Przetwarzanie rozproszone - projekt "Obsługa Pyrkonu"

Sebastian Maciejewski 132275 i Jan Techner 132332
grupa I1, zajęcia w środy o 9:45

15 maja 2019

1 Opis problemu

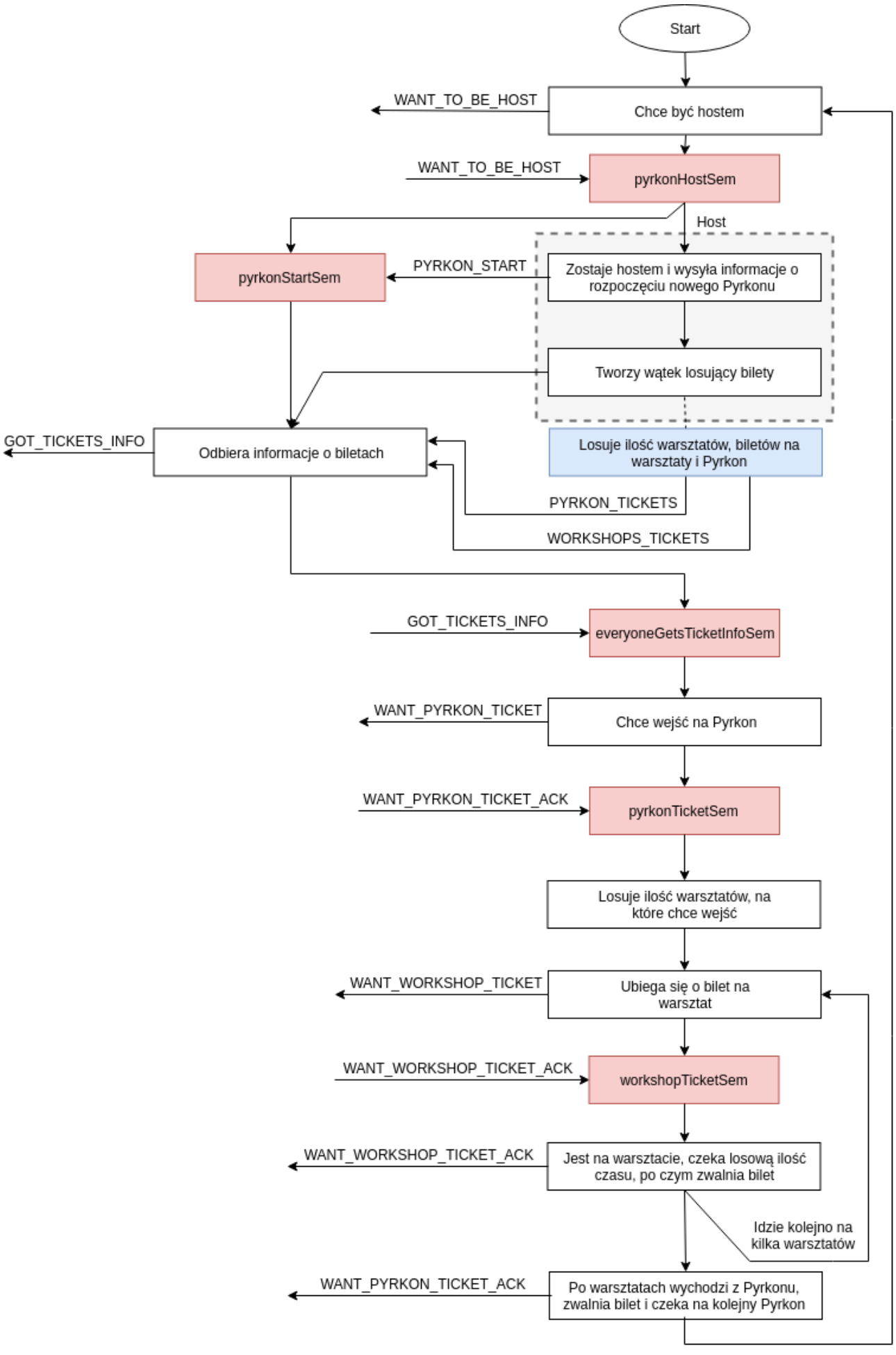
Zadanie polega na implementacji systemu zarządzania biletami na Pyrkon i warsztaty, które się na nim odbywają. Procesy (uczestnicy) ubiegają się o jeden z b biletów na Pyrkon, a następnie na kilka z rozróżnialnych warsztatów, z których każdy ma ograniczoną liczbę miejsc. Liczba warsztatów, miejsc na warsztatach i biletów na Pyrkon jest przed każdym Pyrkonem losowana przez proces, który uruchamia wątek losowania biletów. Poniżej przedstawiony jest opis algorytmu i schemat, który dokładnie obrazuje działanie programu.

