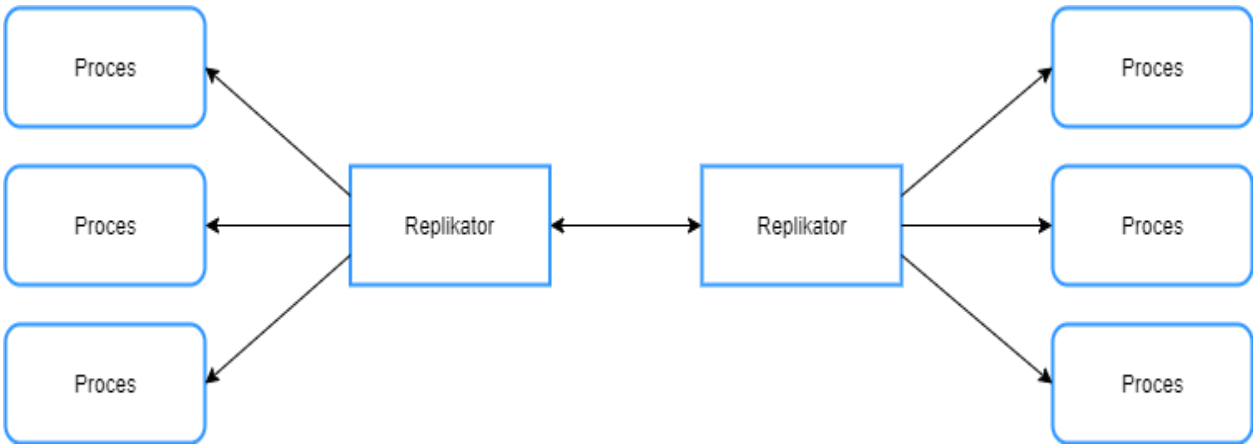
DOKUMENTACIJA – TIM 6

Replicator

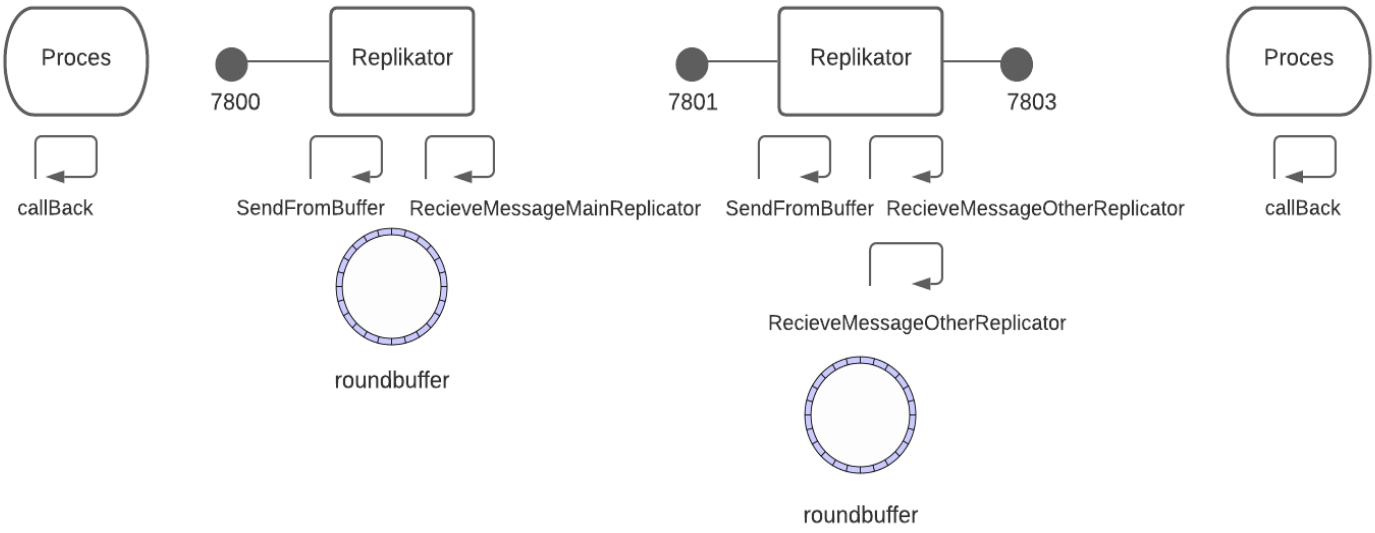
Uvod

Cilj zadatka je bio da se programira replikator podataka. Replikator je servis na koji se povezuje jedan ili više procesa, koji pri pokretanju dobijaju opciju da se registruju svojim jedinstvenim identifikatorom, i ukoliko bi registracija bila uspešna, sa konzole unose podatke i šalju replikatoru. Replikator po prijemu poruke skladišti istu u svoj interni bafer, a zatim je prosleđuje svojoj bratskoj instanci replikatora i ispisuje na konzolu veličinu paketa koji je primio i prosledio. Po prijemu, druga instanca replikatora primljeni paket šalje svim procesima koji su aktivni i registrovani na njemu. Kada proces primi poruku, ispiše na konzolu ID procesa sa suprotne strane koji je poslao poruku, kao i tekstualni sadržaj same poruke. Komunikacija je napravljena da radi u oba smera, tako da procesi sa obe strane mogu da šalju I da primaju poruke.



Slika 1. Tok komunikacije procesa i replikatora

Dizajn



Slika 2. Softverski dizajn

CallBack metoda osluškuje za poruke na procesu, na socketu koji se prethodno povezao na replikator. Kada poruka pristigne, radi se procesiranje iste. Posle deserijalizacije, pretvara je u promenljivu Node strukture, i na osnovu sadržaja poruke proverava kog je tipa. Ukoliko je poruka odgovor na zahtev registracije procesa, u zavisnosti da li je registracija uspešna ili nije, dozvoljava slanje podataka, ili upućuje na ponovno unošenje identifikacionog broja. Ukoliko je proces već registrovan, to znači da je poruka potekla od procesa sa suprotne strane, i ispisuje ID procesa koji je poslao, kao I tekstualni sadržaj poruke na konzolu, a zatim dozvoljava upis stringa za slanje na replikator.

