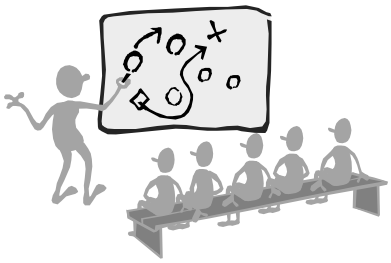


Podstawy przetwarzania rozproszonego

Konstrukcja spójnego obrazu stanu globalnego

Spis treści

Spis algorytmów

- Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *FIFO*
- Konstrukcja spójnego obrazu stanu globalnego z wykorzystaniem historii komunikacji oraz kolorowania wiadomości i procesów dla środowiska z kanałami *nonFIFO*
- Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *nonFIFO* z zastosowaniem zegarów wektorowych
- Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *nonFIFO* z zastosowaniem kolorowania wiadomości i procesów
- Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników *TF*
- Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników typu *BF* i *FF*

Stan globalny systemu

W systemach rozproszonych, bardzo wiele problemów praktycznych i teoretycznych sprowadza się wprost do ciągłej lub okresowej obserwacji stanu globalnego systemu.

Znajomość takiego stanu pozwala bowiem na wykrycie specyficznych sytuacji, niewidocznych z perspektywy pojedynczego procesu, i podjęcie stosownych działań.

Przykładami problemów redukujących się w istocie do oceny stanu globalnego są:

- śledzenie i sterowanie wykonywaniem programu rozproszonego (ang. monitoring and debugging),
- detekcja stanów awaryjnych (np. utraty wiadomości, zakleszczenia) lub też oczekiwanych (np. zakończenia obliczeń rozproszonych),
- dostosowywanie konfiguracji i funkcji systemu do zmieniającego się obciążenia (np. podział i równoważenie obciążenia, adaptacyjny wybór połączeń).

Modele stanów globalnych (1)

Założenia:

- rozważmy przetwarzanie rozproszone obejmujące trzy procesy P_1, P_2 i P_3, które współdzielą pewien zasób w trybie wzajemnego wykluczania.
- warunkiem koniecznym dostępu do współdzielonego zasobu (sekcji krytycznej) jest posiadanie w danej chwili unikalnego komunikatu – znacznika (ang. token).
- znacznik krąży między procesami połączonymi w logiczny pierścień (reprezentowany przez graf cykliczny) wskazując kolejno procesy uprawnione do operowania na zasobie współdzielonym.
- stan $S_i(\tau)$ procesu P_i w każdej chwili τ czasu globalnego (rzeczywistego) zdefiniowany jest przez trzy zmienne: $present_i(\tau)$, $outLog_i(\tau)$, $inLog_i(\tau)$.

$$S_i(\tau) = \langle\, present_i(\tau),\ outLog_i(\tau),\ inLog_i(\tau)\,\rangle$$

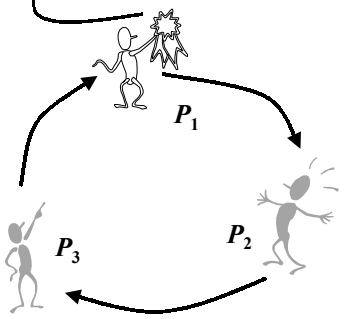
  - $present_i(\tau)$ przyjmuje wartość *True*, tylko wówczas, gdy znacznik typu `TOKEN` znajduje się w chwili τ w procesie P_i.
  - $outLog_i(\tau)$ jest kolejką znaczników wysłanych do chwili τ przez proces P_i.
  - $inLog_i(\tau)$ jest kolejką znaczników odebranych przez proces P_i do chwili τ.

Modele stanów globalnych (2)

Zakładamy, że w globalnym stanie początkowym $\Sigma^0 = \Sigma(\tau_0)$ znacznik znajduje się w procesie P_1, a wszystkie kolejki $outLog_i$, $inLog_i$, $i = 1, 2, 3$ – są puste.

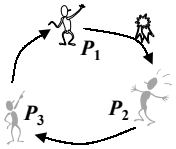
Tym samym:

$$\Sigma(\tau_0) = \langle\, \langle\, \mathbf{present_1(\tau_0) = True} \quad outLog_1(\tau_0) = \varnothing \quad inLog_1(\tau_0) = \varnothing\,\rangle$$
$$\langle\, present_2(\tau_0) = False \quad outLog_2(\tau_0) = \varnothing \quad inLog_2(\tau_0) = \varnothing\,\rangle$$
$$\langle\, present_3(\tau_0) = False \quad outLog_3(\tau_0) = \varnothing \quad inLog_3(\tau_0) = \varnothing\,\rangle\,\rangle$$

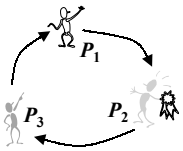


Modele stanów globalnych (3)

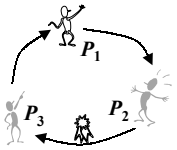
$\Sigma^1 = \langle \langle present_1(\tau_1) = False \quad outLog_1(\tau_1) = \{token_1\} \quad inLog_1(\tau_1) = \varnothing \rangle$
$\langle present_2(\tau_1) = False \quad outLog_2(\tau_1) = \varnothing \quad inLog_2(\tau_1) = \varnothing \rangle$
$\langle present_3(\tau_1) = False \quad outLog_3(\tau_1) = \varnothing \quad inLog_3(\tau_1) = \varnothing \rangle \rangle$



$\Sigma^2 = \langle \langle present_1(\tau_2) = False \quad outLog_1(\tau_2) = \{token_1\} \quad inLog_1(\tau_2) = \varnothing \rangle$
$\langle present_2(\tau_2) = True \quad outLog_2(\tau_2) = \varnothing \quad inLog_2(\tau_2) = \{token_1\} \rangle$
$\langle present_3(\tau_2) = False \quad outLog_3(\tau_2) = \varnothing \quad inLog_3(\tau_2) = \varnothing \rangle \rangle$



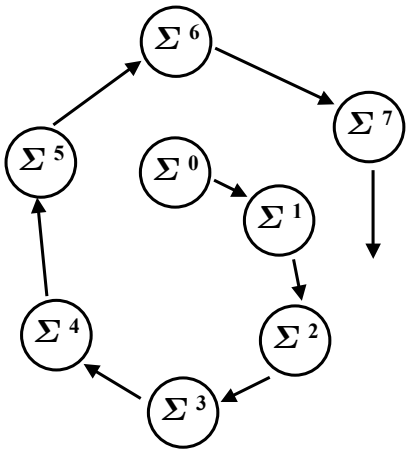
$\Sigma^3 = \langle \langle present_1(\tau_3) = False \quad outLog_1(\tau_3) = \{token_1\} \quad inLog_1(\tau_3) = \varnothing \rangle$
$\langle present_2(\tau_3) = False \quad outLog_2(\tau_3) = \{token_2\} \quad inLog_2(\tau_3) = \{token_1\} \rangle$
$\langle present_3(\tau_3) = False \quad outLog_3(\tau_3) = \varnothing \quad inLog_3(\tau_3) = \varnothing \rangle \rangle$

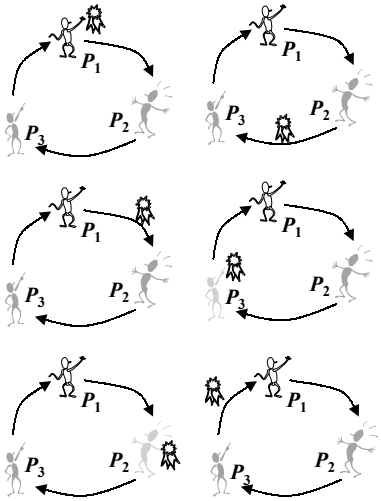


itd...

