Problem detekcji zakończenia (I)

Plan wykładu

Celem wykładu jest zapoznanie studenta z tematyką detekcji zakończenia. Wykład obejmie przedstawienie przykładów ilustrujących potrzebę problemy detekcji zakończenia w systemach rozproszonych (zakończenie sortowania rozproszonego oraz algorytm Matterna konstrukcji spójnego obrazu stanu globalnego), następnie różne definicje zakończenia (zarówno nieformalną jak i formalną, a także definicja klasyczna zakończenia), pojęcia zakończenia dynamicznego i statycznego i relacje między nimi, zagadnienia związane z detekcją zakończenia w różnych modelach przetwarzania, takich jak model synchroniczny i dyfuzyjny. Student zapozna się również z algorytmami detekcji zakończenia Dijkstry, Feijena, van Gastarena, algorytmem Dijkstry-Scholtena oraz algorytmem Misry dla systemów asynchronicznych.

Problem detekcji zakończenia: przykłady

W celu ilustracji problemu detekcji zakończenia zostaną przedstawione dwa proste przykłady: problem sortowania rozproszonego oraz znany już z wcześniejszych wykładów algorytm Matterna konstrukcji spójnego obrazu stanu globalnego w środowisku z kanałami non-FIFO.

Przykład 1 – sortowanie rozproszone

Przykład 1

Należy posortować zbiór liczb naturalnych. Zbiór ten jest podzielony między procesy a zadaniem każdego procesu jest uporządkowanie przypisanej mu części zbioru liczb naturalnych i wyznaczenie elementu minimalnego, który następnie jest wysyłany do lewych sąsiadów. Po otrzymaniu wiadomości z wartością minimalną, proces wyznacza element maksymalny i wysyła go do prawego sąsiada. Kroki te są powtarzane dopóki zbiór nie zostanie uporządkowany.

Sortowanie rozproszone: definicje

Przyjmijmy, że zbiór $\mathcal{X}$ zostaje wstępnie podzielony na podzbiory $\mathcal{X}_i$ w taki sposób, że:

$$\mathcal{X} = \quad \mathcal{X}_i ,$$

oraz

$$\forall i,j :: (1 \leqslant i, j \leqslant n) \wedge (i \neq j) :: (\mathcal{X}_i \cap \mathcal{X}_j = \varnothing) \wedge (\forall i :: 1 \leqslant i \leqslant n :: \mathcal{X}_i \neq \varnothing)$$

Niech:

- v_i – liczba elementów zbioru $\mathcal{X}_i$
- min_i – minimalny element zbioru $\mathcal{X}_i$
- max_i – maksymalny element zbioru $\mathcal{X}_i$
- P_i – procesy tworzące przetwarzanie rozproszone o topologii łańcucha skojarzone ze zbiorami $\mathcal{X}_i$.

Pary procesów składowych $P_i,\ P_{i+1},\ 1 \leqslant i \leqslant n\text{-}1,$ połączone są kanałami dwukierunkowymi

Sortowanie rozproszone – przykład

Każdy z procesów P_{i+1} ma za zadanie uporządkować (posortować) przypisany mu na wstępie zbiór $\mathcal{X}_{i+1}$ i wyznaczyć element min_{i+1}.

Następnie procesy wysyłają elementy min_{i+1} do swoich lewych sąsiadów i oczekują na odpowiedź zawierającą max_i. Po otrzymaniu wiadomości z wartością min_{i+1}, a przed wysłaniem odpowiedzi, proces P_i wyznacza nowy element max_i.

W ogólności, nowo wyznaczony max_i może być równy otrzymanemu ostatnio min_{i+1}. Następnie każdy z procesów wysyła odpowiedź ze swoim elementem maksymalnym do prawego sąsiada. Po otrzymaniu odpowiedzi, procesy znów sortują zbiory $\mathcal{X}_i$ i jeśli w wyniku tego sortowania wartość min_{i+1} różnić się będzie od poprzednio wysłanego elementu minimalnego, to proces wysyła ten nowy element min_{i+1} do lewego sąsiada. Cel sortowania rozproszonego zostaje osiągnięty, gdy uporządkowany zostanie każdy zbiór $\mathcal{X}_i$, a ponadto dla każdego i: $1 \leqslant i \leqslant n\text{-}1,\ max_i < min_{i+1}$.

Sortowanie rozproszone – przykład (2)

Problem zakończenia

Problem jednak w tym, że każdy proces ma tylko wiedzę lokalną, dotyczącą jego lokalnego zbioru i częściowo zbiorów bezpośrednich sąsiadów. Na tej podstawie procesy nie mogą jednak wnioskować o zakończeniu całego przetwarzania. Potrzebny jest zatem dodatkowy mechanizm pozwalający stwierdzić, że globalne warunki zakończenia sortowania rozproszonego zostały spełnione.

