

## Detekcja zakleszczenia (2)

Wykład prowadzą:

Jerzy Brzeziński  
Jacek Kobusiński



Przetwarzanie rozproszone



## Plan wykładu

- Detekcja zakleszczenia w środowisku asynchronicznym dla modelu  $k$  spośród  $r$
- Dwufazowy algorytm detekcji zakleszczenia
- Detekcja zakleszczenia dla modelu predykatowego

Detekcja zakleszczenia (2) (2)

Przetwarzanie rozproszone



## Algorytm detekcji zakleszczenia w środowisku asynchronicznym (1)

Zakładamy, że w rozważanym środowisku rozproszonym czas transmisji w niezawodnych kanałach FIFO jest skończony ale nieznan. Wyznaczenie stanu globalnego takiego systemu musi zatem uwzględnić stany kanałów.

Detekcja zakleszczenia (2) (3)

Przetwarzanie rozproszone



## Algorytm detekcji zakleszczenia w środowisku asynchronicznym (2)

Wyróżnia się cztery kolory łuków:

- *Grey* – jeżeli proces  $P_i$  wysłał do  $P_j$  wiadomość typu REQUEST, a  $P_j$  jeszcze tej wiadomości nie odebrał, ani też  $P_i$  nie wysłał jeszcze wiadomości typu CANCEL.
- *Black* – jeżeli  $P_j$  odebrał już wiadomość typu REQUEST od  $P_i$ , lecz  $P_j$  jeszcze nie wysłał w odpowiedzi wiadomości typu GRANT do  $P_i$ , ani też  $P_i$  nie wysłał jeszcze wiadomości typu CANCEL do  $P_j$ .
- *White* – jeżeli  $P_j$  wysłał już wiadomość typu GRANT do  $P_i$ , lecz  $P_i$  jeszcze jej nie odebrał, ani też  $P_i$  nie wysłał jeszcze do  $P_j$  wiadomości typu CANCEL.
- *Translucent* – jeżeli  $P_i$  wysłał wiadomość typu CANCEL do  $P_j$ , lecz  $P_j$  jeszcze jej nie odebrał.

Detekcja zakleszczenia (2) (4)



## Algorytm detekcji zakleszczenia w środowisku asynchronicznym (3)

Załóżmy teraz, że każdy monitor zna odpowiadające mu zbiory  $IN_i$  oraz  $OUT_i$  oraz kolory wszystkich krawędzi incydentnych grafu WFG<sup>C</sup>. W tym wypadku, w przeciwieństwie do środowiska synchronicznego, nie jest prawdziwa relacja:

$$P_i \in OUT_j \Leftrightarrow P_j \in IN_i. \quad (6.1)$$



## Alg. Bacha, Toueg'a dla środowiska asynchronicznego (1)

```

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

```

```

notifyOut      : NOTIFY
doneOut, doneIn : DONE
confirmOut     : CONFIRM
ackOut, ackIn  : ACK
notifiedi     : BOOLEAN := False
confirmedi    : BOOLEAN := False
deadlockDetectedi : BOOLEAN := False

```



## Alg. Bacha, Toueg'a dla środowiska asynchronicznego (2)

```

AC : set of record of
  predId : PROCESS_ID
  succId : PROCESS_ID
end record

OUTiBlack : set of PROCESS_ID :=
  {Pj : ⟨Pi, PjC ∧ outArcColouri[j]=Black}
INiBlack : set of PROCESS_ID :=
  {Pj : ⟨Pi, PjC ∧ inArcColouri[j]=Black}

expectNoi      : INTEGER
outArcColouri : array [1..n] of enum {Black, White, Grey, Transl}
inArcColouri  : array [1..n] of enum {Black, White, Grey, Transl}
confirmNoi     : INTEGER := 0
outGreyWhiteNoi : INTEGER :=
  |{Ai,jC : outArcColouri[j]=Grey ∨ outArcColouri[j]=White}|

