

Wirtualne zespoły robocze – porównanie architektury klient-serwer do architektury rozproszonej (multicast)

zadania dodatkowe:

1. Funkcja pozwalająca na zachowanie stanu obiektu oraz identyfikację użytkownika, tak aby stan obiektu własnego po ponownym uruchomieniu programu na tym samym lub innym komputerze był taki sam. Identyfikacja (system logowania) może być uproszczony do podawania ostatniej cyfry numeru ID. Należy użyć tylko architektury klient-serwer.
2. Funkcja pozwalająca na zachowanie stanu obiektu oraz identyfikację użytkownika, tak aby stan obiektu własnego po ponownym uruchomieniu programu na tym samym lub innym komputerze był taki sam. Identyfikacja (system logowania) może być uproszczony do podawania ostatniej cyfry numeru ID. Należy użyć tylko architektury multicast.
3. Wybór miejsca dla nowego pojazdu przez serwer, tak aby zjawiał się w miejscu możliwie najmniej widocznym przez innych użytkowników (zakładając, że mają ustawiony widok z kokpitu) lub jak najdalszym od innego, najbliższej położonego pojazdu. Odległość można obliczyć licząc długość wektora pomiędzy środkami pojazdów. W przypadku terenu płaskiego należy pominąć składową y . Należy użyć tylko architektury klient-serwer.
4. Wybór miejsca dla nowego pojazdu, tak aby zjawiał się w miejscu możliwie najmniej widocznym przez innych użytkowników (zakładając, że mają ustawiony widok z kokpitu) lub jak najdalszym od innego, najbliższej położonego pojazdu. Odległość można obliczyć licząc długość wektora pomiędzy środkami pojazdów. W przypadku terenu płaskiego należy pominąć składową y . Należy użyć tylko architektury multicast.
5. Zamiana pojazdów pomiędzy dwoma klientami na ich życzenie (potrzebna jest do tego celu ramka specjalnego typu). Wybór pojazdu innego klienta może odbywać się poprzez podanie jego numeru ID, znalezienie się w małej odległości lub w inny sposób zapewniający jednoznaczność wyboru. Zamiany powinien dokonywać serwer za zgodą obu klientów. Należy użyć tylko architektury klient-serwer.
6. Zamiana pojazdów pomiędzy dwoma użytkownikami na ich życzenie (potrzebna jest do tego celu ramka specjalnego typu). Wybór użytkownika może odbywać się poprzez podanie jego numeru ID, znalezienie się w małej odległości lub w inny sposób zapewniający jednoznaczność wyboru. Należy użyć tylko architektury multicast.
7. Wykrywanie kolizji przez serwer (np. poprzez porównywanie położen pojazdów - v_{Pos}) i poinformowanie każdego z uczestników. Pojazdy biorące udział w kolizji powinny się zatrzymać. Kontynuacja ruchu powinna być możliwa tylko w kierunku, w którym kolizja nie jest pogłębiania. Uwaga, należy sprawdzić, czy funkcja zadziała przy dużych prędkościach ruchu obiektów. Kształty obiektów można uprościć do sfer opisujących te obiekty, np. korzystając ze wzoru $(v_{Pos1}-v_{Pos2}).length() < radius1+radius2$. Należy użyć tylko architektury klient-serwer.
8. Wykrywanie kolizji (np. poprzez porównywanie położen pojazdów - v_{Pos}) i poinformowanie każdego z uczestników. Pojazdy biorące udział w kolizji powinny się zatrzymać. Kontynuacja ruchu powinna być możliwa tylko w kierunku, w którym kolizja nie jest pogłębiania. Uwaga, należy sprawdzić, czy funkcja zadziała przy dużych prędkościach ruchu obiektów. Kształty obiektów można uprościć do sfer opisujących te obiekty, np. korzystając ze wzoru $(v_{Pos1}-v_{Pos2}).length() < radius1+radius2$. Należy użyć tylko architektury multicast. Moment kolizji powinien zostać uzgodniony przez obie aplikacje lub z wykorzystaniem trzeciej aplikacji.

