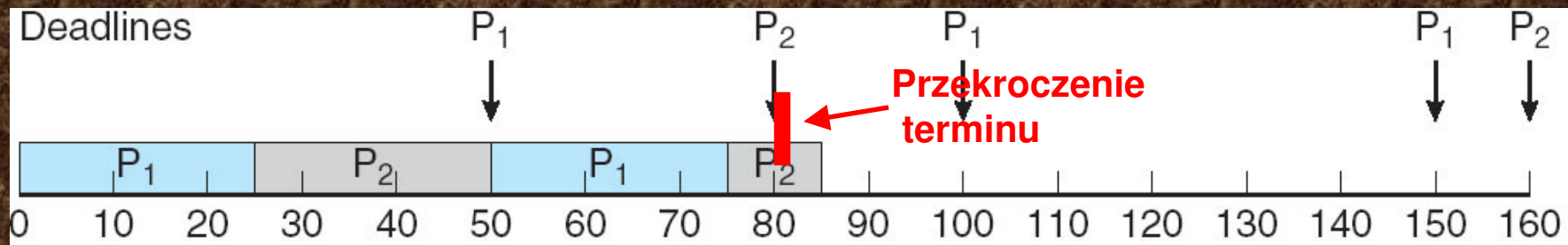
- Procesy są planowane na podstawie statycznego priorytetu równego częstotliwości zdarzeń $1/p$
- Proces o wyższym priorytecie wywłaszcza proces o niższym priorytecie.
- Przykład: dwa procesy:
 - P1: $p=50$, $d=50$, $t=20$ (P1 ma krótszy okres => większą częstotliwość i priorytet)
 - P2: $p=100$, $d=100$, $t=35$
 - Całkowite obciążenie procesora $(20/50) + (35/100)=0.75$



- Spośród klasy algorytmów z priorytetami statycznymi jest to algorytm optymalny, w takim sensie, że jeżeli nie dotrzymuje terminów, to żaden inny algorytm z tej klasy również nie dotrzyma terminów.
- Przykład: zakładamy, że proces P_2 ma większy priorytet:



- Przykład, w którym dotrzymanie terminów nie jest możliwe:
 - P1: p=50, d=50, t=25 (wyższy priorytet)
 - P2: p=80, d=80, t=35.
- Całkowite wykorzystanie procesora $(25/50) + (35/80) = 0.94$.
- Wydaje się że procesy można zaplanować, ale:



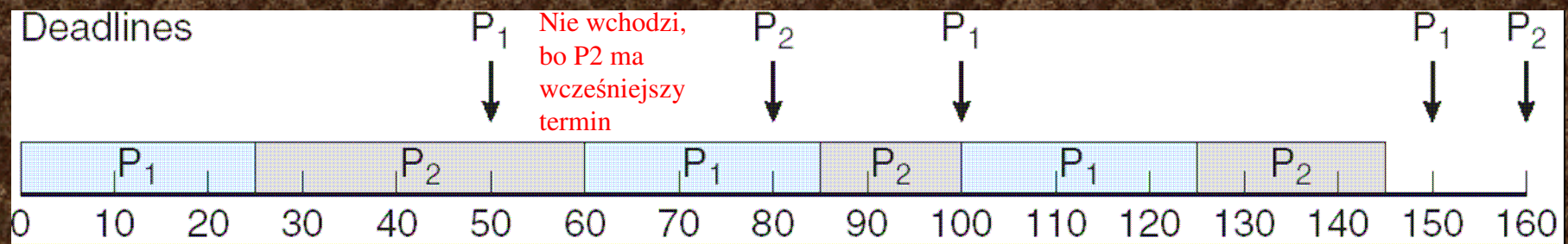
- W pesymistycznym przypadku algorytm nie gwarantuje dotrzymania terminów, gdy całkowite wykorzystanie procesora:

$$u \geq 2(2^{1/N} - 1)$$

- Dla liczby procesów $N=2$ otrzymujemy $u \approx 0.83$

Algorytm najwcześniejszy termin najpierw (ang. earliest deadline first - EDF)

- Priorytet przypisywany dynamicznie
- Przykład:
 - P1: $p=50$, $d=50$, $t=25$ (wyższy priorytet)
 - P2: $p=80$, $d=80$, $t=35$.



Systemy czasu rzeczywistego - wymagania

1. RTOS musi być wielowątkowy i wywłaszczalny.
2. W momencie gdy OS nie jest oparty na deadlinech, musi istnieć pojęcie priorytetu wątku.
3. OS musi wspierać mechanizm przewidywalnej synchronizacji wątków.
4. Musi istnieć dziedziczenie priorytetów.
5. Zachowanie OS powinno być znane.

