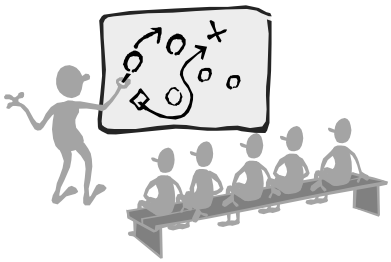


Podstawy przetwarzania rozproszonego

Detekcja zakleszczenia

Spis algorytmów

(c) Zakład Systemów Informatycznych

Wprowadzenie (1)

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.

Jeżeli żądany zasób jest niezbędny dla kontynuacji działania, proces żądający musi czekać w stanie pasywnym aż do momentu nadejścia komunikatu potwierdzającego przydział lub zawierającego żądany zasób (obiekt).

Dla przykładu procesy zarządzające elementami rozległej sieci komputerowej kooperują w celu efektywnego, niezawodnego i bezpiecznego rozdziału miedzy innymi takich zasobów jak:

- bufory łączy komunikacyjnych,
- szerokość udostępnionego pasma transmisji,
- tablice wyboru trasy,
- tablice konwersji nazw

Wprowadzenie (2)

- 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).
- Wystąpienie zakleszczenia wstrzymuje trwale procesy zakleszczone i tym samym blokuje będące w ich posiadaniu **zasoby**. Jeśli więc stan taki nie zostanie wykryty i nie będą podjęte odpowiednie działania, to procesy zakleszczone i blokowane przez nie zasoby mogą spowodować wstrzymanie innych procesów, czy wreszcie doprowadzić do blokady całego systemu rozproszonego.
- Rozwiązanie problemu detekcji zakleszczenia rozproszonego ma duże znaczenie w przetwarzaniu rozproszonym.
- Wobec asynchronizmu komunikacji i przetwarzania oraz braku wspólnego zegara, rozwiązanie tego problemu jest trudne i wymaga rozpatrzenia wielu subtelnych kwestii.

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
- ❑ zmieniające stan procesu na pasywny i definiujące jednocześnie ***zbiór warunkujący*** (sumę zbiorów oczekiwanych nadawców wszystkich dopuszczalnych zdarzeń odbioru) oraz ***warunek uaktywnienia***.

Warunek uaktywnienia jest wyrażony przez predykat:

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

Po spełnieniu warunku uaktywnienia, proces zmienia swój stan na aktywny i w sposób atomowy (niepodzielny) pobiera z kanałów wejściowych wszystkie te wiadomości, których nadejście doprowadziło do spełnienia warunku uaktywnienia.

Definicja problemu

Wprowadzimy następujące oznaczenia:

- ❖ $passive_i$ – zmienna logiczna (predykat) przyjmująca wartość *True* wtedy i tylko wtedy, gdy proces P_i jest pasywny
- ❖ $available_i$ – tablica [1..*n*] zmiennych logicznych procesu P_i skojarzona z wiadomościami dostępnymi
- ❖ $available_i[j]$ – *j*-ty element tablicy $available_i$ przyjmujący wartość *True* wtedy i tylko wtedy, gdy dla P_i jest dostępna wiadomość wysłana przez P_j
- ❖ $in\text{-}transit_i$ – tablica [1..*n*] zmiennych logicznych procesu P_i skojarzona z wiadomościami transmitowanymi
- ❖ $in\text{-}transit_i[j]$ – *j*-ty element tablicy $in\text{-}transit_i$ przyjmujący wartość *True* wtedy i tylko wtedy, gdy wiadomość wysłana przez P_j do P_i należy do $L^T_{j,i}$, a więc jest transmitowana i nie jest jeszcze dostępna

$\mathcal{AV}_i = \{P_j : available_i[j] = True\}$

$\mathcal{IT}_i = \{P_j : intransit_i[j] = True\}$

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

Zakleszczenie w *modelu jednostkowym*

W *modelu jednostkowym* warunkiem uaktywnienia pasywnego procesu P_i jest przybycie wiadomości od ściśle określonego, jednego nadawcy. Tak więc dla każdego zbioru warunkującego $\mathcal{D}_i$, $|\mathcal{D}_i| = 1$.

