Akademia Nauk Stosowanych
Wydział Nauk Inżynieryjnych
Kierunek: Informatyka
studia I stopnia, semestr 2

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Systemy operacyjne

WYKŁAD 7 i 8

dr inż. Stanisława Plichta

splichta@ans-ns.edu.pl

Atrybuty procesu w systemie Linux

PID - identyfikator procesu

PPID - identyfikator procesu rodzica

TTY - reprezentuje terminal procesu

RUID - rzeczywisty identyfikator użytkownika

EUID - efektywny identyfikator użytkownika

RGID - rzeczywisty identyfikator grupy użytkownika

EGID - efektywny identyfikator grupy użytkownika

PRI - aktualny priorytet procesu (obliczany dynamicznie)

liczba nice - liczba mająca wpływ na priorytet procesu

Atrybuty procesu w systemie Linux

SIZE - wielkość pamięci wirtualnej procesu

RSS - wielkość użytej pamięci rzeczywistej

STAT - aktualny stan procesu

- R - run
- S - sleep
- D - oczekujący na operację dyskową
- T - stopped
- Z - zombie

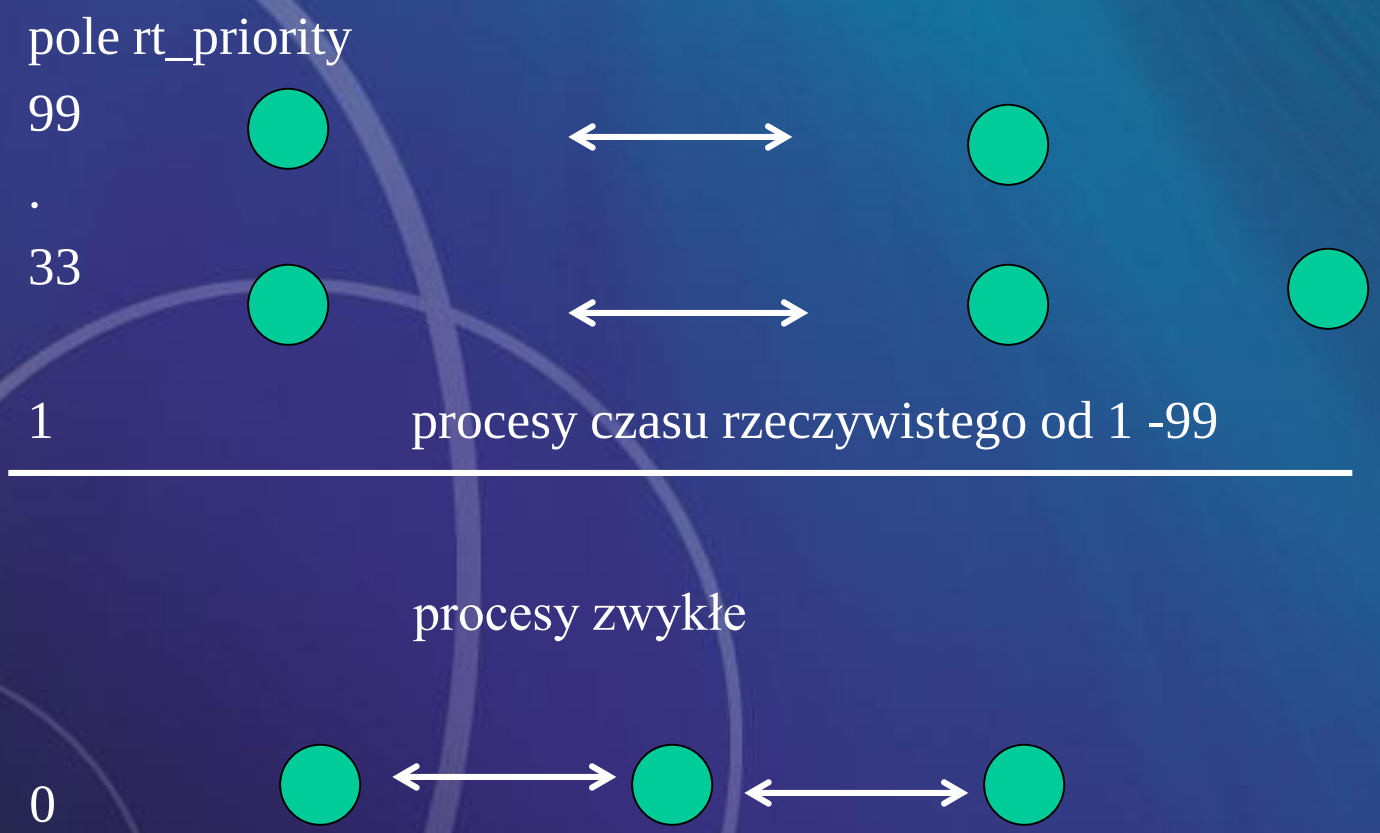
Pola używane do identyfikowania danego procesu

