**1.UVOD**

Heap manager predstavlja komponentu koja rukuje dinamički alociranom memorijom. Zatraženu memoriju stavlja na raspolaganje korisniku, a prethodno je zauzima na memorijskom segmentu koji se naziva heap.

Predmet ovog projekta jeste advanced heap manager (AHM) koji omogućava uporedni rad na više (konfigurabilnom broju) heap-ova i na taj način rešava problem heap contention-a. Heap contention predstavlja jednovremeno pristupanje heap-u od strane više niti, što dovodi do usporavanja programa. Pored AHM projekat sadrži i client-server arhitekturu koja je zadužena za obavljanje ove funkcionalnosti putem računarske mreže, kao i testove koji demonstriraju samu implementaciju.

Funkcionalnosti implementirane u sklopu projekta:

1. Kreiranje i razmena poruka nasumične dužine kroz računarsku mrežu
2. Alokacija (zauzimanje) memorije uz pomoć AHM-a
3. Dealokacija (oslobađanje) memorije uz pomoć AHM-a

**2.ARHITEKTURA REŠENJA**

2.1 Heap manager arhitektura

Heap manager implementiran u ovom projektu se sastoji iz tri komponente:

HeapOperations, HeapManager, AdvancedHeapManager.

2.1.1 HeapOperations

Ova komponenta služi za direktan rad sa Heap-ovima, tj. za direktno rukovanje memorijom.

Ona izlaže operacije stvaranja i uništavanja Heap-ova, kao I operacije za alokaciju i oslobađanje memorije preuzete iz prethodno napravljenih Heap-ova.

2.1.2 HeapManager

HeapManager komponenta izlaže osnovnu funkcionalnost Heap manager-a. Njen osnovni zadatak jeste da drži sve napravljene Heap-ove na okupu i da na zahtev korisnika vrati trenutno selektovan Heap. Selekcija Heap-a se vrši prema Round Robin tehnici, koja nalaže da sledeci izabran element bude sledeći element u nizu, čime se na jednostavan način vrši dobra raspodela posla, odnosno Heap-ova. Definiše strukturu HeapManager, koja služi za centralizovanje podataka samog manager-a.

Komponenta izlaže funkcionalnosti inicijalizovanja i deinicijalizovanja strukture HeapManager-a,

dodavanja novih Heap-ova u manager. Takođe, definiše i funkcije za odabir sledećeg Heap-a, kao i automatsko alociranje memorije date veličine iz sledećeg selektovanog heap-a, uz vraćanje Heap-a iz kojeg je preuzeta memorija. Pored toga, definise i funkciju za vraćanje memorije iz preuzetog Heap-a.

2.1.3 AdvancedHeapManager

Primaran cilj ove komponente je da pruži jednostavan interfejs korisnicima za korišćenje implementiranog Heap manager-a. Drugi zadatak ove komponente jeste da prati poreklo alociranih memorijskih blokova. Definiše strukturu rečnika, koji omogućava „wrap“-ovanje Heš tabele, uz kontrolisanje pristupa iz različitih niti. U rečnik unosi par pokazivač na blok → Heap. Komponenta pruža funkcije za inicijalizaciju i deinicijalizaciju manager-a, kao i funkcije advanced\_malloc i advanced\_free za alociranje, odnosno oslobađanje memorijskih blokova.

2.1.4 Razlozi za odabir dizajna:

1. Izdvajanjem HeapOperation komponente se olakšava promena u implementaciji rada sa memorijom, uz smanjenje prostiranja promena. Takođe, onemogućava se korisniku da koristi nepodržane operacije.
2. HeapManager komponenta je potpuno nezavisna komponenta. To omogućava ponovno iskorištavanje komponente u drugim projektima. Izdvajanjem odvojene komponente se takođe olakšava razvoj celog projekta, jer je sastavljanje testova lakše, bug-ovi se lakše uočavaju i izmene se manje propagiraju.

2.2. Network (client-server) arhitektura

Network implementiran u ovom projektu sastoji se od sledećih komponenti: Client, Server, BaseNetworkOperations i DataNetworkCollections

2.2.1 Client

Komponenta koja modeluje ponašanje više klijenata (multithreading). Klijenti simuliraju slanje poruka nasumičnih dužina koje će se na serveru smeštati na heap koristeći AHM.

2.2.2 Server

Komponenta koja prihvata i uspostavlja konekcije za svakog klijenta i smešta ih u zaseban thread. Nakon toga, za poruku koju prihvati od klijenta zauzme potrebnu količinu memorije i generiše novu koju mu vraća. Server koristi implementirane metode dostupne u okviru AHM.

2.2.3 BaseNetworkOperations

Komponenta koja se bavi komunakacijom između klijenta i servera. Nudi detekciju stanja socket-a, tj. da li je socket u stanju read, write ili u stanju greške, kao i operacije slanja i primanja veće količine podataka.

2.2.4 DataNetworkCollections

Komponenta koja omogućava generisanje poruke nasumične veličine i popunjava je brojevima od 0 – 9 radi eventualnog prikaza.

2.2.4 Razlozi za odabir dizajna

1. Client i Server predstavljaju logički odvojene celine. Više client-a koriste usluge server-a koji se oslanja na implementaciju AHM-a pa ih je bilo neophodno razdvojiti na posebne komponente. Obe ove komponente raspolažu wrapp-ovanim metodama za slanje i prijem poruka koje omogućavaju slanje i prijem veće količine podataka.

