

Nanozombie - Sprawozdanie

Marcin Pastwa 136779
Piotr Tomaszewski 136821

1 Opis problemu i słowny opis rozwiązania

1 Zadanie polegało na zrealizowaniu dostępu do sekcji krytycznej w systemie rozproszonym.
2 W naszym rozwiązaniu wyróżniliśmy dwie sekcje krytyczne. Pierwsza - pobranie stroju kucyka,
3 druga - zajęcie miejsca na łodzi.
4 Współbieżny dostęp do strojów kucyka zrealizowany został bazując na algorytmie Ricarta -
5 Agravali. Różnica, w stosunku do podstawowej wersji algorytmu polega na zwiększeniu rozmiaru
6 sekcji krytycznej do liczby procesów równej liczbie strojów kucyka. Proces może zabrać strój po
7 otrzymaniu co najmniej takiej liczby zgód, że nawet gdyby wszystkie procesy, od których jeszcze
8 nie dostał zgody znajdowały się w sekcji krytycznej (miały strój), to i tak zostałyby co najmniej
9 jeden strój wolny.
10 Po pobraniu stroju kucyka, proces wybiera łódź podwodną. Wyboru dokonuje stosując
11 następującą heurystykę: Spośród łodzi, które jednocześnie uznawane są za dostępne oraz w
12 powiązanej z nimi kolejce znajduje się jakiś proces, wybierana jest taka, która cechuje się na-
13 jmniejszym ilorazem zajętej pojemności do całkowitej pojemności. Jeśli taka łódź nie istnieje,
14 wybierana jest pusta łódź o najniższym id. Jeśli pustych łodzi również nie ma, wybierana jest
15 dowolna łódź.
16 Po wybraniu łodzi proces zaczyna ubiegać się o dostęp do niej. Przyjęty algorytm bazuje
17 na algorytmie Lamporta. Ponownie, została jednak rozszerzona sekcja krytyczna - proces może
18 wejść do sekcji krytycznej, jeśli suma rozmiarów jego i procesów znajdujących się przed nim w
19 kolejce nie przekracza maksymalnej pojemności łodzi.
20 Proces, który znajdzie się w sekcji krytycznej sprawdza czy może mianować się kapitanem
21 łodzi. Kapitanem jest proces, który wygrałby dostęp do sekcji krytycznej w podstawowej wersji
22 alg. Lamporta - czyli jest pierwszy w kolejce.
23 Sygnał do odpłynięcia łodzi wydaje kapitan. Dokonuje tego, gdy ustali, że nikt więcej nie
24 wsiądzie na łódź, tj. gdy otrzyma wiadomość od jakiegoś procesu, że ten nie może zmieścić
25 się na daną łódź albo wykryje, że nikt więcej nie będzie mógł wsiąść, gdyż łącznie na łodziach
26 przebywa maksymalna dopuszczalna liczba turystów (tj. $\min\{\text{liczba strojów kucyka}, \text{liczba tu-}$
27 $\text{rystów}\}$). Kapitan wylicza ile procesów z kolejki zmieści się na łodzi, po czym wysyła do nich
28 zapytanie mające na celu upewnienie się, że wszystkie z nich są już gotowe do podróży.
29 Po otrzymaniu wszystkich potwierdzeń wysyła do nich informację o rozpoczęciu podróży. Po
30 ponownym zebraniu potwierdzeń rozpoczyna podróż.
31 Po zakończeniu podróży informuje pasażerów na swojej łodzi o tym fakcie, dając im czas na
32 opuszczenie łodzi. Wiadomość ta jest odpowiednikiem informacji o zwolnieniu sekcji krytycznej
33 w algorytmie Lamporta, jest ona jednak zgrupowana - kapitan informuje o zwolnieniu sekcji
34 krytycznej przez wszystkich pasażerów. Procesy odsyłają kapitanowi potwierdzenie opuszczenia
35 i wracają na brzeg. Po zebraniu potwierdzeń, kapitan wysyła tę wiadomość ponownie, tym razem
36 do procesów, które nie przebywały z nim na pokładzie, nie oczekuje już jednak potwierdzenia,
37 tylko od razu wraca na brzeg.

38 Na brzegu następuje zwolnienie sekcji krytycznej związanej ze strojem kuczka. Zgodnie z
39 zastosowanym alg. Ricarta - Agravali, następuje wysłanie zgody na pobranie stroju do wszystkich
40 procesów, które wysłały prośbę w czasie, gdy omawiany proces znajdował się w sekcji krytycznej.

