Industrijski komunikacioni protokoli u elektroenergetskim sistemima

Publisher-Subscriber

Projekta dokumentacija

Autori: Dušan Mazić PR88/2017

Nikola Mijonić PR49/2017

Mentori: Bojan Jelačić
Marina Stanojević

Sadržaj

1. Uvod

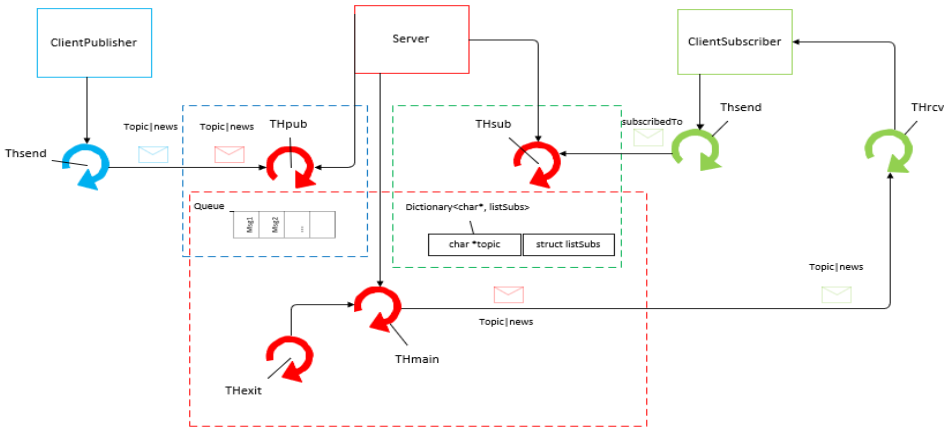
Opis problema:

Neophodno je napraviti klijent-server aplikaciju korišćenjem C programskog jezika koja implementira PubSub servis. Aplikacija pruža mogućnost obrade zahteva neograničenog broja klijenata. Sistem identifikuje sledeće entitete: Server, ClientPublisher i ClientSubscriber. Server opslužuje zahteve klijenata i u zavisnosti od potrebe izvršava određene akcije. ClientPublisher ima mogućnost slanja neograničenog broja poruka za određenu temu. Nakon slanja, poruke se smeštaju u odgovarajuće strukture nakon čega se prosleđuju ClientSubscriber-u. ClientSubscriber-u se pruža mogućnost odabira tema. Nakon odabira, biva obavešten o novostima odabranih tema.

Cilj zadatka:

Osnovni cilj ovog projektnog zadatka je implementacija teorijski opisanih stavki iz opisa problema. Pri implementaciji neophodno je voditi računa o performansama sistema kao i oslobađanju zauzetih resursa. Potrebno je voditi računa o „Clean Code“ standardu, što dalje govori da su implementacije i potpisi funkcija nalaze u okviru posebnih biblioteka. Iz prethodno opisanog problema može se zaključiti da je upotreba niti u vidu konkurentnog programiranja neizbežna. Korišćenje niti zahteva upotrebu odgovarajućih mehanizama zaštite pristupa podacima u vidu kritičnih sekcija i semafora. Takođe, strukure koje se koriste su samostalno implementirane kako bi u potpunosti odgovarale potrebama projekta.

2. Dizajn



Opis dizajna:

Pre samog opisa dizajna neophodno je reći da celokupan sistem implementira komunikacioni protokol koji omogućava komunikaciju između komponenti sistema. Na osnovu ovog protokola moguće je na serveru identifikovati tip klijenta I u zavisnosti od toga preduzeti odgovarajuće akcije. Za svaku klijentsku konekciju kreira se nova nit koja će obrađivati zahteve tog konektovanog klijenta i vršiti kominukaciju sa njime u zavisnosti od potrebe. Takođe, sve komponente sistema koriste biblioteku koja sadrži implementaciju svih metoda potrebnih za očekivano funkcionisanje celokupnog sistema.

