6,50

cz. 1-3

WIELODOSTĘPNE SYSTEMY OPERACYJNE II
Zagadnienia zaawansowane
Lech Kruś,

Cel przedmiotu:

Wprowadzenie słuchaczy do zaawansowanych zagadnień systemów operacyjnych.

Ułatwienie rozumienia podstawowych zagadnień związanych z synchronizacją procesów współbieżnych, komunikacją między procesami, blokowaniem (zakleszczaniem) procesów. Wprowadzenie do systemów rozproszonych i sieci komputerowych.

Uzupełnieniem wykładu są zajęcia laboratoryjne poświęcone pracy w systemie MS Windows 2000.

Wymagane przygotowanie słuchaczy:

- w zakresie podstaw informatyki technicznej (cyfrowej reprezentacji informacji, podstaw arytmetyki komputerów, podstaw teorii układów logicznych: A. Skorupski, Podstawy budowy i działania komputerów, WKŁ 1996),
- organizacji i architektury komputerów (P. Metzger, Anatomia PC, Helion 1996),
- wielodostępnych systemów operacyjnych I (wykłady i laboratorium na Wydz. Informatyki, WSISiZ)

Zaliczenie przedmiotu:
zaliczenie laboratorium (kolokwium), oraz egzamin z tematyki wykładów

Zakres tematyczny

Koordynacja procesów
Problem sekcji krytycznej. Rozwiązania programowe. Sprzętowe środki synchronizacji.

Semafory. Przykłady rozwiązań wybranych problemów synchronizacji: problem ograniczonego buforowania, problem czytelników i pisarzy, problem posilających się filozofów. Konstrukcje synchronizacji w językach wysokiego poziomu.

Komunikacja między procesami

Wprowadzenie - podstawowe schematy komunikacji.

Określenie nadawców i odbiorców. Komunikacja bezpośrednia. Komunikacja pośrednia.

Zagadnienia buforowania. Przykłady działań w sytuacjach wyjątkowych.

Blokady (zakleszczenia) procesów.

Definicja blokady (zakleszczenia). Warunki konieczne wystąpienia blokady.

Graf przydziału zasobów.

Zapobieganie wystąpieniu blokady - eliminacja jednego z warunków koniecznych.

Unikanie blokad. Pojęcie stanu bezpiecznego, stanu zagrożenia i blokady. Algorytm bankiera.

Wykrywanie blokad. Wychodzenie z blokady.

Wprowadzenie do systemów rozproszonych

Co to jest system rozproszony, zalety i wady systemów rozproszonych

Zagadnienia sprzętowe: wieloprocesory szynowe, wieloprocesory przełączane, multikomputery szynowe, multikomputery przełączane.

Klasyfikacja systemów operacyjnych: systemy sieciowe, prawdziwe systemy rozproszone, systemy wieloprocesorowe z podziałem czasu

Wybrane pojęcia związane z projektowaniem: przezroczystość, elastyczność, niezawodność, wydajność, skalowalność.

Wprowadzenie do sieci komputerowych

Pojęcia podstawowe: sieci z wymiana pakietów, sieci LAN i WAN, stos protokołów ISO, kapsułkowanie i demultipleksacja, interakcja klient-serwer.

Standardy w Internecie.

Sieci LAN: topologie sieci LAN, pojecie ramki i adresu sieciowego.

Intersieć i protokół TCP/IP: schemat adresowania IP, protokół ARP, protokół IP, protokół ICMP, protokół TCP, protokół UDP.

Literatura podstawowa:

Abraham Silberschatz, James.L. Peterson, Peter.B. Galvin; Podstawy systemów operacyjnych, WNT, Warszawa 1993.

Berny Goodheart, James Cox; Sekrety magicznego ogrodu. Unix System V, wersja 4 od środka. WNT, Warszawa 2001.

Berny Goodheart, James Cox; Sekrety magicznego ogrodu. Unix System V, wersja 4 od środka. Klucz do zadań. WNT, Warszawa 2001.

Maurice. J. Bach; Budowa systemu operacyjnego UNIX, WNT, Warszawa, 1995.

Andrew S. Tanenbaum: Rozproszone systemy operacyjne. PWN. Warszawa 1997.

Douglas E. Comer: Sieci komputerowe TCP/IP. (Tom 1) Zasady, protokoły i architektura. WNT, Warszawa 1997.

W. Richard Stevens: Biblia TCP/IP, tom1 protokoły. Oficyna Wyd. READ ME, 1998.

Literatura uzupełniająca:

Craig Hunt: TCP/IP Administracja Sieci. O'Reilly&Associates Inc., Oficyna Wyd. READ ME, Warszawa, 1996.

Andrew. S. Tanenbaum; Modern Operating Systems, Prentice-Hall, Inc. London, 1992.

Abraham Silberschatz, Peter.B. Galvin; Operating System Concepts. Addison Wesley, New York, 1994.

KOORDYNACJA PROCESÓW

Zagadnienie współbieżności
problemy uporządkowania wykonywania procesów
mechanizmy synchronizacji i komunikowania

Wprowadzenie

Rozpatrywany jest zbiór procesów sekwencyjnych, wykonywanych asynchronicznie, ew. współdzielących dane.

Model: proces producent,
proces konsument.

Jak zapewnić współbieżne działanie?
założyć pulę buforów,
producent zapełnia bufory,
konsument opróżnia bufory.

Synchronizacja: konsument musi czekać na wyprodukowanie tego, co chce skonsumować.

Przykład algorytmu koordynacji z ograniczonym buforem.
Zmienne dzielone:

```
var n;
type jednostka = ...;
var bufor: array[0..n-1] of jednostka;
we, wy: 0..n-1;
licznik: integer;
```

Wartości początkowe: *we,wy*: 0, *licznik* 0.
Dzielona pula buforów realizowana jest jako tablica cykliczna z dwoma wskaźnikami logicznymi:
we - wskazuje następny pusty bufor,
wy - wskazuje pierwszy zajęty bufor.

Kod producenta

```
repeat
    ...
    produkuj jednostkę w nastp
    ...
    while licznik = n do nic;
    bufor[we] := nastp;
    we := we + 1 mod n;
    licznik := licznik + 1;
until false;
```

Kod konsumenta

```
repeat
    while licznik = 0 do nic;
    nastk:= bufor[wy];
    wy := wy + 1 mod n;
    licznik := licznik - 1;
    ...
    konsumuj jednostkę w nastk
    ...
until false;
```

Oba kody poprawne, gdy rozpatrywane oddzielnie. Czy działają poprawnie przy wykonywaniu współbieżnym?

