UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

Industrijski komunikacioni protokoli u elektroenergetskim sistemima

Projekat P2P

Autori:
Jelena Beader
Stefan Đurović

SADRŽAJ

1. Uvod

Cilj projekta je pružiti klijentima dva tipa komunikacije sa ostalim klijentima. Jedan tip komunikacije je komunikacija posredstvom servera (klijent-server-klijent), pri čemu server prosleđuje poruku određenom klijentu, a drugi tip komunikacije podrazumeva da se klijenti povežu (klijent-klijent), posle čega mogu da direktno razmenjuju poruke. Takođe je moguća komunikacija sa više klijenata.

2. Dizajn

Projekat sadrži dvekomponente: klijent i server.Komunikacija se realizuje TCP protokolom.

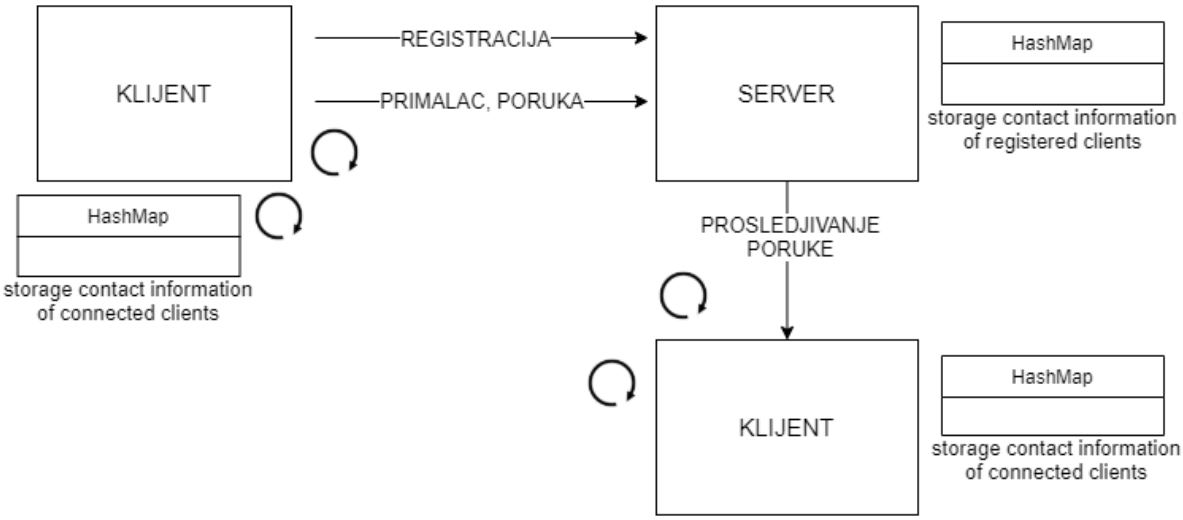
Klijent se na početku svog rada registruje, unosom svog imena, koji će biti poslat u okviru strukture Message_For_Client (Slika 7.) . Ime klijenta je podatak koji će biti jedinstven za svakog klijenta, te se neće moći više klijenata registrovati istim imenom. Registrovani klijenti se na serveru beleže u hash tabelu u kojoj se čuva ime,adresa, port, i kontakt informacije o tom klijentu, gde će ključ biti ime klijenta.

Nakon registracije, podrazumevan način komunikacije između klijenata je posredstvom servera (Slika 1.). Klijentu će, pre svakog slanja poruke, biti ponuđeno da pređe na direktannačin komunikacije, ukoliko to želi.
Od momenta uspešne registracije niti: “thread” i “thread_for_accept_clients” kreću sa radom. Nit “thread” prima poruke od servera na utičnici “connectSocket“, dok nit “thread_for_accept_clients” osluškuje događaje na utičnicama "listenSocket" (koja prihvata nove klijente) i nizu utičnica "acceptedSockets" (na kojima se primaju poruke od drugih klijenata).

