Fakultet tehničkih nauka
Univerzitet u Novom Sadu

**Industrijski komunikacioni protokoli u elektronergetskim sistemima**

# PROJEKAT: REPLIKACIJA

**Mentor:**
**Saša Tošić**

**Studenti:**
**Nataša Šćekić, PR 10/2018**
**Đurđica Marić, PR 4/2018**

# 1. Uvod

## 1.1. Opis problema

Jedan od glavnih problema u računarskim infrastrukturama predstavlja istovremeno izvršavanje zadataka odnosno multitasking. Da bi ovako nešto bilo moguće potrebno je uvesti niti u programske kodove koje daju privid da se više zadataka izvršava u istom trenutku. Niti zapravo rade na način da svaka od njih ima određeni zadatak koji se izvršava određeno vreme. Nakon isteka predviđenog vremenskog intervala se prelazi na izvršavanje zadatka neke druge niti. Na jednu nit se vraća, sve dok se njen zadatak ne izvrši do kraja. U procesu prelaska sa jedne na drugu nit može doći do štetnog preplitanja, koje ima nepovoljne efekte po cijeli sistem.

Još jedno od uskih grla u infrastrukturama predstavljaju komunikacioni kanali između komponenti. Ukoliko ih je premalo, vrlo lako može doći do pojave preopterećenja koje za posljedicu može imati čekanje. S druge strane, prevelik broj kanala koji nisu potrebni opterećuje sistem zauzimanjem memorije koja može da se iskoristi za neke druge procese.

## 1.2. Ciljevi zadatka

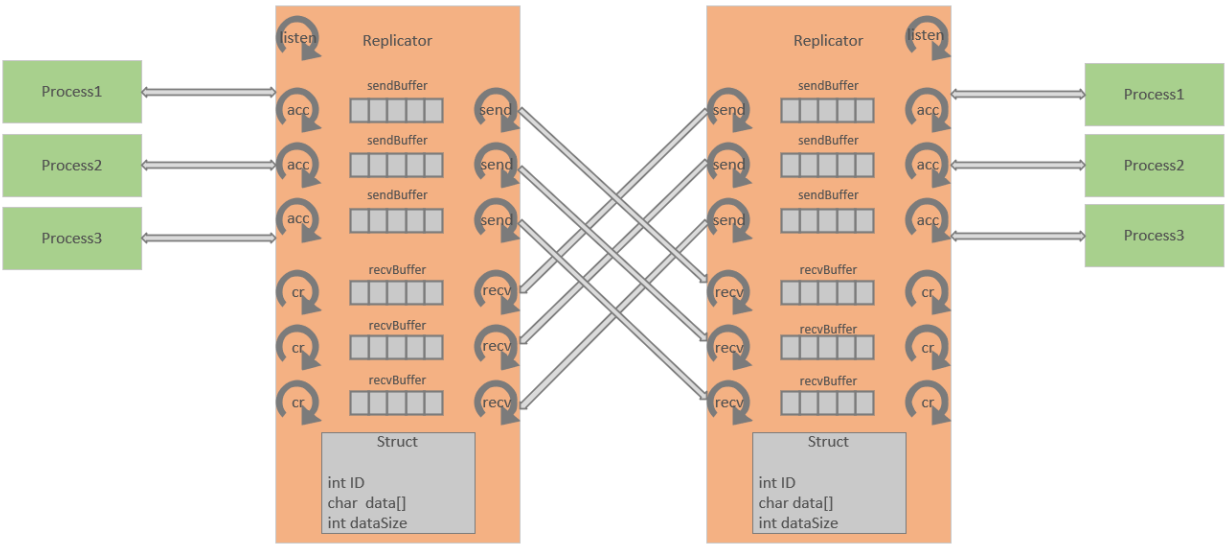
Osnovni cilj zadatka je omogućiti neometanu komunikaciju između *Client* i *Replicator* komponenti. Realizacija je izvršena pomoću četiri niti koje *Replicator* kreira za svaku instancu komponente *Client*. Da bi se izbjeglo potencijalno štetno preplitanje, uvode se kritične sekcije, koje omogućavaju niti da izvrši svoj zadatak bez prekida, odnosno preplitanja sa nekom drugom niti.

Replikatori će imati dva komunikaciona kanala za svaki par klijenata, čime će se izbjeći bespotrebno zauzimanje memorije, a ujedno i otkloniti mogućnost zagušenja.

