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,
  ReadMessages = 6,
  MessageReadCount = 7,
  AsyncMessage = 8,
};

enum SubscriptionType {
  Unsubscribed = 0,
  PermanentAtRequest = 1,
  PermanentAsSoonAsReceived = 3,
  TemporaryAtRequest = 2,
  TemporaryAsSoonAsReceived = 4,
  Oversubscribed = 8,
  UnknownTopic = 9,
};
```

```
typedef struct {
  char name[MAX_USERNAME_LENGTH]; // Nazwa klienta
  int client_id;                  // ID klienta
  int queue;                      // Identyfikator kolejki odbiorczej klienta
} Client;

typedef struct {
  int client_id;              // Identyfikator użytkownika
  int topic_id;               // Identyfikator tematu
  enum SubscriptionType type; // Informacje o subskrybcjach danych klientów
  int duration;               // Długość subskrybcji przejściowej
  int blocked_ids[MAX_BLOCKED_USERS]; // Zablokowani użytkownicy
} SubInfo;                            // Struktura informacji o subskrybentach

typedef struct {
  int topic_id;                     // Identyfikator tematu
  char topic_name[MAX_TOPIC_LENTH]; // Nazwa tematu
} Topic;

typedef struct {
  long type;                    // Typ komunikatu
  int message_id;               // Identyfikator wiadomości
  int topic_id;                 // Identyfikator tematu
  int client_id;                // ID klienta
  char text[MAX_MESSAGE_LENGTH]; // Treść wiadomości
  int priority;                 // Priorytet wiadomości
} Message;
```

```
Client logged_in[MAX_CLIENTS];                  // Zalogowani klienci
int n_logged = 0;                               // Liczba zalogowanych klientów
Topic topics[MAX_TOPICS];                       // Tablica dostępnych tematów
int n_topics = 0;                               // Liczba dostępnych tematów
Message messages[N_PRIORITIES][MAX_MESSAGES]; // Tablica wysłanych wiadomości
int n_messages[N_PRIORITIES] = {0,0,0};         // Liczba wiadomości w systemie
SubInfo subscriptions[MAX_SUBSCRIPTIONS];       // Informacje o subskrybcjach
```

Domyślne wartości parametrów programu

```
#define MAX_CLIENTS 16
#define MAX_TOPICS 128
#define MAX_MESSAGES 1024
#define MAX_SUBSCRIPTIONS 128
#define MAX_USERNAME_LENGTH 127
#define MAX_TOPIC_LENTH 127
#define MAX_BLOCKED_USERS 16
#define MAX_MESSAGE_LENGTH 2047
#define N_PRIORITIES 3
#define INTERRUPT_INPUT 0
```

Architektura systemu

- Komunikacja między procesami klientów odbywa się pośrednio poprzez:
  1. Wysłanie odpowiedniej wiadomości do kolejki komunikatów serwera przez nadawcę.
  2. Zapisanie odpowiednich informacji przez jednakowy dla każdego klienta serwer.
  3. Wysłanie żądania odczytu wiadomości przez klienta do kolejki komunikatów serwera (synchroniczny sposób odbioru wiadomości).
  4. Przetworzenie żądania przez serwer, który wysyła wiadomości tekstowe utworzone w danym temacie uwzględniając preferencje: długości subskrybcji, zablokowanych nadawców oraz priorytet wiadomości.
- Serwer oraz klienci posiadają swoje własne kolejki wiadomości, z których wyłącznie odczytują odpowiednie wiadomości. Komunikacja między klientem a serwerem polega na wysłaniu wybranej wiadomości do kolejki komunikatów serwera, skąd po przetworzeniu może zostać wysłana informacja zwrotna do kolejki komunikatów klienta. Bezpośrednia komunikacja między klientami nie jest możliwa.
- Wszystkie wymagane dane są wprowadzane przez standardowe wejście oraz wyświetlane na standardowym wyjściu. Parametry wielkości struktrur danych nie są konfigurowalne od czasu uruchomienia programu i muszą być jednolite w całym systemie.

Inicjalizacja klienta i logowanie

- Program serwera po uruchomieniu podaje `key_t server_key` kolejki komunikatów serwera.
- Po uruchomieniu programu klienta należy podać powyższy klucz w celu zainicjalizowania komunikacji. Przydział `key_t client_key` kolejek komunikatów klientów następuje w sposób automatyczny.
- W celu zalogowania się klient powienien wysłać do kolejki komunikatów serwera komunikat w formie:

