Industrijski komunikacioni protokoli u EES

Univerzitet u Novom Sadu

MREŽNI PROTOKOL – IKS-OKS

Industrijski komunikacioni protokoli u EES

26. JANUAR 2026.

AUTORI:

PR33-2022 Katarina Veljković

PR20-2022 Dijana Stojković

Sadržaj

1. Uvod

Opis problema koji se rešava

Cilj ovog projekta je realizacija mrežne aplikacije za igru "Tic Tac Toe" (iks-oks), koja omogućava više korisnika da međusobno igraju partije putem lokalne mreže.
Problem koji se rešava je sinhronizovana i pouzdana komunikacija između klijenata i servera, uz mogućnost detekcije grešaka u prenosu podataka, otpornost na prekide i podršku za više korisnika bez zastoja.

Ciljevi zadatka

- Razvoj servera koji može istovremeno prihvatiti i upravljati većim brojem klijenata i njihovo međusobno povezivanje.
- Razvoj klijent aplikacije sa intuitivnim interfejsom za unos korisničkog imena, prikaz igre i odigravanje poteza.
- Dizajn binarnog komunikacionog protokola, sa validacijom putem checksum-a radi otkrivanja grešaka.
- Upotreba višestrukih niti (multi-threading) na strani servera (i delimično klijenta) za paralelnu obradu korisnika i partija, kao i za obradu poruka o završetku igre.
- Jedinično testiranje osnovnih modula, kao i stress testiranje.

---

2. Dizajn

Opis dizajna implementiranog rešenja, dijagram komponenti

Rešenje je implementirano kao klijent-server arhitektura.

- *Server* je centralni entitet koji upravlja tokom igre — prima zahteve za priključenje, čuva listu čekajućih igrača, automatski ih uparuje u parove za partije (matchmaking) i upravlja svakim tokom igre do kraja.
- *Klijent* je zaseban program koji korisniku omogućava: prijavu (unos korisničkog imena), pregled trenutnog stanja igre, unos i slanje poteza, te prijem rezultata.

Komunikacija se vrši preko TCP/IP protokola koristeći WinSock 2.0 biblioteku.

Svaka funkcionalna celina sistema radi kao posebna nit:

- Na serveru:
    - TOZ (obrada zahteva klijenata),
    - TS (matchmaking servis),
    - TG (obrada igre)

svaka kao zasebna serverska nit, omogućava istovremenu obradu više korisnika i partija.