```



## Alg. Bacha, Toueg'a dla środowiska asynchronicznego (3)

```

1. procedure NOTIFYPROC () do
2.   notifiedi := True
3.   for all Qk ∈ OUTiBlack do
4.     send(Qi, Qk, notifyOut)
5.   end for
6.   if expectNoi - outGreyWhiteNoi ≤ 0
7.   then
8.     CONFIRMPROC ()
9.   end if
10.  for all Qk ∈ OUTiBlack do
11.    receive(Qk, Qi, doneIn)
12.  end for
13. end procedure

```



## Alg. Bacha, Toueg'a dla środowiska asynchronicznego (4)

```

14. procedure CONFIRMPROC () do
15.   confirmedi := True
16.   for all Qk ∈ INiBlack do
17.     send(Qi, Qk, confirmOut)
18.   end for
19.   for all Qk ∈ INiBlack do
20.     receive(Qk, Qi, ackIn)
21.   end for
22. end procedure

```



## Alg. Bacha, Toueg'a dla środowiska asynchronicznego (5)

```

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

```



## Alg. Bacha, Toueg'a dla środowiska asynchronicznego (6)

```

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

```



## Alg. Bacha, Toueg'a dla środowiska asynchronicznego (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

```



## Dwufazowy algorytm detekcji zakleszczenia

Algorytm detekcji zakleszczenia rozproszonego w środowisku z niezawodnymi kanałami FIFO dla modelu  $k$  spośród  $r$  składa się z dwóch faz wymiany wiadomości:

- fazy inicjacji (ang. *outward sweep*)
- fazy detekcji (ang. *inward sweep*)

Realizacja obu powyższych faz może nakładać się w czasie.



## Dwufazowy algorytm detekcji zakleszczenia dla modelu $k$ spośród $r$ (1)

```

type CONTROL extends FRAME is
  record of
    initIndex : INTEGER
    initClock : INTEGER
    weight    : REAL
  end record

type FLOOD extends CONTROL
type ECHO   extends CONTROL
type SHORT extends CONTROL

```



## Dwufazowy algorytm detekcji zakleszczenia dla modelu $k$ spośród $r$ (2)

```

floodOut      : FLOOD
echoOut       : ECHO
shortOut      : SHORT
inStatei    : array [1..n] of set of PROCESS_ID
outStatei   : array [1..n] of set of PROCESS_ID
expectNoi  : INTEGER
expectNoStatei : array [1..n] of INTEGER
latestInitClocki : INTEGER := 0
latestBlockClocki : INTEGER := 0
weighti    : REAL
clocki    : INTEGER
αi        : INTEGER
INi       : set of PROCESS_ID := ∅
OUTi      : set of PROCESS_ID := ∅
deadlockDetectedi : BOOLEAN := False

```



## Dwufazowy algorytm detekcji zakleszczenia dla modelu $k$ spośród $r$ (3)

```

1. procedure FILL (pcktOut:CONTROL, index, clock, weight) do
2.   pcktOut.initIndex:= index
3.   pcktOut.clock:= clock
4.   pcktOut.weight:= weight
5. end procedure

6. when e_start(Qα, DeadlockDetection) do
7.   weightα := 0
8.   latestInitClockα[α] := clockα
9.   outStateα[α] := OUTα
10.  inStateα[α] := ∅
11.  expectNoStateα[α] := expectNoα
12.  FILL(floodOut, α, clockα, 1 / |OUTα|)
13.  send(Qα, OUTα, floodOut)
14. end when

```



## Dwufazowy algorytm detekcji zakleszczenia dla modelu $k$ spośród $r$ (4)

```

15. when e_receive( $Q_j$ ,  $Q_i$ , floodIn : FLOOD) do
16.    $\alpha_i :=$  floodIn.initIndex
17.   if latestInitClock $_i[\alpha_i]$  < floodIn.initClock  $\wedge$   $P_j \in \mathcal{IN}_i$  then
18.     inState $_i[\alpha_i] := \{P_j\}$ 
19.     outState $_i[\alpha_i] := OUT_i$ 
20.     expectNoState $_i[\alpha_i] :=$  expectNo $_i$ 
21.     latestInitClock $_i[\alpha_i] :=$  floodIn.initClock
22.     if expectNoState $_i[\alpha_i]$  > 0 then
23.       FILL(floodOut,  $\alpha_i$ , floodIn.initClock,
                floodIn.weight/|OUT $_i$ |)
24.       send( $Q_i$ , OUT $_i$ , floodOut)
25.     end if
26.     if expectNoState $_i[\alpha_i]$  = 0 then
27.       FILL(echoOut,  $\alpha_i$ , floodIn.initClock, floodIn.weight)
28.       send( $Q_i$ ,  $Q_j$ , echoOut)
29.     end if
30.   end if

```

Detekcja zakleszczenia (2) (17)



## Dwufazowy algorytm detekcji zakleszczenia dla modelu $k$ spośród $r$ (5)

