D O K U M E N T A C I J A

Memory Management

Predmet:

Industrijski komunikaconi protokoli u ees

Grupa 33:

Bogdan Drljević PR 106/2021
Vojin Velimirović PR 96/2021

SADRŽAJ:

Uvod

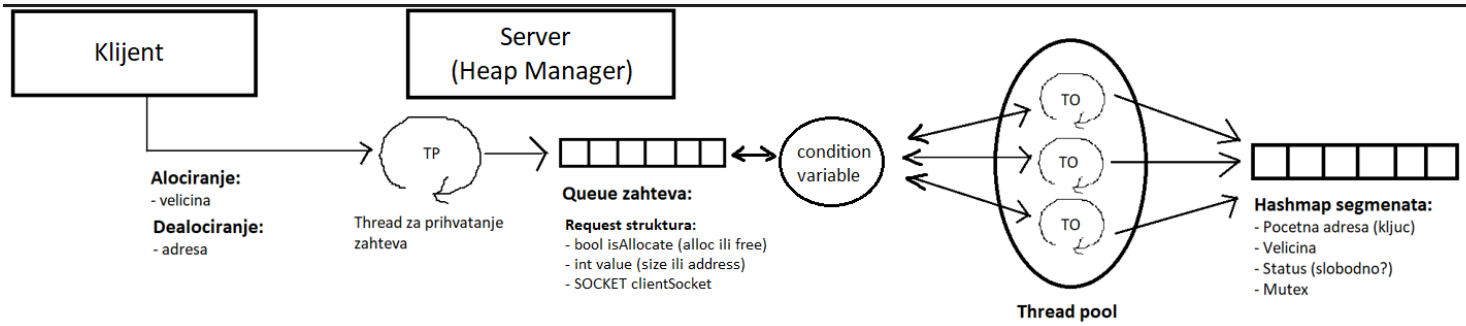
Opis problema koji se rešava

Heap Memory Manager je sistem za alokaciju i dealokaciju blokova memorije. Cilj sistema je omogućiti klijentima izdavanje komandi za zauzimanje i oslobađanje blokova memorije različitih veličina. Server koji obrađuje te zahtjeve mora ih obrađivati na način da se sistemski resursi, poput procesorskog vremena i potrošnje memorije, minimalno upotrebljavaju, istovremeno omogućavajući brz odziv na zadate komande.

Ciljevi zadatka

- Omogućiti klijentima da zauzimaju i oslobađaju memoriju.
- Implementirati serversku aplikaciju koja efikasno obrađuje zahtjeve klijenata.
- Organizovati slobodnu memoriju na način da potrošnja memorije odgovara potrebama sistema.
- Heap manager mora biti dizajniran za rad u okruženju sa više niti.
- Implementirati strategiju za izbegavanje zagušenja u okruženju sa više niti, kako bi se osigurale optimalne performanse i efikasno korišćenje memorije.

Dizajn



Arhitekturu možemo podijeliti na dva dijela: klijent i server (*heap manager*).

Klijent šalje zahtjeve serveru, server odgovara sa povratnom informacijom.

Komunikacija se realizuje korišćenjem TCP/IP protokola sa neblokirajućim soketima. Server je u mogućnosti da istovremeno opslužuje više klijenata.

Klijentski zahtjevi su sljedećeg formata:
(<string, string>) <command_code, value>

- command_code – Označava operaciju koja treba da se izvrši.
  Može biti 1 – allocate_memory ili 2 – free_memory
- value – U zavisnosti od *command_code* parametra tumači se kao *size* ili *address*.

Server prima zahtjeve i stavlja ih u red. Po dodavanju zahtjeva u red spavajuće niti se probude i kreću da izvršavaju zahtjeve, sve dok red zahtjeva nije prazan. Zbog većeg broja zahtjeva i mogućnosti paralelnog izvršavanja ovdje koristimo thread pool.

Memorija na kojoj se vrše operacije je običan niz *Segment* stukture. Da bi se stanje niza pratilo koristimo Hashmapu segmenata koja čuva početne adrese i veličine zauzetih blokova.

