INDUSTRIJSKI KOMUNIKACIONI PROTOKOLI U EES

Igra pogađanja zamišljenog broja

Projekat 32

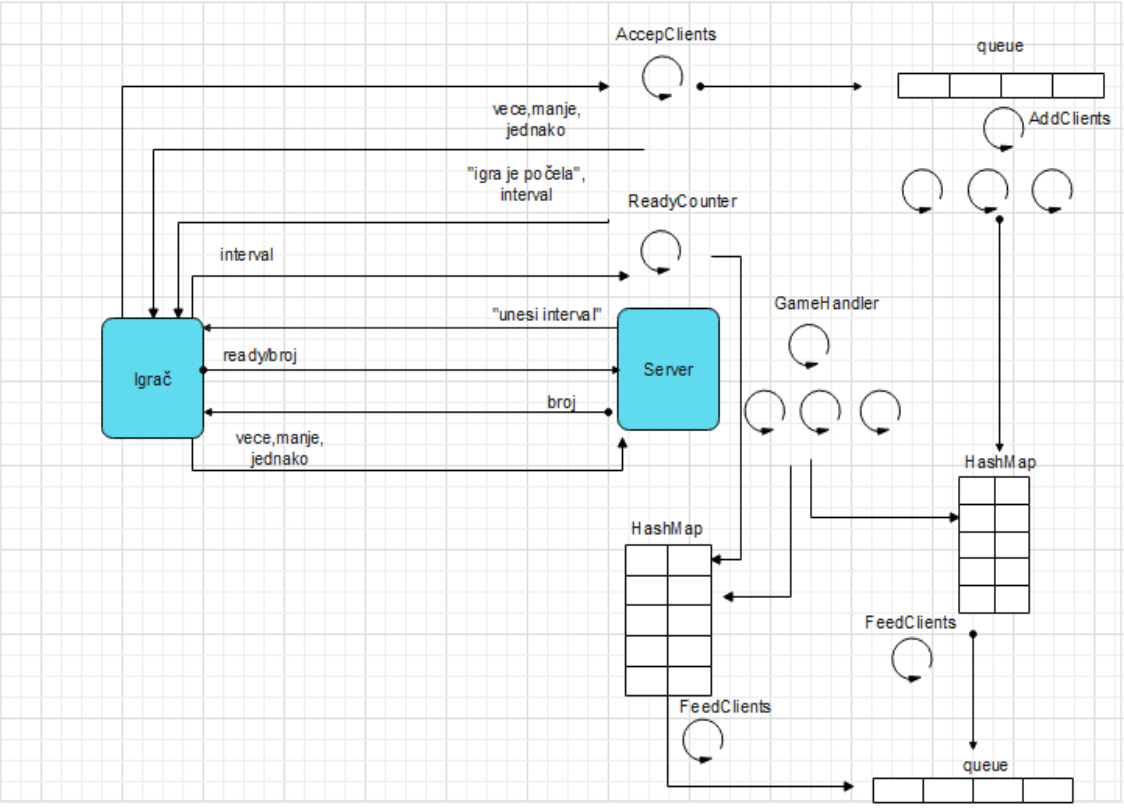
Katarina Prodanović PR15/2019

Marijana Stojanović PR16/2019

1. UVOD

U nastavku će biti opisana klijent-server aplikacija koja omogućava igru pogađanja zamišljenog broja. U igri jedan igrač zamišlja broj, dok neodređeni broj igrača pogađa brojeve. Igrač koji zamišlja broj šalje serveru interval u kome se nalazi zamišljeni broj, zatim server obaveštava ostale igrače da je igra počela. Igrači svoje predloge šalju serveru, on ih prosleđuje igraču koji je zamislio broj, na šta mu on šalje odgovor “veće”, “manje” ili “tačno”. Nakon što server dobije odgovor “tačno” igra se završava.

2. Dizajn aplikacije



Aplikacija je strukturno podeljena na četiri celine: Common, ProducerConsumer, WinSocClient i WinSocServer. U nastavku će biti opisana svaka celina.

2.1 **Common**

Common predstavlja celinu u kojoj se nalaze strukture podataka koje će biti korišćene u aplikaciji. U našem slučaju korišćena je heš mapa u kojoj su čuvani korisnici čiji zahtevi su prihvaćeni. Pored heš mape, struktura podataka koja se koristi je “queue”. U njemu će se u zavisnosti od logike zadatka čuvati korisnici koji su tek pristupili aplikaciji, koji su spremni za igru itd..