```

31.   if latestInitClock $_i[\alpha_i]$  <= floodIn.initClock  $\wedge$   $P_j \notin \mathcal{IN}_i$  then
32.     FILL(echoOut,  $\alpha_i$ , floodIn.initClock, floodIn.weight)
33.     send( $Q_i$ ,  $Q_j$ , echoOut)
34.   end if
35.   if latestInitClock $_i[\alpha_i]$  = floodIn.initClock  $\wedge$   $P_j \in \mathcal{IN}_i$  then
36.     inState $_i[j] :=$  inState $_i[j] \cup \{P_j\}$ 
37.     if expectNoState $_i[\alpha_i]$  > 0 then
38.       FILL(shortOut,  $\alpha_i$ , floodIn.initClock, floodIn.weight)
39.       send( $Q_i$ ,  $Q_j$ , shortOut)
40.     end if
41.     if expectNoState $_i[\alpha_i]$  = 0 then
42.       FILL(echoOut,  $\alpha_i$ , floodIn.initClock, floodIn.weight)
43.       send( $Q_i$ ,  $Q_j$ , echoOut)
44.     end if
45.   end if
46.   if latestInitClock $_i[\alpha_i]$  > floodIn.initClock then
47.     /* zignoruj przetwarzaną wiadomość FLOOD */
48.   end if
49. end when

```

Detekcja zakleszczenia (2) (18)



## Dwufazowy algorytm detekcji zakleszczenia dla modelu $k$ spośród $r$ (6)

```

50. when e_receive( $Q_j$ ,  $Q_i$ , echoIn: ECHO) do
51.    $\alpha_i :=$  echoIn.initIndex
52.   if latestInitClock $_i[\alpha_i]$  = echoIn.initClock then
53.     outState $_i[\alpha_i] :=$  outState $_i[\alpha_i] \setminus \{P_j\}$ 
54.     if expectNoState $_i[\alpha_i]$  = 0
55.       FILL(shortOut,  $\alpha_i$ , echoIn.initClock, echoIn.weight)
56.       send( $Q_i$ ,  $Q_i$ , shortOut)
57.     end if

```

Detekcja zakleszczenia (2) (19)



## Dwufazowy algorytm detekcji zakleszczenia dla modelu $k$ spośród $r$ (7)

```

58.   if expectNoState $_i[\alpha_i]$  > 0 then
59.     expectNoState $_i[\alpha_i] :=$  expectNoState $_i[\alpha_i]$  - 1
60.     if expectNoState $_i[\alpha_i]$  = 0 then
61.       if  $\alpha_i = i$  then
62.         deadlockDetected $_i :=$  False
63.         decide(deadlockDetected $_i$ )
64.       end if
65.       FILL(echoOut,  $\alpha_i$ , echoIn.initClock,
                echoIn.weight/|inState $_i[\alpha_i]$ |)
66.       send( $Q_i$ , inState $_i[\alpha_i]$ , echoOut)
67.     else
68.       FILL(shortOut,  $\alpha_i$ , echoIn.initClock,
                echoIn.weight)
69.       send( $Q_i$ , inState $_i[\alpha_i]$ , shortOut)
70.     end if
71.   end if
72. end if
73. end when

```

Detekcja zakleszczenia (2) (20)



## Dwufazowy algorytm detekcji zakleszczenia dla modelu $k$ spośród $r$ (8)

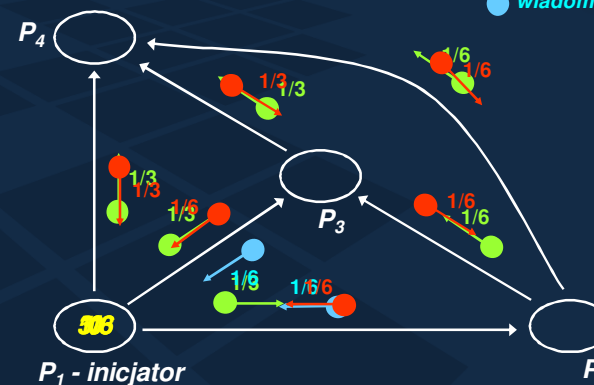
```