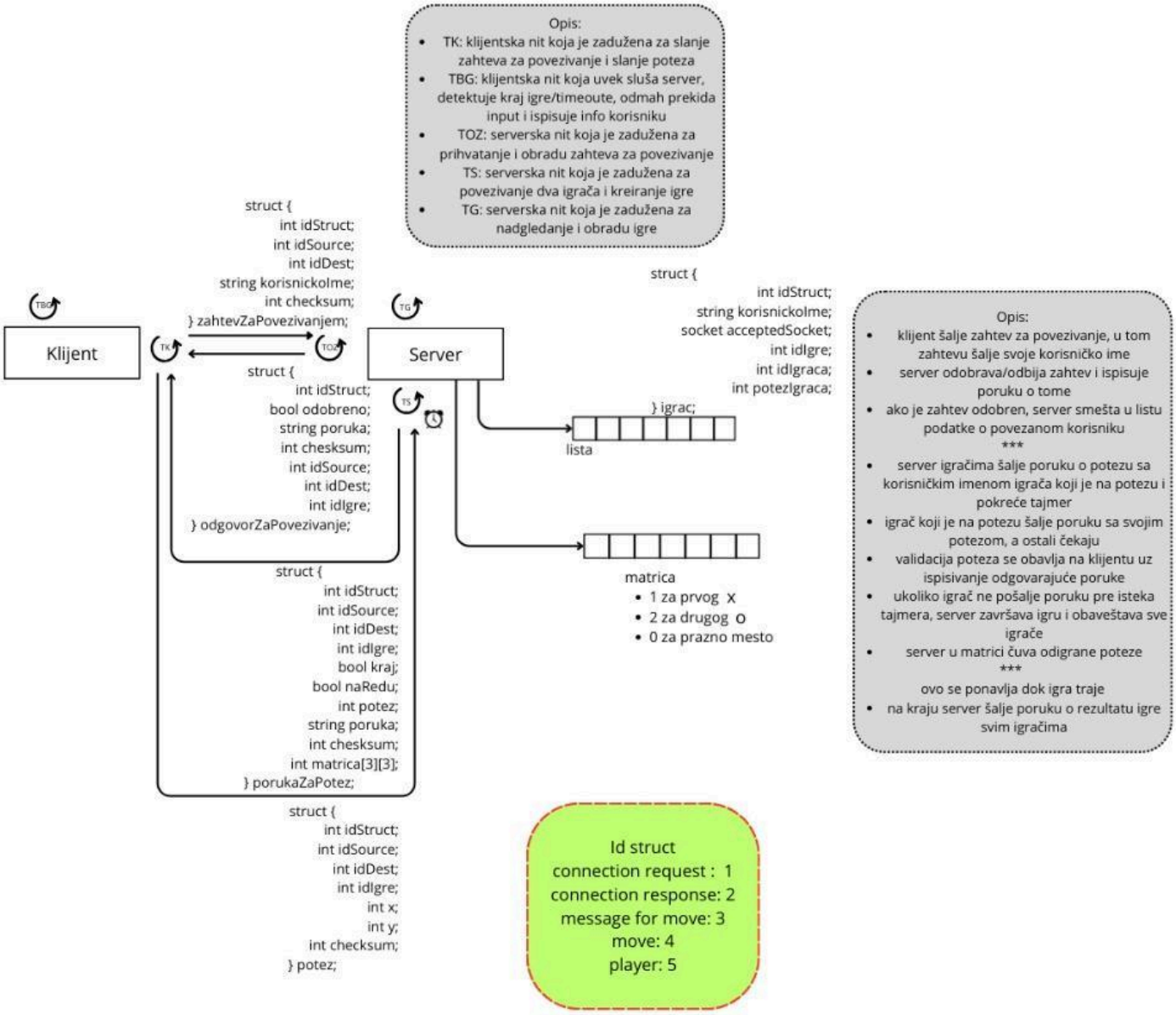
- Na klijentu:
    - Glavna nit (main/UI, unos korisničkih poteza i komunikacija sa serverom)
    - BackgroundListener nit — paralelna nit koja neprekidno presluškuje server. Ova nit omogućava da klijent odmah reaguje na poruke o kraju igre (timeout, diskonekcija, završetak partije), pa korisnički interfejs ostaje responzivan bez obzira na tok igre.

Na taj način, klijentska aplikacija nikada ne "zablokira" u korisničkom unosu kada je igra završena.

- Strukture podataka
    - ConnectionRequest,
    - ConnectionResponse,
    - MessageForMove,
    - Move,
    - Player

Razlozi za izabrani dizajn

- *Server-centric arhitektura*: Server ima kontrolu nad svim logičkim procesima, čime je moguće jednostavno vršiti validaciju i sinhronizaciju između klijenata.
- *Višenitnost*: Svaki igrač i svaka partija obrađuju se paralelno, čime se postiže odlična skalabilnost i nema čekanja/zagušenja među korisnicima.
- *Binarni protokol sa checksumom*: Obezbeđuje brzu razmenu informacija i izuzetno nisku mogućnost prenosa korumpiranih ili izmenjenih podataka.
- *Asinhroni "background listener" na klijentu*: Obezbeđuje responzivnost korisničkog interfejsa i sprečava "zaglavljivanje" inputa u posebnim slučajevima (timeout, povlačenje protivnika, prekid veze).



Slika: Dizajn aplikacije

---

3. Strukture podataka

Razlozi zbog kojih su izabrane baš ove strukture podataka

- *Binarna serializacija custom struct-ova:* Omogućava brzu, "endian-safe" komunikaciju između C++ aplikacija. Svi protokolarni paketi su implementirani kao C++ strukture sa poljima koja najbolje predstavljaju semantiku podataka u igri.

