

**WYŻSZA SZKOŁA INFORMATYKI STOSOWANEJ I ZARZĄDZANIA  
WYDZIAŁ INFORMATYKI**

**WIELODOSTĘPNE SYSTEMY OPERACYJNE II  
(SO2)  
Zagadnienia zaawansowane**

Semestr 4

**CZĘŚĆ 1**

**KONSPEKT WYKŁADÓW**

**KOORDYNACJA PROCESÓW**

**Lech Kruś**

# KONSPEKT

## 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 Server z elementami integracji do rozwiązań Cloud (Microsoft Azure) .

## Wymagane przygotowanie słuchaczy:

- podstawy informatyki technicznej (cyfrowej reprezentacji informacji, podstaw arytmetyki komputerów, podstaw teorii układów logicznych: A. Skorupski, Podstawy budowy i działania komputerów, WKŁ 1996),
- organizacja i architektura komputerów (P. Metzger, Anatomia PC, Helion 1996),
- wielodostępne systemy operacyjne I (wykłady i laboratorium na Wydz. Informatyki, WSISiZ)

## Zaliczenie przedmiotu:

zaliczenie laboratorium oraz egzamin z tematyki wykładów

## **Zakres tematyczny**

### **Koordinacja 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 wieloprocessorowe z podziałem czasu

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

## **Wprowadzenie do sieci komputerowych**

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

Standardy w Internecie.

Sieci LAN: topologie sieci LAN, pojęcie 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. (2000)

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 skosztować.

## 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;

jako rozkazów:

rej1 := *licznik*;

rej1 := rej1 + 1;

*licznik* := rej1;

*licznik* := *licznik* - 1;

jako rozkazów:

rej2 := *licznik*;

rej2 := rej2 - 1;

*licznik* := rej2;

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

T0: producent wykonuje	rej1 := <i>licznik</i> ;	(rej1=5)
T1: producent	rej1 := rej1 + 1 ;	(rej1=6)
T2: konsument	rej2 := <i>licznik</i> ;	(rej2=5)
T3: konsument	rej2 := rej2 - 1;	(rej2=4)
T4: producent	<i>licznik</i> := rej1;	( <i>licznik</i> =6)
T5: konsument	<i>licznik</i> := rej2;	( <i>licznik</i> =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.

WNIOSEK: **Konieczność synchronizacji.**

## Problem sekcji krytycznej

n współbieżnych procesów  $\{P_0, P_1, \dots, P_{n-1}\}$ ,

**sekcja krytyczna** - segment kodu (może wystąpić w każdym z tych procesów), w którym proces przetwarza wspólne zmienne,

**zasada wzajemnego wyłączenia 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 wyłączenie

Postęp

Ograniczone czekanie

## Założenia dot. konstrukcji algorytmów:

niezerowa prędkość wykonywania procesów

podstawowe rozkazy maszynowe wykonywane niepodzielnie

ogólna struktura rozpatrywanego procesu ma postać:

**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

```
repeat
    while numer  $\neq i$  do nic;
        sekcja krytyczna
    numer := j;
        reszta
until false;
```

### Algorytm 2

wspólna tablica *flaga*

```
var flaga: array[0..1] of boolean;
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  $P_i$ :

**repeat**

*flaga*[*i*] := *true*;

*numer* := *j*;

**while** (*flaga*[*j*] **and** *numer* = *j*) **do** *nic*;

        sekcja krytyczna

*flaga*[*i*] := *false*;

        reszta

**until** *false*;

## Sprzętowe środki synchronizacji

Rozkazy sprzętowe pozwalające na sprawdzenie i zmianę zawartości słowa (słów), wykonywane jako niepodzielne jednostki.

### Przykład rozkazu sprzętowego: instrukcja Testuj\_i\_Ustal

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

### Realizacja wzajemnego wyłączenia:

wspólna zmienna

var zamek: boolean;

wartość początkowa: zamek:= false;

### Proces $P_i$ ( $i=1,2$ ):

repeat

while Testuj\_i\_Ustal(zamek) do nic;

sekcja krytyczna

zamek:=false;

reszta

until false;

# Semafor

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

## Definicje operacji:

czekaj(S): **while**  $S \leq 0$  **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 wyłączenie*),  
początkowa wartość *wzajwy*=1,

każdy proces  $P_i$  jest 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 P1 z instrukcją S1, oraz P2 z instr. S2  
wymagane wykonanie S2 po wykonaniu S1.

### Rozwiązanie:

Inicjacja wspólnej zmiennej semafor: *synch*,  
wartość początkowa *synch* = 0,

dołączenie instrukcji w procesie P1:

.....

S1;

*sygnalizuj*(*synch*);

.....

w procesie P2:

.....

*czekaj*(*synch*);

S2;

## 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.

**Semafor:**            *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 wyłączonego dostępu pisarzy do obiektu.

Najprostszy wariant: żaden czytelnik nie powinien czekać, chyba, że pisarz uzyskał dostęp.

## Próba rozwiązania:

procesy czytelników dzielą zmienne:

**var** *wzajwy*, *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

. . .

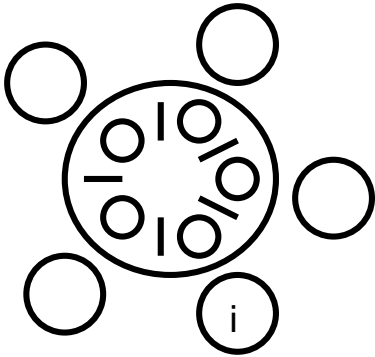
*czekaj*(*wzajwy*);

*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

. . .

sygnalizuj(pałeczka[i]);

sygnalizuj(pałeczka[i +1 **mod** 5]);

. . .

myślenie

. . .

**until** *false*;

# Implementacja semafora zliczającego S przy pomocy semaforów binarnych

## Deklaracje:

```
semafor_binarny S1, S2;  
int C;
```

**Wartości początkowe:**  $S1 = 1$ ,  $S2 = 0$ , zmiennej całkowita  $C$  = wartość początkowa semafora zliczającego S

## Implementacja operacji wait semafora zliczającego S:

```
wait(S1);  
C--;  
if (C < 0) {  
    signal(S1);  
    wait(S2);  
}  
signal(S1);
```

## Operacja signal semafora zliczającego S:

```
wait(S1);  
C++;  
if (C <= 0)  
    signal(S2);  
else  
    signal(S1);
```



## 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  $S_2, S_1$ .

## Implementacja Regionu krytycznego

1. Kompilator generuje zmienną dzieloną:

```
var v_wzajwy: semafor;  
warunek początkowy: v_wzajwy:=1;
```

2. Dla każdej instrukcji: **region** v **do** S;  
kompilator generuje kod zestawu instrukcji:

