IPC Publish-Subscribe

Protokół projektu | Programowanie Systemowe i Współbieżne

Autorzy:

- Tomasz Pawłowski 155965
- Jakub Kamieniarz 155845

Opis funkcjonalności

Pomocnicze struktury danych

```
enum MessageType {
  Login = 1,
  Subscription = 2,
  NewTopic = 3,
  SendMessage = 4,
  BlockUser = 5,
  AvailableTopics = 6,
  TopicsNumber = 7,
  TopicsRequest = 8,
};

enum SubscriptionType {
  Permanent = 0,
  Temporary = 1,
  Unsubscribed = 2,
  OversubscribedTopic = 3
};

typedef struct {
  char name[128]; // Nazwa klienta
  int msgid;      // Identyfikator kolejki odbiorczej klienta
} Client;         // Wewnętrzna reprezentacja klienta po stronie serwera

typedef struct {
  int topic_id;         // Identyfikator tematu
  char topic_name[128]; // Nazwa tematu
} Topic;                // Reprezentacja tematu przekazywana do klienta

typedef struct {
  int topic_id;                       // Identyfikator tematu
  char topic_name[128];               // Nazwa tematu
  unsigned int number_of_subscibers; // Liczba klientów subskrybujących temat
  struct {
    Client *subscriber;    // Tablica klientów subskrybujących temat
    SubscriptionType type; // Informacje o subskrybcjach danych klientów
    int duration;          // Pozostały czas trwania subskrybcji przejściowej
  } subscriber_info[16];
} Topic_; // Wewnętrzna reprezentacja tematu po stronie serwera

Message messages[1024]; // Wiadomości przechowywane przez serwer
Client Logged_in[16];   // Zalogowani klienci
Topic_ topics[128];     // Dostępne tematy
```

Logowanie

- do kolejki komunikatów serwera klient powinien wysłać komunikat w formie:

```
typedef struct {
  long type;      // Typ komunikatu
  char name[128]; // Nazwa klienta
  int msgid;      // Identyfikator kolejki odbiorczej klienta
} LoginMessage;
LoginMessage m_login_user = {Login, "...", 0x123};
```

- pole `name` powinno być unikalne dla każdego klienta, ograniczone jest do 127 znaków ASCII
- pole `msgid` powinno zawierać identyfikator kolejki, gdzie będą wysyłane komunikaty klienta
- uruchomienie programu serwera podaje klucz kolejki komunikatów na ekranie do której należy wysłać komunikat
- serwer po otrzymaniu wiadomości weryfikuje podane informacje oraz wysyła potwierdzenie logowania:
  - w przypadku poprawnego logowania:
    - dodaje klienta do tablicy zalogowanych klientów:

      ```
      struct Client {
        char name[128]; // Nazwa klienta
        int msgid;      // Identyfikator kolejki odbiorczej klienta
      } Logged_in[16];
      ```

    - następnie wysyła komunikat formatu:

      ```
      typedef struct {
        long type;      // Typ komunikatu
        char name[128]; // Nazwa klienta
        int status;     // Stan logowania: 0 - błąd, 1 - ok
      } LoginStatus;
      ```

    - o treści:

      ```
      LoginStatus m_login_proper = {.type = Login, .name = "...", .status = 1};
      ```

  - w przeciwnym przypadku wysyła komunikat powyższego formatu o treści:

    ```
    LoginStatus m_login_error = {.type = Login, .name = "...", .status = 0};
    ```

Rejestracja odbiorcy (Subskrybcja tematu)

- do kolejki komunikatów serwera klient wysyła komunikat w formacie:

```
typedef struct {
  long type;                 // Typ komunikatu
  char name[128];            // Nazwa klienta
  int topic_id;              // Identyfikator tematu
  enum SubscriptionType sub; // Rodzaj subskrybcji
  int duration;              // Długość trwania subskrybcji
} SubscriptionMessage;
```

  - dla subskrypcji przejściowej (np. temat 123 na 4 wiadomości)

