

Nanozombie - algorytm

Marcin Pastwa 136779, Piotr Tomaszewski 136821

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 Kolejka nie zawiera duplikatów - proces wpisany do niej kilkukrotnie
- 8 będzie widniał na niej tylko raz.
- 9 Po wyjściu z sekcji krytycznej (po zwrocie stroju) do wszystkich
- 10 procesów z kolejki wysyłane jest ACK_PONY, kolejka jest
- 11 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 danej
- 3 łodzi.
- 4 Wartość ujęta w nawiasy klamrowe oznacza, że mówimy o kolejce
- 5 powiązanej z łodzią o danym identyfikatorze.
- 6 Dzięki kolejce można wyznaczyć, które procesy mogą zająć miejsce
- 7 na danej łodzi podwodnej (znaleźć się w sekcji krytycznej).
- 8 Kolejka wypełniana jest zgodnie z algorytmem Lamporta, z tą
- 9 różnicą, że w sekcji krytycznej w danym momencie może znajdować
- 10 się >1 proces. Dokładniej, proces, który otrzymał wszystkie
- 11 potwierdzenia, może zająć miejsce (wejść do sekcji krytycznej), jeśli
- 12 suma rozmiaru jego i procesów, które są przed nim w kolejce nie
- 13 przekracza 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, i nie
- 3 jest pełna, 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

- 1 `PONY_NO` Łączna liczba dostępnych strojów kucyka (stała).
- 2 `TOURIST_NO` Łączna liczba turystów (stała).
- 3 `SUBMAR_NO` Łączna liczba łodzi podwodnych (stała).
- 4 `REC_ACK_NO` Zmienna przechowująca liczbę zebranych ACK.
- 5 Jest zerowana w momencie wysłania żądania, które
- 6 wymaga zebrania potwierdzeń.

Zegar Lamporta (Timestamp)

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

Wiadomości

- 1 **W nawiasach klamrowych znajdują się parametry dołączane**
- 2 **do wiadomości.**
- 3 **REQ_PONY** żądanie dostępu do sekcji krytycznej (żądanie
- 4 dostępu do stroju kucyka).
- 5 **ACK_PONY** zgoda na dostęp do stroju kucyka.
- 6 **REQ_SUBMAR{id_łodzi}** żądanie dostępu do sekcji krytycznej
- 7 (do miejsca na łodzi podwodnej).
- 8 **ACK_SUBMAR** potwierdzenie wpisania nadawcy do odpowiedniej
- 9 kolejki **QUEUE_SUBMAR**.
- 10 **FULL_SUBMAR_RETREAT{id_łodzi}** wysyłane przez proces,
- 11 któremu nie udało się wsiąść do łodzi, więc się z niej
- 12 wycofuje.

Wiadomości c.d.

- 1 `FULL_SUBMAR_STAY{id_łodzi}` wysyłane przez proces,
- 2 któremu nie udało się wsiąść do łodzi, ale poddaje się
- 3 i zostaje w kolejce.
- 4 `RETURN_SUBMAR{id_łodzi, liczba_pasażerów}` informacja, że
- 5 łódź o podanym id i wioząca podaną liczbę pasażerów
- 6 wróciła do portu. Dla procesów, które płynęły tą
- 7 łodzią oznacza to rozkaz zwolnienia zasobów
- 8 (opuszczenia jej), dla ogółu procesów, że można
- 9 usunąć z kolejki podaną drugim parametrem liczbę
- 10 procesów.
- 11 `TRAVEL_READY` weryfikacja czy można rozpocząć podróż łodzi
- 12 podwodnej.
- 13 `ACK_TRAVEL` odpowiedź na zapytania kapitana.
- 14 `DEPART_SUBMAR` informacja, że łódź, w której dany proces
- 15 przebywa właśnie wypływa z portu.

Stany

- 1 **RESTING** Jest to stan początkowy. Symuluje odpoczynek
- 2 turysty między zakończeniem jednej wycieczki, a
- 3 rozpoczęciem kolejnej.
- 4 **WAIT_PONY** Proces ubiega się o dostęp do sekcji krytycznej - o
- 5 pobranie stroju kucyka.
- 6 **CHOOSE_SUBMAR** Proces ma już strój i wybiera sobie łódź.
- 7 **WAIT_SUBMAR** Proces ubiega się o dostęp do sekcji krytycznej -
- 8 o miejsce na łodzi podwodnej.
- 9 **BOARDED** Proces zajął miejsce na łodzi podwodnej i czeka na
- 10 rozpoczęcie wyprawy.
- 11 **TRAVEL** Proces podróżuje łodzią podwodną.
- 12 **ON_SHORE** Proes zakończył podróż, wysiadł z łodzi podwodnej
- 13 (zwolnił zasób) i zamierza oddać (zwolnić) strój
- 14 kucyka.