- *Lista čekajućih igrača (List<Player>):* generička povezana lista (*List<T>*) koja omogućava dinamičko dodavanje, čitanje i brisanje igrača u listi čekanja na server. Idealna je za red čekanja u procesu automatskog spajanja igrača (matchmaking).
- *Mutex i atomic tipovi:* Da bi se izbegle trke i potencijalni problemi sa upisom/čitanjem iz zajedničkih podataka tokom rada više niti, korišćeni su mutex-i i atomic flagovi za pravilnu sinhronizaciju (posebno za listu čekanja i prekid/obrada igre).

Korišćene su sopstvene klase za strukture podataka, bez upotrebe standardnih C++ kolekcija. Implementirana je Sve poruke koje se razmenjuju između klijenta i servera (kao i igrački objekti) su definisane kao klase/strukture sa jasno navedenim poljima, što olakšava serializaciju i validaciju podataka.

Opis i pojašnjenje semantike podataka koje sadrže strukture

- *ConnectionRequest* sadrži korisničko ime (username) korisnika koji traži povezivanje i osnovne identifikatore, uz polje *checksum* za proveru tačnosti prenosa.
- *ConnectionResponse* sadrži podatke o tome da li je povezivanje prihvaćeno (polje accepted), odgovarajuću tekstualnu poruku (message), ID igre i neophodne identifikatore, kao i *checksum*.
- *MessageForMove* sadrži trenutno stanje igre: matricu table (board), poruku, informacije čiji je potez (playing), da li je igra završena (end), ID-jeve i broj poteza.
- *Move* prenosi informacije o odigranom potezu – koordinate (x, y), kao i sve relevantne identifikatore (izvor, odredište, igra), uz *checksum*.
- *Player* čuva osnovne podatke o svakom povezanom igraču na serveru: korisničko ime (username), socket, ID-jeve i redosled poteza.

---

4. Rezultati testiranja

Opis testova i pojašnjenje zašto su odabrani baš ovi testovi

U okviru testiranja sistema implementirani su **unit testovi** i **stres testovi**, sa ciljem da se proveri:

- ispravnost funkcionalnosti igre,
- stabilnost sistema pri većem opterećenju,
- pravilno upravljanje konkurentnim klijentima i partijama.

Unit testovi su korišćeni za proveru pojedinačnih funkcionalnih celina (validacija poruka, ispravnost poteza, detekcija pobednika), dok su stres testovi korišćeni za simulaciju realnih uslova rada sa većim brojem istovremenih korisnika.

Izabrani testovi su fokusirani na **najkritičnije delove sistema**, a to su:

- istovremeno povezivanje velikog broja klijenata,
- mehanizam uparivanja igrača (matchmaking),
- paralelno izvršavanje više partija igre.

Zbog toga su odabrana dva stres testa koja najrealnije simuliraju stvarne uslove korišćenja sistema.

Stres test 1 – masovno povezivanje klijenata

Opis testa

U ovom stres testu simulirano je **istovremeno povezivanje velikog broja klijenata** na server. Svaki klijent šalje zahtev za konekciju i ulazi u red čekanja za matchmaking.

Test je osmišljen kako bi se proverilo ponašanje servera pri:

- velikom broju istovremenih TCP konekcija,
- paralelnom kreiranju klijentskih thread-ova,
- konkurentnom pristupu listi igrača koji čekaju partiju.

Rezultati testa

Testiranjem je potvrđeno da server uspešno obrađuje veliki broj istovremenih konekcija.

Rezultati testa za 100 klijenata:

Slika: Performance Monitor za stres test 1

```
================ REZULTATI TESTA ================
 Tip testa:             LOGIN FLOOD
 Broj klijenata:        100
 Ukupno vreme:          1.01671 s
-------------------------------------------------
 Uspesne prijave:       100
 Odbijene prijave:      0
 Greske u mrezi:        0
 Prosecno kasnjenje:    496 ms
=================================================
```

Slika: Rezultati stres testa 1

ID	Time	Allocations (Diff)	Heap Size (Diff)	Label
1	4.66s	316 (n/a)	101.12 KB (n/a)	
3	48.86s	618 (+302 ↑)	151.49 KB (+50.38 KB ↑)	
4	63.78s	528 (-90 ↓)	136.84 KB (-14.65 KB ↓)	
5	92.82s	331 (-197 ↓)	104.70 KB (-32.14 KB ↓)	
7	151.17s	327 (-4 ↓)	102.66 KB (-2.05 KB ↓)	

Slika: Memory snapshots za stres test 1

Stres test 2 – paralelno izvršavanje više partija

Opis testa

U ovom stres testu pokrenuto je više partija igre istovremeno. Svaka partija se izvršava u posebnoj niti, dok igrači paralelno šalju poteze serveru.