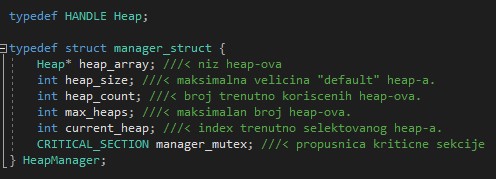
2. NetworkOperations predstavlja komponentu koja obuhvata operacije vezane za razmenu podataka preko mreže i izdvojeno je u posebnu logičku celinu u cilju smanjenja redundancije koda i eventualne ponovne upotrebe ili proširenja funkcionalnosti vezanih za mrežnu komunikaciju.

3. DataNetworkCollections jeste komponenta koja se bavi samim podacima koji se razmenjuju preko mreže. Ideja je da ovaj projekat bude odvojena celina u cilju eventualnog dodavanja struktura podataka koje bi se slale preko mreže i na taj način obuhvatio kompletnu logiku vezanu za podatke.

**3 Strukture podataka**

Strukture podataka koje se koriste u projektu su:

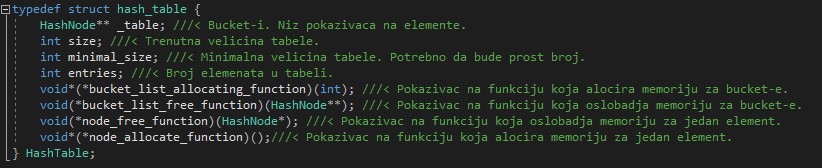
3.1 HeapManager



Predstavlja strukturu koja lokalizuje sve potrebne podatke za funkcionisanje Heap manager-a.

Definise kritičnu sekciju (manager\_mutex) koja omogućava da sve operacije na HeapManager-om budu thread-safe. heap\_array polje predstavlja niz svih mesta za heap-ove. Manager je sposoban da u run-time-u dodaje nove heap-ove, stoga se definiše polje heap\_count koje govori koliko heap-ova je smešteno u manager. Heap-ovi mogu beskonačno da rastu, ali mogu i da budu ograničene veličine, a ta veličina se nalazi u heap\_size polju. current\_heap predstavlja indeks trenutno korištenog Heap-a u nizu Heap-ova.

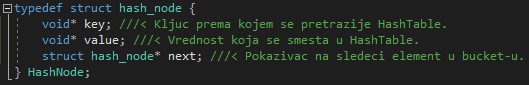
3.2 HashTable



HashTable struktura predstavlja implementaciju hash tabele. Tabela unosi kao vrednosti par pokazivač, Heap, a kao ključ se koristi vrednost pokazivača. Hash tabela je realizovana kao tabela ulančavanja. Ona sadrži niz bucket-a koji sadrže sve elemente na istom indeksu( vrednost hash-a za svaki element modulirana veličinom tablele), tako da svaki bucket predstavlja pokazivač na listu elemenata. Tabela se širi i skuplja u zavisnosti od faktora popunjenosti. Ako je faktor popunjenosti >= 0.75, dolazi do dupliranja tabele, dok kod faktora popunjenosti <= 0.25 tabela se skuplja za pola. Tabela se skuplja samo ako je veća od minimalne veličine koja se prosleđuje pri inicijalizaciji. Zbog bolje distribuiranosti elemenata, tabela mora biti veličine prostog broja, o čemu se sama tabela brine.

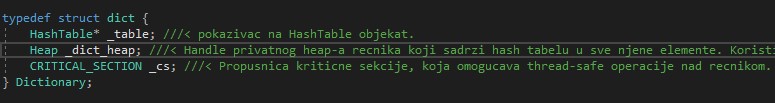
Tabeli je pri inicijalizaciji takođe potrebno proslediti pokazivače na funkcije za alociranje i oslobađanje tabele odnosno jednog pojedinačnog elementa.

3.3 HashNode



HashNode predstavlja strukturu koja čini jedan element u hash tabeli. Za potrebe projekta ključ u tabeli je pokazivač na memorijski blok, a vrednost je Heap u kojem se alocirana memorija nalazi.

3.4 Dictionary



Dictionary je struktura koja predstavlja „wrapper“ za hash tabelu. S obzirom da hash tabela nije „thread-safe“, dictionary u sebi ima kritičnu sekciju, što daje „thread-safety“ svim operacijama nad tabelom. On takođe u sebi ima svoj privatan Heap, u koji se smeštaju svi podaci u tabeli (bucket-i i svi elementi).

3.5 Razlozi za korišćenje struktura podataka

1. Hash tabela se koristi zbog brzog pristupa unikatnim elementima. To je naročitno pogodno za ovaj projekat, s‘obzirom da su pokazivači na memorijske blokove unikatni, a potreban je brz pristup pri oslobađanju prethodno zauzetog memorijskog bloka.

6. Potencijalna unapređenja

Rad aplikacije se može potencijalno unaprediti na sledeće načine:

1. HashTable organizovati da radi u principu otvorenog adresiranja.

Kod otvorenog adresiranja se svi elementi nalaze u nizu, što omogućava brži pristup svakom pojedinačnom elementu. Takođe, na taj način se smanjuje memorijsko zauzeće. Mana ovakvog pristupa jeste to što je potrebno imati veoma dobre hash funkcije, jer su kolizije veoma skupe, i najčešće je potrebno koristiti više hash funkcija za dobijanje indeksa.

2. Unaprediti hash funkciju.

Iako je hash funkcija koja je implementirana u projektu sasvim zadovoljavajućeg performansa, veoma je teško naći perfektnu hash funkciju, tako da je veoma verovatno moguće naći hash funkciju koja brže računa hash, sa veoma malo kolizija.

3. U HeapManager dodati funkcionalnost izbacivanja Heap-ova iz upotrebe.

Na taj način bi se još više generalizovao rad Heap manager-a.