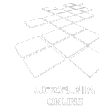


Detekcja zakleszczenia (1)

Wykład prowadzą:

Jerzy Brzeziński
Jacek Kobusiński

Przetwarzanie rozproszone



Plan wykładu

- Procesy aktywne i pasywne
- Definicja zakleszczenia
- Problem detekcji wystąpienia zakleszczenia
- Detekcja zakleszczenia dla modelu OR
- Detekcja zakleszczenia dla modelu AND
- Detekcja zakleszczenia w środowisku synchronicznym dla modelu „k spośród r”

Detekcja zakleszczenia 1 (2)

Przetwarzanie rozproszone



Wprowadzenie

Procesy tworzące przetwarzanie rozproszone komunikują się ze sobą za pomocą mechanizmu wymiany wiadomości, realizując wspólny cel przetwarzania. Jedne procesy wysyłają komunikaty zawierające żądania przydziału pewnych zasobów, inne – w odpowiedzi – przesyłają ewentualnie komunikaty potwierdzające przydział żądanych zasobów.

Detekcja zakleszczenia 1 (3)

Przetwarzanie rozproszone



Nieformalna definicja problemu

W przetwarzaniu rozproszonym może w ogólności wystąpić sytuacja, w której wszystkie procesy pewnego niepustego zbioru procesów oczekują na wiadomości (potwierdzające na przykład przydział zasobów) od innych procesów tego właśnie zbioru. Stan taki nazywany jest **zakleszczeniem rozproszonym** (ang. *distributed deadlock*).

Detekcja zakleszczenia 1 (4)

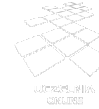


Procesy aktywne i pasywne

W każdej chwili proces może być w jednym z dwóch stanów:
aktywnym albo **pasywnym**.

Proces aktywny może realizować przetwarzanie wykonując operacje odpowiadające zajściu zdarzeń wewnętrznych i komunikacyjnych.

Detekcja zakleszczenia 1 (5)



Warunek uaktywnienia

Warunek uaktywnienia jest wyrażony przez predykat:

$$ready_i(\mathcal{X}) \equiv (\mathcal{P}_i^A \supseteq \mathcal{X}) \wedge activate_i(\mathcal{X}) \quad (5.1)$$

Detekcja zakleszczenia 1 (6)



Definicja problemu

Przez $deadlock(\mathcal{B})$ oznaczamy predykat stwierdzający, że w danej chwili τ , niepusty zbiór procesów \mathcal{B} jest zbiorem procesów zakleszczonych.

Detekcja zakleszczenia 1 (7)



Zakleszczenie w modelu jednostkowym

$$deadlock(\mathcal{B}) \equiv \quad (5.2)$$

$$(\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq \emptyset) \wedge$$

$$(\forall P_i :: P_i \in \mathcal{B} :: (passive_i \wedge |\mathcal{D}_i| = 1 \wedge$$

$$(\exists P_j :: P_j \in \mathcal{D}_i \cap \mathcal{B} :: (\neg in-transit_i[j] \wedge \neg available_i[j]))))$$

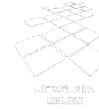
Detekcja zakleszczenia 1 (8)



Zakleszczenie w modelu AND

$$\begin{aligned}
 \text{deadlock}(\mathcal{B}) \equiv & \quad (5.3) \\
 & (\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq \emptyset) \wedge \\
 & (\forall P_i :: P_i \in \mathcal{B} :: (\text{passive}_i \wedge \\
 & \quad (\exists P_j :: P_j \in \mathcal{D}_i \cap \mathcal{B} :: (\neg \text{in-transit}_i[j] \wedge \neg \text{available}_i[j]))))
 \end{aligned}$$

Detekcja zakleszczenia 1 (9)



Zakleszczenie w modelu OR

$$\begin{aligned}
 \text{deadlock}(\mathcal{B}) \equiv & \quad (5.4) \\
 & (\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq \emptyset) \wedge \\
 & (\forall P_i :: P_i \in \mathcal{B} :: (\text{passive}_i \wedge \\
 & \quad \mathcal{D}_i \subseteq \mathcal{B} \wedge \\
 & \quad (\forall P_j :: P_j \in \mathcal{D}_i :: (\neg \text{in-transit}_i[j] \wedge \neg \text{available}_i[j]))))
 \end{aligned}$$