RESTING

- 1 Proces przebywa w stanie RESTING do czasu, aż podejmie decyzję
- 2 o rozpoczęciu wyprawy. W tym celu potrzebuje stroju kucyka,
- 3 zaczyna więc się o niego ubiegać.

- 4 Proces wysyła do wszystkich pozostałych procesów żądanie
- 5 REQ_PONY, po czym przechodzi do stanu WAIT_PONY.

RESTING - odpowiedzi

- 1 `REQ_PONY` odpowiada `ACK_PONY`.
- 2 `ACK_PONY` niemożliwe, ignoruje.
- 3 `REQ_SUBMAR{id_łodzi}` dodaje nadawcę do kolejki powiązanej
4 z tą łodzią (`QUEUE_SUBMAR`) i odpowiada
5 `ACK_SUBMAR`.
- 6 `ACK_SUBMAR` niemożliwe, ignoruje.
- 7 `FULL_SUBMAR_STAY{id_łodzi}` oznacza na liście
8 `LIST_SUBMAR`, że dana łódź jest zajęta.
- 9 `FULL_SUBMAR_RETREAT{id_łodzi}` oznacza na liście
10 `LIST_SUBMAR`, że dana łódź jest zajęta oraz usuwa
11 nadawcę z kolejki powiązanej z daną łodzią
12 `QUEUE_SUBMAR{id_łodzi}`.

RESTING - odpowiedzi c.d.

- 1 RETURN_SUBMAR{id_łodzi, liczba_pasażerów} oznacza łódź
- 2 jako dostępną (LIST_SUBMAR) oraz usuwa pierwsze
- 3 liczba_pasażerów pozycji z kolejki
- 4 QUEUE_SUBMAR{id_łodzi}.
- 5 TRAVEL_READY niemożliwe, ignoruje.
- 6 ACK_TRAVEL niemożliwe, ignoruje.
- 7 DEPART_SUBMAR niemożliwe, ignoruje.

WAIT_PONY

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

- 3 Ubieganie się o dostęp do sekcji krytycznej bazuje na algorytmie
- 4 Ricarta - Agravali. Różnica polega na poszerzeniu sekcji krytycznej
- 5 do liczby procesów równej liczbie strojów kucyka (PONY_NO).
- 6 Zatem, proces może wejść do sekcji krytycznej (zabrać strój) po
- 7 zebraniu co najmniej TOURIST_NO - PONY_NO odpowiedzi
- 8 ACK_PONY (różnica między liczbą procesów, a liczbą strojów).
- 9 Zakładamy, że proces ma od razu jedno pozwolenie - swoje własne.
- 10 Po zebraniu minimalnej liczby pozwoleń proces zajmuje zasoby
- 11 (strój) i przechodzi do stanu GOT_PONY.

WAIT_PONY - odpowiedzi

- 1 **REQ_PONY** jeśli otrzymana wiadomość ma niższy priorytet
- 2 dodaje nadawcę do kolejki QUEUE_PONY i nic nie
- 3 odpowiada. W przeciwnym razie, uznaje
- 4 pierwszeństwo i odpowiada ACK_PONY.
- 5 **ACK_PONY** inkrementuje REC_ACK_NO, po zdobyciu
- 6 wymaganej liczby przechodzi do stanu GOT_PONY.
- 7 **REQ_SUBMAR{id_łodzi}** dodaje nadawcę do kolejki powiązanej
- 8 z tą łodzią (QUEUE_SUBMAR) i odpowiada
- 9 ACK_SUBMAR.
- 10 **ACK_SUBMAR** niemożliwe, ignoruje.
- 11 **FULL_SUBMAR_STAY{id_łodzi}** oznacza na liście
- 12 LIST_SUBMAR, że dana łódź jest zajęta.

WAIT_PONY - odpowiedzi c.d.