ClientPublisher kao što je rečeno u opisu problema ima mogućnost slanja podataka (poruka) kroz mrežu do odgovarajućih klijenata. Ovaj klijent koristi jedan “Thread” zato što u okviru funkcionalnosti koju on pruža ne postoji mogućnost konkurentnog izvršavanja. ClientPublisher šalje poruke serveru u odgovarajućem formatu koji zadovoljava zahteve komunikacionog protokla.

Server predstavlja posrednika između ClientPublisher-a i ClientSubscriber-a. Nakon identifikovanja tipa klijenta, kreira se odgovarajuća nit koja izvršava adekvatnu funkciju u zavisnosti od samog tipa klijenta. THpub je nit koja je zadužena za obradu podataka koji stižu od strane ClientPublisher-a. Server izvlači neophodne informacije i smešta ih u “Queue” odgovarajućih klijenata (pretplaćenih). Nakon smeštanja podataka u strukturu THpub nit signalizira niti zaduženoj za preuzimanje tih podataka. Pored THpub-a postoji nit koja je zadužena za opsluživanje ClientSubscriber-a (THsub). Ova nit preuzima podatke koji nose informacije o temama na koje se klijent pretplatio. Na osnovu tih podataka popunjava se Dictionary koji vezuje temu i listu klijenata koji su na tu temu pretplaćeni. Drugim rečima, ključ pretstavlja temu, a vrednost je jednostruko spregnuta lista koja sadrži identifikatore (ID-eve) pretplaćenih klijenata. Nakon popunjavanja dictionary-a, ova nit preuzima odgovornost za prosleđivanje poruka ClientSubscriber-u. Da bi podaci bili preuzeti sa queue-a, a nakon toga i prosleđeni klijentu, neophodan je signal od strane semafora koji nagoveštava da je poruka dodata na queue. Takođe, na Server-u se nalazi jedna nit koja je korišćena za detekciju pritiska ESC tastera koji signlazira zatvaranja servera korišćenjem signala semafora.

ClientSubscriber nakon uspostave komunikacije sa serverom, šalje podatke u obliku niza celobrojnih vrednosti iz kojih se može zaključiti na koje teme se pretplatio. Nakon što se pretplatio na željene teme, subscriber čeka poruke publisher-a koje mu Server prosleđuje. Čekanje poruka se odvija u posebnoj niti THrcv. Podaci koje dobija od servera sadrže informacije o identifikatoru teme kao I samu tekstualnu formu poruke. Informacija o identifikatoru teme bila je nepohodna kako bi subscriber imao informaciju za koju temu je stigla nova poruka. Poruke se ispisuju na konzoli.

Razlozi prethodno opisanog dizajna:

Komunikacioni protokol je implementiran kako bi dva entiteta mogla da komuniciraju po unapred definisanim pravilima.

Kao što je već rečeno za svakog novog klijenta kreira se nova nit koja u zavisnosti od tipa klijenta izvršava različitu funkciju. Ovaj pristup omogućio je znatno bolju iskorićenost procesora pa samim tim I bolje performanse.

Postojanje različitih komponenti I biblioteka omogućilo je visok nivo apstrakcije I olakšalo kreiranje testova za pojedinačne komponente kao I pregledniji kod.

U okviru ClientPublisher-a postoji samo jedna nit zato što ne postoji mogućnost konkuretnog izvršavanja funkcionalnosti koje on pruža. Iz prethodno navedenog razloga može se zaključiti da korišćenje dodatnih niti nije opravdano zato što njihova upotreba zahteva korišćenje određenih resursa, a benefita u ovom slučaju ne bi bilo.

Poruke se smeštaju na Queue zato što on u osnovi prestavlja FIFO strukturu koja odgovara potrebama samog sistema. Poruka koja se prva smesti na Queue je ujedno I poruka koja se prva skida sa njega.

