Programowanie Rozproszone
Projekt – Obsługa Pyrkonu

Dariusz Max Adamski 136674, Sławomir Gilewski 142192

{dariusz.adamski,slawomir.gilewski}@student.put.poznan.pl

Data oddania: 12 czerwca 2020

1 Struktury danych

1.1 Wiadomość

Wiadomość składa się z zegara, numeru kolejki i jednej z poniższych etykiet:

1. REQ – Wysyłana z wiadomościami z ubieganiem się o zasób
2. OK – Wysyłana jako odpowiedź na ubieganie się o zasób
3. END – Wysyłana w celu zasygnalizowania wyjścia z wydarzenia (numer kolejki ignorowany)

1.2 Stałe

1. num-tickets – liczba biletów na Pyrkon
2. num-workshops – liczba warsztatów na Pyrkonie
3. max-places – liczba miejsc w warsztacie
4. max-workshops – liczba warsztatów które chce zwiedzić uczestnik
5. workshops – numery warsztatów 1...num-workshops
6. num-proc – liczba procesów

1.3 Zmienne

1. executing – num-workshops + 1 wortości logicznych, i-ta wartość jest prawdziwa jeśli proces jest w i-tej sekcji krytycznej
2. requesting – num-workshops + 1 wartości logicznych, i-ta wartość jest prawdziwa jeśli proces ubiega się o dostęp do i-tej sekcji krytycznej
3. accepted – num-workshops + 1 liczb naturalnych, i-ta wartość oznacza liczbę procesów pozwalających procesowi na dostęp do i-tej sekcji
4. deferred – num-workshops + 1 liczb procesów do których należy wysłać wiadomość po wyjściu z i-tej sekcji krytycznej
5. exited – liczba naturalna oznaczająca ilość procesów, które wyszły z wydarzenia
6. time – zegar danego procesu
7. pid – ranga procesu

1.4 Zegar

Do rozwiązania problemu dostępu do sekcji krytycznej używamy algorytmu Ricarta-Agrawala.

Każdy proces posiada jeden zegar „time” i dołącza go do każdej wysyłanej wiadomości.

Po odebraniu wiadomości zegar jest aktualizowany.

Procedura aktualizacji zegara wygląda następująco:

```
function tick(time'):
    time = max(time, time') + 1
```

2 Obsługa wiadomości

Zadaniem procedury ACK jest reagowanie na otrzymane wiadomości. Opcjonalnym parametrem procedury jest req-time, czyli czas zegara, który zawierała wiadomość REQ wysłana w procedurze P podczas ubiegania się o dostęp.

Najpierw odbieramy wiadomość w trybie nieblokującym. Jeśli nie otrzymaliśmy żadnej wiadomości, wychodzimy. Struktura wiadomości zawiera pola z etykietą, zegarem procesu i numerem sekcji krytycznej.

Aktualizujemy lokalny zegar, biorąc pod uwagę zegar z wiadomości.

Następnie, jeśli etykieta wiadomości to REQ, zgodnie z algorytmem Rickarta-Agrawala, jeśli nie ubiegamy się o dostęp do q-tej sekcji krytycznej ani w niej nie jesteśmy, lub ubiegamy się o dostęp i nasz proces ma mniejszy priorytet, to wysyłamy odpowiedź OK.

W przeciwnym przypadku, dodajemy nadawcę do listy deferred[q], po wyjściu z q-tej sekcji krytycznej wyślemy do wszystkich procesów z tej listy wiadomość OK.

Dla ścisłości, nasz proces ma mniejszy priorytet, kiedy czas zegara wysłany z wiadomością REQ, czyli req-time, jest późniejszy (większy) od czasu zegara z wiadomości przychodzącej. W przypadku równych czasów, proces z niższym numerem ma większy priorytet.

Gdy etykieta wiadomości to OK, zwiększamy accepted[q], czyli liczbę procesów zgadzających się na nasze wejście do sekcji q-tej krytycznej.