- PID - identyfikator procesu
- UID - rzeczywisty (wskazuje na użytkownika., który go uruchomił)
- EUID - efektywny (określa jakie prawa przysługują procesowi)
- SUID - zachowany (wskazuje właściciela pliku, z którego został załadowany kod procesu)
- FSUID - służy do określenia praw sprawdzanych przy dostępie do systemów plików
- GID, EGID, SGID, FSGID -mają podobne znaczenie z tą różnicą, że dotyczą odpowiednich grup.
- Informacja o tym do jakiej grupy należy użytkownik, jest przechowywana w tablicy groups[NGROUPS].

Rodzaje procesów w systemie LINUX

- Procesy zwykłe
- Procesy czasu rzeczywistego - muszą mieć dostępny do swojej dyspozycji procesor w ściśle określonych przedziałach czasowych

Zasada działania kolejek w systemie Linux



Szeregowanie procesów w linuxie

Scheduler wykorzystuje następujące pola ze struktury task_struct:

- **counter** - określa ile pozostało czasu procesora do wykorzystania przez dany proces.
- **priority** - służy do obliczania efektywnego priorytetu procesu
- **rt_priority** - przyjmuje wartości od 0 do 99 - oznacza, w której kolejce koncepcyjnej znajduje się dany proces.

zwykłe procesy - 0, procesy czasu rzeczywistego 1-99

- Kolejka nr 99 jest najbardziej uprzywilejowana.
- **policy** - określa tryb szeregowania.

Tryby szeregowania procesów w linuxie

- **SCHED_OTHER** - domyślna metoda - każdy nowo utworzony proces ma w polu policy domyślnie ustawioną tę wartość. W tym trybie szeregowania są zwykłe procesy o rt_priority=0 - wybór procesów z tej kolejki zależy od efektywnie obliczanego priorytetu.
- **SCHED_FIFO** - dotyczy szeregowania procesów czasu rzeczywistego

0<rt_priority<=99

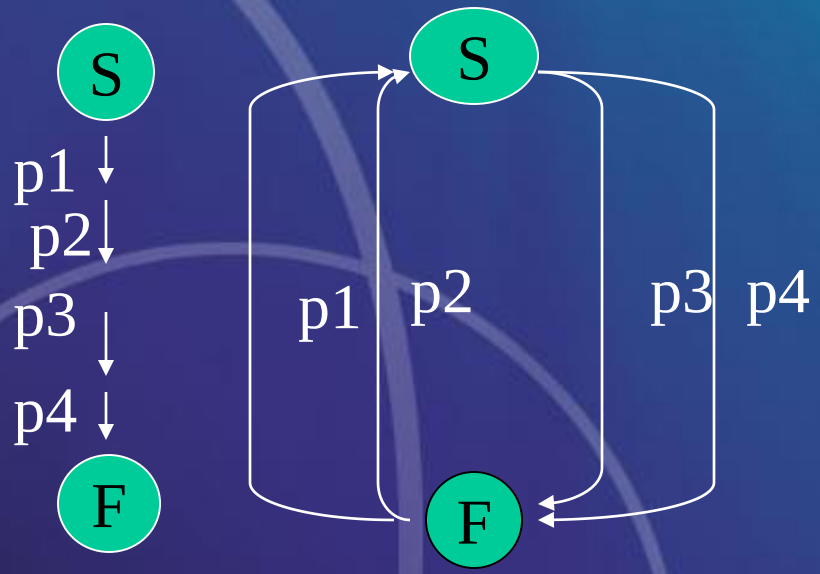
Wybierany jest proces pierwszy z danej kolejki, czas procesora przydzielany jest do chwili gdy sam proces nie zwolni procesora lub pojawi się inny proces bardziej uprzywilejowany.