# 2. Dizajn

Dizajn (*Slika 1.*) se sastoji od sledećih komponenti:

- *Client1*
- *Replicator1*
- *Replicator2*
- *Client2*




*Slika 1.: Dizajn*

Na prikazanom dizajnu se nalaze sledeće niti:

-*listen*, koje predstavljaju *main()* funkciju,
-*acc*, koje predstavljaju C*lientSend()* funkciju,
-*cr*, koje predstavljaju *ClientRecieve()* funkciju,
-*send*, koje predstavljaju *Send()* funkciju,
-*recv*, koje predstavljaju *Recieve()* funkciju.

Svaki od procesa je jednim komunikacionim kanalom povezan sa svojim replikatorom pomoću kojeg komuniciraju. S druge strane, replikatori kreiraju po četiri niti za svaki od procesa koje se registruje na njega. Te niti su zadužene za prijem poruka sa klijenta, slanje poruka na drugi replikator, prijem poruka sa drugog replikatora i slanje poruka na klijenta. Pored toga, na svakom od replikatora se nalaze i dva kružna bafera za svakog klijenta (*sendBuffer i recvBuffer*), kako je definisano tekstom zadatka. *sendBuffer* se koristi za smještanje poruka koje su za slanje, dok *recvBuffer* vrši prijem poruka pristiglih sa drugog replikatora.

Komunikacija između replikatora se odvija na način da ukoliko *Replicator1* želi da pošalje podatke na *Replicator2*, on gađa nit koja je zadužena za prijem podataka na drugom replikatoru, i obrnuto .

## 3. Strukture podataka

U izradi zadatka korišćene su sledeće strukture podataka:

- *messageStruct*
- *listitem*
- *list*
- *RingBuffer*

- *messageStruct*
- *Params*

*messageStruct* struktura se sastoji od sledećih polja:

-ID, tipa int, predstavlja jedinstveni identifikator procesa koji je poslao poruku;
-data[DATA_SIZE], niz koji sadrži elemente tipa char, maksimalne dužine DATA_SIZE, što je u ovom slučaju definisano i ima vrijednost 50, predstavlja niz karaktera koji se šalju u komunikaciji;
-dataSize, tipa int, predstavlja dužinu podataka data.

*listitem* struktura se sastoji od sledećih polja:

-data, tipa messageStruct, koje predstavlja paket koji se šalje/prima;
-next, tipa struct listitem*, koje sadrži pokazivač na sledeći element u listi.

*list* sturktura ima samo jedno polje:

-head, tipa listitem* odnosno pokazivač na prvi element liste.

*RingBuffer* struktura se sastoji od sledećih polja:

-head, tipa unsigned int, koje predstavlja početak kružnog bafera;
-tail, tipa unsigned int, koje predstavlja kraj kružnog bafera;
-data, tipa messageStruct* je poruka koju smještamo u sam kružni bafer.

*Params* struktura se sastoji od sledećih polja:

-doWork, tipa bool*, koje predstavlja pokazivač na promjenljivu koja označava da li nit treba da prestane sa radom,
-listenSocket, tipa SOCKET, koje predstavlja socket za osluškivanje na replikatoru,
-clientSocket, tipa SOCKET, koji se koristi za komunikaciju sa klijentom;
-connectSocket, tipa SOCKET, koji se koristi za prijem podataka sa drugog replikatora,
-replicatorSocket, tipa SOCKET, koji se koristi za slanje na drugi replikator,
-sendBuffer, tipa RingBuffer*, pokazivač na kružni bafer u koji se smještaju poruke koje su spremne da se proslijede ka drugom replikatoru,
-recvBuffer, tipa RingBuffer*, pokazivač na kružni bafer u koji se smještaju poruke koje su pristigle od drugog replikatora, prije nego što se proslijede klijentu,
-clientData, tipa List*, pokazivač na listu u koju smještamo podatke koji pristignu od drugog klijenta,
-csSendBuffer, tipa CRITICAL_SECTION, koristi se prilikom postavljanja i skidanja poruka sa kružnog bafera za slanje da bi se izbjeglo neželjeno preplitanje,
-csRecvBuffer, tipa CRITICAL_SECTION, koristi se prilikom postavljanja i skidanja poruka sa kružnog bafera za prijem da bi se izbjeglo neželjeno preplitanje,