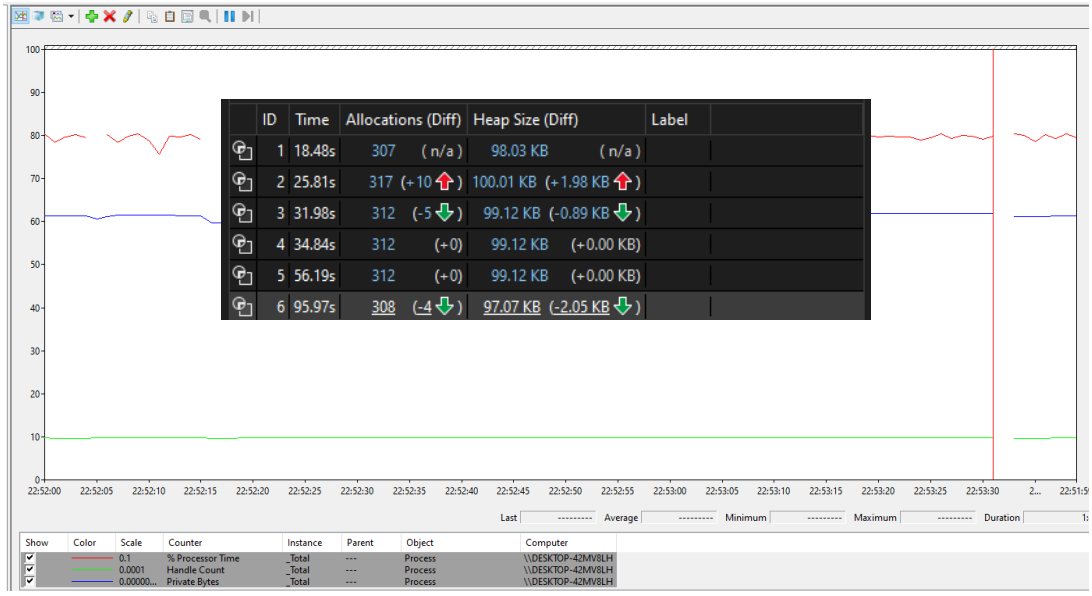
Test uključuje i situacije u kojima:

- igrači kasne sa potezima,
- dolazi do isteka vremena za potez (timeout),
- pojedini klijenti prekidaju konekciju.

Rezultati testa

Rezultati testiranja su pokazali da server uspešno vodi više paralelnih partija bez mešanja stanja igre.

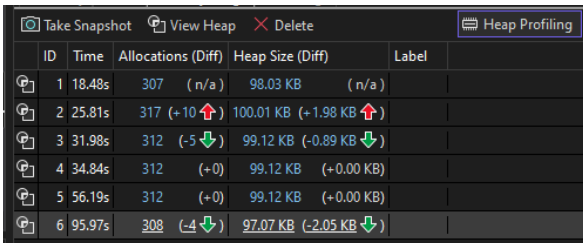
Rezultati testa za 25 partija:



Slika: Performance Monitor za stres test 2

```
================ REZULTATI TESTA ================
Tip testa:              GAME SIMULATION
Broj botova:            50
Ukupno vreme:           26.7339 s
-------------------------------------------------
Zavrsene partije:       25
Ukupno poteza:          197
Greske u mrezi:         0
=================================================
```