Przykład
aktualna wartość zm. *licznik* 5,
współbieżne wykonywanie:
producent: *licznik* := *licznik* + 1;
konsument: *licznik* := *licznik* - 1;
Wykonywanie instrukcji w kodzie maszynowym:

licznik := *licznik* + 1;	*licznik* := *licznik* - 1;
jako rozkazów:	jako rozkazów:
rej1 := *licznik*;	rej2 := *licznik*;
rej1:= rej1 + 1;	rej2:= rej2 - 1;
licznik := rej1;	*licznik* := rej2;

W przypadku współbieżnego wykonywania instrukcji ze zmienną *licznik* rozkazy maszynowe mogą się przeplatać w dowolnym porządku, np.:

T0: producent wykonuje	rej1:= *licznik* ;	(rej1=5)
T1: producent	rej1:= rej1 +1 ;	(rej1=6)
T2: konsument	rej2:= *licznik*;	(rej2=5)
T3: konsument	rej2:= rej2 - 1;	(rej2=4)
T4: producent	*licznik* := rej1;	(*licznik*=6)
T5: konsument	*licznik* := rej2:	(*licznik*=4)

Ostatnia wartość zm. *licznik* wskazuje na istnienie 4 pełnych buforów, a naprawdę jest 5. Gdy zmieni się kolejność T4 i T5 to wynik jest 6.
Jak zapewnić prawidłowy wynik?
Tylko jeden proces może wykonywać operacje na zmiennej *licznik* w czasie wykonywania instrukcji.
Konieczność synchronizacji.

Problem sekcji krytycznej
n procesów $\{P_0, P_1, ..., P_{n-1}\}$,
sekcja krytyczna - segment kodu (w każdym z procesów), w którym proces może zmieniać wspólne zmienne,
zasada **wzajemnego wyłączania w czasie** wykonania sekcji krytycznych procesów,
Problem: skonstruować protokół służący organizacji współpracy procesów.
Kod procesu zawiera sekwencję:
sekcja wejściowa (prośba o zezwolenie na wejście do sekcji krytycznej)
sekcja krytyczna
sekcja wyjściowa (sygnalizacja wyjścia z sekcji krytycznej)
reszta.

Wymagane warunki rozwiązania problemu:
Wzajemne wyłączanie
Postęp
Ograniczone czekanie

Założenia dot. konstrukcji algorytmów:
niezerowa prędkość wykonywania procesów
podstawowe rozkazy maszynowe wykonywane niepodzielnie
ogólna struktura rozpatrywanego procesu:

repeat
sekcja wejściowa
sekcja krytyczna
sekcja wyjściowa
reszta
until *false*;

Rozwiązania problemu sekcji krytycznej

Dwa współbieżne procesy P_0, P_1 (dla wygody ozn. P_i, P_j, i=0,1; j=1-i).

Algorytm 1
wspólna zmienna całkowita *numer* o wartościach 0 lub 1

P_i (i=0,1) :
repeat
while *numer* ≠ *i* **do** *nic*;

sekcja krytyczna

numer := *j*;

reszta
until *false*;

Algorytm 2
wspólna tablica *flaga*

var *flaga*: **array**[0..1] **of** *boolean*; (wartości pocz.: false)
P_i (i=0,1) :
repeat
flaga[i] := *true*;
while *flaga*[j] **do** *nic;*

sekcja krytyczna

flaga[i] := *false*;

reszta
until *false*;

Algorytm 3

Wspólne zmienne procesów:
var *flaga*: **array**[0..1] **of** *boolean*;
numer: 0..1;

wartości początkowe:
flaga[0]=*flaga*[1]=*false*

struktura procesu P_i: i=0,1

repeat
flaga[i] := *true*;
numer := j;
while (*flaga*[j] **and** *numer* = j) **do** *nic*;

sekcja krytyczna

flaga[i] := *false*;

reszta
until *false*;

Semafory

Semafor S - zmienna całkowita dostępna tylko za pomocą standardowych **niepodzielnych** operacji:
czekaj (wait), sygnalizuj (signal).

Definicje:

czekaj(S): **while** S≤ 0 **do** nic;
S := S - 1;

sygnalizuj(S): S := S + 1;

niepodzielność:
gdy jeden proces zmienia wartość S, żaden inny nie może; podczas sprawdzania i zmieniania wartości S nie może nastąpić przerwanie.

Przykłady zastosowań:

1. Problem sekcji krytycznej przy n procesach:
wspólny semafor *wzajwy (wzajemne wyłączanie)*,
początkowa wartość wzajwy=1,

każdy proces P_i zorganizowany:

```
repeat
      czekaj(wzajwy);
            sekcja krytyczna
      sygnalizuj(wzajwy);
            reszta
until false;
```


2. Synchronizacja wykonywania procesów
np. zapewnienie wymaganej kolejności wykonania instrukcji w różnych procesach:
współbieżne procesy P_1 z instrukcją S_1, oraz P_2 z instr. S_2
wymagane wykonanie S_2 po wykonaniu S_1.

Rozwiązanie:
wspólna zmienna semafor *synch*,
wartość początkowa *synch* = 0,
dołączenie instrukcji w procesie P_1:
S_1;
sygnalizuj(synch);

w procesie P_2:
czekaj(synch);
S_2;

Problem aktywnego czekania, wirującej blokady

Procesy stojące pod semaforem wykonują pętle instrukcji - aktywne czekanie.

Sposób rozwiązania:
Definicja semafora nie wymagająca konieczności aktywnego czekania:
(idea - wznowienie zablokowanego pod semaforem procesu przez inny proces)

```
type semafor = record
                    wart: integer;
                    L: list of proces;
               end;

czekaj(S): S.wart := S.wart - 1;
           if S.wart < 0
                then begin
                     dołącz dany proces do S.L;
                     block;
                     end;

sygnalizuj(S): S.wart := S.wart + 1;
               if S.wart ≤ 0
                    then begin
                         usuń jakiś proces P z S.L;
                         wakeup(P);
                         end;
```

Problem blokowania
Zbiór procesów jest w stanie blokady, gdy każdy proces w tym zbiorze czeka na zdarzenie spowodowane przez inny proces z tego zbioru.

Wybrane problemy synchronizacji

Problem ograniczonego buforowania
Proces „producent" przekazuje informacje za pośrednictwem n buforów do procesu „konsument".
Każdy bufor mieści jedną jednostkę.
Semafory: *wzajwy*, wartość pocz. 1,
pusty, wartość pocz. n,
pełny, wartość pocz. 0.

Struktura kodu producenta:

```
repeat
    ...
    produkowanie jednostki w nastp
    ...
    czekaj(pusty);
    czekaj(wzajwy);
    ...
    dodanie jednostki nastp do bufora bufor
    ...
    sygnalizuj( wzajwy);
    sygnalizuj(pełny);
until false;
```

Struktura kodu konsumenta:

```
repeat
    czekaj(pełny);
    czekaj(wzajwy);
    ...
    wyjmowanie jednostki z buforu bufor do nastk
    ...
    sygnalizuj( wzajwy);
    sygnalizuj(pusty);
    ...
    konsumowanie jednostki z nastk
until false;
```

Problem czytelników i pisarzy

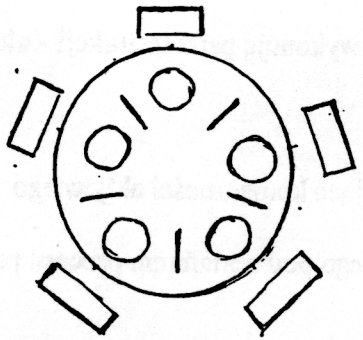
Dzielenie obiektu danych (pliku, rekordu) między kilka współbieżnych procesów.
Niektóre procesy będą tylko czytać (czytelnicy), inne mogą uaktualniać (czytać i pisać - pisarze).
Problem zagwarantowania wyłączności dostępu pisarzy do obiektu.