1 2 Założenia

2 Przyjmujemy, że środowisko jest w pełni asynchroniczne, kanały komunikacyjne są FIFO i nieza-
3 wodne. Procesy nie ulegają awarii.
4 Ponadto, przyjmujemy, że każdy turysta może zmieścić się w każdej pustej łodzi podwodnej, to
5 jest $\max\{S_{tourist}\} \leq \min\{C_{submarine}\}$, gdzie $S_{tourist}$ to zbiór rozmiarów turystów, $C_{submarine}$
6 to zbiór pojemności łodzi podwodnych.
7 Gdyby zrezygnować z tego założenia, w szczególnym wypadku mógłby istnieć turysta, który nie
8 zmieści się na żadnej łodzi. Proces taki nie mógłby brać udziału w przetwarzaniu, pozostawałby
9 więc w uśpieniu, nigdy nie ubiegając się o dostęp do sekcji krytycznej.

1 3 Złożoność czasowa i komunikacyjna

2 Przyjmijmy, że N reprezentuje liczbę procesów w systemie.

3 3.1 Zajęcie i zwolnienie stroju kuczka

4 Zajęcie sekcji krytycznej związanej ze strojem kuczka wymaga wysłania $N-1$ zapytań REQ_PONY.
5 W odpowiedzi otrzymamy $N-1$ ACK_PONY. Zwolnienie sekcji krytycznej nie generuje żadnych
6 dodatkowych wiadomości - wysyłane są tylko zakolejkowane wiadomości. Złożoność komunika-
7 cyjna wynosi więc $2(N-1)$, natomiast złożoność czasowa: 2.

8 3.2 Dostęp do łodzi podwodnej oraz rozpoczęcie i zakończenie wyprawy

9 Przyjmijmy, że M reprezentuje liczbę turystów znajdujących się na danej łodzi podwodnej.
10 Zajęcie sekcji krytycznej związanej z dostępem do łodzi podwodnej wymaga wysłania $N-1$
11 zapytań REQ_SUBMAR. W odpowiedzi otrzymujemy $N-1$ odpowiedzi ACK_SUBMAR. W
12 celu rozpoczęcia wyprawy, kapitan musi wysłać $M-1$ TRAVEL_READY, aby upewnić się,
13 że pozostali pasażerowie są gotowi do drogi. W odpowiedzi dostaje $M-1$ ACK_TRAVEL.
14 Po upewnieniu się, musi ponownie wysłać $M-1$ wiadomości DEPART_SUBAR i otrzymać
15 $M-1$ odpowiedzi. Aby zakończyć podróż musi wysłać $N-1$ wiadomości RETURN_SUBMAR.
16 W rezultacie otrzymuje $M-1$ odpowiedzi. (Dokładniej, najpierw wysyła $M-1$ wiadomości
17 do współpasażerów, otrzymuje $M-1$ potwierdzeń, po czym wysyła $N-M$ wiadomości do
18 pozostałych procesów, nie oczekując już na potwierdzenie). Łącznie, więc należy przesłać $3(N-$
19 $1) + 5(M-1)$ wiadomości. Złożoność czasowa wynosi: 9.

1 3.3 Struktury i zmienne

2 *received_ack_no* Liczba otrzymanych wiadomości typu ACK. Jest ustawiana na 1 przed
3 rozesłaniem wiadomości wymagającej zebrania potwierdzeń.

4 *my_req_pony_timestamp* Zmienna przechowująca wartość zegara Lamporta w momencie
5 wysłania wiadomości REQ_PONY.

6 **queue_pony** Kolejka, na której umieszczane są id nadawców, od których proces otrzymał
7 REQ_PONY, gdy był w sekcji krytycznej. Uwaga, kolejka ta nie zawiera duplikatów -
8 jeśli dane id już widnieje w kolejce, nie zostanie do niej ponownie dodane.

9 **available_submarine_list** Lista łodzi podwodnych, w której proces przechowuje informa-
10 cję, czy jego zdaniem łodzie mają jeszcze wolne miejsca, czy też nie. Jej głównym zas-
11 tosowaniem jest uniknięcie sytuacji, w której proces po wycofaniu się z kolejki powiązanej
12 z łodzią i w trakcie wyboru kolejnej łodzi ponownie wybrałby tę samą. Początkowo, wszys-
13 tkie łodzie są oznaczone jako dostępne.

14 **queue_submar{id}** Lista kolejek (po jednej dla każdej łodzi podwodnej). Proces przechowuje
15 na niej pary (znacznik czasowy, id procesu) dla każdego z procesów, które wysłały do
16 niego REQ_SUBMAR z parametrem stanowiącym id łodzi. Uwaga, procesy w ramach
17 każdej kolejki są posortowane malejąco według priorytetu. Im niższy znacznik czasowy,
18 tym wyższy priorytet. W przypadku równych znaczników, wyższy priorytet ma proces o
19 niższym id. Początkowo, wszystkie kolejki są puste.

20 **try_no** Liczba niepowodzeń przy zajmowaniu miejsca w łodzi podwodnej. Początkowo równa
21 0.

