Protokół
Programowanie systemowe i współbieżne projekt
Aleksander Szymaniak 155922

1 Komunikacja

Komunikacja pomiędzy serwerem a klientami odbywa się poprzez jedną kolejkę komunikatów serwera, do której klienci mogą wysyłać polecenia, oraz do 10 kolejek komunikatów dla każdego klienta, które przekazują informacje zwrotne z serwera bądź wiadomości zasubskrybowanego tematu. Informacje zwrotne od serwera mają typ 100 w przypadku informacji do wyświetlenia, 50 w przypadku chęci wyświetlenia wiadomości przez klienta w asynchronicznym trybie odbioru oraz 12 w przypadku chęci zablokowania jakiegoś użytkownika. Regularne wiadomości od użytkowników mają typ odpowiadający ich id.

2. Podstawowe struktury i funkcje (serwer)

Ogólny pomysł przechowywania informacji o kliencie polega na tym, że podczas logowania, ma on przypisywany indeks, wspólny dla wszystkich tablic, które przechowują informacje o użytkownikach. Dzięki temu, jeśli chcemy uzyskać jakieś informacje na temat użytkownika o id np. 0, to wiemy że w każdej tablicy informacja o nim będzie znajdować się pod indeksem 0. Ułatwia to operowanie na danych.

Struktury i funkcje pomocnicze:

```
struct msgbuf{ // struktura wiadomości w kolejce komunikatów
    long type;
    char text[1024];
    int pid;
    int prior;
};

struct info{ // informacja na temat uzytkownika który subskrybuje dany
temat
    int id;
    int how_long;
};

struct name{ // nazwa uzytkownika
    char name[10];
};

struct topic{ // struktura tematu
    struct info użytkownicy[10];
    int created;
};
int pipes[10]; // tablica przechowująca id kolejek komunikatów
poszczególnych klientów
struct name names[10]; // tablica przechowująca nazwy użytkowników
struct topic topics[10]; // tablica, przechowująca tematy oraz
informacje o użytkownikach, którzy je subskrybują
int pid[10] = {-1}; // tablica zawierająca pid użytkowników
```

```
void sendtoall(char text[]){ // wysyłanie wiadomośći do wszystkich
użytkowników
    struct msgbuf msg1;
    msg1.type = 100;
    strcpy(msg1.text, text);
    int i = 0;
    while(pipes[i] != -1){
        msgsnd(pipes[i], &msg1, sizeof(struct msgbuf) - sizeof(long),
IPC_NOWAIT);
        i++;
    }
    return;
};

void sendto(int idc, char text[]){ // wysyłanie wiadomosci do
konkretnego użytkownika
    struct msgbuf msg1;
    msg1.type = 100;
    strcpy(msg1.text, text);
    msgsnd(idc, &msg1, sizeof(struct msgbuf) - sizeof(long),
IPC_NOWAIT);
    return;
};

int id(int p_id){ // funkcja zwracająca id użytkownika podając pid
procesu
    int i = 0;
    while(pid[i] != p_id) i++;
    if(i <= 9) return i;
    return -1;
};
```

3 Funkcjonalność (serwer):
Serwer przyjmuje 6 typów wiadomości, dla każdego przetwarza inny format oraz podejmuje inne działanie:

- **Typ 1** odpowiedzialny jest za logowanie użytkownika. W tym celu klient musi podać swoją nazwę użytkownika która może zawierać dowolne znaki ASCII oraz nie może być dłuższa niż 10 znaków. Przy logowaniu serwer rejestruje pid, nazwę oraz kolejkę komunikatów klienta, dzięki czemu może się z nim komunikować. Bez logowania pozostała funkcjonalność nie występuje. W przypadku (nie)pomyślnego przebiegu logowania klient zostanie poinformowany, wliczając w to powód ewentualnego błędu (próba zalogowania nazwą użytkownika która już istnieje).