```
czekaj(v_wzajwy);  
S;  
sygnalizuj(v_wzajwy);
```

## Warunkowy region krytyczny

deklaracja:

**var** v: **shared** T

zmienna v dostępna jest w instrukcji:

**region** v **when** B **do** S;

gdzie B jest wyrażeniem boolowskim.

## Przykład ograniczonego buforowania

Deklaracje:

**var** bufor: **shared** record

pula: **array**[0..n-1] of jednostka;

licznik, we, wy: **integer**;

**end**;

Wartości początkowe: licznik:=we:=wy:=0;

### Kod producenta

**region** bufor **when** licznik < n

**do begin**

pula[we]:=nastp;

we:= we+1 mod n;

licznik:= licznik+1;

**end**;

### Kod konsumenta

**region** bufor **when** licznik > 0

**do begin**

nastk:= pula[wy];

wy:= wy+1 mod n;

licznik:= licznik-1;

**end**;

## Monitory

Monitor jest scharakteryzowany przez zbiór operacji zdefiniowanych przez programistę.

Reprezentacja typu monitor zawiera deklaracje zmiennych (ich wartości określają stan monitora), oraz procedury lub funkcje realizujące działania na monitorze.

Konstrukcja monitora gwarantuje, że w jego wnętrzu może być wykonywany tylko jeden proces. Inne czekają w kolejce.

Literatura:

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

A. Silberschatz, P.B. Galvin, G. Gagne; Podstawy systemów operacyjnych. WNT, W-wa, 2005.

## Uwaga

Monitor jest określony jako **typ** w rozumieniu programowania obiektowego: zarówno struktura danych i zbiór określonych na niej działań.

# KOMUNIKACJA MIĘDZY PROCESAMI

## Wprowadzenie

### Dwa uzupełniające się schematy komunikacji:

**pamięć dzielona** - procesy współużytkują 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,

jest dla każdej pary procesów tylko jedno,

jest dwukierunkowe

## (Komunikacja bezpośrednia - kontynuacja)

### 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 łączy:

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**

Pojemność łącza

- tylko określona liczba komunikatów może w nim przebywać,
- kolejka komunikatów przyporządkowanych do łącza.

## **Sposoby implementacji kolejki – pojemność łącza**

pojemność zerowa

pojemność ograniczona

pojemność nieograniczona

## **Przykłady działań w sytuacjach wyjątkowych**

Zakończenie procesu biorącego udział w komunikacji

Utrata komunikatów

Zniekształcenie komunikatów

**WYŻSZA SZKOŁA INFORMATYKI STOSOWANEJ I ZARZĄDZANIA  
WYDZIAŁ INFORMATYKI**

## **WIELODOSTĘPNE SYSTEMY OPERACYJNE II**

### **Zagadnienia zaawansowane**

#### **CZĘŚĆ 2**

### **BLOKADY – ZAKLESZCZENIA PROCESÓW**

Semestr 4

**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**

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

### **Zasady korzystania z zasobów przez procesy**

- Zamówienie zasobu

- Użycie

- Zwolnienie zasobu

## **Def. Wzajemnego blokowania, zakleszczenia procesów (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, semaforey

## **WARUNKI KONIECZNE WYSTĄPIENIA BLOKADY**

### **Wzajemne wyłączenie**

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.

## **WARUNKI KONIECZNE WYSTĄPIENIA BLOKADY (c. d.)**

### **Brak wywłaszczeń**

Tylko proces przetrzymujący określony zasób, może ten zasób zwolnić.

### **Czekanie cykliczne**

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

## GRAF PRZYDZIAŁU ZASOBÓW

Graf skierowany opisujący stan systemu przydziału zasobów, umożliwia analizę sytuacji blokad.