Dictionary sa prethodno opisanom strukturom je korišćen kako bi se postigle maksimalne performanse prilikom pretrage klijenata na čiji queue treba dodati nove poruke. U slucačuju dictionary-a kompleksnost pretrage je O(1) dok je u slučaju sekvencijalne pretrage niza O(n).

Signal semafora koji signalizira kraj je prisutan kako bi se zauzeti resursi oslobodili I kako bi se sprečilo štetno curenje memorije. Pored signala koji signalizira kraj postoji signal koji govori da je poruka dodata na Queue I da se može skinuti sa njega. Ovaj signal je dodat da procesor ne bi bio zauzet konstantnim proverama da li je nešto dodato na Queue. Na ovaj način oslobodili smo procesor I omogućili da izvršava neki drugi proračun ili šta god da je u tom trenutku potrebno.

ClientSubscriber poseduje posebnu nit koja osluškuje poruke od servera kako glavna nit tog klijenta ne bi bila blokirana. Na ovaj način u slučaju potrebe proširenja samog sistema omogućena je nadogradnja bez potrebe menjanja postojećeg koda. Ispoštovan je OPEN/CLOSED princip.

3. Strukture podataka

Jednostruko spregnuta lista - je struktura koja je korišćena na više mesta u okviru projekta. Glavni razlog odabira ove strukture je mogućnost smeštanja proizvoljnog broja elemenata. Slede primeri korišćenja prethodno pomenute strukture.

- **ClientInformationList** je jednostruko spregnuta lista u okviru koje su se čuvale sve relevantne informacije konektovanih klijenata. Kada kažemo relevantne informacije mislimo na sledeće pojmove:

  int clientID – jedinstveni identifikator svakog klijenta
  SOCKET clientSocket – omogućava komunikaciju sa klijentom
  DWORD lpThreadId – identifikator niti zadužene za obradu klijenta
  HANDLE handleClientThread – handle koji nastaje prilikom kreiranja niti
  HANDLE queueSemaphore – handle koji nastaje prilikom kreiranja semafora za Queue
  Queue* subscriberMessages – Queue koji se koristi za smeštanje poruka

  Ova lista je deljenja između niti i pruža mogućnost dobavljanja informacija o klijentima koji su prisutni u sistemu. Na taj način moguće je u bilo kom trenutku dobaviti informaciju o klijentu i primeniti željenu akciju nad njime.

- **ListAllClients** je lista zadužena za čuvanje identifikatora klijenata koji su pretplaćeni na određenu temu. Ova lista koristi se u okviru Dictionary strukture i predstavlja njegovu vrednost. Jedino polje ove liste je celobrojna vrednost identifikatora (int clientID). Korišćenje ove liste doprinelo je uštedi memorije tako što nije potrebno pamtiti sve atribute klijenata, oni se mogu dobaviti iz ClientInformationList-e.

- **ListTopic** je lista koja je korišćena za smeštanje tema na koje se ClientSubscriber može pretplatiti. Spisak ovih tema nalazi se u okviru tekstualnog fajla (Topics.txt). Na ovaj način je omogućeno postojanje proizvoljnog broja tema i pored toga omogućena je jednostavna modifikacija postojećih.

Queue – je struktura koja se koristi za smeštanje poruka koje se prosleđuju pretplaćenim klijentima. U okviru ove strukture čuva se informacija o broju elemenata koji ujedno predstavlja i broj poruka. Nad ovom strukturom implementirane su funkcije koje omogućavaju dodavanje poruke na početak (Enqueue) i preuzimanje poruke sa kraja (Dequeue). Ova struktura u potpunosti odgovara potrebama sistema zbog svoje FIFO (First In First Out) implementacije.

int count – broj poruka u Queue
Queue_node* front – pokazivač na prvi element u Queue
Queue_node* rear – pokazivač na poslednji element u Queue
CRITICAL_SECTION queue_cs – kritična sekcija koja implementira „thread safe“ pristup strukturi