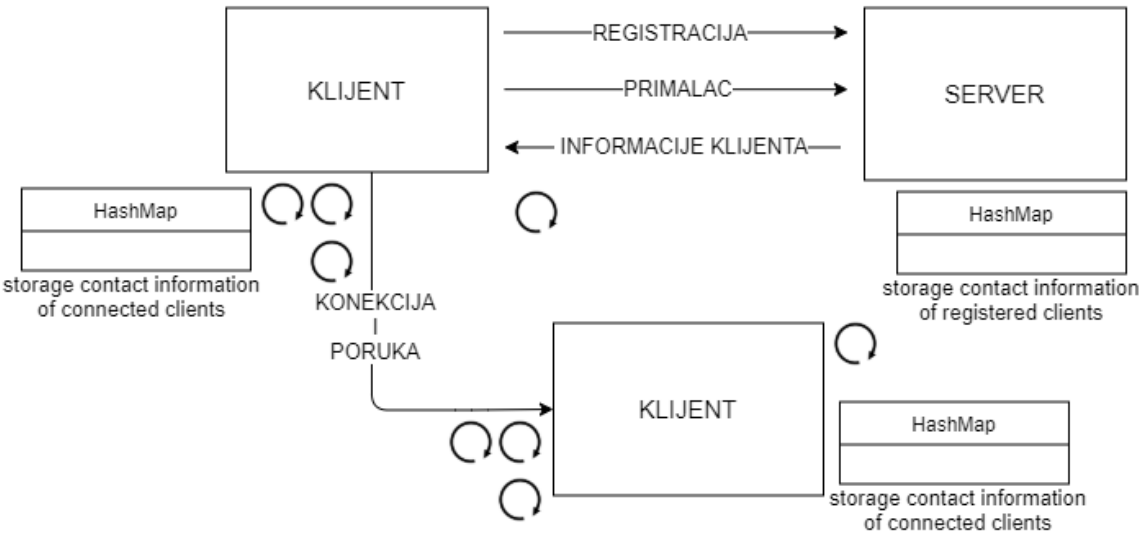
Ukoliko način komunikacije nije promenjen, klijent unosi ime onog klijenta kome želi daprosledi poruku, a zatim i sam sadržaj poruke. Podaci se šalju u vidu strukture Message_For_Client serveru (Slika 7.). Server proverava da li je klijent iz pristigle strukture registrovan, ako nije registrovan biće vraćena poruka o tome, a ako jeste server će proslediti poruku tom klijentu. Omogućeno je slanje više poruka različitim korisnicima.

Ako je klijent prešao na direktan vid komunikacije (Slika 2.), pre nego što uspostavi vezu sa željenim klijentom, on traži od servera kontakt informacije(adresu i port), koje su mu potrebne kako bi se direktno povezao sa njim. Server šalje ime datog klijenta u okviru strukture Message_For_Client (Slika 7.) . Ako klijent čije informacije tražimo nije registrovan, biće vraćena poruka o tome,a ako jeste server vraća informacije o njemu u strukturi Client_Information_Directly (Slika 8.).
Kada klijent pređe na direktnu komunikaciju, nit “thread” se gasi a niti : “thread_directly_recv”, “thread_send_message_directly” i “thread_recv_on_connectSockets” kreću sa radom. Nit “thread_directly_recv” prima poruke od servera, nit “thread_send_message_directly” pravi utičnice preko kojih se povezuje na željene klijente i čuva ih u nizu utičnica “connectSockets_directly”, dok nit

“thread_recv_on_connectSockets” osluškuje događaje na nizu utičnica “connectSockets_directly” (na kojima se primaju poruke od drugih klijenata).



Slika 1. Komunikacija prosleđivanjem poruke preko servera



Slika 2. Direktna komunikacija klijenata

Klijent skladišti dobijene kontakt informacije klijenta, sa kojim se povezuje direktno, u hash tabelu. Nakon toga, uspostavlja se veza i ukoliko je ona uspešno uspostavljena klijent unosi poruku koja se šalje u strukturi Directly_Message (Slika 9.) . Moguće je uzastopno slanje poruka klijentu sa kojim je veza

uspostavljena. Omogućena je direktna komunikacija sa više klijenata, pri čemu klijent koji je prešao na direktnu komunikaciju i dalje može da prima prosleđene poruke od servera koje mu šalju drugi klijenti.