- 1 `FULL_SUBMAR_RETREAT{id_łodzi}` oznacza na liście
- 2 `LIST_SUBMAR`, że dana łódź jest zajęta oraz usuwa
- 3 nadawcę z kolejki powiązanej z daną łodzią
- 4 (`QUEUE_SUBMAR{id_łodzi}`).
- 5 `RETURN_SUBMAR{id_łodzi, liczba_pasażerów}` oznacza łódź
- 6 jako dostępną (`LIST_SUBMAR`) oraz usuwa pierwsze
- 7 `liczba_pasażerów` pozycji z kolejki
- 8 `QUEUE_SUBMAR{id_łodzi}`.
- 9 `TRAVEL_READY` niemożliwe, ignoruje.
- 10 `ACK_TRAVEL` niemożliwe, ignoruje.
- 11 `DEPART_SUBMAR` niemożliwe, ignoruje.

CHOOSE_SUBMAR

- 1 Proces znajduje się w tym stanie, gdy ma już strój kuczka, ale nie
- 2 wybrał sobie jeszcze łodzi.
- 3 Proces wybiera łódź podwodną. Będzie starał się wybrać taką,
- 4 która według jego obecnej wiedzy jest w najmniejszym stopniu
- 5 zajęta. Ponieważ wyznaczanie zajętości wymaga kilku obliczeń,
- 6 proces ogranicza sobie zbiór kandydatów do tych, które, (również
- 7 według obecnej, niekoniecznie aktualnej wiedzy), znajdują się w
- 8 porcie i mają jeszcze wolne miejsca (lista LIST_SUBMAR). Jeśli
- 9 takich łodzi nie ma, czeka na sygnał RETURN_SUBMAR.
- 10 Po wybraniu łodzi podwodnej proces wysyła do pozostałych
- 11 wiadomość REQ_SUBMAR{id_wybranej_łodzi}, dodaje się do
- 12 kolejki QUEUE_SUBMAR{id_wybranej_łodzi} i przechodzi do
- 13 stanu WAIT_SUBMAR.

CHOOSE_SUBMAR - odpowiedzi

- 1 **REQ_PONY** proces jest w sekcji krytycznej, więc dodaje nadawcę
- 2 do kolejki **QUEUE_PONY** i nic nie odpowiada.
- 3 **ACK_PONY** jest już w sekcji krytycznej, więc ignoruje.
- 4 **REQ_SUBMAR{id_łodzi}** dodaje nadawcę do kolejki powiązanej
- 5 z tą łodzią (**QUEUE_SUBMAR**) i odpowiada
- 6 **ACK_SUBMAR**.
- 7 **ACK_SUBMAR** niemożliwe, ignoruje.
- 8 **FULL_SUBMAR_STAY{id_łodzi}** oznacza na liście
- 9 **LIST_SUBMAR**, że dana łódź jest zajęta.

CHOOSE_SUBMAR - odpowiedzi c.d.

- 1 `FULL_SUBMAR_RETREAT{id_łodzi}` oznacza na liście
- 2 `LIST_SUBMAR`, że dana łódź jest zajęta oraz usuwa
- 3 nadawcę z kolejki powiązanej z daną łodzią
- 4 (`QUEUE_SUBMAR{id_łodzi}`).
- 5 `RETURN_SUBMAR{id_łodzi, liczba_pasażerów}` oznacza łódź
- 6 jako dostępną (`LIST_SUBMAR`) oraz usuwa pierwsze
- 7 `liczba_pasażerów` pozycji z kolejki
- 8 `QUEUE_SUBMAR{id_łodzi}`. Jeśli zawiesił się, bo
- 9 nie znalazł łodzi, to ta wiadomość go budzi.
- 10 `TRAVEL_READY` niemożliwe, ignoruje.
- 11 `ACK_TRAVEL` niemożliwe, ignoruje.
- 12 `DEPART_SUBMAR` niemożliwe, ignoruje.

Uwaga do algorytmu

- 1 Uwaga, w dalszej części algorytmu potrzebny będzie proces, który
- 2 wyda sygnał do odpłynięcia i potem powrotu do portu łodzi
- 3 podwodnej - będzie pełnił rolę kapitana. Turyści znajdujący się na
- 4 łodzi mogliby, co prawda, wspólnie podejmować te decyzje. Jednak,
- 5 wprowadziłoby to tylko konieczność przesłania dodatkowych
- 6 wiadomości, bez poprawy współbieżności działania - procesy w tym
- 7 czasie nic nie robią, czekają jedynie na zakończenie podróży.
- 8 Dlatego, postanowiliśmy ponownie wykorzystać już zbudowaną
- 9 kolejkę `QUEUE_SUBMAR{id_łodzi}`. Sygnał do odpłynięcia i
- 10 powrotu wyda zatem proces, który zdobyłby dostęp do sekcji
- 11 krytycznej w algorytmie Lamporta - pierwszy w kolejce.