Jeśli etykieta to END, zwiększamy exited, czyli liczbę procesów które wyszły z pyrkonu.

```
function ACK(req-time = -1):
    if not recv (tag t q) from pid'
        return
    tick(t)
    if tag == REQ:
        lower-priority = pid' < pid if t == time else t < req-time
        if (not requesting[q] and not executing[q])
                or (requesting[q] and lower-priority):
            time += 1
            send (OK time q) to pid'
        else:
            deferred[q].append(pid')
    if tag == OK:
        accepted[q] += 1
    if tag == END:
        exited += 1
```

3 Dostęp do zasobu

Do obsługi dostępu do sekcji krytycznych stworzyliśmy dwie ogólne procedury: P i V.

Aby ubiegać się o dostęp do q-tego zasobu o pojemności n, należy wywołać procedurę $P(q, n)$.

Procedura na początku wysyła do wszystkich procesów (poza sobą) wiadomość z etykietą REQ, swoim aktualym zegarem i indeksem zasobu q. Zapisywany jest przy tym zegar w zmiennej req-time.

Następnie ustawiany jest stan procesu, w odniesieniu do q-tego zasobu i do czasu gdy num-proc - n procesów nie zaakceptuje dostępu do zasobu, odpowadamy na wiadomości w ACK.

Po uzyskaniu akceptacji, aktualizujemy stan.

```
function P(q, n):
    time += 1
    req-time = time
    send (REQ time q) to all except me

    requesting[q] = true
    accepted[q] = 0

    while accepted[q] < num-proc - n:
        ACK(req-time)

    requesting[q] = false
    executing[q] = true

```

Aby zwolnić dostęp do q-tego zasobu, wywołujemy procedurę $V(q)$.

Procodura wysyła wiadomość OK do wszystkich procesów z q-tej listy deferred[q], a następnie czyści tą listę i zaznacza, że proces nie wykonuje tej sekcji.

```
function V(q):
    time += 1
    send (OK time q) to deferred[q]
    deferred[q] = []
    executing[q] = false
```

4 Oczekiwanie na zakończenie

Procedura JOIN służy do oczekiwania na wyjście wszystkich procesów z wydarzenia.

Zaznaczamy fakt, że wyszliśmy z wydarzenia zwiększając liczbę procesów które wyszły. Następnie wysyłamy do wszystkich (poza sobą) procesów wiadomość END (numer zasobu nie ma znaczenia) . Gdy proces otrzymuje wiadomość END, zwiększa licznik exited o 1. Po otrzymaniu wiadomości END od wszystkich procesów, proces może brać udział w kolejnym pyrkonie.

```
function JOIN():
    exited += 1
    time += 1
    send (END time -1) to all except me
    while exited < num-proc:
        ACK()
```

5 Program główny

Mając omówione wszystkie zmienne i podprogramy, możemy przejść do opisu programu głównego dla ludzi odwiedzających Pyrkon.

Na początku każdy ubiega się o dostęp do wydarzenia wywołując procedurę P, z numerem kolejki wydarzenia – 0 i uczciwie podając maksymalną liczbę biletów – num-tickets.

Następnie uczestnik losuje max-worshops warsztatów z całej puli, a następnie dla każdego wylosowanego warsztatu, podobnie jak przy Pyrkonie, ubiega się o miejsce podając do procedury P numer warsztatu jako numer kolejki i maksymalną liczbę miejsc na tym warsztacie (może być ta sama dla wszystkich, albo inna dla każdego).

Podczas warsztatu uczestnik odpowiada na wiadomości od innych uczestników w procedurze ACK.

Po pewnym czasie uczestnik wychodzi z warsztatu i wywołując procedurę V z numerem warsztatu, zwalnia w nim miejsce. Po przejściu wszystkich wybranych warsztatów uczestnik zwalnia bilet wydarzenia wywołując V(0).

Gdy uczestnik wyjdzie z wydarzenia czeka na innych (aż Pyrkon się zakończy), a następnie wraca do początku programu i wykonuje go ponownie ad infinitum.

```
1  while true:
2      P(0, num-tickets)
3      shuffle workshops
4      booked = take max-workshops from workshops
5      for w in booked:
6          P(w, max-places in w)
7          while not timeout:
8              ACK()
9          V(w)
10     V(0)
11     JOIN()
```