(c) Zakład Systemów Informatycznych Slajd 22

Graf stanów osiągalnych





(c) Zakład Systemów Informatycznych Slajd 23

Modele stanów globalnych (4)

Przedstawiony przykład ilustruje dotychczasowe rozumienie stanu lokalnego procesów oraz stanu globalnego przetwarzania rozproszonego – jako historii przetwarzania.

Należy zaznaczyć, że rozumienie to pozwala na określenie w każdej chwili aktualnego stanu kanałów na podstawie stanów lokalnych procesów.

Zalety :

- ogranicza liczbę składowych stanu globalnego do liczby procesów ($\Sigma \subseteq \mathcal{S}_1 \times \mathcal{S}_2 \times ... \times \mathcal{S}_n$)

Wady :

- podejście jest adekwatne tylko dla systemów z kanałami niezawodnymi.
- poważną trudność stanowi konieczność pamiętania wszystkich dotychczas wysłanych i odebranych wiadomości. Z upływem czasu rośnie bowiem istotnie koszt przechowywania i przetwarzania tej informacji

(c) Zakład Systemów Informatycznych Slajd 24

Modele stanów globalnych (5)

W praktyce stosowane jest też podejście alternatywne, polegające na definiowaniu stanu globalnego jako złożenia stanów lokalnych procesów i stanów kanałów komunikacyjnych.

Prowadzi to do reprezentacji stanu globalnego przez $n+m$ składowych odpowiadających stanom lokalnym wszystkich n procesów i wszystkich m kanałów.

Liczba składowych jest w tym wypadku oczywiście większa, lecz reprezentacja stanu procesu może być dużo prostsza i w rezultacie efektywniejsza z punktu widzenia zajętości pamięci i kosztów przetwarzania.

(c) Zakład Systemów Informatycznych Slajd 25

Modele stanów globalnych (6)

Założenia:

- rozważmy przetwarzanie rozproszone obejmujące trzy procesy P_1, P_2 i P_3, które współdzielą pewien zasób w trybie wzajemnego wykluczania.
- warunkiem koniecznym dostępu do współdzielonego zasobu (sekcji krytycznej) jest posiadanie w danej chwili unikalnego komunikatu – znacznika (ang. token).
- znacznik krąży między procesami połączonymi w logiczny pierścień (reprezentowany przez graf cykliczny) wskazując kolejno procesy uprawnione do operowania na zasobie współdzielonym.
- proces P_i jest w każdej chwili w stanie określonym przez zmienną logiczną $present_{i,}$ oraz przez liczniki $sentNo_i$ (ang. sent number) oraz $recvNo_i$ (ang. receive number), których wartości są odpowiednio równe liczbie dotychczas wysłanych i odebranych znaczników typu `TOKEN`.
- stan kanału $L_{i,j}$ określony może być przez zbiór znaczników znajdujących się aktualnie w kanale $C_{i,j}$

Możliwa jest prostsza reprezentacja:

- stan procesu określony przez zmienną $present_i$ (posiadanie znacznika)
- stan kanału określony przez zmienną $present_{i,j}$ (znacznik w kanale)

Modele stanów globalnych (7)

Niech dla prostoty zapisu „1” oznacza obecność znacznika, a „0” – jego brak. W efekcie stan globalny $\Sigma(\tau)$ możemy przedstawić w naszym przypadku w sposób następujący:

$$\Sigma(\tau) = \langle S_1(\tau), S_2(\tau), S_3(\tau), L_{1,2}(\tau), L_{2,3}(\tau), L_{3,1}(\tau) \rangle$$

Tym samym kolejne stany mają postać:

$\Sigma^0 = \langle 1, 0, 0, 0, 0, 0 \rangle$

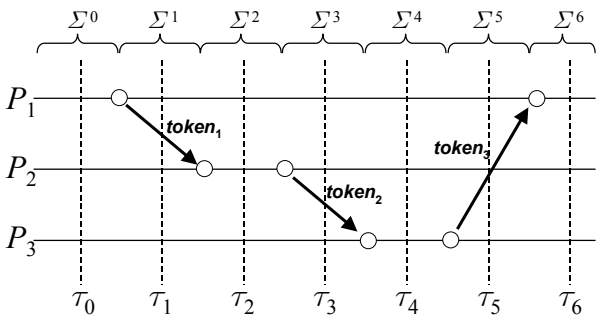
$\Sigma^1 = \langle 0, 0, 0, 1, 0, 0 \rangle$

$\Sigma^2 = \langle 0, 1, 0, 0, 0, 0 \rangle$

$\Sigma^3 = \langle 0, 0, 0, 0, 1, 0 \rangle$

$\Sigma^4 = \langle 0, 0, 1, 0, 0, 0 \rangle$

$\Sigma^5 = \langle 0, 0, 0, 0, 0, 1 \rangle$



Ocena stanów globalnych

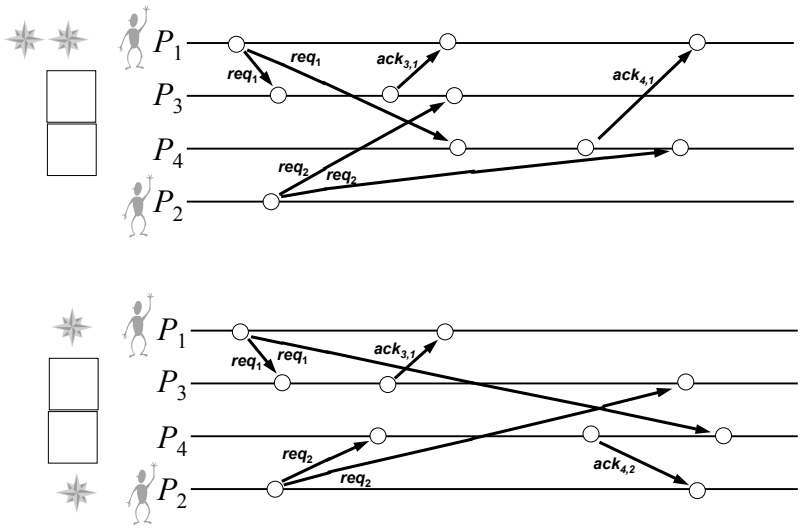
Ocena stanów globalnych w wybranych momentach czasu może mieć na celu sprawdzenie, czy rzeczywiście w każdej chwili w systemie jest dokładnie jeden znacznik.

- ✓ Zaginięcie znacznika, wynikające na przykład z błędu programu w którymkolwiek węźle, uniemożliwia procesom trwale wejście do sekcji krytycznej, co może prowadzić do blokady całego systemu.
- ✓ Stwierdzenie wystąpienia dwóch lub więcej znaczników oznacza możliwość jednoczesnego działania dwóch lub więcej procesów w sekcjach krytycznych, co w ogólności prowadzi do błędów współbieżnego działania procesów.

Przedstawione reprezentacje stanów globalnych nie są równoważne.

- ✓ Pierwsza, operuje na informacji najpełniejszej: unikalnych znacznikach i całej historii komunikacji.
- ✓ Druga reprezentacja, utożsamia wszystkie znaczniki i wyróżnia tylko stany ich obecności oraz nieobecności w poszczególnych procesach i kanałach.

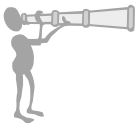
Różne realizacje przetwarzania rozproszonego *klient-serwer* jako wynik nieprzewidywalnych czasów transmisji



Problem jednoczesnego wyznaczenia poszczególnych składowych stanu globalnego

Cechy asynchronicznego środowiska rozproszonego:

- jedynym mechanizm komunikacji i synchronizacji to wymiana komunikatów,
- brak wspólnej pamięci,
- brak wspólnego zegara,
- asynchronizm przetwarzania,
- nieprzewidywalne czasy transmisji (opóźnienia komunikacyjne),



Dotychczas zakładano niejawnie istnienie *idealnego obserwatora zewnętrznego*, który ma dostęp jednocześnie do wszystkich procesów tworzących przetwarzanie co nie jest niestety możliwe w systemie asynchronicznym.

