Opis planowanego rozwiązania

Projekt BMB

Autor: Bartosz Woźniak

Serwer

Główny wątek

Program serwera wywoływany jest z jednym parametrem – numerem portu. W przypadku niepoprawnego wywołania wypisywane jest *Usage* wraz z odpowiednimi instrukcjami. W pierwszym kroku tworzony jest *socket UDP* oraz przypisywane jest do niego adres i numer portu. Serwer będzie ignorował sygnał *SIGPIPE.* Wszyscy klienci (zarówno pierwszego typu – gracze, jak i drugiego typu – zlecenia) będą trzymani w tablicy struktur *connections* (nie będzie to zmienna globalna). Następnie rozpoczyna się główna, ciągła, powtarzająca się część pracy serwera – funkcja *doWork*. W funkcji tej wyróżniamy dwa zasadnicze zadania:

1) Czytanie wiadomości z *socketu*
2) Ustawienie alarmu na 10 sekund i po nadejściu sygnału *SIGALRM* kolejne wybudzanie wszystkich wątków celem obsługi podłączonych klientów

Funkcja *doWork*

W funkcji tej deklarowana jest tablica struktur *connections* zawierająca informacje o wszystkich aktualnie podłączonych klientach oraz tablica zawierająca mapę z aktualnym stanem gry. Zmienne te będą przekazywane przez wskaźnik do tworzonych w przyszłości wątków. Następnie ustawiany jest *alarm* na 10 sekund. W nieskończonej pętli serwer oczekuje wiadomości na *sockecie*. Po otrzymaniu nowej wiadomości odpowiednio je obsługuje. Gdy nadejdzie sygnał *SIGALRM*, to odbieranie nowych wiadomości jest chwilowo wstrzymane i następuje obsługiwanie już podłączonych klientów, kolejne wybudzanie wątków. Gdy wszyscy klienci otrzymali aktualny stan gry, ustawiamy znowu alarm na 10 sekund i wracamy do odbierania wiadomości. Tylko wątek główny (ten pierwszy) będzie reagował na *SIGALRM*, nowo utworzone wątki będą maskowały ten sygnał. *SIGALRM Handler* będzie ustawiał globalną zmienną na 1. Na jej podstawie oczekiwanie na nowe wiadomości zostanie wstrzymane, ewentualna obsługa bieżącej wiadomości zostanie dokończona, a następnie nastąpi wybudzanie kolejnych wątków. Po skończeniu należy wyzerować globalną zmienną związaną z *alarmem*.

Jakie wiadomości może otrzymać serwer na sockecie?

1) Wiadomość powitalna *hello* od klienta pierwszego lub drugiego typu. Musi ona zawierać wyłącznie informację, czy to klient gracz, czy zlecenie. Po odebraniu takiej wiadomości, odpowiedź nie jest odsyłana od razu (klient otrzyma dopiero najbliższą *status message*). Najpierw musimy sprawdzić, czy klient ten nie jest już podłączony (ewentualna zduplikowana wiadomość na *sockecie*) poprzez przeszukanie tablicy *connections*. Jeśli wiadomość jest zduplikowana, to odrzucamy wiadomość – nic nie robimy. W przeciwnym wypadku zaczynamy obsługę wiadomości.

   Jeśli otrzymaliśmy wiadomość od „gracza”, to sprawdzamy, czy na planszy jest wolne miejsce (reguła trzech przecznic – po prostu próbujemy klienta ustawić we wszystkich miejscach aż nam się uda, albo sprawdzimy całą mapę bez sukcesu). Jeśli mamy wolne

miejsce na mapie, to dodajemy klienta do tablicy *connections*, ustawiamy klienta w wylosowanym miejscu na mapie i powołujemy do życia nowy wątek, który będzie go obsługiwał. Do nowego wątku przekazujemy przez wskaźnik planszę i klientów. Uzupełniamy odpowiednią strukturę *connection* o *tid* nowego wątku. Uzupełniamy również odpowiednią flagę w tej strukturze, iż otrzymaliśmy *status message*, aby przypadkowo nie odrzucić klienta przy pierwszym ruchu. Nowy wątek uaktualnia zmienną z wynikiem gracza (100 zł) w strukturze i czeka na swój ruch. Jeśli na planszy nie było wolnego miejsca, to nie możemy dodać nowego klienta, możemy odesłać komunikat o niepowodzeniu.

Jeśli otrzymaliśmy wiadomość „zlecenie", to postępujemy analogicznie z pominięciem rozstawiania na mapie i zmiennej z wynikiem.

Wiadomość *hello* ma postać napisu o treści „A" jeśli jest to wiadomość od „gracza" lub „B" jeśli to „zlecenie".

2) *Status message* – odbierana od klientów obu typów, co około 3 sekundy. Ma postać napisu o treści „S". Odebranie tej wiadomości powoduje uaktualnienie odpowiedniej struktury z *connections* – ustawienie informacji, iż otrzymaliśmy status w tej iteracji (jest on zerowany przez wątek po przesłaniu planszy do klienta).

   Wątek odpowiadający za obsługę klienta sprawdza, czy w bieżącej iteracji otrzymaliśmy choć jeden *status message* dopiero w momencie swojego ruchu – obudzenia. Jeśli nie, to rozłączamy klienta (brak kontaktu z klientem powyżej 5 sekund), w przeciwnym wypadku kontynuujemy normalny proces obsługi, wykonujemy ruch itp. i zerujemy status.