- **SCHED_RR** - podobna do SCHED_FIFO, dla procesów czasu rzeczywistego - proces otrzymuje procesor na kwant czasu, po którym zostaje wywłaszczony i umieszczony na końcu danej kolejki.

OBLICZANIE EFEKTYWNEGO PRIORYTETU PROCESU

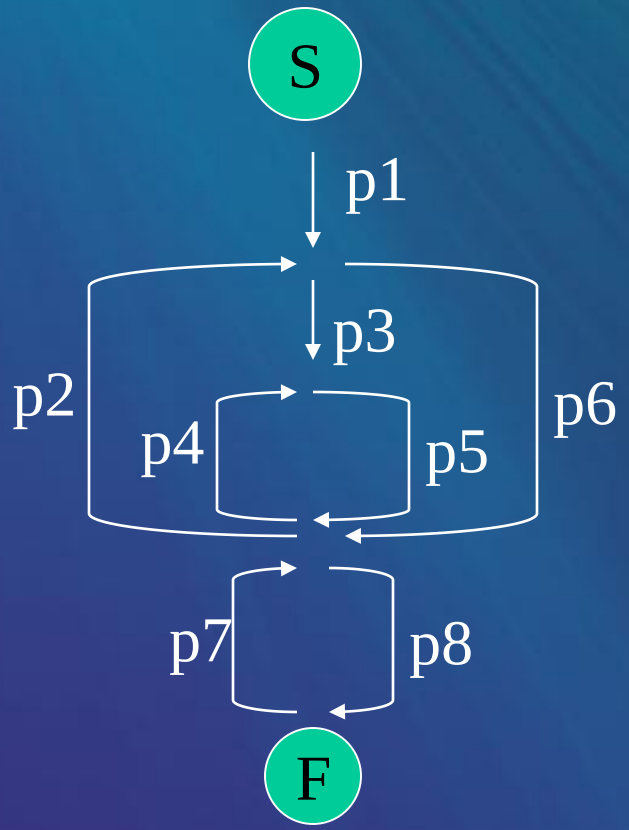
- Dla procesów gotowych nowa wartość counter=priority
- dla uśpionych counter=counter/2+priority (po przebudzeniu będą miały zwiększoną wartość counter)
- Funkcja goodness() - decyduje o wyborze procesu do wykonania - oblicza dla każdego procesu wartość wg wzoru:
- Jeśli policy=SCHED_RR lub SCHED_FIFO to zwraca wartość 1000+rt_priority.
- Jeśli policy=SCHED_OTHER to zwraca wartość counter
- Jeśli policy=SCHED_OTHER i jest to aktualnie wykonywany proces to zwraca wartość counter +1
- Oprócz wpisów w tablicy procesów utrzymywana jest lista wszystkich procesów next_task, prev_task, oraz lista procesów gotowych do wykonania next_run, prev_run.

Relacje pierwszeństwa między procesami



szeregowe

współbieżne



szeregowo-współbieżne

autor: dr inż. Stanisława Plichta

Problem sekcji krytycznej

PROCESY:
P1: .... x:=x+1;....
P2:.....x:=x+1;..... **x - wspólna zmienna**

Sekwencja1:
P1:R1:=x ; R1:=R1+1 ; x:=R1;.....
P2:.....R2:=x;.....R2:=R2+1; x:=R2....

Sekwencja2:
P1:R1:=x ; R1:=R1+1 ; x:=R1;.....
P2:.....R2:=x; R2:=R2+1; x:=R2

Mechanizmy synchronizacji procesów - semafory

- Rok 1965 - Dijkstra'e wprowadza pojęcia semaforów oraz operacji czekaj (wait) i sygnalizuj (signal)
- Semafor jest to nieujemna zmienna całkowita, na której za wyjątkiem nadawania wartości początkowych mogą działać jedynie operacje ***czekaj*** i ***sygnalizuj***

Operacje semaforowe

Operacja sygnalizuj

- powoduje zwiększenie wartości semafora s o 1,
- operacja jest niepodzielna.,
- wykonanie operacji sygnalizuj nie jest równoważne wykonaniu instrukcji przypisania s:=s+1.

