**Dokumentacija za projekat**

**Publisher – Subscriber**

**Inicijalna specifikacija zadatka**

U datom projektu potrebno je implementirati PubSub servis koji može da opslužuje proizvoljan broj klijenata. Servis treba da implementira sledeći interface:

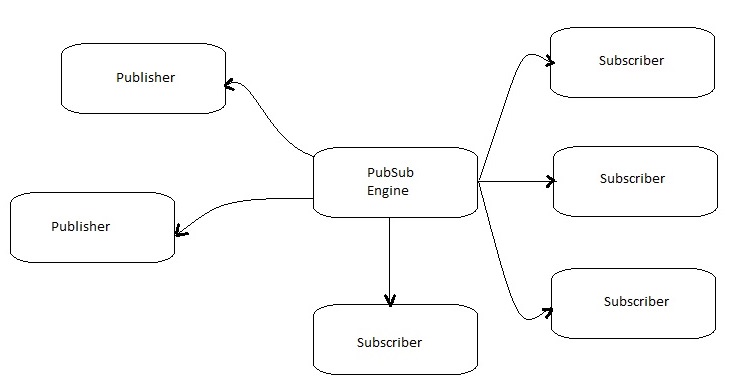
* void Connect();
* void Subscribe(void \*topic);
* void Publish(void \*topic, void \*message);

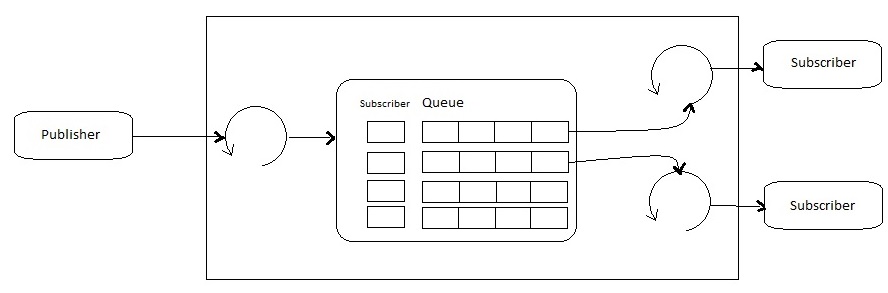
Format topic-a u okviru implementiranog PubSub servisa mora biti *<signal>, <type>, <num>*, gde:

* *signal* – može biti status ili analog
* *type* – može imati vrednost fuse, breaker(status) ili sec\_A, sec\_V(analog)
* *num* – predstavlja redni broj uređaja ili merenja(uint)

U slučaju da subscriber prilikom pretplate navede \* umesto type, tada mora dobijati rezultate za sve tipove uređaja koji pripadaju vrsti signala na koju se pretplatio. Implementirati validaciju ulaznih podataka na subscriber strani.

Arhitektura servisa:



Predlozeni dizajn servisa:

**Opis problema koji se resava**

Simulirati komunikaciju i rad Publish-Subscribe sistema.

Potrebno je uspostaviti komunikaciju (TCP neblokirajucu)

* Publisher<->PubSub Engine
* Subscriber<->PubSub Engine.

Omoguciti klijentima(Subscriber) da se pretplate na tip uredjaja i vrstu signala, kao i drugim klijentima(Publisher) da rucno unesu i objave temu(topic) sa porukom. Glavni servis(PubSub Engine) prosledjuje objavljene teme sa porukom korisnicima koji su pretplaceni na njih.

**Ciljevi zadatka**

* Uspostava komunikacije izmedju klijenata i servisa,
* Objava teme sa porukom od strane Publisher-a,
* Preplata na odredjenu temu od strane Subscriber-a,
* Preuzimanje tema i poruka od Publisher-a i prosledjivanje poruka Subscriber-ima od strane PubSub Engine-a.

**Dizajn i strukture podataka**

Glavne logicke komponente sistema su:

1. Publisher
2. PubSub Engine
3. Subscriber

Publisher kad uspostavi komunikaciju ima pravo da objavi temu. Kada pritisne opciju za objavu teme, postepeno konfigurise zeljenu temu. Nakon poslednjeg koraka uneta tema se automatski “upakuje” i posalje PubSub Engine-u.