Przykład 2 – obraz stanu globalnego

Przykład 2: Algorytm Matterna konstrukcji spójnego obrazu stanu globalnego w środowisku z kanałami *nonFIFO*.

Ze względu na możliwe zmiany uporządkowania wiadomości w kanałach otrzymanie wiadomości koloru *Red* nie przesądzało o tym, że w kanałach nie ma już wcześniej wysłanych wiadomości koloru *White*. Ponieważ jednak zbiór otrzymanych przez proces koloru *Red* wiadomości *White* określa stan kanału w wyznaczanym obrazie stanu globalnego, niezbędne jest niezależne od procesu konstrukcji obrazu stanu globalnego sprawdzenie, czy w kanałach nie ma jeszcze wiadomości koloru *White*. Dopiero bowiem wówczas, gdy stwierdzimy, że wszystkie wiadomości koloru *White* zostały odebrane, wyznaczone stany kanałów odpowiadają spójnemu obrazowi stanu globalnego.

Definicja nieformalna zakończenia

Nieformalnie problem detekcji zakończenia przetwarzania rozproszonego polega na sprawdzeniu, czy wszystkie procesy przetwarzania są w stanie pasywnym oraz czy żadna wiadomość będąca w kanale (transmitowana lub dostępna) nie uaktywni któregokolwiek z tych procesów. Przez proces aktywny rozumiemy tutaj wykonujący kroki algorytmu; w przeciwnym wypadku uznajemy go za pasywny. Dokładne definicje zostały przedstawione na wcześniejszych wykładach.

Przez zakończenie obliczeń rozproszonych mamy tutaj na myśli osiągnięcie pewnej końcowej konfiguracji, w której nie są już możliwe dalsze kroki algorytmu.

Definicja formalna: Oznaczenia (1)

W sformułowaniu formalnym tego problemu, podobnie jak w sformułowaniu problemu zakleszczenia, wykorzystamy 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*, gdy dla P_i jest dostępna wiadomość wysłana przez P_j

Definicja formalna: Oznaczenia (2)

$in\text{-}transit_i$ – 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*, gdy wiadomość wysłana przez P_j do P_i należy do $L^T_{j,i}$, a więc jest transmitowana i nie jest jeszcze dostępna

Oprócz tego przyjmiemy oznaczenie dwóch zbiorów procesów $\mathcal{AV}_i$ oraz $\mathcal{IT}_i$, zdefiniowanych w następujący sposób:

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

Zakończenie dynamiczne

Przetwarzanie rozproszone Π jest w danej chwili w stanie *zakończenia dynamicznego*, jeżeli żaden proces składowy przetwarzania rozproszonego nie będzie już nigdy uaktywniony. Stan ten będzie utrzymywany pomimo, że pewne wiadomości są wciąż transmitowane, a pewne wiadomości są już dostępne.

Zakończenie dynamiczne: definicja formalna

Przedstawioną definicję można bardziej formalnie zapisać z wykorzystaniem przedstawionych wcześniej oznaczeń w następujący sposób: przetwarzanie rozproszone Π jest w danej chwili w stanie *zakończenia dynamicznego*, gdy spełniony jest predykat:

$$\boldsymbol{Dterm}(\mathcal{P}) \equiv \forall P_i :: P_i \in \mathcal{P} :: (passive_i \wedge \neg\, activate_i(\mathcal{AV}_i \cup \mathcal{IT}_i))$$

Predykat ten oznacza, że żaden proces składowy przetwarzania rozproszonego nie będzie już nigdy uaktywniony. Stan ten będzie utrzymywany pomimo, że pewne wiadomości są wciąż transmitowane ($\mathcal{IT}_i \neq \varnothing$), a pewne wiadomości są już dostępne ($\mathcal{AV}_i \neq \varnothing$).

Takie sformułowanie definiującego stan zakończenia jest interesujące z praktycznego punktu widzenia, gdyż pozwala stwierdzić zakończenie przetwarzania nawet przed dotarciem wszystkich wiadomości do procesów (węzłów) docelowych. Definicja zakończenia dynamicznego uwzględnia rzeczywistą aktywność procesów wyrażoną przez zmienną $passive_i$ oraz potencjalną aktywność wyrażoną przez predykat $activate_i(\mathcal{AV}_i \cup \mathcal{IT}_i)$.

Predykat $Dterm(\mathcal{P})$ jest predykatem stabilnym.

Zakończenie statyczne

Przetwarzanie rozproszone Π jest w danej chwili w stanie *zakończenia statycznego*, jeżeli wszystkie procesy są pasywne, wszystkie wiadomości znajdujące się w kanałach są dostępne i dla żadnego procesu nie jest spełniony warunek uaktywnienia.