Slika: Rezultati stres testa 2



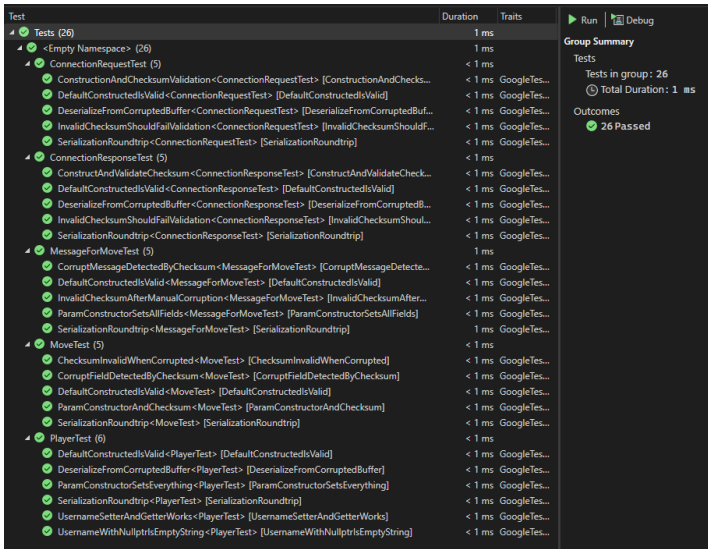
Slika: Memory snapshots za stres test 2

Unit testovi

Opis testova

Unit testovi su odabrani kako bi se proverila ispravnost osnovne logike igre, kao što su:

- validacija checksum-e,
- provera serijalizacije i deserijalizacije,
- provera default-nog konstruktora…



Slika: Rezultati unit testova

Rezultati testova

Svi unit testovi su uspešno prošli. Logika igre funkcioniše u skladu sa očekivanjima, a greške pri nevalidnim potezima su pravilno obrađene.

---

5. Zaključak

Zaključci koji proizilaze iz rezultata testova

Na osnovu sprovedenih testova može se zaključiti da je sistem stabilan i funkcionalan u normalnim i povećanim uslovima opterećenja. Osnovna logika Iks-Oks igre je ispravno implementirana, a sistem pravilno reaguje na nevalidne ulaze i završena stanja igre.

Rezultati stres testova su **bolji od očekivanih**, jer sistem nije pokazao kritične greške ni pri velikom broju istovremenih zahteva. Manje usporenje u vremenu odgovora je očekivano i posledica je ograničenja hardverskih resursa i konkurentnog pristupa zajedničkim resursima.

U budućem razvoju, performanse bi se mogle dodatno unaprediti optimizacijom obrade zahteva i boljim upravljanjem konkurentnim pristupom podacima.

6. Potencijalna unapređenja

Unapređenja

Na osnovu rezultata testiranja i donetih zaključaka, identifikovano je nekoliko potencijalnih unapređenja sistema koja bi mogla dodatno poboljšati performanse, skalabilnost i stabilnost Iks-Oks igre.

1. Korišćenje thread pool-a umesto dinamičkog kreiranja niti

Jedno od mogućih unapređenja jeste korišćenje **thread pool-a** za obradu zahteva umesto kreiranja nove niti za svaki dolazni zahtev.

Kreiranje i uništavanje niti predstavlja vremenski skupu operaciju i može dovesti do povećanog opterećenja sistema, naročito u uslovima velikog broja istovremenih korisnika (što je uočeno tokom stres testova). Korišćenjem thread pool-a, sistem ponovo koristi već postojeće niti, čime se:

- smanjuje vreme čekanja na obradu zahteva,
- poboljšavaju ukupne performanse sistema,
- smanjuje potrošnja sistemskih resursa.

Ovo unapređenje je posebno značajno u scenarijima sa velikim brojem paralelnih zahteva, kao što su masovno kreiranje igara ili istovremeno odigravanje poteza.

2. Optimizacija provere kraja igre

Drugo potencijalno unapređenje odnosi se na **optimizaciju logike za proveru završetka igre** (pobeda ili nerešen rezultat).

Trenutna implementacija proverava kompletno stanje table nakon svakog poteza. Iako je to prihvatljivo za manji broj partija, pri velikom broju aktivnih igara ova provera se izvršava veliki broj puta i može dovesti do nepotrebnog opterećenja sistema.

Optimizacijom ove logike (npr. proverom samo reda, kolone i dijagonala povezanih sa poslednjim potezom), smanjuje se broj operacija po potezu, čime se:

- poboljšavaju performanse sistema,
- smanjuje vreme obrade zahteva,
- povećava efikasnost pri većem broju istovremenih partija.