### Zbiór wierzchołków $W$ składający się z podzbiorów:

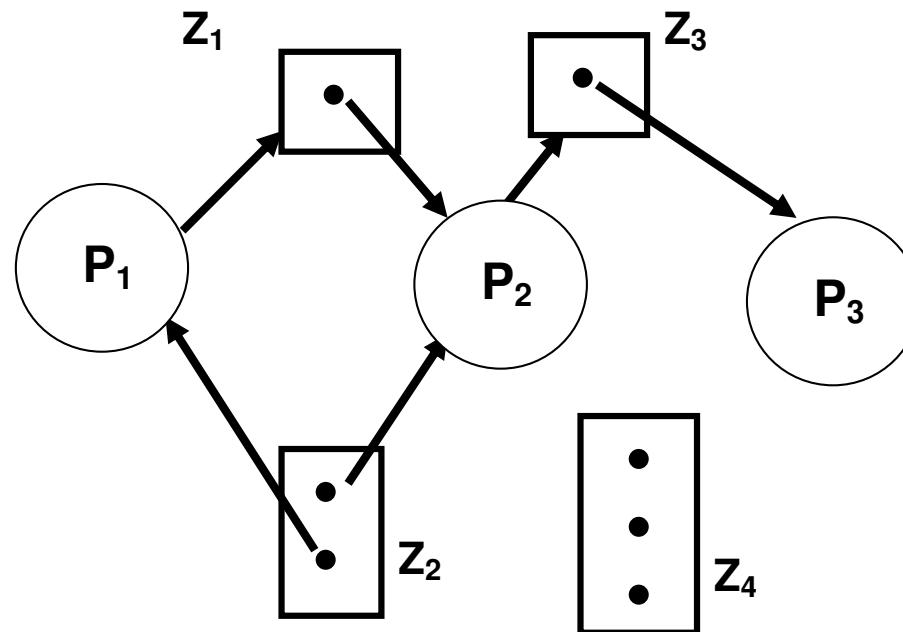
$P = \{ P_1, P_2, \dots, P_n \}$  podzbiór procesów,

$Z = \{ Z_1, Z_2, \dots, Z_m \}$  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  $P_j$ .



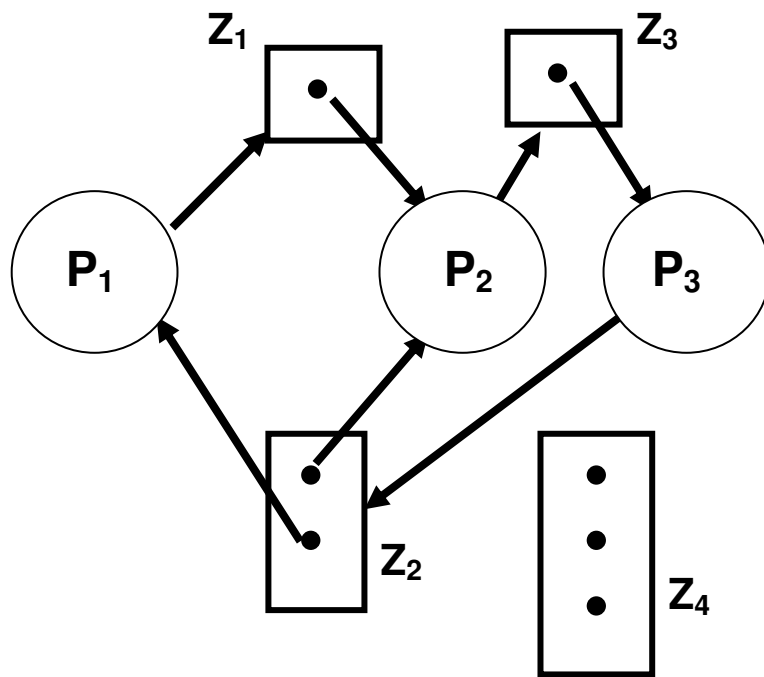
Przykład grafu (oznaczenia na rys. : proces -  $\bigcirc$  , zasób -  $\square$ ).



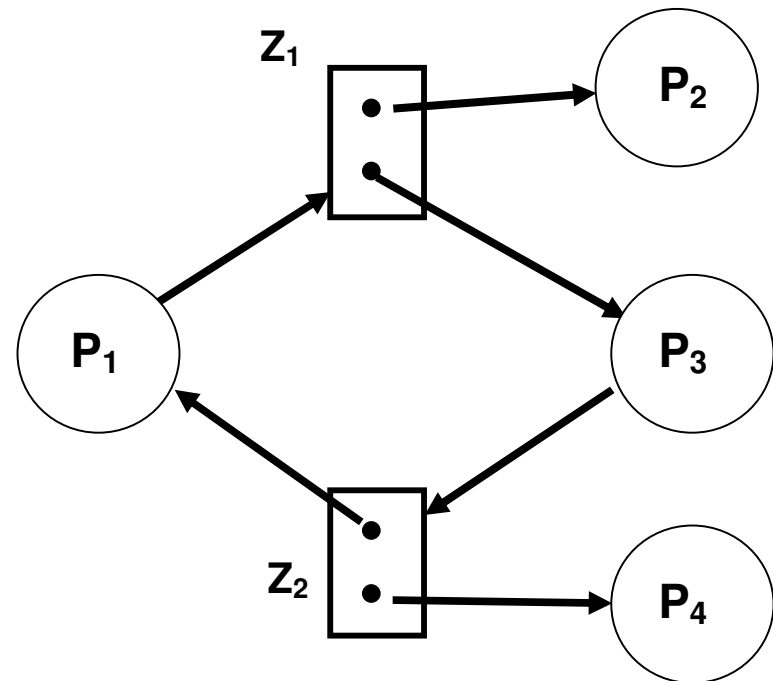
## Warunek konieczny wystąpienia blokowania:

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 wyłączenie**

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.

$Z = \{Z_1, \dots, Z_m\}$  zbiór zasobów

$F: Z \rightarrow N$  funkcja przyporządkowująca każdemu zasobowi liczbę naturalną ze zbioru  $N$ .

**Przykład:**

$Z = \{\text{nap. taśmy, nap. dysku, drukarki}\},$

$N = \{1, 5, 7\},$

$F(\text{nap. taśmy})=1, F(\text{nap. dysku})=5, F(\text{drukarki})=7,$

**Protokół:** każdy proces może zamawiać zasoby tylko we wzrastającym porządku ich numeracji, tzn. na początku proces może zamówić dowolną dostępną liczbę egz. zasobu np.  $Z_1$ .

Następnie jednak każdy egz. zasobu  $Z_k$  tylko wtedy, gdy  $F(Z_k) > F(Z_1)$ .

Wymaga się, aby proces zamawiający zasób  $Z_1$  miał wcześniej zwolnione zasoby  $Z_k$  takie, że  $F(Z_k) \geq F(Z_1)$ .

## 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.

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. Różne algorytmy wymagają różnych ilości i typów informacji.

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

**Stan bezpieczny:**

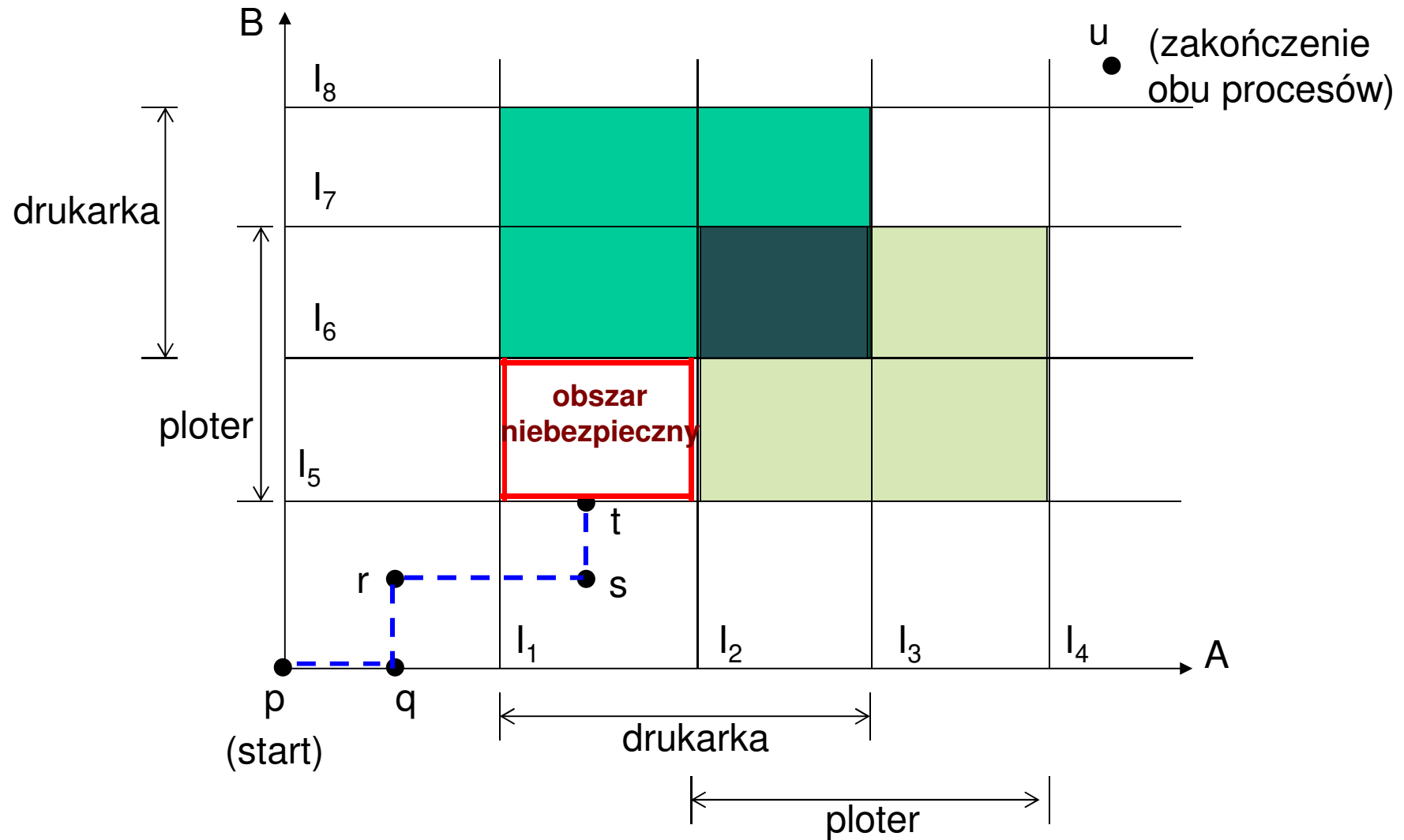
**Stan przydziałów** dla którego istnieje **ciąg bezpieczny** procesów  $P_1, \dots, P_n$ ,

tzn. taki ciąg, 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 bezpieczny nie istnieje.

## Ilustracja stanu bezpiecznego i stanu niebezpiecznego.

**Przykład 1:** trajektoria realizacji dwóch procesów: A i B wykorzystujących dwa zasoby



## Ilustracja stanu bezpiecznego i stanu zagrożenia. Przykład 2.

Jeden zasób dostępny w 10-u egzemplarzach, 3 procesy: A, B, C.

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(b)

	Has	Max
A	3	9
B	0	—
C	2	7

Free: 5

(c)

	Has	Max
A	3	9
B	0	—
C	7	7

Free: 0

(d)

	Has	Max
A	3	9
B	0	—
C	0	—

Free: 7

(e)

Stan (a) jest stanem bezpiecznym

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	0	—
C	2	7

Free: 4

(d)

Stan (b) nie jest stanem bezpiecznym



## **Rozwiązanie dla zasobów reprezentowanych wielokrotnie: algorytm „bankiera”**

Proces wchodzący do systemu musi zadeklarować maksymalną liczbę potrzebnych 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.

dla  $n$  procesów,  $m$  typów zasobów

**Dostępne:** array[0.. $m$ -1] of integer

określa liczbę dostępnych egz. każdego zasobu, np.

**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 formułowaniu algorytmów korzystamy dla uproszczenia zapisu z wektorów: **Dostępne**, **Maksymalne<sub>i</sub>**, **Przydzielone<sub>i</sub>**, **Potrzebne<sub>i</sub>** (wektory względem zasobów dla ustalonego procesu  $i$ ).

Wprowadzamy relację dominacji między wektorami  $X$ ,  $Y$  o wymiarach  $m$ :

$X \leq Y$  wtedy i tylko wtedy gdy  $X[j] \leq Y[j]$  dla każdego  $j=1, \dots, m$ .

Zamówienia procesu  $i$  na zasoby oznaczane są przez:  