WAIT_SUBMAR

- 1 Proces w tym stanie oczekuje na dostęp do kolejnej sekcji
- 2 krytycznej - zajęcie miejsca na wybranej łodzi podwodnej.
- 3 Tym razem, ubieganie się o dostęp do sekcji krytycznej bazuje na
- 4 algorytmie Lamporta. Proces czeka, aż otrzyma potwierdzenia od
- 5 wszystkich pozostałych procesów.
- 6 Następnie, sprawdza czy może zająć miejsce na łodzi. Zająć
- 7 miejsce na łodzi, może, jeśli suma rozmiarów jego i procesów
- 8 znajdujących się przed nim w kolejce nie przekracza maksymalnej
- 9 pojemności łodzi. Jeśli może, to zajmuje miejsca i przechodzi do
- 10 stanu BOARDED.

WAIT_SUBMAR - c.d.

- 1 Jeśli jednak się nie zmieści, inkrementuje licznik niepowodzeń
- 2 (TRY_NO), oznacza łódź jako niedostępną i sprawdza, czy nie
- 3 przekroczył limitu (MAX_TRY_NO). Jeśli mieści się w limicie,
- 4 usuwa się z kolejki, wysyła do pozostałych wiadomość
- 5 FULL_SUBMAR_RETREAT{id_łodzi} i wraca do stanu
- 6 CHOOSE_SUBMAR. Jeśli jednak przekroczył limit, poddaje się i
- 7 postanawia czekać, aż łódź wróci z wyprawy. Wysyła więc do
- 8 pozostałych wiadomość FULL_SUBMAR_STAY{id_łodzi} i
- 9 zawiesza się, do czasu, aż łódź wróci.

WAIT_SUBMAR - odpowiedzi

- 1 **REQ_PONY** proces jest w sekcji krytycznej, więc dodaje nadawcę
- 2 do kolejki **QUEUE_PONY** i nic nie odpowiada.
- 3 **ACK_PONY** jest już w sekcji krytycznej, więc ignoruje.
- 4 **REQ_SUBMAR{id_łodzi}** dodaje nadawcę do kolejki powiązanej
- 5 z tą łodzią (**QUEUE_SUBMAR**) i odpowiada
- 6 **ACK_SUBMAR**.
- 7 **ACK_SUBMAR** inkrementuje licznik zebranych potwierdzeń. Po
- 8 zebraniu wszystkich sprawdza, czy może wejść do
- 9 łodzi.
- 10 **FULL_SUBMAR_STAY{id_łodzi}** oznacza na liście
- 11 **LIST_SUBMAR**, że dana łódź jest zajęta.

WAIT_SUBMAR - odpowiedzi c.d.

- 1 `FULL_SUBMAR_RETREAT{id_łodzi}` oznacza na liście
- 2 `LIST_SUBMAR`, że dana łódź jest zajęta oraz usuwa
- 3 nadawcę z kolejki powiązanej z daną łodzią
- 4 (`QUEUE_SUBMAR{id_łodzi}`).
- 5 `RETURN_SUBMAR{id_łodzi, liczba_pasażerów}` oznacza łódź
- 6 jako dostępną (`LIST_SUBMAR`) oraz usuwa pierwsze
- 7 `liczba_pasażerów` pozycji z kolejki
- 8 `QUEUE_SUBMAR{id_łodzi}`.
- 9 `TRAVEL_READY` kolejkuje odpowiedź. Odpowiedzi udzieli po
- 10 przejściu do stanu `STATE_BOARDED`, o ile nie
- 11 zmieni łodzi.
- 12 `ACK_TRAVEL` niemożliwe, ignoruje.
- 13 `DEPART_SUBMAR` niemożliwe, ignoruje.

BOARDED

- 1 Proces w tym stanie zajął miejsca w łodzi podwodnej i czeka na
- 2 rozpoczęcie wyprawy.
- 3 Jeśli proces został kapitanem czeka na otrzymanie wiadomości, że
- 4 łódź jest pełna. Po otrzymaniu tej informacji, wyznacza na
- 5 podstawie kolejki, które procesy z nim płyną i wysyła do każdego z
- 6 nich zapytanie TRAVEL_READY (ma ono na celu zweryfikowanie
- 7 czy wszyscy już wsiedli), po czym czeka, aż każdy z nich mu
- 8 odpowie ACK_TRAVEL. Po otrzymaniu zgód wysyła do nich
- 9 wiadomość DEPART_SUBMAR, informując je, że wszyscy są
- 10 gotowi i można wyruszać. Ponownie czeka na potwierdzenie
- 11 ACK_TRAVEL. Po ich otrzymaniu przechodzi do stanu TRAVEL.

