*Universitatea Tehnică Cluj-Napoca*

*Facultatea de Automatică și Calculatoare*

**QUEUE SIMULATOR**

*Tania Loreana Gabor*

*Grupa:30224*

**1.Obiectivul principal**

Proiectarea și implementarea unei aplicații de simulare care vizează analiza sistemelor bazate pe cozi pentru determinarea și minimizarea timpului de așteptare al clienților.

Detalii

Aplicația este făcută pentru a simula un sistem de cozi, prin implementare o strategie multithreading pentru minimizarea timpului de așteptare a clienților,în funcție de numărul de cozi disponibile

Cozile sunt implementate folosind următoarele intrări:

- numărul maxim la care un client se poate poziționa la o coadă

- timpul minim și maxim de procesare

- numărul de cozi

- durata simulării

- numărul de clienți care vor fi serviți.

Cozile sunt utilizate în mod obișnuit pentru modelarea domeniilor din lumea reală. Principalul obiectiv al unei cozi este să furnizeze un loc pentru un „client” unde acesta să aștepte înainte de a i se face un ,,serviciu”.Folosirea unei cozieste utiă pentru a minimiza timpul în care „clienții”vor aștepta la aceasta înaintede a fi serviți. Un mod de a minimiza timpul de așteptare este să adăugăm mai multe servere, adică mai multe cozi în sistemu (fiecare coadă este considerată ca având un fir de lucru asociat). Când se adaugă un server nou, clienții în așteptare vor fi uniform distribuiți la toate cozile disponibile curente.

Aplicația ar trebui să simuleze (prin definirea unui timp de simulare tmaxSimulation) o serie de N clienți , intrarea acestor N clienți în cozi Q, așteptarea clienților de a fi serviți și, în final, părăsirea cozilor. Toți clienții sunt generați la începerea simulării și sunt caracterizați de trei parametri: ID-ul

(un număr între 1 și N), timpul de simulare când sunt gata să meargă la coadă; adică ora la care clientul a terminat cumpărăturile și timpul de procesare (intervalul de timp sau durata necesară pentru a servi clientul de către casier; adică timpul de așteptare când clientul este în fața cozii). Aplicația procesează timpul total petrecut de fiecare client la cozi și calculează media timpului de așteptare total.

1. **Ipoteza,analiza și modelarea datelor**

Înainte de a începe procesul de proiectare, voi încerca să explic utilitatea programului și modul în care prelucrează datele.Să presupunem că dorim să avem 5 cozi pentru clienții care așteaptă, 20 de

clienți,un interval de simulare de cel putin 30 de secunde, astfel incat toti clientii sa fie serviți, indiferent de timpul de serviciu (în limitele minime și maxime ale timpului de serviciu).

Se citesc apoi toate datele de intrare in fisierul cu extensia ".txt". După începerea simulării voi porni firul de lucru principal, care este folosit pentru a urmări timpul actual de simulare și pentru a prelucra clienții prin punerea lor la cea mai bună coadă disponibilă.În timpul procesului de rulare a firului de execuție principal,se printeaza detaliile despre fiecare coadă. Cand intervalul de timp s-a încheiat, programul va tipări concluziilesimulării, care este timpul mediu de așteptare.Timpul mediu de așteptare reprezintă timpul mediu petrecut de fiecare client la coadă, între ora de sosire și ora la care părăsește coada.

Procesul de modelare este definit ca fiind procesul de modulare a unei probleme în mai multe părți mai mici, care sunt mai ușor de înțeles și de depanat. Acest lucru contribuie, de asemenea, la clarificarea unei idei abstracte. În dezvoltarea software, modelarea este esențială pentru a construi o aplicație care are o structură puternică în esență.