Zakończenie statyczne: definicja formalna

Przetwarzanie rozproszone Π jest w danej chwili w stanie *zakończenia statycznego*, gdy spełniony jest predykat:

$$\boldsymbol{Sterm}(\mathcal{P}) \equiv \forall P_i :: P_i \in \mathcal{P} :: (passive_i \wedge (\mathcal{IT}_i = \varnothing) \wedge \neg activate_i(\mathcal{AV}_i))$$

Oznaczenia w powyższym wzorze wprost odpowiadają wcześniej podanej definicji: wszystkie procesy są pasywne ($passive_i = True$), wszystkie wiadomości znajdujące się w kanałach są dostępne ($\mathcal{IT}_i = \varnothing$) i dla żadnego procesu nie jest spełniony warunek uaktywnienia ($activate_i(\mathcal{AV}_i) = False$). Definicja ta uwzględnia zatem zarówno stany procesów jak i stany kanałów.

Porównując z $Dterm(\mathcal{P})$, predykat $Sterm(\mathcal{P})$ odpowiada detekcji nieco późniejszej, gdyż dodatkowo wymaga się, by wiadomości nie były już transmitowane ($\mathcal{IT}_i = \varnothing$).

Zakończenie dynamiczne ↔ statyczne

Niech $\vartheta \rightsquigarrow \vartheta'$ oznacza, że zajście predykatu ϑ prowadzi w skończonym choć nieprzewidzianym czasie do zajścia predykatu ϑ'.

Twierdzenie 9.1

$$Dterm(\mathcal{P}) \rightsquigarrow Sterm(\mathcal{P})$$

Dowód

W chwili τ, gdy $Dterm(\mathcal{P})$ = $True$, wszystkie procesy P_i są pasywne, zachodzi $\neg activate_i(\mathcal{AV}_i \cup \mathcal{IT}_i)[\tau]$, a część wiadomości może znajdować się w kanałach. Jednakże, wszystkie wiadomości transmitowane są brane pod uwagę, a ich dotarcie do węzłów docelowych i w konsekwencji ich dostępność, jest uwzględniona w wartości predykatu $\neg activate_i(\mathcal{AV}_i \cup \mathcal{IT}_i)$. Wobec niezawodności kanałów, wiadomości transmitowane osiągną węzły docelowe po skończonym choć nieprzewidywalnym czasie, w pewnym momencie $\tau' > \tau$. Wówczas $\mathcal{IT}_i[\tau'] = \varnothing$. Wobec stałej pasywności wszystkich procesów od chwili τ, w każdej chwili $\tau'' \geqslant \tau'$, $\mathcal{AV}_i[\tau''] = \mathcal{AV}_i[\tau] \cup \mathcal{IT}_i[\tau]$ oraz $\mathcal{IT}_i[\tau''] = \varnothing$.
Stąd otrzymujemy:

$$\neg activate_i(\mathcal{AV}_i[\tau'']) = True, \quad \mathcal{IT}_i[\tau''] = \varnothing$$

Predykat $Sterm(\mathcal{P})$ przyjmuje zatem wartość $True$. □

Udowodniliśmy więc, że opóźnienie między momentem zajścia predykatu $Dterm(\mathcal{P})$ a momentem zajścia predykatu $Sterm(\mathcal{P})$ jest skończone lecz nieprzewidywalne. Wybór między jedną a drugą definicją zakończenia zależy oczywiście od użytkownika. Łatwo przewidzieć, że detekcja stanu opisanego predykatem $Dterm(\mathcal{P})$ będzie trudniejsza, a więc w ogólności bardziej kosztowna. Z drugiej jednak strony, zajście $Dterm(\mathcal{P})$ może pozwolić, na przykład, na uznanie wyników przetwarzania za ostateczne (w konsekwencji możliwe do dalszego wykorzystania) nawet, gdy pewne wiadomości są jeszcze transmitowane.

Klasyczna definicja zakończenia

W klasycznej definicji zakończenia przyjmowano, że przetwarzanie rozproszone jest w stanie zakończenia, jeżeli w danej chwili wszystkie procesy są pasywne i wszystkie kanały są puste, a więc gdy zachodzi następujący predykat:

$$Cterm(\mathcal{P}) \equiv \forall P_i :: P_i \in \mathcal{P} :: (passive_i \wedge (\mathcal{IT}_i = \varnothing) \wedge (\mathcal{AV}_i = \varnothing).$$

Klasyczna definicja zakończenia a zakończenie statyczne

Rozważmy teraz relację między $Sterm(\mathcal{P})$ a $Cterm(\mathcal{P})$.