Prvobitno niz segmenata bi se realizovao kao bit mapa, ali budući da moramo da obezbijedimo višenitni rad programa moramo uvesti novu strukturu. Svaki segment sadrži svoj mutex, tako da više niti ne može istovremeno pristupiti istim segmentima.

Ovo omogućava da više niti može pristupiti strukturi u isto vrijeme, ako bismo napravili mutex na nivou čitave strukture, ne bismo imali mogućnost paralelnog izvršavanja. Threadpool bi bio beskoristan i došlo bi do zagušenja.

Kao pomoćna struktura je implementirana povezana lista *(eng. linked list)* slobodnih segmenata. Ovo ubrzava rad first fit algoritma, jer više ne mora da iterira kroz niz segmenata element po element. Lista slobodnih segmenata sadrži adresu početnog slobodnog segmenta u nizu slobodnih segmenata i broj slobodnih segmenata (tog niza) do sljedećeg zauzetog segmenta. Samim tim lista slobodnih segmenata ima znatno manje elemenata od niza segmenata, pa je iteriranje kroz nju kraće, u najboljem slučaju *O(1)*.

Za traženje slobodnih segmenata se koristi first fit algoritam.

Strukture podataka

Razlozi zbog kojih su izabrane baš te strukture podataka

Za segmente je izabran niz zbog minimalnog vremena pristupa. Možemo direktno da pristupimo nekoj adresi korišćenjem indeksa, što znači da prilikom zauzimanja ili oslobađanja memorije pristupamo segmentima brzinom *O(1)*.

Za praćenje stanja niza segmenata i bilježenje zauzetih blokova memorije koristi se *hashmapa* zbog minimalnog vremena pristupa. Početna adresa se koristi kao ključ sa kojim se pristupa bloku.

Opis i pojašnjenje semantike podataka koje sadrže strukture

Struktura segment sadrži adresu, tj. indeks, logički tip (*bool*) za zauzetost segmenta i mutex koji onemogućava da više niti pristupi segmenti u istom trenutku, održavajući time konzistentnost.

Struktura bloka sadrži početnu adresu bloka, veličinu bloka i broj segmenata koji taj blok zauzima. Na osnovu početne adrese i broja zauzetih segmenata možemo da dobijemo informaciju o kojim segmentima u nizu se radi. Polje veličina bloka nema funkcionalnu svrhu u realizaciji zadatka, služi prilikom grafičkog prikaza i radi eventualnog proširenja logike bloka.

Struktura FirstFitResult je pomoćna struktura koja pomaže u implementaciji First Fit algoritma.

Ostale strukture su vezane za povezanu listu, red i hashmapu, te se njihova semantika neće objašnjavati.

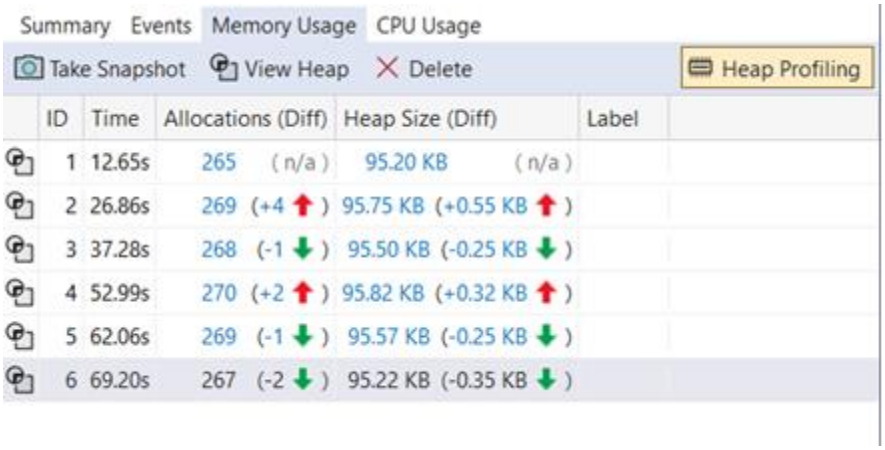
Rezultati testiranja

Opis testova. Pojašnjenje zašto su odabrani baš ti testovi

Kao stres test je izabrano mnogo istovremenih zahtjeva od klijenta ka serveru. Pošto je jedan od zahtjeva da sistem mora da funckioniše u višenitnom okruženju i da se izbori sa zagušenjem, ovaj test će testirati otpornost sistema.