Detekcja zakleszczenia 1 (10)



Zakleszczenie w podstawowym modelu k spośród r

$$\begin{aligned}
 \text{deadlock}(\mathcal{B}) \equiv & \quad (5.5) \\
 & (\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq \emptyset) \wedge \\
 & (\forall P_i :: P_i \in \mathcal{B} :: (\text{passive}_i \wedge \\
 & \quad (\exists \mathcal{B}_i :: \mathcal{B}_i \subseteq \mathcal{D}_i \cap \mathcal{B} :: \\
 & \quad \quad (|\mathcal{D}_i \setminus \mathcal{B}_i| < k_i \wedge \\
 & \quad \quad (\forall P_j :: P_j \in \mathcal{B}_i :: (\neg \text{in-transit}_i[j] \wedge \neg \text{available}_i[j]))))))
 \end{aligned}$$

Detekcja zakleszczenia 1 (11)



Zakleszczenie w modelu OR – AND

$$\begin{aligned}
 \text{deadlock}(\mathcal{B}) \equiv & \quad (5.6) \\
 & (\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq \emptyset) \wedge \\
 & (\forall P_i :: P_i \in \mathcal{B} :: (\text{passive}_i \wedge \\
 & \quad (\forall u :: 1 \leq u \leq q_i :: \\
 & \quad \quad (\exists P_j :: P_j \in \mathcal{D}_i^u \cap \mathcal{B} :: (\neg \text{in-transit}_i[j] \wedge \neg \text{available}_i[j]))))))
 \end{aligned}$$

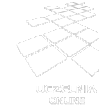
Detekcja zakleszczenia 1 (12)



Zakleszczenie w modelu dysjunkcyjnym k spośród r

$$\begin{aligned}
 \text{deadlock}(\mathcal{B}) \equiv & \quad (5.7) \\
 & (\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq \emptyset) \wedge \\
 & (\forall P_i :: P_i \in \mathcal{B} :: (\text{passive}_i \wedge (\forall u :: 1 \leq u \leq q_i :: \\
 & (\exists \mathcal{B}_i^u :: \mathcal{B}_i^u \subseteq \mathcal{D}_i^u \cap \mathcal{B} :: \\
 & (|\mathcal{D}_i^u \setminus \mathcal{B}_i^u| < k_i^u \wedge \\
 & (\forall P_j :: P_j \in \mathcal{B}_i^u :: (\neg \text{in-transit}_i[j] \wedge \neg \text{available}_i[j]))))))))
 \end{aligned}$$

Detekcja zakleszczenia 1 (13)



Zakleszczenie w modelu predykatowym

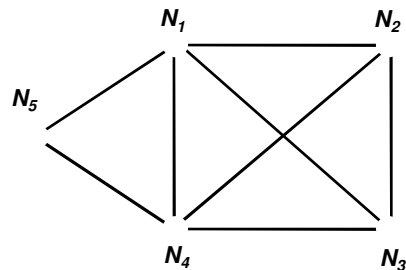
$$\begin{aligned}
 \text{deadlock}(\mathcal{B}) \equiv & \quad (5.8) \\
 & (\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq \emptyset) \wedge \\
 & (\forall P_i :: P_i \in \mathcal{B} :: (\text{passive}_i \wedge \neg \text{activate}_i(\mathcal{AV}_i \cup \mathcal{IT}_i \cup (\mathcal{P} \setminus \mathcal{B}))))
 \end{aligned}$$

Detekcja zakleszczenia 1 (14)



Przykłady zakleszczeń

Wait-For Graph (WFG):



