Sprawozdanie SK2
Komunikator internetowy

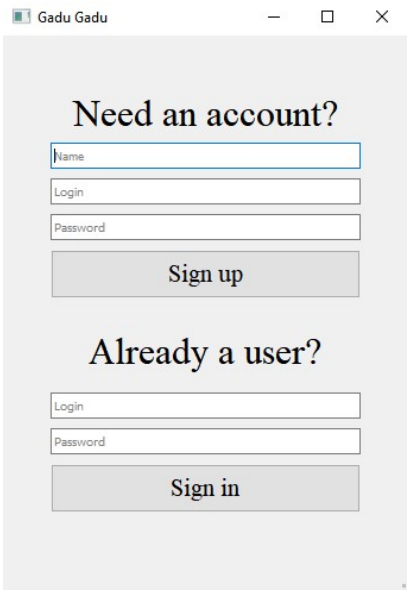
Natalia Szymczyk 145250
Jan Świątek 145390

Styczeń 2022

1 Opis projektu

Jako projekt wykonaliśmy aplikację pozwalającą na komunikację między klientami. Użytkownik może się zarejestrować, jeśli pierwszy raz korzysta z naszej aplikacji, bądź zalogować swoimi danymi, jeśli posiada już konto. Podstawową funkcjonalnością dla zalogowanych użytkowników jest wysyłanie oraz odbieranie wiadomości z użytkownikami, których posiada w znajomych. Możliwe jest oczywiście wyszukiwanie oraz dodawanie znajomych na podstawie loginu. Dodatkowo zalogowany użytkownik może ustawić/zmienić swój opis, który jest widoczny dla jego znajomych.

Serwer został napisany w języku C, zaś do implementacji klienta wykorzystaliśmy język Python oraz bibliotekę PyQt5 do pracy z GUI.



2 Opis komunikacji

Do zaimplementowania naszego komunikatora skorzystaliśmy z podejścia współbieżnego. Serwer TCP, na którym działa nasz program, jest wielowątkowy.

Na początku tworzymy gniazdo serwera (*server_ socket*) i związujemy go z odpowiednim adresem (funkcja *bind*).

```
server_socket = socket(PF_INET, SOCK_STREAM, 0);

server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
server_addr.sin_addr.s_addr = inet_addr(HOST);

bind(server_socket, (struct sockaddr *) &server_addr, sizeof(server_addr));
```

Za pomocą funkcji *listen* przygotowujemy socket do odbierania zgłoszeń i dla każdego zaakceptowanego połączenia tworzony jest nowy socket dla klienta. Klient nie jest przyjmowany, jeśli liczba aktualnie zalogowanych użytkowników przekracza zdefiniowaną wcześniej maksymalną dopuszczalną ilość klientów. Dzięki zastosowaniu współbieżności, wątki mogą równolegle obsługiwać sockety zalogowanych użytkowników.

```
listen(server_socket,50)

client_socket = accept(server_socket, (struct sockaddr *) &server_storage, &addr_size);

pthread_create(&thread_id, NULL, socketThread, (void*)client);
```

Komunikacja z klientem odbywa się za pomocą poniższych funkcji. Deskryptor klienta przechowywany jest w strukturze, która zostanie opisana później. Wiadomość musi zostać przekazana w odpowiedniej formie. Mianowicie najważniejsze elementy powinny być oddzielone określonym znakiem (*delimiter*). Serwer analizuje pierwszy istotny element takiej wiadomości (m.in. *LOGIN, LOGOUT, MSG*) i na jego podstawie wykonuje dalsze czynności.

```
recv(client->sockfd, client_message, MSG_SIZE, 0);
token = strtok(client_message, delimiter); // key word
send(client->sockfd, message, strlen(message), 0);
```

Aby uniknąć równoczesnego dostępu do współdzielonego zasobu, przy każdej wymagającej tego funkcji, stosujemy *mutex*:

```
pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_lock(&clients_mutex);
...
pthread_mutex_unlock(&clients_mutex);
```

Na koniec oczywiście stosujemy funkcję *close*, aby zamknąć gniazdo i usunąć jego deskryptor.

```
close(client->sockfd);
```

Kod klienta oraz graficzny interfejs użytkownika zaimplementowane zostały w języku Python.