Najprostszy wariant: żaden czytelnik nie powinien czekać, chyba, że pisarz uzyskał dostęp.
Próba rozwiązania:
procesy czytelników dzielą zmienne:
var *wzjawy, pis: semafor*;
liczp: integer;
wartości początkowe: *wzajwy, pis* 1, *liczp* 0.

Struktura procesu pisarza:
czekaj(pis);
. . .
pisanie
. . .
sygnalizuj(pis);

Struktura procesu czytelnika:
czekaj(wzajwy);
liczp:=liczp + 1;
if *liczp* = 1 **then** *czekaj(pis)*;
sygnalizuj(wzajwy);
. . .
czytanie
. . .
czekaj(wzajwy);
liczp:= liczp - 1;
if *liczp* = 0 **then** *sygnalizuj(pis)*;
sygnalizuj(wzajwy);

Problem posilających się filozofów



Przykład prostego rozwiązania:
pałeczka - semafor
próba podniesienia pałeczki - operacja czekaj
odłożenie pałeczki - operacja sygnalizuj
Dane dzielone:
var *pałeczka*: **array**[0..4] **of** *semafor*;
wartości początkowe = 1.

Struktura procesu filozofa nr i:
repeat
czekaj(*pałeczka*[i]);
czekaj(*pałeczka*[i+1 **mod** 5]);
. . .
jedzenie
. . .
synalizuj(*paleczka*[i]);
sygnalizuj(*pałeczka*[i +1 **mod** 5]);
. . .
myślenie
. . .
until *false*;

Konstrukcje synchronizacji w językach wysokiego poziomu

Regiony krytyczne
Specjalna deklaracja zmiennej v typu T dzielonej przez wiele procesów:
var *v*: **shared** *T*;
zmienna *v* dostępna jest tylko w obrębie specjalnej instrukcji
region:
region *v* **do** *S*;

Przykład:
dwa procesy współbieżne P_0, P_1 , zawierające instrukcje:
P_0:
region *v* **do** S_1;

P_1:
region *v* **do** S_2;

wynik odpowiada sekwencyjnemu wykonaniu S_1, S_2 lub S_2, S_1.

Monitory
Monitor składa się ze zbioru operacji zdefiniowanych przez programistę.
Konstrukcja monitora gwarantuje, że w jego wnętrzu może przebywać tylko jeden proces. Inne czekają w kolejce.

Literatura: A. Silberschatz, J.L. Peterson, P.B. Galvin; Podstawy systemów operacyjnych. WNT, W-wa, 1993.

KOMUNIKACJA MIĘDZYPROCESOWA

Wprowadzenie

Dwa uzupełniające się schematy komunikacji:
pamięć dzielona - procesy współużytkowują pewne zmienne,
system komunikatów - wymiana komunikatów między procesami bez użycia zmiennych dzielonych.

Podstawowe operacje komunikacji między procesami:
nadaj(komunikat)
odbierz(komunikat)

Zagadnienia dotyczące ustanowienia łącza

Zagadnienia logicznej implementacji łącza

OKREŚLENIE NADAWCÓW I ODBIORCÓW

Komunikacja bezpośrednia
proces nadający (lub odbierający) komunikat jawnie określa odbiorcę (lub nadawcę)

Cechy łącza:
ustanawiane automatycznie między dwoma procesami na podstawie ich identyfikatorów,
dotyczy dokładnie dwóch procesów i jest dla każdej pary procesów tylko jedno,
jest dwukierunkowe.

Definicja podstawowych operacji:
nadaj(P, komunikat) , gdzie P jest odbiorcą
odbierz(Q, komunikat), gdzie Q jest nadawcą

Przykład rozwiązania problemu producenta i konsumenta

Proces producenta:

```
repeat
    ...
    wytwarzaj jednostkę w nastp
    ...
    nadaj(konsument, nastp);
until false;
```

Proces konsumenta:

```
repeat
    odbierz(producent, nastk);
    ...
    konsumuj jednostkę z nastk
    ...
until false;
```


Komunikacja pośrednia
wykorzystanie skrzynek pocztowych (portów) o jednoznacznej identyfikacji.

Definicja podstawowych operacji
nadaj(A, komunikat),
odbierz(A, komunikat),
gdzie A jest identyfikatorem skrzynki pocztowej.

Cechy łącza:
ustanawiane między procesami, gdy dzielą skrzynkę pocztową,
może dotyczyć więcej niż dwóch procesów,
każda para procesów może mieć kilka różnych łączy,
może być jedno lub dwukierunkowe

Różne rozwiązania systemowe dotyczące:
własności skrzynki (procesu lub systemu operacyjnego),
tworzenia skrzynki, nadawania i odbierania komunikatów, usuwania skrzynki.

Zagadnienia buforowania
Łącze ma określoną pojemność - tylko określona liczba komunikatów może w nim przebywać - kolejka komunikatów przyporządkowanych do łącza.

Sposoby implementacji kolejki:
pojemność zerowa,
pojemność ograniczona,
pojemność nieograniczona .

Przykłady działań w sytuacjach wyjątkowych:

Zakończenie procesu

Utrata komunikatów

Zniekształcenie komunikatów

WSO II Slajdy cz. 2.
Lech Kruś

BLOKADY - ZAKLESZCZENIA (DEADLOCKS)

Blokada: sytuacja, w której procesy wzajemnie przetrzymują zasoby, do których chciałyby uzyskać dostęp. W sytuacji takiej procesy są w stanie oczekiwania, z którego nie mogą wyjść.

OPIS SYSTEMU

Zasoby systemu
System zawiera skończoną liczbę zasobów różnego typu.
Mogą występować grupy równoważnych zasobów.

Zasady korzystania z zasobów przez procesy
- **Zamówienie zasobu**
- **Użycie**
- **Zwolnienie zasobu**

Def. blokowania (deadlock):
Zbiór procesów jest w stanie blokady, jeśli każdy proces z tego zbioru czeka na zdarzenie spowodowane przez inny proces z tego samego zbioru.

Przykłady zasobów:
zasoby fizyczne: drukarki, napędy taśmy, cykle procesora, pamięć
zasoby logiczne: pliki, semafory, monitory

WARUNKI KONIECZNE WYSTĄPIENIA BLOKADY

Wzajemne wyłączanie
Co najmniej jeden zasób jest niepodzielny.
Tylko jeden proces może korzystać z tego zasobu, inne procesy zamawiające ten zasób są opóźniane.