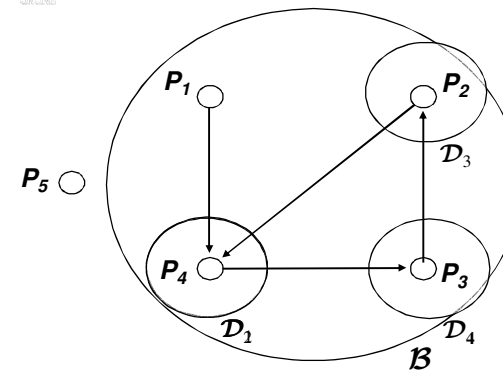
N_1, N_2, N_3, N_4, N_5 – węzły środowiska przetwarzania
 P_1, P_2, P_3, P_4, P_5 – procesy wykonywane w odpowiednich węzłach
 $P_i \rightarrow P_j$ – łuk $\langle P_i, P_j \rangle$ reprezentujący fakt, że proces P_i oczekuje na wiadomość od procesu P_j

- Każdy proces w grafie z łukiem wychodzącym jest pasywny.
- Procesy bez łuków wychodzących są aktywne.
- Zakładamy, że wszystkie kanały są puste.

Detekcja zakleszczenia 1 (15)



Przykład – model jednostkowy



Zbiory warunkujące:

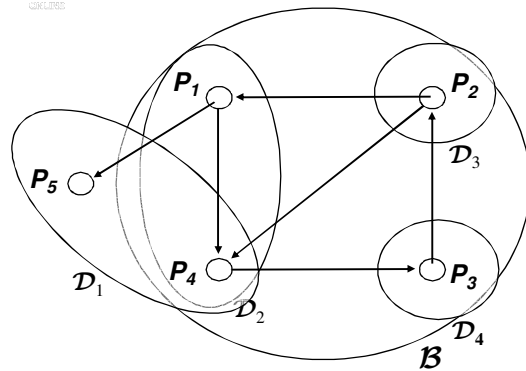
- $\mathcal{D}_1 = \{P_4\}$,
- $\mathcal{D}_2 = \{P_4\}$,
- $\mathcal{D}_3 = \{P_2\}$,
- $\mathcal{D}_4 = \{P_3\}$.

$\text{deadlock}(\mathcal{B})$ zachodzi dla $\mathcal{B} = \{P_1, P_2, P_3, P_4\}$

Detekcja zakleszczenia 1 (16)



Przykład – model AND

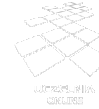


Zbiory warunkujące:

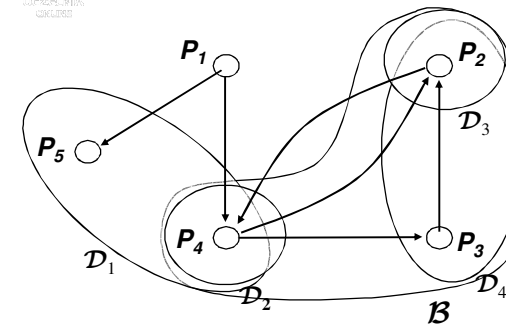
$$\begin{aligned} \mathcal{D}_1 &= \{P_4, P_5\}, \\ \mathcal{D}_2 &= \{P_1, P_4\}, \\ \mathcal{D}_3 &= \{P_2\}, \\ \mathcal{D}_4 &= \{P_3\}. \end{aligned}$$

$deadlock(\mathcal{B})$ zachodzi dla $\mathcal{B} = \{P_1, P_2, P_3, P_4\}$

Detekcja zakleszczenia 1 (17)



Przykład – model OR



Zbiory warunkujące:

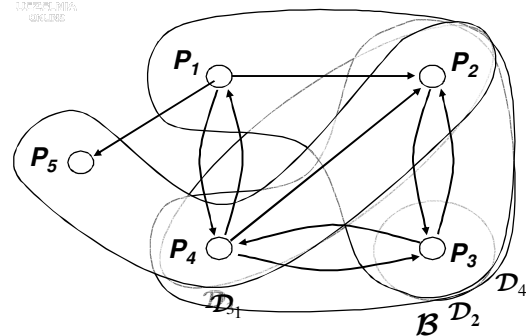
$$\begin{aligned} \mathcal{D}_1 &= \{P_4, P_5\}, \\ \mathcal{D}_2 &= \{P_4\}, \\ \mathcal{D}_3 &= \{P_2\}, \\ \mathcal{D}_4 &= \{P_2, P_3\}. \end{aligned}$$