  Przykładowe polecenie typu 1:
  `1 Aleksander`

Struktury i funkcje odpowiedzialne za logowanie:

```
case 1: // logowanie
                for(int i = 0; i < 10; i++){ // sprawdzanie czy
 użytkownik czasami już się nie zalogował
                    if(pid[i] == msg.pid){
                        sendto(pipes[i], "Już jesteś zalogowany");
                        carry = 1;
                    }
                }
                if(carry == 0){
                    int idc = msgget(msg.pid, 0666 | IPC_CREAT); //
 wyłuskanie id kolejki klienta
                    sscanf(msg.text, "%10[^\n]", nazwa); // wyłuskanie
 nazwy
                    int a = login(msg.pid, idc, nazwa); //
 zarejestrowanie klienta w serwerze
                    if(a == -1){ // obsługa błędu
                        strcpy(text,"Logowanie nie powiodło się - ta
 nazwa już istnieje\n");
                        sendto(idc, text);
                    }else if(a == 1){ // powodzenie
                        strcpy(text,"Logowanie powiodło się\n");
                        sendto(pipes[id(msg.pid)], text);
                    }
                }
                break;
```

```
int login(int p_id, int idc, char name[]){ // funkcja logująca
 użytkownika
    int i = 0;
    while(pipes[i] != -1){ // sprawdzanie czy nazwa się nie powtarza
        if(strcmp(names[i].name, name) == 0){
            return -1;
        }
        i++;
    }
    //przypisywanie informacji o kliencie
    pipes[i] = idc;
    strcpy(names[i].name, name);
    pid[i] = p_id;
    return 1; };
```

- **Typ 2** odpowiedzialny jest za subskrybowanie tematu. W tym celu klient musi podać kolejno nr tematu, który chce zasubskrybować, tryb subskrybcji (1 - trwała, 2 - przejściowa) oraz opcjonalnie, jeśli wybrał subskrypcję przejściową, ile wiadomości danego tematu chce otrzymać. Serwer weryfikuje różne scenariusze, oraz informuje o ich wystąpieniu, np. przedłużenie subskrypcji, nieudana subskrypcja (użytkownik już subskrybuje w trybie trwałym) lub przejście z subskrypcji przejściowej na trwałą.

  Przykładowe polecenie typu 2:
  `2 1 2 5` (subskrypcja tematu 1, w trybie przejściowym na 5 wiadomości
-

  Struktury i funkcje odpowiedzialne za sybskrybowanie:

```
case 2: // subskrypcja
                carry = 1;
                for(int i = 0; i < 10; i++){ // sprawdzanie czy
użytkownik jest zalogowany
                    if(pid[i] == msg.pid){
                        carry = 0;
                    }
                }
                if(carry == 0){
                    int topicnr;
                    int how;
                    sscanf(msg.text, "%d%d", &topicnr, &how);
                    if(topics[topicnr - 1].created == 0) { //
sprawdzanie czy temat istnieje
                        sendto(pipes[id(msg.pid)], "Podany temat nie
istnieje\n");
                    } else {
                        int how_long = -1;
                        if(how == 2) sscanf(msg.text, "%*d%*d%d",
&how_long); // how_long domyślnie -1 (subskrypcja trwała), chyba że
how == 2, to przypisujemy wartość z wejścia
                        int c = sub(id(msg.pid), topicnr - 1,
how_long);
                        if(c == 1 ){ // pomyślna subskrypcja
                            strcpy(text, "Zasubskrybowano temat ");
                            if(how == 1) sprintf(text, "%s%d%s", text,
topicnr, " w trybie trwałym\n");
                            else sprintf(text, "%s%d%s%d%s", text,
topicnr, " w trybie przejściowym na ", how_long, " wiadomości\n");
                            sendto(pipes[id(msg.pid)], text);
```

```
                        } else if(c != -1) { // przedłużenie
subskrybcji
                            sprintf(text, "%s%d%s%d%s%d%s",
"Przedłużyłeś subskrypcję tematu ", topicnr, " o ", how_long, "
wiadomości, twoja subskrypcja obejmuje jeszcze ", c, " wiadomości\n");
                            sendto(pipes[id(msg.pid)], text);
                        } else { // błąd
                            strcpy(text, "Zasubskrybowałeś już ten
temat\n");
                            sendto(pipes[id(msg.pid)], text);
                        }
                    }
                }
                break;
```

```
int sub(int id, int topicnr, int how_long){ // funkcja subskrybująca
temat dla danego użytkownika
    int issub = 0;
        if(topics[topicnr].użytkownicy[id].id == id){ // jeśli
użytkownik już subskrybował
            issub == 1;
            if(topics[topicnr].użytkownicy[id].how_long == -1) return
-1; // jeśli temat został już zasubskrybowany w trybie trwałym
            else {
                if(how_long == -1) { // zmiana subskrypcji z
przejściowej na trwałą
                    topics[topicnr].użytkownicy[id].how_long = -1;
                    return 1;
                }
                else { // przedłużenie subskrypcji
                    topics[topicnr].użytkownicy[id].how_long +=
how_long;
                    return topics[topicnr].użytkownicy[id].how_long;
                }
            }
        }

    if(issub == 0){ // jeśli użytkownik nie subskrybował
        topics[topicnr].użytkownicy[id].id = id;
        topics[topicnr].użytkownicy[id].how_long = how_long;
    }
    return 1; };
```