Twierdzenie 9.2

Jeżeli procesy są uaktywnione w sposób natychmiastowy w chwili zajścia predykatu $activate_i(\mathcal{AV}_i)$ przez każdą dostępną wiadomość, to przetwarzanie rozproszone obejmujące zbiór procesów $\mathcal{P}$ jest statycznie zakończone wtedy i tylko wtedy, gdy zachodzi predykat $Cterm(\mathcal{P})$.

Dowód
Zgodnie z założeniem, procesy stają się aktywne natychmiast w chwili zajścia predykatu $activate_i(\mathcal{AV}_i)$. W konsekwencji $passive_i$ jest równe $True$ tylko wówczas, gdy $\neg activate_i(\mathcal{AV}_i)$. Stąd, w wypadku procesów uaktywnianych każdą wiadomością:

$$passive_i \wedge \neg activate_i(\mathcal{AV}_i) \equiv passive_i \wedge (\mathcal{AV}_i = \varnothing).$$

W konsekwencji:

$$Sterm(\mathcal{P}) \equiv \forall P_i :: P_i \in \mathcal{P} :: (passive_i \wedge (\mathcal{IT}_i = \varnothing) \wedge \neg activate_i(\mathcal{AV}_i))$$
$$\equiv \forall P_i :: P_i \in \mathcal{P} :: (passive_i \wedge (\mathcal{IT}_i = \varnothing) \wedge (\mathcal{AV}_i = \varnothing) \equiv Cterm(\mathcal{P}) \ . \ \square$$

Warto zauważyć silne związki między pojęciami zakończenia i zakleszczenia. W istocie, zakończenie jest szczególnym wypadkiem zakleszczenia, w którym zakleszczone są wszystkie procesy przetwarzania rozproszonego.

Problem detekcji zakończenia

Problem detekcji zakończenia przetwarzania rozproszonego obejmującego zbiór procesów, sprowadza się do sprawdzenia czy przetwarzanie osiągnęło określony stan zakończenia, a więc – czy zachodzi odpowiedni predykat: $Dterm(\mathcal{P})$, $Sterm(\mathcal{P})$ lub $Cterm(\mathcal{P})$.

Można dowieść, że jeżeli w czasie przetwarzania aplikacyjnego wymienianych jest *m* wiadomości, to niemożliwa jest konstrukcja algorytmu detekcji zakończenia o złożoności komunikacyjnej mniejszej niż *m*.

Model przetwarzania synchronicznego

Na początku rozważony zostanie problem detekcji zakończenia w modelu *przetwarzania synchronicznego*. W modelu tym przyjmuje się, że transmisje są natychmiastowe. Stąd kanały mogą być uznane za puste przez cały czas i problem zakończenia sprowadza się do sprawdzenia czy wszystkie procesy są jednocześnie pasywne.

Stan zakończenia opisuje więc następujący predykat:

$$Iterm(\mathcal{P}) \equiv \forall P_i :: P_i \in \mathcal{P} :: passive_i$$

Detekcja zakończenia dla synchronicznego modelu przetwarzania

Zostanie obecnie omówiony algorytm autorstwa Dijkstry, Feijen oraz van Gasterena detekcji zakończenia dla modelu przetwarzania synchronicznego. Wykorzystuje on wykorzystuje koncepcję ciągu cykli detekcyjnych i zakłada, że wszystkie monitory procesów aplikacyjnych połączone są w logiczny pierścień i obserwują stany procesów aplikacyjnych.

Monitorom (procesom) przypisany jest kolor *White* albo *Black*.

Monitory przesyłają wzdłuż pierścienia wiadomość kontrolną – znacznik typu TOKEN, który również może mieć kolor *White* albo *Black.*

Początkowo monitory mają kolor *White*, a zmieniają kolor na *Black*, gdy odpowiadający im proces aplikacyjny wyśle wiadomości do procesu o indeksie większym.

Monitor inicjujący detekcję zakończenia $Q_\alpha = Q_1$ wysyła znacznik koloru *White* do swego następnika w pierścieniu Q_n jeżeli obserwowany przez niego proces aplikacyjny jest pasywny.

Każdy kolejny monitor Q_i odbierający znacznik czeka aż obserwowany przez niego proces stanie się pasywny i wówczas wysyła znacznik o kolorze zgodnym z kolorem monitora.

Po wysłaniu znacznika monitorowi przypisywany jest kolor *White*.