Operacja czekaj

- powoduje zmniejszenie wartości semafora o 1, o ile wartość ta nie stałaby się ujemna po odjęciu jedynki,
- operacja jest niepodzielna - jeżeli kilka procesów było wstrzymanych, to po uzyskaniu przez semafor wartości dodatniej tylko jeden proces może zakończyć wykonywanie tej operacji,
- może spowodować wstrzymanie jakiegoś procesu, ponieważ jeśli dotyczy ona semafora mającego wartość 0, to proces, w którym ta operacja wystąpiła, będzie mógł być nadal wykonywany tylko wówczas, gdy inny proces zwiększy wartość semafora o 1 w wyniku operacji sygnalizuj.

Operacje semaforowe

- Zasoby niepodzielne trzeba chronić przed tym, aby równocześnie korzystało z nich kilka procesów
- Zapobiega się współbieżnemu wykonywaniu przez procesy tych fragmentów programu, poprzez które procesy uzyskują dostęp do zasobów - te fragmenty programu nazywa się *sekcjami krytycznymi*

Sekcję krytyczną programuje się w następujący sposób:

czekaj (nazwa semafora)

sekcja krytyczna

sygnalizuj (nazwa semafora)

Operacje semaforowe

- Dla zapewnienia takiego dostępu do sekcji krytycznej należy związać z nią zmienną typu semaforowego, która może przyjmować jedynie wartości całkowite nieujemne.
- Na zmiennej tej, zwanej semaforem są dopuszczalne wyłącznie dwie operacje

P - opuszczenie semafora

P(s) : wstrzymanie działania procesu, dopóki s nie stanie się większe od 0

s := s - 1

V - podniesienie semafora

V(s) : s := s + 1

Semafory binarne

SEMAFORY BINARNE

- Bardzo ważną odmianą uprzednio zdefiniowanych semaforów, są semafory binarne. Mogą one przyjmować dwie wartości zero lub jeden.
- Użycie semaforów binarnych pozwala na rozwiązanie problemów wzajemnego wykluczania.

ZASTOSOWANIE SEMAFORÓW

- Semafory są uniwersalnym mechanizmem, pozwalającym oprócz znanego już problemu wykluczenia wzajemnego rozwiązać wiele innych problemów synchronizacji i komunikacji.

Algorytm Dekkera

Mamy dwa procesy 1 i 2

zmienna p – numer procesu

tablica k - dwuelementowa 0, 1

kolejnosc =1 – rozstrzyga, który proces ustępuje drugiemu 1, 2

Sekcja wejściowa:

```
k [p] = 0
while (k [3 – p] = 0)
{
    if (kolejnosc <> p)
     {
          /* Ustąp drugiemu procesowi */
          k [p]= 1
          while (kolejnosc ≠ p)
          k [p] = 0
     }
}
```

Sekcja wyjściowa:

```
k [p] = 1
kolejnosc= 3 - p
```

Algorytm piekarniany dla n procesów

Mamy n procesów

tablica numerek – numer procesu (integer)

tablica wybor – n elementowa (boolean)

(numerek, numer procesu)

Sekcja wejściowa:

```
wybieranie [p] = true
 numerek [p] := max (numerek [1], numerek [2], ..., numerek [n]) + 1 wybieranie
[p] := false
for (j = 1 ; n )
{
    while (wybieranie [j] )
    while (numerek [j] <> 0 and (numerek [j], j) < (numerek [p], p))
}
```

Sekcja wyjściowa:

```
 numerek [p] = 0
```

Przykład

Mamy zbiór trzech sekwencyjnych procesów: P1, P2, P3.
Proces P1 składa się z trzech tasków:

t_{11} – trwającej 2 jednostki czasu,
t_{12} – trwającej 1 jednostkę czasu,
t_{13} – trwającej 1 jednostkę czasu,

Proces P2 składa się z dwóch akcji:

t_{21} – trwającej 1 jednostkę czasu,
t_{22} – trwającej 4 jednostki czasu,

Proces P3 składa się z dwóch akcji:

t_{31} – trwającej 1 jednostkę czasu,
t_{32} – trwającej 2 jednostki czasu.