Przetrzymywanie i oczekiwanie
Musi istnieć proces mający przydzielony pewien zasób (co najmniej jeden) i oczekujący na przydział dodatkowego zasobu, przetrzymywanego przez inny proces.

Brak wywłaszczeń
Tylko proces przetrzymujący określony zasób, może ten zasób zwolnić.

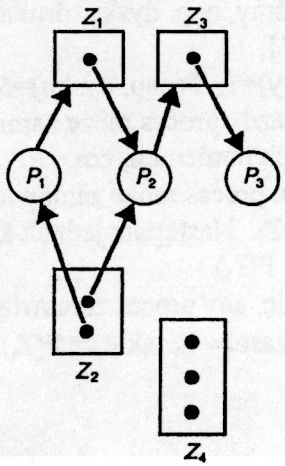
Czekanie cykliczne
Musi istnieć zbiór oczekujących procesów $\{P_0, P_1, \ldots, P_{n-1}\}$, takich, że P_0 czeka na zasób przetrzymywany przez P_1, P_1 czeka na zasób przetrzymywany przez P_2, itd. . . . , aż P_{n-1} czeka na zasób przetrzymywany przez P_0.

GRAF PRZYDZIAŁU ZASOBÓW

Graf skierowany opisujący blokady.

Zbiór wierzchołków W składający się z podzbiorów:
$P=\{ P_1 , P_2 , \ldots , P_n\}$ podzbiór procesów,
$Z=\{ Z_1 , Z_2 , \ldots , Z_m\}$ podzbiór typów zasobów.

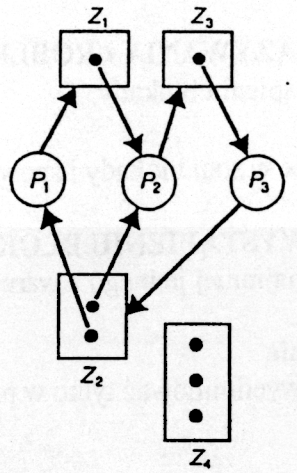
Zbiór krawędzi skierowanych K:
$P_i \rightarrow Z_j$ oznacza, że proces P_i zamówił zasób Z_j,
$Z_i \rightarrow P_j$ oznacza, że zasób Z_i został przydzielony procesowi P_j.



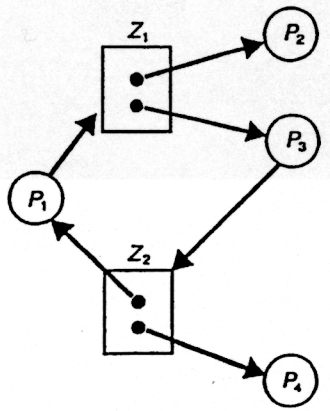
Przykład grafu (oznaczenia na rys. : proces - ○ , zasób - □).

Warunek konieczny wystąpienia blokowania:
graf przydziału zasobów zawiera cykl.

W przypadku, gdy każdy typ zasobów zawiera tylko jeden egzemplarz, to cykl zawarty w grafie przydziału zasobów jest warunkiem koniecznym i dostatecznym.



Przykład grafu z blokadą



Przykład grafu z cyklem, ale bez blokady

SPOSOBY ROZWIĄZYWANIA PROBLEMU BLOKADY

- Zapobieganie wystąpieniu blokady,
- Unikanie blokad.
- Pozwolić na wejście w stan blokady i spowodować jej usunięcie.

ZAPOBIEGANIE WYSTĄPIENIU BLOKADY

Wyeliminowanie co najmniej jednego z warunków koniecznych.

Wzajemne wyłączanie

Warunek ten można wyeliminować tylko w przypadku zasobów podzielnych.

Przetrzymywanie i oczekiwanie

idea: zastosować protokół zapewniający, że proces zamawiający określony zasób nie powinien przetrzymywać innych.

Brak wywłaszczeń

idea: zastosować odpowiedni protokół wywłaszczeniowy

Czekanie cykliczne

idea: wyeliminować czekanie cykliczne przez uporządkowanie zasobów i zastosowanie odpowiedniego protokołu przydzielania zasobów procesom.

$Z=\{Z_1, \ldots Z_m\}$ zbiór zasobów

F: Z -> N funkcja przyporządkowująca każdemu zasobowi liczbę naturalną ze zbioru N.

Przykład:
Z={nap. taśmy, nap. dysku, drukarki},
N={ 1, 5, 7},
F(nap.taśmy)=1, F(nap. dysku)=5, F(drukarki)=7,
protokół: każdy proces może zamawiać zasoby tylko we wzrastającym porządku ich numeracji, tzn.
na początku proces może zamówić dowolną dostępną liczbę egz. zasobu np. Z_i. Następnie jednak każdy egz. zasobu Z_j tylko wtedy, gdy $F(Z_j) > F(Z_i)$.
Wymaga się, aby proces zamawiający zasób Z_j miał wcześniej zwolnione zasoby Z_i takie,że $F(Z_i) \geq F(Z_j)$.

UNIKANIE BLOKAD

idea: przy każdym zamawianiu zasobów przez proces, system operacyjny decyduje czy ten proces ma czekać czy nie. Wymagana jest wcześniejsza informacja jak procesy będą zamawiać i zwalniać zasoby. Różne algorytmy wymagają różnych ilości i typów informacji.

Na podstawie deklaracji procesów o maksymalnej liczbie potrzebnych zasobów konstruuje się algorytmy przydzielania zasobów, tak aby system nie wszedł w stan blokady. Algorytm dynamicznie sprawdza stan przydziału zasobów i decyzja o przydziale podejmowana jest tak aby nie dopuścić do spełnienia warunku czekania cyklicznego.

Stan przydziału zasobów określony jest przez liczbę zasobów dostępnych, przydzielonych oraz przez maksymalne zapotrzebowania procesów.

Stan bezpieczny -
gdy istnieje **ciąg bezpieczny procesów $P_1, \ldots, P_n$ w danym stanie przydziałów,** tzn. taki, że dla każdego procesu P_i jego potencjalne zapotrzebowanie na zasoby musi być zaspokojone przez zasoby aktualnie dostępne oraz przez zasoby użytkowane przez procesy P_j, $j<i$.

Stan zagrożenia - gdy żaden taki ciąg nie istnieje.