ReceiveMessageMainReplicator osluškuje na socketu glavnog replikatora. Kada poruka pristigne, deserijalizuje je, i prvo proverava da li je poruku poslao drugi replikator, ili proces. Ukoliko je poruku poslao

proces, proverava se kog je tipa poruka, da li je registraciona ili je poruka sa sadržajem. Registraciona poruka pokreće petlju u kojoj se primljeni ID poredi sa svim registrovanim procesima, i u odgovara istom procesu da li je uspešna registracija. Poruka sa sadržajem se upisuje u kružni baferr kako bi SendFromBuffer metoda mogla da je pročita i prosledi. Zatim se proverava da li su već sa tog socketa stizale poruke, da bi se utvrdilo da li ima potrebe da se taj socket upiše u listu socketa registrovanih procesa na replikatoru, jer u suprotnom bi se više puta prosleđivale poruke na isti proces. Ukoliko je ipak poruka pristigla od replikatora, prolazi se kroz listu socketa povezanih i zapamćenih procesa, i na svakom se radi slanje poruke.

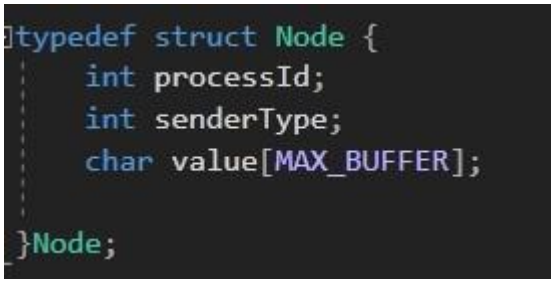
ReceiveMessageOtherReplicator funkcioniše na isti princip kao i na glavnom replikatoru, sa izuzetkom što se metoda pokreće na dve niti, jer drugi replikator osluškuje na dva porta, jedan za poruke od glavnog replikatora, drugi za poruke od procesa. Pri pokretanju, prvo je potrebno pokrenuti sporedni replikator, jer on prvi čeka da mu se javi glavni.

SendDataFromBuffer je petlja koja se vrti i proverava da li je bafer prazan ili ne. Bafer neće biti prazan jedino kada je nit koja čeka poruke procesa/replikatora upisivala u njega. Zatim se poruka preuzima i briše iz bafera, deserijalizuje se u promeniljivu strukture Node, označava se polje senderType na vrednost 1, kako bi se znalo da poruku šalje replikator, a zatim se ponovo serijalizuje i šalje se bratskoj instanci replikatora i nit se uspavljuje na dve sekunde.