Działanie algorytmu rozpoczyna się od zgłoszenia przez każdy z procesów chęci rozpoczęcia nowego Pyrkonu. W tym celu każdy proces wysyła wiadomość typu WANT_TO_BE_HOST zawierającą znacznik czasowy wysłania wiadomości i czeka na $p-1$ (p jest ilością procesów) wiadomości od innych procesów. Po otrzymaniu wszystkich wiadomości kązdy proces sprawdza czy jego znacznik czasowy jest największy i jeżeli tak, to proces zostaje hostem Pyrkonu.

Następnym krokiem jest inkrementacja numeru Pyrkonu - jest on wykorzystywany do odrzucania przestarzałych wiadomości, które mogłyby zaburzyć działanie programu. W naszej implementacji host wysyła wiadomość PYRKON_START do pozostałych procesów, po której odebraniu procesy inkrementują swoje numery Pyrkonu. Następnie host zwiększa swój numer Pyrkonu i tworzy wątek losujący.

Ta procedura jest zobrazowana w górnej części poniższego diagramu, dodatkowy wątek (zaznaczony na niebiesko) losuje ilość warsztatów, biletów na warsztaty i ilość biletów na Pyrkon, po czym przekazuje ją wszystkim procesom za pomocą wiadomości PYRKON_TICKETS i WORKSHOPS_TICKETS. Wykorzystany w tym miejscu semafor (odblokowywany w momencie, gdy wszystkie procesy otrzymają potwierdzenie otrzymania kompletnej informacji o biletach od innych procesów - GOT_TICKETS_INFO) służy wyrównaniu szans procesów na wejście na Pyrkon.
Dopiero wtedy mamy pewność, że wszyscy mają ten sam numer Pyrkonu i te same (aktualne) informacje o biletach - można zatem rozpocząć procedurę ubiegania się o bilety.

Każdy z Procesów informuje pozostałe o chęci wejścia na konwent (za pomocą wiadomości WANT_PYRKON_TICKET) i po otrzymaniu odpowiedniej ilości ($p-b$) pozytywnych odpowiedzi zajmuje bilet. Następuje losowanie ilości warsztatów, w których proces chce wziąć udział, po czym rozpoczyna się procedura ubiegania się procesu o bilet na warsztaty. Jak widać na poniższym diagramie, kroki, które wykonuje proces są podobne do tych przy pobieraniu biletu na Pyrkon. Głównymi różnicami są tu typy wiadomości i fakt, że proces czeka na $p-w_i$ wiadomości ze zgodą na wejście na warsztat, gdzie w_i to ilość biletów na i-ty warsztat.
Po otrzymaniu biletu i odczekaniu losowej ilości czasu (kiedy bierze udział w warsztacie), proces zwalnia bilet i, jeśli chce wziąć udział w jeszcze jakimś warsztacie, ponownie ubiega się o bilet, tym razem na kolejny, losowo wybrany, warsztat. Jeżeli proces przeszedł już wszystkie warsztaty, którymi był zainteresowany, wówczas zwalnia bilet na Pyrkon (wysyła odpowiedź ze zgodą na wejście oczekującym procesom) i wraca do momentu, w którym ubiegał się o zostanie hostem. Kiedy wszystkie procesy opuszczą Pyrkon, cały opisany proces rozpoczyna się od nowa.