Algorytm kończy się, gdy znacznik koloru *White* dotrze do inicjatora. Dla uproszczenia prezentacji, w algorytmie wykorzystano funkcje:

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

Przykład detekcji zakończenia runda zakończona niepowodzeniem

Na przedstawionym slajdzie inicjator (proces P_1) przesyła znacznik koloru *White* do swojego następnika, P_6. Następnie znacznik ten jest przesyłany w pierścieniu do kolejnego procesu, P_5. Oba te procesy posiadają przypisany kolor *White*, a więc był pasywny w bieżącej rundzie detekcji. Znacznik więc nie zmienia koloru. Tymczasem proces P_4 wysyła wiadomość aplikacyjną do procesu P_6, co powoduje zmianę jego koloru na *Black*. Kiedy więc znacznik dociera do P_4 zmienia kolor również na *Black*. Po przejściu całego pierścienia znacznik powraca do inicjatora. Ponieważ znacznik przyjął kolor *Black,* detekcja zakończenia nie została uwieńczona powodzeniem.

Przykład detekcji zakończenia runda zakończona sukcesem

Monitor zmienia więc kolor na *White* i ponownie wysyła znacznik o kolorze *White* do swojego następnika w pierścieniu. Sytuacja jest podobna jak poprzednio – znacznik jest przesyłany między procesami od inicjatora, P_1, do jego następnika P_6, dalej do P_5. W tym momencie proces P_4 wysyła wiadomość do procesu P_2 – nie powoduje to jednak zmiany koloru przypisanego do procesu, gdyż indeks adresata wiadomości jest mniejszy niż indeks nadawcy. Kiedy więc znacznik dociera do procesu P_4, jego kolor się nie zmienia i dalej jest przesyłany również z przypisanym kolorem *White*. Ostatecznie znacznik dociera do inicjatora detekcji, a ponieważ jego kolor pozostaje *White*, inicjator może stwierdzić wykrycie zakończenia.

Detekcja zakończenia: Dijkstra, Feijen, van Gasteren (1)

Algorytm wykorzystuje pakiety dwóch typów: pierwszy, typ PACKET, opakowują wiadomości aplikacyjne. Drugi typ, TOKEN, to znacznik posiadający pole *colour* oznaczające kolor znacznika.

Detekcja zakończenia: Dijkstra, Feijen, van Gasteren (2)

Wiadomość *msgIn* oznacza wiadomość aplikacyjną wysyłaną przez proces P_i, która jest opakowywane w postaci *pcktOut*. Wiadomość *tokenOut* jest znacznikiem typu TOKEN. Zmienna $tokenPresent_i$ określa obecność znacznika w procesie P_i. Zmienna $procColour_i$ służy do określenia koloru monitora (procesu). Zmienna $terminationDetected_i$ zostaje ustawiona na *True* jeżeli wykryte zostało zakończenie. Początkowo monitory mają kolor *White.*

Detekcja zakończenia: Dijkstra, Feijen, van Gasteren (3)

Procedura INITPROC wywoływana jest przez monitor Q_α ($\alpha = 1$) podczas inicjacji kolejnej rundy. Powoduje ona przesłanie następnikowi w pierścieniu znacznika o kolorze *White* oraz przypisanie wartości *White* zmiennej $procColour_\alpha$.

Detekcja zakończenia: Dijkstra, Feijen, van Gasteren (4)

Monitor inicjujący detekcję zakończenia $Q_\alpha = Q_1$ wysyła znacznik koloru *White* (wywołując procedurę INITPROC) do swego następnika w pierścieniu Q_n jeżeli obserwowany przez niego proces aplikacyjny jest pasywny.

Monitor Q_i zmienia kolor na *Black*, jeżeli obserwowany przez niego proces wysyła wiadomość aplikacyjną do procesu o wyższym indeksie.

Detekcja zakończenia: Dijkstra, Feijen, van Gasteren (5)

Algorytm nie przewiduje żadnych specjalnych akcji dla zdarzenia odbioru wiadomości aplikacyjnej. Jest ona po prostu dostarczana do skojarzonego z monitorem procesu aplikacyjnego.

Detekcja zakończenia: Dijkstra, Feijen, van Gasteren (6)

Odebranie znacznika (tokena) przez proces Q_i wysłanego przez jego następnika w pierścieniu $Q_{succ(i)}$, powoduje ustawienie zmiennej $tokenPresent_i$ na wartość *True.* Znacznik jest zatrzymywany w monitorze Q_i do czasu aż obserwowany przez niego proces aplikacyjny P_i stanie się pasywny.

Jeżeli $Q_i = Q_\alpha$ i kolor zarówno obserwowanego procesu jak i znacznika jest *White*, inicjator kończy algorytm decydując o wykryciu zakończenia. Jeżeli $Q_i = Q_\alpha$ ale albo kolor procesu, albo kolor znacznika równy jest *Black*, inicjator rozpoczyna kolejną rundę algorytmu.

Detekcja zakończenia: Dijkstra, Feijen, van Gasteren (7)

Monitory $Q_i \neq Q_\alpha$ przesyła dalej token o kolorze takim samym jak kolor obserwowanego procesu aplikacyjnego, zmienia następnie kolor procesu na *White* oraz zmienia wartość $tokenPresent_i$ na wartość *False*.

