Detekcja zakleszczenia (2)

Wykład prowadzą:

Jerzy Brzeziński
Jacek Kobusiński

Plan wykładu

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

Algorytm detekcji zakleszczenia w środowisku asynchronicznym (1)

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

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.

Algorytm detekcji zakleszczenia w środowisku asynchronicznym (3)

Załóżmy teraz, że każdy monitor zna odpowiadające mu zbiory $\mathcal{IN}_i$ oraz $\mathcal{OUT}_i$ oraz kolory wszystkich krawędzi incydentnych grafu WFG^C. W tym wypadku, w przeciwieństwie do środowiska synchronicznego, nie jest prawdziwa relacja:

$$P_i \in \mathcal{OUT}_j \Leftrightarrow P_j \in \mathcal{IN}_i. \qquad (6.1)$$

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
notified_i          : BOOLEAN := False
confirmed_i         : BOOLEAN := False
deadlockDetected_i  : BOOLEAN := False
```


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

$\mathcal{A}^C$: **set of record of**
predId : PROCESS_ID
succId : PROCESS_ID
end record

$\mathcal{OUT}_i^{Black}$: **set of** PROCESS_ID :=
$\{P_j : \langle P_i, P_j\rangle \in \mathcal{A}^C \wedge outArcColour_i[j]=Black\}$

$\mathcal{IN}_i^{Black}$: **set of** PROCESS_ID :=
$\{P_j : \langle P_i, P_j\rangle \in \mathcal{A}^C \wedge inArcColour_i[j]=Black\}$

$expectNo_i$: INTEGER
$outArcColour_i$: **array** [1..n] **of enum** {*Black,White,Grey,Transl*}
$inArcColour_i$: **array** [1..n] **of enum** {*Black,White,Grey,Transl*}
$confirmNo_i$: INTEGER := 0
$outGreyWhiteNo_i$: INTEGER :=
$|\{A_{i,j}{}^C : outArcColour_i[j]=Grey \vee outArcColour_i[j]=White\}|$

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

```
1.  procedure NOTIFYPROC() do
2.   notified_i := True
3.   for all Q_k ∈ OUT_i^Black do
4.      send(Q_i, Q_k, notifyOut)
5.   end for
6.   if expectNo_i − outGreyWhiteNo_i ⩽ 0
7.   then
8.      CONFIRMPROC()
9.   end if
10.  for all Q_k ∈ OUT_i^Black do
11.     receive(Q_k, Q_i, doneIn)
12.  end for
13. end procedure
```

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

```
14. procedure CONFIRMPROC() do
15.   confirmed_i := True
16.   for all Q_k ∈ IN_i^Black do
17.         send(Q_i, Q_k, confirmOut)
18.   end for
19.   for all Q_k ∈ IN_i^Black do
20.         receive(Q_k, Q_i, ackIn)
21.   end for
22. end procedure
```


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

```
23. when e_start(Q_α, DeadlockDetection) do
24.   NOTIFYPROC()
25.  if confirmNo_α < expectNo_α − outGreyWhiteNo_α
26.  then
27.     deadlockDetected_α := True
28.     decide(deadlockDetected_α)
29.  end if
30. end when
```


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

```
31. when e_receive(Q_j, Q_i, confirmIn: CONFIRM) do
32.   confirmNo_i := confirmNo_i + 1
33.   if ¬confirmed_i ∧
          (confirmNo_i ⩾ expectNo_i − outGreyWhiteNo_i)
34.   then
35.       CONFIRMPROC()
36.   end if
37.   send(Q_i, Q_j, ackOut)
38. end when
```


Alg. Bacha, Toueg'a dla środowiska asynchronicznego (7)

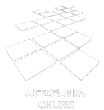
```
39. when e_receive(Q_j, Q_i, notifyIn: NOTIFY) do
40.    if  ¬notified_i
41.    then
42.       NOTIFYPROC()
43.    end if
44.    send(Q_i, Q_j, doneOut)
45. end when
```

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
inState_i         : array [1..n] of set of PROCESS_ID
outState_i        : array [1..n] of set of PROCESS_ID
expectNo_i        : INTEGER
expectNoState_i   : array [1..n] of  INTEGER
latestInitClock_i : INTEGER := 0
latestBlockClock_i: INTEGER := 0
weight_i          : REAL
clock_i           : INTEGER
α_i               : INTEGER
IN_i              : set of PROCESS_ID := ∅
OUT_i             : set of PROCESS_ID := ∅
deadlockDetected_i: BOOLEAN := False
```