Proces chcący wyznaczyć stan globalny może zatem tylko wysyłać żądania wyznaczania stanów lokalnych, a po otrzymaniu odpowiedzi, konstruować na ich podstawie stan globalny.

Otrzymane składowe stany lokalne procesów mogą być :

- przestarzałe,
- niekompletne,
- odpowiadać konfiguracjom niespójnym – reprezentującym stany nieosiągalne

(c) Zakład Systemów Informatycznych Slajd 30

Problem konstrukcji stanu globalnego

Trudność problemu konstrukcji spójnego obrazu stanu globalnego przetwarzania rozproszonego wynika z:

- braku zegara globalnego,
- asynchronizmu przetwarzania.

Rozważmy na początek problem przy następujących dodatkowych założeniach:

- dostępny jest dla wszystkich procesów globalny zegar czasu rzeczywistego,
- znane jest maksymalne opóźnienie komunikacji,
- względne prędkości przetwarzania w poszczególnych węzłach są ograniczone.

Koncepcja wyznaczenia spójnego obrazu jest bardzo prosta. Wystarczy, by dowolny proces monitora Q_i, wybrał chwilę τ_s na tyle odległą w przyszłości, by zagwarantować, że wiadomość wysłana w danej chwili dotrze do wszystkich procesów Q_j przed τ_s. W celu wyznaczenia stanu kanałów, można przyjąć, że każda wiadomość zawiera etykietę czasową, określającą czas rzeczywisty wysłania wiadomości.

(c) Zakład Systemów Informatycznych Slajd 31

Koncepcja konstrukcji obrazu spójnego

1) Proces inicjatora Q_α wysyła do monitorów wszystkich procesów przetwarzania rozproszonego wiadomość: „zapamiętaj stan lokalny w chwili τ_s".
2) Monitory Q_i procesów zapamiętują stan lokalny S_i odpowiadających im procesów aplikacyjnych w chwili τ_s i natychmiast wysyłają wiadomość kontrolną do wszystkich monitorów $Q_j \in \mathcal{Q}_i^{OUT}$ (wiadomość ta jest wysyłana wszystkimi kanałami wyjściowymi). Zapamiętują też stan $L_{k,i}$ kanałów wejściowych jako zbiorów wszystkich wiadomości aplikacyjnych, które zostały wysłane przed τ_s a dotarły do Q_i po chwili τ_s. Zapamiętywanie stanu lokalnego oraz wysyłanie wiadomości kontrolnej poprzedza przy tym zajście jakiegokolwiek zdarzenia w procesie aplikacyjnym P_i, dla $\tau \geqslant \tau_s$.
3) Gdy monitor Q_i otrzyma przez kanał $C_{j,i}$ pierwszą wiadomość o etykiecie czasowej większej od τ_s (w szczególności może to być wiadomość kontrolna), aktualną wartość $L_{j,i}$ uznaje jako stan tego kanału w chwili τ_s.
4) Gdy monitor Q_i otrzyma wiadomości o etykiecie czasowej większej od τ_s przez wszystkie swoje kanały wejściowe, przesyła stan lokalny zapamiętany w chwili τ_s oraz wyznaczone stany kanałów wejściowych do inicjatora detekcji Q_α.
5) Po odebraniu od wszystkich monitorów wiadomości zawierających stany lokalne oraz stany ich kanałów wejściowych w chwili τ_s, inicjator Q_α konstruuje obraz stanu globalnego Γ.

Dowód poprawności koncepcji

Skonstruowany obraz reprezentuje stan globalny, który wystąpił w chwili τ_s. Argumentując poprawność bardziej formalnie, zauważmy, że wyznaczony obraz uwzględnia wszystkie zdarzenia lokalne, które zaszły przed chwilą τ_s. Stąd,

$$(E' \in H_i(\tau_s) \wedge \mathcal{RT}(E) < \mathcal{RT}(E')) \Rightarrow (E \in H_i(\tau_s))$$

gdzie $\mathcal{RT}(E)$ jest czasem rzeczywistym zajścia zdarzenia E. Ponieważ zegar czasu rzeczywistego spełnia warunek poprawności zegara

$$(E \mapsto E') \Rightarrow \mathcal{RT}(E) < \mathcal{RT}(E'),$$

otrzymujemy

$$(E' \in H_i(\tau_s) \wedge E \mapsto E') \Rightarrow (E \in H_i(\tau_s))$$

a to właśnie należało wykazać.

Konstrukcja spójnego obrazu dla środowiska z kanałami FIFO (1)

Chandy i Lamport jako pierwsi przedstawili rozwiązanie problemu konstrukcji obrazu spójnego w środowisku rozproszonym.

Założenia dotyczące środowiska przetwarzania:

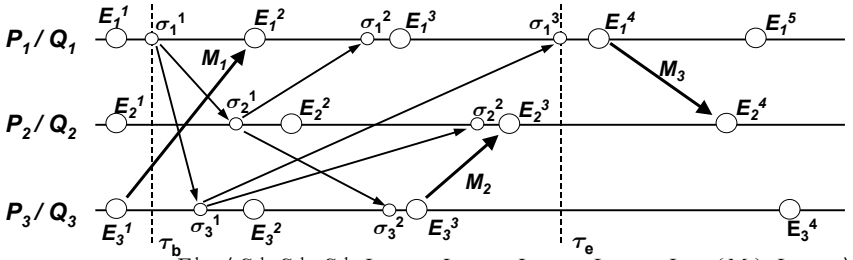
- ❑ niezawodne kanały zachowują uporządkowanie wiadomości (niezawodne kanały FIFO)
- ❑ stan reprezentowany jest w postaci złożenia lokalnych stanów procesów i stanów kanałów
- ❑ pełen asynchronizm komunikacji i przetwarzania
- ❑ brak zegara globalnego

Konstrukcja spójnego obrazu dla środowiska z kanałami FIFO (2)

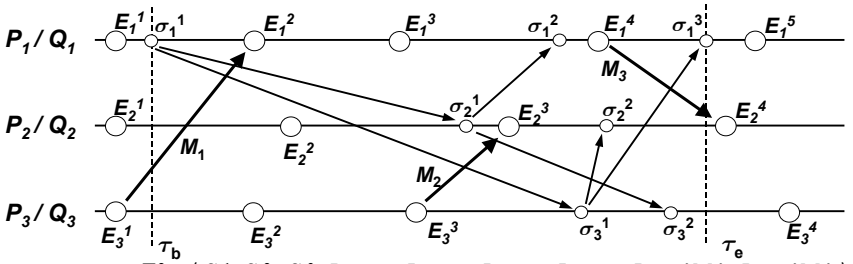
Założenia algorytmu:

- ❑ Pewien monitor Q_α (inicjator) spontanicznie podejmuje decyzje o zainicjowaniu *procesu konstrukcji* (*detekcji*) spójnego obrazu stanu globalnego.
- ❑ W pierwszej kolejności Q_α zapamiętuje stan lokalny skojarzonego z nim procesu aplikacyjnego P_α i wysyła wiadomość kontrolną (znacznik) typu MARKER, do wszystkich incydentnych monitorów.
- ❑ Każdy monitor Q_i po odebraniu znacznika z kanału $C_{j,i}$, sprawdza czy jest to pierwszy znacznik odebrany w danym procesie detekcji (tzn. czy stan procesu został już w tym procesie detekcji zapamiętany).
- ❑ Jeżeli jest to pierwszy znacznik, to Q_i zapamiętuje aktualny stan S_i procesu P_i, uznaje stan kanału $C_{j,i}$ za pusty, i propaguje znacznik przez wszystkie swoje kanały wyjściowe (wysłanie znacznika musi poprzedzić zdarzenie wysłania danym kanałem kolejnej, po zapamiętaniu stanu, wiadomości aplikacyjnej)
- ❑ Jeżeli stan procesu P_i został już wcześniej zapamiętany, to Q_i uznaje za stan kanału $C_{j,i}$ zbiór tych wszystkich wiadomości aplikacyjnych, które dotarły tym kanałem po zapamiętaniu stanu a przed otrzymaniem znacznika.
- ❑ Po odebraniu znaczników ze wszystkich kanałów wejściowych, monitor Q_i przesyła zapamiętany stan lokalny $procState_i$ procesu P_i oraz stan $chanState_i$ wszystkich kanałów wejściowych do monitora Q_β, konstruującego obraz stanu globalnego.
- ❑ W szczególności możliwe jest oczywiście, że $Q_\alpha = Q_\beta$.

