

Nanozombie - algorytm

Marcin Pastwa, Piotr Tomaszewski

QUEUE_PONY

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

QUEUE_SUBMAR{id_łodzi}

- 1 Każdy proces posiada po jednej kolejce dla każdej łodzi podwodnej.
- 2 W kolejce znajdują się procesy ubiegające się o miejsce na niej.
- 3 Wartość ujęta w nawiasy klamrowe oznacza, że mówimy o kolejce
- 4 powiązanej z łodzią o danym identyfikatorze.
- 5 Dzięki kolejce można wyznaczyć, które procesy mogą zająć miejsce
- 6 na danej łodzi podwodnej (znaleźć się w strefie krytycznej).
- 7 Kolejka wypełniana jest zgodnie z algorytmem Lamporta, z tą
- 8 różnicą, że w sekcji krytycznej na raz może znajdować się >1
- 9 proces. Dokładniej, proces, który otrzymał wszystkie potwierdzenia
- 10 i w kolejce ma pozycję "i", może zająć miejsce (wejść do strefy
- 11 krytycznej), jeśli suma rozmiarów procesów na pozycjach $\leq "i"$
- 12 jest \leq maksymalnej pojemności danej łodzi.

TRY_NO i MAX_TRY_NO

- 1 Aby proces, który czeka w kolejce do łodzi, ale nie starczyło dla
2 niego miejsca nie musiał oczekiwać, aż ta łódź odbędzie podróż i
3 ponownie przybije do brzegu, wycofuje się i próbuje wsiąść do innej
4 łodzi. Takie zachowanie stwarza jednak ryzyko, że proces nigdy nie
5 wyruszy na wyprawę. W celu rozwiązania tego problemu proces
6 zlicza w zmiennej TRY_NO ile razy musiał się wycofać. Zmienna
7 ta jest na początku równa 0 i jest inkrementowana przy każdym
8 wycofaniu. Gdy procesowi uda się zająć miejsce na łodzi zmienna
9 ta jest zerowana. Jeśli jednak liczba prób przekroczy próg
10 MAX_TRY_NO, wtedy proces poddaje się i nie próbuje już
11 zmieniać łodzi, tylko zostaje w kolejce do obecnej.
12 MAX_TRY_NO jest parametrem, jego wartość jest pewną liczbą
13 ≥ 0 . Dokładna wartość tej stałej powinna zostać dobrana
14 eksperymentalnie.

LIST_SUBMAR

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

DICT_TOURIST_SIZES

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

Pozostałe 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 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. (Na przykład czy
- 11 ACK_PONY nie dotyczy poprzedniej iteracji)

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 uzyskał dostęp do
- 5 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

- 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 QUEUE_PONY.
- 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.

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
- 2 swojej listy QUEUE_PONY.

- 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 RESTING.
- 7 (2.) Po upływie losowo wybranego czasu przechodzi do stanu
- 8 WAIT_PONY i zaczyna ubiegać się o dostęp do sekcji krytycznej
- 9 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 RESTING odpowiada
- 14 ACK_PONY.
- 15 (b) Jeśli znajduje się w WAIT_PONY i odebrana
- 16 wiadomość ma niższy priorytet, niż jego własna, nic nie
- 17 odpowiada, tylko dodaje nadawcę do
- 18 QUEUE_PONY.
- 19 (c) Jeśli znajduje się w 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 QUEUE_PONY.

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 razu
31 ma jedno potwierdzenie - swoje własne.

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

Algorytm

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

Algorytm

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

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

Algorytm

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

Algorytm

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

Algorytm

69 (11.) Procesy, które otrzymały
70 FULL_SUBMAR_RETREAT{id_łodzi} usuwają nadawcę z kolejki
71 QUEUE_SUBMAR{id_łodzi} i oznaczają na LIST_SUBMAR, że
72 dana łódź jest już niedostępna. Jeśli była to wiadomość
73 FULL_SUBMAR_STAY{id_łodzi} jedynie oznaczają łódź jako
74 niedostępna.

75

76 (11.a) Proces na pierwszej pozycji w kolejce rozpoczyna
77 przygotowanie do rozpoczęcia podróży. Jeśli sam jeszcze nie zajął
78 zasobów (jest w stanie WAIT_SUBMAR) odkłada to działanie, aż
79 nie przejdzie do BOARDED. Jeśli jest już w BOARDED sprawdza
80 czy otrzymał już gotowość (TRAVEL_READY) od pozostałych
81 procesów w sekcji krytycznej. Kiedy już otrzyma wszystkie
82 potwierdzenia wysyła do wszystkich procesów w łodzi wiadomość
83 DEPARTED_SUBMAR. Czeką, aż wszyscy odpowiedzą

Algorytm

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

88

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

Algorytm

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

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

Algorytm

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