Procesi takođe poseduju metode RegisterProcess i SendData koje se bave upisivanjem i slanjem sadržaja poruke.

Strukture podataka

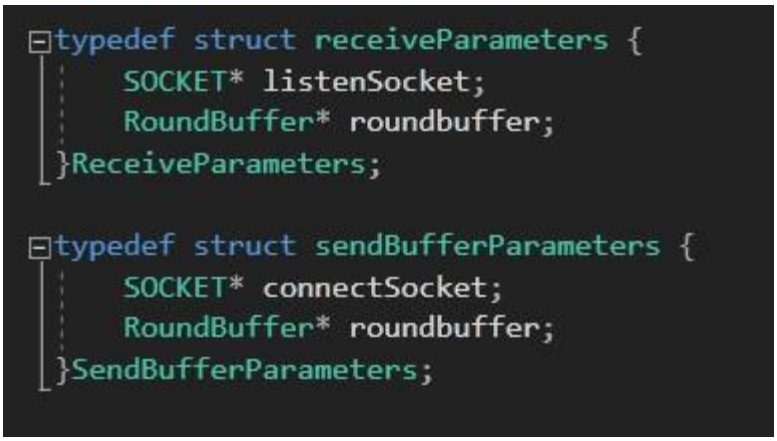
Glavna i osnovna stuktura na kojoj se zasniva komunikacija je struktura Node. Ona sadrži sva polja potrebna za određivanje smera i tipa komunikacije. Funkcije Serialize i Deserialize služe da bi se polja pretvarala u niz karaktera kako bi se olakšalo slanje, kao i obratno.



Slika 3. Struktura Node

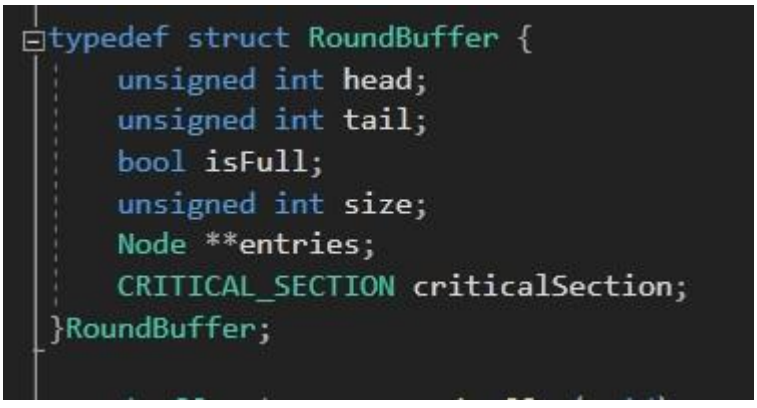
Ukoliko senderType ima vrednost 0 znači da je poruka pristigla od procesa, a u suprotnom znači da je pošiljaoc poruke replikator. Maksimalna dužina stringa za slanje je definisana u zaglavlju “Limitations.h” biblioteke Common, dok je Node definisan u biblioteci “DataNode.h” iste biblioteke.

Strukture koje se prosleđuju metodama SendFromBuffer i RecieveMessage(Main/Other)Replicator su recieveParameters i SendBufferParameters.



Slika 4. Strukture replikatora

Strukture su slične, gde je jedina razlika u vrsti socketa čija se adresa prosleđuje. Replikator je moguće implementirati i korišćenjem samo jedne ovakve strukture umesto dve različite, ali na ovaj način je izbegnuta zabuna za šta se koja struktura koristi. Sem adrese socketa, struktura čuva i adresu strukture RoundBuffer, radi izvršenja operacija nad istim. Definicije ovih struktura se nalazi samo u izvornom fajlu "replicator.cpp" projekta Replikator.



Slika 5. Struktura Buffer-a

RoundBuffer je implementiran tako da sadrži listu pokazivača na strukture Node, a količina dostupnih polja u baferu je postavljena na 4 pri inicijalizaciji. U samoj strukturi se pamti indeks prvog i poslednjeg elementa u baferu, a ukoliko su sva dostupna polja popunjena, može se

pozvati metoda koja alocira dodatna polja za Node-ove. Ova metoda se poziva pri unosu novog elementa u bafer, ukoliko je polje isFull postavljeno na istinitu vrednost. Čuva se I polje strukture kritične sekcije kako bi se sam bafer zaštitio od višestrukog pristupa. Implementacija svih metoda se nalazi u zajedničkoj biblioteci “Common”.

```
SOCKET acceptSocketsMain[MAX_PROCESSES];
SOCKET acceptSocketsOther[MAX_PROCESSES];
int RegisteredProcessMain[MAX_PROCESSES];
int RegisteredProcessOther[MAX_PROCESSES];
```

Slika 6. Nizovi na replikatoru

Nizovi na replikatoru se koriste da bi se sačuvali socketi povezanih procesa, kao i jedinstvenih identifikatora registrovanih procesa.