Common takođe sadrži .h i .cpp fajlove u kojima su definisane i implementirane funkcije za komunikaciju preko mreže. To su funkcije za prihvatanje novog klijenta, slanje i prijem podataka. Pored toga sadrži enumeracije koje će biti korišćene u implementaciji funkcija za mrežnu komunikaciju.

2.2 **ProducerConsumer**

ProducerConsumer predstavlja odvojenu celinu koja predstavlja nešto što koristi “queue”. Postoje tri semafora, jedan koji ima signala koliko je mesta praznih u redu(hEmptyCounter), on se smanjuje za jedan svaki put kada se nešto novo doda u red, a drugi označava koliko ih postoji u redu(hFullCounter) i kada god se nešto ubaci u red on se povećava za jedan. Treći semafor će kasnije biti opisan.

Pored semafora, funkcije koje su najznačajnije u ovom delu su Produce i Consume. Funkcija Produce ima zadatak da ubaci novi element u “queue”, pri čemu proverava da li ima dovoljno mesta(koristeći hEmptyCounter), ako ima mesta dodaje novi element i povećava hFullCounter, a hEmptyCounter se smanjuje. Funkcija Consume proverava

da li postoji nešto u redu(hFullCounter), zatim ako postoji uzima element i povećava hEmptyCounter, a hFullCounter se smanjuje.

Kao što je ranije spomenuto postoji i treći semafor(finishSignal), njegova uloga je da signalizira da je došlo do gašenja servera. Svaka funkcija u okviru aplikacije imaće proveru da li je taj semafor signaliziran.

2.3 WinSockClient

Pri pokretanju aplikacije klijent ima mogućnost da izabere da li će igru vršiti ručno(svaki put unoseći broj koji želi) ili pomoću binarne pretrage kako bi broj iteracija između servera i klijenta bio što manji.

Ono što je karakteristično za klijentsku aplikaciju je to što klijent ima svoja stanja. Inicijalno stanje je CLIENT_READY. U tom stanju klijent ima mogućnost da se registruje, odnosno pošalje serveru "Ready" kada je spreman za igru i nakon toga prelazi u novo stanje.

Novo stanje klijenta je WAITING. On u ovom stanju čeka da se desi neka nova promena, a ono što se može desiti je da nisu svi igrači spremni, pa ih je potrebno sačekati, pri čemu od servera dobija *READY_COUNT* i ispisuje poruku koliko igrača je spremno, a koliko se jos čeka. Igra počinje kada se za igru prijavilo dva ili više klijenata. Pored toga može se desiti da su svi igrači spremni i da je taj trenutni izabran za onog koji će zamišljati broj, pri tome dobija od servera *CHOOSE_NUMBER*. Zatim bira broj i interval u kome se nalazi zamišljeni broj i šalje ga serveru. Nakon toga prelazi u stanje HOST koje će kasnije biti opisano. Pored te dve mogućnosti, takođe je moguće da su svi spremni ali da nije izabran za onog koji će zamišljati broj, u tom slučaju dobija od servera *GAME_IN_PROGRES*, znači da će biti onaj koji pogađa broj i prelazi u stanje PLAYING.

U stanju PLAYING, u zavisnosti od toga šta je klijent na početku aplikacije odabrao, zamišljeni broj će se pogađati pomoću binarne pretrage ili tako što igrač ručno unosi broj koji želi. Nakon toga dobija odgovor da li je njegov broj manji, veći od zamišljenog ili je ipak tačan. Kada neko pogodi broj ponovo se prelazi u stanje CLIENT_READY, nakon čega može početi nova igra ukoliko su svi igrači spremni.

U stanju HOST koje je ranije spomenuto, nakon sto je odabrani igrač zamislio broj, čeka odgovor od servera sa predlozima brojeva koje su ostali igrači slali, zatim proverava da li je broj pogođen i vraća odgovor serveru. Ukoliko je neko pogodio broj, dobija od servera *FINISHED*, igra se završava i prelazi se u stanje CLIENT_READY.

Napomena: Za slanje poruka između klijenta i servera koristi se struktura DataTransfer koja sadrži:

1. DataType enumeracija - Označava šta se trenutno izvršava u aplikaciji(GAME_IN_PROGRESS, READY_COUNT, FINISHED itd..)

2.Poruka(char Message)

3.Gornja, donja granica i broj koji igrač pogađa

2.4 WinSockServer

Serverska aplikacija implementirana je pomoću nekoliko niti, zadaci koje one imaju bice opisani u nastavku.