    ```
    SubscriptionMessage m_sub_temp = {Subscription, "...", 123, Temporary, 4};
    ```

  - dla subskrybcji trwałej (np. temat 123), pole `duration` ignorowane

    ```
    SubscriptionMessage m_sub_perm = {Subscription, "...", 123, Permanent};
    ```

- Serwer przechowuje informacje o klientach subskrybujących dany temat w tablicy:

```
typedef struct {
  int topic_id;                       // Identyfikator tematu
  char topic_name[128];               // Nazwa tematu
  unsigned int number_of_subscibers; // Liczba klientów subskrybujących temat
  struct {
    Client *subscriber;     // Wskaźnik do klienta subskrybującego temat
    SubscriptionType type; // Informacje o subskrybcjach danych klientów
    int duration;          // Pozostały czas trwania subskrybcji przejściowej
  } subscriber_info[16];
} Topic_; // Wewnętrzna reprezentacja tematu po stronie serwera
Topic_ topics[128]
```

- Maksymalna liczba klientów subskrybujących temat to 16. Dopuszczalne jest istnienie maksymalnie 128 tematów.
- Jeśli dany klient posiada już subskrypcję tematu: (obecna → wysłana)
  - `Temporary` → `Temporary`: przedłużenie o `duration`, bądź skrócenie dla ujemnych wartości
  - `Temporary` → `Permanent`: zamiana na `Permanent`
  - `Permanent` → `Permanent`: brak zmian
  - `Permanent` → `Temporary`: zamiana na `Temporary` o długości `duration`
- Aby odsubskrybować temat (np. 123) należy wysłać komunikat:

```
SubscriptionMessage m_unsub = {Subscription, "...", 123, Unsubscribed};
```

- Serwer po otrzymaniu wiadomości odsyła informację zwrotną, która odzwierciedla obecny stan subskrypcji tematu:

```
typedef SubscriptionMessage SubscriptionStatus;
SubscriptionStatus m_sub_stat = {Subscription, "...", 123, Unsubscribed, -1};
```

- W przypadku gdy dany temat jest zasubskrybowany przez 16 klientów, to kolejne zapytania o subskrybcję zostaną odrzucone poprzez wiadomość:

```
SubscriptionStatus m_oversub_stat = {Subscription, "...", 123, OversubscribedTopic, -1};
```

  Wiadomości zawierające `{.sub=OversubscribedTopic}` wysłane do serwera są ignorowane.

Rejestracja typu wiadomości (tematu)

- do kolejki komunikatów serwera klient wysyła komunikat:

```
typedef struct {
    long type;            // Typ komunikatu
    char name[128];       // Nazwa klienta
    char topic_name[128]; // Nazwa tematu
} NewTopicMessage;
NewTopicMessage m_topic = {NewTopic, "...", "..."};
```

- Nazwa tematu powinna być unikatowa dla każdego tematu, składa się wyłącznie ze znaków ASCII o maksymalnej długości 127 znaków.
- następnie serwer sprawdza czy dany temat już istnieje
  - do klienta odsyłany jest komunikat o formacie:

```
typedef struct {
  long type;            // Typ komunikatu
  int topic_id;         // Identyfikator tematu = 0
```

```
  char topic_name[128]; // Nazwa tematu
} NewTopicStatus;
```

- jeśli tak, to:

  ```
  NewTopicStatus m_top_stat_error = {NewTopic, 0, "..."};
  ```

- jeśli nie, to do klienta odsyłany jest komunikat (poprzez wysłanie komunikatu do kolejki):

  ```
  NewTopicStatus m_top_stat_proper = {NewTopic, 123, "..."};
  ```

Rozgłoszenie nowej wiadomości

- Klient wysyła komunikat do serwera:

```
typedef struct {
  long type;       // Typ komunikatu
  int topic_id;    // Identyfikator tematu
  char name[128];  // Nazwa klienta
  char text[2048]; // Treść wiadomości
  int priority;    // Priorytet wiadomości
} Message;
Message m_message = {SendMessage, 123, "...", "Hello, World!", 7};
```

- Wiadomości są zapisywane w globalnej tablicy `Message messages[1024]`.
- Następnie serwer rozsyła wiadomość do odpowiednich subskrybentów tematu o ile autor wiadmości nie został wcześniej zablokowany przez danego subskybenta.