Dictionary – je struktura koja je implementirana po uzoru na način funkcionisanja rečnika u objektno-orijentisanim programskim jezicima kao što je C#. Sastoji se od elemenata koji se nazivaju Key_value_pair. Svaki element sadrži informaciju o temi, koja ujedno predstavlja ključ na osnovu kojeg je moguće veoma brzo pronaći vrednost vezanu za taj ključ. Vrednost koja se vezuje za ključ predstavlja listu identifikatora klijenata (ListAllClients) koji su pretplaćeni na tu temu.

char* topic – naziv teme
Node* listAllClients – identifikatori klijenata pretplaćenih na tu temu

Pored gore navedenih struktura postoje i one koje su korišćene za smeštanje bitnih podataka u okviru same aplikacije. Slede primeri:

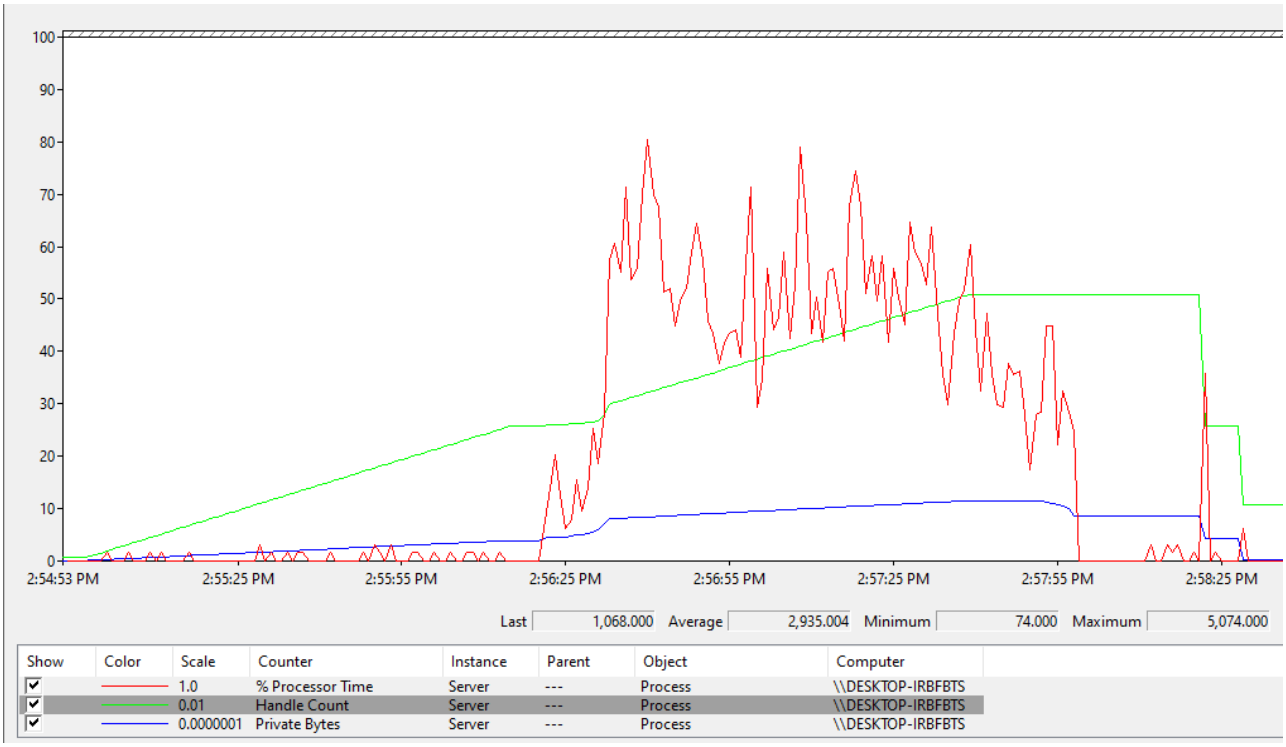
- PublisherNode – struktura koja sadrži informaciju o identifikatoru teme, dužinu poruke koja se šalje kao i sam tekstualni format poruke. Ova struktura je dodata kako bi u svakom trenutku bilo poznato gde se nalaze informacije koje se šalju.
- Important_data – struktura koja se koristi za prenos podataka niti (THrcv) koja je zadužena za preuzimanje poruka.
- Client_thread_data – struktura koja je zadužena za prosleđivanje neophodnih informacija nitima koje se kreiraju za klijentske konekcije.