**Zamówienia <sub>$i$</sub> : array[0.. $m$ -1] of integer**

### **ALGORYTM DZIAŁAŃ**

podejmowanych, gdy proces  $i$  wykonuje zamówienia

**Krok 1:** Sprawdź czy **Zamówienia <sub>$i$</sub>  ≤ Potrzebne <sub>$i$</sub>**   
**tak:** wykonaj **krok 2**,  
**nie:** warunek błędu.

**Krok 2:** Sprawdź czy **Zamówienia <sub>$i$</sub>  ≤ Dostępne**  
**tak:** wykonaj **krok 3**,  
**nie:** proces  $i$  musi czekać.

**Krok 3:** Wykonaj próbę przydziału zasobów:  
**Dostępne := Dostępne - Zamówienia <sub>$i$</sub> ;**  
**Przydzielone <sub>$i$</sub>  := Przydzielone <sub>$i$</sub>  + Zamówienia <sub>$i$</sub> ;**  
**Potrzebne <sub>$i$</sub>  := Potrzebne <sub>$i$</sub>  - Zamówienia <sub>$i$</sub> ;**

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 <sub>$i$</sub> .**

## ALGORYTM BEZPIECZEŃSTWA

Wprowadzamy wektory **Praca** o wymiarze  $m$ , **Koniec** o wymiarze  $n$ .

**Krok 1:** Przypisania początkowe:

**Praca := Dostępne;**

**Koniec[i] := false; dla  $i=0, \dots, n-1$ .**

**Krok 2:** Znajdź takie  $i$ , że jednocześnie:

**Koniec[i] = false and**

**Potrzebne<sub>i</sub> ≤ Praca**

czy istnieje takie  $i$ ?

**tak:** wykonaj **krok 3**

**nie:** wykonaj **krok 4**.

**Krok 3:** **Praca := Praca + Przydzielone<sub>i</sub>;**

**Koniec[i] := true;**

**wykonaj krok 2.**

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

## UNIKANIE ZAKLESZCZEŃ W PRZYPADKU ZASOBÓW POJEDYNCZYCH

Algorytm bankiera jest ogólny, jest jednak pracochłonny.

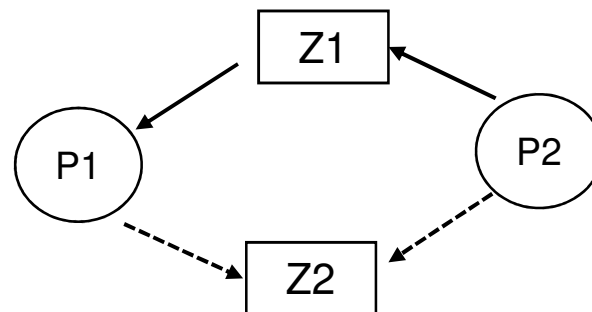
Wymaga do  $m \times n^2$  operacji, gdzie  $m$  – liczba zasobów,  $n$  – liczba procesów.

W przypadku zasobów pojedynczych można zdefiniować algorytm efektywniejszy.

Stosuje się wariant grafu przydziału zasobów, w którym są krawędzie zamówień i przydziałów, a dodatkowo krawędzie deklaracji.

Krawędź deklaracji oznacza, że proces zamówi określony zasób w przyszłości.

Przykład:



Zanim proces zacznie działać, wszystkie deklaracje muszą być znane.

Gdy proces zamawia deklarowany zasób, krawędź deklaracji zastępowana jest krawędzią zamówienia.

## Idea algorytmu

Niech proces  $P_i$  zamawia zasób  $Z_j$ . Zamówienie jest realizowane tylko wtedy, gdy zamiana krawędzi zamówienia na krawędź przydziału, nie spowoduje wystąpienia cyklu.

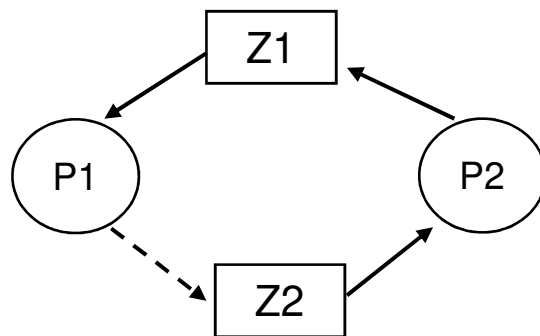
Sprawdzenie bezpieczeństwa polega na wykryciu cyklu w grafie - są odpowiednie algorytmy.

Jeśli nie ma cyklu, to realizacja zamówienia pozostawi system w stanie bezpiecznym.

Jeśli jest cykl, to realizacja zamówienia wprowadzi system w stan niebezpieczny. Proces  $P_i$  musi poczekać.

Ilustracja:

P1 i P2 zadeklarowały wcześniej zamówienie zasobu Z2. Gdy P2 go zamówił następuje zamiana deklaracji zamówienia na krawędź przydziału. Jest cykl w grafie. Nie należy realizować tego zamówienia.



# 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.

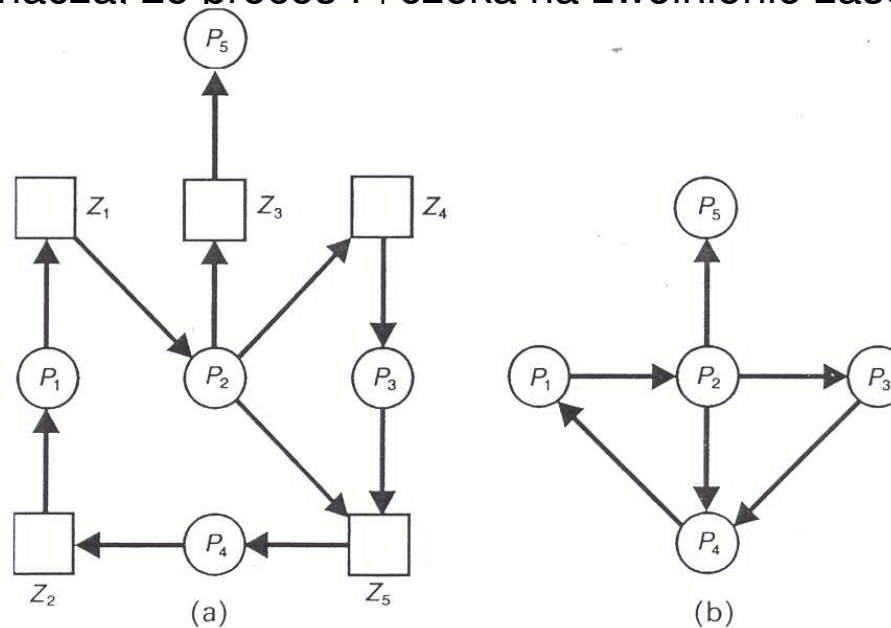
## Zasoby reprezentowane pojedynczo

idea algorytmu:

**Budowa grafu przydziałów**

**Budowa grafu oczekiwań** (przez usunięcie węzłów zasobów)

Krawędź  $P_i \rightarrow P_j$  oznacza, że proces  $P_i$  czeka na zwolnienie zasobów przez proces  $P_j$ .



**Wykrywanie blokady** polega na wykrywaniu pętli w grafie oczekiwania.

Niezbędne jest przechowywanie i aktualizowanie „na bieżąco” grafu oczekiwania oraz wykonywanie algorytmu szukającego pętli w grafie.