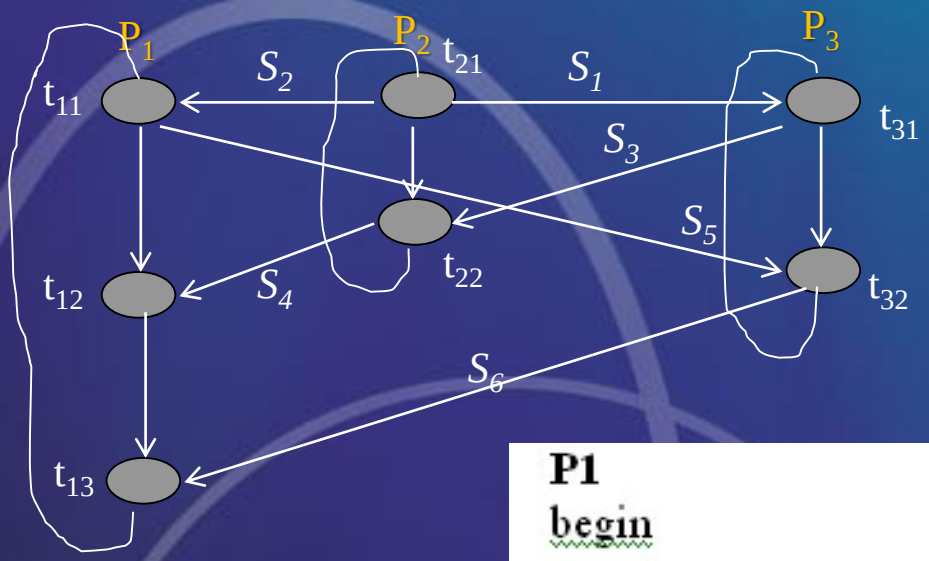
Kolejność czasowa akcji zdefiniowana jest zbiorem par G

G={(t21, t31), (t21, t11), (t31, t22), (t22, t12), (t11, t32), (t32, t13)}.

Jeśli para (**a,b**) należy do zbioru, to akcja **b** musi być poprzedzona akcją **a**.

- Narysuj graf pierwszeństwa oraz wykres czasowy.
- Przy użyciu semaforów zsynchronizuj procesy P1, P2, P3 .
- Rozwiązanie przedstaw w postaci pseudokodów dla P1, P2, P3.
- Ile czasu potrzeba na zakończenie procesów P1, P2, P3?
- Ile procesorów potrzeba na osiągnięcie takiego czasu?

