

Nanozombie - algorytm

Marcin Pastwa, Piotr Tomaszewski

LIST_PONY_POSTPONED

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

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

QUEUE_SUBMAR{id_łodzi}

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

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

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

LIST_SUBMAR

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

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

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

DICT_TOURISTS_SIZES

- 1 Tablica pogładowa lub w ogólności słownik, w którym kluczem jest
 - 2 identyfikator procesu, natomiast wartością jest rozmiar turysty (ile
 - 3 miejsc na łodzi zajmuje). Wartości te są stałe, więc na potrzeby
 - 4 algorytmu przyjmujemy, że są już każdemu procesowi znane.
-
- 5 Jeśli jednak założyć, że wartości te nie są znane z góry, procesy
 - 6 musiałyby wymienić się nimi między sobą przed rozpoczęciem pętli
 - 7 głównej.

DICT_SUBMAR_CAPACITY

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

Znacznik Lamporta (Timestamp)

- 1 W celu określenia relacji uprzedniości zdarzeń w algorytmie
- 2 zastosowany jest zegar logiczny Lamporta.
- 3 Do wysyłanych przez proces wiadomości dołączane są znaczniki
- 4 czasowe.
- 5 Uznaliśmy, że uwzględnienie aktualizacji zegarów i przesyłu
- 6 znaczników jedynie zmniejszyłoby czytelność algorytmu, dlatego
- 7 zostało w dalszym opisie ograniczone do minimum.
- 8 Rozważaliśmy również dołączanie do odpowiedzi na zapytania
- 9 znacznika tego zapytania. Pozwoliłoby to na weryfikację czy zgoda
- 10 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

- 1 Stanowi potwierdzenie otrzymania prośby o dostęp do stroju
- 2 kucyka.

- 3 Wysyłana w odpowiedzi na zapytanie REQ_PONY, gdy
- 4 odpowiadający proces zgadza się aby pytający uzyskać dostęp do
- 5 zasobu.

STATE_RESTING

- 1 Jest to stan początkowy.
- 2 Symuluje odpoczynek turysty między zakończeniem jednej
- 3 wycieczki, a rozpoczęciem kolejnej.
- 4 Do kolejnego stanu – STATE_WAIT_PONY – proces przechodzi
- 5 w pewnym, nieokreślonym momencie. Przyjmujemy, że ten czas
- 6 jest pewną losową wartością ≥ 0 .

STATE_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.

STATE_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 – STATE_WAIT_SUBMAR – proces
- 4 przechodzi po zabraniu stroju kucyka.

STATE_WAIT_PONY - odpowiedzi

- 1 Na REQ_PONY odpowiada:
- 2 Jeśli otrzymane zapytanie ma niższy priorytet od wysłanego
- 3 przez ten proces nic nie odpowiada, tylko wstawia id nadawcy
- 4 do swojej listy LIST_PONY_POSTPONED.
- 5 W przeciwnym razie, uznaje priorytet rywala i wysyła
- 6 ACK_PONY.
- 7 Proces w tym stanie ubiega się o możliwość zabrania stroju kucyka,
- 8 czyli na dostęp do sekcji krytycznej.

STATE_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 – STATE_BOARDED – przechodzi, kiedy
- 4 zajmie zasoby – miejsca na pokładzie.

STATE_WAIT_SUBMAR - odpowiedzi

- 1 REQ_PONY – nic nie odpowiada, tylko dodaje id nadawcy do
- 2 swojej listy LIST_PONY_POSTPONED.

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

Algorytm

- 6 (1.) Proces znajduje się w stanie STATE_RESTING.
- 7 (2.) Po upływie losowo wybranego czasu przechodzi do stanu
- 8 STATE_WAIT_PONY i zaczyna ubiegać się o dostęp do sekcji
- 9 krytycznej algorytmem bazującym na alg.Ricarta-Agrawali.
- 10 (3.) Proces wysyła do pozostałych wiadomość REQ_PONY i czeka
- 11 na odpowiedź.

Algorytm

- 12 (4.) Każdy proces, który otrzyma REQ_PONY:
- 13 (a) Jeśli znajduje się w stanie STATE_RESTING odpowiada
- 14 ACK_PONY.
- 15 (b) Jeśli znajduje się w STATE_WAIT_PONY i odebrana
- 16 wiadomość ma niższy priorytet, niż jego własna, nic nie
- 17 odpowiada, tylko dodaje nadawcę do
- 18 LIST_PONY_POSTPONED.
- 19 (c) Jeśli znajduje się w STATE_WAIT_PONY i odebrana
- 20 wiadomość ma wyższy priorytet, uznaje pierwszeństwo
- 21 nadawcy i odpowiada ACK_PONY.
- 22 (d) Jeśli znajduje się w którymś z pozostałych stanów
- 23 ma przyznany strój kucyka (Jest w sekcji krytycznej).
- 24 Nic nie odpowiada, tylko dodaje nadawcę do listy
- 25 LIST_PONY_POSTPONED.