Start
WANT_TO_BE_HOST
Chce być hostem
WANT_TO_BE_HOST
pyrkonHostSem
Host
pyrkonStartSem
PYRKON_START
Zostaje hostem i wysyła informacje o rozpoczęciu nowego Pyrkonu
Tworzy wątek losujący bilety
GOT_TICKETS_INFO
Odbiera informacje o biletach
Losuje ilość warsztatów, biletów na warsztaty i Pyrkon
PYRKON_TICKETS
WORKSHOPS_TICKETS
GOT_TICKETS_INFO
everyoneGetsTicketInfoSem
WANT_PYRKON_TICKET
Chce wejść na Pyrkon
WANT_PYRKON_TICKET_ACK
pyrkonTicketSem
Losuje ilość warsztatów, na które chce wejść
WANT_WORKSHOP_TICKET
Ubiega się o bilet na warsztat
WANT_WORKSHOP_TICKET_ACK
workshopTicketSem
WANT_WORKSHOP_TICKET_ACK
Jest na warsztacie, czeka losową ilość czasu, po czym zwalnia bilet
Idzie kolejno na kilka warsztatów
WANT_PYRKON_TICKET_ACK
Po warsztatach wychodzi z Pyrkonu, zwalnia bilet i czeka na kolejny Pyrkon

2 Założenia na temat środowiska

W naszym rozwiązaniu zakładamy, że kolejki komunikatów są **FIFO** i kanały komunikacji są niezwaodne o nieskończonej pojemności. Należy jednak zauważyć, że zastosowanie semaforów, na których proces musi czekać na wszystkie pozostałe procesy, w rzeczywistości ogranicza maksymalną ilość wiadomości obecnych jednocześnie w systemie.

3 Złożoność komunikacyjna i czasowa

W pojedynczym Pyrkonie można wyznaczyć ilości wysyłanych komunikatów - przy założeniu, że p to ilość procesów a w to ilość warsztatów, złożoność komunikacyjna jest dla pszczególych typów wiadomości następująca:

- $p \cdot (p-1)$ - WANT_TO_BE_HOST
- $(p-1)$ - PYRKON_START
- $p+p+w \cdot p$ - informacje o biletach (ilość warsztatów, biletów na Pykon i ilość biletów na każdy warsztat)
- $p \cdot (p-1)$ - GOT_TICKETS_INFO
- $p \cdot [(p-1)+(p-1)]$ - WANT_PYRKON_TICKET i WANT_PYRKON_TICKET_ACK
- $p \cdot (w-1) \cdot [(p-1)+(p-1)]$ - WANT_WORKSHOP_TICKET i WANT_WORKSHOP_TICKET_ACK

Sumaryczna (pesymistyczna) ilość komunikatów wysyłanych w ciągu jednego Pyrkonu to $2p^2(w+1)+p(1-w)-1$

Podobnie można opisać złożoność czasową (w pesymistycznym przypadku - gdy na Pyrkon jest tylko jeden bilet i każdy proces idzie na $w-1$ warsztatów):

- 1 - WANT_TO_BE_HOST
- 1 - PYRKON_START
- $1+1+w$ - informacje o biletach (ilość warsztatów, biletów na Pykon i ilość biletów na każdy warsztat)
- 1 - GOT_TICKETS_INFO
- $1+p$ - WANT_PYRKON_TICKET i WANT_PYRKON_TICKET_ACK
- $p \cdot (1+1) \cdot w$ - WANT_WORKSHOP_TICKET i WANT_WORKSHOP_TICKET_ACK

Złożoność czasowa w pesymistycznym przypadku (na Pyrkon jest tylko jeden bilet) to $6+w+p+2pw$.
Zakładamy przy tym, że przetwarzanie jest realizowane w nieskończenie krótkim czasie a komunikacja przy użyciu pojednynczej wiadomości trwa 1.