Przykład konstrukcji spójnego obrazu z zastosowaniem algorytmu Chandy-Lamporta



$\Gamma^1 = \langle S_1^1, S_2^1, S_3^1, L_{1,2}=\varnothing, L_{1,3}=\varnothing, L_{2,1}=\varnothing, L_{2,3}=\varnothing, L_{3,1}=\{M_1\}, L_{3,2}=\varnothing \rangle$



$\Gamma^2 = \langle S_1^1, S_2^2, S_3^3, L_{1,2}=\varnothing, L_{1,3}=\varnothing, L_{2,1}=\varnothing, L_{2,3}=\varnothing, L_{3,1}=\{M_1\}, L_{3,2}=\{M_2\} \rangle$

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *FIFO*

- Wstęp
- Typy komunikatów
- Deklaracje
- Procedury
- Akcje

Algorytm Chandy-Lamporta.

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *FIFO* (typy komunikatów)

TYPY KOMUNIKATÓW

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

type MARKER **extends** FRAME

type STATE **extends** FRAME **is record of**
procState : PROCESS_STATE
chanState : **array** [1..*n*] **of set of** MESSAGE
end record

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *FIFO* (deklaracje)

DEKLARACJE

msgIn : MESSAGE
pcktOut : PACKET
markerOut : MARKER
stateOut : STATE
$recvMark_i$: **array** [1..*n*] **of** BOOLEAN := *False*
$chanState_i$: **array** [1..*n*] **of set of** MESSAGE := ∅
$procState_i$: PROCESS_STATE
$\mathcal{C}_i^{IN}$: **set of** CHANNEL_ID
$\mathcal{C}_i^{OUT}$: **set of** CHANNEL_ID
$involved_i$: BOOLEAN := *False*

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *FIFO* (procedury)

➲ Wstęp
➲ Typy komunikatów
➲ Deklaracje
➲ Procedury
➲ Akcje

PROCEDURY

```
1.  procedure RECORDSTATE( ) do
2.    procState_i := S_i
3.    for all C_{j,i} ∈ C_i^IN do
4.          chanState_i[j] := ∅
5.    end for
6.    involved_i := True
7.    send(Q_i, Q_i^IN, markerOut )
8.  end procedure
```


Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *FIFO* (akcje)

➲ Wstęp
➲ Typy komunikatów
➲ Deklaracje
➲ Procedury
➲ Akcje

AKCJE

```
1.  when e_start( Q_α, TakeSnapshot ) do
2.     RECORDSTATE()
3.     recvMark_α[α] := True
4.     if C_α^IN = ∅ then
5.       stateOut.procState := procState_α
6.       stateOut.chanState := chanState_α
7.       send(Q_α, Q_β, stateOut)
8.     end if
9.  end when

10. when e_send
      (P_i, P_j, msgOut : MESSAGE) do
11.    pcktOut.data := msgOut
12.    send(Q_i, Q_j, pcktOut)
13. end when

14. when e_receive(Q_j, Q_i, markerIn: MARKER) do
15.  if ¬ involved_i then
16.    RECORDSTATE( )
17.  end if
18.  recvMark_i[j] := True
19.  if ∀ C_{j,i} ∈ C_i^IN :: recvMark_i[j] then
20.    stateOut.procState := procState_i
21.    stateOut.chanState := chanState_i
22.    send(Q_i, Q_β, stateOut)
23.  end if
24. end when

25. when e_receive (Q_j, Q_i, pcktIn : PACKET) do
26.    msgIn := pcktIn.data
27.    if involved_i ∧ ¬ recvMark_i[j] then
28.      chanState_i[j] := chanState_i[j]∪{msgIn}
29.    end if
30.    deliver (P_j, P_i, msgIn)
31. end when
```

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *FIFO* – twierdzenia

Algorytm Chandy-Lamporta wyznacza w skończonym czasie konfigurację spójną.

Jeżeli proces konstrukcji konfiguracji spójnej rozpoczął się w chwili τ_b, a zakończył w chwili τ_e, to wyznaczona konfiguracja Γ, reprezentująca pewien stan globalny Σ, jest osiągalna ze stanu $\Sigma(\tau_b)$, a stan globalny $\Sigma(\tau_e)$ jest osiągalny ze stanu $\Sigma = \Gamma$. Tym samym:

$$\Sigma(\tau_b) \rightsquigarrow \Gamma \rightsquigarrow \Sigma(\tau_e)$$

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *FIFO*

Przyjmujemy, że *graf zorientowany* odpowiadający topologii przetwarzania rozproszonego scharakteryzowany jest przez następujące parametry :

- n – liczba wierzchołków grafu (procesów),
- m – liczba krawędzi grafu (jednokierunkowych kanałów komunikacyjnych),
- d – średnica grafu,
- l – długość najdłuższej ścieżki w grafie.

Zatem złożoność czasowa algorytmu Chandy-Lamporta wynosi $d + 1$, a złożoność komunikacyjna, w sensie liczby przesyłanych znaczników, wynosi m.

Konstrukcja spójnego obrazu dla środowiska z kanałami *nonFIFO* (1)

Lai i Yang przedstawili algorytm, który nie wykorzystuje znaczników i nie wymaga by kanały były typu FIFO.

W algorytmie zakłada się, że reprezentacje stanów lokalnych obejmują *historię komunikacji*, a więc odpowiednie zbiory wiadomości dotychczas wysłanych i odebranych.

τ_i – wysłanie znacznika

τ_j' – odebranie znacznika

$$outLog_{i,j}(\tau_i) = inLog_{i,j}(\tau_j')$$

W efekcie, monitor Q_β konstruujący obraz spójny może łatwo wyznaczyć [illegible] poszczególnych kanałów po zapamiętaniu stanów lokalnych i komunikacji od wszystkich mo[illegible] jako różnicę

P_i P_j

$$outLog_{i,j}(\tau_i) \setminus inLog_{i,j}(\tau_j) \text{ ; gdzie } \tau_j \leqslant \tau_j'.$$

Nie ma więc też potrzeby specjalnego odnotowywania wiadomości odebranych z danego kanału po zapamiętaniu stanu lokalnego (τ_j) a przed otrzymaniem znacznika (τ_j').

(c) Zakład Systemów Informatycznych Slajd 44

Konstrukcja spójnego obrazu dla środowiska z kanałami *nonFIFO* (2)

Jeśli po zapamiętaniu stanu proces P_i już nigdy nie wyśle wiadomości do P_j, to nie ma potrzeby przesłania do Q_j znacznika.

Jeśli wiadomości będą w dalszym ciągu wysyłane, to dołączany jest do nich znacznik w formie etykiety, a tym samym odróżniane są te pakiety od wszystkich pakietów wysłanych przed zapamiętaniem stanu.