Problema noastră, așa cum s-a menționat mai sus în „Obiectivul principal”, include cozi pe care trebuie să le gestionăm, să le optimizăm și să le simulăm pentru un interval de timp dat, cu datele de intrare fiind date într-un fișier de intrare din care acestea vor fi citite.Cozile din viața reală pot fi modelate destul de ușor, folosind datele cu același nume. Ele lucrează pe principiul F I F O, care

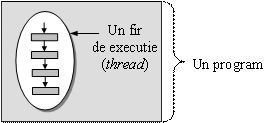
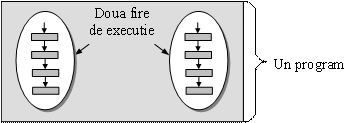
înseamnă :primul venit, primul servit. Putem implementa aceste metode în aplicația noastră

și făcând acest lucru, modelarea cozilor noastre devine mult mai mult simplă.Pentru că, așa cum știm cu toții, cozile evoluează în paralel, cel mai bun mod pentru implementarea acestora este folosind "multithreading" care este o modalitate extrem de utilă. "Multithreading"-ul este capacitatea

unui program sau a unui proces dintr-un sistem de operare pentru a-și gestiona utilizarea către mai mulți utilizatori simultan și chiar să gestioneze mai multe solicitări de către același utilizator fără a fi nevoie de a avea mai multe copii ale programului care să ruleze în computer. Fiecare utilizator solicită un program sau un serviciu și în acest caz un utilizator poate fi și un alt program,deci pentru fiecare utilizator este evidențiat un alt fir de execuție cu o identitate separată. Pe măsură ce programele rulează threadurile actuale sunt întrerupte de alte solicitări, chiar daca firul de lucrul al acelui thread este urmărit până la finalizarea acestuia.

Funcția principală a multithreading-ului este de a efectua mai multe sarcini în același timp. Într-un program Java, aceste sarcini sunt reprezentate ca fire și au o cale de execuție separată. De asemenea, manipularea sau lucrul cu programele Java care folosesc mai multe thread-uri este ușoară,deoarece se poate decide ușor secvența în care are loc executarea firelor Java. Unele dintre

avantajele multithreading sunt: performanțe sporite prin scăderea timpului de dezvoltare, simplificarea urmăririi codului și simplificarea codificării programelor,utilizarea mai bună a stocării în memoria cache prin utilizarea resurselor, scăderea costului de întreținere, utilizarea mai bună a resurselor procesorului.Multithreading-ul nu vă oferă numai avantaje, ci are și uneledezavantaje . De exemplu, procesul de depanare și testare estefoarte complex, implică comutarea contextului, creștepotențialul de apariție a unui eșec, nivelul de dificultate crescut la scriere a unui program și șansa obținerii unor rezultate imprevizibile.

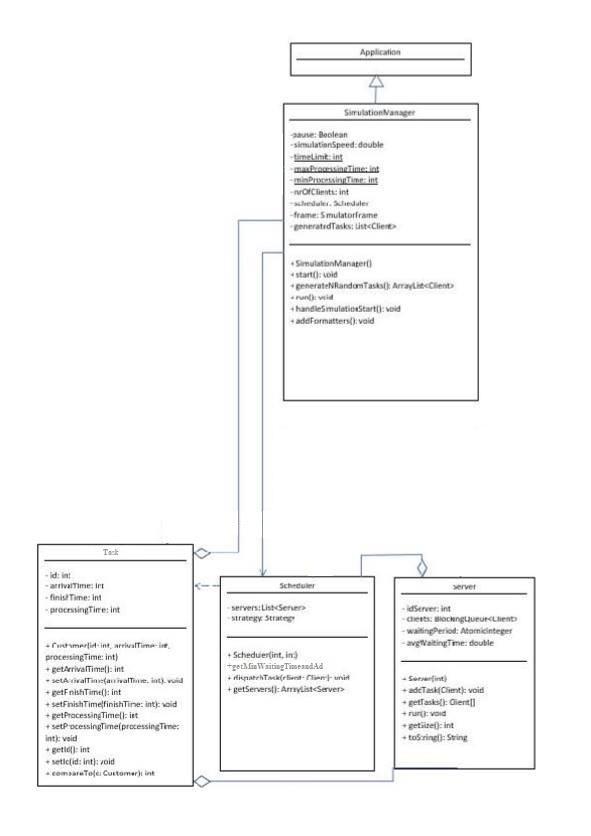
 

1. **Utilitatea aplicației**

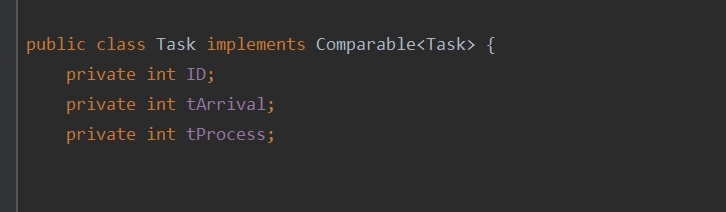
Aplicația software dezvoltată este una de bază, de aceea aceasta doar simulează cozile și le optimizează folosind un algoritm greeedy. Aplicația este utilă pentru organizarea cozilor simple, cum ar fi, de exemplu, cele dintr-un supermarket, oferind clienților indicații pentru coada cu cel mai scurt timp de așteptare.



