Nanozombie - algorytm

Marcin Pastwa, Piotr Tomaszewski

QUEUE_PONY

Kolejka procesów oczekujących na ACK_PONY.
Kolejka jest początkowo pusta.
Do kolejki trafiają procesy, które ubiegają się o dostęp do sekcji krytycznej, gdy ten proces się w niej znajduje. Czyli turyści, którzy proszą go o zgodę na pobranie stroju kucyka, gdy ma on przy sobie taki strój.
Po wyjściu z sekcji krytycznej (po zwrocie stroju) do wszystkich procesów z kolejki wysyłane jest ACK_PONY, kolejka jest następnie czyszczona.

QUEUE_SUBMAR{id_łodzi}

Każdy proces posiada po jednej kolejce dla każdej łodzi podwodnej. W kolejce znajdują się procesy ubiegające się o miejsce na niej.

Wartość ujęta w nawiasy klamrowe oznacza, że mówimy o kolejce powiązanej z łodzią o danym indentyfikatorze.

Dzięki kolejce można wyznaczyć, które procesy mogą zająć miejsce na danej łodzi podwodnej (znaleźć się w strefie krytycznej).

Kolejka wypełniana jest zgodnie z algorytmem Lamporta, z tą różnicą, że w sekcji krytycznej na raz może znajdować się >1 proces. Dokładniej, proces, który otrzymał wszystkie potwierdzenia i w kolejce ma pozycję "*i*", może zająć miejsce (wejść do strefy krytycznej), jeśli suma rozmiarów procesów na pozycjach <="*i*" jest <= maksymalnej pojemności danej łodzi.

TRY_NO i MAX_TRY_NO

Aby proces, który czeka w kolejce do łodzi, ale nie starczyło dla niego miejsca nie musiał oczekiwać, aż ta łódź odbędzie podróż i ponownie przybije do brzegu, wycofuje się i próbuje wsiąść do innej łodzi. Takie zachowanie stwarza jednak ryzyko, że proces nigdy nie wyruszy na wyprawę. W celu rozwiązania tego problemu proces zlicza w zmiennej TRY_NO ile razy musiał się wycofać. Zmienna ta jest na początku równa 0 i jest inkrementowana przy każdym wycofaniu. Gdy procesowi uda się zająć miejsce na łodzi zmienna ta jest zerowana. Jeśli jednak liczba prób przekroczy próg MAX_TRY_NO, wtedy proces poddaje się i nie próbuje już zmieniać łodzi, tylko zostaje w kolejce do obecnej.

MAX_TRY_NO jest parametrem, jego wartość jest pewną liczbą $>= 0$. Dokładna wartość tej stałej powinna zostać dobrana eksperymentalnie.

LIST_SUBMAR

Lista, w której proces przechowuje informację o każdej łodzi podwodnej, czy jego zdaniem znajduje się ona teraz w porcie, niezapełniona, czy też nie.
Proces preferuje wybór łodzi o najmniejszym stopniu zapełnienia. Wyznaczanie zapełnienia jest złożone obliczeniowo, dlatego jako pewien rodzaj heurystyki przyjmujemy, że proces rozważa tylko te łodzie, które (według jego obecnej wiedzy) stoją w porcie i są niezapełnione. Jak zostało to już podkreślone, zawartość tej listy może być nieaktualna. Nie spowoduje to jednak błędów. W najgorszym wypadku, proces ustawi się w kolejce do łodzi, która odpłynęła i będzie musiał się z niej wycofać i wybrać inną. Jednak, ten problem występuje niezależnie od tego, czy wykorzystamy tę listę, czy nie. Próba uniknięcia tego zjawiska wiązałaby się z ograniczeniem współbieżności.

DICT_TOURIST_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 przesłać swój rozmiar pozostałym 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.

Pozostałę stałe i zmienne

PONY_NO Łączna liczba dostępnych strojów kucyka (stała).

TOURIST_NO Łączna liczba turystów (stała).

SUBMAR_NO Łączna liczba łodzi podwodnych (stała).

Znacznik Lamporta (Timestamp)

1. Do każdej wiadomości dołączany jest znacznik czasowy (timestamp)
2. modyfikowany zgodnie z zasadami zegara logicznego Lamporta.

Wiadomości

REQ_PONY żądanie dostępu do sekcji krytycznej (żądanie dostępu do stroju kucyka).

ACK_PONY potwierdzenie dostępu do stroju kucyka.

REQ_SUBMAR{id_łodzi} żądanie dostępu do sekcji krytycznej (do miejsca na łodzi podwodnej). Id_łodzi jest parametrem wiadomości.

ACK_SUBMAR{id_łodzi} potwierdzenie wpisania do kolejki do wskazanej łodzi (id_łodzi).

FULL_SUBMAR_RETREAT{id_łodzi} wysyłane przez proces, któremu nie udało się wsiąść do łodzi, więc się z niej wycofuje.

FULL_SUBMAR_STAY{id_łodzi} wysyłane przez proces, któremu nie udało się wsiąść do łodzi, ale poddaje się i zostaje w kolejce.

Wiadomości c.d.

RETURN_SUBMAR{id_łodzi, liczba_pasażerów} informacja, że łódź o podanym id i wioząca podaną liczbę pasażerów wróciła do portu. Dla procesów, które płynęły tą łodzią oznacza to rozkaz zwolnienia zasobów (opuszczenia jej), dla ogółu procesów, że można usunąć z kolejki podaną drugim parametrem liczbę procesów.

TRAVEL_READY informacja, że proces zajął zasoby (wsiadł do łodzi).

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

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

Do kolejnego stanu – BOARDED – przechodzi, kiedy zajmie zasoby – 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 razu ma jedno potwierdzenie - swoje własne.

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

Algorytm

(7.) Proces wybiera łódź. Przyjęliśmy, że będzie to łódź, która według jego aktualnej wiedzy jest w najmniejszym stopniu zajęta. Wyznaczenie zajętości może być wymagające obliczeniowo, dlatego proces rozważa tylko te łodzie, na które jego zdaniem są jeszcze dostępne (LIST_SUBMAR). Jeśli takiej łodzi nie mat, to proces czeka na sygnał RETURN_SUBMAR.

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_RETREAT{id_łodzi}, po czym usuwa się z kolejki.

Wybiera kolejną łódź i wraca do kroku (8.).

Algorytm

(11.) Procesy, które otrzymały FULL_SUBMAR_RETREAT{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. ????TODO:Kiedy już otrzyma wszystkie potwierdzenia wysyła do wszystkich procesów w łodzi wiadomość DEPART_SUBMAR. Czeka, aż wszyscy odpowiedzą

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 ON_SHORE.
(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 ON_SHORE zwalniaja strój kucyka wysyłając ACK_PONY do wszystkich procesów z QUEUE_PONY oraz czyści tę listę. Następnie przechodzi do RESTING, czym wraca do kroku (1.).