Protokół komunikacyjny klienci-serwer

Spis treści:

1. Informacje ogólne:

- Protokół opiera się na jednej kolejce głównej o numerze 0x660 i dodatkowych kolejkach, po jednej na klienta, których numery to 0x1000 + cid, gdzie cid to numer id danego klienta.
- Program klienta dzieli się na dwa procesy, gdzie macierzysty obsługuje większość programu, a potomny czeka w pętli na wiadomości z subskrypcji trwałych lub komunikaty o błędach (proces macierzysty korzysta z kolejki 0x660, a potomny z 0x1000 + cid)
- Maksymalna liczba zalogowanych klientów to 25.

2. Struktura komunikatu:

```
struct msgbuf
{
  long type;
  char message[50];
  char name[50];
  int option;
  int tid;
  int cid;
  int sub_type;
  int ms_number;
  int priority;
}
```

- type - nie pełni roli (po części jego rolę pełni parameter option).
- message - zawiera treść wiadomości, zarówno przy wysyłaniu wiadomości od klienta do serwera (kolejka 0x660), jak i od serwera do klienta (kolejka 0x660 przy subskrybcji przejściowej lub 0x1000 + cid, przy subskrybcji trwałej). Służy też do wysyłania treści komunikatu o błędzie w przypadku nieprawidłowej akcji (kolejka 0x1000 + cid).
- name - zawiera nazwę klienta.
- option - obecnie wybrana opcja, na podstawie której serwer wybiera co ma robić.

- tid - id wątku [1], którego dotyczy dana akcja np. przy zakładaniu nowego wątku itp. Przy odbiorze wiadomości z kolejki 0x1000 + cid zmienna może przybierać wartość 0, która oznacza, że przyszedł komunikat o błędzie, a nie nowa wiadomość.
- cid - id klienta, który wysyła komunikat.
- sub_type - rodzaj subskrypcji przy rejestracji do subskrybcji wątku, gdzie 1 odpowiada przejściowej, a 2 trwałej.
- ms_number - liczba wiadomości, które zostaną otrzymane przy subskrybcji przejściowej.
- priority - priorytet wysyłanej wiadomości.

3. Typy komunikatów

opis	**używane parametry**	**kolejka**	**nadawca**
logowanie	option=0, cid, name	0x660	klient
rejestracja wątku	option=1, tid	0x660	klient
informacja o nowym wątku	cid, tid	0x1000 + cid	serwer
subskrypcja wątku	option=2, cid, tid, sub_type, ms_number	0x660	klient
wysłanie wiadomości	option=3, cid, tid, message, priority	0x660	klient
odbiór wiadomości	option=4, cid, tid	0x660	klient
wysłanie synchroniczne	cid, message	0x660	serwer
wysłanie asynchroniczne	cid, tid, message	0x1000 + cid	serwer
wysłanie treści błędu	cid, tid, message	0x1000 + cid	serwer

4. Obsługa błędów

Serwer reaguje na poniższe, nieprawidłowe akcje ze strony klienta:

- Rejestracja wątku, który już istnieje.
- Rejestracja do subskrybcji wątku, który nie istnieje.
- Rejestracja do subskrybcji wątku, który już subskrybuje.
- Odebranie wiadomości z wątku, którego nie subskrybuje.

[1] Program skojarzył mi się z forami internetowymi, stąd w nim i tutaj mowa o wątkach, a nie typach wiadomości

- Odebranie wiadomości z wątku, który nie istnieje.
- Wysłanie wiadomości do wątku, który nie istnieje.

Klient może za to wysyłać wiadomości do wątków, których sam nie subskrybuje i odbiera swoje wiadomości z subskrybowanych wątków (w treści projektu nie było sprecyzowane, czy tak ma być, czy nie).
Uwaga! Serwer nie obsługuje zalogowania dwóch klientów o tym samym id i należy wtedy zacząć całe działanie programów od początku.

5. Poprawność danych

Id klientów i wątków to liczby pomiędzy 1, a 25 (maksymalnie 25 klientów i wątków).

Wiadomości i nazwy klientów to dowolne ciągi znaków nieprzekraczające 50 liter.

Każdy użytkownik może w sumie odebrać 1000 wiadomości.

Maksymalna liczba odebranych wiadomości przy jednorazowej subskrypcji przejściowej to 99 (po wyczerpaniu wiadomości subskrypcja automatycznie się kończy).

Przy wpisaniu błędnych wartości program pyta o nie ponownie, ale nie działa to idealnie i zdarzają się nieskończone pętle, więc warto wpisywać poprawne wartości od razu.

6. Scenariusze komunikacji

Piszę ten punkt po wstępnej realizacji programu i scenariusze komunikacji odbywają się zgodnie z tabelą typów komunikatów i opisami parametrów w pkt. 2. Chciałbym zamiast nich, poświęcić ten punkt opisowi struktur serwera, ponieważ są one kluczowe, a mogą wydawać się dość nieintuicyjne.

Struktury serwera wyglądają następująco:

```
struct client {
  char name[25];
};

struct message {
  char content[50];
  int thread_id;
  int priority;
};

struct subscriber {
  int subbed_threads[25];
  struct message messages[1000];
  int mi;
```

```
};

struct client clients[25];
struct subscriber subscribers[25];
for (int i = 0; i < 25; i++) {
  memset(subscribers[i].subbed_threads, 0, 25);
}
int threads[25];
memset(threads, 0, 25);
int users[25];
memset(threads, 0, 25);
```

Indeksy czterech list: clients, subscribers, threads i users odpowiadają numerom id klientów/wątków np. w polu 3 w tabeli subscribers przechowywane są dane subskrybenta o id 3. Listy clients, threads i users nie pełnią dużej roli i prawie całe dane przechowywane są w tabeli subscribers.

Struktura subscriber zawiera trzy pola, gdzie drugie jest bezpośrednio połączone z trzecim.

Pole subbet_threads to tablica, która reguluje, które wątki i w jaki sposób subskrybuje dany klient (subscriber). Podobnie jak wcześniej, każdy indeks tej tablicy odpowiada id wątku.
0 na danym indeksie z oznacza, że klient nie subskrybuje wątku o id z.
-1 na danym indeksie z oznacza, że klient subskrybuje wątek o id z w sposób trwały.
Wartość x > 0 na danym indeksie z oznacza, że klient subskrybuje wątek o id z w sposób przejściowy i jednocześnie zostało mu x wiadomości do dostarczenia (przed zakończeniem subskrypcji).
Dzięki takiemu działaniu tej tabeli operacje odbioru wiadomości przez klienta sprowadzają się tylko do niej.

Pole messages zawiera wiadomości z subskrybowanych przejściowo wątków[2]. Każda wiadomość zawiera parametry content (treść wiadomości), thread_id (id wątku, do którego była wysłana) i priority (priorytet wiadomości). Kiedy klient deklaruje chęć odebrania wiadomości z danego wątku serwer przeszukuje całą tablicę i szuka wiadomości o najwyższym priorytecie, która jednocześnie należy do tego wątku. Po znalezieniu nie usuwa jej w żaden sposób, a jedynie zmienia jej priorytet na -1 (dlatego maksymalna liczba wiadomości do każdego klienta to 1000). Pole mi struktury subscriber, to po prostu indeks, na który należy wstawić nową wiadomość.

[2] Warto tutaj zaznaczyć, że klient w żaden sposób nie może odebrać wiadomości wysłanych przed momentem, w którym zasubskrybował dany wątek.