4. Testiranje projekta

Za potrebe testiranja kreiran je poseban projekat StressTest koji simulira konfigurabilan broj klijenata. Na ovaj način je omogućeno jednostavno podešavanje parametara koji su potrebni da bi se sistem testirao. User ima mogućnost unosa broja ClientPublisher-a i ClientSubscriber-a koji će biti kreirani. Takođe, postoji konstanta (NUMBER_OF_PUBLISHER_MESSAGE) koja označava broj poruka koji se šalje kroz mrežu.

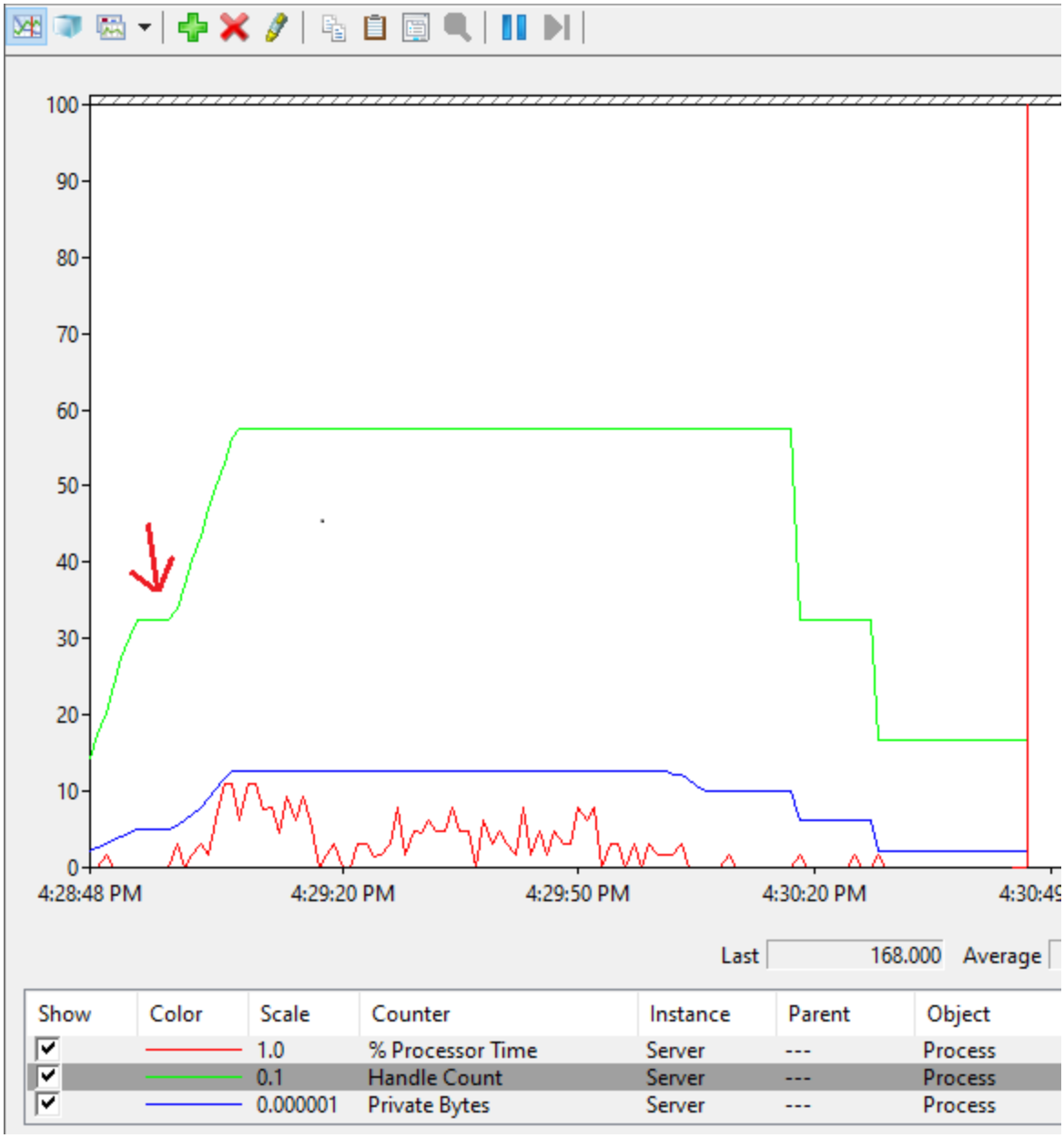
1. Test – Simuliranje konekcije velikog broja klijenata na server kako bi se testirala skalabilnost sistema.

