Problem detekcji zakończenia

Przykład – sortowanie rozproszone

Rozważmy problem sortowania rozproszonego zbioru $\mathcal{X}$ składającego się z v różnych liczb naturalnych, w środowisku rozproszonym o n węzłach (procesorach), $n < v$.

- Zadaniem każdego procesu jest uporządkowanie przypisanej mu części zbioru liczb naturalnych i wyznaczenie elementu minimalnego
- Elementy minimalne są wysyłane do lewych sąsiadów.
- Po otrzymaniu wiadomości z wartością minimalną, proces wyznacza element maksymalny i wysyła go do prawego sąsiada.

Sortowanie rozproszone: definicje

Zbiór wstępnie podzielony na podzbiory $\mathcal{X}_i$

v_i – liczba elementów zbioru $\mathcal{X}_i$

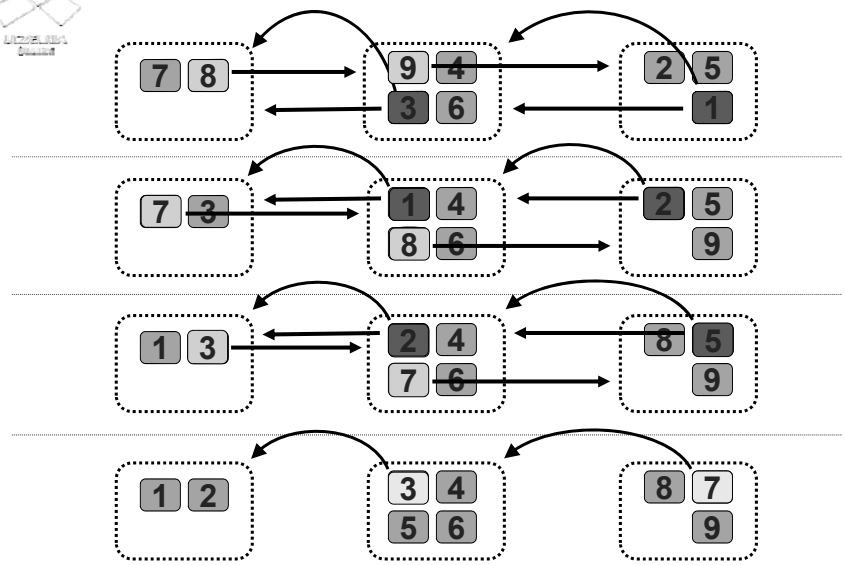
min_i – minimalny element zbioru $\mathcal{X}_i$

max_i – maksymalny element zbioru $\mathcal{X}_i$

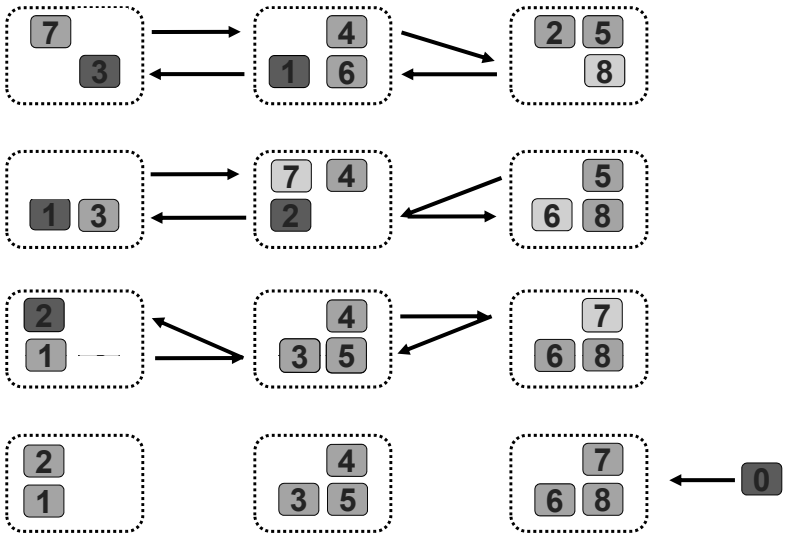
P_i – procesy tworzące przetwarzanie rozproszone topologii łańcucha skojarzone ze zbiorami $\mathcal{X}_{i.}$

Pary procesów składowych P_i, P_{i+1}, połączone są kanałami dwukierunkowymi.