Przykład

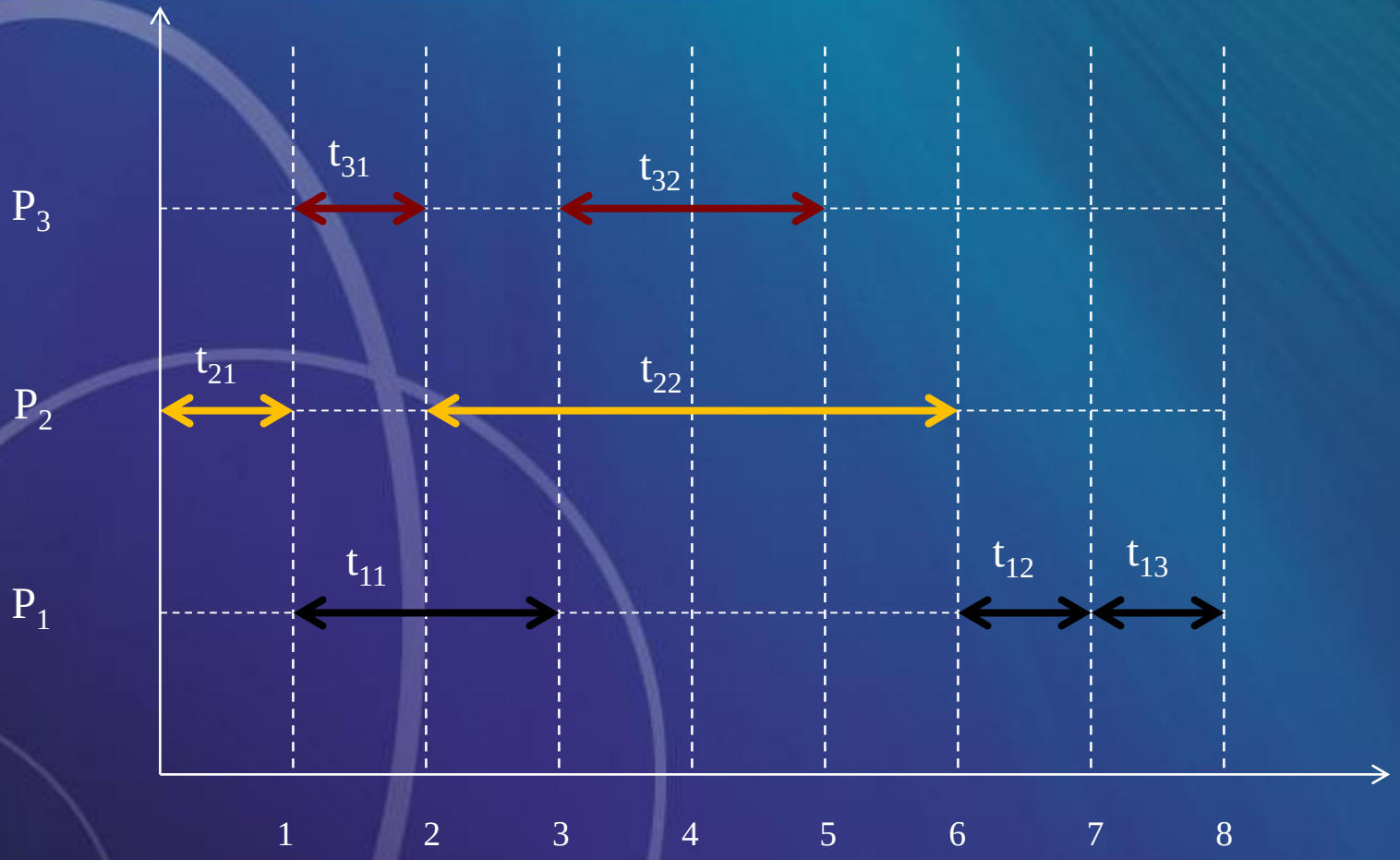


Wartości początkowe semaforów są równe 0

G={(t21, t31), (t21, t11), (t31, t22), (t22, t12), (t11, t32), (t32, t13)}.

P1	P2	P3
begin	begin	begin
repeat	repeat	repeat
wait(*S2*)	do t_{21}	wait(*S1*)
do t_{11}	signal(*S1*)	do t_{31}
signal(*S5*)	signal(*S2*)	signal(*S3*)
wait(*S4*)	wait(*S3*)	wait(*S5*)
do t_{12}	do t_{22}	do t_{32}
wait(*S6*)	signal(*S4*)	signal(*S6*)
do t_{13}	end	end
end		

Przykład



Inne mechanizmy synchronizacji

var R: **shared** *T*;
region R **do**
I1;...;In;
end;