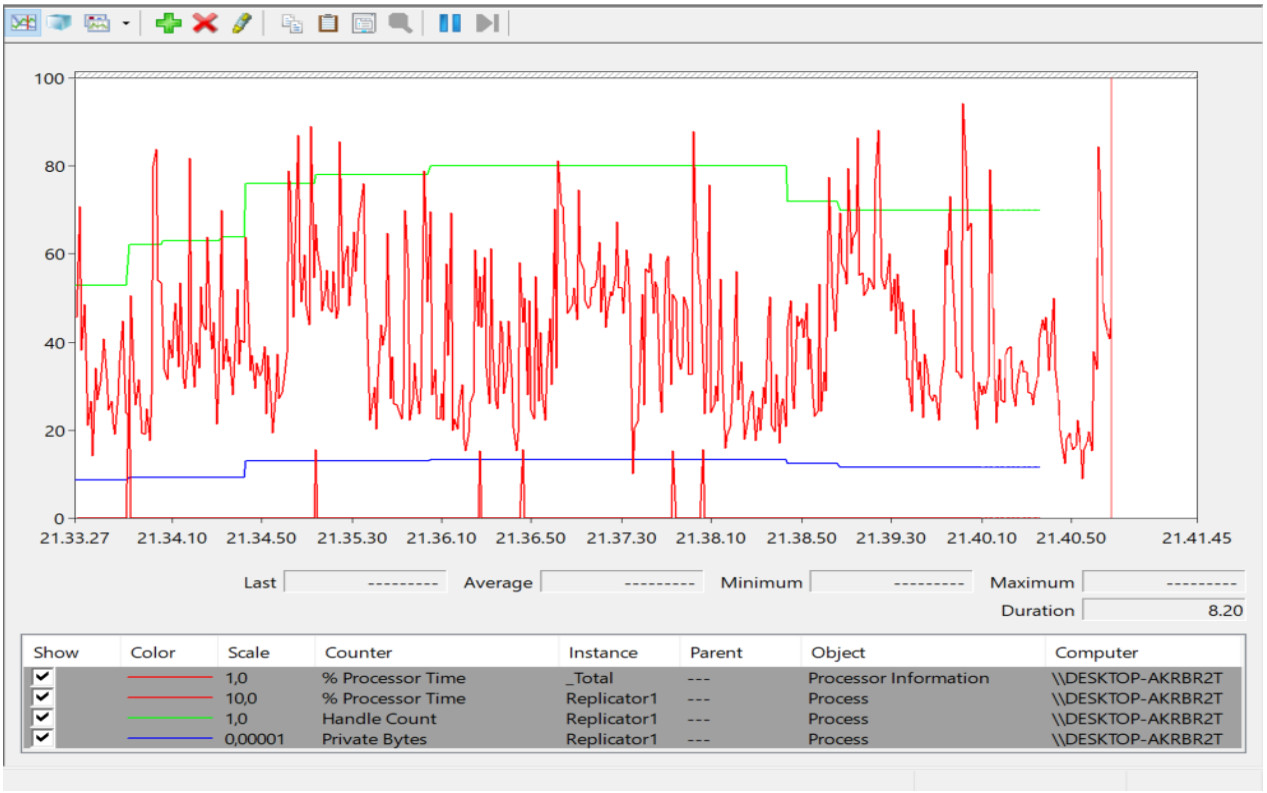
-csClientData, tipa CRITICAL_SECTION, koristi se prilikom stavljanja i uklanjanja podataka iz liste koja sadrži sačuvane poruke klijenta da bi se izbeglo neželjeno preplitanje,
-hSemaphoreSend, tipa HANDLE, koji se koristi za obavještavanje niti Send koja je zadužena za slanje poruka, da je nešto postavljeno u kružni bafer za slanje,
-hSemaphoreRecv, tipa HANDLE, koji se koristi za obavještavanje niti Recieve koja je zadužena za prijem poruka, da je nešto postavljeno u kružni bafer za prijem.

Razlozi upotrebe gorenavedenih struktura:

*messageStruct* – formirana na osnovu parametara koji se prenose u komunikaciji, definisanih tekstom zadatka,
*listitem*, *list* – najintuitivniji način skladištenja registrovanih korisnika i podaka na njima,
*RingBuffer* – definisano tekstom zadatka,
*Params* – pri kreiranju niti na replikatoru, moguće je proslijediti samo jedan parametar, dok je prilikom izrade zadatka bilo potrebno više njih. S obzirom da se radi o ugrađenim funkcijama i nije bila moguća njihova izmjena, svi neophodni parametri su smješteni u jednu strukturu koja se prosljeđuje.