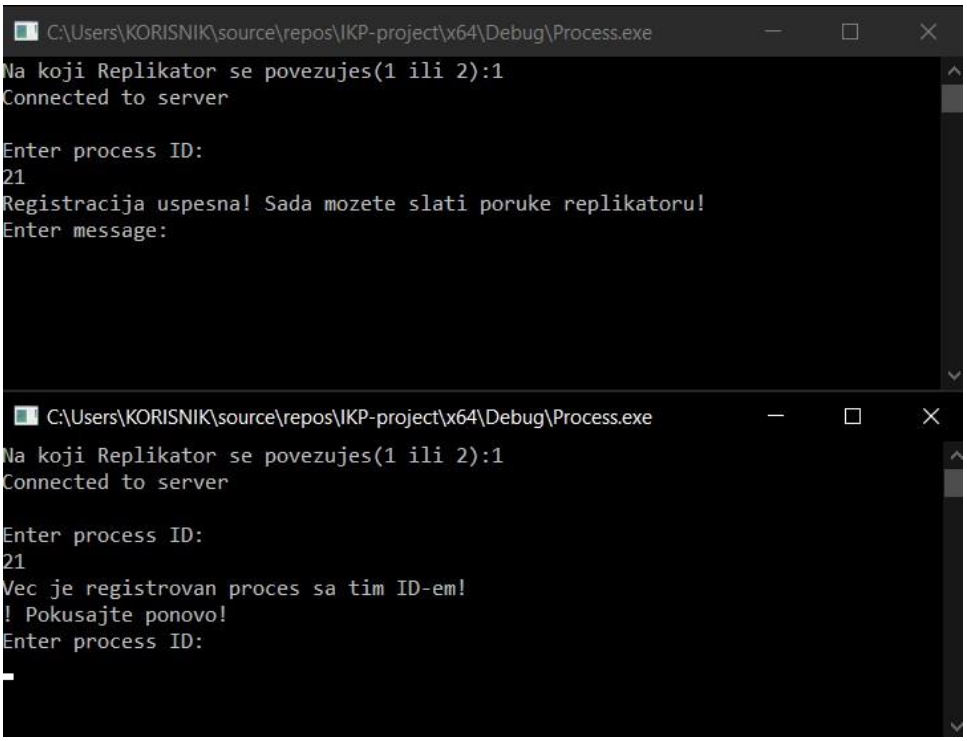
```
Node inbox[100];
int messagesRecieved = 0;
```

Slika 7. Nizovi na procesu

Nizovi na procesu čuvaju sve pristigle poruke na tom procesu.