BOARDED - c.d.

- 1 Pozostałe procesy czekają na pytanie TRAVEL_READY.
- 2 Odpowiadają na nie ACK_TRAVEL. Jeżeli to zapytanie uzyskał
- 3 jeszcze w poprzednim stanie i nie zmienił w tym czasie łodzi, to
- 4 teraz na nie odpowiada.
- 5 Następnie proces czeka na DEPART_SUBMAR. Po otrzymaniu
- 6 wiadomości odpowiada ACK_TRAVEL i przechodzi do stanu
- 7 TRAVEL.

BOARDED - odpowiedzi

- 1 **REQ_PONY** proces jest w sekcji krytycznej, więc dodaje nadawcę
- 2 do kolejki **QUEUE_PONY** i nic nie odpowiada.
- 3 **ACK_PONY** jest już w sekcji krytycznej, więc ignoruje.
- 4 **REQ_SUBMAR{id_łodzi}** dodaje nadawcę do kolejki powiązanej
- 5 z tą łodzią (**QUEUE_SUBMAR**) i odpowiada
- 6 **ACK_SUBMAR**.
- 7 **ACK_SUBMAR** niemożliwe, ignoruje.
- 8 **FULL_SUBMAR_STAY{id_łodzi}** oznacza na liście
- 9 **LIST_SUBMAR**, że dana łódź jest zajęta. Jeśli jest
- 10 kapitanem tej łodzi rozpoczyna procedurę
- 11 wypłynięcia.

BOARDED - odpowiedzi c.d.

- 1 `FULL_SUBMAR_RETREAT{id_łodzi}` oznacza na liście
- 2 `LIST_SUBMAR`, że dana łódź jest zajęta oraz usuwa
- 3 nadawcę z kolejki powiązanej z daną łodzią
- 4 (`QUEUE_SUBMAR{id_łodzi}`). Jeśli jest kapitanem
- 5 tej łodzi rozpoczyna procedurę wypłynięcia.
- 6 `RETURN_SUBMAR{id_łodzi, liczba_pasażerów}` oznacza łódź
- 7 jako dostępną (`LIST_SUBMAR`) oraz usuwa pierwsze
- 8 `liczba_pasażerów` pozycji z kolejki
- 9 `QUEUE_SUBMAR{id_łodzi}`.
- 10 `TRAVEL_READY` odpowiada `ACK_TRAVEL`.
- 11 `ACK_TRAVEL` jeśli jest kapitanem: postępuje według kroków
- 12 opisanych wcześniej. Jeśli nie jest kapitanem,
- 13 wiadomość niemożliwa, ignoruje.
- 14 `DEPART_SUBMAR` Odpowiada `ACK_TRAVEL` i przechodzi do
- 15 stanu `TRAVEL`.

TRAVEL

- 1 Ten stan symuluje podróż i zwiedzanie wyspy przez turystów.
- 2 Jeśli proces został kapitanem czeka przez pewną, losową ilość
- 3 czasu. Po tym czasie, uznajemy, że łódź zakończyła podróż i
- 4 wróciła do brzegu. Chcemy, aby turyści, którzy przebywają w łodzi
- 5 mogli z niej wysiąść, nim wsiądą do niej kolejni. Dlatego, kapitan
- 6 najpierw wysyła wiadomość `RETURN_SUBMAR{id_łodzi,`
- 7 `liczba_pasażerów}` do procesów, które z nim płynęły. (Liczba
- 8 pasażerów jest parametrem, ponieważ wyznaczanie, kto płynął
- 9 łodzią wymaga obliczeń. Dzięki dodaniu tego parametru pozostałe
- 10 procesy nie muszą już same tego wyznaczać).
- 11 Kapitan zwalnia zasoby (miejsce na łodzi), usuwa z kolejki tyle
- 12 procesów, ile było pasażerów i czeka, aż pozostali pasażerowie
- 13 udzielą odpowiedzi `ACK_TRAVEL`.

TRAVEL - c.d.

- 1 Po otrzymaniu potwierdzeń od współpasażerów kapitan wysyła
- 2 `RETURN_SUBMAR{id_łodzi, liczba_pasażerów}` do pozostałych
- 3 procesów (tym razem nie oczekując już potwierdzenia). Po czym
- 4 przechodzi do stanu `ON_SHORE`.

