Marcin Olter 160095
Aleksandra Loret 160106

Sprawozdanie sk2

Temat: Komunikator

1. Opis projektu

Projekt polegał na stworzeniu prostego komunikatora typu klient–serwer, umożliwiającego rejestrację i logowanie użytkowników, zarządzanie listą znajomych, tworzenie rozmów prywatnych oraz grupowych, a także wymianę wiadomości tekstowych. System działa w architekturze klasycznej: jeden serwer obsługuje wielu klientów jednocześnie, a każdy klient posiada graficzny interfejs użytkownika.

Po stronie serwera zaimplementowano logikę protokołu tekstowego (komendy wysyłane liniami zakończonymi \n), obsługę wielu połączeń za pomocą wątków oraz prostą "bazę danych" opartą o pliki (users.db, friends.db, groups.db). Hasła użytkowników nie są przechowywane jawnie – serwer zapisuje jedynie ich skróty (hash). Dodatkowo zrealizowano mechanizm obecności (online/offline): gdy użytkownik się loguje lub wylogowuje, jego znajomi otrzymują zdarzenie informujące o zmianie statusu.

Technologie:

- Serwer: język C (GNU/Linux), BSD sockets (TCP), wielowątkowość pthread, obsługa sygnałów (CTRL+C), funkcja crypt() do hashowania haseł.
- Klient: Python 3, biblioteka Tkinter do GUI, gniazda TCP (socket), wątek odbioru + kolejka do aktualizacji UI.

2. Opis komunikacji pomiędzy serwerem i klientem

Komunikacja między klientem a serwerem opiera się o jedno trwałe połączenie TCP. Klient wysyła do serwera tekstowe komendy w postaci pojedynczych linii zakończonych znakiem nowej linii \n. Serwer odpowiada również liniami tekstu.

W projekcie przyjęto dwa typy wiadomości od serwera:

1. Odpowiedzi statusowe

    - OK <coś> – informacja o powodzeniu operacji (np. OK LOGIN, OK ADD_FRIEND)
    - ERROR <opis> – błąd wraz z czytelnym opisem przyczyny

2. Zdarzenia asynchroniczne (EVENT) – serwer może je wysłać niezależnie od tego, czy klient aktualnie coś zrobił:

    - EVENT FRIENDS a,b,c – aktualna lista znajomych
    - EVENT PRESENCE user ONLINE|OFFLINE – zmiana statusu znajomego
    - EVENT GROUPINFO id name members_csv – informacje o grupie
    - EVENT MSG conv from msg... – wiadomość do rozmowy prywatnej lub grupy

Identyfikatory rozmów (conv_id) mają dwa formaty:

- prywatne: u:login1:login2 (zawsze w kolejności alfabetycznej)
- grupowe: g:<id> (np. g:12)

Schemat komunikacji (przykład logowania i pobrania list):

- Klient → LOGIN ala haslo
- Serwer → OK LOGIN
- Klient → LIST_FRIENDS
- Serwer → OK LIST_FRIENDS
- Serwer → EVENT FRIENDS ola,jan
- Serwer → EVENT PRESENCE ola ONLINE
- Serwer → EVENT PRESENCE jan OFFLINE

Po stronie klienta odbiór danych działa w osobnym wątku, który przekazuje linie do kolejki. Główna pętla GUI co pewien czas pobiera dane z kolejki i aktualizuje interfejs (np. listę znajomych, statusy, okna czatu).

3. Podsumowanie

Najważniejsze elementy implementacji:

- Serwer jest wielowątkowy: każde połączenie klienta jest obsługiwane w osobnym wątku (pthread_create). Dzięki temu wielu użytkowników może jednocześnie korzystać z komunikatora.
- Dane są przechowywane w plikach tekstowych (prosta forma bazy danych): users.db zawiera loginy i hashe haseł, friends.db relacje znajomych, groups.db grupy i członków.
- Hasła użytkowników nie są przechowywane jawnie. Serwer generuje sól i zapisuje hash (SHA-512 z crypt()), a podczas logowania porównuje wynik hashowania z wartością w pliku.
- Mechanizm online/offline został rozwiązany poprzez przechowywanie listy zalogowanych użytkowników w pamięci RAM (mapa user -> socket). Podczas logowania/wylogowania serwer wysyła do znajomych zdarzenie EVENT PRESENCE.
- Klient ma GUI w Tkinter. Najważniejsze było rozdzielenie wątku odbioru (network) od wątku GUI – bez tego aplikacja by się “zawieszała”. Dlatego odbiór działa w wątku, a aktualizacja GUI jest robiona przez kolejkę.

Trudności / problemy:

- Największą trudnością była synchronizacja: serwer używa mutexów do ochrony dostępu do plików i listy online, żeby wątki nie wchodziły sobie w drogę.
- Problemem praktycznym było poprawne zatrzymywanie serwera przez CTRL+C. Serwer często “wisiał” na blokującym accept(), dlatego trzeba było dodać obsługę sygnałów przez sigaction i zamykanie socketu nasłuchującego, żeby przerwać accept().

Projekt pozwolił przećwiczyć praktyczną pracę z gniazdami TCP, protokołem tekstowym, wielowątkowością oraz podstawami tworzenia aplikacji klient-serwer z interfejsem graficznym.