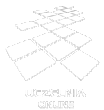

Dwufazowy algorytm detekcji zakleszczenia dla *modelu k spośród r* (3)

```
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(Q_α, DeadlockDetection) do
7.      weight_α := 0
8.      latestInitClock_α[α] := clock_α
9.      outState_α[α] := OUT_α
10.     inState_α[α] := ∅
11.     expectNoState_α[α] := expectNo_α
12.     FILL(floodOut, α, clock_α, 1 / |OUT_α|)
13.     send(Q_α, OUT_α, 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.   α_i := floodIn.initIndex
17.   if latestInitClock_i[α_i] < floodIn.initClock ∧ P_j∈IN_i then
18.    inState_i[α_i] := {P_j}
19.    outState_i[α_i] := OUT_i
20.    expectNoState_i[α_i] := expectNo_i
21.    latestInitClock_i[α_i] := floodIn.initClock
22.    if expectNoState_i[α_i] > 0 then
23.     FILL(floodOut, α_i, floodIn.initClock,
                                  floodIn.weight/|OUT_i|)
24.     send(Q_i, OUT_i ,floodOut)
25.    end if
26.    if expectNoState_i[α_i] = 0 then
27.     FILL(echoOut, α_i, floodIn.initClock, floodIn.weight)
28.     send(Q_i, Q_j, echoOut)
29.    end if
30.   end if
```

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

```
31.   if latestInitClock_i[α_i]≤floodIn.initClock ∧ P_j∉IN_i then
32.     FILL(echoOut,α_i,floodIn.initClock,floodIn.weight)
33.     send(Q_i, Q_j, echoOut)
34.   end if
35.   if latestInitClock_i[α_i]=floodIn.initClock ∧ P_j∈IN_i then
36.     inState_i[j]:= inState_i[j] ∪ {P_j}
37.     if expectNoState_i[α_i] > 0 then
38.      FILL(shortOut,α_i,floodIn.initClock,floodIn.weight)
39.      send(Q_i, Q_j, shortOut)
40.     end if
41.     if expectNoState_i[α_i] = 0 then
42.      FILL(echoOut,α_i,floodIn.initClock,floodIn.weight)
43.      send( Q_i, Q_j, echoOut )
44.    end if
45.   end if
46.   if latestInitClock_i[α_i] > floodIn.initClock then
47.           /* zignoruj przetwarzaną wiadomość FLOOD */
48.   end if
49.  end when
```


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

```
50.  when e_receive(Q_j, Q_i, echoIn: ECHO) do
51.    α_i:= echoIn.initIndex
52.    if latestInitClock_i[α_i] = echoIn.initClock then
53.      outState_i[α_i]:= outState_i[α_i]\{P_j}
54.   if expectNoState_i[α_i] = 0
55.     FILL(shortOut, α_i, echoIn.initClock, echoIn.weight)
56.     send(Q_i, Q_α, shortOut)
57.   end if
```

