Wrocław, dnia 11 czerwca 2007

Grzegorz Pietrzak (133329)
Jacek Symonowicz (133375)

Projekt:
Gra sieciowa - Labirynt

Kurs: Sieciowe systemy operacyjne UNIX (2)
Prowadzący: dr inż. Tomasz Surmacz

1. Cel projektu

Celem projektu było stworzenie gry labiryntowej rozgrywanej w czasie rzeczywistym, z wykorzystaniem biblioteki *ncurses*. Gra powinna składać się z klientów wyświetlających labirynt w oknie tekstowym 80 × 25 oraz serwera synchronizującego rozgrywkę.

2. Realizacja

Zasady zrealizowanej przez nas gry są proste – dowolna ilość graczy porusza się po labiryncie nie interferując między sobą. Elementem rywalizacji jest tu gonienie uciekającego elementu[1]. Jego złapanie oznacza inkrementację liczby punktów danego gracza.

Gra została zrealizowana w oparciu o strumieniowy protokół TCP. Konkurencyjne UDP, pomimo znacznie lepszego dopasowania do aplikacji działających w czasie rzeczywistym, nie zapewnia niestety wszystkich potrzebnych nam mechanizmów. Na potrzeby projektu opracowany został prosty protokół transmisji w warstwie aplikacji.

Ogólna idea jest następująca: każdy klient co pół sekundy wysyła do serwera bufor cykliczny zawierający listę ruchów, jakie wykonał. Serwer rozsyła tę listę do pozostałych graczy, którzy na podstawie otrzymanych danych przewidują ruchy tego klienta (na ok. 4 sekundy). Po opróżnieniu bufora predykcji następuje zatrzymanie klienta na ekranach przeciwników i sygnalizacja laga. Obiekt, który musi złapać klient zachowuje się nieco inaczej – serwer generuję listę ruchów obiektu na najbliższe parę sekund i rozsyła ją do wszystkich graczy. Jeśli lista ta się wyczerpie, a nie nadejdą nowe dane obiekt zostanie wstrzymany. Rozgrywką zarządza gracz (dalej nazywany hostem), który pierwszy zalogował się do serwera.

Postacie występujące w grze to: gracz `@`, przeciwnicy `&` oraz łapany obiekt `*`. Podczas normalnej komunikacji mają one kolor biały, w przypadku dłuższej nieaktywności połączenia ich kolor zmienia się na czerwony. Gracze mogą dołączać do gry w każdej chwili, zarówno przed, jak i po jej starcie. Rozgrywka odbywa się na planszy zdefiniowanej w pliku tekstowym, rozsyłanej klientom.

2.1. Budowa klienta

Aplikacja klienta jest wielowątkowa (biblioteka *pthreads*), co pozwala zarówno na upłynnienie rozgrywki, jak i realizację trybu full-duplex w komunikacji z serwerem. 5 wątków odpowiada za: odświeżanie ekranu, pobieranie klawiszy, inkrementację zegara timestamp, wysyłanie do socketa oraz odbiór z socketa. Łączenie z serwerem następuje zaraz po uruchomieniu klienta (numer IP podawany jest w parametrze), w przypadku niepowodzenia zwracane są odpowiednie komunikaty. Zerwanie połączenia przez serwer powoduje natychmiastowe zakończenie pracy klienta.

Synchronizacja z zegarem serwera następuje wraz z przyjściem każdego pakietu, z testów aplikacji wynika, że takie rozwiązanie jest w naszym przypadku wystarczające. Pakiety od serwera przychodzą na tyle często, że korekcja timestamp-u nie jest wymagana. Ponadto inkrementacja zegara gry przebiega w osobnym wątku, co nie gwarantuje dokładnego odmierzania czasu.

Przerwanie fizycznego połączenia między serwerem a klientem spowoduje zatrzymanie przeciwników i łapanego obiektu, a następnie wyświetlenie ostrzeżenia LAG w dolnej części ekranu. Po przywróceniu połączenia gra toczy się dalej.

1 Inspiracja: kreskówka „Dastardly and Muttley in Their Flying Machines” (Hanna Barbera, 1969) w której kilku lotników z czasów I wojny światowej otrzymało rozkaz złapania amerykańskiego gołębia pocztowego

Wspomniana wcześniej predykcja ruchów odbywa się w całości w kliencie, polega na interpolacji kolejnych ruchów w celu wyeliminowania przeskoków pomiędzy kolejnymi odebranymi porcjami danych. Płynność predykcji zależy od wielu czynników, obecne ustawienia zostały zoptymalizowane dla sieci LAN.

Sterowanie w grze odbywa się przy pomocy klawiszy 'w' (góra), 's' (dół), 'a' (lewo), 'd' (prawo), 'x' (koniec) oraz 'p' (start/stop gry, tylko dla hosta).

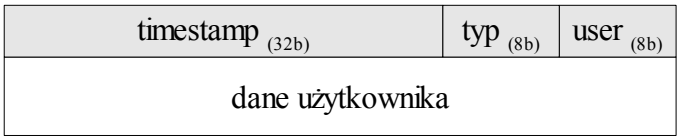
2.2. Budowa serwera

Serwer jest aplikacją mającą na celu obsługę wielu połączeń jak najmniejszym kosztem. Składa się z dwóch wątków, z których jeden jest bardzo rozbudowany i odpowiada za odbieranie i wysyłanie danych przez wiele gniazd. Drugi jest małym uzupełnieniem, bowiem ma za zadanie jedynie inkrementację wewnętrznego czasomierza. Przychodzące połączenia są przypisywane do kolejnych gniazd ustawianych na nieblokujące przy odczycie i zapisie. Zarządza nimi funkcja select (biblioteka sys/select) wraz z makrami służącymi do obsługi wielu deskryptorów jednocześnie. Każda próba wysłania i odebrania strumienia bajtów poprzez gniazda sprawdzana jest pod kątem ewentualnych błędów. W przypadku ich wystąpienia gniazdo zostaje zamknięte, a na miejsce odłączonego gracza może dołączyć się kolejny. Aby zapewnić odporność na zrywanie połączeń, zaimplementowano również prosty interfejs obsługi sygnału SIGPIPE.