Wyróżnione pakiety ze znacznikiem przechwytywane są przez monitor Q_j odbiorcy i powodują najpierw zapamiętanie stanu procesu P_j, a dopiero w następnej kolejności przekazanie wiadomości aplikacyjnej procesowi P_j.

Żadna z wiadomości wysłanych przez P_i do P_j po zapamiętaniu stanu procesu P_i, nie zostanie odebrana przez P_j przed zapamiętaniem jego stanu. Oznacza to dalej, że spełniona będzie relacja:

$$inLog_{i,j}(\tau_j) \subseteq outLog_{i,j}(\tau_i)$$

która gwarantuje spójność konstruowanego obrazu stanu globalnego przetwarzania rozproszonego.

(c) Zakład Systemów Informatycznych Slajd 45

Konstrukcja spójnego obrazu stanu globalnego z wykorzystaniem historii komunikacji oraz kolorowania wiadomości i procesów dla środowiska z kanałami *nonFIFO*

Algorytm Lai-Yanga.

Algorytm przydziela procesom (kontrolerom) oraz wiadomościom jeden z dwóch kolorów ***White*** i ***Red***.

TYPY KOMUNIKATÓW

```
type PACKET extends FRAME is record of
    colour      : enum {White, Red}
    data        : MESSAGE
 end record

type STATE extends FRAME is record of
    procState   : PROCESS_STATE
    sentLog     : set of MESSAGE
    recvLog     : set of MESSAGE
 end record
```

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z wykorzystaniem historii komunikacji oraz kolorowania wiadomości i procesów dla środowiska z kanałami *nonFIFO* (deklaracje)

DEKLARACJE

msgIn : MESSAGE
pcktOut : PACKET
dummyOut : PACKET
stateOut : STATE
$procState_i$: PROCESS_STATE
$procColour_i$: **enum** {*White*, *Red*} := *White*
$sentLog_i$: **set of** MESSAGE := ∅
$outLog_i$: **set of** MESSAGE := ∅
$recvLog_i$: **set of** MESSAGE := ∅
$inLog_i$: **set of** MESSAGE := ∅

znacznik inicjujący

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z wykorzystaniem historii komunikacji oraz kolorowania wiadomości i procesów dla środowiska z kanałami *nonFIFO* (procedury)

PROCEDURY

1. **procedure** RECORDSTATE( ) **do**
2. $procState_i := S_i$
3. $sentLog_i := outLog_i$
4. $recvLog_i := inLog_i$
5. **end procedure**

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z wykorzystaniem historii komunikacji oraz kolorowania wiadomości i procesów dla środowiska z kanałami *nonFIFO* (akcje)

➲ Wstęp
➲ Typy komunikatów
➲ Deklaracje
➲ Procedury
➲ Akcje

AKCJE

1. **when** *e_start*(Q_α, TakeSnapshot) **do**
2. RECORDSTATE()
3. $procColour_\alpha := Red$
4. $dummyOut.pcktColour := Red$
5. $dummyOut.data := \varnothing$
6. **send**(Q_α, $\mathcal{Q} \setminus \{Q_\alpha\}$, *dummyOut*)
7. $stateOut.procState := procState_\alpha$
8. $stateOut.sentLog := sentLog_\alpha$
9. $stateOut.recvLog := recvLog_\alpha$
10. **send**(Q_α, Q_β , *stateOut*)
11. **end when**

12. **when** *e_send*(P_i, P_j, *msgOut* : MESSAGE) **do**
13. $outLog_i := outLog_i \cup \{msgOut\}$
14. $pcktOut.colour := procColour_i$
15. $pcktOut.data := msgOut$
16. **send**(Q_i, Q_j, *pcktOut*)
17. **end when**

18. **when** *e_receive* (Q_j, Q_i, *pcktIn* : PACKET) **do**
19. **if** $pcktIn.colour = Red \wedge procColour_i = White$
20. **then**
21. $procColour_i := Red$
22. $stateOut.procState := procState_i$
23. $stateOut.sentLog := sentLog_i$
24. $stateOut.recvLog := recvLog_i$
25. **send**(Q_i, Q_β , *stateOut*)
26. **end if**
27. $msgIn := pcktIn.data$
28. **if msgIn** $\neq \varnothing$
29. **then**
30. $inLog_i := inLog_i \cup \{msgIn\}$
31. **deliver**(P_j,P_i, *msgIn*)
32. **end if**
33. **end when**

Algorytm wykorzystujący historię komunikacji oraz kolorowanie wiadomości i procesów (1)

Przypisanie odpowiedniego koloru pakietowi oznacza:

- ***White*** – pakiet bez znacznika
- ***Red*** – dołączenie znacznika do wiadomości

Analogicznie, kolory przypisane są procesom (monitorom):

- ***White*** – proces nie zapamiętał jeszcze stanu
- ***Red*** – stan procesu został już zapamiętany

Ogólna idea algorytmu jest następująca:

1) Każdy proces jest postrzegany przez monitor początkowo jako ***White***, a monitor zmienia jego kolor na ***Red*** po zapamiętaniu stanu procesu.
2) Każdej wiadomości wysłanej przez proces koloru ***White*** przypisany jest kolor ***White***, a każdej wysłanej przez proces ***Red*** – kolor ***Red*** .
3) Stan procesu koloru ***White*** można zapamiętać w dowolnej chwili, lecz koniecznie przed odebraniem wiadomości koloru ***Red*** .

Algorytm wykorzystujący historię komunikacji oraz kolorowanie wiadomości i procesów (2)

Zaniechanie wysyłania znaczników, a jedynie dopisywanie pewnej informacji do wiadomości aplikacyjnych (kolorowanie pakietów), grozi tym, że:

- w wypadku braku komunikacji na poziomie aplikacyjnym stany pewnych procesów nigdy nie zostaną zapamiętane,
- konfiguracja globalna nigdy nie zostanie wyznaczona.

Aby takiej sytuacji zapobiec, inicjator może wysłać specjalny pakiet sterujący z pustym polem danych, do monitorów wszystkich procesów tworzących przetwarzanie. Przy takim rozszerzeniu algorytm Lai-Yanga wyznacza spójny obraz w skończonym czasie.

Niech pakiet sterujący będzie wysyłany kanałami odpowiadającymi krawędziom drzewa rozpinającego o wysokości d, grafu reprezentującego topologię przetwarzania. Pomijając fazę zbierania informacji z poszczególnych węzłów i koszt zmiany koloru pakietów, to złożoność czasowa i komunikacyjna algorytmu Lai-Yanga wynosi odpowiednio d i n-1.

Algorytm stosujący zegary wektorowe (1)

Rozważamy system, w którym kanały są kanałami *nonFIFO*.

Algorytm wyznaczający stan globalny dla takiego systemu wywodzi się z koncepcji wyznaczania stanu globalnego z użyciem zegara czasu rzeczywistego.

Ogólna koncepcja:

- określenie momentu czasu τ_s odpowiadającego przyszłości, w którym zostaną zapamiętane stany lokalne wszystkich procesów.
- po wyznaczeniu stanów lokalnych w momencie τ_s, monitory przesyłają informację o stanach lokalnych do monitora konstruującego obraz globalny.

W celu adaptacji powyższej koncepcji do asynchronicznego systemu rozproszonego, proces inicjatora określa wektorowy czas wirtualny momentu zapamiętania w przyszłości stanu lokalnego procesu.