Algorytm

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

32 (6.) Po zebraniu wymaganej liczby potwierdzeń proces przechodzi
33 do stanu STATE_WAIT_SUBMAR.

34 (7.) Proces wybiera łódź. Przyjeliśmy, że będzie to pierwsza, która
35 jego zdaniem jest teraz dostępna (LIST_SUBMAR). Jeśli takiej
36 łodzi nie ma, czeka na otrzymanie RETURN_SUBMAR{id_łodzi}
37 i wybiera tę łódź.

Algorytm

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

41 (9.) Proces, który otrzymał zapytanie REQ_SUBMAR{id_łodzi}
42 dodaje nadawcę do kolejki QUEUE_SUBMAR{id_łodzi} i wysyła
43 odpowiedź ACK_SUBMAR{id_łodzi}. Dodatkowo, proces
44 znajdujący się na pierwszej pozycji na liście sprawdza

Algorytm

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

Algorytm

53 (10.) Po otrzymaniu wszystkich ACK_SUBMAR proces sprawdza,
54 czy zmieści się na łodzi. Tutaj następuje rozszerzenie alg.
55 Lamporta. Zająć miejsce na łodzi, czyli wejść do sekcji krytycznej
56 może proces, który w kolejce powiązanej z łodzią znajduje się na
57 pozycji i , jeśli suma rozmiarów turystów na pozycjach $\leq i$ nie
58 przekracza maksymalnej pojemności łodzi. Jeśli się zmieści to
59 zajmuje miejsce, wysyła do pierwszego procesu z kolejki wiadomość
60 TRAVEL_READY i przechodzi do stanu STATE_BOARDED. Jeśli
61 nie, to wysyła do procesów wiadomość
62 FULL_SUBMAR{id_łodzi}, po czym usuwa się z kolejki, wybiera
63 kolejną łódź i wraca do kroku (8.).

Algorytm

64 (11.) Procesy, które otrzymały FULL_SUBMAR{id_łodzi}
65 usuwają nadawcę z kolejki QUEUE_SUBMAR{id_łodzi} i
66 oznaczają na LIST_SUBMAR, że dana łódź jest już niedostępna.
67

68 (11.a) Proces na pierwszej pozycji w kolejce rozpoczyna
69 przygotowanie do rozpoczęcia podróży. Jeśli sam jeszcze nie zajął
70 zasobów (jest w stanie STATE_WAIT_SUBMAR) odkłada to
71 działanie, aż nie przejdzie do STATE_BOARDED. Jeśli jest już w
72 STATE_BOARDED sprawdza czy otrzymał już gotowość
73 (TRAVEL_READY) od pozostałych procesów w sekcji krytycznej.
74 Kiedy już otrzyma wszystkie potwierdzenia wysyła do wszystkich
75 procesów w łodzi wiadomość DEPARTED_SUBMAR. Czeki, aż
76 wszyscy odpowiedzą ACK_TRAVEL, po czym wydaje wygnął do
77 odpłynięcia i przechodzi w stan STATE_TRAVEL.

Algorytm

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

80

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

Algorytm

87 (14.) Procesy, które otrzymały SUB_RETURN{id_łodzi,
88 liczba_pasażerów} zwalniają łódź, usuwają z kolejki pierwsze
89 liczba_pasażerów pozycji, odnotowują przybycie w
90 LIST_SUBMAR oraz odpowiadają ACK_TRAVEL. Na końcu
91 przechodzą do stanu STATE_TRAVEL_END.

92 (15.) Po otrzymaniu wszystkich potwierdzeń "kapitan" wysyła
93 SUB_RETURN{id_łodzi, liczba_pasażerów} do pozostałych
94 procesów, informując je, że łódź jest już dostępna. Po czym usuwa
95 pierwsze liczba_pasażerów pozycji z kolejki.

Algorytm

96 (16.) Proces, który otrzymał SUB_RETURN{id_łodzi,
97 liczba_pasażerów} usuwa pierwsze liczba_pasażerów z kolejki i
98 odnotowuje fakt przybycia łodzi w LIST_SUBMAR.
99 (17.) Proces w stanie STATE_TRAVEL_END zwalniając strój
100 kucyka wysyłając ACK_PONY do wszystkich procesów z
101 LIST_PONY_POSTPONED oraz czyści tę listę. Następnie
102 przechodzi do STATE_RESTING, czym wraca do kroku (1.).