Wówczas:

$$
\begin{aligned}
&deadlock(\mathcal{B}) \equiv \\
&\quad (\ \mathcal{B} \subseteq \mathcal{P}\ ) \wedge (\ \mathcal{B} \neq \emptyset) \wedge \\
&\quad (\forall P_i :: P_i \in \mathcal{B} :: (\ passive_i \ \wedge \\
&\qquad |\mathcal{D}_i| = 1 \ \wedge \\
&\qquad (\exists\ P_j :: P_j \in \mathcal{D}_i \cap \mathcal{B} :: (\neg\ in\text{-}transit_i[j] \wedge \neg\ available_i[j]\ ))))
\end{aligned}
$$

Zakleszczenie w *modelu AND*

W *modelu AND* proces pasywny P_i staje się aktywny, jeżeli dotarły do niego wiadomości od każdego z procesów tworzących jego zbiór warunkujący $\mathcal{D}_i$.

Wówczas:

$$
\begin{aligned}
&deadlock(\mathcal{B}) \equiv \\
&\quad (\ \mathcal{B} \subseteq \mathcal{P}\ ) \wedge (\ \mathcal{B} \neq 0) \wedge \\
&\quad (\forall P_i :: P_i \in \mathcal{B} :: (\ passive_i \wedge \\
&\qquad (\exists\ P_j :: P_j \in \mathcal{D}_i \cap \mathcal{B} :: (\neg\ in\text{-}transit_i[j] \wedge \neg\ available_i[j]\ ))))
\end{aligned}
$$

Zakleszczenie w *modelu OR*

W *modelu OR* do uaktywnienia procesu P_i wystarczy jedna wiadomość od któregokolwiek z procesów jego zbioru warunkującego $\mathcal{D}_i$.

Dlatego:

$$
\begin{aligned}
&deadlock(\mathcal{B}) \equiv \\
&\quad (\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq 0) \wedge \\
&\quad (\forall P_i :: P_i \in \mathcal{B} :: (passive_i \wedge \\
&\qquad \mathcal{D}_i \subseteq \mathcal{B} \ \wedge \\
&\qquad (\forall P_j :: P_j \in \mathcal{D}_i :: (\neg\, in\text{-}transit_i[j] \wedge \neg\, available_i[j]))))
\end{aligned}
$$

Zakleszczenie w podstawowym modelu *k spośród r*

W *podstawowym modelu k spośród r,* z pasywnym procesem P_i skojarzony jest zbiór warunkujący $\mathcal{D}_i$, liczba naturalna k_i, $1 \leqslant k_i \leqslant |\mathcal{D}_i|$, oraz liczba naturalna $r_i = |\mathcal{D}_i|$. W modelu tym proces P_i staje się aktywny wówczas, gdy uzyska wiadomości od co najmniej k_i różnych procesów ze zbioru warunkującego $\mathcal{D}_i$.

Wówczas:

$$
\begin{aligned}
&deadlock(\mathcal{B}) \equiv \\
&\quad (\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq 0) \wedge \\
&\quad (\forall P_i :: P_i \in \mathcal{B} :: (passive_i \wedge \\
&\qquad (\exists \mathcal{B}_i :: \mathcal{B}_i \subseteq \mathcal{D}_i \cap \mathcal{B} :: \\
&\qquad\quad (|\mathcal{D}_i \setminus \mathcal{B}_i| < k_i \ \wedge \\
&\qquad\quad (\forall P_j :: P_j \in \mathcal{B}_i :: (\neg\, in\text{-}transit_i[j] \wedge \neg\, available_i[j])))))
\end{aligned}
$$

Oznacza to, że dla każdego procesu P_i można znaleźć zbiór $\mathcal{B}_i$ procesów, od których nie jest możliwe otrzymanie wiadomości ($\mathcal{B}_i \subseteq \mathcal{B}$) i jednocześnie $\forall P_j :: P_j \in \mathcal{B}_i :: (\neg\, in\text{-}transit_i[j] \wedge \neg\, available_i[j])$. P_i potencjalnie otrzyma co najwyżej $|\mathcal{D}_i \setminus \mathcal{B}_i|$, co nie wystarcza do uaktywnienia, gdyż $|\mathcal{D}_i \setminus \mathcal{B}_i| < k_i$.