Algorytm stosujący zegary wektorowe (2)

Wykorzystywana jest tu następująca właściwość zegarów wektorowych:

W chwili, gdy uaktualniony został zegar wektorowy procesu, zachodzi:

$$\nexists\, P_j :: P_j \in \mathcal{P} :: vClock_i < vClock_j$$

Powyższa właściwość oznacza w praktyce, że zmiana wektorowego zegara lokalnego w pewnym procesie P_i określa czas wirtualny, który z perspektywy wszystkich pozostałych procesów P_j ($i \neq j$) z pewnością nie jest czasem odnoszącym się do przeszłości.

Algorytm stosujący zegary wektorowe (3)

Algorytm stosujący zegary wektorowe wykorzystuje powyższe spostrzeżenie i wygląda następująco:

- Monitor inicjatora zwiększa pozornie wartość swojego zegara wektorowego i tym samym wyznacza moment z jego perspektywy w przyszłości.
- O czasie tym informuje wszystkie pozostałe monitory, wstrzymując jednocześnie postęp lokalnego czasu wirtualnego (blokuje postęp przetwarzania aplikacyjnego).
- Dopiero po upewnieniu się, że wszystkie monitory znają wyznaczony moment w przyszłości, uwalniane jest przetwarzanie aplikacyjne a więc również zegar lokalny.
- W wyniku dalszej komunikacji między procesami aplikacyjnymi, bądź w wyniku przesłania dodatkowych wiadomości synchronizujących między monitorami, każdy z procesów osiągnie w końcu, lub będzie usiłował przekroczyć, wyznaczony moment τ_s.
- Przed przekroczeniem wartości τ_s, zapamiętany zostanie stan lokalny procesu.

Kolekcja takich stanów tworzy konfigurację spójną. W algorytmie dla uproszczenia, będziemy dalej zakładać, że $Q_\alpha = Q_\beta$.

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *nonFIFO* z zastosowaniem zegarów wektorowych

Algorytm Matterna.

TYPY KOMUNIKATÓW

```
type PACKET extends FRAME is record of
      vClock   : array [1..n] of INTEGER
      data     : MESSAGE
 end record

type CONTROL extends FRAME is record of
      vRecordClock : array [1..n] of INTEGER
 end record

type PROC_STATE extends FRAME is record of
      procState : PROCESS_STATE
 end record

type CHAN_STATE extends FRAME is record of
      chanState : MESSAGE
 end record

type ACK extends FRAME
```

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *nonFIFO* z zastosowaniem zegarów wektorowych (deklaracje)

DEKLARACJE

msgIn	: MESSAGE
pcktOut	: PACKET
dummyOut	: PACKET
ctrlOut	: CONTROL
ackIn, *ackOut*	: ACK
procStateOut	: PROC_STATE
chanStateOut	: CHAN_STATE
$vClock_i$	: **array** [1..*n*] **of** INTEGER
$vRecordClock_i$	: **array** [1..*n*] **of** INTEGER := $\{\infty, \infty, ..., \infty\}$
$procState_i$	: PROCESS_STATE
$recorded_i$	: BOOLEAN := *False*

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *nonFIFO* z zastosowaniem zegarów wektorowych (akcje)

AKCJE

1. **when** $e\_start(Q_\alpha, TakeSnapshot)$ **do**
2. $ctrlOut.vRecordClock := vClock_\alpha$
3. $ctrlOut.vRecordClock[\alpha] := ctrlOut.vRecordClock[\alpha] + 1$
4. **send**(Q_α, $\mathcal{Q} \setminus \{Q_\alpha\}$, *ctrlOut*)
5. **for all** $Q_k \in \mathcal{Q} \setminus \{Q_\alpha\}$ **do**
6. **receive** (Q_k, Q_α, *ackIn*)
7. **end for**
8. $vClock_\alpha[\alpha] := vClock_\alpha[\alpha] + 1$
9. $procState_\alpha := S_\alpha$
10. $recorded_\alpha := True$
11. $dummyOut.vClock := vClock_\alpha$
12. $dummyOut.data := \varnothing$
13. **send**(Q_α, $\mathcal{Q} \setminus \{Q_\alpha\}$, *dummyOut*)
14. **end when**

15. **when** $e\_receive(Q_j, Q_i, ctrlIn$: CONTROL) **do**
16. $vRecordClock_i := ctrlIn.vRecordClock$
17. **send** (Q_i, Q_j, *ackOut*)
18. **end when**

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *nonFIFO* z zastosowaniem zegarów wektorowych (akcje)


```
19.  when e_send( P_i, P_j, msgOut : MESSAGE ) do
20.    vClock_i[i] := vClock_i[i] + 1
21.    pcktOut.vClock := vClock_i
22.    pcktOut.data := msgOut
23.    send ( Q_i, Q_j, pcktOut )
24.  end when

25.  when e_receive( Q_j, Q_i, pcktIn : PACKET ) do
26.    for all k ∈ {1, 2, ... , n} do
27.      vClock_i[k] := max( vClock_i[k], pcktIn.vClock[k] )
28.    end for
29.    msgIn := pcktIn.data
30.    if recorded_i ∧ pcktIn.vClock ≱ vRecordClock_i
31.     then
32.       chanStateOut.chanState := msgIn
33.       send( Q_i, Q_α, chanStateOut )
34.    end if
35.  if ¬ recorded_i ∧ vClock_i ⩾ vRecordClock_i
36.   then
37.      procState_i := S_i
38.      procStateOut.procState := procState_i
39.      send( Q_i, Q_α, procStateOut )
40.    recorded_i := True
41.  end if
42.  if msgIn ≠ ∅
43.   then
44.     deliver( P_j, P_i, msgIn )
45.  end if
46. end when
```

(c) Zakład Systemów Informatycznych Slajd 60

Algorytm stosujący kolorowanie wiadomości i procesów (1)

Wadą przedstawionego algorytmu jest zawieszanie przetwarzania aplikacyjnego (procesu P_α) do momentu uzyskania przez Q_α potwierdzenia od wszystkich monitorów $Q_j \neq Q_\alpha$ o odebraniu pakietu zawierającego zadany moment zapamiętania w przyszłości stanu lokalnego.

Problemu tego można uniknąć wprowadzając dodatkową n+1 pozycję w wektorze czasu, której wartość jest zwiększana tylko przez wyróżniony monitor $Q_{n+1} = Q_\alpha$ nie skojarzony z żadnym procesem aplikacyjnym.

W nowym wektorze czasu $vClock_i'$ istotne znaczenie będzie miała teraz tylko pozycja n+1, a wszystkie pozostałe mogą być pominięte. W konsekwencji można też pominąć całą fazę rozgłaszania wartości *vRecordClock*. W tym wypadku procesy będą zapamiętywać stan lokalny, gdy uzyskają wektor $vClock_i'$ z nową, powiększoną wartością pozycji n+1.

(c) Zakład Systemów Informatycznych Slajd 61

Algorytm stosujący kolorowanie wiadomości i procesów (2)

W kontekście detekcji stanu globalnego znaczenie ma w istocie tylko zmiana stanu na pozycji n+1, i stąd dwa stany tej pozycji są wystarczające dla rozróżnienia kolejnych faz wyznaczania obrazu globalnego:

- ***White*** – przed zapamiętaniem stanu lokalnego procesu
- ***Red*** – po zapamiętaniu stanu lokalnego.

Podobnie, kolory można przypisać pakietom. Kolor ***White*** i ***Red*** pakietu oznacza, że został on wysłany, odpowiednio, przed lub po zapamiętaniu stanu lokalnego procesu. Monitory procesów *White* wysyłają tylko pakiety ***White***, a monitory procesów ***Red*** – tylko pakiety koloru ***Red***.

- Każdy proces pierwotnie ma kolor ***White***, a staje się ***Red*** po zapamiętaniu jego stanu lokalnego a przed przekazaniem mu pierwszej wiadomości z pakietu koloru ***Red***.
- Inicjator detekcji stanu globalnego zapamiętuje stan lokalny procesu, staje się ***Red*** i wysyła pusty pakiet *dummyOut* koloru ***Red*** do wszystkich monitorów.