AcceptClients čeka da se pojavi neki novi klijent , zatim ga prihvata i smešta u "queue" (*Lobby*) za obradu koristeći *Produce* funkciju koja je ranije spomenuta. Na početku proverava da li se desio finishSignal koji je takođe objašnjen ranije.

AddClients ima zadatak da iz reda preuzima klijente koji su prethodno prihvaćeni i smešta ih u heš mapu (*Clients*). Smeštanje u heš mapu zaštitili smo kritičnom sekcijom u slučaju da ne želimo da neko dodaje nove klijente dok mi radimo nešto sa njima. Takođe, kritičnom sekcijom je zaštićena i sama funkcija koju pozivamo za davanje klijenata u heš mapu (ako više njih radi dodavanje u isto vreme).

FeedClientsToProcess u zavisnosti od toga da li je igra u toku ili nije preuzima ključeve igrače iz heš mape i smešta ih u red. Ako igra nije u toku, iz heš mape *Clients* smešta u red *ClientsQueue,* a ukoliko je igra počela onda smešta iz heš mape *InGame .*

GameHandler proverava da li je neki od igrača poslao odgovor. Ako je poslao "ready" ažurira to u heš mapi *Clients,* a u slučaju da je igra u toku, onda prihvaćene brojeve od igrača koji su pogađali, prosleđuje onom igraču koji je zamislio broj, nakon čega prihvata odgovor od njega (veće, manje ili tačno). Prihvaćeni odgovor insertuje u heš mapu *InGame*.

ReadyCounter se sastoji iz dva slučaja. Prvi slučaj je ako igra nije u toku, već tek počinje. U tom slučaju prolazi kroz sve u heš mapi *Clients* i proverava da li su svi oni spremni za igru, takođe proverava da li postoje bar dva igrača inače igra ne može da počne i potrebno je sačekati dovoljan broj igrača. Nakon toga, ako su ti uslovi ispunjeni, random bira igrača koji će zamišljati broj. Šalje mu CHOOSE_NUMBER kako bi znao da treba da zamišlja broj. Zatim prihvata interval od tog igrača i ostalima šalje da je igra počela i u kom intervalu se nalazi zamišljeni broj. Drugi slučaj se odnosi na to da je igra već u toku. Proverava da li su svi igrači poslali odgovore, ako jesu onda im prosleđuje odgovor od igrača koji je zamislio broj, kako bi znali da li je veće, manje ili jednako. Ako se desi da nisu svi igrači poslali brojeve, svima se šalje poruka da je potrebno sačekati ih. Ako je neki igrač pogodio broj, šalje se obaveštenje svima i igra može ponovo da počne pod uslovom da su svi igrači spremni.

3. Strukture podataka

U projektu se korišćene strukture podataka: Queue i HashMap.

Queue: FIFO struktura koja je korišćena za prihvatanje novih klijenata. Queue se koristi da bi znali redosled po kome su se klijenti prihvatali, kako bi pamtili njihov id i informacije vezane baš za određenog klijenta. Takođe, ova struktura je u našem slučaju bila pogodna zbog provere da li su, redom, svim klijenti spremni za igru.

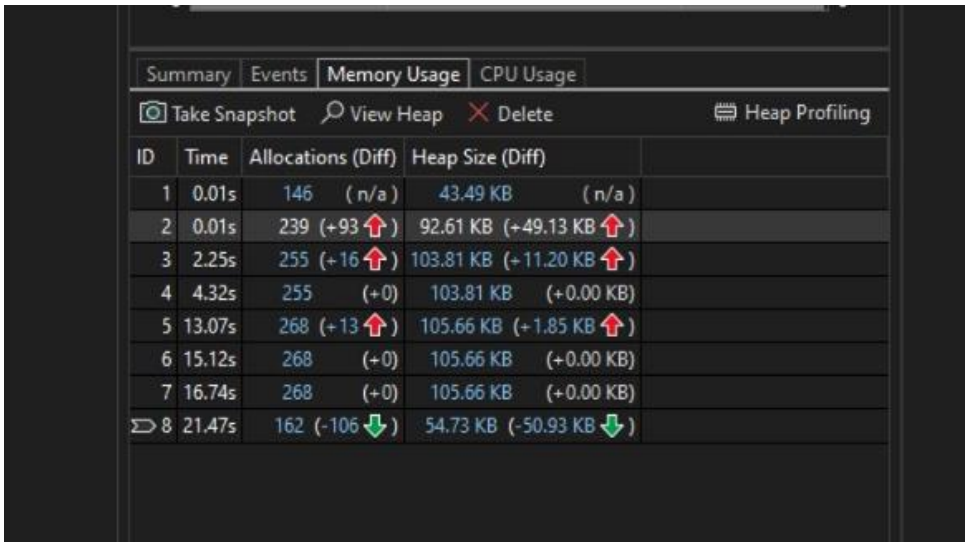
HashMap: Struktura koja omogućava mapiranje ključa na proizvoljan tip podataka. U našoj aplikaciji korišćena je kako bi lakše pronašli igrača koji zamišlja broj, kao i onog koji je pogodio zamišljeni broj.