**Problem: 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

## **ROZWIĄZYWANIE PROBLEMU BLOKAD W PRAKTYCE**

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.

## Pytania podstawowe z zakresu blokad - zakleszczeń (deadlock) procesów przykłady

1. Co to jest blokada – zakleszczenie (deadlock) procesów?
2. Jakie są warunki konieczne wystąpienia blokady - zakleszczenia?
3. Co oznacza warunek czekania cyklicznego?
4. Na czym polega warunek przetrzymywania i oczekiwania?
5. Na czym polegają metody zapobiegania blokadom-zakleszczeniom?
6. W jaki sposób można wyeliminować warunek przetrzymywania i oczekiwania?
7. W jaki sposób można wyeliminować warunek braku wywłaszczeń?
8. W jaki sposób można wyeliminować warunek czekania cyklicznego?
9. Na czym polegają metody unikania blokad?
10. Jakie informacje są niezbędne do opisanie stanu systemu przydziału zasobów?
11. Co to jest stan bezpieczny?
12. Co to jest stan zagrożenia?
13. Podać ideę algorytmu bankiera.
14. Na czym polegają metody wykrywania i wychodzenia z blokady?
15. W jaki sposób można zidentyfikować stan blokady?
16. W jaki sposób można wyjść z istniejącej blokady i jakie wiążą się z tym koszty?

**WYŻSZA SZKOŁA INFORMATYKI STOSOWANEJ I ZARZĄDZANIA  
WYDZIAŁ INFORMATYKI**

## **WIELODOSTĘPNE SYSTEMY OPERACYJNE II**

### **Zagadnienia zaawansowane**

#### **CZĘŚĆ 3**

### **WPROWADZENIE DO SYSTEMÓW ROZPROSZONYCH**

Semestr 4

**Lech Kruś,**

# WPROWADZENIE 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

## **OBLICZENIA W CHMURZE (CLOUD COMPUTING)**

Synonim obliczeń rozproszonych realizowanych przez sieć komputerową, z możliwością wykonywania programu lub aplikacji na wielu połączonych komputerach w tym samym czasie.

Użytkownicy mogą się łączyć przez sieć z serwerem (realizowanym przez jedną lub grupę maszyn) i wykorzystywać jego zasoby: moc obliczeniową, pamięć operacyjną, przestrzeń dyskową,...

### **Oferowane modele usług w ramach cloud computing:**

#### **Infrastruktura jako usługa - Infrastructure as a service (IaaS)**

Dostawca oferuje dostęp do maszyn wirtualnych i innych zasobów.

#### **Platforma jako usługa - Platform as a service (PaaS)**

Oferta korzystania z platformy obliczeniowej obejmującej system operacyjny, środowisko języków programowania i wykonywania programów, bazy danych, serwer web-owy.

#### **Oprogramowanie jako usługa - Software as a service (SaaS)**

Dostawca instaluje i administruje oprogramowanie aplikacyjne w „chmurze” i udostępnia je klientom. Klienci nie muszą zarządzać infrastrukturą ani platformą obliczeniową.

#### **Bezpieczeństwo jako usługa - Security as a service (SECaaS)**

Model biznesowy, w którym duży dostawca usług integruje usługi bezpieczeństwa w infrastrukturze korporacyjnej – efektywniej niż mogą to zrealizować usługobiorcy samodzielnie.

## Pojęcia związane z „cloud computing”

### „Client – server model”

Model pracy rozproszonej aplikacji, w której rozróżnia się dostawcę usługi (serwer) oraz klienta/klientów .

### „Grid Computing” (obliczenia w „ Gridzie”)

Forma obliczeń rozproszonych, w której wirtualny superkomputer jest realizowany jako klaster komputerów luźno połączonych siecią, realizujących bardzo duże zadania obliczeniowe.

### „Utility computing”

Traktowanie zasobów komputerowych dostępnych przez sieć analogicznie jak inne media, takie jak energia elektryczna, woda, gaz.

