# PSiW - Protokół

Anna Ogorzałek 151805 Bartosz Nowak 151891

## 1 Komunikacja między serwerem i klientem

### 1.1 Wstęp

Cały komunikacja opiera się wyłącznie na jednej kolejce komunikatów IPC. Klucz jest stałą dostępną dla wszystkich programów. Serwer oczekuje na jakiekolwiek wiadomości, natomiast wysyłając komunikaty do użytkownika, przypisuje do typu konkretne ID użytkownika - czyli PID procesu. Użytkownik odbiera wiadomości oczekując na wiadomość z jego ID.
Serwer przechowuje również PID procesu potomnego użytkownika, który jest odpowiedzialny za odbiór wiadomości od innych użytkowników. Wykorzystanie współbieżności jest uzasadnione tym, że program klienta musi cały czas oczekiwać na przychodzące wiadomości wysłane przez innych użytkowników, a w przypadku zapytań bezpośrednich do serwera robi to tylko w momencie wysłania zapytania.

### 1.2 Polecenia

Przy włączeniu programu, jedyne co może zrobić użytkownik to wprowadzenie nazwy użytkownika i hasła w celu zalogowania się. Komendy z których mogą korzystać użytkownicy, dostępne są dopiero po zalogowaniu się na konto. W celu ich wyświetlenia można użyć komendy: "!help"
Wykonanie każdego polecenia wymaga rozróżnienia tych poleceń przez serwer, dlatego każde żądanie ze strony użytkownika jest wysyłane z konkretnym typem, dla każdego z poleceń.

Komendy razem z funkcjonalnością i typem:

| **polecenie** | **funkcja** | **typ** |
|---|---|---|
| — | loguje użytkownika | LOGIN_TYPE |
| !help | wyświetla dostępne komendy | brak |
| !ulist | wyświetla listę zalogowanych użytkowników | USERS_LIST_TYPE |
| !glist | wyświetla dostępne grupy | GROUP_LIST_TYPE |
| !guser | wyświetla użytkowników w danej grupie | GROUP_LIST_TYPE |
| !gjoin | pozwala na dołączenie do danej grupy | GROUP_JOIN_TYPE |
| !gexit | pozwala na opuszczenie danej grupy | GROUP_EXIT_TYPE |
| !dm | wysyła wiadomość do danego użytkownika | DIRECT_MESSAGE_TYPE |
| !gm | wysyła wiadomość do grupy | GROUP_MESSAGE_TYPE |
| !muteuser | blokuje konkretnego użytkownika | MUTE_USER_TYPE |
| !unmuteuser | odblokowuje użytkownika | UNMUTE_USER_TYPE |
| !mutegroup | blokuje konkretną grupę | MUTE_GROUP_TYPE |
| !unmutegroup | odblokowuje grupę | UNMUTE_GROUP_TYPE |
| !logout | wylogowuje użytkownika i wyłącza program | LOGOUT_TYPE |

### 1.3 Komunikacja

Serwer przechowuje dane o użytkownikach i grupach w następujących strukturach:

```
struct User{
  char name[USERNAME_SIZE];
  char password[PASSWORD_SIZE];
  int id;
  int sendId;
  int isLogin;
  short incorrectLoginAttempts;

  char mutedUsersList[NUM_OF_USERS][USERNAME_SIZE];
  char mutedGroupsList[NUM_OF_GROUPS][GROUPNAME_SIZE];
};

struct Group{
  char name[GROUPNAME_SIZE];
  short usersInGroup;
  struct User *users[NUM_OF_USERS];
};

```

Gdzie:
**id** – identyfikator użytkownika na serwerze (PID), wykorzystywany również do komunikacji serwera z użytkownikiem (adresat -> serwer -> adresat)
**sendId** – PID procesu potomnego, wykorzystywany w komunikacji adresat -> serwer -> odbiorca/grupa
**usersInGroup** – aktualna liczba osób w grupie

Struktura przekazywanych komunikatów jest następująca:

```
struct msgbuf{
  long type;
  int error;
  int receiverId;
  int senderId;
  char message[1024];
};
```

za jej pomocą przekazywane są wszystkie informacje pomiędzy użytkownikiem a serwerem.