$deadlock(\mathcal{B})$ zachodzi dla $\mathcal{B} = \{P_2, P_3, P_4\}$

Detekcja zakleszczenia 1 (18)



Przykład – model k spośród r



Modele żądań:

- dla P_1 : 1 spośród 3
- dla P_2 : 1 spośród 1
- dla P_3 : 2 spośród 2
- dla P_4 : 2 spośród 3

Zbiory warunkujące:

$$\begin{aligned} \mathcal{D}_1 &= \{P_2, P_4, P_5\}, \\ \mathcal{D}_2 &= \{P_3\}, \\ \mathcal{D}_3 &= \{P_2, P_4\}, \\ \mathcal{D}_4 &= \{P_1, P_2, P_3\}. \end{aligned}$$

$deadlock(\mathcal{B})$ zachodzi dla $\mathcal{B} = \{P_2, P_3, P_4\}$

Detekcja zakleszczenia 1 (19)



Klasyfikacja problemów detekcji zakleszczenia

- problem detekcji wystąpienia zakleszczenia
- problem detekcji zakleszczenia procesu
- problem detekcji zakleszczenia zbioru procesów
- problem detekcji maksymalnego zbioru procesów zakleszczonych

Detekcja zakleszczenia 1 (20)



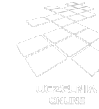
Detekcja wystąpienia zakleszczenia

Czy istnieje w pewnej chwili zbiór \mathcal{B} , dla którego predykat $deadlock(\mathcal{B})$ jest prawdziwy ?

Odpowiedź na to pytanie określa wartość predykatu:

$$dE \equiv (\exists \mathcal{B} :: deadlock(\mathcal{B})) \quad (5.9)$$

Detekcja zakleszczenia 1 (21)



Detekcja wystąpienia zakleszczenia procesu

Detekcja zakleszczenia procesu P_i sprowadza się do sprawdzenia czy prawdziwy jest predykat:

$$dP_i \equiv (\exists \mathcal{B} :: deadlock(\mathcal{B}) \wedge P_i \in \mathcal{B}) \quad (5.10)$$

Detekcja zakleszczenia 1 (22)



Detekcja wystąpienia zakleszczenia zbioru procesów

Detekcja zakleszczenia zbioru procesów polega na znalezieniu zbioru \mathcal{B}^* , dla którego prawdziwy jest następujący predykat:

$$deadlock(\mathcal{B}^*) \vee ((\mathcal{B}^* = \emptyset) \wedge (\nexists \mathcal{B} :: deadlock(\mathcal{B}))) \quad (5.11)$$

Detekcja zakleszczenia 1 (23)



Detekcja maksymalnego zbioru zakleszczonego

Detekcja maksymalnego zbioru zakleszczonego, sprowadza się do znalezienia takiego zbioru \mathcal{B}^* , dla którego spełniony jest warunek:

$$(deadlock(\mathcal{B}^*) \vee \mathcal{B}^* = \emptyset) \wedge (maxdead(\mathcal{B}^*)), \quad (5.12)$$

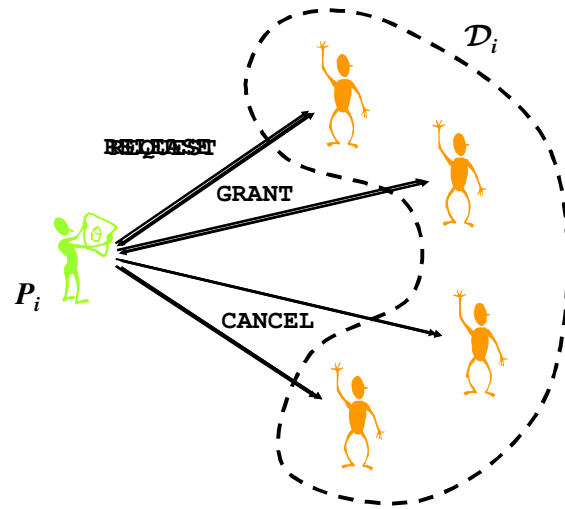
gdzie

$$maxdead(\mathcal{B}^*) \equiv (\forall \mathcal{B} :: deadlock(\mathcal{B}) \Rightarrow (\mathcal{B} \subseteq \mathcal{B}^*)) \quad (5.13)$$