Zakleszczenie w *modelu OR – AND*

W *modelu OR – AND* zbiór warunkujący $\mathcal{D}_i$ pasywnego procesu P_i jest zdefiniowany jako $\mathcal{D}_i^1 \cup \mathcal{D}_i^1 \cup ... \cup \mathcal{D}_i^{q_i}$, gdzie dla każdego naturalnego u, $1 \leqslant u \leqslant q_i$, $\mathcal{D}_i^u \subseteq \mathcal{P}$. Proces staje się aktywny po otrzymaniu wiadomości: od każdego z procesów tworzących zbiór $\mathcal{D}_i^1$ lub od każdego z procesów tworzących zbiór $\mathcal{D}_i^2$, lub ... lub od każdego z procesów tworzących zbiór $\mathcal{D}_i^{q_i}$

$$
\begin{aligned}
&deadlock(\mathcal{B}) \equiv \\
&\quad (\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq 0) \wedge \\
&\quad (\forall P_i :: P_i \in \mathcal{B} :: (passive_i \wedge \\
&\qquad (\forall u :: 1 \leqslant u \leqslant q_i :: \\
&\qquad\quad (\exists P_j :: P_j \in \mathcal{D}_i^u \cap \mathcal{B} :: (\neg in\text{-}transit_i[j] \wedge \neg available_i[j]))))))
\end{aligned}
$$

Zakleszczenie w *modelu dysjunkcyjnym k spośród r*

W *modelu dysjunkcyjnym k spośród r* z każdym pasywnym procesem P_i skojarzony jest zbiór warunkujący $\mathcal{D}_i = \mathcal{D}_i^1 \cup \mathcal{D}_i^2 \cup ... \cup \mathcal{D}_i^{q_i}$, liczby naturalne $k_i^1, k_i^2, ..., k_i^{q_i}$ i liczby naturalne $r_i^1, r_i^2, ..., r_i^{q_i}$, gdzie $\mathcal{D}_i \subseteq \mathcal{P}$ oraz dla każdego naturalnego u, $1 \leqslant u \leqslant q_i$, $1 \leqslant k_i^u \leqslant r_i^u = |\mathcal{D}_i^u|$. Proces staje się aktywny po otrzymaniu: wiadomości od co najmniej k_i^1 różnych procesów ze zbioru $\mathcal{D}_i^1$, lub wiadomości od co najmniej k_i^2 różnych procesów ze zbioru $\mathcal{D}_i^2$, lub ... lub wiadomości od co najmniej $k_i^{q_i}$ różnych procesów ze zbioru $\mathcal{D}_i^{q_i}$. Wówczas:

$$
\begin{aligned}
&deadlock(\mathcal{B}) \equiv \\
&\quad (\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq 0) \wedge \\
&\quad (\forall P_i :: P_i \in \mathcal{B} :: (passive_i \wedge \\
&\qquad (\forall u :: 1 \leqslant u \leqslant q_i :: \\
&\qquad (\exists \mathcal{B}_i^u :: \mathcal{B}_i^u \subseteq \mathcal{D}_i^u \cap \mathcal{B} :: \\
&\qquad\quad (|\mathcal{D}_i^u \setminus \mathcal{B}_i^u| < k_i^u \ \wedge \\
&\qquad\qquad (\forall P_j :: P_j \in \mathcal{B}_i^u :: (\neg in\text{-}transit_i[j] \wedge \neg available_i[j])))))))
\end{aligned}
$$

Zakleszczenie w *modelu predykatowym*

W *modelu predykatowym* dla każdego pasywnego procesu P_i ze zbiorem warunkującym $\mathcal{D}_i$ i dla każdego zbioru $\mathcal{X} \subseteq \mathcal{D}_i$ zdefiniowany jest predykat $activate_i(\mathcal{X})$. Predykat ten zachodzi, wtedy i tylko wtedy, gdy w związku z dostępnością wiadomości od wszystkich procesów tworzących zbiór $\mathcal{X}$, gotowe stanie się którekolwiek z dopuszczalnych zdarzeń odbioru. Wówczas:

$$deadlock(\mathcal{B}) \equiv$$
$$(\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq 0) \wedge$$
$$(\forall P_i :: P_i \in \mathcal{B} :: (passive_i \wedge \neg activate_i(\mathcal{AV}_i \cup \mathcal{IT}_i \cup (\mathcal{P} \setminus \mathcal{B}))))$$

Definicja ta oznacza, że żaden proces $P_i \in \mathcal{B}$ nawet jeżeli uwzględnimy wszystkie wiadomości znajdujące się aktualnie w kanałach od oczekiwanych nadawców ($\mathcal{AV}_i \cup \mathcal{IT}_i$) oaz potencjalne wiadomości od wszystkich procesów niezakleszczonych. Dotarcie wszystkich tych wiadomości nie jest dla spełnienia warunku uaktywnienia któregokolwiek z procesów $P_i \in \mathcal{B}$.