Topic se sastoji od:

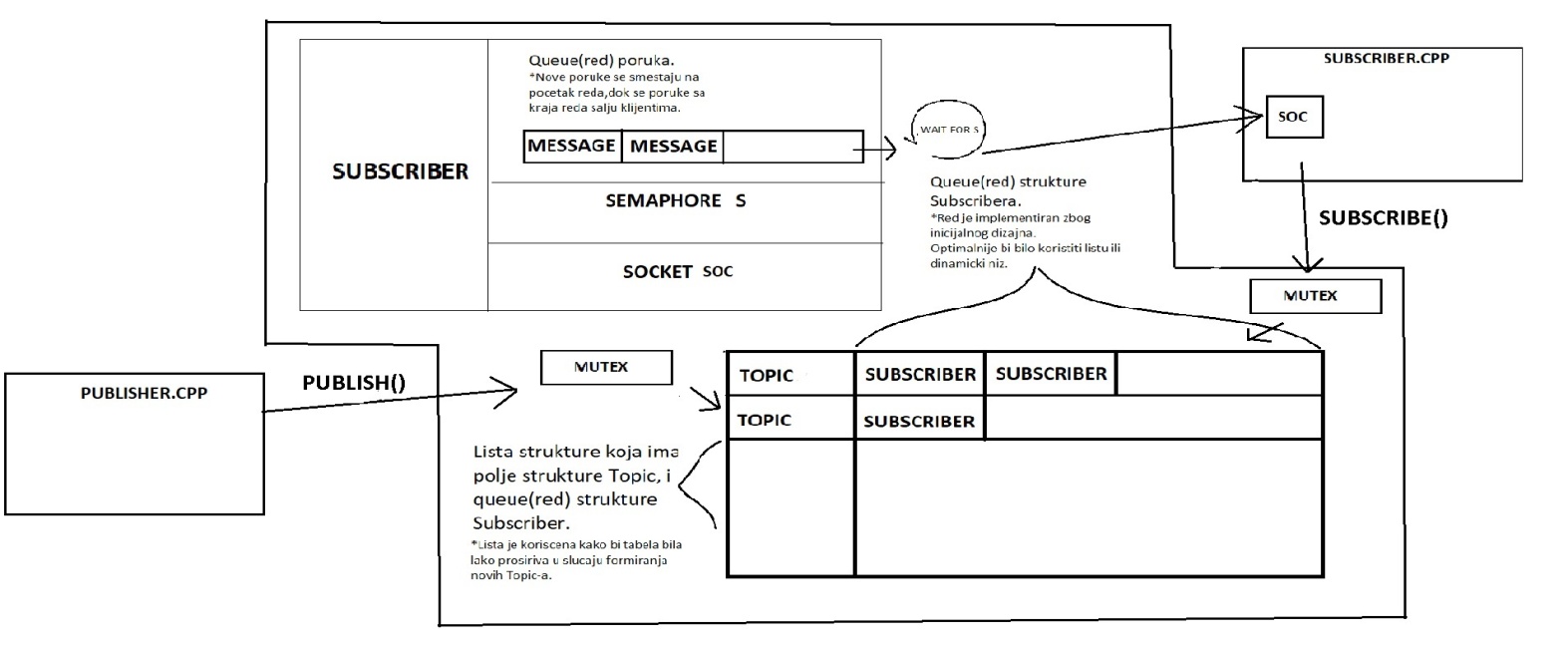
* Signal (*enum{status, analog}*)
* Type(*enum{fuse, breaker, sec\_A, sec\_V, all}*)
* Num(*uint*)

Korisnici(samo Subscriber-i) prilikom unosa tipa mogu da unesu i \* sto predstavlja pretplatu na sve tipove uredjaja koji pripadaju datoj vrsti signala i zbog toga je dodat element *all* u enumeraciji *type*.

Sve poruke u sistemu se primaju i salju sa header-om na pocetku(sizeof(int)) koji sadrzi samo velicinu poslatog/primljenog paketa pa zatim sadrzajem paketa.

Implementirana je trajna konekcija Publisher-a koji kada se podigne i uspostavi konekciju, ne gasi svoju konekciju tokom rada programa. Publisher prilikom objave pakuje poruku u strukturu (Topic,Message) i prosledjuje PubSub-u.

Sledeca slika prikazuje dizajn koda, kao i osnovne strukture.