Rezultati testova

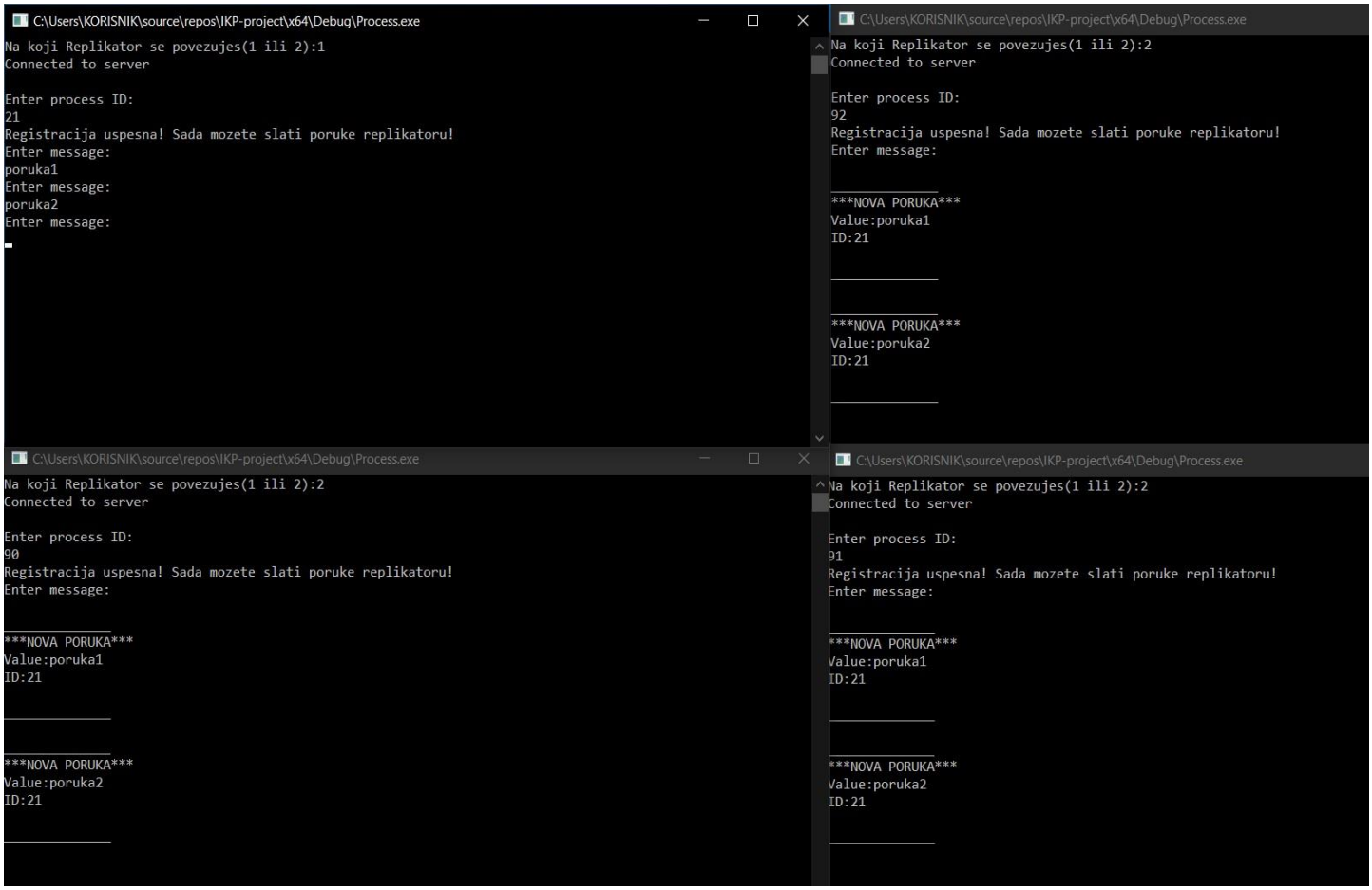
Prvi test koji smo odabrali je testiranje metode za registraciju procesa. U slučaju jedinstvenog identifikatora trebalo bi da prikaže poruke uspešne registracije, a u suprotnom da insistira na ponovnom unošenju ID-a procesa.