- **Typ 3** odpowiedzialny jest za utworzenie tematu. W tym celu klient musi podać tylko numer tematu który chce utworzyć. Numer tematu musi mieścić się w zakresie <1,10>. Jeśli temat istnieje użytkownik będzie o tym poinformowany, jeśli nie, wiadomość o jego utworzeniu otrzymają wszyscy zalogowani klienci.

  przykładowe polecenie typu 3:
  `3 1` (utworzenie tematu nr 1)

  Struktury i funkcje odpowiedzialne za tworzenie tematu:

```
case 3: // tworzenie tematu
                carry = 1;
                for(int i = 0; i < 10; i++){ // sprawdzanie czy
użytkownik jest zalogowany
                    if(pid[i] == msg.pid){
                        carry = 0;
                    }
                }
                if(carry == 0){
                    int newtopic;
                    sscanf(msg.text, "%d", &newtopic);
                    int b = new(msg.pid, newtopic);
                    if(b == -1){ // błąd
                        sprintf(text, "%s%d%s", "Temat ", newtopic, "
juz istnieje\n");
                        sendto(pipes[id(msg.pid)], text);
                    }else if(b == 1){ // powodzenie
                        sprintf(text, "%s%d%s", "Utworzono temat ",
newtopic, "\n");
                        topics[newtopic - 1].created = 1;
                        for(int i = 0; i < 10; i++){
                        }
                        sendtoall(text); // wysyłanie do wszystkich
zalogowanych użytkowników
                    }
                }
                break;
```

```
int new(int pid, int newtopic){ // funkcja tworząca nowy temat
    if(topics[newtopic - 1].created == 0){ // jeśli temat nie istnieje
        topics[newtopic - 1].created = 1;
        return 1;
    }
    else return -1;
};
```

- **Typ 4** odpowiedzialny jest za wysyłanie wiadomości na jakiś temat. W tym celu klient musi podać kolejno numer tematu, priorytet wiadomości, (mniejszy priorytet oznacza ważniejszą wiadomość) oraz treść wiadomości (bezpiecznie do 1000 znaków). Serwer sprawdza czy użytkownik subskrybuje temat, jeśli tak, wysyła wiadomości do odpowiednich użytkowników, jeśli nie, użytkownik zostanie o tym poinformowany.

