DOKUMENTACIJA – MEMORY MANAGEMENT

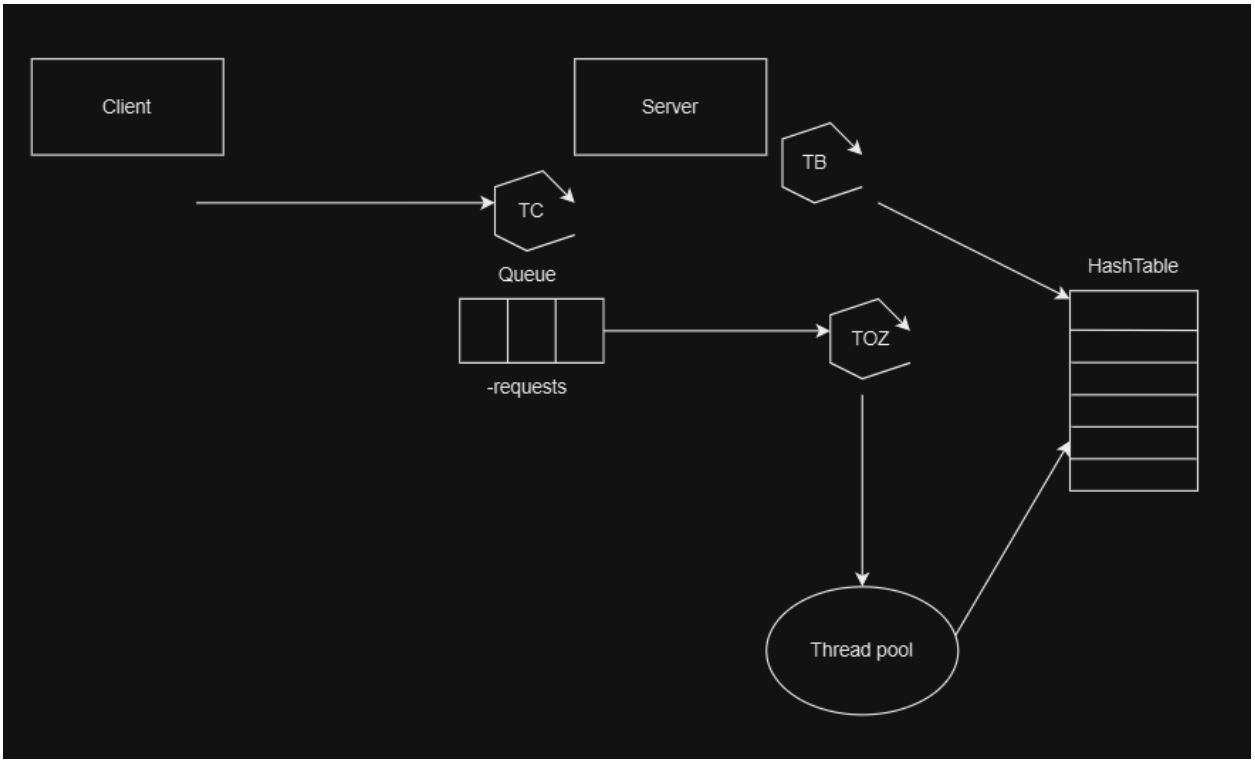
Opis problema:

Problem koji se rešava u ovom projektu jeste implementacija Heap Manager-a koji omogućava dinamičku alokaciju i dealokaciju memorijskih segmenata sa podrškom za rad u okruženju sa više niti, optimizovanom potrošnjom memorije i izbegavanjem zagušenja.

Ciljevi zadatka:

- Potrebno je implementirati **Heap Manager** koji ima podršku za više niti, algoritam za traženje slobodnog prostora i praćenje performansi.
- Implementirati algoritam **next fit** za traženje slobodnih segmenata.
- Obezbediti thread-safety pomoću sinhronizacijskih mehanizama.
- Optimizovati performanse sistema smanjivanjem zagušenja u višedretvenom okruženju.
- Testirati rešenje kroz jednostavnu serversku aplikaciju.

Dizajn:



TC - nit će prihvatati zahteve klijenata i smeštati ih u red (QUEUE), kako bi se zahtevi obrađivali po redosledu kojim su pristigli.
TOZ - uzimaće zahteve iz reda (QUEUE) i prosleđivati ih nitima iz thread pool-a koji će obrađivati svaki zahtev.
Implementiraćemo kritičnu sekciju u obradi zahteva kako bismo sprečili istovremeni pristup istom delu memorije, čime bismo izbegli potencijalne greške.
U zavisnosti od tipa zahteva, uslovna promenljiva će se povećavati ili smanjivati, signalizirajući čistačku nit (TB) da treba osloboditi memoriju.

Strukture podataka:

- **Queue** – Čuvanje korisničkih zahteva.

  Queue smo koristili jer je idealan za obradu korisničkih zahteva u redosledu u kojem su primljeni (FIFO), čime se postiže fer tretman svih korisnika sistema.

- **HashMap** (HashTable) – Brzi pristup memorijskim blokovima po adresi.

  Hashmap smo izabrali zbog njegove efikasnosti u pristupu I manipualciji podacima. Operacije pretrage, dodavanja I brisanja imaju složenost O(1) što ga čini idealnim za upravljanje memorijskim blokovima.Pored toga HashMap osigurava jedinstvenost ključeva čime se eliminiše mogućnost konflikata prilikom upravljanja memorijom. Još jedan značajan razlog za izbor HashMap-e je njegova fleksibilnost I dinamičnost. HashMap može da menja svoju veličinu prema potrebama zahteva našeg sistema.

Opis testova:

Implementirali smo dva testa za simulaciju našeg sistema. Testovi sadrže više klijenata koji istovremeno nasumično šalju zahteve serveru za alokaciju i delokaciju memorije i time proveravamo performanse našeg sistema. Kako test sadrži više klijenata takođe proveravamo rad sa više niti, odnosno zaključavanje kritičkih sekcija pomoću mutex-a.

Sprovedeni testovi su pokazali da implementacija sistema uspešno podržava paralelno izvršavanje operacija sa više klijenata. Red čekanja (queue) na serverskoj strani omogućio je pravilno redosledno obrađivanje zahteva bez gubitaka podataka. Testovi su pokazali da je sistem skalabilan i pouzdan, zadržavajući visok nivo performansi i stabilnosti čak i pod velikim opterećenjem.

Potencijalna unapređenja:

- Hash funkcija trenutno može dovesti do neravnomerne raspodele elemenata po korpama, što može rezultirati situacijama gde pojedine korpe ostaju prazne, dok druge sadrže veliki broj elemenata. Unapredili bi Hash funkciju kako bi ravnomernije raspodelila ključeve, time bi optimizovali operacije dodavanja, pretrage I brisanja.

- Red čekanja trenutno obrađuje zahteve u redosledu kojim pristižu. Potencijalno unapređenje bi moglo uključivati uvođenje prioriteta za određene vrste zahteva.