Slika 8. Provera registracije procesa

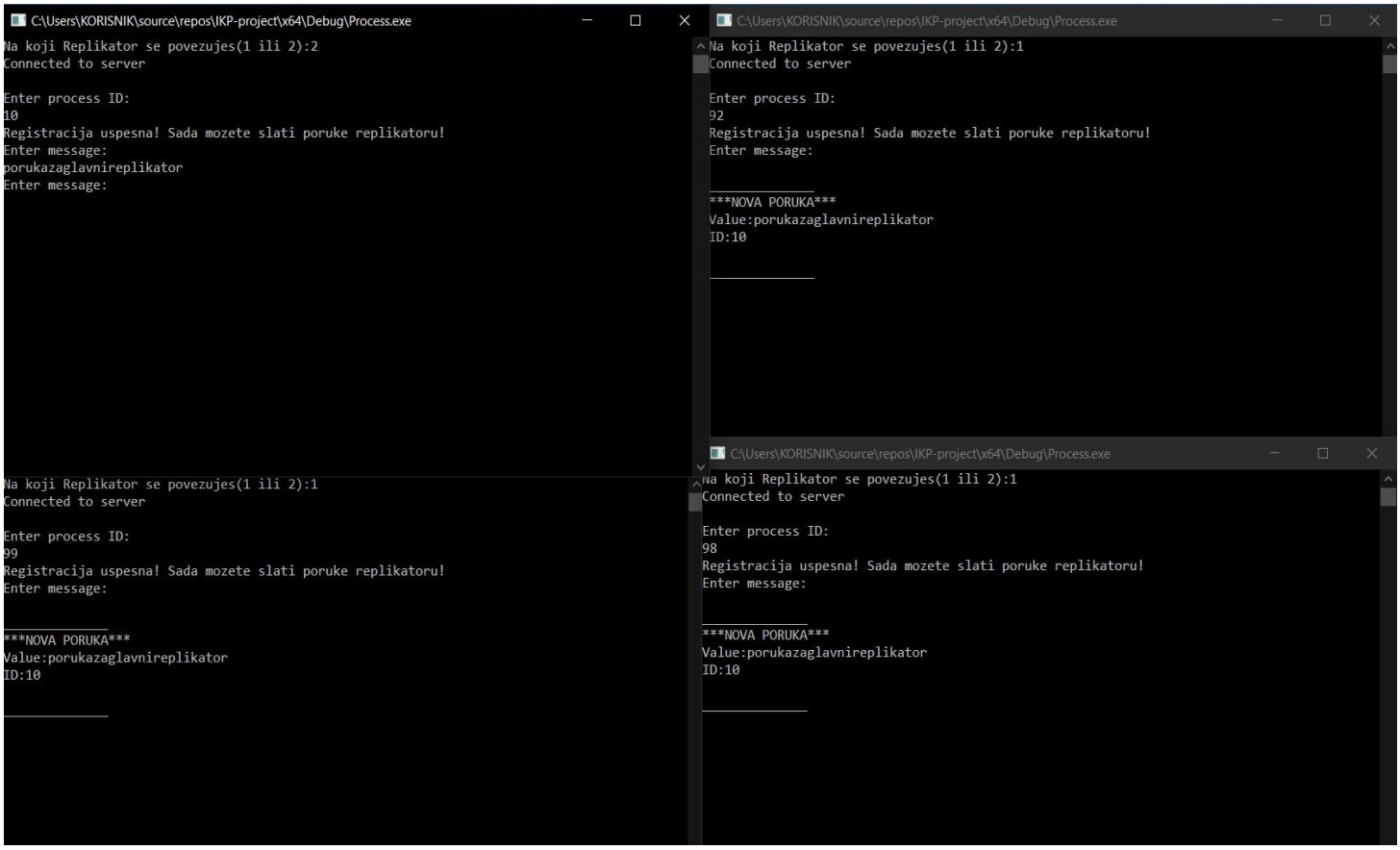
Replikator uspešno uspeva da prepozna da je proces već registrovan na tom identifikatoru, I ne dozvoljava slanje dok se uspešno ne registruju.

Sledeći test je poruke na glavni replikator, i prijem na sve sporedne procese koji su u tom trenutku aktivni.



Slika 9. Test slanja poruke

Sledeći test je isti kao i prethodni, samo je slanje poruke u suprotnom smeru.



Slika 10. Test slanja poruke

Zaključak

Ostvarivanje asinhrone i sinhrone komunikacije procesa u jezicima gde programer ima potpunu kontrolu je mnogo teže nego što se čini na prvi pogled. Obostrana komunikacija više komponenti zahteva planiranje implementacije softverskog dizajna i ispravljanje grešaka u toku razvoja. Dinamički alocirana memorija i konkurentno programiranje niti predstavljaju veliki izazov, kako bi se ne bi dešavalo curenje radne memorije i izbegao višestruki pristup deljenoj memoriji. U toku razvoja,

naš tim je naišao na mnogo grešaka i izazova, kako u planiranju, tako i u implementiranju, što nas je navelo da se više potrudimo i naučimo.

Potencionalna unapređenja

U ovom projektu ostalo je dosta prostora za unapređenja. Rukovanje memorijom je sigurno unapređenje kojim bi se naš tim pozabavio u daljem razvoju softvera. Osim toga, smatramo da kod može biti organizovan mnogo lepše i razumljivije, izdeljen u manje metode i funkcije, što ne bi doprinelo samo čitljivosti, već i funkcionalnosti samog projekta. Korisnički meni na procesima bi mogao biti lepše organizovan i uređen, i trebao bi se smisliti način da replikator sam sebi generiše identifikacioni broj procesa i odgovori mu, čime bi se izbegao prostor za grešku korisnika pri unosu. Uvođenjem dodatnih struktura bi se kod mogao još više pojednostaviti.

Radili: Dimitrije Bajagić, PR64/2017,
Nikola Ostojić, PR57/2017.