3) Wiadomość typu „L" lub „P" informująca, iż gracz zamierza skręcić przy najbliższym ruchu. Znajdujemy odpowiedniego klienta w *connections* i uaktualniamy strukturę. Zawsze nadpisujemy bieżącą wartość.

4) Wiadomość ze współrzędnymi „zlecenia" – napis postaci „X1;Y1;X2;Y2", gdzie xi, yi to współrzędne punktu początkowego i końcowego. Parsujemy wiadomość i uaktualniamy odpowiednią strukturę. Nie nadpisujemy bieżącej wartości, jeśli współrzędne w strukturze nie są puste (ustawione na -1), to ignorujemy wiadomość.

Otrzymanie każdej z powyższych wiadomości wymaga przeszukania tablicy *connections* i znalezienia odpowiedniego klienta – funkcja *findIndex* (wyszukiwanie po adresie). W przypadku wiadomości typu 2), 3) lub 4) i nie znalezieniu pasującego klienta, wiadomość jest ignorowana. W przypadku otrzymania wiadomości o innej treści również jest ona ignorowana.

Synchronizacja wątków

Wszystkie wątki obsługujące klientów czekają na swoją kolej na jednej *conditional variable*. Każdy wątek czeka, aż wartość zmiennej będzie równa indeksowi w tablicy klienta, którego obsługuje. Wątek znalazł ten indeks chwilę po wystartowaniu. W posiadaniu *mutexa* jest wątek główny (ten obsługujący *socket*). On może czekać na wartość -1 na *conditional variable*. Wątek główny ustawia *cond_var* na wartość odpowiedniego indexu (od 0 do

connections.Length - 1) w pętli, zwalnia *mutex,* sygnalizuje zmianę i czeka na odzyskanie kontroli (wartość -1 na *cond_var*), następnie przechodzi do kolejnej iteracji pętli, czyli budzi kolejny wątek.

Wątek obsługujący klienta po obudzeniu i zablokowaniu *mutexa* zaczyna aktualizować stan gry:

1. Znajduje strukturę swojego klienta.
2. Sprawdza, czy klient się nie rozłączył (odpowiednie pole status w strukturze). Jeśli się rozłączył, usuwamy klienta z tablicy *connections* oraz z planszy. Kończymy wątek. Wątek może być *detached* – nie musimy sprawdzać jego statusu po zakończeniu.
3. Jeśli klient wciąż jest podłączony, to sprawdzamy, czy klient chciał skręcić (pole w strukturze), następnie wykonujemy ruch, wykrywamy kolizję i opcjonalnych klientów do zabrania itp. Uaktualniamy wynik klienta (sprawdzamy, czy większy od 0).
4. Zaznaczamy w strukturze, że wykonaliśmy ruch.
5. Oddajemy kontrolę do wątku głównego (*cond_var* = -1).

Klient „zlecenie" i „gracz" są obsługiwanie nieco inaczej, wątek rozpoznaje jakiego klienta obsługuje po odpowiednim polu w strukturze.

Proces budzenia wątków powtórzymy jeszcze raz, aby każdy klient mógł otrzymać w pełni zaktualizowaną mapę. Za drugim razem wysyłamy tylko mapę na *socket*, każdy wątek do swojego klienta.

Serwer podsumowanie

Ilość wątków wynosi: liczba_klientów („zleceń" i „graczy" razem) + 1.
Wątki synchronizowane są poprzez jedną *conditional variable (*z *mutexem).* Każdy wątek czeka na swój *index,* a wątek główny na -1.
Komunikacja poprzez jeden *socket UDP.* Wątek główny czyta wszystkie datagramy i przetwarza je. Wątki odpowiedzialne za klientów wysyłają datagramy przez *socket*.
Główna część programu to następujące funkcje:

- *doServer* – pętla główna, odbiór komunikatów, alarm 10 sekund, synchronizowanie wątków
- *findIndex* – znajdź odpowiednie klienta po adresie w tablicy *connections*
- funkcje pomocniczy do wykonywania ruchów i odświeżania mapy itp.

Klient

Klient to prosty program wywoływany z adresem serwera i numerem portu. Wypisuje odpowiednie *Usage* w przypadku niepoprawnego wywołania. Łączy się on z odpowiednim *socketem* i wysyła wiadomość powitalną o treści „A" lub „B" (w zależności od typu klienta). Czeka 10 sekund na odpowiedź, jeśli jej nie otrzyma, ponawia próbę. Jeśli otrzymał odpowiedź, to zaczyna się zasadnicza część programu:

1) Tworzony jest nowy wątek, który będzie wysyłał *status message* co 3 sekundy.
2) Tworzony jest nowy wątek, który czyta z konsoli i wysyła wiadomości o treści „L" lub „P" przez *socket*.
3) Obecny wątek odbiera wiadomości z *socketu* i wypisuje na ekran.

Klient „zlecenie” działa analogicznie z tą różnicą, że czeka na współrzędne z konsoli, a nie „L” lub „P” i współrzędne te powinny być przyjęte tylko raz. Wątek powinien poczekać na odpowiedź od serwera, że zostały przyjęte lub nie. Jeśli zostały przyjęte, to powinien się skończyć i więcej nie reagować na wpisy z konsoli.