

**DOCUMENTATIE TEMA 2**

**Simulator Cozi**

**Kasler Madalina**

**Grupa 30221**

**Profesor Laborator: Dorin Moldovan**

1. **Cerințe funcționale**

Proiectați și implementați o aplicație de simulare care vizează analiza sistemelor bazate pe coadă

determinarea și minimizarea timpului de așteptare al clienților.

Cozile sunt utilizate în mod obișnuit pentru modelarea domeniilor din lumea reală. Obiectivul principal al unei cozi este să

oferiți un loc pentru ca un „client” să aștepte înainte de a primi un „serviciu”. Gestionarea bazate pe coadă

sistemele sunt interesate să minimizeze timpul pe care „clienții” lor îl așteaptă în cozi înainte

sunt servite. O modalitate de a minimiza timpul de așteptare este de a adăuga mai multe servere, adică mai multe cozi în

sistemul (fiecare coadă este considerată ca având un procesor asociat), dar această abordare crește

costurile furnizorului de servicii.

Aplicația ar trebui să simuleze (prin definirea unui timp de simulare 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛) o serie de N clienți

sosind pentru serviciu, intrând în cozi Q, așteptând, fiind servit și în cele din urmă părăsind cozile. Toate

clienții sunt generați la pornirea simulării și sunt caracterizați prin trei parametri: ID

(un număr între 1 și N), 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 (timpul de simulare când sunt gata să meargă la coadă; adică

ora când clientul a terminat cumpărăturile) și 𝑡𝑠𝑒𝑟𝑣𝑖𝑐𝑒 (intervalul de timp sau durata necesară pentru a servi serviciul

client; adică timpul de așteptare când clientul se află în fața cozii). Aplicația urmărește totalul

timpul petrecut de fiecare client în cozi și calculează timpul mediu de așteptare. Fiecare client este

adăugat la coadă cu timp minim de așteptare când timpul său 𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 este mai mare sau egal cu

timpul de simulare (𝑡𝑎𝑟𝑟𝑖𝑣𝑎𝑙 ≥ 𝑡𝑠𝑖𝑚𝑢𝑙𝑎𝑡𝑖𝑜𝑛). Datele de ieșire vor fi afișate în interfața grafică in timp real pentru a vedea concret momentele în care clienții intră sau ies din coadă, precum și timpul pe care îl petrec la coadă. De asemenea, la final se va afișa și ora de vârf a tuturor cozilor (adică momentul de timp în care se aflau cei mai mulți clienti in cozi), timpul mediu de procesare și timpul mediu de așteptare.

**2.Obiectivul temei**

**2.1Obiectivele principale**

Această temă a avut ca și obiectiv principal crearea unei aplicații care constă într-un sistem de simulare a unui caz real, și anume statul clienților la o coadă, servind astfel și statistici legate de timpul lor mediu de așteptare sau de procesare. La fel ca și in viața reala, în contextul programării, coada merge pe același principiu, și anume “primul venit-primul servit”. Acest lucru servește sub numele de FIFO care provine de la first-in-first-out. Într-o structură de date de tip FIFO, primul element care a fost adăugat va fi mereu si primul element care o să fie sters. Cel mai bun exemplu din viata reală va fi coada de la un magazin, când clienții așteaptă să plătească, urmând ca apoi sa plece. Mereu primul client care se pune la coada va fi cel care va iesi primul

**2.2Obiectivele secundare**

**Obiectivele secundare ale temei reprezintă pașii careau fost urmați pentru atingerea obiectivului final.**

* **THREADS**

Procesul și firul sunt asociate esențial. Procesul este o execuție a unui program, în timp ce thread-ul este o execuție a unui program condus de mediul unui proces. Firul este o execuție de program care folosește resurse de proces pentru îndeplinirea sarcinii. Toate subiectele dintr-un singur program sunt conținute logic în cadrul unui proces. Nucleul alocă o stivă și un bloc de control a firului (TCB) fiecărui fir. Sistemul de operare salvează doar indicatorul stivei și starea procesorului în momentul comutării între firele aceluiași proces.