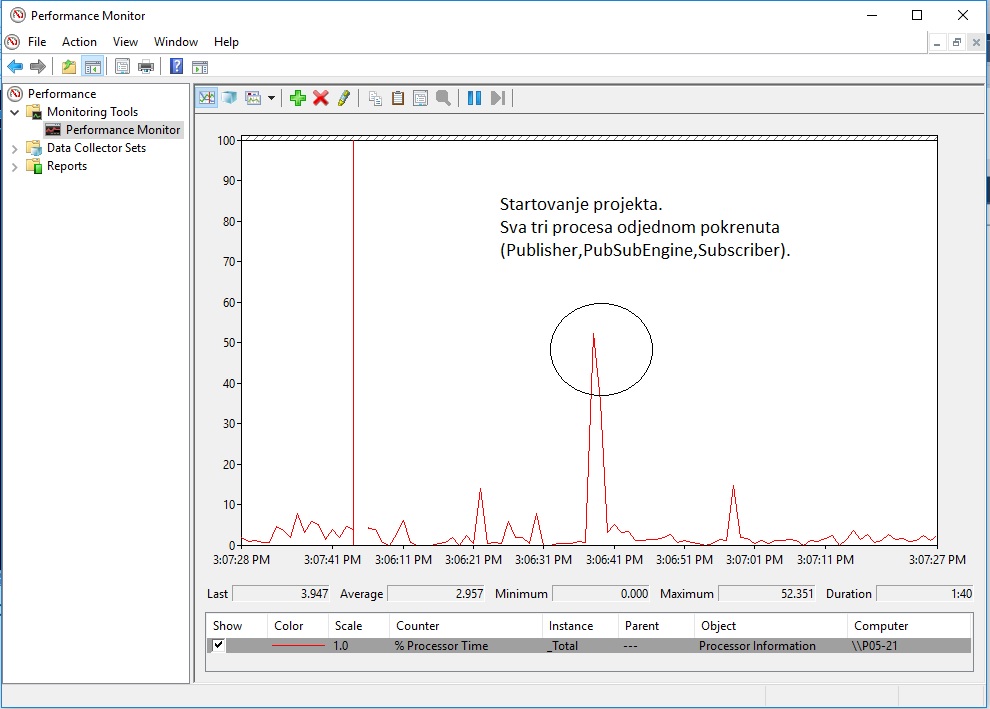


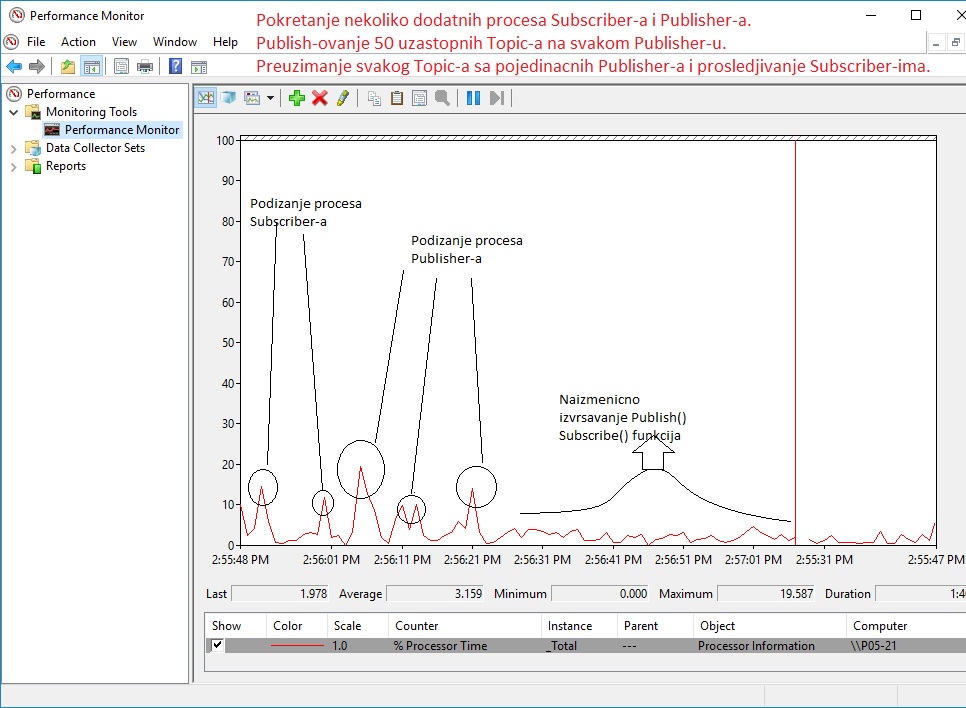
**Rezultati testiranja i zakljucak**

Stres test – podizanje vise procesa Publisher-a koji u beskonacnoj petlji salju ogroman broj publish-eva. Postepeno podizanje procesa Subscriber-a, subscribe-ovanje na sve aktivne publisher-e.

Analiziranjem rezultata stres testa otkiveno je manje curenje memorije (do 180kb). Ovo se desava zbog same prirode realizacije projekta, u kojem niti ostaju aktivne, memorija se alocira za svakog novog subscriber-a i nikad nece biti oslobodjena.

Uz pomoc alata Performance Monitor, analizirana zauzetost procesora od strane aplikacije.





**Potencijalna unapredjenja**

Na osnovu zakljucka, mesta za unapredjenje ima u vidu ocuvanja memorije. Napraviti threadpool, umesti sadasnjeg niza niti, koji bi smanjijo trosenje vremena i memorije na nepotrebne niti.

Takodje, predlaze se koriscenje optimalnijih struktura u slucaju Switcha(Tabele). Koristiti HashSet, gde ce Topic vrednosti biti zakucane i smestene u vector, i svaki bi bio kljuc za spregnutu listu subscriber-a(trenutno je ta struktura queue subscriber-a, zbog predlozenog dizajna). Dobili bismo mnogo brzu operaciju subscribe-ovanja kao i publish-ovanja.