Przetwarzanie Rozproszone
Nanozombie - algorytm

Krzysztof Sychla 136807, Eryk Szpotański 136811

1 Założenia

- Procesy połączone są każdy z każdym
- Kanały są FIFO
- liczba strojów kucyków oraz pojemność w każdej łodzi podwodnej są znane każdemu procesowi
- każdy proces zna przedział pojemności jakie może zajmować turysta
- średnia pojemność zajmowana przez turystę * liczba strojów kucyka > suma pojemności wszystkich łodzi podwodnych

Pojemność kanałów: 4

2 Przebieg algorytmu

I Proces inicjalizuje zmienne i ustawia wartość zegara Lamporta na 0

II Proces czeka losową ilość czasu

III Proces ubiega się o strój kucyka, zgodnie z poniżej opisanym algorytmem

IV Po uzyskaniu stroju, proces ubiega się o dostęp do losowej łodzi, również zgodnie z opisanym algorytmem

V Po uzyskaniu odpowiedniej pojemności w łodzi, proces czeka do czasu jej zapełnienia

VI Łódź pływa przez losowy czas, po którym proces zwalnia uzyskane wcześniej zasoby i wraca do punktu II

3 Uzyskiwanie jednego z zasobów

3.1 Proces chcący uzyskać zasób

I Proces chcący uzyskać dany zasób ustawia odpowiednią zmienną (np. dla stroju kucyka zmienną boolowską na true lub dla jednej z łodzi podwodnych odpowiednie pole w tablicy booli, przy czym maksymalnie jedna wartość na raz powinna być ustawiona na true) dalej nazywaną *desire*.

II Następnie powiększa o 1 i zapamiętuje wartość zegara Lamporta dalej nazywaną *time*.

III Następnie rozsyła do wszystkich pozostałych procesów wiadomość *REQ* wraz z *time* i znacznikiem wskazującym który zasób chce zająć, nazywanym *resource* (+ oczywiście wartość zegara Lamporta) i czeka na odpowiedzi od wszystkich procesów, oczywiście w tym czasie powinien, a w ręcz musi też odpowiadać na wiadomości innych procesów.

IV Jeśli którąkolwiek z odpowiedzi będzie *DEN* proces musi cofnąć się do poprzedniego podpunktu i rozpocząć rozsyłanie od nowa (z tą samą wartością *time*, doprecyzowując, po otrzymaniu wiadomości stan zegaru Lamporta jest aktualizowany zgodnie z jego zasadą działania, a jedynie ponowana wiadomośc wysyłana jest ze starą wartością).

V W przeciwnym przypadku sumuje wartości z otrzymanych wiadomości *REP* i jeśli otrzymana suma + 1 (w przypadku stroju) lub pojemność turysty (w przypadku łodzi) jest mniejsza lub równa od liczby zasobów, proces zabiera ów zasób (np. zmienia wartość na 1 dla zmiennej oznaczającej ilość posiadanych strojów, albo na odpowiednią liczbę zajmowanych miejsc w tabeli (niekoniecznie tabela, mogą być dwa inty jeden do numeru łodzi, a drugi do miejsca)).

VI Jeśli otrzymana suma nie będzie mniejsza od ilości zasobów to w przypadku strojów proces czeka losowy czas i próbuje ponownie zaczynając od rozsyłania wiadomości do wszystkich procesów, a w przypadku łodzi próbuje uzyskać miejsce w innym transporcie, również zaczynając od rozsyłania wiadomości. Także w obu przypadkach *time* pozostaje takie same.

3.2 Proces odbierający nie żądający zasobu

- Gdy otrzyma wiadomość *REQ* proces sprawdza na podstawie *resource* czy i ile danego zasobu zajmuje i odsyła wiadomość *RES* z odczytaną wartością.

3.3 Proces odbierający żądający zasobu

- Jeśli proces żąda innego zasobu niż, ten który wysłał odebraną wiadomość, proces postępuje jak w przypadku "odbierający nie żądający zasobu" (3.2).

- Jeśli żądany jest ten sam zasób, proces sprawdza wartość *time* z otrzymanej wiadomości *REQ* i porównuje ją z własną wartością *time* (gdy są równe porównywane są id procesów (nie uwzględniałem wysyłania ich w wiadomości, bo w mpi mamy informację od którego procesu otrzymaliśmy wiadomość, chyba).

- Jeśli otrzymane w wiadomości *time* (lub id w przypadku równości) jest mniejsze to proces odsyła wiadomość *RES* z wartością zajmowanego zasobu (czyli 0, a przynajmniej powinno być 0), a sam proces po odebraniu wszystkich odpowiedzi musi rozpocząć na nowo od rozsyłania *REQ* do wszystkich procesów (z tą samą wartością *time*).

- Jeśli otrzymane *time* (lub id w przypadku równości) jest większe to odsyłana zostaje wiadomość *DEN*.

3.4 Uznanie statku za pełen

Gdy proces zabiera zasób, który jest łodzią podwodną i wyliczona suma powiększona o pojemność turysty jest w przedziale (pojemność danej łodzi - maksymalna pojemność jaką może zajmować turysta, pojemność danej łodzi), to wysyłane są wiadomości *TIC* do wszystkich procesów z informacją ile miejsca pozostało w łodzi oraz z id łodzi (lub innym oznaczeniem jednoznacznie ją określającym), gdy powyższa suma jest równa pojemności łodzi, proces od razu przechodzi do rozsyłania wiadomości *OUT* (opisanych poniżej). Jeśli proces, który odebrał taką wiadomość:

nie jest w trakcie żądania zasobu i nie ma stroju:
odsyła wiadomość *DEN*

jest w trakcie żądania dostępu do stroju:
odsyła wiadomość *DEN*

jest w trakcie żądania dostępu do innej łodzi:
czeka na odebranie wszystkich wiadomości (odpowiedzi na wysłane *REQ*), a następnie jeśli otrzymał dostęp, odsyła wiadomość *DEN*, w przeciwnym przypadku odsyła wiadomość *ACK* i zaczyna ubieganie się o dostęp do łodzi z wiadomości *TIC*

jest w trakcie żądania dostępu do tej samej łodzi:
odsyła wiadomość *ACK*

Gdy proces, który rozesłał wiadomości *TIC* otrzyma wszystkie wiadomości zwrotne, sprawdza, czy którakolwiek z nich była *ACK*, jeśli tak nie robi nic. W przeciwnym przypadku rozsyła wiadomość *OUT* wraz z id łodzi oraz wylosowanym czasem podróży (po której procesy w tej łodzi muszą zwolnić zasoby), po czym sam czeka określoną ilość czasu i zwalnia zasoby.

4 Złożoność

n - liczba procesów
max_t - maksymalna pojemność jaką może zajmować turysta Dla uzyskania dostępu dla zasobu złożoność

komunikacyjna: optymistyczna: $2(n-1)$, pesymistyczna $2n(n-1)$

czasowa: optymistyczna: 2, pesymistyczna $2n$, dla strojów kucyka powiększona o losowy czas, który proces czeka jeśli wszystkie stroje są zajęte (zależy on od liczby wszystkich strojów i turystów)

Dla uznania statku za pełen:

komunikacyjna: optymistyczna: 0, pesymistyczna $2n \cdot max_t$

czasowa: optymistyczna: 0, pesymistyczna $4max_t$