Razlozi navedenog dizajna:

- Zbog velikog broja događaja koji treba da se izvršavaju paralerno, na klijentu su dodate prethodno pomenute niti.
- Korišćenje semafora za razmenu notifikacija između niti, kako bi sinhronizovali izvršavanje niti i uskladili izvršavanje programa.
- Korišćenjem kritičnih sekcija smo obezbedili da ne dođe istovremenog pristupa određenim resursima(hash tabeli, standarnom ulazu/izlazu, utičnicama) od strane više niti.
- U hash tabeli "HashMap", na klijentu, skladište se klijenti sa kojima postoji direktna veza, kako se ne bi svaki put od server tražile kontakt inforamcije, kada želimo da komuniciramo sa klijentom sa kojim smo već povezani.

3. Strukture podataka

Podaci u okviru projekata se skladište u hash tabelama tj. mapama. Glavni razlog iza odabira za korišćnjem hash tabela predstavlja razlog da prosečna cena (tj. broj računarskih instrukcija) svakog pronalaženja je nazavisne od broja elemenata uskladištenih u tabeli.

U okviru projekta implementiramo hash tabela na serverskoj aplikaciji, ali takođe svaki klijent ima svoju lokalnu hash tabela (čuvanje kontakt informacija).

Hash tabela je struktura podataka koja koristi hash funkciju (Slika 3.) za efikasno preslikavanje određenih ključeva (na primer korisničkog imena klijenata) u njima pridružene vrednosti. Hash funkcija se koristi za transformisanje ključa u indeks (hash) to jest mesto u nizu elemenata gde treba tražiti odgovarajuću vrednost.

3. 1 Hash funckija i kolizije

```
unsigned long GenerateHashValue(unsigned char *str)
{
	unsigned long hash = 5381;
	int c;

	while (c = *str++)
		hash = ((hash << 5) + hash) + c;
	return hash;
}
```

Slika 3. Bernstajnov hash algoritam

Bernstajnov hash algoritam (Slika 3.) se oslanja na magičnu vrednost broja 33 (zašto to deluje bolje od mnogih drugih konstanti, prostih ili ne, nikada nije adekvatno objašnjeno).

Idealno, hash funkcija bi trebalo da preslikava svaki mogući ključ u zaseban indeks, ali ovaj cilj je u praksi nije uvek ispunjen. Većina implementacija hash tabela podrazumeva da su hash kolizije – parovi različitih ključeva sa istim hash vrednostima – normalna pojava, i za neki način se stara da se ovaj problem prevaziđe.

Pomenuti problem kolizije hash vrednosti, prevazišli smo korišćenjem heširanja sa ulančavanjem. Heširanje ulančavanjem, koje se drugačije naziva i heširanje spajanjem, je strategija za rešavanja sudara u hash tabeli koja predstavlja hibrid odvojenog nizanja i otvorenog adresiranja. Kod heširanja ulančavanjem (Slika 4.), više ključeva se preslikava u isti indeks, tj. elementi sa različitim ključevima se mogu smestiti u isto polje hashmape.

Glavna razlika u odnosu na obično dodavanje elemenata u hash mapu, jeste da ako naletimo na poklapanje indeksa, taj novi element nadovezujemu na postojeći, koristeći ulančanu listu, time rešavamo problem kolizije indeksa, ako su lanci (ulančane liste) krakti kao u našem slučaju strategija ulančavanja je veoma efikasna i koristi malo memorije.

```
bool AddValueToHashMap(ClientData* clientData)
{
	struct Element *newElement = (struct Element*)malloc(sizeof(struct
	Element));
	strcpy_s((char*)clientData->directly, sizeof(clientData->directly), "0\0");
	newElement->clientData = clientData;
	newElement->nextElement = NULL;

	unsigned int key = GenerateHashValue(clientData->name) % MAX_CLIENTS;

	if (HashMap[key] == NULL)
	{
		HashMap[key] = newElement;
		return true;
	}
	else
	{
		struct Element *tempElement = HashMap[key];
		while (tempElement->nextElement)
		{
			tempElement = tempElement->nextElement;
		}
		tempElement->nextElement = newElement;
		return true;
	}
	return false;
}
```

Slika 4. Funkcija dodavanje elementa u hash mapu (sa ulančavanjem.)