22 **my_submarine_id** Id łodzi podwodnej wybranej przez proces. Początkowo może mieć dowolną
23 wartość - pierwszy odczyt wartości tej zmiennej zawsze występuje po pierwszym zapisie.

24 **boarded_on_my_submarine** Lista tworzona przez kapitana, w której przechowuje id pro-
25 cesów, które znajdują się na pokładzie jego łodzi podwodnej. Początkowo pusta.

26 **lambport_clock** Zmienna, w której proces przechowuje aktualną wartość zegara Lamporta.
27 Początkowo, wartość ta równa jest 0. Jest zwiększana o 1 przy wysyłaniu wiadomości
28 (niezależnie od tego, do ilu procesów wiadomość ma dotrzeć), nowa wartość dołączana jest
29 do wiadomości. Po otrzymaniu wiadomości jest ustawiana na $\max\{\text{lambport_clock}, \text{znacznik}$
30 $\text{z otrzymanej wiadomości}\} + 1$.

31 **Packet** Struktura reprezentująca wiadomość. Zawiera następujące pola: typ wiadomości, wartość
32 zegara Lamporta, id łodzi podwodnej, liczba pasażerów. Pierwsze dwa parametry są za-
33 wsze wymagane, kolejne dwa, w zależności od typu wiadomości, są wymagane albo mogą
34 mieć dowolną wartość.

1 3.4 Stałe

2 **Tourist_no** Całkowita liczba turystów w systemie.

3 **Pony_no** Całkowita liczba strojów kucyka.

4 **Submar_no** Całkowita liczba łodzi podwodnych.

5 **Dict_tourist_sizes** Słownik, w którym kluczem jest id turysty, wartością jest jego rozmiar.

6 **Dict_submar_capacity** Słownik, w którym kluczem jest id łodzi podwodnej, wartością jest
7 jej całkowita pojemność.

8 **Max_try_no** Próg decydujący o tym, ile prób wejścia na łódź może podjąć proces, nim
9 postanowi zostać w kolejce do łodzi, w której aktualnie nie może się zmieścić. Ma na
10 celu zapewnienie warunku postępu. Może mieć wartość równą 0, wtedy proces nigdy nie
11 będzie się wycofywał z kolejki.

1 3.5 Wiadomości

- 2 **REQ_PONY** Żądanie dostępu do pierwszej sekcji krytycznej (żądanie dostępu do stroju kucyka).
- 3 **ACK_PONY** Zgoda na pobranie stroju kucyka.
- 4 **REQ_SUBMAR**{*submarine_id*} Żądanie dostępu do drugiej sekcji krytycznej (żądanie
5 dostępu do wskazanej łodzi podwodnej).
- 6 **ACK_SUBMAR** Potwierdzenie wpisania nadawcy do odpowiedniej kolejki *queue_submar*.
- 7 **FULL_SUBMAR_RETREAT**{*submarine_id*} Wysyłana przez proces, który nie zmieścił
8 się na łodzi podwodnej, nie przekroczył jeszcze progu *Max_try_no*, więc się wycofuje.
- 9 **FULL_SUBMAR_STAY**{*submarine_id*} Wysyłana przez proces, który nie zmieścił się
10 na łodzi podwodnej, przekroczył jednak próg *Max_try_no*, informuje więc, że pozostaje
11 w kolejce do łodzi.
- 12 **RETURN_SUBMAR**{*submarine_id*, *passenger_no*} Wysyłana przez kapitana w celu
13 poinformowania o powrocie łodzi, którą ten kapitan płynął.
- 14 **TRAVEL_READY** Wysyłana przez kapitana w celu weryfikacji, czy wszystkie procesy, które
15 mogą wsiąść na łódź podwodną przeszły już do stanu **BOARDED**.
- 16 **ACK_TRAVEL** Potwierdzenie odsyłane kapitanowi zgodnie z zasadami podanymi w sekcji
17 szczegółowego opisu.
- 18 **DEPART_SUBMAR** Wysyłana przez kapitana w celu poinformowania, że łódź, na której
19 zarówno kapitan, jak i odbiorca przebywają właśnie odpływa z portu.
- 20 **DEPART_SUBMAR_NOT_FULL** Wysyłana przez kapitana w celu poinformowania, że
21 łódź, na której zarówno kapitan, jak i odbiorca przebywają właśnie odpływa z portu oraz,
22 że wykryte zostało zakleszczenie.