**4.Diagrama UML**

****

**5.Clasa Task**

****

Clasa Task conține informațiile necesare cu privire la

fiecare client, adică ID-ul generat random al fiecărui client, timpul de serviciu de care are nevoie, stocat ca int și ora la care elajunge la coadă,adică ora la care el începe să aștepte să fie servit, depozitat tot într-un int. Singurul constructor al clasei este cel care primește ID-ul,timpul de intrare în coadă pentru clientul creat si timpul de procesare

Clasa conține, de asemenea, getteri și setteri pentru variabilele particulare

tArrival și tProcess.

Task.java :

Atribute

1. - tArrive: timpul de simulare la care ajunge clientul (la întâmplare între două valori)
2. - tProcess: timpul de procesare
3. -Id-ul

\* tProcess: timpul de care clientul

are nevoie pentru a putea fi servit și scăzut

(la întâmplare între două valori)

Metode: -

getters și setters pentru atribute

- toString (): returnează ca String atributele clientului:

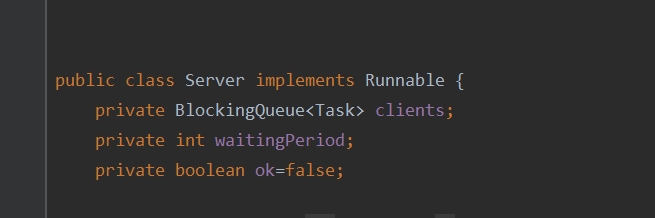
(tArrive ,ID,tProcess)

- compareTo ():

deoarecTaskull implementează interfața Comparable, trebuie să înlocuim această metodă,

în care clasa Task este setată să fie comparată în funcție de ora de sosire.

1. **Clasa Server**

****

Clasa QueueServer o listă de elemente Task,adică o listp de clienți,aceștia fiind clienții care au

s-au alăturat acelei cozi. Coada are, de asemenea, o variabilă waitingPeriod, care este timpul

un client nou va trebui să aștepte dacă se alătură acestei cozi. O nouă instanță a acestei clase este

creat fără argumente, așa cum este vizibil în constructor. Clasa conține mai multe metode.Prima metodă este un getter pentru timpul de așteptare al listei. Următoarea este o funcție care adaugă un client la BlockingQueue a clienților. Această metodă stabilește, de asemenea

timpul de așteptare al clientului și adaugă timpul de serviciu al noului client adăugat latimpul de așteptare al cozii. Următoarele metode reprezintă getteri si setteri pentru atribute.În metoda “run” care

Trebuie suprascrisă se scade timpul de așteptare al fiecărui element din coadă, toate acestea se întâmplă într-un fir separat. Astfel fiecare coadă evoluează în paralel cu celelalte cozi, simulând cozi cât mai reale. O altă metodădin această clasă returnează variabilele boolean”ok”, care

vor fi setatate pe “true” doar când se va sfârși threadul principal de rulat.

Server.java :

Atribute:

a) - clienți: coada de clienți

b) pentru fiecare coadă - waitingPeriod: timpul în care un client trebuie să aștepte pentru a fi servit atunci când este plasat într-o coadă

c)ok-variabila boolean pentru a semnala sfârșitul threadului principal

Constructor:

- Server (): obișnuit să inițializează Serverul cu o coadă goală și timp de aștepare 0(gol)

Metode -

- addTask (Task newTask): adaugă un client la sfârșitul cozii și majorează timpul de serviciu al cozii în funcție de timpul de serviciu al noului client

- run (): această metodă înlocuiește metoda run () a interfeței Runnable și ia capul cozii (primul client)

și îi micșorează timpul de serviciu până la 0, îndepărtându-l după

-getteri si setteri pentru coada de clienți si waitingPeriod specific fiecărei cozi, dar și pentru variabila ok

1. **Clasa Scheduler**

Atribute:

1. - servere: lista de cozi care urmează să fie procesată de programator
2. atribute utilizate pentru inițializarea cozilor

Constructor: - utilizat pentru inițializarea cozilor

Metode:-getMinWaitingTimeandAdd: add un client la coada cu cel mai mic număr de clienți ,

update la noul timp de aștepare al cozii

-getteri si setteri pentru atribute

1. **Clasa Simulation Manager**

SimulationManager.java

Atribute:

timeLimit-timpul în care are loc simularea

maxProcessingTime-timpul maxim de realizare a unui serviciu

minProcessingTime-timpul minim de realizare a unui serviciu

numberofServices-numarul de servicii(numarul de cozi)

numberofClients-numarul de clienți care trebuie generați

minArrivalTimeInterval-timpul minim de intrare în coadă

maxArrivalTimeInterval-timpul maxim de intrare în coadă

Scheduler scheduler-schedulerul conținut , cel care deține și cozile

public List<Server> servers1-lista care se adaugă în constrcutorul schedulerului

Constructor:

- SimulationManager (): constructor necesar pentru inițializarea clasei de simulare în clasa principală ,se iitializeaza cu datele citite din fisierul de intrare

Metode:

- generateNRandomTasks (): inițiază toți clienții și îi introduce în listă, sortându-i după ora de sosire - run (): această metodă înlocuiește metoda run () din interfața Runnable și o iterează prin lista de clienți în fiecare moment din simulare și adaugă clienții la coadă atunci când ora lor de sosire este egală cu timpul curent de simulare

Clasa Simulation Manager este cea care gestionează setul de cozi, de aceea

acesta conține o listă de obiecte de coadă. În setul de variabile de acolo

pot fii găsite atributeleȘtimeLimit-timpul în care are loc simularea,maxProcessingTime-timpul maxim de realizare a unui serviciu,minProcessingTime-timpul minim de realizare a unui serviciu

,numberofServices-numarul de servicii(numarul de cozi),numberofClients-numarul de clienți care trebuie generați,minArrivalTimeInterval-timpul minim de intrare în coadă,maxArrivalTimeInterval-timpul maxim de intrare în coadă

Scheduler scheduler-schedulerul conținut , cel care deține și cozile

,List<Server> servers1-lista care se adaugă în constrcutorul schedulerului

Toate variabilele sunt, de asemenea, ceva definitoriu pentru această clasă.Metoda run sin aceasta clasa este cea mai imprtantă metoda deoarece în această metoda au loc si printările din fisierul de iesire.Tot în această metodă, timpul de așteptare al clienților care nu au apucat sa fie procesati trebuie scazut din timpul total al clientilor.Fiecare coada are propriul ei timp de asteptare. Pentru a realiza timpul mediu am adunat timpii fiecarei cozii iar mai apoiam impartit acest rezultat la numarul de clienti procesati. Tot in metoda run am scăzut din totalul de clienti, clientii care nu au apucat sa fie procesati.

1. **Rezultate**

Aplicația finală este în măsură să calculeze timpul mediu pe care un client trebuie să-l aștepte la coadă și, de asemenea, poate afișa cozile și cum sunt distribuiți clienții la acestea.

1. **Concluzii**

Scopul principal al acestei teme a fost familiarizarea studentului cu programarea asincronă și cu mulithreadingul. Aceasta este prima mea aplicație în care am folosit thread-uri pentru a distribui timpul de rulare general al programului .Am învățat să lucrez mai bine în OOP,am învățat cum să sincronizez threadurile.

1. **Posibile îmbunătățiri**

Unele dintre îmbunătățirile care pot fi aduse aplicației este crearea unei interfete grafice in locul citirii datelor din fisier. Aplicația poate avea, de asemenea, mai multe funcționalități în ceea ce privește parametrii simulării și cum ar trebui să arate raportul final. Mai multe statistici pot fi realizate pe baza parametrilor dati și mai multe strategii de adaugare la coada pot fi implementate . Sistemele reale pot fi modelate în funcție de rezultatele acestei aplicații.

1. **Bibliografie**

https://www.codecogs.com/latex/eqneditor.php https://docs.oracle.com/javase/8/javafx/api/overview-summary.html https://docs.oracle.com /javase/8/javafx/api/toc.html Tehnici de programare - Prelegerea prof. univ. Ioan SALOMIE