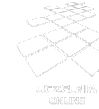
Detekcja zakleszczenia 1 (24)



Model aplikacyjnego przetwarzania rozproszonego



Detekcja zakleszczenia 1 (25)



Alg. Chandy, Misra, Hass dla modelu AND (1)

```

type PROBE extends FRAME is
record of
    initIndex : INTEGER
end record

probeOut    : PROBE
grantedi    : array [1..n] of BOOLEAN := False
 $\mathcal{D}_i$       : set of PROCESS_ID
recvProbei : array [1..n] of BOOLEAN := False
 $\alpha_i$       : INTEGER
k           : INTEGER
deadlockDetectedi : BOOLEAN := False
  
```

Detekcja zakleszczenia 1 (26)



Alg. Chandy, Misra, Hass dla modelu AND (2)

```

1. when e_start( $Q_\alpha$ , DeadlockDetection) do
2.   if passive\alpha
3.     then
4.       for all  $Q_k :: P_k \in \mathcal{D}_\alpha$  do
5.         probeOut.initIndex :=  $\alpha$ 
6.         send( $Q_\alpha$ ,  $Q_k$ , probeOut)
7.       end for
8.     end if
9.   end when
  
```

Detekcja zakleszczenia 1 (27)



Alg. Chandy, Misra, Hass dla modelu AND (3)

```

10. when e_activate( $P_i$ ) do
11.   for all  $k \in \{1, 2, \dots, n\}$  do
12.     recvProbei[k] := False
13.   end for
14. end when
  
```

Detekcja zakleszczenia 1 (28)



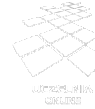
Alg. Chandy, Misra, Hass dla modelu AND (4)

```

15. when e_receive(Qj, Qi, probeIn: PROBE) do
16.   αi := probeIn.InitIndex
17.   if passivei ∧ (¬recvProbei[αi]) ∧ (¬grantedi[j])
18.   then
19.     recvProbei[αi] := True
20.     if αi = i
21.     then
22.       deadlockDetectedi := True
23.       decide (deadlockDetectedi)
24.     else
25.       probeOut.initIndex := αi
26.       for all Qk :: Pk ∈ Di do
27.         send(Qi, Qk, probeOut)
28.       end for
29.     end if
30.   end if
31. end when

```

Detekcja zakleszczenia 1 (29)



Detekcja zakleszczenia dla modelu OR (1)

Algorytm detekcji zakleszczenia dla modelu *OR* opiera się na **przetwarzaniu dyfuzyjnym** (ang. *query computation*).

Detekcja zakleszczenia 1 (30)



Detekcja zakleszczenia dla modelu OR (2)

Twierdzenie 5.2

Jeżeli inicjator Q_α rozpoczyna detekcję w chwili, gdy jego proces aplikacyjny P_α jest zakleszczony, to Q_α stwierdzi zakleszczenie procesu P_α w skończonym czasie (algorytm detekcji zakończy się).

Detekcja zakleszczenia 1 (31)



Detekcja zakleszczenia dla modelu OR (3)

Twierdzenie 5.2

Jeżeli inicjator Q_α deklaruje, że jego proces aplikacyjny P_α jest zakleszczony, to P_α należy do pewnego zbioru procesów zakleszczonych w chwili zakończenia algorytmu.