Serwer uruchomić można z parametrem oznaczającym numer portu, na którym ma nasłuchiwać. drugim parametrem może być ścieżka do pliku z planszą, na której ma być przeprowadzona rozgrywka.

2.3. Komunikacja klient → serwer

W komunikacji od klienta do serwera zastosowany jest prosty format pakietu, w którym można zmieścić tylko jeden typ informacji.



Rysunek 1: Ogólna budowa pakietu relacji klient → serwer

- timestamp - znacznik czasowy wysłanej wiadomości (odczytany w kliencie)
- typ – rodzaj pakietu:
  - PACKET_START_ISSUED – klient będący zarządcą gry chce rozpocząć grę
  - PACKET_STOP_ISSUED – klient będący zarządcą gry chce zatrzymać grę
  - PACKET_PIGEON_ISSUED – klient zgłasza złapanie obiektu
  - PACKET_PLAYER_MOVES – klient przesyła listę swoich ruchów
- user – numer użytkownika
- dane użytkownika – dodatkowe informacje specyficzne dla danego typu pakietu

Należy nadmienić, że dla każdego socketu ustawiana jest flaga TCP_NODELAY pozwalająca ograniczyć opóźnienie przesyłu danych.

2.4. Komunikacja serwer → klient

Komunikacja od serwera do klienta jest nieco bardziej skomplikowana, używane są tu pakiety o zmiennej ilości pól, co pozwala na bardzo dużą elastyczność komunikacji.

timestamp (32 b)		ile_pol (8b)
typ (8b)	dane użytkownika 1	
typ (8b)	dane użytkownika 2	
typ (8b)	dane użytkownika n	

Rysunek 2: Ogólna budowa pakietu relacji serwer → klient

- timestamp - znacznik czasowy wysłanej wiadomości (odczytany w serwerze)
- ile_pól – liczba pól w dalszej części pakietu, związana z możliwością umieszczenia w pojedynczym pakiecie wielu typów informacji
- typ – rodzaj kolejnego pola:
  - PACKET_OPPONENT_MOVES – lista ruchów przeciwnika, na ich podstawie następuje predykcja ruchu w kliencie
  - PACKET_PIGEON_MOVES – lista ruchów obiektu, nie ma ty predykcji – ruchy są znane a priori (generowane z wyprzedzeniem paru sekund)
  - PACKET_POINT – zmiana liczby punktów dowolnego gracza
  - PACKET_MAP – przesłanie klientowi planszy
  - PACKET_PLAYER_POSITION – natychmiastowe ustawienie gracza / przeciwnika na ściśle określonej pozycji na planszy
  - PACKET_PIGEON_POSITION – natychmiastowe ustawienie uciekającego obiektu na ściśle określonej pozycji na planszy
  - PACKET_START_GAME – rozpoczęcie rozgrywki
  - PACKET_STOP_GAME – zakończenie rozgrywki
  - PACKET_PLAYER_CONNECTED – przyłączenie się nowego przeciwnika
  - PACKET_PLAYER_DISCONNECTED – odłączenie się przeciwnika
  - PACKET_MY_NUMBER – powiadomienie klienta o jego numerze w tablicy graczy oraz o uprawnieniach do rozpoczynania i kończenia gry
  - PACKET_PLAYER_REFUSED – odrzucenie żądania przyłączenia do gry
- dane użytkownika – dodatkowe informacje specyficzne dla danego typu pakietu

3. Wnioski

Tworzenie gier sieciowych działających w trybie rzeczywistym to duże wyzwanie dla programistów, w systemach UNIX jest to znacznie ułatwione poprzez przejrzystą implementację gniazd sieciowych. W trakcie realizacji projektu okazało się, że w protokole TCP występują znaczne (dla szybkiej komunikacji) opóźnienia, co powoduje konieczność predykcji zachowań przeciwników. Predykcja w porównaniu do rzeczywistych danych zwraca gorsze wyniki i wpływa negatywnie na płynność rozgrywki, jednakże jest stosowana w większości dzisiejszych gier w celu uodpornienia ich na problemy z działaniem sieci.

Testy na localhost nie wykazały nieprawidłowości w działaniu interfejsu sieciowego, komunikacja była wystarczająco szybka i niezawodna. Wystąpiły trudności z dopasowaniem parametrów buforowania i predykcji ruchu, co jest widoczne podczas poruszania się przeciwników.

Testując grę na dwóch odległych hostach dało się zauważyć duże opóźnienie (szczególnie w kierunku serwer->klient), stąd też dosyć często sygnalizowane były przestoje w komunikacji. W przypadku niektórych serwerów nie było możliwe uruchomienie gry ze względu na niekompatybilność zainstalowanych bibliotek. Kompilacja i uruchamianie przebiegały bezproblemowo w dystrybucjach Fedora Core 6, Ubuntu 7.04 oraz OpenSUSE.