  Przykładowe polecenie typu 4:
  `4 1 2 Hello World` (wiadomość na temat nr 1, o priorytecie 2 i treści “Hello World”

```
case 4: // nowa wiadomość na dany temat
                carry = 1;
                for(int i = 0; i < 10; i++){ // sprawdzanie czy
użytkownik jest zalogowany
                    if(pid[i] == msg.pid){
                        carry = 0;
                    }
                }
                if(carry == 0){
                    int typ, pior;
                    sscanf(msg.text, "%d%d%*c%[^\n]", &typ, &pior,
text);
                    int d = message(msg.pid, typ, pior, text);
                    if (d == 1){ // powodzenie
                        sendto(pipes[id(msg.pid)], "Wysłano
wiadomość\n");
                    } else if(d == -1) { // błąd
                        sendto(pipes[id(msg.pid)], "Podany temat nie
isnieje\n");
                    } else if (d == -2){ // błąd
                        sendto(pipes[id(msg.pid)], "Nie subskrybujesz
tego tematu\n");
                    }
                }
                break;
```

```
int message(int p_id, int type, int prior, char text[]){ // wysyłanie
wiadomości na dany temat
    if(topics[type - 1].created == 0) return -1; // temat nie istnieje
    int carry = 1;
    for(int i = 0; i < 10; i++){ // sprawdzanie czy użytkownik
subskrybuje dany temat
        if(topics[type - 1].użytkownicy[i].id == id(p_id)){
            carry = 0;
            break;
        }
```

```
    }
    if(carry == 0){
        char nazwa[10] = "";
        strcpy(nazwa, names[id(p_id)].name);
        struct msgbuf msg;
        msg.type = id(p_id) + 1;
        msg.prior = prior;
        sprintf(msg.text, "%s%s%d%s%s\n", nazwa, " w temacie ", type,
": ", text);
        for(int i = 0; i < 10; i++){ // wysyłanie do odpowiednich
użytkowników
            int id = topics[type - 1].użytkownicy[i].id;
            if(id != -1 && pid[id] != p_id){
                msgsnd(pipes[id], &msg, sizeof(struct msgbuf) -
sizeof(long), IPC_NOWAIT);
                if(topics[type - 1].użytkownicy[i].how_long != -1){ //
how_long == -1 oznacza sub. trwałą
                    topics[type - 1].użytkownicy[i].how_long -= 1;
                    if(topics[type - 1].użytkownicy[i].how_long == 0){
                        topics[type - 1].użytkownicy[i].id = -1; // id
== -1 oznacza brak subskrypcji
                        sendto(pipes[id], "Subskrypcja powyższego
tematu zakończyła się\n");
                    }
                };
            }
        }
        return 1; // powodzenie
    } else return -2;  // użytkownik nie subskrybuje

};
```

- **Typ 5** odpowiedzialny jest za blokowanie użytkowników. W tym celu klient musi podać nazwę użytkownika którego chce zablokować. Serwer przekazuje klientowi id tego użytkownika, klient zapamiętuje go i później sam radzi sobie z filtrowaniem wiadomości. (serwer nie przechowuje żadnych informacji na temat zablokowanych użytkowników).

  Przykładowe polecenie typu 5:
  `5 Wojtek`

```
case 5: // blokowanie użytkownika
                carry = 1;
                for(int i = 0; i < 10; i++){ // sprawdzanie czy
użytkownik jest zalogowany
                    if(pid[i] == msg.pid){
                        carry = 0;
                    }
                }
                if(carry == 0){
                    sscanf(msg.text, "%s", nazwa);
                    for(int i = 0; i < 10; i++){
                        if(strcmp(names[i].name, nazwa) == 0){ //
odnalezienie użytkownika o podanej nazwie
                            struct msgbuf msg1;
                            msg1.type = 12;
                            sprintf(msg1.text, "%d\n", i);
                            msgsnd(pipes[id(msg.pid)], &msg1, size,
IPC_NOWAIT);
                        }
                    }
                }
                break;
```

- **Typ 6** odpowiedzialny jest za wysłanie informacji zwrotnej, dzięki której klient wyświetla wiadomości. Typ 6 okazał się być pomocny w przypadku asynchronicznego odbioru wiadomości. W moim projekcie, serwer nie wie, w jaki sposób dany klient odbiera wiadomości, a co za tym idzie, wysyła je każdemu od razu. Program klienta jest odpowiedzialny za to, aby odebrać je i w zależności od metody odbioru, albo je wyświetlić, albo przechować, aż do wywołania polecenia 6.