Detekcja zakleszczenia 1 (32)



Alg. Chandy, Misra, Hass dla modelu OR (1)

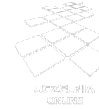
```

type CONTROL extends FRAME is
  record of
    initIndex : INTEGER
    queryNo   : INTEGER
  end record

type QUERY extends CONTROL
type REPLY extends CONTROL

```

Detekcja zakleszczenia 1 (33)



Alg. Chandy, Misra, Hass dla modelu OR (2)

```

queryOut       : QUERY
replyOut      : REPLY
 $\mathcal{D}_i$        : set of PROCESS_ID
maxQueryNoi  : array [1..n] of INTEGER := 0
engageri    : array [1..n] of INTEGER := 0
QRBalancei  : array [1..n] of INTEGER := 0
contPassivei : array [1..n] of BOOLEAN := False
queryNoi    : INTEGER
 $\alpha_i$        : INTEGER
deadlockDetectedi : BOOLEAN := False

```

Detekcja zakleszczenia 1 (34)



Alg. Chandy, Misra, Hass dla modelu OR (3)

```

1. when e_start( $Q_\alpha$ , DeadlockDetection) do
2.   if passive\alpha
3.     then
4.       maxQueryNo\alpha[ $\alpha$ ] := maxQueryNo\alpha[ $\alpha$ ] + 1
5.       contPassive\alpha[ $\alpha$ ] := True
6.       queryOut.queryNo := maxQueryNo\alpha[ $\alpha$ ]
7.       queryOut.initIndex :=  $\alpha$ 
8.       for all  $Q_k :: P_k \in \mathcal{D}_\alpha$  do
9.         send( $Q_\alpha$ ,  $Q_k$ , queryOut)
10.      end for
11.      QRBalance\alpha[ $\alpha$ ] := |  $\mathcal{D}_\alpha$  |
12.    end if
13.  end when

```

Detekcja zakleszczenia 1 (35)



Alg. Chandy, Misra, Hass dla modelu OR (4)

```

14. when e_receive( $Q_j$ ,  $Q_i$ , queryIn : QUERY) do
15.   if passivei
16.     then
17.       queryNoi := queryIn.queryNo
18.        $\alpha_i$  := queryIn.initIndex
19.       if queryNoi > maxQueryNoi[ $\alpha_i$ ]
20.         then
21.           maxQueryNoi[ $\alpha_i$ ] := queryNoi
22.           engageri[ $\alpha_i$ ] := j
23.           contPassivei[ $\alpha_i$ ] := True
24.           queryOut.queryNo := queryNoi
25.           queryOut.initIndex :=  $\alpha_i$ 
26.           for all  $Q_k :: P_k \in \mathcal{D}_i$  do
27.             send( $Q_i$ ,  $Q_k$ , queryOut)
28.           end for
29.           QRBalancei[ $\alpha_i$ ] := |  $\mathcal{D}_i$  |

```

Detekcja zakleszczenia 1 (36)



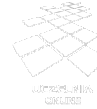
Alg. Chandy, Misra, Hass dla modelu OR (5)

```

30.  else
31.    if  $contPassive_i[\alpha_i] \wedge queryNo_i = maxQueryNo_i[\alpha_i]$ 
32.    then
33.       $replyOut.queryNo := queryNo_i$ 
34.       $replyOut.initIndex := \alpha_i$ 
35.      send( $Q_i, Q_j, replyOut$ )
36.    end if
37.  end if
38. end if
39. end when

```

Detekcja zakleszczenia 1 (37)



Alg. Chandy, Misra, Hass dla modelu OR (6)

```

40.  when  $e\_receive(Q_j, Q_i, replyIn : REPLY)$  do
41.    if  $passive_i$ 
42.    then
43.       $queryNo_i := queryIn.queryNo$ 
44.       $\alpha_i := queryIn.initIndex$ 
45.      if  $maxQueryNo_i[\alpha_i] = queryNo_i \wedge contPassive_i[\alpha_i]$ 
46.      then
47.         $QRBalance_i[\alpha_i] := QRBalance_i[\alpha_i] - 1$ 
48.        if  $QRBalance_i[\alpha_i] = 0$ 
49.        then
50.          if  $\alpha_i = i$ 
51.          then
52.             $deadlockDetected_i := True$ 
53.            decide( $deadlockDetected_i$ )