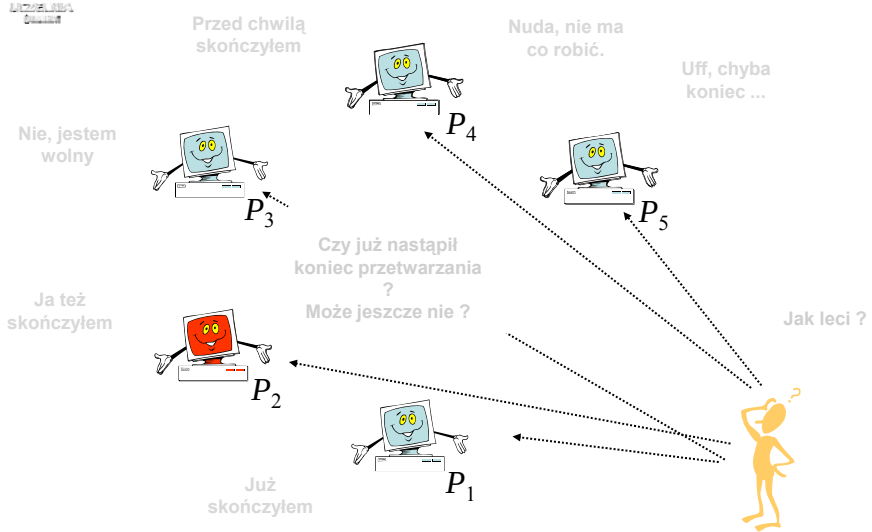
Sortowanie rozproszone – przykład



Sortowanie rozproszone – przykład (2)



Problem zakończenia



Definicja nieformalna zakończenia

Definicja nieformalna:

Nieformalnie problem detekcji zakończenia przetwarzania rozproszonego polega na sprawdzeniu, czy wszystkie procesy przetwarzania są w stanie pasywnym oraz czy żadna wiadomość będąca w kanale (transmitowana lub dostępna) nie uaktywni któregokolwiek z tych procesów.

Zakończenie dynamiczne

Nieformalnie, przetwarzanie rozproszone jest w stanie **zakończenia dynamicznego**, jeżeli żaden proces składowy przetwarzania rozproszonego nie będzie już nigdy uaktywniony. Stan ten będzie utrzymywany pomimo, że pewne wiadomości są wciąż transmitowane, a pewne wiadomości są już dostępne.

Zakończenie statyczne

Nieformalnie, przetwarzanie rozproszone jest w stanie **zakończenia statycznego**, jeżeli:

- wszystkie procesy są pasywne
- wszystkie wiadomości znajdujące się w kanałach są dostępne
- dla żadnego procesu nie jest spełniony warunek uaktywnienia

Klasyczna definicja zakończenia

W klasycznej definicji zakończenia przyjmowano, że przetwarzanie rozproszone jest w stanie zakończenia, jeżeli w danej chwili wszystkie procesy są pasywne i wszystkie kanały są puste.

Problem detekcji zakończenia

Problem detekcji zakończenia przetwarzania rozproszonego obejmującego zbiór procesów, sprowadza się do sprawdzenia czy przetwarzanie osiągnęło określony stan zakończenia.

Model przetwarzania synchronicznego

W modelu **przetwarzania synchronicznego** przyjmuje się, że transmisje są natychmiastowe. Stąd kanały mogą być uznane za puste przez cały czas i problem zakończenia sprowadza się do sprawdzenia czy wszystkie procesy są jednocześnie pasywne.