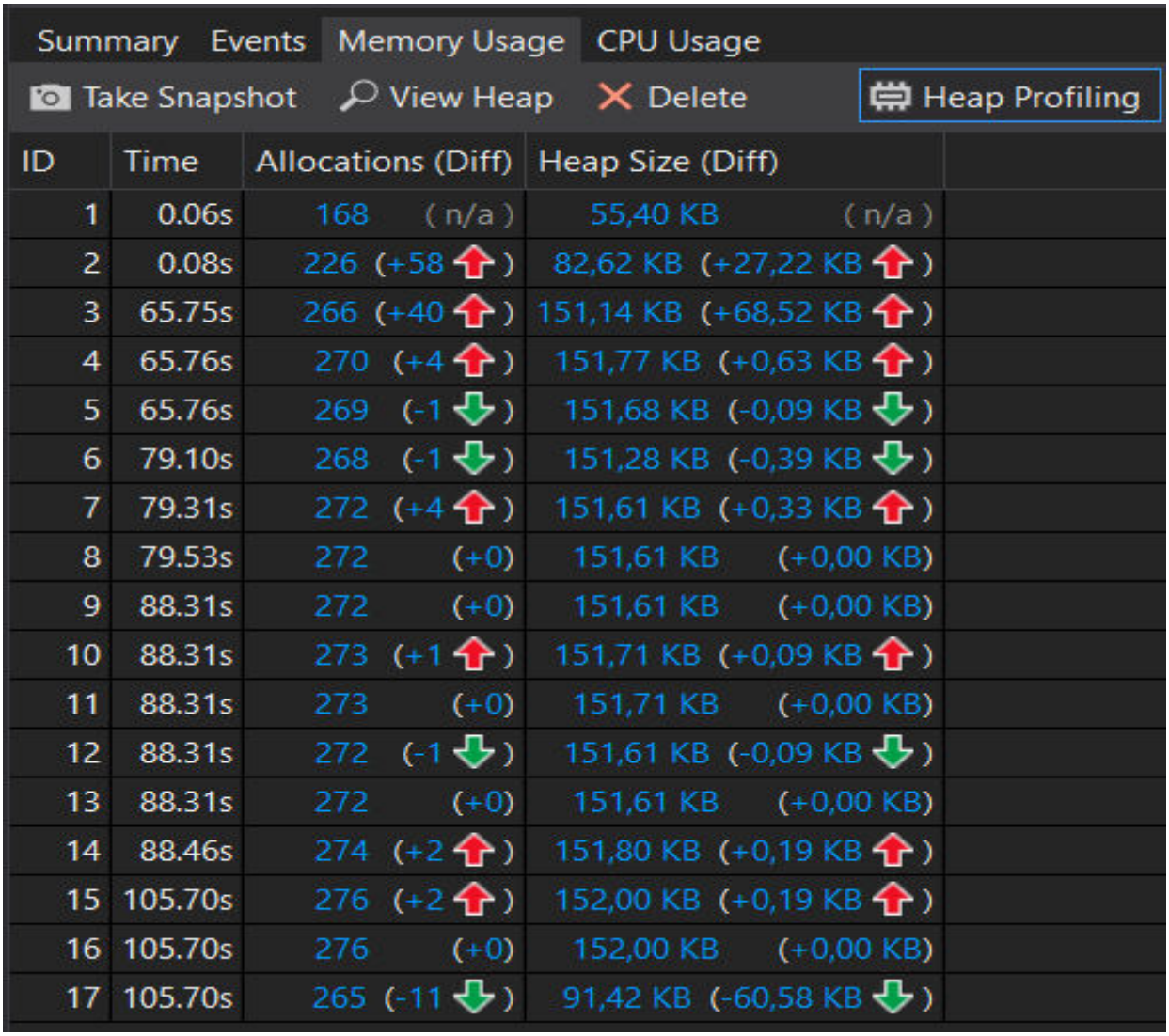
## 4. Rezultati testiranja

Prilikom testiranja, da bi se precizno vidjela efikasnost aplikacije i njene performanse, urađen je određen Stress test.(*Slika 2, Slika 3*)