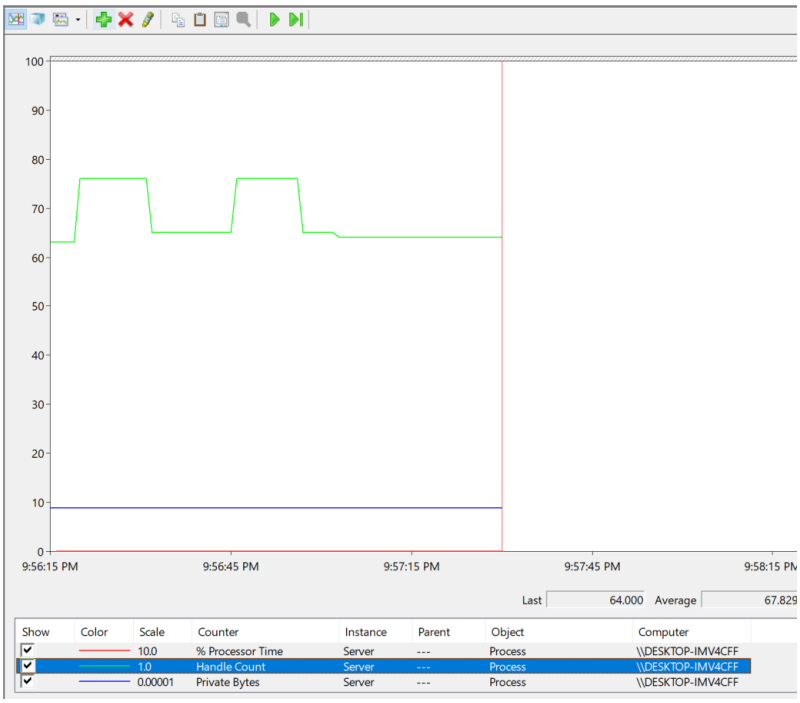
Prikazani rezultati testova

Testiranje memorije:



ID	Time	Allocations (Diff)	Heap Size (Diff)	Label
1	12.65s	265 (n/a)	95.20 KB (n/a)	
2	26.86s	269 (+4 ↑)	95.75 KB (+0.55 KB ↑)	
3	37.28s	268 (-1 ↓)	95.50 KB (-0.25 KB ↓)	
4	52.99s	270 (+2 ↑)	95.82 KB (+0.32 KB ↑)	
5	62.06s	269 (-1 ↓)	95.57 KB (-0.25 KB ↓)	
6	69.20s	267 (-2 ↓)	95.22 KB (-0.35 KB ↓)	

Performance monitor:



Zaključak

Navesti zaključke koji proizilaze iz rezultata testova

Rezultati testiranja memorije pokazuju da se skoro svi resursi koji su zauzeti nakon završetka rada programa očiste što znači da je curenje minimalno, odnosno da su rezultati dobri.

Korišćenje procesorskog vremena je minimalno, što implicira da je rješenje efikasno.

Potencijalna unapređenja

Opis unapređenja koja je moguće uvesti na osnovu onoga što je navedeno u zaključku

Mogu se uvesti naprednije metode sinhronizacije kako bi se smanjili gubici zbog zaključavanja u višenitnim okruženjima.

Navesti i objasniti razloge zbog kojih su predložena potencijalna unapređenja

Korišćenje mutex-a na nivou segmenata pruža osnovnu sigurnost, ali uvođenje mehanizama poput *read-write locks* ili *lock-free* algoritama može dodatno poboljšati paralelizam.