#### 1.3.1 Logowanie

Użytkownik wprowadza nazwę użytkownika i hasło, które zostaje przesłane w formie username;password. Serwer po otrzymaniu wiadomości sprawdza czy istnieje taki użytkownik (na podstawie danych odczytanych wcześniej z pliku), następnie sprawdza czy hasło jest poprawne, a także czy użytkownik nie jest już zalogowany, i czy nie przekroczył dostępnych prób logowania. Serwer odsyła zaistniały błąd lub potwierdzenie zalogowania.
**Wymagane: long type, int senderId (PID), int receiverId (child PID), char message[1024]**

#### 1.3.2 Lista użytkowników

Do serwera przesyłana zostaje informacja z ID użytkownika i typem zapytania. Serwer po otrzymaniu wiadomości, przeszukuje wszystkich zalogowanych użytkowników i odsyła informację w formie user1;user2;...;usern, co wyświetlane jest u użytkownika odpowiednio sformatowane.
**Wymagane: long type, int senderId (PID)**

### 1.3.3 Dostępne grupy

Do serwera przesyłana zostaje informacja z ID użytkownika i typem zapytania. Serwer po otrzymaniu wiadomości, przeszukuje wszystkie istniejące grupy i odsyła informację w formie group1;group2;...;groupn co wyświetlane jest u użytkownika odpowiednio sformatowane.
**Wymagane: long type, int senderId (PID)**

### 1.3.4 Użytkownicy danej grupy

Do serwera przesyłana zostaje informacja z ID użytkownika, typem zapytania i nazwą grupy. Serwer po otrzymaniu wiadomości, wyszukuje podaną grupę - odsyłając błąd jeśli nie istnieje. Następnie sprawdza czy w grupie są użytkownicy i jeśli wszystko jest w porządku, odsyła informację w formie user1;user2;...;usern, co wyświetlane jest u użytkownika odpowiednio sformatowane.
**Wymagane: long type, int senderId (PID), char message[1024]**

### 1.3.5 Dołączanie do grupy

Do serwera przesyłana zostaje informacja z ID użytkownika, typem zapytania i nazwą grupy. Serwer po otrzymaniu wiadomości, wyszukuje podaną grupę - odsyłając błąd jeśli nie istnieje. Następnie sprawdza czy w grupie nie ma już tego użytkownika, a potem czy jest jakieś wolne miejsce, do którego może dołączyć użytkownika. Serwer odsyła zaisniały błąd lub potwierdzenie dołączenia do grupy.
**Wymagane: long type, int senderId (PID), char message[1024]**

### 1.3.6 Opuszczanie grupy

Do serwera przesyłana zostaje informacja z ID użytkownika, typem zapytania i nazwą grupy. Serwer po otrzymaniu wiadomości, wyszukuje podaną grupę - odsyłając błąd jeśli nie istnieje. Następnie sprawdza czy użytkownik znajduje się w tej grupie, po czym go z niej usuwa. Serwer odsyła zaistniały błąd lub potwierdzenie usunięcia użytkownika z grupy.
**Wymagane: long type, int senderId (PID), char message[1024]**

### 1.3.7 Wysyłanie wiadomości

Do serwera zostaje wysłana informacja z ID użytkownika - adresata, typem zapytania oraz wiadomość w formacie: nazwa_odbiorcy;treść. Serwer przetwarza otrzymaną wiadomość i gdy nie zostanie wykryty żaden błąd, przekazuje jej treść do odbiorcy - wiadomość ta jest odbierana przez proces potomny odbiorcy. W przypadku niepowodzenia adresat otrzymuje odpowiedni komunikat z treścią błędu, w przeciwnym przypadku otrzymuje potwierdzenie wysłania wiadomości.
Możliwe błędy: próba wysyłania wiadomości do nieistniejącego użytkownika, użytkownik nie jest zalogowany, adresat jest zablokowany przez odbiorcę, próba wysłania wiadomości do samego siebie.
**Wymagane: long type, int senderID, char message[1024]**