Algorytm stosujący kolorowanie wiadomości i procesów (3)

Poprawność algorytmu wynika z faktu, że nie istnieje pakiet wysłany przez monitor ***Red***, którego wiadomość przekazana zostałaby procesowi aplikacyjnemu koloru ***White***, gdyż taki pakiet spowodowałby zmianę koloru procesu odbierającego na ***Red*** przed przekazaniem wiadomości.

Zakłada się, że inicjator nie inicjuje współbieżnie wielu detekcji, dlatego wprowadzenie większej liczby kolorów (wartości elementu czasu wektorowego) nie jest konieczne.

Algorytm stosujący kolorowanie wiadomości i procesów (4)

Zakładamy prostszą reprezentację stanu lokalnego – gdzie nie ma potrzeby pamiętania historii komunikacji, co jednak pociąga za sobą konieczność wprowadzenia dodatkowego mechanizmu wyznaczania stanów kanałów komunikacyjnych.

- Wiadomości będące w wyznaczonym obrazie stanu globalnego w kanałach, to wiadomości w pakietach koloru ***White*** odebrane przez monitor koloru ***Red***.
- Za każdym razem, gdy monitor otrzymuje tego typu pakiet, przesyła zawartą w nim wiadomość do inicjatora.

Jedynym problemem w tym przypadku jest wyznaczenie momentu zakończenia konstrukcji obrazu stanu globalnego – tzn. podjęcia przez inicjatora decyzji, że zebrany dotychczas stan jest stanem kompletnym i żaden z monitorów nie prześle w przyszłości pakietu koloru ***White***.

Problem ten, znany jako *problem detekcji zakończenia przetwarzania rozproszonego.*

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *nonFIFO* z zastosowaniem kolorowania wiadomości i procesów

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *nonFIFO* z zastosowaniem kolorowania wiadomości i procesów (typy komunikatów)

TYPY KOMUNIKATÓW

type PACKET **extends** FRAME **is record of**
colour : **enum** { *White*, *Red* }
data : MESSAGE
end record

type PROC_STATE **extends** FRAME **is record of**
procState : PROCESS_STATE
end record

type CHAN_STATE **extends** FRAME **is record of**
chanState : MESSAGE
end record

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *nonFIFO* z zastosowaniem kolorowania wiadomości i procesów (deklaracje)

DEKLARACJE

msgIn : MESSAGE
pcktOut : PACKET
dummyOut : PACKET
procStateOut : PROC_STATE
chanStateOut : CHAN_STATE
$procState_i$: PROCESS_STATE
$procColour_i$: **enum** { *White*, *Red* } := *White*

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami *nonFIFO* z zastosowaniem kolorowania wiadomości i procesów (akcje)

➲ Wstęp
➲ Typy komunikatów
➲ Deklaracje
➲ Procedury
➲ Akcje

AKCJE

1. **when** *e_start*(Q_α, *TakeSnapshot*) **do**
2. $procState_\alpha := S_\alpha$
3. $procColour_i := Red;$
4. *dummyOut.colour* := *Red;*
5. *dummyOut.data* := ∅
6. **send** (Q_α, $\mathcal{Q} \setminus \{Q_\alpha\}$, *dummyOut*)
7. **end when**

8. **when** *e_send*
   (P_i, P_j, *msgOut* : `MESSAGE`) **do**
9. $pcktOut.colour := procColour_i$
10. *pcktOut.data* := *msgOut*
11. **send** (Q_i, Q_j, *pcktOut*)
12. **end when**

13. **when** *e_receive* (Q_j, Q_i, *pcktIn* : `PACKET`) **do**
14. *msgIn* := *pcktIn.data*
15. **if** *:pcktIn.colour = White* ∧ $procColour_i = Red$
16. **then**
17. *chanStateOut.chanState* := *msgIn*
18. **send**(Q_i, Q_α, *chanStateOut*)
19. **end if**
20. **if** *pcktIn.colour = Red* ∧ $procColour_i = White$
21. **then**
22. $procColour_i := Red$
23. $procState_i := S_i$
24. $procStateOut.procState := procState_i$
25. **send** (Q_i, Q_α, , *procStateOut*)
26. **end if**
27. **if** *msgIn* ≠ ∅
28. **then**
29. **deliver** (P_j, P_i, *msgIn*)
30. **end if**
31. **end when**

Konstrukcja obrazu spójnego dla środowiska z kanałami typu FC

Kanały typu FC są mechanizmem komunikacji udostępniającym cztery operacje:

$send^t$ (ang. *two-way-flush send*) $send^b$ (ang. *backward-flush send*)
$send^f$ (ang. *forward-flush send*) $send^o$ (ang. *ordinary send*).

Wiadomości przysyłane z użyciem tych operacji oznaczone są odpowiednio

- M^t – nie może wyprzedzić ani zostać wyprzedzona przez jakąkolwiek inną wiadomość
- M^f – nie wyprzedza innych wiadomości, lecz może zostać wyprzedzona
- M^b – może wyprzedzać inne lecz sama nie jest wyprzedzana
- M^o – może sama wyprzedzać jak i być wyprzedzana przez inne wiadomości

Kanały *FC* umożliwiają dobór operacji komunikacji i tym samym własności przysyłanych wiadomości, stosownie do wymaganego zastosowania.

- Użycie wyłącznie operacji $send^t$ powoduje, że kanał typu *FC* ma własności kanału *FIFO*.
- Użycie tylko operacji $send^o$, to kanał typu *FC* równoważny byłby kanałowi *nonFIFO*.

Tak więc kanały typu *FC* oferują zróżnicowane mechanizmy synchronizacji i tym samym różne stopnie współbieżności komunikacji

Algorytm stosujący znaczniki typu TF

Możliwa jest modyfikacja algorytmu Chandy-Lamporta, polegająca na zastąpieniu znacznika przysłanego kanałem *FIFO*, przez znacznik typu *TF* – `TMARKER`:

- Inicjator Q_α zapamiętuje spontanicznie stan skojarzonego z nim procesu aplikacyjnego P_α i wysyła wiadomość kontrolną (znacznik) typu `TMARKER` do wszystkich incydentnych monitorów.
- Monitory Q_i po odebraniu znacznika z kanału $C_{j,i}$ sprawdzają, czy stan lokalny został już wcześniej zapamiętany.
- Jeśli okazuje się, że jest to pierwszy znacznik `TMARKER` odebrany w danym procesie konstrukcji obrazu spójnego, to stan lokalny jest zapamiętany i przed dopuszczeniem do wysłania kolejnej wiadomości aplikacyjnej, znacznik typu `TMARKER` wysyłany jest poprzez wszystkie kanały wyjściowe. Stan kanału $C_{j,i}$ przyjmuje się przy tym jako zbiór pusty.
- Jeżeli natomiast okazuje się, że stan lokalny był już zapamiętany, to wszystkie wiadomości odebrane między momentem zapamiętania stanu a momentem otrzymania znacznika są włączane do zbioru określającego stan kanału wejściowego $C_{j,i}$.