```

Detekcja zakleszczenia 1 (38)



Alg. Chandy, Misra, Hass dla modelu OR (7)

```

54.  else
55.     $k := engager_i[\alpha_i]$ 
56.     $replyOut.queryNo := queryNo_i$ 
57.     $replyOut.initIndex := \alpha_i$ 
58.    send( $Q_i, Q_k, replyOut$ )
59.  end if
60. end if
61. end if
62. end if
63. end when

```

Detekcja zakleszczenia 1 (39)



Alg. Chandy, Misra, Hass dla modelu OR (8)

```

64.  when  $e\_activate(P_i)$  do
65.    for all  $k \in \{1, 2, \dots, n\}$  do
66.       $contPassive_i[k] := False$ 
67.    end for
68.  end when

```

Detekcja zakleszczenia 1 (40)



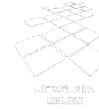
Alg. Bracha, Toueg'a (1)

```

type NOTIFY extends SIGNAL
type DONE extends SIGNAL
type CONFIRM extends SIGNAL
type ACK extends SIGNAL

```

Detekcja zakleszczenia 1 (41)



Alg. Bracha, Toueg'a (2)

```

notifyOut : NOTIFY
doneOut, doneIn : DONE
confirmOut : CONFIRM
ackOut, ackIn : ACK
A : set of record of
      predId : PROCESS_ID
      succId : PROCESS_ID
      end record := {  $\langle P_i, P_j \rangle : P_j \in \mathcal{D}_i$  }
OUTi : set of PROCESS_ID := {  $P_j : \langle P_i, P_j \rangle \in \mathcal{A}$  }
INi : set of PROCESS_ID := {  $P_j : \langle P_i, P_j \rangle \in \mathcal{A}$  }
confirmNoi : INTEGER := 0
notifiedi : BOOLEAN := False
confirmedi : BOOLEAN := False
expectNoi : INTEGER
deadlockDetectedi : BOOLEAN := False

```

Detekcja zakleszczenia 1 (42)



Alg. Bracha, Toueg'a (3)

```

1. procedure NOTIFYPROC() do
2.   notifiedi := True
3.   for all  $Q_k \in \mathbf{OUT}_i$  do
4.     send(  $Q_i, Q_k, notifyOut$  )
5.   end for
6.   if expectNoi = 0
7.     then
8.       CONFIRMPROC()
9.     end if
10.  for all  $Q_k \in \mathbf{OUT}_i$  do
11.    receive(  $Q_k, Q_i, doneIn$  )
12.  end for
13. end procedure

```

Detekcja zakleszczenia 1 (43)



Alg. Bracha, Toueg'a (4)

```

14. procedure CONFIRMPROC() do
15.   confirmedi := True
16.   for all  $Q_k \in \mathbf{IN}_i$  do
17.     send(  $Q_i, Q_k, confirmOut$  )
18.   end for
19.   for all  $Q_k \in \mathbf{IN}_i$  do
20.     receive(  $Q_k, Q_i, ackIn$  )
21.   end for
22. end procedure

```

Detekcja zakleszczenia 1 (44)



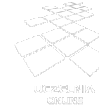
Alg. Bracha, Toueg'a (5)

```

23. when e_start(Qα, DeadlockDetection) do
24.   NOTIFYPROC()
25.   if confirmNoα < expectNoα
26.   then
27.     deadlockDetectedα := True
28.     decide(deadlockDetectedα)
29.   end if
30. end when

```

Detekcja zakleszczenia 1 (45)



Alg. Bracha, Toueg'a (6)

```

31. when e_receive(Qj, Qi, confirmIn: CONFIRM) do
32.   confirmNoi := confirmNoi + 1
33.   if ¬confirmedi ∧ (confirmNoi ≥ expectNoi)
34.   then
35.     CONFIRMPROC()
36.   end if
37.   send(Qi, Qj, ackOut)
38. end when

```

Detekcja zakleszczenia 1 (46)