Detekcja zakończenia dla modelu synchronicznego

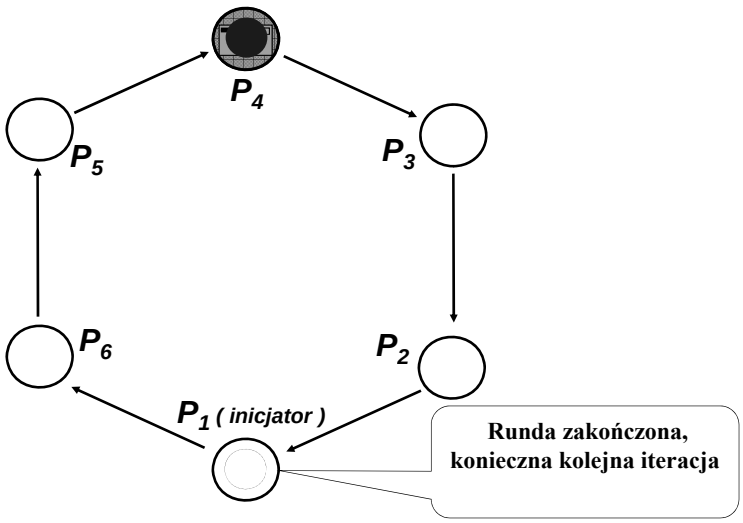
- Monitorom przypisany jest kolor: *White* lub *Black* (początkowo *White*)
- W pierścieniu przesyłany jest znacznik (z przypisanym kolorem)
- Początkowo monitory mają kolor *White*, a zmieniają kolor na *Black*, gdy odpowiadający im proces aplikacyjny wyśle wiadomości do procesu o indeksie większym.
- Znacznik jest przesyłany dalej, gdy obserwowany proces staje się pasywny
- Po wysłaniu znacznika monitorowi przypisywany jest kolor *White*
- Algorytm kończy się, gdy znacznik koloru *White* dotrze do inicjatora

Dla uproszczenia prezentacji, w algorytmie wykorzystano funkcje:

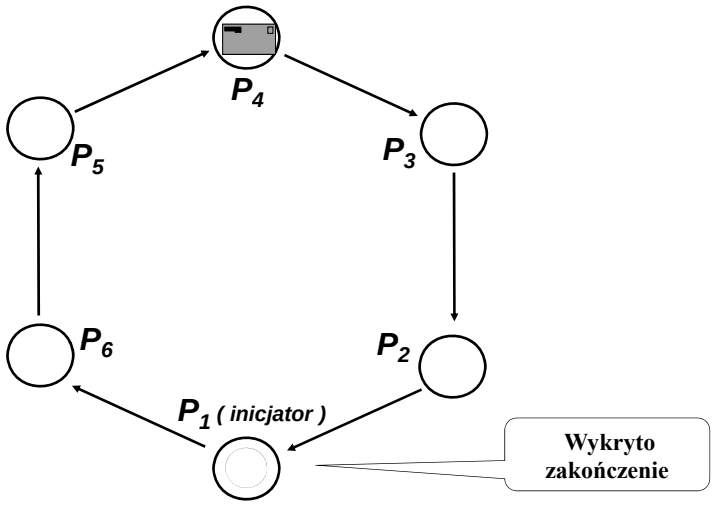
$succ(i) = (i) \ mod_n + 1$

$pred(i) = (i+n-2) \ mod_n + 1$

Przykład detekcji zakończenia runda zakończona niepowodzeniem



Przykład detekcji zakończenia runda zakończona sukcesem



Alg. Dijkstra, Feijen, van Gasteren (1)

```
type PACKET extends FRAME is record of
      data : MESSAGE
   end record

type TOKEN extends FRAME is record of
      colour : enum  { White, Black }
   end record
```

Alg. Dijkstra, Feijen, van Gasteren (2)

```
msgIn                : MESSAGE
pcktOut              : PACKET
tokenOut             : TOKEN
tokenPresent_i       : BOOLEAN := False
procColour_i         : enum {White, Black} := White
terminationDetected_i : BOOLEAN := False
```