Firele sunt implementate în trei moduri diferite; acestea sunt fire la nivel de nucleu, fire la nivel de utilizator, fire hibride. Firele pot avea trei stări rulate, gata și blocate; include numai starea de calcul, nu alocarea resurselor și starea de comunicare, care reduce comutarea aerului.Îmbunătățește concurența (paralelismul), de aceea crește și viteza.

Multithreading vine, de asemenea, cu demerits, mai multe fire nu creează complexitate, dar interacțiunea dintre ele se întâmplă. Un thread trebuie să aibă proprietatea prioritară atunci când există mai multe fire active. Timpul necesar pentru execuție, respectiv la alte fire active din același proces, este specificat de prioritatea thread-ului.

Pentru mine, conceptul de thread și punerea sa în aplicare a fost unul destul de nou. Thread-ul reprezintă cea mai mica unitate de procesare care este programată spre execuție de către program. Acesta este folosit pentru a eficentiza execuția programelor, pentru a le structura, deoarece acestea execută porțiuni diferite de cod în parallel în interiorul aceluiaș mare proces. Nu de fiecare dată thread-urile sunt independente, astfel in timpul execuției poate apărea situația ca pentru ca un thread să iși inceapă execuția, trebuie să aștepte ca altul sa iși finalizeze execuția, iar aici apare conceptul de sincronizare a thread-urilor. Sincronizarea thread-urilor este o tehnica prin care un thread asteaptă execuția altor threaduri înainte de a continua propria execuție.

Pentru a permite calculatoarelor să desfășoare mai mult de o activitate la un moment dat, procesul și firul oferă un serviciu excelent, dar există o diferență între ele în modul în care funcționează. Toate programele care rulează pe un computer utilizează cel puțin un proces sau un fir. Procesul și firul lăsa procesorul să treacă fără probleme între mai multe sarcini în timp ce permite partajarea resurselor calculatorului. Deci este datoria unui programator să utilizeze fire și procese într-un mod eficient pentru a face un procesor cu performanțe ridicate. Implementarea firelor și a proceselor diferă în funcție de sistemul de operare disponibil. Procesul și firul sunt două tehnici folosite de programatori pentru a controla procesorul și executarea instrucțiunilor pe un computer într-o manieră eficientă și eficientă. Un proces poate conține mai multe fire. Firele furnizează o modalitate eficientă de a partaja memoria, deși operează mai multe execuții decât procesele. Prin urmare, firele sunt o alternativă la mai multe procese. Odată cu tendința crescândă spre procesoarele cu mai multe nuclee, firele vor deveni cel mai important instrument din lumea programatorilor. Practic, aceasta poate să fie vazută ca și un magazin cu număr maxim de cozi, un număr maxim de clienți care vor avea ca și limită un timp limită de sosire la coadă și un timp limită de stat la casă, sub forma unui timp de procesare. Aceste date se vor introduce de către utilizator in textfield-urile din GUI.

**3.Proiectare**

**3.1. Structuri de date și clasele folosite**

Clasele folosite sunt: Client, Server, Scheduler, Controller, SimulationManager. Clasa principală

Client va descrie obiectele de tip Client care vor fi introduse în Server. Clasa Server reprezintă cozile în care vor fi stocați clienții. De asemenea se va ocupa cu parcurgerea cozii. Clasa Scheduler realizează conexiunea dintre clasa Client și Server.În această clasă se va aplica strategia de atribuire a clienților in coz Clasa SimulationManager se va ocupa de alocarea cliențlor in cozi, si se va tine mereu cont de strategia stabilită in clasa Scheduler. Clasele ConcreteStrategyTime și ConcreteStrategyQueue implementează interfața Strategy. Strategiile luate în considerare de către aplicație sunt: distriubuirea cliențiol la cozi în funcție de timpul minim sau în funcție de coada care are cele mai puține persoane in așteptare. Strategia folosita de aplicație este cea care alege coada în care va fi distribuit clientul în funcșie de timplul minim de așteptare. Pentru a imi usura munca, ca si structuri de date am ales ArrayList-urile și LinkedBlockingQueues. ArrayListurile reprizintă o structură de date ușor de utilizat, deoarce au extrem de multe metode predefinite,gata implementate, fapt ce ma va ajuta foarte mult in organizarea temei.LinkedBlockingQueues au fost utilizate pentru reprezentarea cozilor de clienți, deoarece oferă o stabilitate foarte mare în lucrul cu thread-urile.