Definicja zakleszczenia dla modelu predykatowego jest ogólna i może być dostosowana do innych modeli.

UWAGA

Przy założeniu, że żaden proces nie osiąga *stanu zakończenia*, zakleszczenie dotyczy zawsze co najmniej dwóch procesów.

Przypuśćmy, że:

- $\mathcal{B} = \{P_i\}$,
- zachodzi $deadlock(\mathcal{B})$,
- żaden inny proces nie jest zakleszczony.

Ponieważ $P_i \notin \mathcal{AV}_i \cup \mathcal{IT}_i$ otrzymujemy

$$\mathcal{AV}_i \cup \mathcal{IT}_i \cup \mathcal{P} \setminus \mathcal{B} = \mathcal{P} \setminus \mathcal{B} .$$

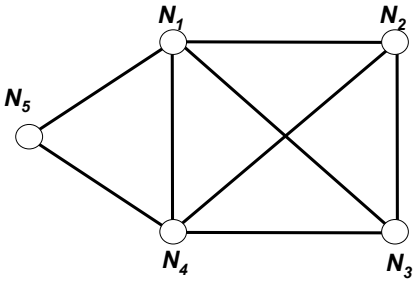
Z drugiej strony $\mathcal{D}_i \subseteq \mathcal{P} \setminus \mathcal{B} = \mathcal{P} \setminus \{P_i\}$ i stąd:

$$\neg activate_i(\mathcal{P} \setminus \mathcal{B}) \Rightarrow \neg activate_i(\mathcal{D}_i).$$

Otrzymujemy zatem sprzeczność, gdyż z założenia $\mathcal{D}_i \neq \emptyset$, a z definicji predykatu $activate_i(\mathcal{X})$, $activate_i(\mathcal{D}_i) = True$.

Przykłady zakleszczeń dla różnych modeli żądań

***Wait-For Graph* (*WFG*):**

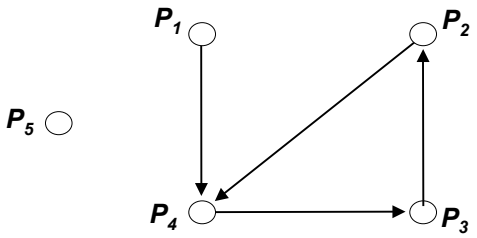


- N_1, N_2, N_3, N_4, N_5 – węzły środowiska przetwarzania
- P_1, P_2, P_3, P_4, P_5 – procesy wykonywane w odpowiednich węzłach
- $P_i \rightarrow P_j$ – łuk $\langle P_i, P_j \rangle$ reprezentujący fakt, że proces P_i, oczekuje na wiadomość od procesu P_j

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

(c) Zakład Systemów Informatycznych Slajd 18

Przykład – *model jednostkowy*



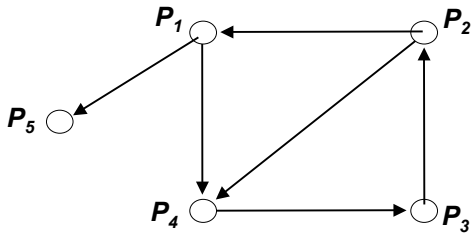
Zbiory warunkujące:

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

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

(c) Zakład Systemów Informatycznych Slajd 19

Przykład – *model AND*

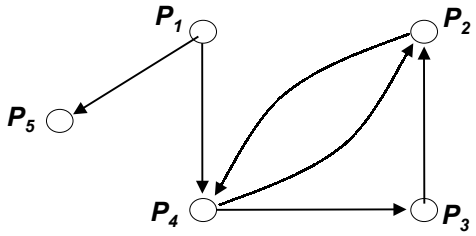


Zbiory warunkujące:

- $\mathcal{D}_1 = \{P_4, P_5\}$,
- $\mathcal{D}_2 = \{P_1, P_4\}$,
- $\mathcal{D}_3 = \{P_2\}$,
- $\mathcal{D}_4 = \{P_3\}$.