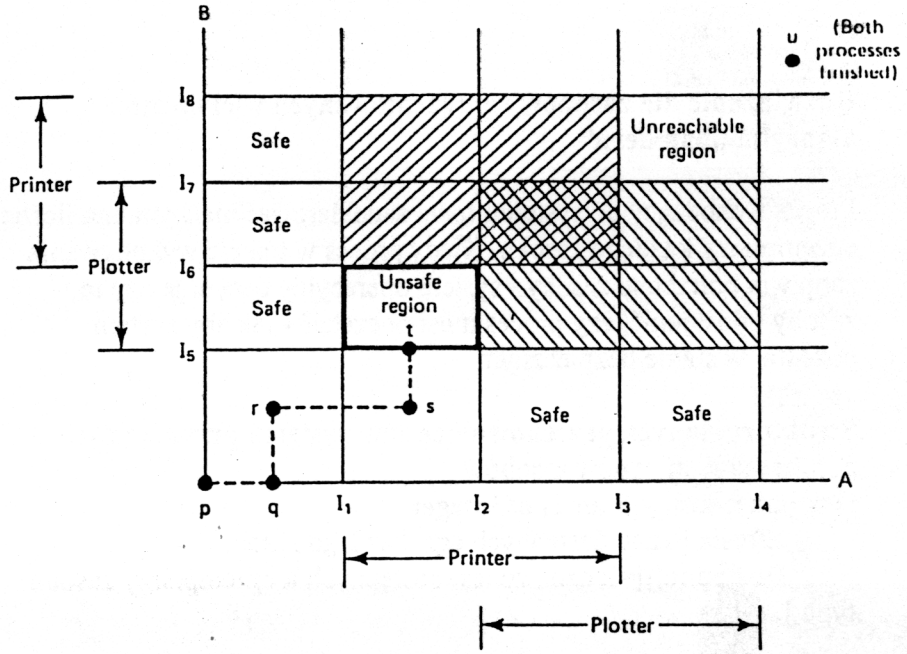


Fig. 6-6. Two process resource trajectories.

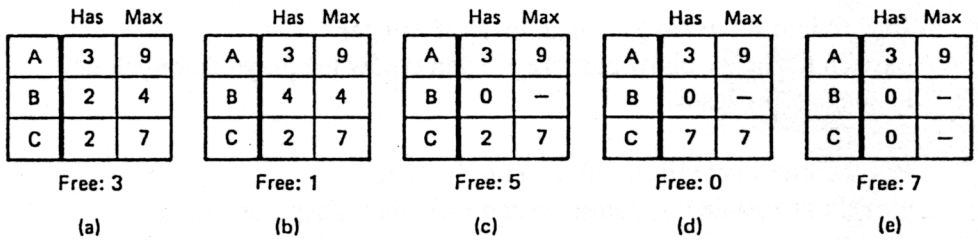


Fig. 6-7. Demonstration that the state in (a) is safe.

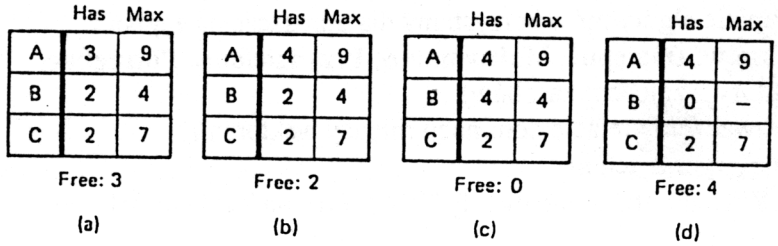


Fig. 6-8. Demonstration that the state in (b) is not safe.

Rozwiązanie dla zasobów reprezentowanych wielokrotnie algorytm „bankiera"

Proces wchodzący do systemu musi zadeklarować maksymalna liczbę egzemplarzy każdego zasobu. Kiedy proces w trakcie wykonywania zamawia potrzebne zasoby - system operacyjny decyduje czy te zasoby udostępnić, czy proces musi poczekać - tak aby system pozostał w stanie bezpiecznym

Struktury danych przechowujące stan systemu przydziałów.
n procesów, m typów zasobów
Dostępne: array[0..m-1] of integer
określa liczbę dostępnych egz. każdego zasobu
Dostępne[j]=k ozn., że jest dostępnych k egzemplarzy zasobu typu **j.**

Maksymalne: array[0..n-1,0..m-1] of integer
określa maksymalne żądania procesów względem zasobów poszczególnych typów.

Przydzielone: array[0..n-1,0..m-1] of integer
określa liczbę zasobów poszczególnych typów już przydzielonych do każdego z procesów.

Potrzebne: array[0..n-1,0..m-1] of integer
określa pozostałe do spełnienia zamówienia każdego procesów.

Uwagi:
Potrzebne[i,j]=Maksymalne[i,j]-Przydzielone[i,j]
Struktury te są dynamiczne - zmieniają w czasie wymiary i wartości
W zapisie algorytmów korzystamy dla uproszczenia zapisu z wektorów: **Dostępne, Maksymalne$_i$, Przydzielone$_i$, Potrzebne$_i$** (wektory względem zasobów).
Wprowadzamy relację dominacji między wektorami
X, Y o wymiarach m:
$X \leq Y$ wtedy i tylko wtedy gdy $X[i] \leq Y[i]$ dla każdego i=1,..,m.

Zamówienia procesu i na zasoby oznaczane są przez:
Zamówienia$_i$: array[0..m-1] of integer

Algorytm działań
podejmowanych, gdy proces i wykonuje zamówienia

Krok 1: Sprawdź czy **Zamówienia$_i \leq$ Potrzebne$_i$**
tak: wykonaj **krok 2,**
nie: warunek błędu.

Krok 2: Sprawdź czy **Zamówienia$_i \leq$ Dostępne**
tak: wykonaj **krok 3,**
nie: proces **i** musi czekać.

Krok 3: Wykonaj próbę przydziału zasobów:
Dostępne:= Dostępne - Zamówienia$_i$;
Przydzielone$_i$:= Przydzielone$_i$ + Zamówienia$_i$;
Potrzebne$_i$:= Potrzebne$_i$ - Zamówienia$_i$;

Sprawdź czy **stan jest bezpieczny** (algorytm bezpieczeństwa).
tak: przydziel zasoby procesowi **i** zgodnie z zamówieniem,
nie: przywróć poprzedni stan zasobów, proces **i** musi czekać na realizację zamówienia: **Zamówienia$_i$.**

Algorytm bezpieczeństwa
Wprowadzamy wektory **Praca** o wymiarze m, **Koniec** o wymiarze n .
Krok 1: Przypisania początkowe:
Praca:= Dostępne;
Koniec[i]:= false; dla i=0,...,n-1.

Krok 2: Znajdź takie **i** , że jednocześnie:
Koniec[i]=false and
$Potrzebne_i \leq Praca$
czy istnieje takie **i?**
tak: wykonaj **krok 3**
nie: wykonaj **krok 4.**

Krok 3: Praca := Praca + $Przydzielone_i$;
Koniec[i]:= true;
wykonaj krok 2.