   Broj Publisher-a: 500
   Broj Subscriber-a: 500
   Broj poruka za slanje: 5



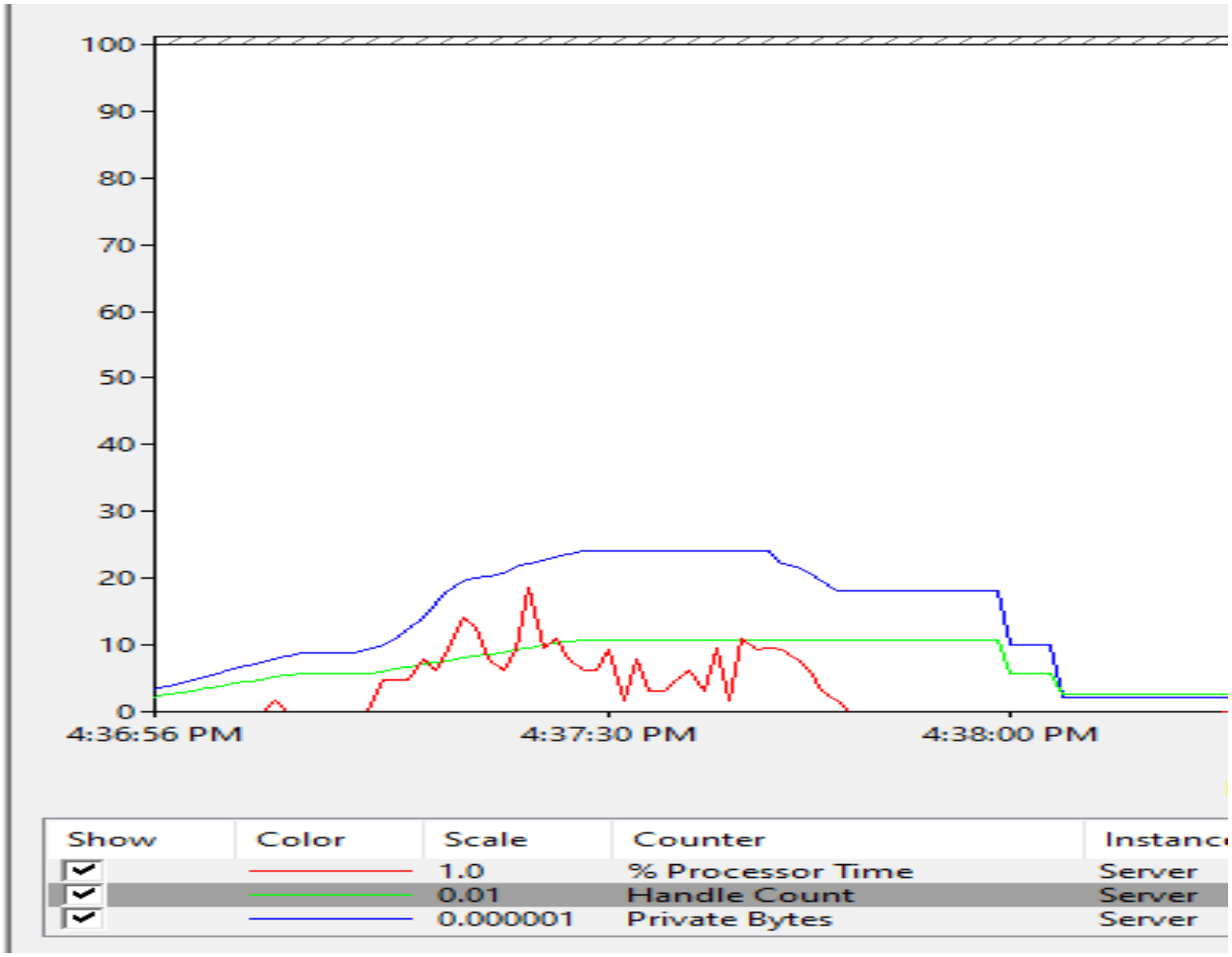
Slika 1. - Prikaz rezultata prvog testa

2. Test – Simulacija konekcije 50 ClientPublishera-a i 50 ClientSubscriber-a i slanje 50 poruka. Ovaj test proverava kritične delove sistema i pokušava pronaći granice pucanja.