Alg. Dijkstra, Feijen, van Gasteren (3)

```
1.  procedure InitProc()
2.     tokenOut.colour := White
3.     send(Q_α, Q_pred(α), tokenOut)
4.     tokenPresent_α := False
5.     procColour_α := White
6.  end procedure
```

Alg. Dijkstra, Feijen, van Gasteren (4)

```
7.  when e_start(Q_α, TerminationDetection) do
8.    wait until passive_α
9.    InitProc()
10. end when

11. when e_send(P_i, P_j, msgOut: MESSAGE) do
12.   if  i < j then
13.         procColour_i := Black
14.   end if
15.   pcktOut.data := msgOut
16.   send(Q_i, Q_j, pcktOut)
17. end when
```

Alg. Dijkstra, Feijen, van Gasteren (5)

```
18. when e_receive(Q_j, Q_i, pcktIn: PACKET) do
19.   msgIn := pcktIn.data
20.   deliver(P_j, P_i, msgIn)
21. end when
```

Alg. Dijkstra, Feijen, van Gasteren (6)

```
22.  when e_receive(Q_succ(i), Q_i, tokenIn: TOKEN) do
23.    tokenPresent_i := True
24.    wait until passive_i
25.    if i = α then
26.      if procColour_i = White ∧
                    tokenIn.colour = White then
27.          terminationDetected_i := True
28.          decide(terminationDetected_i)
29.      else
30.          InitProc()
31.      end if
```

Alg. Dijkstra, Feijen, van Gasteren (7)

```
32.    else
33.      tokenOut.colour := tokenIn.colour
34.      if procColour_i = Black then
35.          tokenOut.colour := Black
36.      end if
37.      send(Q_i, Q_pred(i), tokenOut)
38.      tokenPresent_i := False
39.      procColour_i := White
40.    end if
41.  end when
```

Model przetwarzania dyfuzyjnego

Przetwarzanie dyfuzyjne (ang. *diffusing computation*) jest specyficznym przetwarzaniem rozproszonym, w którym wyróżnia się:

- inicjatora – może w dowolnej chwili rozpocząć przetwarzanie dyfuzyjne wysyłając wiadomość aplikacyjną do jednego lub wielu procesów kooperujących
- hierarchię kooperujących procesów

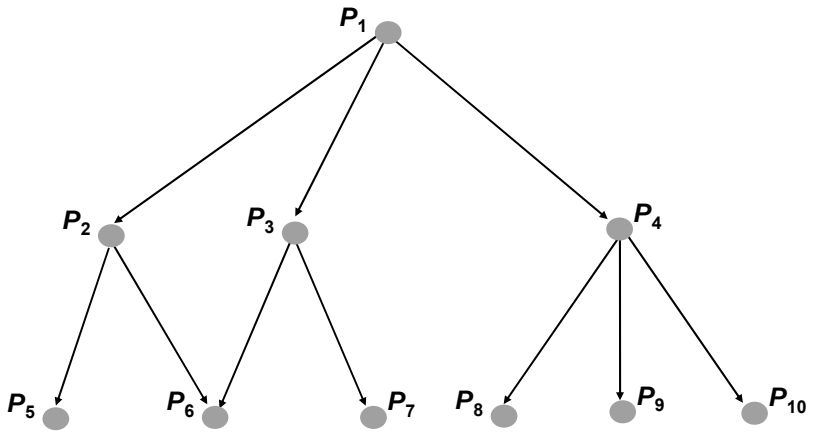
Założenia dodatkowe

- Proces aktywny staje się procesem pasywnym tylko w wyniku pewnego zdarzenia wewnętrznego
- Proces zawsze staje się aktywny po otrzymaniu wiadomości
- Proces pasywny może stać się aktywny **tylko** w wyniku otrzymaniu wiadomości