**3.2. Diagrama de clase**

Diagrama de clase este folosită pentru a modela structura (viziunea statică asupra) unui

sistem. O astfel de diagramă conţine clase / interfeţe, obiecte şi relaţii care se stabilesc între acestea.

Relaţiile pot fi de tipul:

• asociere;

• agregare;

• generalizare;

• dependenţă;

• realizare.

Clasele sunt folosite pentru a surprinde vocabularul sistemului ce trebuie dezvoltat. Ele pot

include:

• abstracţii care fac parte din domeniul problemei;

• clase necesare la momentul implementării.

O clasă poate reprezenta entităţi software, entităţi hardware sau concepte. Modelarea unui

sistem presupune identificarea elementelor importante din punctul de vedere al celui care

modelează. Aceste elemente formează vocabularul sistemului. Fiecare dintre aceste elemente are o

mulţime de proprietăţi. **4.3 Algoritmi**

Server – Runnable, extends Thread:

Atributele folosite:

* Clienți(BlockingQueue<Client>… : LinkedBlockingQueue)
* WaitingPeriod (AtomicInteger): această variabila o să fie decrementata de thread-ul curent din Server, odată ce un Client a petrecut suficient timp la coadă ( adica si-a terminat perioada de procesare) și va fi incrementată de Scheduler atunci cand noi clienti vor fi adaugati
* Int indice- pentru a contoriza numarul de cozi
* Boolean open- pentru a vedea daca coada este deschisă sau nu

Scheduler

Mută clienții in cozi în funțtie de strategia aleasa

Atributele folosite:

* Servere
* Constrângeri: maxNoServers, Strategy

Scheduler – Strategy Pattern

* Se alege modalitatea de împărțitre a clienților in cozi

SimulationManager – Runnable:

* Generarea Random a clienților cu: timp de sosire, timp de procesare și id
* Bucla de simulare: modificare a timpului curent, apelare Scheduler pentru a aloca clienții in cozi
* - SelectionPolicy selectionPolicy- strategia de adaugare a clientilor in cozi

1. Implementare

Clasa Scheduler – are un vector de Servere și strategia aleasa. Are ca si metode principale changeStrategy și dispatchClient. Metoda de schimbare a strategiei stabilește în funcție atributul strategy, care va alege din cele 2 clase, care strategie urmează să fie implementata.: ori cea din clasa ConcreteSTrategyQueue, ori cea din clasa ConcreteSTrategyTime.

Clasa Server – fiecare obiect din clasa server o să fie un Thread, astfel această clasă extinde clasa Thread și implementează interfața Runnabale pentru a putea suprascrie meoda run(). Fiecare obiect de acest tip prezintă un indice pentru a putea fi identificată coada. Aceasta are ca metoda principala, metoda run, prin care se iterează prin server (prin coada), ia clientul din vârful cozii, îl lasă acolo un timp egal cu timpul lui de procesare și apoi îl scoate din coada.

Clasa SimulationManager – această clasă are ca field-uri toate datele necesare simulării care vor fi citite din interfața grafică care va sta la dispoziția utilizatorului. Cele doua field-uri importnate ar fi: selectionPolcy și o listă cu clienții generați random, sortați în ordine crescătoare în funcție de timpul la care au ajuns la coadă.Pentru a-i sorta am ales ca, clasa Client sa implementeze interfata Comparable, scriind metoda compareTo in funcție de timpul de sosire. Constructorul acestei clasei va porni serverele, adică cozile (threadurile corespunzătoare acestora). Deasemenea, si aceasta clasa va implementa clasa Runnable si va mosteni clasa Thread. Metodele esențiale ale acestei clase sunt : run() care se va ocupa cu distribuirea clientilor in cozi, dar și generateClients care va genera random cei n clienți supusi simulării

Interfața Strategy - metoda addClient care primește ca și parametrii o listă de Servere, adică o listă de cozi și un singur client care va introdus într-o coadă conform

Clasa ConcreSTrategyQueue – implementează interfața Strategy. Se folosește suprascrierea metodei din interfața Strategy și va fi implementată în mod corespunzptor.