1 3.6 Stany

- 2 Początkowym stanem procesu jest **RESTING**.
- 3 **RESTING** Nie ubiega się o dostęp do żadnej sekcji krytycznej. (Turysta odpoczywa).
- 4 **WAIT_PONY** Ubiega się o dostęp do pierwszej sekcji krytycznej - o pobranie stroju kucyka.
- 5 **CHOOSE_SUBMAR** Pobrał strój kucyka, wybiera sobie łódź podwodną.
- 6 **WAIT_SUBMAR** Ubiega się o dostęp do drugiej sekcji krytycznej - miejsce na łodzi pod-
7 wodnej.
- 8 **BOARDED** Oczekuje na rozpoczęcie podróży.
- 9 **TRAVEL** Podróżuje łodzią podwodną.
- 10 **ON_SHORE** Zakończył podróż, opuszcza obie sekcje krytyczne.

1 3.7 Opis szczegółowy

2 3.7.1 RESTING

3 Stan początkowy.

4 Proces przebywa w stanie **RESTING** do czasu, aż podejmie decyzję o rozpoczęciu wyprawy.

5 Gdy postanowi wyruszyć, zaczyna ubiegać się o dostęp do pierwszej sekcji krytycznej.

6 Ze stanu **RESTING** następuje przejście do **WAIT_PONY** po uprzednim ustawieniu zmiennej *received_ack_no* na 1 (przyjmujemy, że proces ma już swoje pozwolenie) oraz wysłaniu

8 REQ_PONY do wszystkich pozostałych procesów. Wartość zegara Lamporta wysłanej wiadomości jest zapisywana do zmiennej *my_req_pony_timestamp*.

11 Reakcja na wiadomości

12 **REQ_PONY** Odpowiada ACK_PONY.

13 **ACK_PONY** Ignoruje. Otrzymanie tej wiadomości jest możliwe tylko, jeśli proces był już chociaż raz w sekcji krytycznej.

15 **REQ_SUBMAR**{*submarine_id*} Dodaje nadawcę do kolejki *queue_submar{id}* powiązanej ze wskazaną przez parametr *submarine_id* łodzią podwodną i odpowiada ACK_SUBMAR.

17 **ACK_SUBMAR** Niemożliwe, ignoruje.

18 **FULL_SUBMAR_STAY**{*submarine_id*} Oznacza na liście *available_submarine_list*, że łódź wskazywana przez parametr *submarine_id* jest niedostępna.

20 **FULL_SUBMAR_RETREAT**{*submarine_id*} Oznacza na liście *available_submarine_list*, że łódź wskazywana przez parametr *submarine_id* jest niedostępna oraz usuwa nadawcę z kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez parametr *submarine_id*.

24 **RETURN_SUBMAR**{*submarine_id*, *passenger_no*} Oznacza na liście *available_submarine_list*, że łódź wskazywana przez parametr *submarine_id* jest dostępna oraz usuwa z początku kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez parametr *submarine_id* *passenger_no* pozycji.

28 **TRAVEL_READY** Niemożliwe, ignoruje.

29 **ACK_TRAVEL** Niemożliwe, ignoruje.

30 **DEPART_SUBMAR** Niemożliwe, ignoruje.

31 **DEPART_SUBMAR_NOT_FULL** Niemożliwe, ignoruje.

1 3.7.2 WAIT_PONY

2 Proces przechodzi ze stanu **WAIT_PONY** do **CHOOSE_SUBMAR** po otrzymaniu wystarczającej liczby odpowiedzi ACK_PONY, tak aby mieć pewność, że może bezpiecznie zająć sekcję krytyczną (tj. zabrać strój kucyka). Pewność tę uzyskuje, gdy jego zmienna *received_ack_no* ma wartość co najmniej *Tourist_no* – *Pony_no* + 1. W szczególnym przypadku wymagana liczba potwierdzeń może być mniejsza od 1. W takiej sytuacji, proces nie musi czekać na żadne potwierdzenie, może od razu przejść do następnego stanu.

9 Reakcja na wiadomości

10 **REQ_PONY** Porównuje znacznik czasowy otrzymanej wiadomości z *my_req_pony_timestamp*.
 11 Jeśli otrzymana wiadomość ma niższy priorytet (jej znacznik czasowy jest większy od
 12 *my_req_pony_timestamp* albo, jeśli znaczniki są równe, id nadawcy jest większe od id
 13 omawianego procesu), dodaje nadawcę do kolejki *queue_pony* i nic nie odpowiada. W
 14 przeciwnym razie odpowiada ACK_PONY.

15 **ACK_PONY** Inkrementuje zmienną *received_ack_no*.

16 **REQ_SUBMAR{submarine_id}** Dodaje nadawcę do kolejki *queue_submar{id}* powiązanej
 17 ze wskazaną przez parametr *submarine_id* łodzią podwodną i odpowiada ACK_SUBMAR.

18 **ACK_SUBMAR** Niemożliwe, ignoruje.

19 **FULL_SUBMAR_STAY{submarine_id}** Oznacza na liście *available_submarine_list*, że
 20 łódź wskazywana przez parametr *submarine_id* jest niedostępna.

