Aleksandra Ostrowska 156121

Sprawozdanie sk2

Sieciowa turowa gra logiczna w warcaby

1. Opis projektu

Wybranym przeze mnie projektem jest sieciowa gra w warcaby. W projekcie są zaimplementowane są podstawowe zasady gry, gdzie dwójka graczy stacza ze sobą pojedynek wcielając się w białe lub czarne pionki, a serwer sprawdza prawidłowość ruchów graczy. Serwer zajmuje się również zwalnianiem zasobów po zakończeniu rozgrywki i informacji graczy o przebiegu gry. Na serwerze mogą się odbywać do 10 rozgrywek równocześnie.

Do zaimplementowania współbieżności użyłam biblioteki <pthread.h>. Serwer korzysta również z standardowych bibliotek Unix/Linux takich jak <arpa/inet.> i <sys/socket.h>.
Używany jest pthread.h to tworzenia wątków za pomocą pthread_create. Wykorzystuje się również funkcji do terminacji wątków, zwalniania zasobów oraz zamków mutex na wątkach.
Sys/socket.h jest używany do struktury sockaddr_in oraz operacji na deskryptorach typu connect(), accept(), socket().

Komunikacja odbywa się według szablonu architektury klient-serwer, używając protokołu TCP oraz protokołu IPv4. Do nasłuchiwania i akceptowania połączeń użyte zostały funkcje listen oraz accept. Komunikacja obsługiwana jest przez funkcje recv oraz send. Dane o stanach gry, deskryptorach graczy oraz obecnych turach znajdują się w tablicach o wielkości ilości maksymalnych obsługiwanych gier. Gry są rozróżniane przez gameID.

Serwer posiada liczne funkcje pomocnicze w celu sprawdzania logiki gry. Sprawdza on czy ruchy nie wybiegają poza granice gry, czy gracz używa swoich pionków, czy dane pole jest puste. Używa on tych funkcji w głównej funkcji uwierzytelniającej prawidłowość ruchu is_valid_move.
Oprócz tego serwer tworzy i uaktualnia plansze gier.

2. Opis komunikacji pomiędzy serwerem i klientem

Komunikacja, jak wspomniałam wcześniej odbywa się przez nawiązanie połączenia przez gniazdo protokołu TCP. Serwer nasłuchuje na porcie 2222 i odbiera połączenia od wszystkich adresów (INADDR_ANY), jednak do testów korzystałam z 127.0.0.1. Dla każdego połączenia tworzony jest wątek przez funkcję pthread_create a dla wątku jest wykonywana funkcja socketThread obsługująca grę dla każdego gracza.

Wstępnie socketThread wyszukuje dla gracza wolny slot w danej grze, przydziela mu kolor i informuje o statusie rozgrywki (szukanie drugiego gracza/ gra się rozpoczyna).
Następnie dla każdego wątku serwer odbiera informację o ruchach graczy, sprawdzi ich prawidłowość, odpowiednio uaktualnia stan planszy, upewniając się o niezakończonej rozgrywce i przekazuje odpowiedniemu graczowi turę. W sytuacji gdy

gracz wpisze nieodpowiedni ruch, serwer będzie dopytywać aż do skutku otrzyma prawidłową odpowiedź.

Gra kończy się brakiem jakichkolwiek pionków jednego z graczy lub odejściem jednego z graczy. Pierwszy przypadek kończy się wysłaniem obu graczy wyniku rozgrywki i zakończeniem połączenia, wraz z zwolnieniem zasobów.
W drugim przypadku, kiedy jeden z graczy podaje komendę ‘exit’, serwer rozłącza połączenie z tym graczem, informuje drugiego gracza o sytuacji gry, po czym również z nim terminuje wątek. Klient na wiadomość serwera również przerywa pętle i zamyka swój deskryptor.

3. Podsumowanie

Podsumowując, architektura opiera się na połączeniu TCP, na protokole AF_INET (IPv4). Serwer używa lpthreads do monitorowania 10 gier naraz. Klient łączy się z serwerem za pomocą funkcji connect(), a komunikacja odbywa się za pomocą funkcji send() i recv().
Ruchy graczy są implementowane w formacie Startowy wiersz Startowa kolumna Końcowy wiersz Końcowa Kolumna (np. 5243, które jest startowym ruchem dla białych pionków). Gracze trwają w nieskończonej pętli do czasu zakończenia rozgrywki lub przesłania komendy umożliwiającej wyjście z gry.

W programach zaimplementowana jest obsługa błędów, która była w stanie oszczędzić czasu w rozwiązywaniu problemów z projektem, oraz zwalnianie zasobów przez zamykanie deskryptorów i terminowanie wątków.

Podczas samej implementacji gry napotkałam na swojej drodze kilka problemów których rozwiązanie sprawiło mi trudności, między innymi były nimi:
Komunikacja serwera z graczem, ze względu na problemy w debugowaniu, przez to że wiadomości nie były czasem odbierana po stronie klienta i ciężko było znaleźć przyczynę.

Zaimplementowanie czyszczenie buforów i ograniczanie przesyłania białych znaków między odbiorcami, które często przechwytywała funkcja scanf w kliencie.

Chwilową trudnością była także sama implementacja warcabów, których zasady musiałam sobie przypomnieć aby ich adaptacja była jak najbardziej klarowna dla graczy w celu zapewnienia płynność i zrozumienie rozgrywek.

Samą trudnością było także testowanie rozgrywek i poprawności działania systemu ze względu na monotonność tych procesów i czas trwania, w celu zdiagnozowania potencjalnych błędów które mogą wystąpić podczas rozgrywki a także w trakcie oczekiwania i zakończenia spotkania, co musiało zostać sprawdzone z różnych poziomów, gracza jaki i samego serwera.