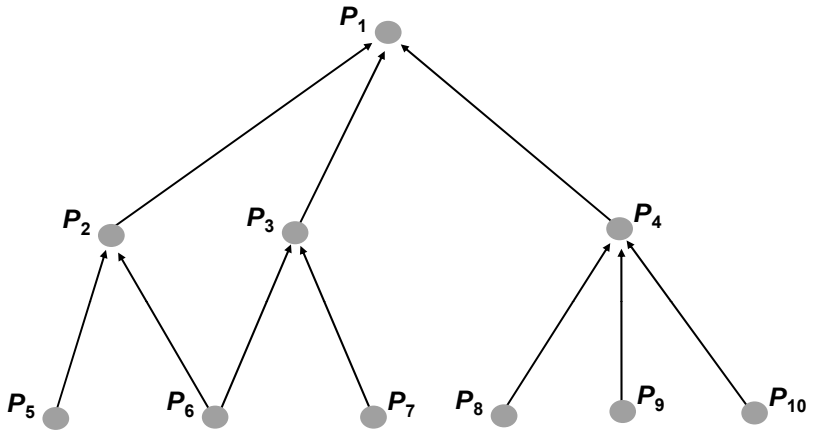
Algorytm Dijkstry-Scholten'a: Koncepcja

- Monitory procesów aplikacyjnych przesyłają wiadomości kontrolne jako pewnego rodzaju odpowiedzi na wiadomości aplikacyjne.
- Monitor pasywnego inicjatora może stwierdzić zakończenie przetwarzania po odebraniu wiadomości kontrolnych od wszystkich monitorów związanych z procesami uaktywnionymi przez inicjatora.

Graf przetwarzania dyfuzyjnego



Graf przetwarzania dyfuzyjnego (2)



Algorytm detekcji zakończenia dla modelu dyfuzyjnego (1)

```
type PACKET extends FRAME is record of
     data : MESSAGE
end record

type SIGNAL extends FRAME
```

Algorytm detekcji zakończenia dla modelu dyfuzyjnego (2)

```
msgIn                : MESSAGE
pcktOut              : PACKET
signalIn             : SIGNAL
engager_i            : PROCESS_ID
notEngager_i         : set of PROCESS_ID
recvNo_i             : INTEGER:= 0
sentNo_i             : INTEGER:= 0
terminationDetected_i : BOOLEAN:= False
```

Algorytm detekcji zakończenia dla modelu dyfuzyjnego (3)

```
1.  when e_start(P_α, 𝒫_α^R, msgOut: MESSAGE,
                 DiffusingComputation) do
2.    𝒬_α^R := {Q_j: P_j ∈ 𝒫_α^R }
3.    pcktOut.data := msgOut
4.    sentNo_α := |𝒬_α^R|
5.    send(Q_α, 𝒬_α^R, pcktOut)
6.  end when
```

Algorytm detekcji zakończenia dla modelu dyfuzyjnego (4)

```
7.  when e_send(Q_i, Q_j, signalOut : SIGNAL) do
8.    if recvNo_i=1 ∧ sentNo_i=0 ∧ passive_i
9.    then
10.     Q_j := engager_i
11.     send(Q_i, Q_j, signalOut)
12.   else
13.     for Q_j ∈ notEngager_i do
14.       notEngager_i := notEngager_i \ {Q_j}
15.       send(Q_i, Q_j, signalOut)
16.     end for
17.   end if
18.   recvNo_i:= recvNo_i - 1
19. end when
```

Algorytm detekcji zakończenia dla modelu dyfuzyjnego (5)

```
20. when e_receive(Q_j, Q_i, signalIn : SIGNAL) do
21.   sentNo_i:= sentNo_i - 1
22.   if Q_i=Q_α ∧ sentNo_i=0  then
23.     wait until passive_i
24.     terminationDetected_i:= True
25.     decide(terminationDetected_i)
26.   end if
27. end when
```