Musi się dokładnie oszacować parametry czasowe:

1. Opóźnienie przerwania (czyli czas od wygenerowania przerwania do rozpoczęcia wykonywania zadania) - musi być zgodne z wymaganiami aplikacji, i musi być przewidywalne. Wartość ta zależy od liczby jednocześnie oczekujących przerwania.
2. Dla każdego przerwania systemowego - maksymalny czas, jaki zajmujemy. Czas powinien być przewidywalny i niezależny od liczby obiektów w systemie.
3. Maksymalny czas na jaki OS i sterowniki maskują przerwania.

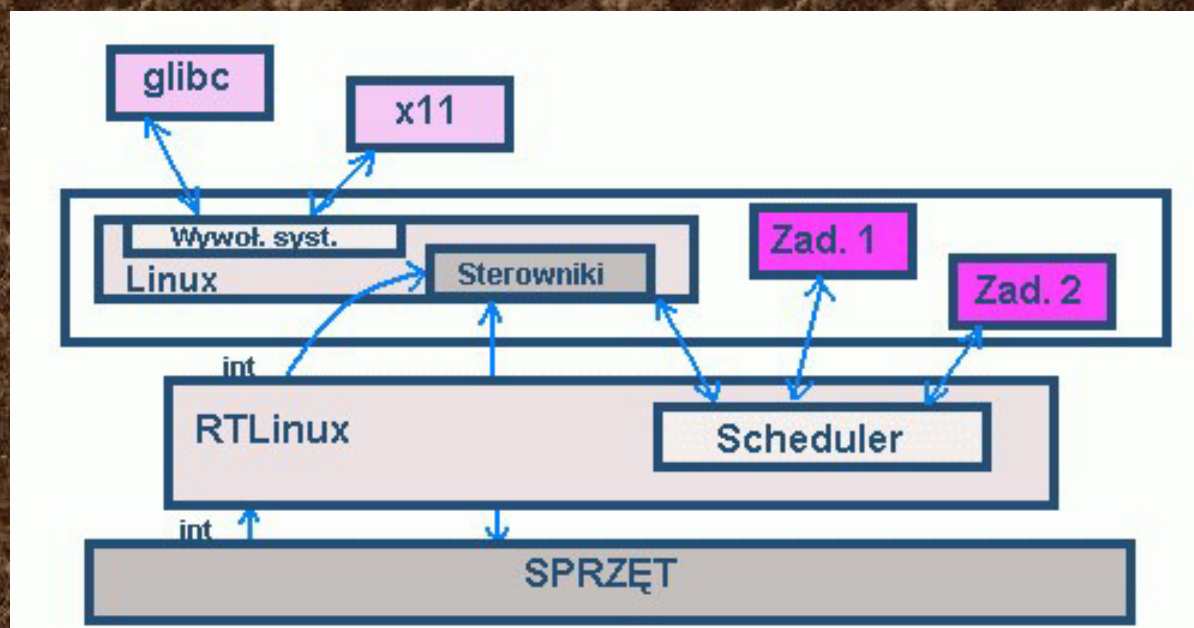
- W niektórych przypadkach, aby uzyskać RTOS, podejmuje się próby modyfikacji lub wykorzystania istniejących systemów operacyjnych. Obserwuje się dwa główne podejścia do tej kwestii:
 1. Próby balansowania pesymistycznego i średniego czasu reakcji, "robienie ogólnozadaniowego systemu operacyjnego a`la real-time".
Próby optymalizowania dwóch, zasadniczo przeciwstawnych, parametrów rzadko prowadzą do olśniewających rezultatów, a w przypadku systemów operacyjnych prowadzą raczej do soft RTOSów

2. rozbitcie systemu na dwa systemy operacyjne - real-time i "zwykczajny" (nie real-time).

Podejście to wydaje się być często skuteczne, czego przykładem jest choćby RTLinux. Niemniej jednak uważane jest za konserwatywne i potencjalnie ograniczające możliwości końcowego użytkownika.

- Windows 9x czy Windows NT mogą, przy użyciu specjalnego oprogramowania, być konwertowane do systemów real-time . Niestety, jedyne co udaje się osiągnąć to soft real-time.