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników *TF*

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników *TF* (typy komunikatów)

TYPY KOMUNIKATÓW

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

type TMARKER **extends** FRAME

type STATE **extends** FRAME **is record of**
procState : PROCESS_STATE
chanState : **array** [1 .. *n*] **of set of** MESSAGE
end record

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników *TF* (deklaracje)

DEKLARACJE

msgIn : MESSAGE
pcktOut : PACKET
tMarkOut : TMARKER
stateOut : STATE
$recvMark_i$: **array** [1..*n*] **of** BOOLEAN := *False*
$chanState_i$: **array** [1..*n*] **of set of** MESSAGE := ∅
$procState_i$: PROCESS_STATE
$\mathcal{C}_i^{IN}$: **set of** CHANNEL_ID
$\mathcal{C}_i^{OUT}$: **set of** CHANNEL_ID
$involved_i$: BOOLEAN := *False*

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników *TF* (procedury)

PROCEDURY

```
1.  procedure RECORDSTATE( ) do
2.    procState_i := S_i
3.    for all C_{j,i} ∈ C_i^IN do
4.          chanState_i[j] := ∅
5.    end for
6.    involved_i := True
7.    send^t ( Q_i, Q_i^OUT, tMarkOut )
8.  end procedure
```


Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników *TF* (akcje)

AKCJE

```
1.  when e_start( Q_α, TakeSnapshot ) do
2.    RECORDSTATE( )
3.    recvMark_α[α] := True
4.  end when

5.  when e_receive
      (Q_j, Q_i, tMarkIn : TMARKER) do
6.    if ¬ involved_i
7.      then
8.          RECORDSTATE( )
9.    end if
10.   recvMark_i[j] := True
11.   if ∀ C_{j,i} ∈ C_i^IN :: recvMark_i[j]
12.    then
13.         stateOut.procState := procState_i
14.         stateOut.chanState := chanState_i
15.         send^o ( Q_i, Q_α, stateOut )
16.   end if
17. end when

18. when e_send^x
      (P_i, P_j, msgOut : MESSAGE ) do
19.   pcktOut.data := msgOut
20.   send^x ( Q_i, Q_j, pcktOut )
21. end when

22. when e_receive
        (Q_j, Q_i, pcktIn : PACKET ) do
23.   msgIn := pcktIn.data
24.   if involved_i ∧ ¬ recvMark_i[j]
25.    then
26.     chanState_i[j] := chanState_i[j] ∪ {msgIn}
27.    end if
28.    deliver( P_j, P_i, msgIn)
29. end when
```

Algorytm stosujący znaczniki typu BF i FF

Rozwinięcie poprzedniego algorytmu opiera się na spostrzeżeniu, że do utworzenia spójnego obrazu przetwarzania rozproszonego wystarczy by stany lokalne zapamiętane były w wyniku odebrania znaczników typu *BF* (`BMARKER`).

Znacznik ten nie może być wyprzedzony przez żadną wiadomość, i stąd pakiety wysłane po zapamiętaniu stanu i po wysłaniu znacznika typu *BF* dotrą do monitora docelowego z pewnością po zapamiętaniu stanu lokalnego adresata. Nie zaistnieje więc sytuacja prowadząca do niespójnej konfiguracji, w której wiadomość nie uwzględniona w stanie lokalnym przez nadawcę (wysłana po zapamiętaniu stanu) zostanie uwzględniona w stanie lokalnym odbiorcy (odebrana przed zapamiętaniem stanu).

Znacznik typu *FF*, `FMARKER`, wysłany kanałem zaraz po zapamiętaniu stanu lokalnego procesu, nie wyprzedzi pakietów wysłanych przed nim. Jeżeli zatem monitor będzie zapamiętywał wszystkie wiadomości otrzymane danym kanałem od momentu zapamiętania stanu do momentu odebrania znacznika `FMARKER`, to zbiór tych wiadomości obejmie wszystkie wiadomości tworzące stan kanału (wysłane przed znacznikiem a odebrane po zapamiętaniu stanu procesu docelowego) oraz wiadomości wysłane po znaczniku `FMARKER`, które znacznik ten wyprzedziły.

Aby wyróżnić interesujące nas wiadomości tworzące stan kanału $C_{j,i}$ w obrazie spójnym, wystarczy dołączyć do wszystkich wiadomości aplikacyjnych oraz znaczników `FMARKER`, etykietę określającą numer sekwencyjny ostatnio wysłanego komunikatu.

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników typu *BF* i *FF*

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników typu *BF* i *FF* (typy komunikatów)

TYPY KOMUNIKATÓW

type PACKET **extends** FRAME **is record of**
seqNo : INTEGER
data : MESSAGE
end record

type FMARKER **extends** FRAME **is record of**
seqNoPred : INTEGER
end record

type BMARKER **extends** FRAME

type STATE **extends** FRAME **is record of**
procState : PROCESS_STATE
chanState : **array** [1..*n*] **of set of** MESSAGE
end record

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników typu *BF* i *FF* (deklaracje)

DEKLARACJE

msgIn : MESSAGE
pcktOut : PACKET
fMarkOut : FMARKER
bMarkOut : BMARKER
stateOut : STATE
$pcktBuf_i$: **array** [1..*n*] **of set of** PACKET
$recvMark_i$: array [1..*n*] **of** BOOLEAN := *False*
$chanState_i$: array [1..*n*] **of set of** MESSAGE := ∅
$procState_i$: PROCESS_STATE
$\mathcal{C}_i^{IN}$: **set of** CHANNEL_ID
$\mathcal{C}_i^{OUT}$: **set of** CHANNEL_ID
$involved_i$: BOOLEAN := *False*
$seqNo_i$: INTEGER := 0

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników typu *BF* i *FF* (procedury)

PROCEDURY

```
1.  procedure RECORDSTATE( ) do
2.    procState_i := S_i
3.    for all C_{j,i} ∈ C_i^IN do
4.          pcktBuf_i[j] := ∅
5.    end for
6.    involved_i := True
7.    send^b (Q_i, Q_i^OUT, bMarkOut )
8.    fMarkOut.seqNoPred := seqNo_i
9.    send^f ( Q_i, Q_i^OUT, fMarkOut )
10. end procedure
```


Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników typu *BF* i *FF* (akcje I)

AKCJE

```
1.  when e_start( Q_α, TakeSnapshot ) do
2.    RECORDSTATE( )
3.    recvMark_α[ α ] := True
4.  end when

5.  when e_receive
        ( Q_j, Q_i, bMarkIn : BMARKER ) do
6.    if ¬ involved_i
7.      then
8.        RECORDSTATE( )
9.    end if
10. end when

11. when e_receive
        ( Q_j, Q_i, pcktIn : PACKET ) do
12.   msgIn := pcktIn.data
13.   if involved_i ∧ ¬ recvMark_i[j]
14.     then
15.       pcktBuf_i[j] := pcktBuf_i[_j] ∪ {pcktIn}
16.   end if
17.   deliver( P_j, P_i, msgIn)
18. end when
```

Konstrukcja spójnego obrazu stanu globalnego dla środowiska z kanałami typu *FC* z zastosowaniem znaczników typu *BF* i *FF* (akcje II)

AKCJE

19. **when** *e_receive* (Q_j, Q_i, *fMarkIn* : `FMARKER`) **do**
20. $recvMark_i[j] := True$
21. $chanState_i[j] := \varnothing$
22. **for all** $pckt \in pcktBuf_i[j]$ **do**
23. **if** $pckt.seqNo \leqslant fMarkIn.seqNoPred$
24. **then**
25. $chanState_i[j] := chanState_i[j] \cup \{pckt.data\}$
26. **end if**
27. **end for**
28. **if** $\forall\ C_{j,i} \in \mathbf{C}_i^{IN} :: recvMark_i[j]$
29. **then**
30. $stateOut.procState := procState_i$
31. $stateOut.chanState := chanState_i$
32. **send**^{x} (Q_i, Q_α, *stateOut*)
33. **end if**
34. **end when**

35. **when** *e_send*^{x} (P_i, P_j, *msgOut* : `MESSAGE`) **do**
36. $seqNo_i := seqNo_i + 1$
37. $pcktOut.seqNo := seqNo_i$
38. $pcktOut.data := msgOut$
39. **send**^{x} (Q_i, Q_j, *pcktOut*)
40. **end when**