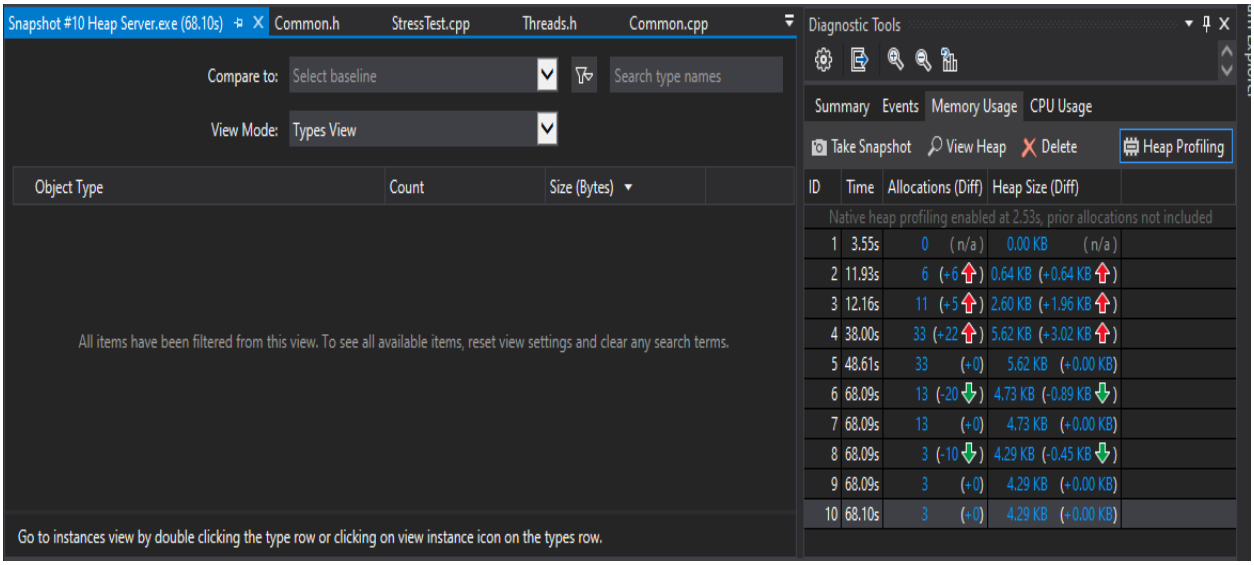


Slika 2. - Prikaz rezultata drugog test

3. Test - Simulacija konekcije 100 ClientPublishera-a i 100 ClientSubscriber-a i slanje 1000 poruka. Ovaj test proverava kritične delove sistema i dokazuje je sistem otporan na opterećenje.