3.2 Strukture podataka unutar hash mapa

Elementi koje se čuvaju unutar hash tabela, predstavljaju C-ovske strukture pod kojima su objedninjeni klijentski podaci koje se čuvaju u okviru hash tabele na serverskoj aplikaciji (Slika 5.)

```
typedef struct ClientData {
	unsigned char name[MAX_USERNAME]; // korisničko ime
	unsigned char address[MAX_ADDRESS]; // logička adresa korisnika
	unsigned int port; // korisnikov port
	unsigned char listen_address[MAX_ADDRESS]; // logička adresa za prijem
	poruka
	unsigned int listen_port; // port korisnika za prijem poruka
	unsigned char directly[2]; // da li se komunikacija odvija direktno
} ClientData;
```

Slika 5. Struktura koja se čuva na serverskoj heš mapi

Pored serverske hash mape, svaki klijent na svojoj strani poseduje svoju hash mapu gde se skladište klijenti sa kojima postoji direktna veza, kako se ne bi svaki put od server tražile kontakt informacije kada želimo da komuniciramo sa klijentom sa kojim smo već povezani.

```
typedef struct ClientData {
	unsigned char name[MAX_USERNAME]; // korisničko ime
	unsigned char address[MAX_ADDRESS]; // logička adresa korisnika
	unsigned int port; // korisnikov port
	unsigned char socket_type[2];  // direktna/serverske komunikacije
} ClientData;
```

Slika 6. Struktura koja se čuva na klijentskoj heš mapi

3.3 Strukture podataka u komunikaciji

U komunikaciji između servera i klijenata odvija se neprestana razmena C-ovskih struktura. Razmenom struktura, obe strane razmenjunju podatke bitne za rad kako serverske tako i klijentskih aplikacija.

```
struct Message_For_Client  // struktura koju dobijam od klijenta
{
	unsigned char sender[MAX_USERNAME]; // ime posiljaoca
	unsigned char receiver[MAX_USERNAME]; // ime primaoca
	unsigned char message[MAX_MESSAGE]; // poruka
	unsigned char listen_address[MAX_ADDRESS]; // logička adresa korisnika
	unsigned int listen_port; // port korisnika
	unsigned char flag[2];  // vrednosti:
				"1"(registracija) /
				"2"(prosleđivanje) /
				"3"(direktno) /
				"4"(presao sam na direktnu) + null terminator
};
```

Slika 7. Struktura koju klijent salje server

Klijent, serveru prosleđuje strukturu (Slika 7.) prilikom registracije, kako bi server skladištio datog klijenta i omogućio komunikaciju u okviru sistema. Pomenuta struktura pored registracije može imati više svrha, koji diktira podatak “flag” unutar same strukture.

U zavisnosti od vrednosti flag-a, server inicira određene akcije sa podacima koje primi od klijenta kretajući se od inicijalne registracije pa sve do prosleđivanje poruke , promene načina komunikacije. Time smo objedinili pod jednom strukturom mogućnost izvršavanja različitih akcija na serveru a da pri tome nismo menjali strukturu koja pristiže sa klijentske strane.

Sa druge strane server, sa klijentima razmenjuje još jednu strukturu, koja je zadužena za prosleđivanje informacija klijentima kada se opredele za direktnu komunikacija sa drugim registrovanim klijentom.

```
struct Client_Information_Directly  // informacije o klijentu sa kojim želimo direktno da
                                    komuniciramo
{
              unsigned char my_username[MAX_USERNAME]; // klijent koji traži podatke
              unsigned char client_username[MAX_USERNAME]; // ime klijent za koga se
                                                           // traže podaci
              unsigned char message[MAX_MESSAGE]; // poruka
              unsigned char listen_address[MAX_ADDRESS]; //  logička adresa korisnika
              unsigned int listen_port; // port korisnika
};
```