Model przetwarzania dyfuzyjnego

Przetwarzanie dyfuzyjne (ang. *diffusing computation*) jest specyficznym przetwarzaniem rozproszonym, w którym wyróżnia się:

- *inicjatora* (środowisko) – może w dowolnej chwili rozpocząć przetwarzanie dyfuzyjne wysyłając wiadomość aplikacyjną do jednego lub wielu procesów kooperujących (zakłada się, że inicjacja taka zachodzi tylko raz),
- hierarchię kooperujących procesów.

Każdy proces po uzyskaniu pierwszej wiadomości aplikacyjnej, nadawcę tej wiadomości traktuje jako proces *angażujący* (ang. *engager*) i realizuje dalsze przetwarzanie wysyłając wiadomości do innych procesów, w tym ewentualnie do inicjatora.

Założenia dodatkowe

Dla ułatwienia opisu przedstawianych algorytmów, przyjmiemy pewne dodatkowe upraszczające założenia:

- Proces aktywny staje się procesem pasywnym tylko w wyniku pewnego zdarzenia wewnętrznego
- Proces zawsze staje się aktywny po otrzymaniu wiadomości
- Proces pasywny może stać się aktywny *tylko* w wyniku otrzymaniu wiadomości

Innymi słowy, proces aplikacyjny może w dowolnej chwili stać się pasywny i oczekiwać na uaktywniającą go wiadomość aplikacyjną od dowolnego innego procesu. Warunek uaktywnienia procesu definiuje zatem model żądań *OR* i zbiór warunkujący $\mathcal{D}_i = \mathcal{P} \setminus \{ P_i \}$.

Koncepcja algorytmu detekcji zakończenia (Dijkstra-Scholten ’80)

Monitory procesów aplikacyjnych przesyłają wiadomości kontrolne typu SIGNAL jako pewnego rodzaju odpowiedzi na wiadomości aplikacyjne.

Monitor pasywnego inicjatora może stwierdzić zakończenie całego przetwarzania aplikacyjnego, gdy odbierze wiadomości SIGNAL od wszystkich monitorów związanych z procesami uaktywnionymi przez inicjatora.

Graf przetwarzania dyfuzyjnego

Inicjatorem przetwarzania dyfuzyjnego w przestawionym przykładzie jest P_1. Staje się on procesem angażującym dla procesów P_2, P_3 oraz P_4. Każdy z nich z kolei sam angażuje kolejne procesy. Procesy tworzą więc drzewo, a wiadomości są przesyłane od inicjatora (korzenia drzewa) i niżej.

Graf przetwarzania dyfuzyjnego (2)

Procesy odsyłają specjalne wiadomości, poczynając od liści drzewa. Każdy proces – węzeł drzewa przesyła wiadomość do swojego procesu angażującego, jeśli zebrał już wiadomości od wszystkich

procesów potomnych. W momencie, w którym inicjator zbierze wszystkie wiadomości od swoich procesów potomnych, może uznać, że zostało wykryte zakończenie przetwarzania.

etekcji zakończenia dla dyfuzyjnego modelu przetwarzania (1)

Algorytm wykorzystuje pakiety dwóch typów: pierwszy, typ PACKET, opakowują wiadomości aplikacyjne. Drugi typ, SIGNAL to specjalna wiadomość kontrolna przesyłana między monitorami.

etekcji zakończenia dla dyfuzyjnego modelu przetwarzania (2)

Wiadomość *msgIn* oznacza wiadomość aplikacyjną wysyłaną przez proces P_i, która jest opakowywane w postaci *pcktOut*. Zmienna $engager_i$ zawiera identyfikator monitora, którego proces uaktywnił P_i. Z kolei zbiór $notEngager_i$ oznacza zbiór monitorów różnych od monitora angażującego, od których odebrano pakiet. Zmienne $recvNo_i$ oraz $sentNo_i$ oznaczają, odpowiednio, liczbę wiadomości odebranych bądź wysłanych przez P_i a nie potwierdzonych jeszcze przez Q_i wiadomością typu SIGNAL. Zmienna $terminationDetected_i$ zostaje ustawiona na *True* jeżeli wykryte zostało zakończenie.

etekcji zakończenia dla dyfuzyjnego modelu przetwarzania (3)

Przetwarzanie dyfuzyjne jest inicjowane przez proces P_α który wysyła wiadomości do swoich bezpośrednich potomków w hierarchii należących do zbioru $\mathcal{P}_\alpha^{\mathrm{R}}$. Zmienna $sentNo_\alpha$ zawiera liczbę tych wysłanych wiadomości.

etekcji zakończenia dla dyfuzyjnego modelu przetwarzania (4)