### „Peer-to-peer”

Rozproszona architektura bez potrzeby centralnej koordynacji.

### „Cloud gaming”

Gry na żądanie. Forma dostarczania gier na komputery graczy. Dane procesu gry są przechowywane na serwerze dostawcy.



## **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**

**podział ze względu na budowę:**

wieloprocesory

multikomputery

**podział ze względu na architekturę sieci powiązań:**

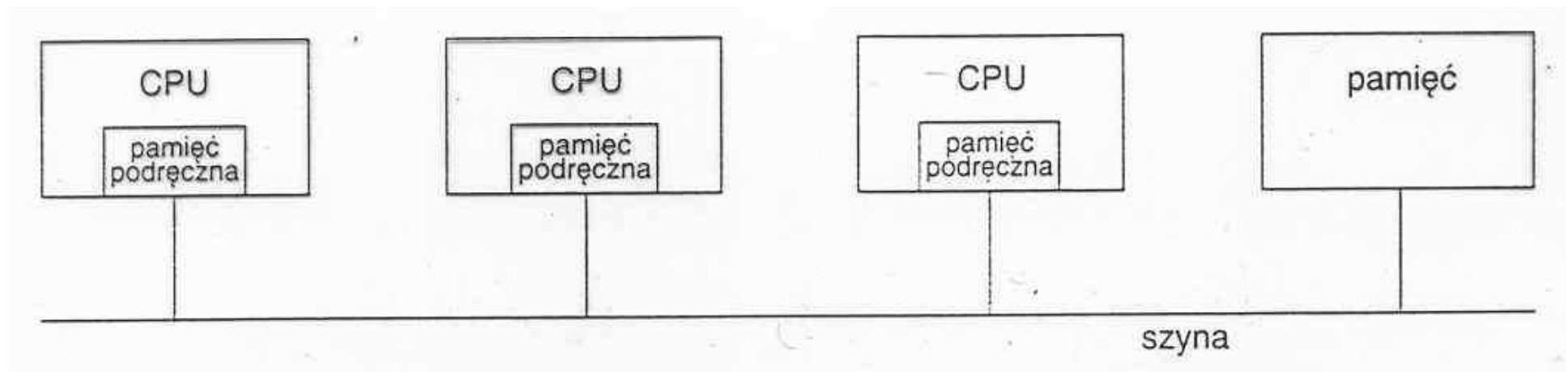
szynowe (bus)

przełączane (switched)

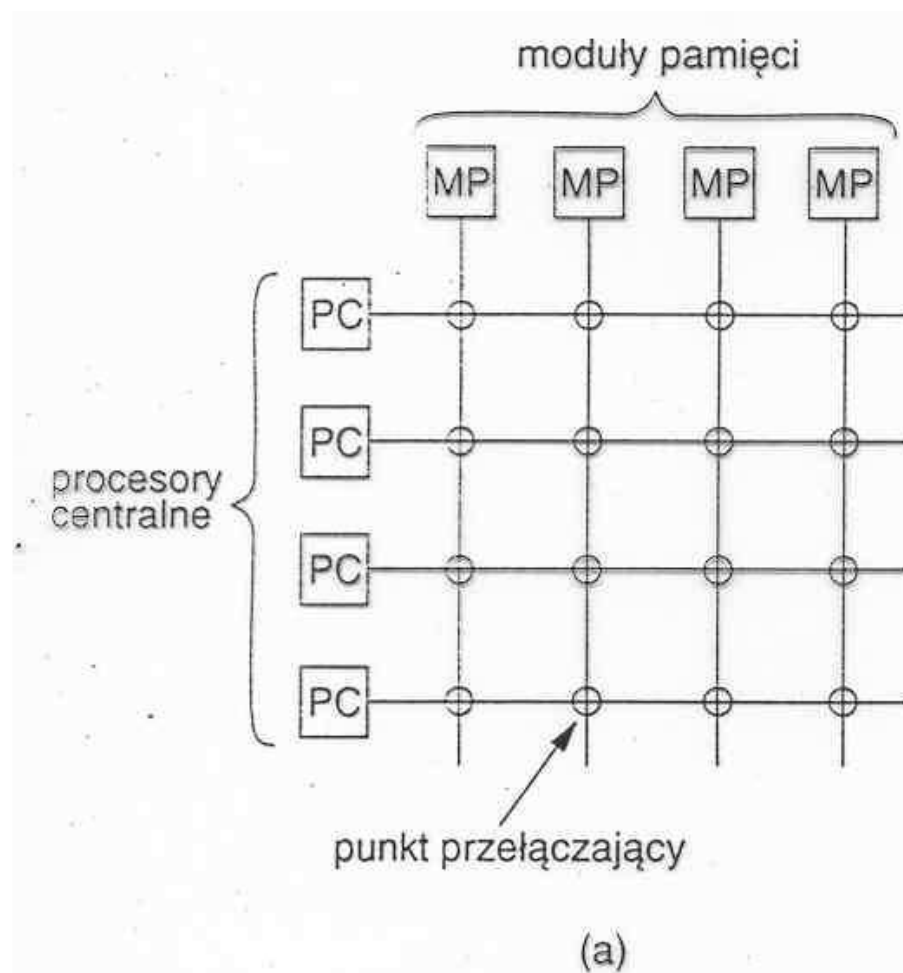
systemy ściśle powiązane ..

systemy luźno powiązane

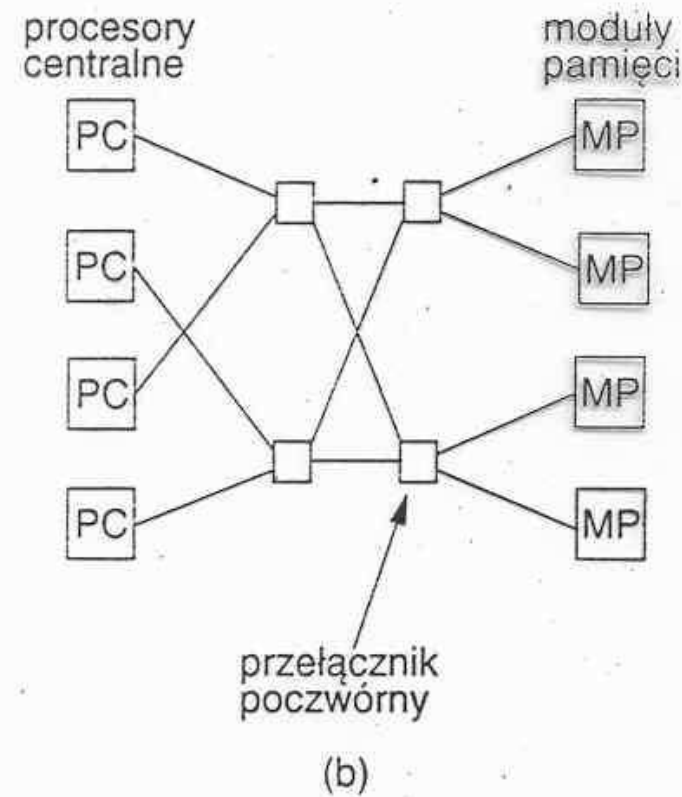
## WIELOPROCESORY SZYNOWE



## WIELOPROCESORY PRZEŁĄCZANE



Wybierak krzyżowy



Sieć przełączająca omega

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

## Problem opóźnienia w sieci Omega z poczwórnymi przełącznikami

$n$  procesorów,  $n$  modułów pamięci

liczba potrzebnych stopni przełączających:  $\log_2 n$ ,

w każdym stopniu potrzeba  $n/2$  przełączników

### Przykład

$n = 1024$ : potrzeba 10 stopni przełączających,

zamówienie od procesora do pamięci musi przejść 10 stopni przełączających,  
a wracające słowo - również 10 stopni przełączających.

Procesory RISC o szybkości 100 MIPS  $\Rightarrow$  czas wykonania instrukcji 10 ns.

Jeśli zamówienie ma przejść 20 stopni w czasie wykonywania instrukcji,  
to czas działania przełącznika  $\leq 0,5$  ns = 500 ps.

### **Zadanie A**

2048 procesorów RISC o szybkości 50 MIPS połączono w sieci Omega z poczwórnymi przełącznikami.

Jakie powinny być czasy przełączników, aby zamówienie do pamięci wróciło do procesora w czasie wykonywania jednej instrukcji?

Odpowiedź proszę uzasadnić.

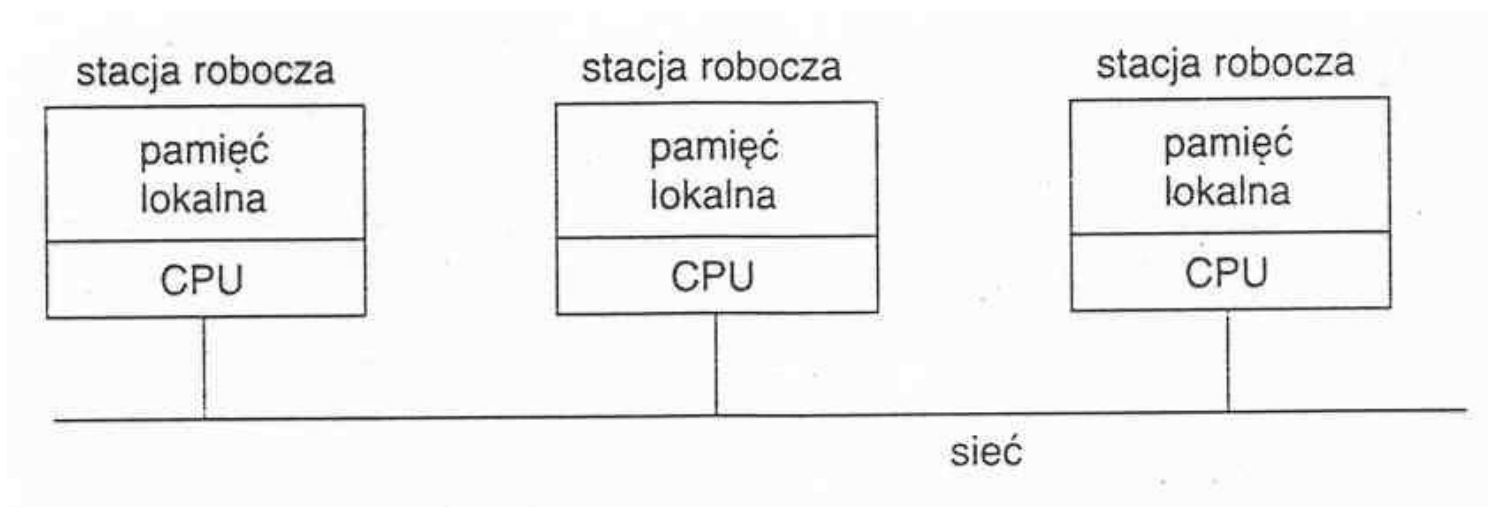
### **Zadanie B**

W wieloprocessorze zawierającym 4096 procesorów RISC połączonych w sieci Omega poczwórnymi przełącznikami zastosowano przełączniki o czasie działania 0,5 ns.

Jak szybkie mogą być procesory, aby zamówienie skierowane do pamięci wróciło do procesora w czasie wykonywania jednej instrukcji?

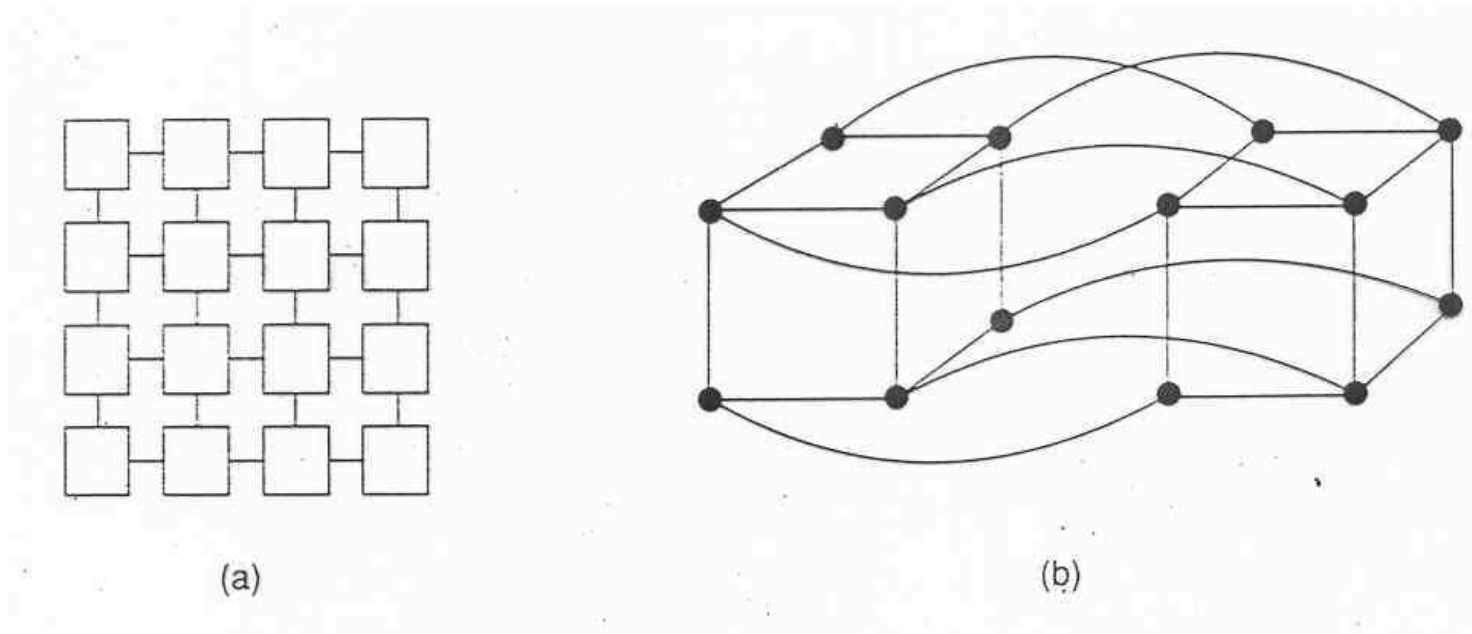
Wynik proszę podać w liczbie MIPS. Odpowiedź uzasadnić.

## MULTIKOMPUTERY SZYNOWE



Rys. 3. Multikomputer złożony ze stacji roboczych i sieci LAN

## MULTIKOMPUTERY PRZEŁĄCZANE



Rys. 4. (a) Krata. (b) Hiperkostka

# **OPROGRAMOWANIE**

## **Sieciowe systemy operacyjne**

Stacje robocze połączone siecią LAN

Każda maszyna ma własny system operacyjny

Podstawowe usługi:

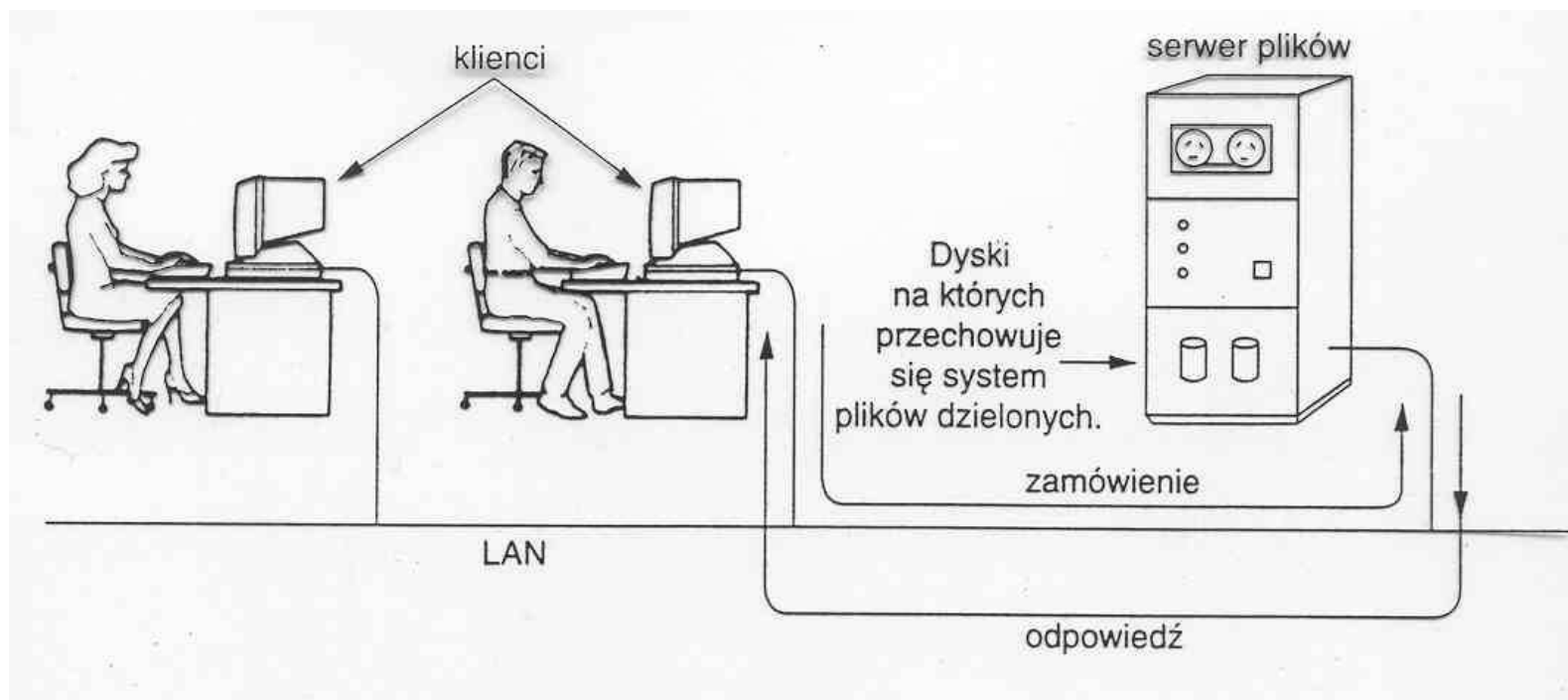
- logowanie na innej maszynie

- kopiowanie plików między maszynami

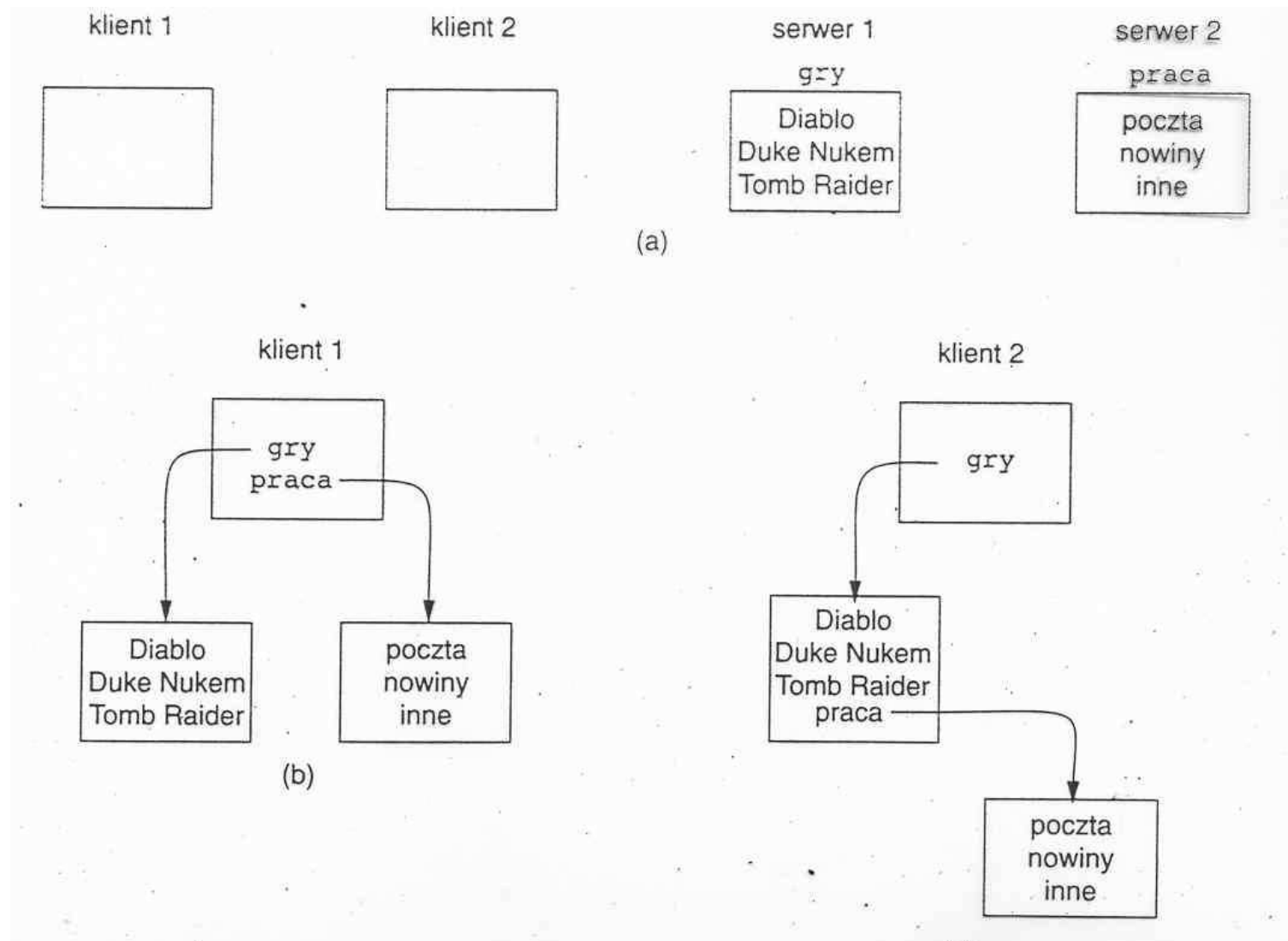
Serwer plików realizuje globalny system plików dzielonych

Program użytkowy wykonywany tylko na lokalnej maszynie





Rys.5. Dwaj klienci i serwer w sieciowym systemie operacyjnym



Rys. 6. Możliwy różny obraz zasobów widzianych przez klientów w systemie sieciowym

## **Prawdziwe systemy rozproszone**

Wiele komputerów połączonych siecią

Wrażenie jednolitego systemu (single system image)  
(wirtualny monoprocesor - virtual uniprocessor)

### **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 wieloprocessorowe z podziałem czasu

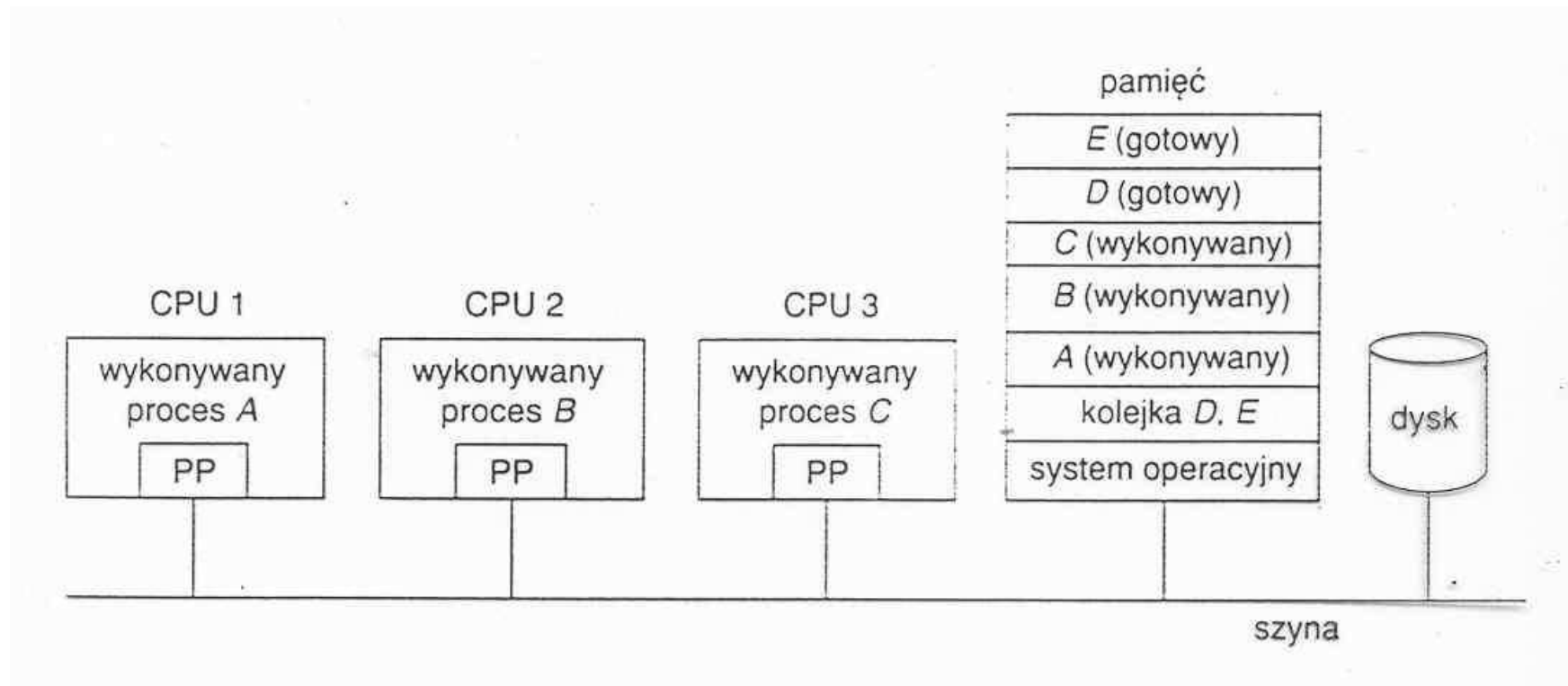
Wiele jednostek centralnych z pamięcią podręczną