  Przykładowe użycie polecenia typu 6:
  `6`

  Struktury i funkcje odpowiedzialne za polecenie 6:

```
case 6:
                struct msgbuf msg1;
                msg1.type = 50;
                msgsnd(pipes[id(msg.pid)], &msg1, size, IPC_NOWAIT);
                break;
```

4. Funkcjonalność (klient):

Klient dzieli się na dwa procesy, jeden odpowiedzialny jest za czytanie z wejścia i wysyłanie poleceń do serwera, a drugi za przyjmowanie informacji od serwera i przetwarzanie ich. Sposób odbioru wiadomości definiuje się przy uruchamianiu programu klienta gdzie 1 - odbiór synchroniczny, 2 - odbiór asynchroniczny. Kod jest podzielony na dwie części, jedna obsługuje odbiór synchroniczny, a druga, asynchroniczny.

Podstawowe struktury i funkcje (klient):

```
struct msgbuf{ // bufor na wiadomości kolejki komunikatów
    long type;
    char text[1024];
    int pid;
    int prior;
};

struct messages{ // wiadomość używana w tablicy zapisanych wiadomości w przypadku odbioru asynchronicznego
    char text[1024];
    int prior, displayed, expire;
};
```

```
int blocked[10] = {0};
struct messages mess[100];
int saved_mess = 0;
```

```
if(atoi(argv[1]) == 1){
// program dla odbioru synchronicznego
} else {
// program dla odbioru asynchronicznego
}
```

```
if (fork() == 0){ // kod procesu potomnego odpowiedzialnego za odbiór wejścia i wysyłanie poleceń
            while(1){
                char buf[1024] = "";
                int bytes = read(0, buf, 1024);
                struct msgbuf msg;
                msg.type = atoi(&buf[0]);
                msg.pid = key;
                for(int i = 0; i < bytes - 2; i++){
                    msg.text[i] = buf[i+2];
                }
                msgsnd(ids, &msg, sizeof(struct msgbuf) - sizeof(long), 0);
```

```
            }
        }
```

Program klienta otrzymuje 4 główne typ wiadomości:

Typ 100 to informacje od serwera które natychmiast wyświetlamy:

```
case 100:
                        if (strcmp(msg.text, "Subskrypcja powyższego
tematu zakończyła się\n") != 0) printf("%s\n", msg.text); // ta
informacja dotyczy tylko odbioru synchronicznego
                        break;
```

Typ 50 to informacja dla klienta, żeby wyświetlić wszystkie zapisane wiadomości, w przypadku odbioru asynchronicznego:

```
case 50:
                        display(saved_mess, mess);
                        for(int i = 0; i < saved_mess; i++){
                            strcpy(mess[i].text, "");
                            mess[i].displayed = 0;
                        }
                        saved_mess = 0;
                        break;
```

```
int display(int saved_mess, struct messages mess[]){
    for(int j = 0; j < saved_mess; j++){
        int min = 10000;
        int min_i;
        for(int i = 0; i < saved_mess; i++){ // szukanie wiadomości o
najniższym priorytecie
            if(mess[i].displayed == 0 && mess[i].prior < min){
                min_i = i;
                min = mess[i].prior;
            }
        }
        mess[min_i].displayed = 1; // zapamiętanie, że została
wyświetlona
        printf("%s\n", mess[min_i].text); // wyświetlenie jej
    }
    return 1;
};
```

Typ 12 to informacja zwrotna z id użytkownika, którego klient chciał zablokować:

```
case 12:
         if(blocked[atoi(msg.text)] == 0){
           blocked[atoi(msg.text)] = 1;
           printf("%s\n", "Zablokowano użytkownika");
         } else {
            blocked[atoi(msg.text)] = 0;
            printf("%s\n", "Odblokowano użytkownika");
         }
         break;
```

Ostatni rodzaj wiadomości to wiadomości od użytkowników na dany, subskrybowany przez klienta temat. Takie wiadomości mają typ odpowiadający id użytkownika, który ją wysłał oraz zawierają jej treść:

```
default:
         if(blocked[msg.type - 1] == 0){ // wyświetlanie jeśli
użytkownik nie jest zablokowany
              strcpy(mess[saved_mess].text, msg.text);
              mess[saved_mess].prior = msg.prior;
              saved_mess++;
         }
         break;
```

Kod programu od odbioru synchronicznego nie ma sekcji odpowiedzialnej za interpretowanie wiadomości typu 50.