### 1.3.8 Wysyłanie wiadomości do grupy

Do serwera zostaje wysłana informacja z ID użytkownika - adresata, typem zapytania oraz wiadomość w formacie: nazwa_grupy;treść. Serwer przetwarza otrzymaną wiadomość i gdy nie zostanie wykryty żaden błąd, przekazuje jej treść do zalogowanych użytkowników w danej grupie. Wiadomość ta jest odbierana przez procesy potomne odbiorców. W przypadku niepowodzenia adresat otrzymuje odpowiedni komunikat z treścią błędu, w przeciwnym przypadku otrzymuje potwierdzenie wysłania wiadomości.

Możliwe błędy: próba wysyłania wiadomości do nieistniejącej grupy.
**Wymagane: long type, int senderID, char message[1024]**

### 1.3.9 Blokowanie użytkownika

Do serwera zostaje wysłana informacja z ID użytkownika - adresata, typem zapytania oraz wiadomość w formacie: nazwa_użytkownika. Serwer przechowuje informację o zablokowaniu użytkownika przez adresata. W przypadku niepowodzenia zablokowania adresat otrzymuje komunikat o błędzie, w przeciwnym przypadku otrzymuje potwierdzenie zablokowania.
Możliwe błędy: próba zablokowania nieistniejącego użytkownika, użytkownik jest już zablokowany, próba zablokowania samego siebie.
Gdy zablokowany użytkownik podejmie próbę wysłania wiadomości - otrzyma odpowiedni komunikat o błędzie.
**Wymagane:long type, int senderID, char message[1024]**

### 1.3.10 Odblokowanie użytkownika

Do serwera zostaje wysłana informacja z ID użytkownika - adresata, typem zapytania oraz wiadomość w formacie: nazwa_użytkownika. Serwer usuwa informację o zablokowaniu użytkownika i przywraca możliwość komunikacji zablokowanemu użytkownikowi z adresatem. W przypadku niepowodzenia adresat otrzymuje komunikat o błędzie, w przeciwnym przypadku otrzymuje potwierdzenie odblokowania.
Możliwe błędy: próba odblokowania nieistniejącego użytkownika, próba odblokowania użytkownika, który nie był zablokowany, próba odblokowania samego siebie.
**Wymagane:long type, int senderID, char message[1024]**

### 1.3.11 Blokowanie grupy

Do serwera zostaje wysłana informacja z ID użytkownika - adresata, typem zapytania oraz w wiadomość w formacie: nazwa_grupy. Serwer przechowuje informacje o zablokowaniu grupy przez adresata. W przypadku niepowodzenia zablokowania adresat otrzymuje komunikat o błędzie, w przeciwnym przypadku otrzymuje potwierdzenie zablokowania.
Możliwe błędy: próba zablokowania nieistniejącej grupy, grupa jest już zablokowana.
W przypadku wysłania wiadomości do zablokowanej grupy, adresat zostanie zignorowany i wiadomość nie zostanie mu dostarczona.
**Wymagane:long type, int senderID, char message[1024]**

### 1.3.12 Odblokowanie grupy

Do serwera zostaje wysłana informacja z ID użytkownika - adresata, typem zapytania oraz wiadomość w formacie: nazwa_grupy. Serwer usuwa informację o zablokowaniu grupy przez adresata. W przypadku niepowodzenia adresat otrzymuje komunikat o błędzie, w przeciwnym przypadku otrzymuje potwierdzenie odblokowania.
Możliwe błędy: próba odblokowania nieistniejącej grupy, próba odblokowania grupy, która nie była zablokowana
**Wymagane: long type, int senderID, char message[1024]**

### 1.3.13 Wylogowanie użytkownika

Do serwera przesyłana zostaje informacja z ID użytkownika, typem zapytania i nazwą grupy. Serwer usuwa użytkownika z aktywnych użytkowników i wysyła potwierdzenie wykonania żądania.
**Wymagane: long type, int senderId (PID)**