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

Przykład – *model OR*

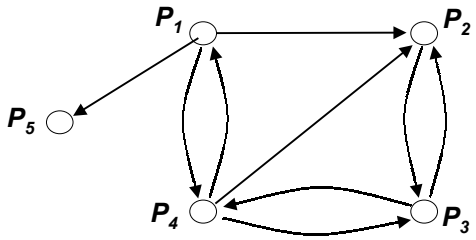


Zbiory warunkujące:

- $\mathcal{D}_1 = \{P_4, P_5\}$,
- $\mathcal{D}_2 = \{P_4\}$,
- $\mathcal{D}_3 = \{P_2\}$,
- $\mathcal{D}_4 = \{P_2, P_3\}$.

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

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:

- $\mathcal{D}_1 = \{P_2, P_4, P_5\}$,
- $\mathcal{D}_2 = \{P_3\}$,
- $\mathcal{D}_3 = \{P_2, P_4\}$,
- $\mathcal{D}_4 = \{P_1, P_2, P_3\}$.

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

(c) Zakład Systemów Informatycznych Slajd 22

Problem detekcji wystąpienia zakleszczenia

Problem *detekcji wystąpienia zakleszczenia*, sprowadza się do znalezienia odpowiedzi na pytanie: Czy istnieje w pewnej chwili zbiór $\mathcal{B}$, dla którego predykat *deadlock*($\mathcal{B}$) jest prawdziwy?

Odpowiedź na to pytanie określa wartość predykatu:

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

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

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

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

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

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

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

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

(c) Zakład Systemów Informatycznych Slajd 23

Algorytmy detekcji zakleszczenia rozproszonego

W wielu zaproponowanych dotychczas algorytmach detekcji zakleszczenia przyjmuje się, że procesy aplikacyjne P_i wysyłają jawnie żądanie przydziału zasobów w formie wiadomości typu `REQUEST` do procesów tworzących zbiór warunkujący $\mathcal{D}_i$ i oczekują w stanie pasywnym na wiadomości typu `GRANT` potwierdzające przyznanie żądanych zasobów.

Gdy do P_i dotrze odpowiedni zbiór wiadomości typu `GRANT`, proces aplikacyjny staje się aktywny i wysyła wiadomość typu `CANCEL` do pozostałych procesów ze zbioru warunkującego (od których nie odebrał `GRANT`), w celu wycofania (unieważnienia) wcześniej wysłanego żądania. Po wykorzystaniu zasobu, proces aplikacyjny może sygnalizować jego zwolnienie przez wysłanie wiadomości typu `RELEASE` do procesu zarządzającego przydziałem zasobów.

Przy takim modelu aplikacyjnego przetwarzania rozproszonego stan globalny może być reprezentowany przez *graf oczekiwanych potwierdzeń* WFG, w którym wierzchołki odpowiadają procesom P_i, łuki $\langle P_i,P_j \rangle$ reprezentują fakt, że P_i wysłał już żądanie `REQUEST` do P_j, lecz nie otrzymał potwierdzenia `GRANT`, ani nie wysłał wiadomości `CANCEL`.

Detekcja zakleszczenia dla *modelu AND* (1)

- Zakładamy, że kanały są kanałami *FIFO*.
- Monitor pasywnego procesu inicjuje spontanicznie *przetwarzanie detekcyjne* (ang. probe computation).
- W procesie detekcji monitory przesyłają wiadomości kontrolne typu `PROBE`. Wysłanie wiadomości kontrolnej może być realizowane jednocześnie przez wiele procesów i dlatego wiadomości te zawierają pole *initIndex*, określające indeks inicjatora.
- Monitor Q_i akceptuje wiadomość typu `PROBE` odebraną od Q_j i przesyła ją dalej do Q_k w wypadku, gdy spełnione są jednocześnie następujące warunki:
  1) P_i jest pasywny,
  2) P_i wysłał żądanie `REQUEST` i oczekuje na potwierdzenie `GRANT` od P_k ($P_k \in \mathcal{D}_i$),
  3) P_i nie wysłał potwierdzenia `GRANT` na ostatnie żądanie `REQUEST` od P_j,
  4) Otrzymana wiadomość `PROBE` jest pierwszą wiadomością tego typu od danego inicjatora Q_α, od momentu, gdy P_i zmienił swój stan na pasywny.
- Jeżeli Q_α zaakceptuje wiadomość `PROBE` przez siebie zainicjowaną, to proces P_α jest zakleszczony.