Po stronie klienta na początku także tworzone jest jego gniazdo, po czym następuje próba połączenia z odpowiednim adresem serwera. Komunikacja odbywa się za pomocą funkcji recv oraz send. W celu poprawnego odczytu wiadomości muszą zostać zakodowane przy wysyłaniu oraz odkodowane po odebraniu.

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((host, port))

data = client_socket.recv(BUFFER_SIZE).decode("unicode_escape")
client_socket.send(bytes(message,"utf-8"))

client_socket.close()
```

Klient wysyła wiadomości w konkretnych sytuacjach, jakimi są m.in. zalogowanie do komunikatora, wysłanie wiadomości do znajomego lub też wylogowanie z aplikacji.

3 Napotkane trudności oraz podsumowanie

Początkowo największą zagadką było dla nas, w jaki sposób serwer ma wiedzieć, do którego klienta ma przekazać wiadomość. Pierwszym, niezbyt prawidłowym pomysłem było wysyłanie wiadomości do wszystkich klientów oraz weryfikacja na poziomie klienta, czy to on powinien otrzymać tę wiadomość. Jednak szybko się z tego wycofaliśmy oraz zaimplementowaliśmy następującą strukturę na serwerze:

```
typedef struct Client{
  struct sockaddr_storage address;
  int sockfd;
  int uid;
  char login[32];
} Client;
```

Oprócz deskryptora socketu, przechowuje dla nas bardzo istotną informację, jaką jest login użytkownika. Teraz na podstawie tych danych jesteśmy dokładnie w stanie stwierdzić, do którego klienta powinna zostać wysłana wiadomość. Stworzyliśmy także listę przechowującą wszystkich klientów, której używamy m.in. do znalezienia deskryptora użytkownika o konkretnym loginie:

```
void send_message(char *s, char* login){
  pthread_mutex_lock(&clients_mutex);

  for(int i = 0; i < MAX_CLIENTS; ++i)
    if(clients[i])
      if(strcmp(clients[i]->login, login) == 0 )
        if(send(clients[i]->sockfd, s, strlen(s), 0) < 0){
          perror("ERROR: Sending failed");
          break;
        }
  pthread_mutex_unlock(&clients_mutex);
}
```

Serwer można uruchomić na terminalu z jądrem Linux (np. WSL, WSL2) za pomocą komendy:

```
gcc -g -Wall -pthread serverTCPmultithread.c -lpthread -o server && ./server
```

Klienta można uruchomić na systemie Windows z odpowiednio skonfigurowanym pakietem PyQt5 za pomocą komendy:

```
python client.py
```

Ostatecznie wszystko działa bardzo sprawnie. Wiadomości przesyłane są od razu do prawidłowych użytkowników (zalogowanych lub też nie).

Rejestracja użytkowników odbywa się z odpowiednią weryfikacją. Długości podawanych danych nie mogą być ani za krótkie, ani za długie. Jeśli podany login jest już zajęty, zostanie ukazany odpowiedni komunikat. Weryfikacja następuje oczywiście też przy logowaniu. Aby przejść do głównego menu, należy podać login z odpowiednim hasłem.

Na ekranie głównym wyświetlana jest lista znajomych wraz z ich opisami. Za pomocą odpowiedniego przycisku można wyszukać oraz dodać nowego znajomego. Przejście do czatu z konkretnym znajomym następuje po dwukrotnym naciśnięciu na jego dane. Użytkownicy mogą swobodnie prowadzić wiele konwersacji jednocześnie. Dodatkowo mogą ustawić swój opis oraz zmienić go, gdy już im się znudzi.