21 **FULL_SUBMAR_RETREAT{submarine_id}** Oznacza na liście
 22 *available_submarine_list*, że łódź wskazywana przez parametr *submarine_id* jest niedostępna
 23 oraz usuwa nadawcę z kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez
 24 parametr *submarine_id*.

25 **RETURN_SUBMAR{submarine_id, passenger_no}** Oznacza na liście *available_submarine_list*,
 26 że łódź wskazywana przez parametr *submarine_id* jest dostępna oraz usuwa z początku
 27 kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez parametr *submarine_id*
 28 *passenger_no* pozycji.

29 **TRAVEL_READY** Niemożliwe, ignoruje.

30 **ACK_TRAVEL** Niemożliwe, ignoruje.

31 **DEPART_SUBMAR** Niemożliwe, ignoruje.

32 **DEPART_SUBMAR_NOT_FULL** Niemożliwe, ignoruje.

1 3.7.3 CHOOSE_SUBMAR

2 Aby przejść ze stanu **CHOOSE_SUBMAR** do **WAIT_SUBMAR** proces musi najpierw
 3 wybrać, o miejsce na której łodzi będzie się ubiegał. Wyboru dokonuje według następującego
 4 algorytmu: Proces wybiera spośród łodzi, w których kolejce *queue_submar{id}* jest przynajm-
 5 niej jeden proces i jednocześnie, na liście *available_submarine_list* są oznaczone jako 'available'.
 6 Jeśli istnieją łodzie spełniające te warunki, to wybrana zostaje taka, która jest w najmniejszym
 7 stopniu zajęta (iloraz zajętego miejsca do całkowitej pojemności - wartość ta jest szacowana
 8 na podstawie listy *queue_submar* i słownika *Dict_tourist_sizes*). Jeśli taka łódź nie istnieje,
 9 wybierana jest pusta łódź o najniższym id. Jeśli takiej łodzi też nie ma, wybierana jest dowolna
 10 łódź. Id wybranej łodzi zostaje zapisane w zmiennej *my_submarine_id*
 11 Po wyborze łodzi, proces ustawia zmienną *received_ack_no* na 1, wysyła do wszystkich po-
 12 zostających procesów wiadomość REQ_SUBMAR{*my_submarine_id*}. Może teraz przejść do
 13 stanu **WAIT_SUBMAR**.

14 Reakcja na wiadomości

16 **REQ_PONY** Proces jest w sekcji krytycznej, więc dodaje nadawcę do kolejki *queue_pony* i
 17 nic nie odpowiada.

18 **ACK_PONY** Ignoruje.

19 **REQ_SUBMAR**{*submarine_id*} Dodaje nadawcę do kolejki *queue_submar{id}* powiązanej
 20 ze wskazaną przez parametr *submarine_id* łodzią podwodną i odpowiada **ACK_SUBMAR**.

21 **ACK_SUBMAR** Niemożliwe, ignoruje.

22 **FULL_SUBMAR_STAY**{*submarine_id*} Oznacza na liście *available_submarine_list*, że
 23 łódź wskazywana przez parametr *submarine_id* jest niedostępna.

24 **FULL_SUBMAR_RETREAT**{*submarine_id*} Oznacza na liście
 25 *available_submarine_list*, że łódź wskazywana przez parametr *submarine_id* jest niedostępna
 26 oraz usuwa nadawcę z kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez
 27 parametr *submarine_id*.

28 **RETURN_SUBMAR**{*submarine_id*, *passenger_no*} Oznacza na liście *available_submarine_list*,
 29 że łódź wskazywana przez parametr *submarine_id* jest dostępna oraz usuwa z początku
 30 kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez parametr *submarine_id*
 31 *passenger_no* pozycji.

32 **TRAVEL_READY** Niemożliwe, ignoruje.

33 **ACK_TRAVEL** Niemożliwe, ignoruje.

34 **DEPART_SUBMAR** Niemożliwe, ignoruje.

35 **DEPART_SUBMAR_NOT_FULL** Niemożliwe, ignoruje.

1 3.7.4 WAIT_SUBMAR

2 Z tego stanu proces może przejść do **BOARDED** albo wrócić do **CHOOSE_SUBMAR**. Pro-
 3 ces czeka na otrzymanie **ACK_SUBMAR** od każdego innego procesu, następnie na podstawie
 4 listy *queue_submar{my_submarine_id}* sprawdza czy zmieści się na łodzi. Jeśli tak, to prze-
 5 chodzi do stanu **BOARDED**.