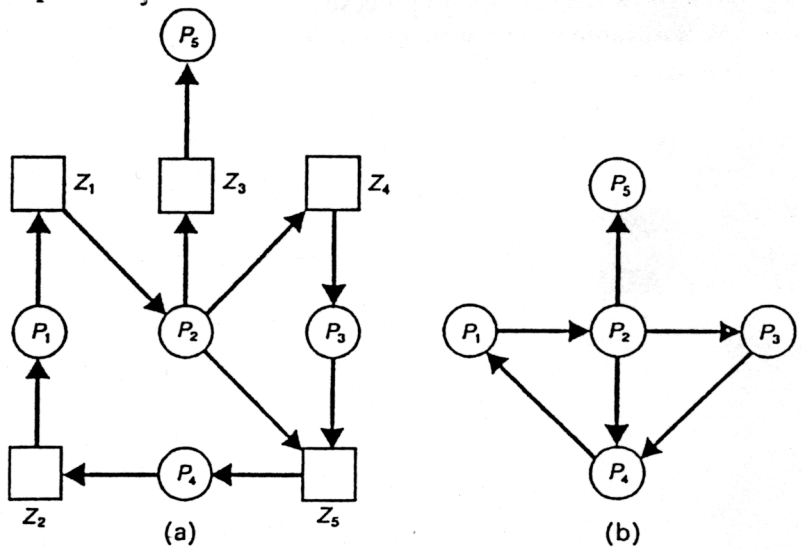
Krok 4: Jeśli **Koniec[i] = true** dla wszystkich **i** to system jest w stanie bezpiecznym.

WYKRYWANIE I WYCHODZENIE Z BLOKADY

WYKRYWANIE BLOKAD

Algorytmy wykrywania blokad dla zasobów reprezentowanych wielokrotnie i jednokrotnie.
Przykład algorytmu dla pierwszej klasy - patrz Silberschatz i inni, Podstawy systemów operacyjnych.WNT, W-wa, 1993.

Zasoby reprezentowane pojedynczo
idea algorytmu:
Budowa grafu przydziałów.
Budowa grafu oczekiwań (przez usunięcie węzłów zasobów)
Prawędź P_i -> P_j oznacza, że proces P_i czeka na zwolnienie zasoów przez proces P_j.



Wykrywanie blokady przez wykrywanie pętli w grafie oczekiwań.

Niezbędne „utrzymywanie" na bieżąco grafu oczekiwań i wykonywanie algorytmu szukającego pętli w grafie.

Kiedy i jak często wykrywać blokady?

WYCHODZENIE Z BLOKADY

Usunięcie jednego (lub kilku) procesów w celu przerwania cyklicznego czekania

usunąć wszystkie procesy w blokadzie
znaczny koszt

usuwać procesy pojedynczo, aż do usunięcia blokady
Sposób działania: usunąć proces,
wykonać algorytm wykrywania blokady.
Problem wyboru procesu do usunięcia.

Wywłaszczenie procesów z zasobów
Wybór wywłaszczanego procesu.
Wznawianie wycofywanego procesu.
Głodzenie procesu.

Połączenie metod rozwiązywania problemu blokady
Zastosowanie różnych metod dla różnych klas zasobów.

Zasoby wewnętrzne systemu (np. bloki kontrolne procesów)
Zapobieganie powstawaniu blokad przez uporządkowanie zasobów (nie trzeba dokonywać wyborów między realizowanymi zamówieniami.

Pamięć główna
Wywłaszczanie.

Zasoby zadania (przydzielane urządzenia, pliki, ..)
Unikanie blokad.

Obszar wymiany
Zastosowanie wstępnego przydziału.

WSO II Cz. 3.

WSTĘP DO SYSTEMÓW ROZPROSZONYCH

Postęp technologii:
- mikroprocesory
- szybkie sieci komputerowe
- łatwość budowy sieci - łączenia wielu jednostek centralnych w jeden system - budowa sieci LAN, WAN

Podstawowy problem:
- wymagane jest oprogramowanie odmienne niż w systemach scentralizowanych

Co to jest system rozproszony?
- Układ niezależnych komputerów, który sprawia wrażenie na jego użytkownikach, że jest jednym komputerem

Cele budowy systemów rozproszonych

Zalety (w porównaniu z systemami scentralizowanymi)
- lepszy współczynnik cena/wydajność
- uzyskanie wydajności nieosiągalnych w systemach scentralizowanych
- wymagane w przypadku wewnętrznego rozproszenia zastosowań
- większa niezawodność
- możliwość stopniowego rozszerzania systemu

Zalety (w porównaniu z niezależnymi komputerami PC)
- umożliwienie użytkownikom dostępu do wspólnej bazy danych
- dostęp wielu użytkowników do wspólnych urządzeń zewnętrznych
- komunikacja między użytkownikami
- powiększenie elastyczności

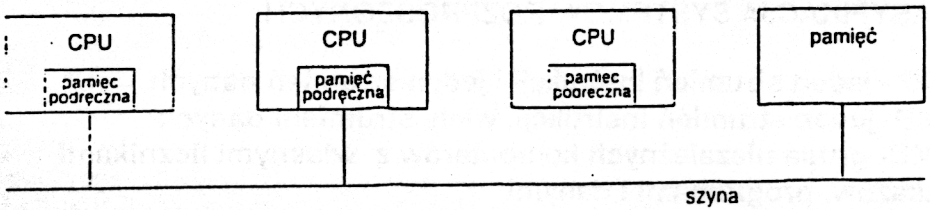
Wady systemów rozproszonych - dotyczące
- oprogramowania
- sieci
- bezpieczeństwa

KLASYFIKACJA SYSTEMÓW ROZPROSZONYCH

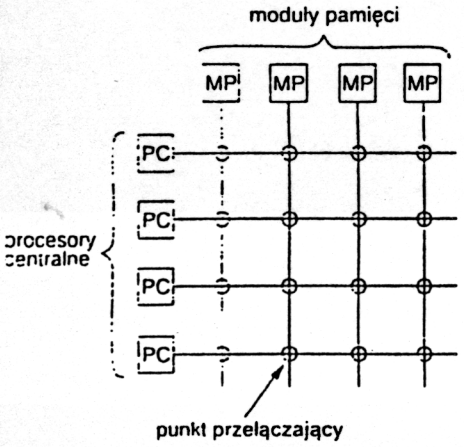
SISD - jeden strumień instrukcji i jeden strumień danych
SIMD- jeden strumień instrukcji, wiele strumieni danych
MIMD- grupa niezależnych komputerów z własnymi licznikami rozkazów, programami i danymi

Systemy MIMD
- ze względu na budowę:
  - wieloprocesory
  - multikomputery
- ze względu na architekturę sieci powiązań:
  - szynowe (bus)
  - przełączane (switched)
- systemy ściśle powiązane
- systemy luźno powiązane

Wieloprocesory szynowe

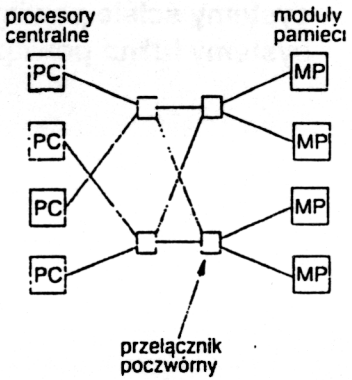


Wieloprocesory przełączane



(a)