RT LINUX



RT LINUX – procesy czasu rzeczywistego

- Procesy cz. rz. to programy zdefiniowane przez użytkownika, które wykonują się zgodnie z podanym harmonogramem (mogą być okresowe, zasypiać się na określony czas) i mają ścisłe wymagania czasowe.
- Procesy czasu rzeczywistego implementuje się jako ładowalne moduły jądra. Z punktu widzenia RTLinuxa procesy RT to wątki jądra
- RT. Scheduler RTLinuxa uważa, że jest tylko jeden prawdziwy proces, który ma wiele wątków. Jeden z tych wątków jest wybierany do wykonania.
- Linux jest tylko jednym z wątków, ma najniższy priorytet.

RT LINUX – procesy czasu rzeczywistego

- Procesy wykonują się w jednej przestrzeni adresowej, zatem przy zmianie kontekstu nie trzeba unieważniać rejestrów asocjacyjnych (**TLB – translation look-aside buffers**). Gdyby procesy wykonywały się we własnej przestrzeni adresowej, przy każdej zmianie kontekstu trzeba by unieważniać rejestry TLB, co przy częstym przełączaniu kontekstu daje duże obniżenie wydajności.
- Przy wywołaniach systemowych nie trzeba zmieniać poziomu uprzywilejowania.

RT LINUX – procesy czasu rzeczywistego

- Przełączanie kontekstu jest łatwiejsze -- przełącza się tylko kontekst sprzętowy: zachowuje się zawartość rejestrów na stosie i zmienia wskaźnik stosu tak, żeby wskazywał nowy stos nowego procesu. Przełączanie kontekstu jest programowe, a nie sprzętowe, bo sprzętowe przełączenie kontekstu na procesorach i486 jest powolne.
- Program i jego dane nie podlegają stronicowaniu. Nie mogą zatem zostać wysłane na dysk. Nie występują błędy braku strony, więc nie ma opóźnień z tym związanych.

RT LINUX – procesy czasu rzeczywistego

- Błąd w programie użytkownika, jakim jest proces RT, może spowodować załamanie się systemu. Pisanie programów RT wymaga takiego samego natężenia uwagi, co programowanie jądra systemu operacyjnego.
- Dodatkowym ograniczeniem nałożonym na procesy RT jest fakt, że ich zasoby są definiowane statycznie. W szczególności nie ma (w standardowym RTLinuxie) wsparcia dla dynamicznej alokacji pamięci. . W nowszych wersjach RTLinuxa dostępny jest moduł mbuff, który umożliwia korzystanie z dynamicznie alokowanej pamięci.

QNX

- Został opracowany na początku lat 80 przez założycieli kanadyjskiej firmy Quantum Software System, Limited
- Jego początkowa nazwa QUNIX (Quick UNIX), ze względu na zbyt duże podobieństwo do nazwy UNIX, nie mogła przetrwać zbyt długo i w kilka miesięcy później, za sprawą firmy AT&T, musiała zostać zmieniona i przybrała znane do dzisiaj brzmienie: QNX.
- Pierwsza wersja, była przeznaczona na komputery klasy IBM PC oraz wymagała 64kB pamięci RAM i 180kB napęd dyskietek. Pomimo swojej prostoty, umożliwia już uruchomienie kilku procesów jednocześnie.
- *"gdyby firma IBM wybrała system QNX dla mikrokomputera IBM PC, wprowadzenie na rynek modelu AT mogłoby zostać opóźnione, gdyż aplikacje uruchamiane pod systemem QNX w komputerach PC zachowują się tak, jakby zostały uruchomione pod systemem DOS w komputerze 386"*