Zdarzenie wysłania wiadomości typu SIGNAL może zajść dla monitora Q_i dla tylko wtedy, gdy ($recvNo_i > 1$) $\vee$ ($recvNo_i = 1 \wedge sentNo_i = 1 \wedge passive_i$). Monitor Q_i wysyłając wiadomość typu SIGNAL sprawdza, czy odebrał już wiadomości od wszystkich procesów poza angażującym ($recvNo_i = 1$), nie wysłał żadnej nie potwierdzonej wiadomości ($sentNo_i = 0$) a skojarzony zeń proces jest pasywny. W takim wypadku wysyła procesowi angażującemu sygnał. W przeciwnym wypadku wysyła wiadomość kontrolną typu SIGNAL do wszystkich monitorów procesów od których otrzymał jakąś wiadomość i jeszcze jej nie potwierdził do tej pory wiadomością SIGNAL, poza monitorem procesu angażującego, i usuwa te procesy z zbioru $notEngager_i$. Wreszcie dekrementuje zmienną $recvNo_i$.

etekcji zakończenia dla dyfuzyjnego modelu przetwarzania (5)

Monitor Q_i odbierając sygnał dekrementuje zmienną $sentNo_i$ (gdyż zmienna ta oznacza ilość nie potwierdzonych wysłanych wiadomości). Jeżeli

$Q_i = Q_\alpha \wedge sentNo_i = 0$, to w momencie w którym skojarzony proces aplikacyjny P_α stanie się pasywny, monitor decyduje o wykryciu zakończenia.

etekcji zakończenia dla dyfuzyjnego modelu przetwarzania (6)

Zdarzenie wysłania wiadomości aplikacyjnej może zajść dla monitora Q_i dla $i \neq \alpha$ tylko wtedy, gdy $recvNo_i > 0 \wedge \neg\, passive_i$. Wysyłając dowolną wiadomość aplikacyjną monitor inkrementuje licznik nie potwierdzonych wiadomości $sentNo_i$.

etekcji zakończenia dla dyfuzyjnego modelu przetwarzania (7)

W przypadku otrzymania wiadomości, jeżeli jest to pierwsza otrzymana wiadomość ($recvNo_i = 0$) to monitor Q_i uznaje nadawcę za proces angażujący, a w przeciwnym wypadku dodaje go do zbioru $notEngager_i$. Następnie inkrementuje licznik $recvNo_i$ i dostarcza wiadomość procesowi aplikacyjnemu.

Twierdzenie 9.3

Twierdzenie 9.3

Jeżeli przetwarzanie dyfuzyjne uległo zakończeniu, fakt ten ulega wykryciu przez algorytm Dijkstry-Scholtena.

Dowód

Jeżeli przetwarzanie dyfuzyjne ulega zakończeniu, oznacza to, że nie ma żadnego aktywnego procesu ani nie ma żadnych wiadomości ani sygnałów w kanałach komunikacyjnych. Oznacza to

także, że żaden proces nie może wysyłać żadnych wiadomości ani potwierdzeń. W związku z tym, można wywnioskować, że jeżeli przetwarzanie ulegnie zakończeniu, to w każdym procesie P_i (za wyjątkiem procesu-inicjatora) zachodzi: $recvNo_i \geq 0 \wedge sentNo_i \geq 0$ oraz $recvNo_i > 1 \vee (recvNo_i = 1 \wedge sentNo_i = 0)$. Spostrzeżenie to można uprościć do postaci $recvNo_i = 0 \vee (recvNo_i = 1 \wedge sentNo_i > 0)$.

Z kolei dla procesu inicjującego oczywiste jest, że zachodzi $recvNo_i = 0 \wedge sentNo_i \geq 0)$.

Dalej można wywnioskować, że w stanie zakończenia przetwarzania $\forall P_i$: $recvNo_i \leq sentNo_i$. Ponieważ żadne wiadomości ani sygnały nie są przesyłane, więc suma wszystkich $recvNo_i$ wszystkich procesów musi być równa sumie wszystkich $sentNo_i$. Z tych dwóch wniosków oraz z spostrzeżenia tyczącego wartości zmiennych u inicjatora, wynika, że w wyniku zakończenia przetwarzania w procesie inicjatora zachodzi $recvNo_i - sentNo_i = 0$, a ponieważ proces ten jest pasywny, więc zostaną spełnione warunki wykrycia zakończenia.

Detekcja zakończenia dla systemów asynchronicznych (Misra '83)

Dla systemów asynchronicznych można wykorzystać w celu detekcji zakończenia algorytm (Misra'83), który nie czyni żadnych założeń co do topologii przetwarzania i czasu przesyłania. Algorytm ten wymaga natomiast, by żadne nie wiadomości nie były gubione podczas transmisji (niezawodność komunikacji) oraz by kanały komunikacyjne zachowywały porządek wysyłania wiadomości (kanały FIFO).