Wybierak krzyżowy

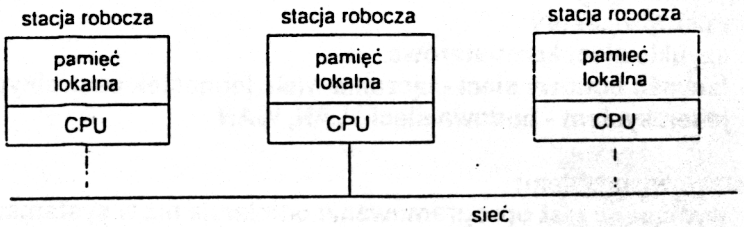


(b)

Sieć przełączająca omega

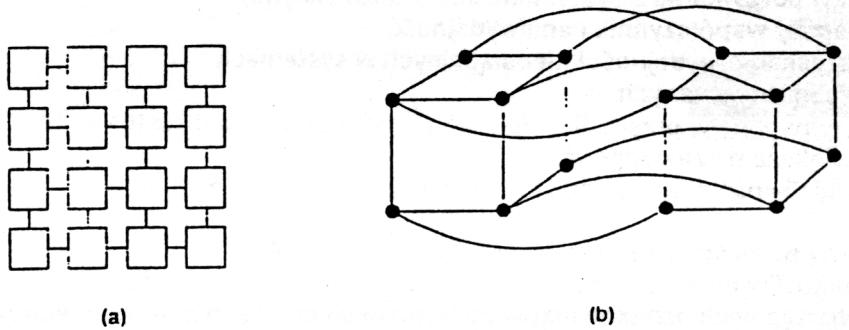
Rysunki podano za pracą:
A. S. Tanenbaum; Rozproszone systemy operacyjne. PWN, Warszawa, 1997

Multikomputery szynowe



Rys. 1.7. Multikomputer złożony ze stacji roboczych i sieci LAN

Multikomputery przełączane



(a) (b)

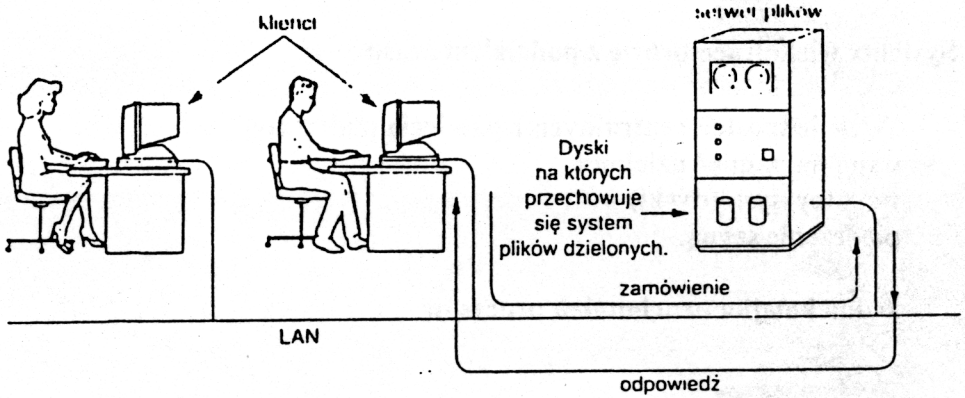
Rys. 1.8. (a) Krata. (b) Hiperkostka

OPROGRAMOWANIE

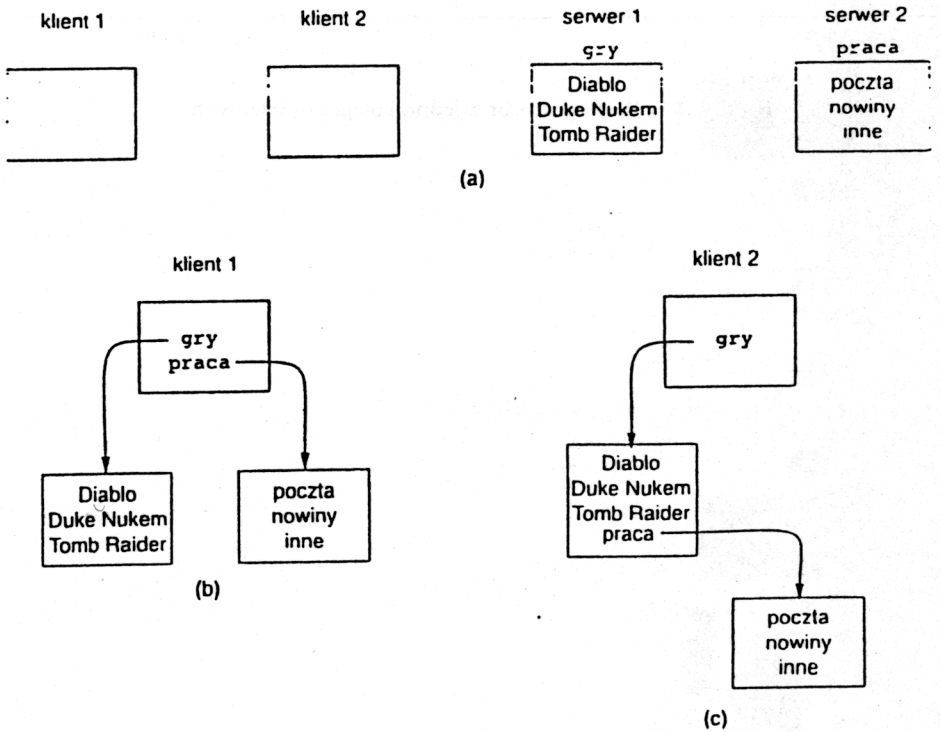
Sieciowe systemy operacyjne
- Stacje robocze połączone siecią LAN
- Każda maszyna ma własny system operacyjny
- Podstawowe usługi: logowanie na innej maszynie, kopiowanie plików między maszynami
- Serwer plików realizuje globalny system plików dzielonych
- Program użytkowy wykonywany tylko na lokalnej maszynie

Prawdziwe systemy rozproszone
- Wiele komputerów połączonych siecią
- Wrażenie jednolitego systemu (single system image) (wirtualny monoprocesor - virtual uniprocesor)
- Wymagania:
- Jednolity, globalny system komunikacji między procesami
- Jednakowe zarządzanie procesami
- Jednolity system plików
- Ten sam interfejs odwołań systemowych
- Znaczna kontrola sprawowana przez jądro nad własnymi zasobami

Systemy wieloprocesorowe z podziałem czasu
- Wiele jednostek centralnych z pamięcią podręczną
- wspólna pamięć dzielona
- wspólny dysk (dyski)
- połączenie szyną.
- Jedna kolejka uruchomień procesów



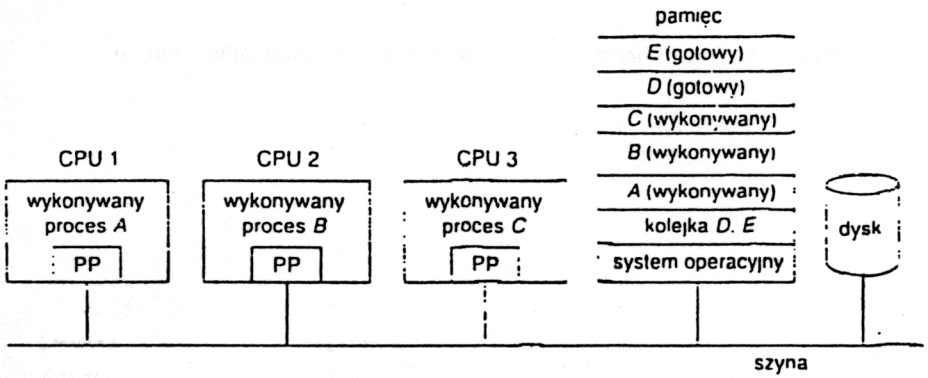
Rys. 1.9. Dwaj klienci i serwer w sieciowym systemie operacyjnym



Systemy wieloprocesorowe z podziałem czasu

Wiele jednostek centralnych z pamięcią podręczną
wspólna pamięć dzielona
wspólny dysk (dyski)
połączenie szyną.