Alg. Bracha, Toueg'a (7)

```

39. when e_receive(Qj, Qi, notifyIn: NOTIFY) do
40.   if ¬notifiedi
41.   then
42.     NOTIFYPROC()
43.   end if
44.   send(Qi, Qj, doneOut)
45. end when

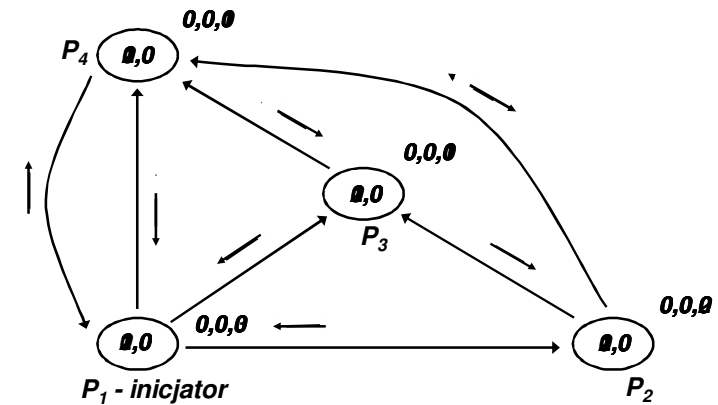
```

Detekcja zakleszczenia 1 (47)



Przykład działania algorytmu (1)

wiadomość typu NOTIFY
wiadomość typu DONE

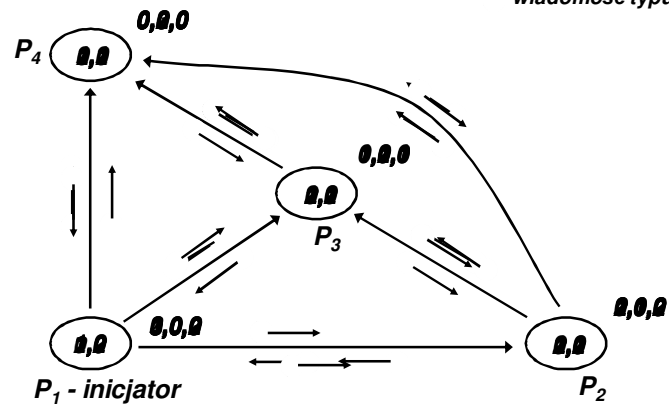


Detekcja zakleszczenia 1 (48)

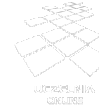


Przykład działania algorytmu (2)

wiadomość typu NOTIFY
 wiadomość typu DONE
 wiadomość typu CONFIRM
 wiadomość typu ACK



Detekcja zakleszczenia 1 (49)



Złożoność czasowa algorytmu detekcji zakleszczenia w środowisku synchronicznym dla modelu k spośród r ⁽¹⁾

Graf niezorientowany odpowiadający grafowi WFG scharakteryzowany jest przez: średnicę grafu d , najdłuższą ścieżkę w grafie l .

Złożoność całego algorytmu to:

- złożoność fazy potwierdzania
- złożoność powiadomienia inicjatora fazy potwierdzania,
- złożoność odpowiadającą przesłaniu wiadomości typu DONE od inicjatora fazy potwierdzania do inicjatora detekcji.

Detekcja zakleszczenia 1 (50)



Złożoność czasowa algorytmu detekcji zakleszczenia w środowisku synchronicznym dla modelu k spośród r ⁽²⁾

- przesłanie wiadomości typu CONFIRM może zabrać l jednostek czasu
- transmisja wiadomości typu ACK również zabiera w najgorszym wypadku l jednostek czasu
- przesłanie wiadomości typu NOTIFY (równoległe do monitorów wszystkich procesów tworzących zbiór OUT_i) wymaga co najwyżej d kroków.
- przesłanie wiadomości typu DONE po zakończeniu fazy potwierdzania wymaga co najwyżej d kroków.

W efekcie złożoność czasowa algorytmu wynosi $2d + 2l$.

Detekcja zakleszczenia 1 (51)