4. Rezultati testiranja aplikacije

Kako je aplikacija implementirana na način da igra započinje tek kada su svi klijenti spremni za igru, odnosno kada serveru pošalju “ready” i nakon toga mogu početi sa pogađanjem brojeva, ali pri tome svako unosi jedan broj ili se pomoću binarne pretrage ti brojevi sami generišu, a nakon toga je potrebno sačekati da svi igrači unesu svoj broj, stres test se ne može realizovati pomoću slanja velikog broja podataka sa jednog klijenta. Umesto toga, simulirano je pokretanje velikog broja klijenata.

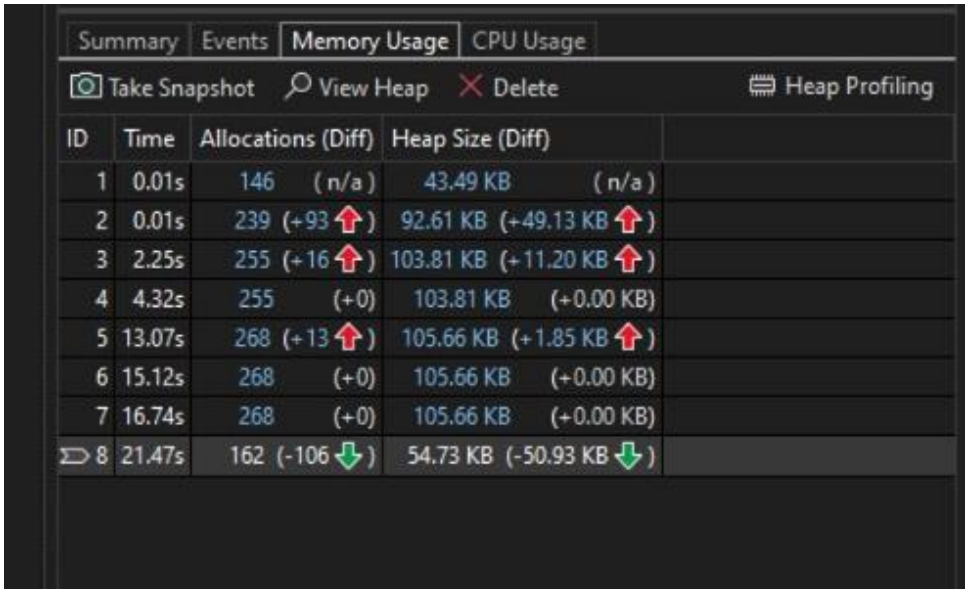
U svrhu testiranja napravljen je novi projekat, koji ima ulogu da simulira pokretanje velikog broja klijenata. U njegovoj *main* funkciji napravljena je nit koja će simulirati klijente. Kada se projekat pokrene na ovaj način, igra će trajati beskonačno, a za to vreme možemo pratiti način na koji se memorija zauzima i oslobađa. Igra se izvršava na način da igrači uvek pogađaju brojeve u istom intervalu i pomoću binarne pretrage, što za nas nije od velikog značaja, pošto je cilj ovog testa pokazati da je sistem optimizovan i za veliki broj igrača.

Na slikama ispod može se videti koliko je memorije zauzeto i na šta tačno se troši ta memorija.