Wspólna pamięć dzielona

Wspólny dysk (dyski)

Połączenie szyną.

Jedna kolejka uruchomień procesów



Rys. 7. Wieloprocessor z jedną kolejką uruchomień

## Porównanie klas systemów operacyjnych z wieloma (N) jednostkami centralnymi

Zagadnienie	Sieciowy system operacyjny	Rozproszony system operacyjny	Wieloprocessorowy 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 dostępu i położenia:** dostęp do lokalnych i zdalnych obiektów informacji za pomocą identycznych działań, bez znajomości ich lokalizacji.

**Przezroczystość wędrówki (migration transparency):** zasoby mogą być przemieszczane bez wpływu na działania użytkowników i programów użytkowych.

**Przezroczystość zwielokrotniania (replication transparency):** możliwość użycia wielu kopii obiektów informacji bez wiedzy użytkowników i programów użytkowych o zwielokrotnieniach.

**Przezroczystość współbieżności (concurrency transparency):** automatyczne, niezakłócone dzielenie zasobów między użytkowników działających współbieżnie.

**Przezroczystość działań równoległych (parallelism transparency):** zadania wykonywane równolegle bez wiedzy (konieczności działań) użytkowników

## Elastyczność

Dwie struktury systemów:

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

## Niezawodność - Tolerowanie awarii - Dostępność

### Integralność danych, Bezpieczeństwo

### Tolerowanie awarii

### Wydajność

### Skalowalność

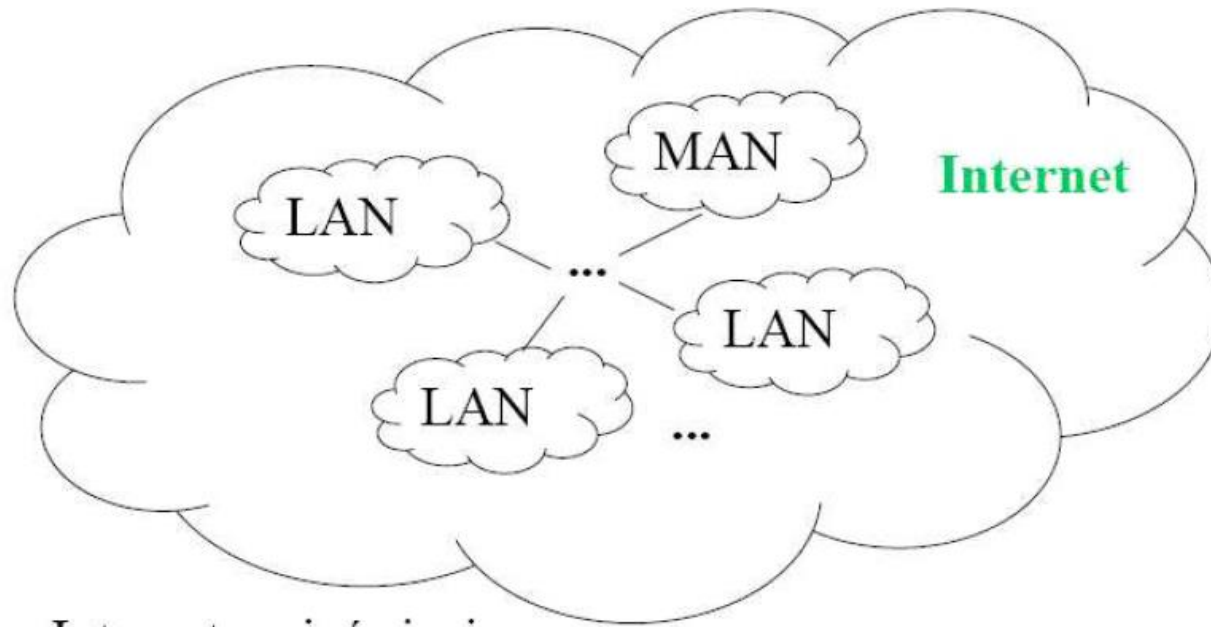
### Idea algorytmów zdecentralizowanych

## **Pytania podstawowe z zakresu wprowadzenia do systemów rozproszonych (przykłady)**

1. Co to jest system rozproszony?
2. Jakie są różnice między wieloprocessorami a multikomputerami?
3. Czym różni się architektura powiązań szynowych od przełączanych?
4. Co to jest szyna? Jak procesory korzystają z szyny porozumiewając się z pamięcią?
5. Wyjaśnić pojęcie spójności pamięci w wieloprocessorach.
6. Jakie właściwości muszą posiadać pamięci podręczne w wieloprocessorach, aby zapewnić spójność pamięci?
7. Czy wieloprocessory szynowe mogą być budowane z większej liczby procesorów niż przełączane, czy z mniejszej? Wyjaśnić, dlaczego?
8. Wyjaśnić ideę przełącznika krzyżowego stosowanego w wieloprocessorach.
9. Wyjaśnić ideę sieci "Omega" stosowaną w wieloprocessorach.
10. Czym różnią się prawdziwe systemy rozproszone od stosowanych obecnie powszechnie systemów sieciowych?
11. Jak działa system operacyjny w przypadku wieloprocessora?
12. Wyjaśnić pojęcie przezroczystości w systemach rozproszonych?
13. Co oznacza przezroczystość położenia (location transparency) w systemach rozproszonych?
14. Co oznacza przezroczystość zwielokrotnienia w systemach rozproszonych?
15. Co oznacza przezroczystość wędrówki - migracji (migration transparency) w systemach rozproszonych?
16. Jakie są dwie podstawowe koncepcje budowy operacyjnych systemów rozproszonych?



# Sieci komputerowe



- Internet = sieć sieci
- Problem – jak adresować urządzenia w takiej sieci?

## Podstawowe cechy odróżniające usługi podstawowe sieci TCP/IP od usług w innych sieciach

- Niezależność od techniki sieciowej (definicja wirtualnej jednostki transmisji danych – datagramu, sposoby przesyłania datagramu w poszczególnych sieciach)