*Slika 2.: Dijagram Performance Monitor*

Iz dijagrama dobijenog iz *Performance Monitor* aplikacije, izvodi se zaključak da prije izvršavanja prve linije main funkcije programa *Replikator*, postoji već zauzeta memorija, kao i Handle-ovi. Na početku njenog rada, ti brojevi se povećavaju jer se kreiraju neophodni Handle-ovi i zauzima se memorija. Kada rad aplikacije privodimo kraju, zatvaraju se Handle-ovi i oslobađa se memorija, međutim računar se ne vraća na stanje u kojem je bio prije pokretanja aplikacije jer neki Handle-ovi ostaju otvoreni, a memorija još uvijek zauzeta. Ovo se riješava gašenjem servera, nakon čeka se performanse računara vraćaju na prvobitno stanje.




| ID | Time | Allocations (Diff) | Heap Size (Diff) |
|---|---|---|---|
| 1 | 0.06s | 168 ( n/a ) | 55,40 KB ( n/a ) |
| 2 | 0.08s | 226 (+58 ↑) | 82,62 KB (+27,22 KB ↑) |
| 3 | 65.75s | 266 (+40 ↑) | 151,14 KB (+68,52 KB ↑) |
| 4 | 65.76s | 270 (+4 ↑) | 151,77 KB (+0,63 KB ↑) |
| 5 | 65.76s | 269 (-1 ↓) | 151,68 KB (-0,09 KB ↓) |
| 6 | 79.10s | 268 (-1 ↓) | 151,28 KB (-0,39 KB ↓) |
| 7 | 79.31s | 272 (+4 ↑) | 151,61 KB (+0,33 KB ↑) |
| 8 | 79.53s | 272 (+0) | 151,61 KB (+0,00 KB) |
| 9 | 88.31s | 272 (+0) | 151,61 KB (+0,00 KB) |
| 10 | 88.31s | 273 (+1 ↑) | 151,71 KB (+0,09 KB ↑) |
| 11 | 88.31s | 273 (+0) | 151,71 KB (+0,00 KB) |
| 12 | 88.31s | 272 (-1 ↓) | 151,61 KB (-0,09 KB ↓) |
| 13 | 88.31s | 272 (+0) | 151,61 KB (+0,00 KB) |
| 14 | 88.46s | 274 (+2 ↑) | 151,80 KB (+0,19 KB ↑) |
| 15 | 105.70s | 276 (+2 ↑) | 152,00 KB (+0,19 KB ↑) |
| 16 | 105.70s | 276 (+0) | 152,00 KB (+0,00 KB) |
| 17 | 105.70s | 265 (-11 ↓) | 91,42 KB (-60,58 KB ↓) |

*Slika 3.: Snapshot*

Za vrijeme rada programa *Replikator*, na određenim linijama koda su stavljeni breakpoint-i da bi se dobio presjek stanja datog programa za vrijeme njegovog rada. Prva tri Snapshot-a ukazuju na zauzimanje memorije koja je neophodna za početak rada programa, dok Snapshot-i od broja 4 zaključno sa brojem 16 govore o zauzimanju i oslobađanju memorije u toku rada klijenta. Na poslednjem Snapshot-u (17) prikazan je trenutak kada se klijent ugasi.

## 5. Zaključak

Na osnovu gorenavedenih testova izvode se sledeći zaključci:

- Prije pokretanja samog programa, postoje određeni Handle-ovi, kao i zauzeće memorije
- Za vrijeme rada, zauzimanje/oslobađanje memorije i kreiranje/brisanje Handle-ova se vrši u zavisnosti zahtjeva koji program trenutno obrađuje

- Na samom kraju rada, stanje računara nije isto kao i prije početka rada programa *Replikator* jer svi zauzeti resursi nisu oslobođeni. To će se desiti tek nakon gašenja servera.

Dobijeni rezultati su očekivani i u skladu sa sa samom implemetacijom datog zadatka.

## 6. Potencijalna unapređenja

Na osnovu analize urađenog zadatka, zaključuje se da je rad programa moguće optimizovati na neki od sledećih načina:

-Memoriju, koja ostane zauzeta, i Handle-ove, koji se ne obrišu, trebalo bi osloboditi, odnosno obrisati, za vreme rada programa.

-U slučaju poziva funkcije *Recieve* šalju se svi podaci koji se nalaze na drugom klijentu. Da bi se izbjeglo slanje podataka koji se već nalaze na klijentu koji je pozvao funkciju, potrebno je provjeriti koji su to i poslati samo one koji mu nedostaju. Na ovaj način bi se rasteretio komunikacioni kanal jer bi se manje podataka slalo.

-Pri registraciji novog korisnika, u listu smještamo cijelu *messageStruct* strukturu, čime se zauzima redudantna memorija, iako bi se mogao čuvati samo klijentov ID.