74. when e_receive( $Q_j$ ,  $Q_\alpha$ , shortIn: SHORT) do
75.   if shortIn.initClock < latestBlockClock $_\alpha$  then
76.     /* zignoruj nieaktualną wiadomość SHORT */
77.   end if
78.   if shortIn.initClock=latestBlockClock $_\alpha$   $\wedge$ 
       expectNoState $_\alpha$ [ $\alpha$ ]=0 then
79.     /* zignoruj wiadomość */
80.   end if
81.   if shortIn.initClock=latestBlockClock $_\alpha$   $\wedge$ 
       expectNoState $_\alpha$ [ $\alpha$ ] > 0 then
82.     weight $_\alpha$ := weight $_\alpha$  + shortIn.weight;
83.     if weight $_\alpha$  = 1 then
84.       deadlockDetected $_\alpha$ := True
85.       decide (deadlockDetected $_\alpha$ )
86.     end if
87.   end if
88. end when
  
```



## Przykład działania dwufazowego algorytmu detekcji zakleszczenia rozproszonego

- wiadomość typu FLOOD
- wiadomość typu ECHO
- wiadomość typu SHORT



## Analiza złożoności czasowej dwufazowego algorytmu detekcji zakleszczenia

Parametry graf niezorientowanego odpowiadającego WFG:

- $d$  – średnica grafu,
  - $l$  – najdłuższa ścieżka w grafie
1. Przesłanie wiadomości typu ECHO przez wszystkie krawędzie musi być poprzedzone wysłaniem wiadomości FLOOD w przeciwnym kierunku. Przejście wiadomości typu FLOOD przez wszystkie krawędzie wymaga  $d+1$  kroków.
  2. Najdłuższa ścieżka w grafie determinuje czas wymagany na propagację wiadomości typu ECHO od monitora procesu aktywnego do inicjatora.

**Złożoność czasowa algorytmu wynosi  $(d+1)+l$**



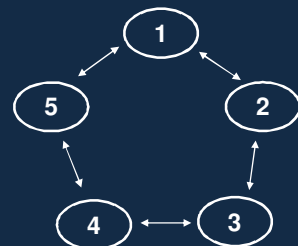
## Detekcja zakleszczenia dla modelu predykatowego (1)

W algorytmie detekcji zakleszczenia dla modelu predykatowego wykorzystana jest koncepcja **ciągu** (łańcucha, sekwencji) **cykli detekcyjnych** (ang. *wave sequence*), polegająca na wielokrotnym w ogólności komunikowaniu się inicjatora procesu detekcji z pozostałymi monitorami w celu wyznaczenia stanu globalnego, lub wartości predykatu globalnego.



## Detekcja zakleszczenia dla modelu predykatowego (2)

W celu uproszczenia prezentacji algorytmu detekcji zakleszczenia dla modelu predykatowego przyjmujemy, że monitory procesów aplikacyjnych tworzą strukturę pierścienia.



Niech ponadto:

$$\text{succ}(i) = (i) \bmod_n + 1$$

$$\text{pred}(i) = (i + n - 2) \bmod_n + 1$$



## Alg. detekcja zakleszczenia dla modelu predykatowego (1)

```

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

type TOKEN extends FRAME is
  record of
    PD       : set of PROCESS_ID
    firstWave : BOOLEAN
  end record

type ACK extends SIGNAL
  
```



## Alg. detekcja zakleszczenia dla modelu predykatowego (2)

```

msgIn      : MESSAGE
pcktOut    : PACKET
tokenOut   : TOKEN
ackOut     : ACK
firstWavei : BOOLEAN:= True
contPassivei : BOOLEAN
prevWavei  : INTEGER
PDi       : set of PROCESS_ID
AVi      : set of PROCESS_ID={Pj:availablei[j]=True}
notAcki   : INTEGER
deadlockDetectedi : BOOLEAN:= False
passivei  : BOOLEAN
  
```



## Alg. detekcja zakleszczenia dla modelu predykatowego (3)