- Jednolitość połączeń (możliwość komunikowania się dowolnych dwóch komputerów, adres logiczny – jednoznacznie przepisany do komputera, datagram – adres nadawcy i odbiorcy, pośrednie węzły używają adresu odbiorcy do wyznaczania trasy)
- Potwierdzenie na końcach (między nadawcą i odbiorcą zamiast potwierdzeń między kolejnymi komputerami)
- Standardy protokołów programów użytkowych

# Datagram IP i ramka (ethernetowa)



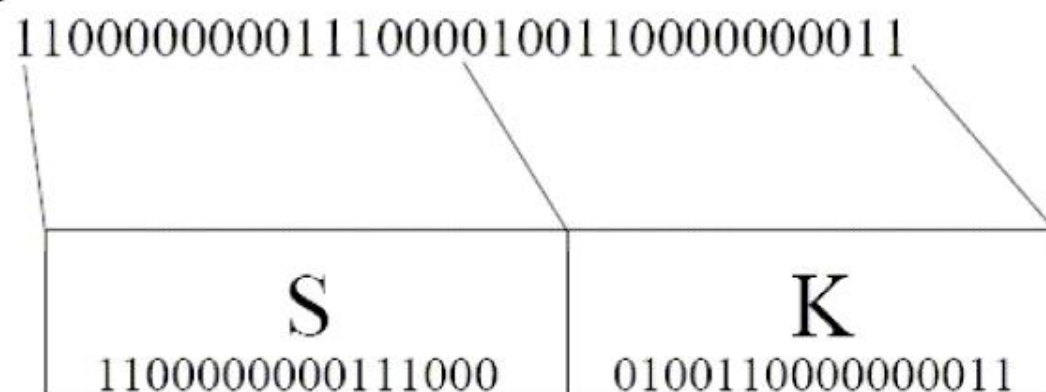
Bity						
1	4	8	16	19	24	32
Wersja	Długość	Rodzaj usługi	Długość pakietu			
Identyfikator			Flagi	Przesunięcie pakietu		
Czas życia		Protokół	Suma kontrolna nagłówka			
Adres nadawcy						
Adres odbiorcy						

## Format ramki Ethernetowej

Preambuła	Adres odbiorcy	Adres nadawcy	Typ ramki	Dane	CRC
8	6	6	2	46–1500	4

# Budowa adresu IP

- rozmiar adresu IP: 4 bajty (32 bity)
- Adres IP jest hierarchiczny - pierwsza część określa numer sieci, a pozostałe bity - numer komputera wewnątrz tej sieci
- Przykład:



S – numer sieci (in. **część sieciowa**)

K – numer komputera w sieci (in. **część komputerowa**)



# Zapis adresu - notacja dziesiętna

- każdy bajt z osobna zostaje przekształcony do postaci dziesiętnej
- poszczególne liczby dziesiętne oddzielone są kropką
- Przykład:

11000000001110000100110000000011

11000000	00111000	01001100	00000011

192 . 56 . 76 . 3

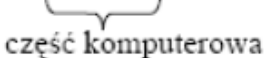
- zakres wartości dziesiętnej bajtu: 0 (00000000) – 255 (11111111)

# Adres IP sieci

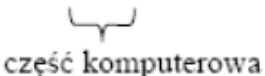
- nie tylko komputerom nadaje się adresy IP – posiadają je również sieci
- adres IP sieci charakteryzuje się tym, że część komputerowa składa się z samych zer

- Przykłady:

Założmy, że część sieciowa obejmuje 8 bitów. Wówczas przykładowym adresem sieci jest 120.0.0.0

część komputerowa

Gdyby część sieciowa zajmowała 16 bitów, wtedy przykładowym adresem sieci jest 150.150.0.0

część komputerowa

- proces nadawania adresów IP komputerom **zaczyna się od uzyskania adresu IP sieci**; numerację komputerów ustala administrator sieci

# Maska sieci

- określa proporcje między częściami sieciową i komputerową  
rozmiar części sieciowej w adresie komputera jest równy  
liczbie jedynek w masce
- służy do wyodrębnienia adresu IP sieci z adresu IP komputera  
odpowiadające sobie bity adresu komputera i maski są mnożone  
operacją AND
- tym samym „maskuje” (czyli zasłania, ignoruje) wartość części  
komputerowej adresu
- jest zawsze obecna, razem z adresem IP
- budowa i notacja maski są takie same, jak dla adresu IP (32 bity,  
notacja dziesiętna)
- maska ma charakterystyczną strukturę (zawartość) – tworzą ją  
dwa bloki: blok jedynek i blok zer: 111111....000000

# Obliczanie adresu IP sieci

- mnożenie operacją AND adresu IP komputera przez maskę

- Przykład:

adres komputera 150.150.10.10

maska 255.255.0.0

	10010110	10010110	00001010	00001010
AND	↑↑↑↑↑↑↑↑	↑↑↑↑	...	
	↓	↓	↓	↓
	11111111	11111111	00000000	00000000
Adres sieci	10010110	10010110	00000000	00000000

Dziesiętnie: 150.150.0.0



# Adresy szczególne

Poniższe adresy są szczególne, ponieważ nie można ich nadawać komputerom w sieci:

- adres sieci (część komputerowa złożona z samych zer)
- adres rozgłoszeniowy (część komputerowa złożona z samych jedynek) – pakiet wysłany pod ten adres ma być odebrany przez wszystkie komputery w danej sieci

Numery komputerów mieszczą się zawsze między tymi dwoma skrajami.