- 5 Pozostałe procesy czekają na zakończenie podróży. Po otrzymaniu
- 6 wiadomości `RETURN_SUBMAR` zwalniają swoje miejsca, usuwają
- 7 z kolejki odpowiednią liczbę procesów, odpowiadają `ACK_TRAVEL`
- 8 i przechodzą do stanu `ON_SHORE`.

TRAVEL - odpowiedzi

- 1 `REQ_PONY` proces jest w sekcji krytycznej, więc dodaje nadawcę
- 2 do kolejki `QUEUE_PONY` i nic nie odpowiada.
- 3 `ACK_PONY` jest już w sekcji krytycznej, więc ignoruje.
- 4 `REQ_SUBMAR{id_łodzi}` dodaje nadawcę do kolejki powiązanej
- 5 z tą łodzią (`QUEUE_SUBMAR`) i odpowiada
- 6 `ACK_SUBMAR`.
- 7 `ACK_SUBMAR` niemożliwe, ignoruje.
- 8 `FULL_SUBMAR_STAY{id_łodzi}` oznacza na liście
- 9 `LIST_SUBMAR`, że dana łódź jest zajęta.

TRAVEL - odpowiedzi c.d.

- 1 `FULL_SUBMAR_RETREAT{id_łodzi}` oznacza na liście
- 2 `LIST_SUBMAR`, że dana łódź jest zajęta oraz usuwa
- 3 nadawcę z kolejki powiązanej z daną łodzią
- 4 (`QUEUE_SUBMAR{id_łodzi}`).
- 5 `RETURN_SUBMAR{id_łodzi, liczba_pasażerów}` oznacza łódź
- 6 jako dostępną (`LIST_SUBMAR`) oraz usuwa pierwsze
- 7 `liczba_pasażerów` pozycji z kolejki
- 8 `QUEUE_SUBMAR{id_łodzi}`. Jeśli płynie tą łodzią,
- 9 to dodatkowo zwalnia miejsce na łodzi, odpowiada
- 10 `ACK_TRAVEL` i przechodzi do stanu `ON_SHORE`.
- 11 `TRAVEL_READY` niemożliwe, ignoruje.
- 12 `ACK_TRAVEL` jeśli jest kapitanem: postępuje według kroków
- 13 opisanych wcześniej. Jeśli nie jest kapitanem,
- 14 wiadomość niemożliwa, ignoruje.
- 15 `DEPART_SUBMAR` niemożliwe, ignoruje.

ON_SHORE

- 1 Proces w tym stanie zakończył właśnie podróż, zwolnił jedną sekcję
- 2 krytyczną (łódź) i zamierza zwolnić kolejną - oddać strój kucyka.
- 3 Proces zwalnia strój kucyka, wysyła ACK_PONY do wszystkich
- 4 procesów z QUEUE_PONY oraz czyści tę listę. Następnie
- 5 przechodzi do stanu RESTING.

ON_SHORE - odpowiedzi

- 1 REQ_PONY odpowiada ACK_PONY.
- 2 ACK_PONY jest jeszcze w sekcji krytycznej, więc ignoruje.
- 3 REQ_SUBMAR{id_łodzi} dodaje nadawcę do kolejki powiązanej
4 z tą łodzią (QUEUE_SUBMAR) i odpowiada
5 ACK_SUBMAR.
- 6 ACK_SUBMAR niemożliwe, ignoruje.
- 7 FULL_SUBMAR_STAY{id_łodzi} oznacza na liście
8 LIST_SUBMAR, że dana łódź jest zajęta.

ON_SHORE - odpowiedzi c.d.

- 1 `FULL_SUBMAR_RETREAT{id_łodzi}` oznacza na liście
- 2 `LIST_SUBMAR`, że dana łódź jest zajęta oraz usuwa
- 3 nadawcę z kolejki powiązanej z daną łodzią
- 4 (`QUEUE_SUBMAR{id_łodzi}`).
- 5 `RETURN_SUBMAR{id_łodzi, liczba_pasażerów}` oznacza łódź
- 6 jako dostępną (`LIST_SUBMAR`) oraz usuwa pierwsze
- 7 `liczba_pasażerów` pozycji z kolejki
- 8 `QUEUE_SUBMAR{id_łodzi}`.
- 9 `TRAVEL_READY` niemożliwe, ignoruje.
- 10 `ACK_TRAVEL` niemożliwe, ignoruje.
- 11 `DEPART_SUBMAR` niemożliwe, ignoruje.