Nanozombie - algorytm

Marcin Pastwa, Piotr Tomaszewski

QUEUE_PONY

Lista, na której proces przechowuje identyfikatory procesów, które prosiły go o dostęp do sekcji krytycznej (stroju kucyka), gdy ten proces się w niej znajdował (algorytm Ricarta-Agrawali).

Procesy z tej listy są informowane, gdy strój jest zwracany (otrzymują wiadomość ACK_PONY).

QUEUE_SUBMAR{id_łodzi}

Każdy proces posiada kolejkę, w której znajdują się procesy ubiegające się o dostęp do danej łodzi podwodnej.

Na jej podstawie można wyznaczyć, które procesy mogą zająć miejsce na danej łodzi podwodnej (znajdują się w strefie krytycznej).

Kolejka jest bardzo podobna do tej z alg. Lamporta, z tą różnicą, że w sekcji krytycznej na raz może znajdować się >1 proces. Dokładny opis znajduje się w części algorytmicznej.

LIST_SUBMAR

Proces przechowuje informację o każdej z łodzi o tym czy, jego zdaniem, znajduje się ona obecnie w porcie, czy też nie. TODO: Również czy nie pełna

Proces preferuje wybór łodzi stojących w porcie. Jest to próba minimalizacji czasu, który dana łódź spędzi nieużywana.

Zawartość tej listy może być nieaktualna i nie spowoduje to błędu. W najgorszym wypadku, proces zacznie ubiegać się o łódź, która jest w podróży i będzie musiał się wycofać (ALBO CZEKAĆ TODO) i wybrać inną.

DICT_TOURISTS_SIZES

Tablica poglądowa lub w ogólności słownik, w którym kluczem jest identyfikator procesu, natomiast wartością jest rozmiar turysty (ile miejsc na łodzi zajmuje). Wartości te są stałe, więc na potrzeby algorytmu przyjmujemy, że są już każdemu procesowi znane.

Jeśli jednak założyć, że wartości te nie są znane z góry, procesy musiałyby wymienić się nimi między sobą przed rozpoczęciem pętli głównej.

DICT_SUBMAR_CAPACITY

Tablica poglądowa lub w ogólności słownik, w którym kluczem jest identyfikator łodzi podwodnej, natomiast wartością jest jej maksymalna pojemność. Wartości te są stałe, więc na potrzeby algorytmu przyjmujemy, że są już każdemu procesowi znane.

Znacznik Lamporta (Timestamp)

W celu określenia relacji uprzedniości zdarzeń w algorytmie zastosowany jest zegar logiczny Lamporta.

Do wysyłanych przez proces wiadomości dołączane są znaczniki czasowe.

Uznaliśmy, że uwzględnienie aktualizacji zegarów i przesyłu znaczników jedynie zmniejszyłoby czytelność algorytmu, dlatego zostało w dalszym opisie ograniczone do minimum.

Rozważaliśmy również dołączanie do odpowiedzi na zapytania znacznika tego zapytania. Pozwoliłoby to na weryfikację czy zgoda nie dotyczy jakiegoś przedawnionego zapytania.

REQ_PONY

1 Proces wysyła tę wiadomość, gdy chce uzyskać dostęp do sekcji
2 krytycznej – dostęp do stroju kucyka.

ACK_PONY

Stanowi potwierdzenie otrzymania prośby o dostęp do stroju kucyka.

Wysyłana w odpowiedzi na zapytanie REQ_PONY, gdy odpowiadający proces zgadza się aby pytający uzyskał dostęp do zasobu.

RESTING

1 Jest to stan początkowy.

2 Symuluje odpoczynek turysty między zakończeniem jednej
3 wycieczki, a rozpoczęciem kolejnej.

4 Do kolejnego stanu – WAIT_PONY – proces przechodzi w
5 pewnym, nieokreślonym momencie. Przyjmujemy, że ten czas jest
6 pewną losową wartością >= 0.

RESTING - odpowiedzi

1. Po otrzymaniu REQ_PONY odpowiada ACK_PONY.

2. Po otrzymaniu REQ_SUBMAR proces dodaje nadawcę do kolejki
3. QUEUE_SUBMAR{id_łodzi} i odpowiada ACK_SUBMAR.

4. ACK_PONY – ignoruje.

5. ACK_SUBMAR – ignoruje.

WAIT_PONY

1. Proces w tym stanie ubiega się o możliwość zabrania stroju kucyka,
2. czyli na dostęp do sekcji krytycznej.

3. Do kolejnego stanu – WAIT_SUBMAR – proces przechodzi po
4. zabraniu stroju kucyka.

WAIT_PONY - odpowiedzi

Na REQ_PONY odpowiada:

Jeśli otrzymane zapytanie ma niższy priorytet od wysłanego przez ten proces nic nie odpowiada, tylko wstawia id nadawcy do swojej listy QUEUE_PONY.