Jedna kolejka uruchomień procesów



Rys. 1.11. Wieloprocesor z jedną kolejką uruchomień

Porównanie klas systemów operacyjnych z wieloma (N) jednostkami centralnymi

Zagadnienie	Sieciowy system operacyjny	Rozproszony system operacyjny	Wieloprocesorowy system operacyjny
Czy wygląda jak wirtualny monoprocesor	nie	tak	tak
Czy wszyscy muszą wykonywać ten sam system operacyjny	nie	tak	tak
Ile jest kopii systemu operacyjnego	N	N	1
Sposób komunikacji	pliki dzielone	komunikaty	pamięć dzielona
Czy uzgadnia się protokoły komunikacji	tak	tak	nie
Czy istnieje jedna kolejka uruchomień	nie	nie	tak
Czy dzielenie plików ma dobrze określoną semantykę?	zwykle nie	tak	tak

PODSTAWOWE WYMAGANIA (OCZEKIWANIA) WOBEC SYSTEMÓW ROZPROSZONYCH

Przezroczystość

Przezroczystość położenia: użytkownicy nie mogą określić lokalizacji zasobów.

Przezroczystość wędrówki (migration transparency): zasoby mogą być przemieszczane bez zmiany nazw

Przezroczystość zwielokrotniania (replication transparency): użytkownicy nie mogą określić liczby istniejących kopii

Przezroczystość współbieżności (concurrency transparency): automatyczne dzielenie zasobów między użytkowników - współbieżnie, a nie sekwencyjnie.

Przezroczystość działań równoległych (parallelisn transparency): zadania wykonywane równolegle bez wiedzy użytkowników

Elastyczność

Dwie struktury systemów:

1. Każda maszyna wykonuje **monolityczne jądro** dostarczające **większości usług.**
2. Idea **Mikrojądra** zapewniającego nieliczne usługi, a większość usług zapewniana przez specjalizowane serwery poziomu użytkownika.

Niezawodność

Dostępność
Integralność danych
Bezpieczeństwo
Tolerowanie awarii

Wydajność

Skalowalność

Idea algorytmów zdecentralizowanych

SYNCHRONIZACJA PROCESÓW W SYSTEMACH ROZPROSZONYCH

Cechy algorytmów rozproszonych

- Informacje rozmieszczone na wielu maszynach
- Procesy podejmują decyzje tylko na podstawie informacji lokalnych
- Należy unikać skupiania elementów wrażliwych na awarie w jednym punkcie systemu
- Nie istnieje wspólne precyzyjne źródło czasu (wspólny zegar)

SYNCHRONIZACJA ZEGARÓW

Ilustracja działania programu **make** w systemie dwóch maszyn o niezależnych zegarach.

Logiczna synchronizacja zegarów

Zegary logiczne (logical clocks): zapewniające wewnętrzną zgodność czasu
Zegary fizyczne (physical clocks): czas wskazywany przez zegary jest zgodny z czasem rzeczywistym (z określoną dokładnością)

Algorytm synchronizacji zegarów logicznych (Lamport)

Rozpatrujemy system rozproszony, w którym jest wiele procesów, każdy na innej maszynie, każdy ma własny czasomierz.

Relacja uprzedniości zdarzeń (happens-before relation):
Def. : Mówimy, że zdarzenie **a** poprzedza zdarzenie **b** i piszemy
a -> b
wtedy i tylko wtedy, gdy wszystkie procesy są zgodne co do tego, że zdarzenie **a** zachodzi najpierw, a potem dopiero zdarzenie **b.**

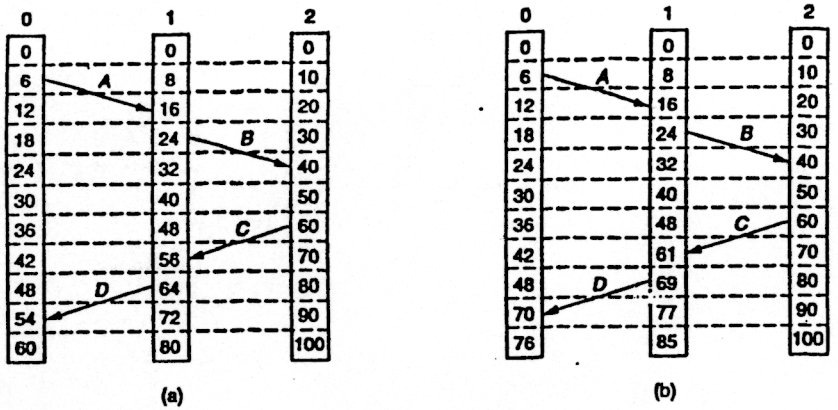
Relacja ta zachodzi bezpośrednio w przypadkach:
1. Jeżeli **a** i **b** są zdarzeniami w tym samym procesie i **a** występuje przed **b**, to relacja **a -> b** jest prawdziwa.
2. Jeżeli **a** jest zdarzeniem wysłania komunikatu przez jeden proces i **b** jest zdarzeniem odebrania komunikatu przez inny proces, to relacja **a -> b** jest prawdziwa.

Przechodniość relacji: jeżeli **a -> b i b -> c,** to **a -> c.**

Zdarzenia współbieżne (concurrent): dwa zdarzenie występują w różnych procesach, które nie wymieniają komunikatów.

Przypisanie wartości czasu zdarzeniom
(ozn. C(a)) powinno mieć własności:

1. Jeżeli **a -> b** to **C(a) < C(b).**
2. Czas zegarowy **C** musi zawsze wzrastać.



Rys. Trzy procesy (0, 1, 2) z własnymi zegarami bez korekty i z korektą wg. algorytmu Lamporta.

Warunki przypisania czasu wszystkim zdarzeniom w systemie rozproszonym (zastosowane w algorytmie Lamporta)

1. Jeżeli zdarzenie **a** poprzedza zdarzenie **b** w tym samym procesie, to **C(a) < C(b)**.
2. Jeżeli **a** oznacza nadanie komunikatu, a **b** jego odebranie, to **C(a) < C(b)**.
3. Dla wszystkich zdarzeń **a** i **b**, **C(a) ≠ C(b)**.