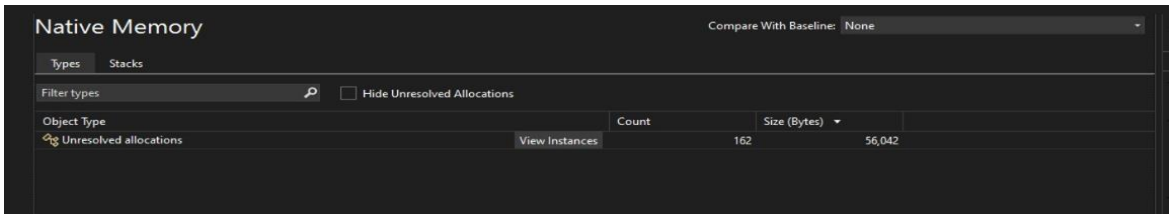


Klikom na strelicu može se videti šta je zauzelo određenu veličinu memorije, što je prikazano na slici ispod.

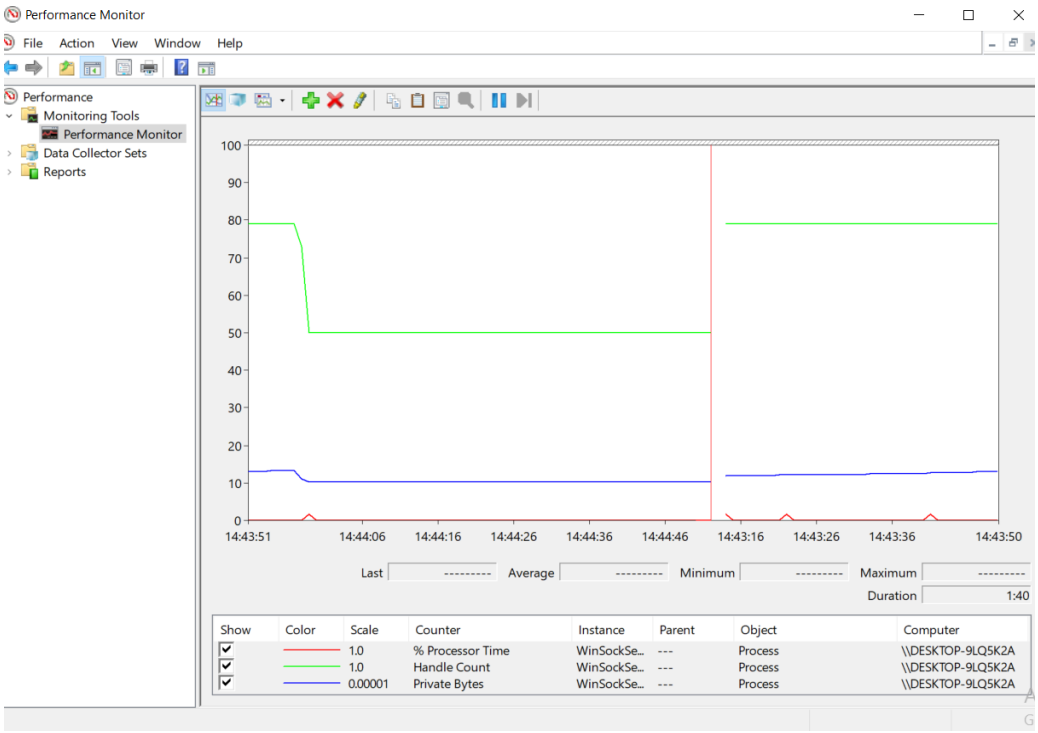
Zatim, pri završetku jednog kruga pogađanja brojeva, može se videti da je memorija uspešno oslobođena.



Zatim, klikom na zelenu strelicu, možemo se uveriti da je sva memorija koja je alocirana u našoj aplikaciji i oslobođena, što se može videti na sledećoj slici.



Pored VS Profiler-a, na dijagramu performansi servera takođe možemo videti da su sve niti oslobođene i da pored toga nema curenja memorije.



5. Zaključak

Rešenje je realizovano klijent-server aplikacijom upotrebom TCP transportnog protokola koji radi u neblokirajućem režimu. Prilagođeno je radu sa većim brojem klijenata, u našem slučaju igrača.

Iz stres testa koji predstavlja simulaciju velikog broja igrača, može se videti da u aplikaciji nema curenja memorije, sto je pokazano pomoću dijagrama performansi, kao i *VS Profiler*-a.

6. Potencijalna unapređenja

Kako bi rešenje bilo još optimalnije i da bi se aplikacija brže izvršavala dobro bi bilo napraviti više niti koje će brojati da li su svi igrači spremni za igru, čime bi se trošilo manje procesorskog vremena. Takođe, dobro bi bilo napraviti više niti koje će proveravati da li su svi igrači uneli broj koji žele,

jer u slučaju da postoji veliki broj igrača, bilo bi potrebno čekati duži vremenski period.