Algorytm detekcji zakończenia dla modelu dyfuzyjnego (6)

```
28.  when e_send(P_i, P_j, msgOut: MESSAGE) do
29.    pcktOut.data := msgOut
30.    sentNo_i := sentNo_i + 1
31.    send(Q_i, Q_j, pcktOut)
32.  end when
```

Algorytm detekcji zakończenia dla modelu dyfuzyjnego (7)

```
33.  when e_receive(Q_j, Q_i, pcktIn: PACKET) do
34.    if   recvNo_i = 0  then
35.           engager_i := Q_j
36.    else
37.           notEngager_i := notEngager_i ∪ {Q_j}
38.    end if
39.    recvNo_i := recvNo_i + 1
40.    msgIn := pcktIn.data
41.    deliver(P_j, P_i, msgIn)
42.  end when
```

Twierdzenie

Twierdzenie

Jeżeli przetwarzanie dyfuzyjne uległo zakończeniu, to fakt ten stwierdzi algorytm Dijkstry-Scholtena.

Algorytm detekcji zakończenia dla systemów asynchronicznych (Misra ’83): Założenia

- Brak założeń o topologii przetwarzania
- Brak założeń o czasie przesyłania wiadomości
- Niezawodna komunikacja
- Kanały FIFO
- Używa znacznika (ang. *token*)

Algorytm detekcji zakończenia dla systemów asynchronicznych (1)

```
type PACKET extends FRAME is record of
    data : MESSAGE
end record

type TOKEN extends FRAME is record of
    nb   : INTEGER
end record
```

Algorytm detekcji zakończenia dla systemów asynchronicznych (2)

```
msgIn                : MESSAGE
pcktOut              : PACKET
tokenOut             : TOKEN
tokenPresent_i       : BOOLEAN := False
succ_i               : PROCESS_ID
C                    : set of CHANNEL_ID
colour_i             : enum { White, Black } := Black
terminationDetected_i : BOOLEAN := False
```

Algorytm detekcji zakończenia dla systemów asynchronicznych (3)

```
1.  when e_start(P_α , TerminationDetection) do
2.    tokenOut.nb := 0
3.    send(Q_α, succ_α, tokenOut)
4.  end when
```

Algorytm detekcji zakończenia dla systemów asynchronicznych (4)

```
12. when e_receive(Q_j, Q_i, pcktIn: PACKET) do
13.   msgIn := pcktIn.data
14.   colour_i := Black
15.   passive_i := False
16.   deliver(P_j, P_i, msgIn)
17. end when
```

Algorytm detekcji zakończenia dla systemów asynchronicznych (5)

```
18. when e_receive(Q_j, Q_{i≠α}, tokenIn: TOKEN) do
19.   tokenPresent_i := True
20.   wait until passive_i = True
21.   if colour_i=Black then
22.      tokenOut.nb := 0
23.   else
24.      tokenOut.nb := tokenOut.nb + 1
25.   end if
26.   send(Q_i, succ_i, tokenOut)
27.   colour_i := White
28.   tokenPresent_i := False
29. end when
```

Algorytm detekcji zakończenia dla systemów asynchronicznych (6)

```
30. when e_receive(Q_j, Q_α, tokenIn: TOKEN) do
31.   tokenPresent_i := True
32.   if colour_i=White ∧ tokenOut.nb=|C| then
33.     terminationDetected_i := True
34.     decide(terminationDetected_i)
35.   else
36.     if colour_i = Black then
37.       tokenOut.nb := 0
38.     else
39.       tokenOut.nb := tokenOut.nb + 1
40.     end if
41.     send(Q_i, succ_i, tokenOut)
42.     colour_i := White
43.     tokenPresent_i := False
44.   end if
45. end when
```

Cechy algorytmu Misra’83

- Możliwość rozszerzenia do dowolnej topologii
- Wiele monitorów naraz może wykrywać zakończenia
- Cykl obejmujący wszystkie kanały komunikacyjne musi być znany z góry