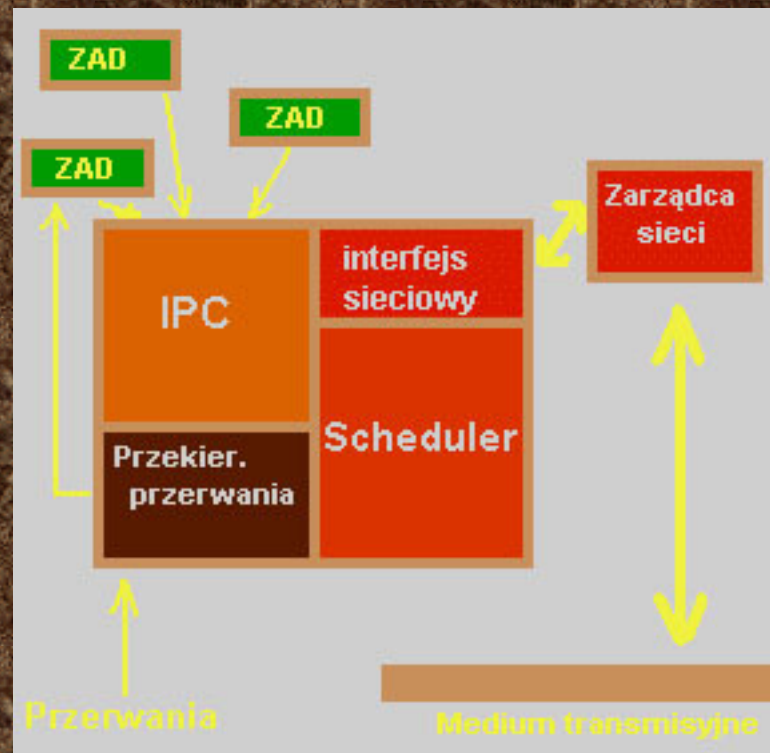
QNX - mikrojądro

- rozmiar ok. 8kB (jądro UNIX > 700kB), stąd nazwa mikrojądra - Neutrino.
- To, co odróżnia ten system ten od rodziny UNIX, to przede wszystkim struktura modułowa oraz architektura oparta o przesyłanie komunikatów (model klient - serwer).
- QNX daje możliwość zdeterminowania czasu reakcji na zdarzenia występujące w systemie.
- dzięki rozbudowanym możliwościom definiowania priorytetów, QNX jest stosowany jako system służący do sterowania automatyką przemysłową, gdzie pewne zdarzenia są krytyczne (np. otwarcie zaworu bezpieczeństwa w zbiorniku kiedy gwałtownie wzrasta ciśnienie) i muszą być zawsze obsłużone na czas.

QNX – budowa mikrojądra

- IPC - (ang. *Interprocess communication*) - obsługuje komunikację między procesami.
- Network Interface - przeźroczysta komunikacja pomiędzy procesami w obrębie sieci lokalnej.
- Hardware Interrupt Redirector - przechwytuje pojawiające się przerwania oraz przekazuje je do odpowiednich procesów obsługujących je. Część ta sama nie obsługuje przerwań!
- Realtime Scheduler - decyduje, który proces ma uzyskać dostęp do procesora w danej chwili (POSIX 1003.4 - dotyczy zagadnień czasu rzeczywistego).