Slika 3. - Prikaz rezultata trećeg test



Slika 4. Dokaz da se alocirana memorija Heap-a oslobađa

5. Zaključak

Rezultati prvog testa potvrđuju očekivanja. Posmatrani su sledeći brojači: Processor Time, Handle Count i Private bytes. Handle Count kao i Private bytes progresivno raste zato što se novi klijenti konstantno konektuju. Kada se svi konektuju, ova dva brojača imaju konstantnu vrednost. Ovaj test je najviše pogađao procesor zbog toga što je bilo neophodno kreirati veliki broj niti, a poznato je da je kreiranje niti vremenski zahtevna operacija. Nakon slanja i preuzimanja svih poruka započinje se oslobađanje zauzetih resursa što je i prikazano naglim padom grafika koji predstavlja Handle Count i Bytes Count. Nakon završetka slanja poruka procesorsko vreme se znatno smanjilo. Takođe procesorsko vreme doživljava još jedan manji pik koji simbolizuje signaliziranje i oslobađanje zauzetih resursa. U ovom testu zauzete su velike količine podataka pa je i opravdano da je procesoru potrebno određeno vreme za oslobađanje. Ovaj test takođe pokazuje da se zauzeti resursi oslobađaju, tj. ne postoji curenje memorije.

Drugi test nam dokazuje da je ono što je uočeno u prvom testu prisutno i u drugom. Posmatrani su isti parametri. Kako bi se uverili da su pravilnosti iz prvog testa validne, dodat je mala pauza između kreiranja niti ClientPublisher-a i ClientSubscriber-a. Ova pauza se može vrlo lako uočiti na grafiku (označena na slici 2). U slučaju ovog testa nema naglih skokova iskorišćenosti procesorskog vremena zato što se konektuje znatno manji broj klijenata, a samim tim je i broj kreiranih niti i zauzetih resursa manji.

Kao i prethodna dva testa i ovaj (treći) test pokazuje da se prilikom kreiranja niti procesorsko vreme koristi u velikoj meri. Takođe, potvrđuje da je „gracefully shutdown“ prisutan tj. oslobađaju se zauzeti resursi i zatvaraju se kreirane niti. Slika 4 je prikazana kao validan dokaz da nema curenja memorije.

6. Potencijalna unapređenja

Nakon pokrenutih StressTest-ova uočene su odeđene mane konkretne implementacije sistema. Kao što smo već više puta rekli, kreiranje niti je vremenski zahtevna operacija što se može uočiti i na graficima. Ovaj nedostatak mogao bi se ublažiti kreiranjem ThreadPool-a. Niti bi se kreirale pri pokretanju servera i bile bi spremne za obradu klijentskih zahteva. Na taj način bi se izbeglo okupiranje procesorskog vremena koje je trenutno izraženo. Ovo rešenje takođe može uzrokovati određene poteškoće u radu sistema. Konkretan primer bi detektovao test 1. Najverovatnije ThreadPool ne bi sadržao dovoljan broj niti da obradi sve zahteve i javila bi se potreba za dodatnim kreiranjem niti.

Trenutna implementacija smešta informacije o klijentima u ClientInformationList. Ovaj pristup je adekvatan zato što omogućava smeštanje neograničenog broja podataka. Problem se javlja ukoliko u sistemu postoji veliki broj klijenata i konekcija sa nekim od njih se zatvori. Kada se konekcija zatvori, neophodno je proći kroz celu listu kako bi se pronašao element koji treba da se obriše iz liste. Ovo predstavlja O(n) kompleksnost što se i moglo primetiti na prvom testu. Ovaj problem mogao bi se jednostavno rešiti upotrebom već implementirane strukture (Dictionary). Ključ bi bio identifikator klijenta, a vrednost svi ostali podaci. Ovo bi se u velikoj meri odrazilo na performanse sistema zato što bi pronalazak informacija o klijentu bio znatno brži.