W przeciwnym razie, uznaje priorytet rywala i wysyła ACK_PONY.

Proces w tym stanie ubiega się o możliwość zabrania stroju kucyka, czyli na dostęp do sekcji krytycznej.

WAIT_SUBMAR

1. W tym stanie proces ubiega się o dostęp do kolejnej sekcji
2. krytycznej – o zajęcie n miejsc na jednej z łodzi podwodnych.

3. Do kolejnego stanu – BOARDED – przechodzi, kiedy zajmie zasoby
4. – miejsca na pokładzie.

WAIT_SUBMAR - odpowiedzi

1. REQ_PONY – nic nie odpowiada, tylko dodaje id nadawcy do swojej listy QUEUE_PONY.

3. REQ_SUBMAR{id_łodzi} – proces dodaje nadawcę do kolejki QUEUE_SUBMAR{id_łodzi} i odpowiada ACK_SUBMAR{id_łodzi}.

Algorytm

(1.) Proces znajduje się w stanie RESTING.

(2.) Po upływie losowo wybranego czasu przechodzi do stanu WAIT_PONY i zaczyna ubiegać się o dostęp do sekcji krytycznej algorytmem bazującym na alg.Ricarta-Agrawali.

(3.) Proces wysyła do pozostałych wiadomość REQ_PONY i czeka na odpowiedź.

Algorytm

(4.) Każdy proces, który otrzyma REQ_PONY:
  (a) Jeśli znajduje się w stanie RESTING odpowiada ACK_PONY.
  (b) Jeśli znajduje się w WAIT_PONY i odebrana wiadomość ma niższy priorytet, niż jego własna, nic nie odpowiada, tylko dodaje nadawcę do QUEUE_PONY.
  (c) Jeśli znajduje się w WAIT_PONY i odebrana wiadomość ma wyższy priorytet, uznaje pierwszeństwo nadawcy i odpowiada ACK_PONY.
  (d) Jeśli znajduje się w którymś z pozostałych stanów ma przyznany strój kucyka (Jest w sekcji krytycznej). Nic nie odpowiada, tylko dodaje nadawcę do listy QUEUE_PONY.

Algorytm

(5.) Tutaj następuje modyfikacja alg. Ricarta - Agrawali. Proces ubiegający się o strój kucyka nie musi czekać na otrzymanie wszystkich potwierdzeń, bo strojów w systemie jest >= 1. Dlatego proces może pobrać strój kucyka, gdy otrzyma *(liczba procesów - liczba strojów)* odpowiedzi ACK_PONY. Przyjmujemy, że od razuma jedno potwierdzenie - swoje własne.

(6.) Po zebraniu wymaganej liczby potwierdzeń proces przechodzi do stanu WAIT_SUBMAR.

(7.) Proces wybiera łódź. Przyjęliśmy, że będzie to łódź, która według jego aktualnej wiedzy jest w najmniejszym stopniu zajęta.

Algorytm

(8.) Proces wysyła do wszystkich pozostałych zapytanie REQ_SUBMAR{id_łodzi} i dodaje siebie do kolejki QUEUE_SUBMAR{id_łodzi}.

(9.) Proces, który otrzymał zapytanie REQ_SUBMAR{id_łodzi} dodaje nadawcę do kolejki QUEUE_SUBMAR{id_łodzi} i wysyła odpowiedź ACK_SUBMAR{id_łodzi}.

Algorytm

W dalszej części algorytmu potrzebny będzie proces, który wyda sygnał do odpłynięcia i potem powrotu. Turyści znajdujący się na łodzi mogliby ubiegać się o dostęp do kolejnej sekcji krytycznej. Jednakże, możemy połączyć tę sekcję z sekcją wsiadania do łodzi i ponownie skorzystając z kolejki QUEUE_SUBMAR{id_łodzi}, ograniczając tym samym konieczną liczbę przesłanych wiadomości. Zatem sygnał do odpłynięcia i powrotu wyda proces mający pierwszą pozycję w kolejce.