Detekcja zakleszczenia w *modelu AND*

- Wstęp
- Typy komunikatów
- Deklaracje
- Procedury
- Akcje

(Chandy-Misra-Hass)

- Wstęp
- Typy komunikatów
- Deklaracje
- Procedury
- Akcje

TYPY KOMUNIKATÓW

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

Detekcja zakleszczenia w *modelu AND* (deklaracje)

DEKLARACJE

probeOut	: PROBE
$granted_i$	: **array** [1..*n*] **of** BOOLEAN := *False*
$\mathcal{D}_i$	: **set of** PROCESS_ID
$recvProbe_i$	: **array** [1..*n*] **of** BOOLEAN := *False*
α_i	: INTEGER
k	: INTEGER
$deadlockDetected_i$	: BOOLEAN := *False*

Detekcja zakleszczenia w *modelu AND* (akcje)

AKCJE

```
1.  when e_start
        ( Q_α, DeadlockDetection ) do
2.   if passive_α
3.   then
4.      for all Q_k :: P_k ∈ D_α do
5.          probeOut.initIndex := α
6.          send( Q_α, Q_k, probeOut )
7.      end for
8.   end if
9.  end when

10. when e_activate( P_i ) do
11.     for all k ∈ {1, 2, ... , n} do
12.         recvProbe_i[k] := False
13.     end for
14. end when

15. when e_receive( Q_j, Q_i, probeIn : PROBE ) do
16.  α_i := probeIn.InitIndex
17.  if passive_i and ( ¬recvProbe_i[α_i] ) ∧ (¬granted_i[j] )
18.  then
19.      recvProbe_i[α_i] := True
20.      if α_i = i
21.      then
22.          deadlockDetected_i := True
23.          decide( deadlockDetected_i )
24.      else
25.          probeOut.initIndex := α
26.          for all Q_k :: P_k ∈ D_i do
27.              send( Q_i, Q_k, probeOut )
28.          end for
29.      end if
30.  end if
31. end when
```

Detekcja zakleszczenia dla *modelu OR* – przetwarzanie dyfuzyjne

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

Detekcja jest inicjowana przez jeden z monitorów Q_α, który wysyła do wszystkich monitorów procesów zbioru warunkującego $\mathcal{D}_\alpha$, wiadomość kontrolną `QUERY` zawierającą:

- indeks *initIndex* inicjatora Q_α,
- numer sekwencyjny *queryNo* przetwarzania detekcyjnego zainicjowanego przez Q_α tym zapytaniem.

W odpowiedzi na zapytanie `QUERY`, monitory Q_i oczekują odpowiedzi `REPLY` zawierającej takie same wartości *initIndex* i *queryNo*.

Detekcja zakleszczenia dla *modelu OR* (1)

Algorytm detekcji zakleszczenia dla *modelu OR* wykorzystuje następujące zmienne monitora:

- $maxQuerNo_i[j]$ – oznacza największy numer sekwencyjny *queryNo* spośród wszystkich zapytań `QUERY` zainicjowanych przez Q_j, a wysłanych lub odebranych przez Q_i.
- $engager_i[j]$ – indeks monitora Q_k, $k \neq i$, który spowodował zapisanie aktualnej wartości do $maxQuerNo_i[j]$.
- $QRBalance_i[j]$ – jest różnicą liczby zapytań `QUERY` zainicjowanych przez Q_j i wysłanych dalej przez oraz liczby odpowiedzi REPLY na te zapytania; wartość $QRBalance_i[j] = 0$ oznacza, że Q_i otrzymał odpowiedzi na wszystkie zapytania związane z ostatnim procesem detekcji zainicjowanym przez Q_j.
- $contPassive_i[j]$ – jest *True* wtedy i tylko wtedy gdy P_i pozostawał pasywny przez cały czas od momentu ostatniego uaktualniania zmiennej $maxQuerNo_i[j]$; początkowo $contPassive_i[j]$ jest równe *False*.

Detekcja zakleszczenia dla *modelu OR* – twierdzenia

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).