6 W przeciwnym razie, inkrementuje licznik niepowodzeń *try_no*. Jeśli nowa wartość licznika nie
 7 przekracza limitu *Max_try_no*, proces usuwa się z kolejki *queue_submar{my_submarine_id}*,
 8 oznacza łódź jako niedostępna w *available_submarine_list*, wysyła do wszystkich pozostałych
 9 procesów wiadomość **FULL_SUBMAR_RETREAT**{*my_submarine_id*}, po czym wraca do
 10 stanu **CHOOSE_SUBMAR**.

11 Jeśli jednak próg licznika został przekroczony, proces wysyła do pozostałych
 12 **FULL_SUBMAR_STAY**{*my_submarine_id*}, po czym zawiesza się, aż otrzyma
 13 **RETURN_SUBMAR**{*my_submarine_id*, ...}. Po przebudzeniu i stosownym obsłużeniu wiado-
 14 mości ponownie sprawdza czy może się zmieścić. Jeśli tak, przechodzi do stanu **BOARDED**,
 15 jeśli nie, czeka ponownie.

16 Reakcja na wiadomości

18 **REQ_PONY** Proces jest w sekcji krytycznej, więc dodaje nadawcę do kolejki *queue_pony* i
 19 nic nie odpowiada.

20 **ACK_PONY** Ignoruje.

21 **REQ_SUBMAR**{*submarine_id*} Dodaje nadawcę do kolejki *queue_submar{id}* powiązanej
 22 ze wskazaną przez parametr *submarine_id* łodzią podwodną i odpowiada **ACK_SUBMAR**.

23 **ACK_SUBMAR** Inkrementuje licznik *received_ack_no*.

24 **FULL_SUBMAR_STAY**{*submarine_id*} Oznacza na liście *available_submarine_list*, że
 25 łódź wskazywana przez parametr *submarine_id* jest niedostępna.

26 **FULL_SUBMAR_RETREAT**{*submarine_id*} Oznacza na liście
 27 *available_submarine_list*, że łódź wskazywana przez parametr *submarine_id* jest niedostępna
 28 oraz usuwa nadawcę z kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez
 29 parametr *submarine_id*.

30 **RETURN_SUBMAR**{*submarine_id*, *passenger_no*} Oznacza na liście *available_submarine_list*,
 31 że łódź wskazywana przez parametr *submarine_id* jest dostępna oraz usuwa z początku
 32 kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez parametr *submarine_id*
 33 *passenger_no* pozycji. Jeśli proces oczekuje na powrót tej łodzi, budzi się i wykonuje kroki
 34 opisane powyżej.

35 **TRAVEL_READY** Kolejkuje odpowiedź. Odpowiedzi udzieli po przejściu do **BOARDED**,
 36 o ile do tego czasu nie zmieni łodzi.

37 **ACK_TRAVEL** Niemożliwe, ignoruje.

38 **DEPART_SUBMAR** Niemożliwe, ignoruje.

39 **DEPART_SUBMAR_NOT_FULL** Niemożliwe, ignoruje.

1 3.7.5 BOARDED

2 Na początku proces zeruje wartość *try_no*. Z tego stanu proces może przejść tylko do **TRAVEL**.
 3 Jednak, moment przjścia jest zależny od tego czy proces został kapitanem łodzi podwodnej, czy
 4 też nie. Kapitanem mianowany jest proces, który znajduje się na pierwszej pozycji w kolejce
 5 powiązanej z łodzią, na której przebywa.

6 **Jest kapitanem**
 7 Oczekuje na wiadomość, że łódź jest pełna: **FULL_SUBMAR_STAY** albo
 8 **FULL_SUBMAR_RETREAT** z parametrem równym *my_submarine_id*. Może się jednak
 9 zdarzyć, że taka wiadomość nigdy nie nadejdzie - dojdzie do zakleszczenia. Kapitan stara
 10 się wykryć taką sytuację poprzez sprawdzanie, po każdej otrzymanej i obsłużonej wiadomości
 11 **REQ_SUBMAR**, czy łączna liczba procesów w kolejkach *queue_submar* jest równa $\min\{Pony_no,$
 12 *Tourist_no\}.
 13 Niezależnie od przyczyny, po wykryciu konieczności wypłynięcia proces, na podstawie właściwiej
 14 *queue_submar* wyznacza listę procesów znajdujących się na łodzi i umieszcza ją w uprzednio
 15 wyczyszczonej zmiennej *boarded_on_my_submarine*. Proces ustawia *received_ack_no* na 1.
 16 Do każdego procesu z tej listy (oprócz siebie) wysyła zapytanie **TRAVEL_READY** i oczekuje
 17 na otrzymanie od każdego z nich **ACK_TRAVEL**. Mając już pewność, że pozostałe procesy
 18 są gotowe do wypłynięcia, ustawia *received_ack_no* na 1, po czym wysyła do tych samych
 19 procesów: jeśli nie wykrył zakleszczenia wiadomość **DEPART_SUBMAR**, w przeciwnym ra-
 20 zie **DEPART_SUBMAR_NOT_FULL**. Ponownie czeka na potwierdzenie **ACK_TRAVEL**, po
 21 czym przechodzi do stanu **TRAVEL**.*