9. System głosowania: przyjęcie każdego nowego uczestnika grupy roboczej wymaga uzyskania większości głosów pozostałych uczestników. Liczenie głosów oraz decyzja o przyjęciu powinny być realizowane przez serwer. Należy użyć tylko architektury klient-serwer.
10. System głosowania: przyjęcie każdego nowego uczestnika grupy roboczej wymaga uzyskania większości głosów pozostałych uczestników. Należy użyć tylko architektury multicast. Aplikacja nowego uczestnika nie może uczestniczyć w liczeniu głosów ani w podejmowaniu decyzji o przyjęciu.
11. System zapobiegający kolizjom (podobny do lotniczego TCAS) poprzez wysyłanie komunikatów przez serwer- poleceń (np. skrętu w określoną stronę) klientom, których obiekty byłyby na kursie kolizyjnym. Groźbę kolizji można wykryć na podstawie położenia (v_{POS}) i prędkości (vV) pojazdów. Należy użyć tylko architektury klient-serwer.
12. System zapobiegający kolizjom (podobny do lotniczego TCAS) poprzez wysyłanie komunikatów - poleceń (np. skrętu w określoną stronę) aplikacjom, których obiekty byłyby na kursie kolizyjnym. Groźbę kolizji można wykryć na podstawie położenia (v_{POS}) i prędkości (vV) pojazdów. Należy użyć tylko architektury multicast. Należy przyjąć założenie, że jedna z aplikacji może samodzielnie nie wykryć zagrożenia.
13. Ujednolicanie numerów ID: każdy numer powinien być unikatowy przy wyszukiwaniu obiektu w jednym kroku. Należy użyć tylko architektury klient-serwer.
14. Ujednolicanie numerów ID: każdy numer powinien być unikatowy przy wyszukiwaniu obiektu w jednym kroku. Należy użyć tylko architektury multicast.
15. Wprowadzenie dodatkowego wirtualnego pojazdu z dowolnym algorytmem sterowania. Rozwiązanie powinno być odporne na zamknięcie części aplikacji. Należy użyć tylko architektury multicast.
16. Gra w „berka”: berkiem jest ten, kto ostatnio dołączył do grupy roboczej lub ten kto ostatnio został najechany przez innego berka. Przegrywa ten, kto jest berkiem po określonym czasie (np. 20 s.) od dołączenia się ostatniego gracza. Berek powinien być specjalnie oznakowany. Należy użyć tylko architektury klient-serwer.
17. Gra w „berka”: berkiem jest ten, kto ostatnio dołączył do grupy roboczej lub ten kto ostatnio został najechany przez innego berka. Przegrywa ten, kto jest berkiem po określonym czasie (np. 20 s.) od dołączenia się ostatniego gracza. Berek powinien być specjalnie oznakowany. Należy użyć tylko architektury multicast.
18. Mechanizm wyrównania długookresowych procentowych obciążeń obliczeniowych. Przykładowo przekazanie zadania symulacji obiektu w miejsce o ponad dwukrotnie krótszych czasach obliczeń. Na czas obliczeń wpływa ogólna moc obliczeniowa komputera oraz stopień obciążenia procesami. Można więc obciążenie regulować dynamicznie używając średniego czasu pomiędzy dwiema kolejnymi symulacjami pojedynczego obiektu. Należy użyć tylko architektury klient-serwer.
19. Kontrola kompatybilności ramek, gdy jakaś aplikacja związana z inną grupą roboczą przesyła do naszej aplikacji ramki innego typu. Należy użyć tylko architektury multicast.
20. Synchronizacja pór dnia. Należy użyć tylko architektury klient-serwer.
21. Synchronizacja pór dnia. Należy użyć tylko architektury multicast.