```

1. procedure WAITUPDATEPROC()
2.   if tokenIn.firstWave
3.   then
4.     contPassivei := passivei
5.   end if
6.   wait until (¬contPassivei
7.             ∨ activatei(AVi ∪ (P \ PDi)) ∨ (notAcki=0)
8.             if (¬contPassivei) ∨ activatei(AVi ∪ (P \ PDi)) then
9.               PDi := PDi \ {Pi}
10.            end if
11.            contPassivei := passivei
11.   end procedure
  
```



## Alg. detekcja zakleszczenia dla modelu predykatowego (4)

```

12. when e_start( $Q_\alpha$ , DeadlockDetection) do
13.   prevWave $_\alpha$  := | $\mathcal{P}$ |
14.   tokenOut. $\mathcal{PD}$  :=  $\mathcal{P}$ 
15.   tokenOut.firstWave := firstWave $_\alpha$ 
16.   send( $Q_\alpha$ ,  $Q_{succ(\alpha)}$ , tokenOut)
17. end when

```



## Alg. detekcja zakleszczenia dla modelu predykatowego (5)

```

18. when e_receive( $Q_j$ ,  $Q_i$ , tokenIn:TOKEN) do
19.    $\mathcal{PD}_i$  := tokenIn. $\mathcal{PD}$ 
20.   if  $P_i \in \mathcal{PD}_i$  then
21.     WAITUPDATEPROC()
22.   end if
23.   tokenOut. $\mathcal{PD}$  :=  $\mathcal{PD}_i$ 
24.   tokenOut.firstWave := tokenIn.firstWave
25.   send( $Q_i$ ,  $Q_{succ(i)}$ , tokenOut)
26. end when

```



## Alg. detekcja zakleszczenia dla modelu predykatowego (6)

```

27. when e_receive( $Q_j$ ,  $Q_\alpha$ , tokenIn:TOKEN) do
28.    $\mathcal{PD}_\alpha$  := tokenIn. $\mathcal{PD}$ 
29.   if  $P_\alpha \in \mathcal{PD}_\alpha$  then
30.     WAITUPDATEPROC()
31.   end if
32.   if (tokenIn.firstWave  $\vee$  prevWave $_\alpha \neq |\mathcal{PD}_\alpha|$ )  $\wedge$  ( $|\mathcal{PD}_\alpha| \neq 0$ ) then
33.     prevWave $_\alpha$  :=  $|\mathcal{PD}_\alpha|$ 
34.     tokenOut. $\mathcal{PD}$  :=  $\mathcal{PD}_\alpha$ 
35.     tokenOut.firstWave := False
36.     send( $Q_\alpha$ ,  $Q_{succ(\alpha)}$ , tokenOut)
37.   else
38.     deadlockDetected $_\alpha$  := ( $\mathcal{PD}_\alpha \neq \emptyset$ )
39.     decide(deadlockDetected $_\alpha$ ,  $\mathcal{PD}_\alpha$ )
40.   end if
41. end when

```



## Alg. detekcja zakleszczenia dla modelu predykatowego (7)

```

42. when e_send( $P_i$ ,  $P_j$ , msgOut:MESSAGE) do
43.   notAck $_i$  := notAck $_i$  + 1
44.   pcktOut.data := msgOut
45.   send( $Q_i$ ,  $Q_j$ , pcktOut)
46. end when
47. when e_receive( $Q_j$ ,  $Q_i$ , pcktIn:PACKET) do
48.   msgIn := pcktIn.data
49.   deliver( $P_i$ ,  $P_j$ , msgIn)
50.   send( $Q_i$ ,  $Q_j$ , ackOut)
51. end when
52. when e_receive( $Q_j$ ,  $Q_i$ , ackIn:ACK) do
53.   notAck $_i$  := notAck $_i$  - 1
54. end when
55. when e_activate( $P_i$ ) do
56.   contPassive $_i$  := False
57. end when

```