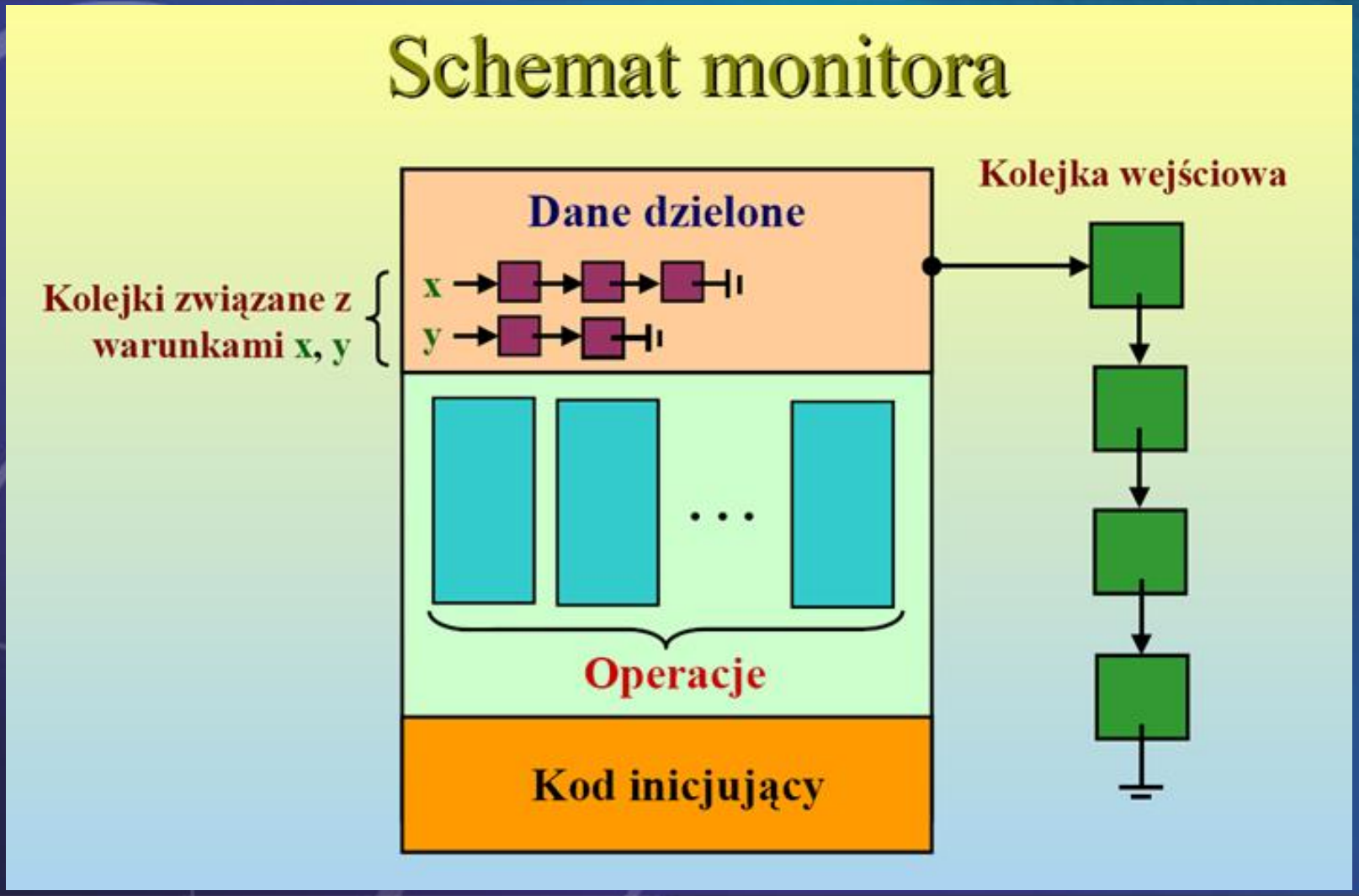
Warunkowe rejony krytyczne

var R: **shared** *T*;
region R **do**
$I_1;...;I_n$; await $W_{1;}$
.....
$I_i;...;I_{i+1}$; await $W_{j;}$
.....
end;

uproszczone postacie:

with R **when** W **do** I
region when W **do** I - sprawdzanie warunku przy wejściu
region R **do** I **await** W - sprawdzanie warunku przy wyjściu

Monitory


Schemat monitora
Kolejka wejściowa
Dane dzielone
Kolejki związane z warunkami x, y
x
y
Operacje
Kod inicjujący

Sprzętowe mechanizmy synchronizacji

- Klasyczne narzędzia programistyczne - aktywne oczekiwanie
- Komputer jednoprocesorowy - blokowanie przerwań
- Komputer wieloprocesorowy - blokowanie przerwań jest niewystarczające - potrzebne jest inne wsparcie sprzętowe
- Instrukcja procesora test-and-set (testuj i ustaw)

Sprzętowe mechanizmy synchronizacji

- Korzystając z instrukcji TestAndSet możemy rozwiązać problem sekcji krytycznej:

```
function TestAndSet (var x: boolean): boolean;
    begin
            TestAndSet := x;
            x := true
    end;
```

Sekcja wejściowa

```
while TestAndSet (S) do;
```

Sekcja wyjściowa

```
S := false;
```