  ```
  typedef struct {
    long type;                          // Typ komunikatu
    char name[MAX_USERNAME_LENGTH+1]; // Nazwa klienta
    key_t queue_key;                    // Identyfikator kolejki odbiorczej klienta
  } LoginMessage;
  LoginMessage m_login_user = {Login, "...", 0x123};
  ```

- Pole `name` powinno być unikalne dla każdego klienta, ograniczone jest do `MAX_USERNAME_LENGTH` znaków ASCII (ostatni znak `\0`).
- Pole `queue_key` powinno zawierać `key_t client_key`, gdzie będą wysyłane komunikaty klienta. Jego wartość jest generowana automatycznie.

- Serwer po otrzymaniu wiadomości weryfikuje podane informacje oraz wysyła potwierdzenie logowania:
  - w przypadku poprawnego logowania:
    - dodaje klienta do tablicy zalogowanych klientów `logged_in` o type `Client` zawierającej maksymalnie `MAX_CLIENTS` zalogowanych użytkowników:

      ```
      typedef struct {
        char name[MAX_USERNAME_LENGTH+1]; // Nazwa klienta
        int client_id;                    // ID klienta
        int queue;                        // Identyfikator kolejki odbiorczej klienta
      } Client;
      Client logged_in[MAX_CLIENTS];
      ```

      Pole `id > 0` jest przydzielone klientowi przez serwer.
    - następnie wysyła komunikat formatu:

      ```
      typedef struct {
        long type;                        // Typ komunikatu
        char name[MAX_USERNAME_LENGTH+1]; // Nazwa klienta
        int status;                       // Stan logowania: 0 - błąd
        int id;                           // ID klienta
      } LoginStatus;
      ```
    - o treści:

      ```
      LoginStatus m_login_proper = {.type = Login, .name = "...", .status = 1, .id=...};
      ```
  - w przeciwnym przypadku wysyła komunikat powyższego formatu o treści:

    ```
    LoginStatus m_login_error = {.type = Login, .name = "...", .status = 0, .id=-1};
    ```

Rejestracja odbiorcy (Subskrybcja tematu)

- Do kolejki komunikatów serwera klient wysyła komunikat w formacie:

  ```
  typedef struct {
  long type;                  // Typ komunikatu
  int client_id;              // ID klienta
  int topic_id;               // Identyfikator tematu
  enum SubscriptionType sub; // Rodzaj subskrybcji
  int duration;               // Długość trwania subskrybcji
  } SubscriptionMessage;
  ```
  - dla subskrypcji przejściowej, gdzie wiadomości są przesyłane po wysłaniu zapytania (w sposób synchroniczny), np. temat 123 na 4 wiadomości

    ```
    SubscriptionMessage m_tmp_s = {Subscription, ..., 123, TemporaryAtRequest, 4};
    ```
  - dla subskrypcji przejściowej, gdzie wiadomości są przesyłane natychmiast (w sposób asynchroniczny), np. temat 123 na 4 wiadomości

    ```
    SubscriptionMessage m_tmp_a = {Subscription, ..., 123, TemporaryAsSoonAsReceived, 4};
    ```
  - dla subskrybcji trwałej synchronicznej (np. temat 123), pole `duration` nie jest wykorzystane

    ```
    SubscriptionMessage m_per_s = {Subscription, ... , 123, PermanentAtRequest, 0};
    ```
  - dla subskrybcji trwałej asynchronicznej (np. temat 123), pole `duration` nie jest wykorzystane

    ```
    SubscriptionMessage m_per_a = {Subscription, ... , 123, PermanentAsSoonAsReceived, 0};
    ```
- Serwer przechowuje informacje o subskrybcjach klientów na dany temat w tablicy:

  ```
  typedef struct {
    int client_id;                    // Identyfikator użytkownika
    int topic_id;                     // Identyfikator tematu
    enum SubscriptionType type;       // Informacje o subskrybcjach danych klientów
    int duration;                     // Czas trwania subskrybcji przejściowej
  ```

```
  int blocked_ids[MAX_BLOCKED_USERS]; // Zablokowani użytkownicy
} SubInfo;                            // Struktura informacji o subskrybentach
SubInfo subscriptions[MAX_SUBSCRIPTIONS];
```

- Maksymalnie w systemie może być jednocześnie `MAX_SUBSCRIPTIONS` subskrybcji.
- Klient może posiadać co najwyżej jedną subskrybcję danego tematu.
- Jeśli dany klient posiada już subskrypcję tematu: (obecna → wysłana)

  1. W zależności od długości subskrybcji:

  – `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`

  2. W zależności od sposobu przesyłania wiadomości:

  – `*AsSoonAsReceived` → `*AtRequest`: zamiana asynchronicznej w synchroniczną
  – `*AtRequest` → `*AsSoonAsReceived`: zamiana synchronicznej w asynchroniczną

- Aby odsubskrybować temat (np. 123) należy wysłać komunikat:

  ```
  SubscriptionMessage m_unsub = {Subscription, ..., 123, Unsubscribed, -1};
  ```

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

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

- Jeśli w systemie osiągnięto limit subskrybcji to wysyłana jest wiadomość:

  ```
  SubscriptionStatus m_oversub_stat = {Subscription, ..., 123, Oversubscribed, -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
    int client_id;                      // ID klienta
    char topic_name[MAX_TOPIC_LENTH + 1]; // Nazwa tematu
  } NewTopicMessage;
  NewTopicMessage m_new_topic = {.type = NewTopic,...,"..."};
  ```

- Nazwa tematu powinna być unikatowa dla każdego tematu, składa się wyłącznie ze znaków ASCII o maksymalnej długości `MAX_TOPIC_LENTH` 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[MAX_TOPIC_LENTH + 1]; // Nazwa tematu
    } NewTopicStatus;
    ```

  – jeśli temat istnieje, to odsyłany jest komunikat:

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

  – jeśli nie, to do klienta odsyłany jest komunikat:

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

    Pole `topic_id > 0` zostało wygenerowane przez serwer.

- Serwer zapisuje tematy w tablicy:

```
typedef struct {
  int topic_id;                          // Identyfikator tematu
  char topic_name[MAX_TOPIC_LENTH + 1]; // Nazwa tematu
} Topic;
Topic topics[MAX_TOPICS];
```

- W systmie może istnieć maksymalnie `MAX_TOPICS` tematów o długości tytułu co najwyżej `MAX_TOPIC_LENGTH`.

Rozgłoszenie nowej wiadomości

- Klient wysyła komunikat do serwera:

```
typedef struct {
  long type;                          // Typ komunikatu
  int message_id;                     // Identyfikator wiadomości
  int topic_id;                       // Identyfikator tematu
  int client_id;                      // ID klienta
  char text[MAX_MESSAGE_LENGTH + 1]; // Treść wiadomości
  int priority;                       // Priorytet wiadomości
} Message;
Message m_text = {.type=SendMessage, ...};
```

- Pole `message_id` nie jest wykorzystywane przy tworzeniu wiadomości. Służy do identyfikacji wiadomości przez serwer przy rozsyłaniu.
- Wiadomości są zapisywane w globalnej tablicy `Message messages[N_PRIORITIES][MAX_MESSAGES]` w zależności od priorytetu.
- W zależności od preferencji subskrybcji danego klienta:
  - Subskrybcja synchroniczna `*AtRequest`: do czasu przesłania zapytania o przesłanie nowych wiadomości nie są one przesyłane dalej.
  - Subskrybcja asynchroniczna `*AsSoonAsReceived`: wiadomości są przesyłane natychmiast do klienta po pojawieniu się w systemie.

Odbiór wiadomości w sposób synchroniczny

- W celu otrzymania wiadomości napisanych w tematach subskrybowanych przez danego klienta należy wysłać wiadomość:

```
typedef struct {
  long type;     // Typ komunikatu
  int client_id; // ID klienta
  int priority;  // priorytet wiadomości
  int last_read; // ID ostatniej odczytanej wiadomości
} ReadMessage;
ReadMessage m_read = {.type=ReadMessages,...};
```

- Użytkownik powinien podać priorytet wiadomości od którego wyświetlone zostaną wiadomości.
- Serwer w odpowiedzi na powyższe zapytanie odpowiada wysyłając:
  - na początek liczbę wiadomości spełniających podane kryteria:

    ```
    typedef struct {
    long type;   // Typ komunikatu
      int count; // Liczba wiadomości spełniających kryteria
    } MessageCount;
    MessageCount m_count = {.type = MessageReadCount,...};
    ```

  - następnie serwer wysyła wiadomości jedna po drugiej, uzupełnia również pole `message_id`:

    ```
    Message m_text = {.type=SendMessage,.message_id=...,...};
    ```

Odbiór wiadomości w sposób asynchroniczny

- Po zasubskrybowaniu tematu w sposób asynchroniczny serwer może rozesłać wiadomość do klienta, gdy tylko serwer otrzyma nową wiadomość na ten temat.
- Program klienta cały czas sprawdza w tle w swojej kolejce wiadomości czy nie pojawiły się nowe wiadomości w formacie:

  ```
  Message m_async_text = {.type=AsyncMessage,...};
  ```

- Wiadomości są wysyłane i odbierane natychmiast, jednak ich wyświetlenie na ekranie może zostać opóźnione w zależności od parametru `INTERRUPT_INPUT`:
  - dla `INTERRUPT_INPUT=0`: wyświetlenie wiadomości nigdy nie przerywa wpisywania danych przez standardowe wejście.
  - dla `INTERRUPT_INPUT=1`: wyświetlenie wiadomości może nastąpić podczas wpisywania danych przez standardowe wejście. Wpisywany tekst pozostaje ciągły, przykładowo w sytuacji:

    ```
    > abc                 // <- wpisywanie przez standardowe wejście
    ---Nowa wiadomość--- // <- pojawia się nowa wiadomość
    ...
    --------------------
    def\n                 // <- kontynuacja wpisywania do znaku \n
    ```

    zostaje zinterpretowane jako `abcdef`.

Zablokowanie użytkownika

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

```
typedef struct {
  long type;     // Typ komunikatu
  int client_id; // ID blokującego klienta
  int block_id;  // ID blokowanego klienta
  int topic_id;  // Identyfikator tematu
} BlockUserMessage;
```

- Aby zablokować użytkownika o ID `321` na dany temat `123` należy wysłać wiadomość o treści:

  ```
  BlockUserMessage m_block_user_by_topic = {BlockUser, ..., 312, 123};
  ```

- Aby zablokować użytkownika o ID `321` na każdy temat należy wysłać wiadmość o treści:

  ```
  BlockUserMessage m_block_global = {BlockUser, ..., 321, 0};
  ```

  Zostawiając pole `{.topic_id = 0}`