22 **Nie jest kapitanem**
 23 Jeśli w poprzednim stanie otrzymał zapytanie **TRAVEL_READY** i nie zmienił łodzi, odpowiada
 24 **ACK_TRAVEL**, w przeciwnym razie, oczekuje na **TRAVEL_READY**, na który odpowiada
 25 **ACK_TRAVEL**. Następnie oczekuje na **DEPART_SUBMAR** albo **DEPART_SUBMAR_NOT_FULL**.
 26 Ponownie odpowiada **ACK_TRAVEL** i przechodzi do stanu **TRAVEL**.

27 Czasami może się zdarzyć (np. w przypadku wykrycia zakleszczenia, gdyż wtedy łodzie odpły-
 28 wają mając wolne miejsca), że proces znajdzie się w stanie **BOARDED**, gdy wybrana przez
 29 niego łódź jest w podróży. Dlatego, śpiący proces, oprócz obu wersji DEPART_SUBMAR może
 30 być obudzony przez RETURN_SUBMAR{*my_submarine_id*, ...}. Taki proces zachowuje się
 31 tak, jak gdyby dopiero wszedł do stanu **BOARDED**.
 32

33 **Reakcja na wiadomości**

34 **REQ_PONY** Proces jest w sekcji krytycznej, więc dodaje nadawcę do kolejki *queue_pony* i
 35 nic nie odpowiada.

36 **ACK_PONY** Ignoruje.

37 **REQ_SUBMAR{submarine_id}** Dodaje nadawcę do kolejki *queue_submar{id}* powiązanej
 38 ze wskazaną przez parametr *submarine_id* łodzią podwodną i odpowiada ACK_SUBMAR.
 39 Jeśli jest kapitanem, sprawdza czy doszło do zakleszczenia.

40 **ACK_SUBMAR** Niemożliwe, ignoruje.

41 **FULL_SUBMAR_STAY{submarine_id}** Oznacza na liście *available_submarine_list*, że
 42 łódź wskazywana przez parametr *submarine_id* jest niedostępna. Jeśli jest kapitanem,
 43 rozpoczyna procedurę wypłynięcia.

44 **FULL_SUBMAR_RETREAT{submarine_id}** Oznacza na liście
 45 *available_submarine_list*, że łódź wskazywana przez parametr *submarine_id* jest niedostępna
 46 oraz usuwa nadawcę z kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez
 47 parametr *submarine_id*. Jeśli jest kapitanem, rozpoczyna procedurę wypłynięcia oraz
 48 usuwa proces z listy *boarded_on_my_submarine*, dodatkowo przestaje oczekiwać na ACK_TRAVEL
 49 od tego procesu.

50 **RETURN_SUBMAR{submarine_id, passenger_no}** Oznacza na liście *available_submarine_list*,
 51 że łódź wskazywana przez parametr *submarine_id* jest dostępna oraz usuwa z początku
 52 kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez parametr *submarine_id*
 53 *passenger_no* pozycji. Jeśli *submarine_id* = *my_submarine_id*, proces budzi się.

54 **TRAVEL_READY** Jeśli nim jest - niemożliwe, ignoruje, jeśli nim nie jest, odpowiada ACK_TRAVEL.

55 **ACK_TRAVEL** Jeśli jest kapitanem inkrementuje licznik otrzymanych potwierdzeń, jeśli nim
 56 nie jest, niemożliwe, ignoruje.

57 **DEPART_SUBMAR** Jeśli jest kapitanem - niemożliwe, ignoruje. Jeśli nim nie jest, odpowiada
 58 ACK_TRAVEL i przechodzi do stanu TRAVEL.

59 **DEPART_SUBMAR_NOT_FULL** Jeśli jest kapitanem - niemożliwe, ignoruje. Jeśli nim
 60 nie jest, postępuje tak jak w przypadku DEPART_SUBMAR oraz oznacza wszystkie
 61 łodzie, w których kolejce *queue_submar* znajduje się jakiś proces jako niedostępne (lista
 62 *available_submarines*).