QNX – budowa mikrojądra



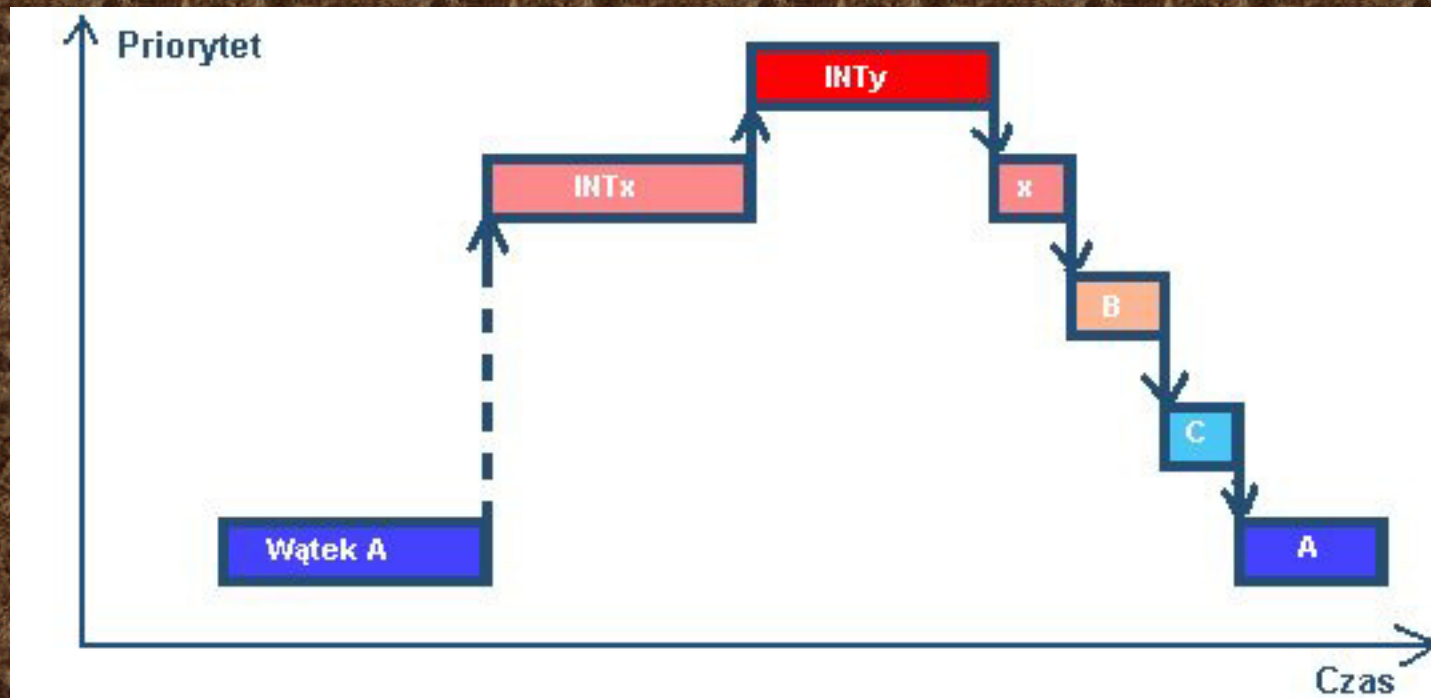
QNX - mikrojądro

- Procesy zarządzające są traktowane w identyczny sposób jak procesy użytkowe. Można je ładować, uruchamiać, zawieszać oraz usuwać niezależnie od siebie. Poza tym czynności te mogą być wykonywane dynamicznie w czasie normalnej pracy systemu
- w zależności od wymagań zewnętrznych dany proces zarządzający może zostać zainstalowany bądź usunięty. Wyjątkiem jest tutaj tzw. *Process Manager*, który musi być zawsze obecny w systemie
- Procesy zarządzające od procesów użytkowych odróżniają priorytety. P.z. mają najwyższe. Dodatkowo zyskują poziomy uprzywilejowania, które zezwalają im na realizację niektórych instrukcji mikroprocesora.
- W systemie QNX zaimplementowano trzy poziomy uprzywilejowania:
 - poziom 0 - mikrojądro
 - poziom 1 - procesy zarządzające (np. *Proc*, *Fsys* itp.)
 - poziom 2 - nie jest aktualnie wykorzystywany
 - poziom 3 - procesy użytkowe

QNX – szeregowanie zadań

- Algorytmy szeregujące wybierają zadanie gotowe do wykonania i najważniejsze, czyli to o najwyższym priorytecie w danej chwili (priorytet $0 \div 31$)
- Są trzy algorytmy szeregowania zadań (o tym samym priorytecie).
 1. kolejka FIFO. Zadanie wybrane do wykonywania, kontynuuje swoje działania aż samo odda sterowanie (np. wątek zostanie zablokowany lub proces wywoła funkcję systemową) bądź zostanie wywłaszczony przez zadanie o wyższym priorytecie.
 2. przydział czasu procesora (ang. *time slice*), który wynosi 50ms. Wówczas zadanie zostaje wywłaszczony, poza warunkami opisanymi we wcześniejszym algorytmie, również wtedy, kiedy wykorzysta przydzielony mu czas.
 3. adaptacyjny (ang. *adaptive scheduling*). Jeśli dane zadanie wykorzysta przydzielony czas procesora i nie zostanie zablokowane, to jego priorytet jest dekrementowany. Wtedy zazwyczaj następuje przełączenie kontekstu do innego zadania. Oryginalna wartość priorytetu jest natychmiast przywracana, kiedy zadanie przechodzi do stanu blokowane. Algorytm ten znajduje zastosowanie w sytuacjach, kiedy zadania wykonujące ogromne ilości obliczeń dzielą czas procesora z zadaniami interaktywnymi.

QNX – obsługa przerwań



VxWorks 5.x

wbudowana aplikacja czasu rzeczywistego

POSIX library

Java library

file systems

TCP/IP

virtual memory
VxVMI

graphics
library

mikrojądro Wind

hardware level
(Pentium, Power PC, MIPS, customized, etc.)

VxWorks 5.x

- Opcjonalne podsystemy: grafiki, systemy plików, Java, sieci TCP/IP, biblioteka Posix, pamięć wirtualna. Pozwala to na minimalizację zajętości (ang. footprint) pamięci.
- Mikrojądro (ang. microkernel) Wind
 - wywłaszczalne
 - gwarantowany czas reakcji na przerwania
- Zastosowanie: samochody, urządzenia konsumenckie, przełączniki sieciowe oraz *Marsjańskie łaziki Spirit i Opportunity*.