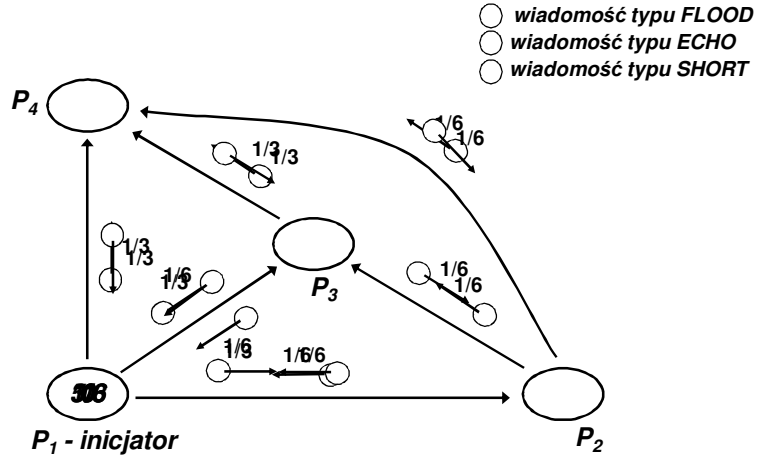

Dwufazowy algorytm detekcji zakleszczenia dla *modelu k spośród r* (7)

```
58.      if expectNoState_i[α_i] > 0 then
59.        expectNoState_i[α_i]:= expectNoState_i[α_i] - 1
60.        if expectNoState_i[α_i]=0  then
61.          if α_i=i then
62.            deadlockDetected_i:= False
63.            decide(deadlockDetected_i)
64.          end if
65.          FILL(echoOut, α_i, echoIn.initClock,
                              echoIn.weight/|inState_i[α_i]|)
66.          send(Q_i ,inState_i[α_i], echoOut)
67.        else
68.          FILL(shortOut, α_i, echoIn.initClock,
                              echoIn.weight)
69.            send(Q_i, inState_i[α_i], shortOut)
70.          end if
71.      end if
72.   end if
73.  end when
```

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

```
74.  when e_receive(Q_j, Q_α, shortIn: SHORT) do
75.   if shortIn.initClock < latestBlockClock_α  then
76.    /* zignoruj nieaktualną wiadomość SHORT */
77.   end if
78.   if shortIn.initClock=latestBlockClock_α ∧
                                  expectNoState_α[α]=0 then
79.    /* zignoruj wiadomość */
80.   end if
81.   if shortIn.initClock=latestBlockClock_α ∧
                                  expectNoState_α[α] > 0 then
82.    weight_α:= weight_α + shortIn.weight;
83.    if weight_α = 1 then
84.       deadlockDetected_α:= True
85.       decide(deadlockDetected_α)
86.    end if
87.   end if
88.  end when
```


Przykład działania dwufazowego algorytmu detekcji zakleszczenia rozproszonego



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.

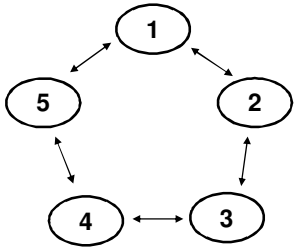
Złożoność czasowa algorytmu wynosi $(d+1)+l$

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 przyjmiemy, że monitory procesów aplikacyjnych tworzą strukturę pierścienia.



Niech ponadto:

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

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

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
firstWave_i        : BOOLEAN:= True
contPassive_i      : BOOLEAN
prevWave_i         : INTEGER
PD_i               : set of PROCESS_ID
AV_i               : set of PROCESS_ID={P_j:available_i[j]=True}
notAck_i           : INTEGER
deadlockDetected_i : BOOLEAN:= False
passive_i          : BOOLEAN
```


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

```
1.  procedure WAITUPDATEPROC()
2.     if tokenIn.firstWave
3.     then
4.        contPassive_i :=  passive_i
5.     end if
6.     wait until (¬contPassive_i)
                  ∨ activate_i(AV_i∪(P\PD_i))∨ (notAck_i=0)
7.       if (¬contPassive_i) ∨ activate_i(AV_i ∪ (P\PD_i)) then
8.          PD_i := PD_i\{P_i}
9.       end if
10.      contPassive_i := passive_i
11.  end procedure
```

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

```
12.  when e_start(Q_α,DeadlockDetection) do
13.    prevWave_α := |P|
14.    tokenOut.PD := P
15.    tokenOut.firstWave := firstWave_α
16.    send(Q_α, Q_succ(α), tokenOut)
17.  end when
```


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

```
18.  when e_receive(Q_j,Q_i,tokenIn:TOKEN) do
19.     PD_i := tokenIn.PD
20.     if P_i ∈ PD_i then
21.       WAITUPDATEPROC()
22.     end if
23.     tokenOut.PD := 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_α,tokenIn: TOKEN) do
28.    PD_α := tokenIn.PD
29.    if P_α ∈ PD_α then
30.      WAITUPDATEPROC()
31.    end if
32.    if (tokenIn.firstWave∨prevWave_α≠|PD_α|)∧(|PD_α|≠0) then
33.      prevWave_α := |PD_α|
34.      tokenOut.PD := PD_α
35.      tokenOut.firstWave := False
36.      send(Q_α, Q_succ(α), tokenOut)
37.    else
38.      deadlockDetected_α := (PD_α≠∅)
39.      decide(deadlockDetected_α,PD_α)
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
```