1 3.7.6 TRAVEL

2 Ponownie, przetwarzanie jest zależne od tego czy proces jest kapitanem, zawsze jednak przejście
 3 następuje do **ON_SHORE**.
 4 **Jest kapitanem**

5 Czekaj, do czasu, aż uzna, że podróż się skończyła i można dać pozostałym pasażerom sygnał do
6 opuszczenia łodzi.
7 W tym celu, ustawia *received_ack_no* na 1, wysyła do każdego procesu z listy *boarded_on_my_submarine*
8 (oprócz siebie samego) wiadomość RETURN_SUBMAR dołączając do niej id łodzi oraz liczbę
9 procesów na liście *boarded_on_my_submarine*. Usuwa z odpowiedniej kolejki *queue_submar*
10 liczbę procesów równą liczbie pasażerów i czeka, na odpowiedzi ACK_TRAVEL.
11 Następnie wysyła tę samą wiadomość do wszystkich procesów, które nie znajdowały się z nim
12 na pokładzie, tym razem nie oczekując już jednak potwierdzenia. Po czym przechodzi do stanu
13 **ON_SHORE**.
14 **Nie jest kapitanem**
15 Czekaj na wiadomość RETURN_SUBMAR od kapitana, usuwa wskazaną liczbę procesów z kole-
16 jki, odpowiada ACK_TRAVEL i przechodzi do stanu **ON_SHORE**.
17
18 **Reakcja na wiadomości**
19 **REQ_PONY** Proces jest w sekcji krytycznej, więc dodaje nadawcę do kolejki *queue_pony* i
20 nic nie odpowiada.
21 **ACK_PONY** Ignoruje.
22 **REQ_SUBMAR{submarine_id}** Dodaje nadawcę do kolejki *queue_submar{id}* powiązanej
23 ze wskazaną przez parametr *submarine_id* łodzią podwodną i odpowiada ACK_SUBMAR.
24 **ACK_SUBMAR** Niemożliwe, ignoruje.
25 **FULL_SUBMAR_STAY{submarine_id}** Oznacza na liście *available_submarine_list*, że
26 łódź wskazywana przez parametr *submarine_id* jest niedostępna.
27 **FULL_SUBMAR_RETREAT{submarine_id}** Oznacza na liście
28 *available_submarine_list*, że łódź wskazywana przez parametr *submarine_id* jest niedostępna
29 oraz usuwa nadawcę z kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez
30 parametr *submarine_id*.
31 **RETURN_SUBMAR{submarine_id, passenger_no}** Jeśli wiadomość dotyczy łodzi, na
32 której się znajduje postępuje według kroków opisanych powyżej. W przeciwnym razie,
33 oznacza na liście *available_submarine_list*, że łódź wskazywana przez parametr *subma-*
34 *rine_id* jest dostępna oraz usuwa z początku kolejki *queue_submar{id}* powiązanej z łodzią
35 wskazywaną przez parametr *submarine_id passenger_no* pozycji.
36 **TRAVEL_READY** Niemożliwe, ignoruje.
37 **ACK_TRAVEL** Jeśli jest kapitanem inkrementuje licznik otrzymanych potwierdzeń, jeśli nim
38 nie jest, niemożliwe, ignoruje.
39 **DEPART_SUBMAR** Niemożliwe, ignoruje.
40 **DEPART_SUBMAR_NOT_FULL** Niemożliwe, ignoruje.

1 3.8 ON_SHORE

2 Przechodzi do stanu **RESTING** po wysłaniu ACK_PONY do wszystkich procesów z kolejki
3 *queue_pony* czyszcząc tę listę.
4
5 **Reakcja na wiadomości**

- 6 **REQ_PONY** Odpowiada ACK_PONY.
- 7 **ACK_PONY** Ignoruje.
- 8 **REQ_SUBMAR**{*submarine_id*} Dodaje nadawcę do kolejki *queue_submar{id}* powiązanej
9 ze wskazaną przez parametr *submarine_id* łodzią podwodną i odpowiada ACK_SUBMAR.
- 10 **ACK_SUBMAR** Niemożliwe, ignoruje.
- 11 **FULL_SUBMAR_STAY**{*submarine_id*} Oznacza na liście *available_submarine_list*, że
12 łódź wskazywana przez parametr *submarine_id* jest niedostępna.
- 13 **FULL_SUBMAR_RETREAT**{*submarine_id*} Oznacza na liście
14 *available_submarine_list*, że łódź wskazywana przez parametr *submarine_id* jest niedostępna
15 oraz usuwa nadawcę z kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez
16 parametr *submarine_id*.
- 17 **RETURN_SUBMAR**{*submarine_id*, *passenger_no*} Oznacza na liście *available_submarine_list*,
18 że łódź wskazywana przez parametr *submarine_id* jest dostępna oraz usuwa z początku
19 kolejki *queue_submar{id}* powiązanej z łodzią wskazywaną przez parametr *submarine_id*
20 *passenger_no* pozycji.
- 21 **TRAVEL_READY** Niemożliwe, ignoruje.
- 22 **ACK_TRAVEL** Niemożliwe, ignoruje.
- 23 **DEPART_SUBMAR** Niemożliwe, ignoruje.
- 24 **DEPART_SUBMAR_NOT_FULL** Niemożliwe, ignoruje.