Algorytm

(10.) Po otrzymaniu wszystkich ACK_SUBMAR proces sprawdza, czy zmieści się na łodzi. Tutaj następuje rozszerzenie alg. Lamporta. Zająć miejsce na łodzi, czyli wejść do sekcji krytycznej może proces, który w kolejce powiązanej z łodzią znajduje się na pozycji i, jeśli suma rozmiarów turystów na pozycjach $<= i$ nie przekracza maksymalnej pojemności łodzi. Jeśli się zmieści to zajmuje miejsce, wysyła do pierwszego procesu z kolejki wiadomość TRAVEL_READY i przechodzi do stanu BOARDED. Jeśli nie, sprawdza czy przekroczył już maksymalną liczbę prób, jeśli tak to się poddaje i stwierdza, że poczeka sobie w kolejce. Wysyła wtedy do procesów FULL_SUBMAR_STAY{id_łodzi}. W przeciwnym razie wysyła do procesów wiadomość FULL_SUBMAR{id_łodzi}, po czym usuwa się z kolejki.
Wybiera kolejną łódź i wraca do kroku (8.).

Algorytm

(11.) Procesy, które otrzymały FULL_SUBMAR{id_łodzi} usuwają nadawcę z kolejki QUEUE_SUBMAR{id_łodzi} i oznaczają na LIST_SUBMAR, że dana łódź jest już niedostępna. Jeśli była to wiadomość FULL_SUBMAR_STAY{id_łodzi} jedynie oznaczają łódź jako niedostępną.

(11.a) Proces na pierwszej pozycji w kolejce rozpoczyna przygotowanie do rozpoczęcia podróży. Jeśli sam jeszcze nie zajął zasobów (jest w stanie WAIT_SUBMAR) odkłada to działanie, aż nie przejdzie do BOARDED. Jeśli jest już w BOARDED sprawdza czy otrzymał już gotowość (TRAVEL_READY) od pozostałych procesów w sekcji krytycznej. Kiedy już otrzyma wszystkie potwierdzenia wysyła do wszystkich procesów w łodzi wiadomość DEPARTED_SUBMAR. Czeka, aż wszyscy odpowiedzą ACK_TRAVEL, po czym wydaje wygnał do odpłynięcia i

Algorytm

(12.) Proces, który otrzyma ACK_TRAVEL przechodzi w stan TRAVEL i czeka na zakończenie zwiedzania.

(13.) Po pewnym losowym czasie proces informuje pozostałe o zakończeniu podróży. W pierwszej kolejności, wysyła RETURN_SUBMAR{id_łodzi, liczba_pasażerów}, do turystów, którzy z nim płynęli (może to stwierdzić patrząc na kolejkę). Chcemy, aby mogli opuścić łódź, nim nowi turyści na nią wsiądą. Po czym zwalnia łódź.

Algorytm

(14.) Procesy, które otrzymały RETURN_SUBMAR{id_łodzi, liczba_pasażerów} zwalniają łódź, usuwają z kolejki pierwsze liczba_pasażerów pozycji, odnotowują przybycie w LIST_SUBMAR oraz odpowiadają ACK_TRAVEL. Na końcu przechodzą do stanu TRAVEL_END.

(15.) Po otrzymaniu wszystkich potwierdzeń "kapitan" wysyła RETURN_SUBMAR{id_łodzi, liczba_pasażerów} do pozostałych procesów, informując je, że łódź jest już dostępna. Po czym usuwa pierwsze liczba_pasażerów pozycji z kolejki. W ten sposób redukujemy liczbę potrzebnych wiadomości. Normalnie, każdy proces zwalniający sekcję krytyczną musiałby poinformować o tym pozostałe. Ponieważ wszyscy turyści w łodzi opuszczają ją w tym samym czasie, to możemy połączyć wszystkie te wiadomości w jedną.

Algorytm

(16.) Proces, który otrzymał RETURN_SUBMAR{id_łodzi, liczba_pasażerów} usuwa pierwsze liczba_pasażerów z kolejki i odnotowuje fakt przybycia łodzi w LIST_SUBMAR.
(17.) Proces w stanie TRAVEL_END zwalniają strój kucyka wysyłając ACK_PONY do wszystkich procesów z QUEUE_PONY oraz czyści tę listę. Następnie przechodzi do RESTING, czym wraca do kroku (1.).