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 dla *modelu OR*

- Wstęp
- Typy komunikatów
- Deklaracje
- Procedury
- Akcje I
- Akcje II

Detekcja zakleszczenia w *modelu OR* (typy komunikatów)

➲ Wstęp
➲ Typy komunikatów
➲ Deklaracje
➲ Procedury
➲ Akcje I
➲ Akcje II

TYPY KOMUNIKATÓW

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

type QUERY extends CONTROL

type REPLY extends CONTROL
```


Detekcja zakleszczenia w *modelu OR* (deklaracje)

➲ Wstęp
➲ Typy komunikatów
➲ Deklaracje
➲ Procedury
➲ Akcje I
➲ Akcje II

DEKLARACJE

queryOut	: QUERY
replyOut	: REPLY
$\mathcal{D}_i$	: **set of** PROCESS_ID
$maxQueryNo_i$	: **array** [1..*n*] **of** INTEGER := 0
$engager_i$	: **array** [1..*n*] **of** INTEGER := 0
$QRBalance_i$	: **array** [1..*n*] **of** INTEGER := 0
$contPassive_i$	: **array** [1..*n*] **of** BOOLEAN := *False*
$queryNo_i$	: INTEGER
α_i	: INTEGER
$deadlockDetected_i$	: BOOLEAN := *False*

Detekcja zakleszczenia w *modelu OR* (akcje I)

➲ Wstęp
➲ Typy komunikatów
➲ Deklaracje
➲ Procedury
➲ Akcje I
➲ Akcje II

AKCJE

```
1.  when e_start ( Q_α, DeadlockDetection ) do
2.    if passive_α
3.    then
4.      maxQueryNo_α[α] := maxQueryNo_α[α] + 1
5.      contPassive_α[α] := True
6.      queryOut.queryNo := maxQueryNo_α[α]
7.      queryOut.initIndex := α
8.      for all Q_k :: P_k ∈ D_α do
9.        send( Q_α , Q_k , queryOut)
10.     end for
11.     QRBalance_α[α] := | D_α |
12.   end if
13. end when

14. when e_receive
      ( Q_j, Q_i, queryIn : QUERY) do
15.   if passive_i
16.   then
17.     queryNo_i := queryIn.queryNo
18.     α_i := queryIn.initIndex
19.     if queryNo_i > maxQueryNo_i[α_i]
20.     then
21.       maxQueryNo_i[α_i] := queryNo_i
22.       engager_i[α_i] := j
23.       contPassive_i[α_i] := True
24.       queryOut.queryNo := queryNo_i
25.       queryOut.initIndex := α_i
26.     for all Q_k :: P_k ∈ D_i do
27.         send( Q_i, Q_k, queryOut )
28.       end for
29.       QRBalance_i[α_i] := | D_i |
30.   else
31.       if contPassive_i[α_i] ∧
              queryNo_i := maxQueryNo_i[α_i]
32.       then
33.         replyOut.queryNo := queryNo_i
34.         replyOut.initIndex := α_i
35.         send( Q_i, Q_j, replyOut)
36.       end if
37.   end if
38. end if
39. end when
```


Detekcja zakleszczenia w *modelu OR* (akcje II)

➲ Wstęp
➲ Typy komunikatów
➲ Deklaracje
➲ Procedury
➲ Akcje I
➲ Akcje II

AKCJE

```
40.  when e_receive( Q_j , Q_i , replyIn : REPLY )
     do
41.   if passive_i
42.   then
43.     queryNo_i := queryIn.queryNo
44.     α_i := queryIn.initIndex
45.     if maxQueryNo_i[α_i ] = queryNo_i ∧
          contPassive_i[α_i ]
46.       then
47.       QRBalance_i[α_i ] := QRBalance_i[α_i ] - 1
48.       if QRBalance_i[α_i ] = 0
49.       then
50.         if α_i = i
51.         then
52.           deadlockDetected_i := True
53.           decide( deadlockDetected_i )
54.       else
55.         k := engager_i[α_i]
56.         replyOut.queryNo := queryNo_i
57.         replyOut.initIndex := α_i
58.         send( Q_i, Q_k, replyOut )
59.       end if
60.     end if
61.   end if
62.  end if
63. end when

64. when e_activate( P_i ) do
65.   for all k ∈ {1, 2, ... , n } do
66.     contPassive_i[k] := False
67.   end for
68. end when
```