Slika 8. Struktura koja se šalje klijentu za direktnu komunikaciju sa drugom registrovanom klijentom

Na klijentskoj strani dolazi do razmene još jednog tipa strukture (Slika 9.) koje je vezana za direktan tip komunikacije između klijenata. Struktura poseduje flag, koji nam označava da li stukturom prenosimo korisnikovo ime, ili poruku koja je namanjena njemu (korisnikovo ime se prenosi, kako bi ga upisali u klijentskoj hash mapi) .

```
struct Directly_Message {
       unsigned char message[MAX_DIRECTLY_MESSAGE];
       unsigned char flag[2];
};
```

Slika 9. Struktura koja se razmenjuje između klijenata (direktna komunikacija)

3.4 Dodatne operacije nad heš mapom

Pored operacija za heširanjem i dodavanjem elemenata u hash tabelu, implementiramo dodatne funkcije za pretragu, brisanje i ažuriranjem elemenata koji su skladišteni. Pošto smo se odlučili za implementaciju hash tabela sa ulančavanjem, pomenute operacije su složenije u odnosu na standarne operacije hash mape, kojima izostaje rešavanja problema kolizije.

Funckija za pretragu (Slika 10.) elementa iz hashtabele, vraća element ukoliko on postoji u okviru tabele, u suprtonom vraća *NULL*. Pretraga hešuje vrednost po kojoj pretražujemo našu hashtabelu s obzirom da koristimo ulančavanje, moramo takođe proći kroz ulančanu listu, da proverim da li više elemenata nije mapirano pod istim indeksom.

```
ClientData* FindValueInHashMap(unsigned char *clientName)
{
	unsigned int key = GenerateHashValue(clientName) % MAX_CLIENTS;
	if (HashMap[key] != NULL)
	{
		struct Element *tempElement = HashMap[key];
		while (tempElement)
		{
			if (strcmp((const char*)clientName,
			(const char*)tempElement->clientData->name) == 0)
			{
				return tempElement->clientData;
			}
			tempElement = tempElement->nextElement;
		}
}
```

Slika 8. Funkcija za pretragu hash mape

Brisanje elemenata predstavlja najteži deo posla, zbog primene ulančavanja, ako želim da ukolonimo jedan element iz ulančane liste, moramo pomeriti elemente kao da taj element nikad nije ni postojao, što dodaje svoju težinu prilikom implementacije. Ukoliko nema više ulančanih elemenata pod istim ključem, na mestu tog element postavljamo NULL, čime smo obrisali element iz skladišta tj. hash mape.

```
bool RemoveValueFromHashMap(unsigned char *clientName)
{
	unsigned int key = GenerateHashValue(clientName) % MAX_CLIENTS;
	struct Element *tempElement = HashMap[key];
	if (tempElement != NULL)
	{
		if (strcmp((const char*)clientName,
				(const char*)tempElement->clientData->name) == 0)
		{
			HashMap[key] = NULL;
			return true;
		}
		else
		{
			while (tempElement->nextElement)
			{
				if (strcmp((const char*)clientName,
				(const char*)tempElement->nextElement->clientData->name) == 0)
				{
					HashMap[key]->nextElement =
							tempElement->nextElement->nextElement;
					return true;
				}
				tempElement = tempElement->nextElement;
			}
		}
	}
	return false;
}
```

Slika 10. Brisanje element iz hash mape.

4. Potencijalna unapređenja

Unapređenje koja bi bilo dobro uvesti je pravljenje nove niti na serveru za svakog novog klijenta koji se registruje. Svaka nit bi imala zadatak da opslužuje samo jednog klijenta, čime bi sav posao na serveru bio podeljen.

U aplikaciji postoji ograničenje maksimalnog broja klijenata, što bi se takođe moglo unaprediti. Način za rešenje ovog problema bi bio pravljenje liste klijenata, koja bi bila dinamički proširiva i tako bi se otklonio ovaj nedostatak.

Trebalo bi posvetiti pažnju boljoj memorijskoj organiziciji na nivou čitave aplikacije, kako bi došlo do veće uštede memorije,a samim tim i kvalitetnijeg rada programa.