Odbiór wiadomości w sposób synchroniczny (blokujący)

- Wiadomości kontrolne:
  - status logowania,
  - status subskrybcji
  - status rejestracji nowego tematu
  - status zablokowania użytkownika

Odbiór wiadomości w sposób asynchroniczny

- wiadomości odbierane przez serwer
- wiadomości dotyczące tematu subskrybowanego przez użytkownika

Zablokowanie użytkownika

- W celu zablokowanie użytkownika klient powienien wysłać wiadomość o następującym formacie danych:

```
typedef struct {
  long type;            // Typ komunikatu
  char name[128];       // Nazwa blokującego klienta
  char block_name[128]; // Nazwa blokowanego klienta
  int topic_id;         // Identyfikator tematu
} BlockUserMessage;
```

- Aby zablokować użytkownika `"blockMe"` na dany temat `123` należy wysłać wiadomość o treści:

  ```
  BlockUserMessage m_block_user_by_topic = {BlockUser, "...", "blockMe", 123};
  ```

- Aby zablokować użytkownika `"blockMe"` na każdy temat należy wysłać wiadmość o treści:

  ```
  BlockUserMessage m_block_global = {BlockUser, "...", "blockMe", 0};
  ```

Lista dostępnych tematów - dodatkowo

- W celu otrzymania listy dostępnych tematów klient powienien wysłać wiadomość o formacie i treści:

```
typedef struct {
  long type;      // Typ komunikatu
  char name[128]; // Nazwa klienta
} TopicsRequestMessage;
TopicsRequestMessage m_request_topics = {TopicsRequest, "..."};
```

- Następnie serwer po otrzymaniu zapytania o listę tematów odeśle dwie wiadomości:
  - Odpowiedź 1.

```
typedef struct {
  long type;                      // Typ komunikatu
  unsigned long number_of_topics; // Liczba dostępnych tematów
} TopicsNumberMessage;
TopicsNumberMessage m_n_topics = {TopicsNumber, 2};
```

  - Odpowiedź 2.

```
typedef struct {
  long type;      // Typ komunikatu
  Topic topics[]; // Tablica Tematów
} AvailableTopicsMessage;

AvailableTopicsMessage m_available_topics = {
    .type = AvailableTopics,
    .topics = {{1, "Topic1"}, {2, "Topic2"}}};
```

TODO:

- ☐ opisz lepiej sposób przesyłania wiadomości przez osobne kolejki klientów i serwera
- ☐ msgid to identyfikator kolejki komunikatów, ale nwm co z key, chyba można użyć `IPC_PRIVATE`
- ☐ klient mógłby przesyłać nazwę, jak jest ok, to serwer zwraca `id`, którym klient musi się dalej posługiwać, potem np. tematy przechowują tylko `id` klienta zamiast wskaźnika
- ☐ dla subskrybcji stałej pole `duration` wypełnione 0
- ☐ przechowywanie info o zablokowanych klientach
- ☐ rozdzielenie `Topic_` na dwie struktury:
  1. `topic_id`, `topic_name` i `id` klientów subskrybujących,
  2. `id` klienta, `topic_id`, sposób subskrybcji `sub`, sync czy async `sub_sync`, blokowani klienci `blocked_id`
- ☐ lista dostępnych tematów tylko druga odp z założeniem że n=max_liczba_tematów, sama liczba tematów w drugiej wiadomości
- ☐ chyba w protokole założyłem domyślnie (trzeba dopisać), że wszystkie wiadomości są odbierane asynchronicznie, nie implementowałbym tej drugiej wersji jak nie trzeba
- ☐ **dodaj sub synchroniczny / asynchroniczny**