Algorytm ten, podobnie jak przedstawiony wcześniej algorytm dla modelu synchronicznego (Dijkstra, Feijen, van Gasteren) używa znacznika, oraz kolorowanie procesów. Dla ułatwienia analizy, można przyjąć, że wszystkie topologia przetwarzania to graf w pełni połączony, jednakże algorytm łatwo można dostosować do dowolnej topologii, co zostanie pokazane dalej.

Detekcja zakończenia dla systemów asynchronicznych: algorytm (1)

Algorytm wykorzystuje pakiety dwóch typów: pierwszy, typ PACKET, opakowują wiadomości aplikacyjne. Drugi typ, TOKEN to specjalna wiadomość kontrolna, znacznik, przesyłana między monitorami, zawierające pole *nb* oznaczające liczbę procesów odwiedzonych przez znacznik, które były pasywne.

etekcji zakończenia dla systemów asynchronicznych (2)

Wiadomość *msgIn* oznacza wiadomość aplikacyjną wysyłaną przez proces P_i, która jest opakowywane w postaci *pcktOut*. Wiadomość *tokenOut* jest znacznikiem typu TOKEN. Zmienna $tokenPresent_i$ określa obecność znacznika w procesie P_i. Zmienna $colour_i$ służy do określenia koloru monitora (procesu). Zmienna $terminationDetected_i$ zostaje ustawiona na *True* jeżeli wykryte zostało zakończenie. Początkowo monitory mają kolor *White.*

Dwie zmienne związane są z topologią przetwarzania. Zmienna $succ_i$ zawiera następnika procesu P_i w cyklu obejmującym wszystkie kanały komunikacyjne. Związane jest to z wymaganiem, by znacznik typu TOKEN odwiedził wszystkie kanały łączące procesy. $\mathcal{C}$ oznacza zbiór wszystkich kanałów tworzących cykl obejmujący wszystkie kanały komunikacyjne.

etekcji zakończenia dla systemów asynchronicznych (3)

Zaczynając detekcję, proces wysyła znacznik do swojego następnika w cyklu, przypisując polu *nb* tego znacznika wartość 0.

etekcji zakończenia dla systemów asynchronicznych (4)

Otrzymanie wiadomości aplikacyjnej przez monitor Q_i powoduje, poza uaktywnieniem procesu P_i i dostarczeniem wiadomości, zmianę koloru procesu na *Black.*

etekcji zakończenia dla systemów asynchronicznych (5)

Po otrzymaniu znacznika, monitor Q_i przesyła go dalej w momencie, gdy obserwowany przez niego proces staje się pasywny. Jeżeli kolor procesu równa się *Black*, pole *nb* znacznika jest zerowane. W przeciwnym razie jest ono inkrementowane. Kolor procesu następnie ustawiany jest na *White* oraz

monitor przesyła znacznik do swojego następnika w cyklu. Oznacza to, że każdy proces musi pozostać pasywny pomiędzy wszystkimi odwiedzinami znacznika, a znacznik musi co najmniej dwa razy przejść cykl.

Detekcja zakończenia dla systemów asynchronicznych: algorytm (6)

Inicjator P_α przetwarzania, po otrzymaniu znacznika, jeżeli kolor procesu P_α posiada wartość *White* oraz jeżeli wartość zapisana w polu nb otrzymanego znacznika równa jest liczbie kanałów komunikacyjnych (liczbie krawędzi grafu topologii przetwarzania), decyduje o wykryciu zakończenia. W przeciwnym wypadku przesyła znacznik dalej, być może zerując wartość pola nb w podobny sposób, jak każdy inny proces.

Warte podkreślenia jest, że w ogólności każdy monitor, a nie tylko inicjator może dokonać decyzji o wykryciu detekcji zakończenia obserwując pole nb oraz kolor obserwowanego przez siebie procesu.

Cechy algorytmu Misra'83

Aby dostosować przedstawiony algorytm do dowolnej topologii, wystarczy ustawić odpowiednio wartości zmiennej $succ_i$ – wystarczy podzielić graf topologii przetwarzania na zbiór cykli o maksymalnej długości i odwiedzać je po kolei. Wiele monitorów naraz może dokonywać detekcji zakończenia – w takim wypadku wystarczy dodać do znacznika pole wskazujące na inicjatora detekcji. Dużą wadą algorytmu jest wymaganie, by cykle obejmujące kanały komunikacyjne były znane z góry.

Należy tutaj podkreślić, że w algorytmie dowolny monitor może decydować o wykryciu zakończenia i w gruncie rzeczy żaden z nich nie jest wyróżniony.