Clasa ConcreSTrategyTime – – implementează interfața Strategy. Se folosește suprascrierea metodei din interfața Strategy și va fi implementată în mod corespunzptor. Împreună cu interfața Strategy și clasele ConcreteSTrategyTime și Scheduler formeaza un Strategy Pattern. (strategia este folosita pentru a optimiza timpul de așteptare a clientului la coadă)

Clasa SelectionPolicy – aceasta o sa fie o clasă de tip enumerație care prezintă cele două strategii posibile de adăugare a unui client într-o coada: SHORTEST\_QUEUE, SHORTEST\_TIME. Clientul va fi adăugat in coada dupa cele 2 strategii, ori la coada cea mai scurtă, ori la acea coada unde timpul de așteptare este cel mai scurt.

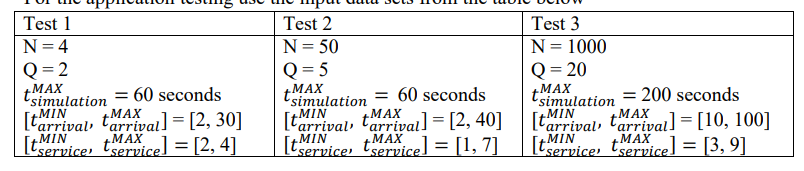
Clasa Client – descrie obiectele cu care se modelează această simulare,.Aceasta are ca și attribute ID-ul, timpul de sosire și timpul de procesare..

Clasa View – variabile instanță: frame-ul, un panel,1 JTextArea, 7 JtextField-uri, 7Jlabel-uri și 1 Jbutton. Cele 7 TextField-uri sunt folosite fie pentru introducerea datelor în interfața grafică de către utilizator, cât și pentru afișarea rezultatului așteptat în urma realzării simuării. Butonul este folosit pentru a confirma alegerea facută de catre utilizator, altfel spus, pentru a porni simularea

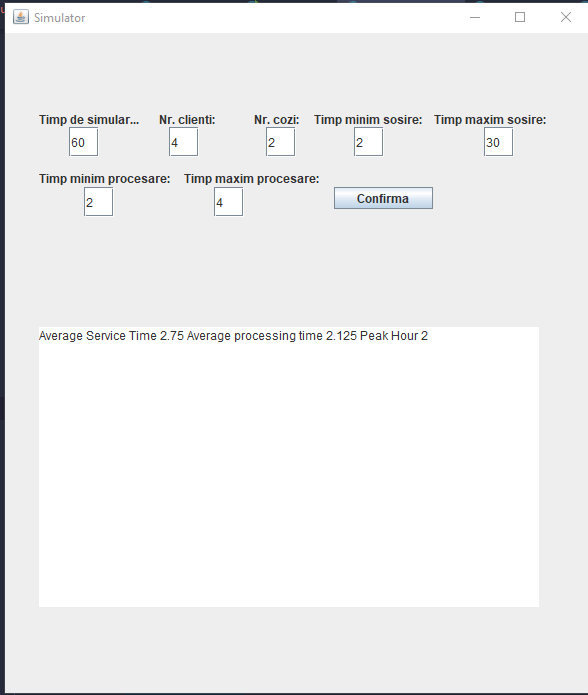
Clasa Controller – variabile instanță: un view și metoda care va instanția un obiect din clasa SimulationManager, se vor porni thread-urile care vor reprezenta cozile și thread-ul principal.

**Rezultate**

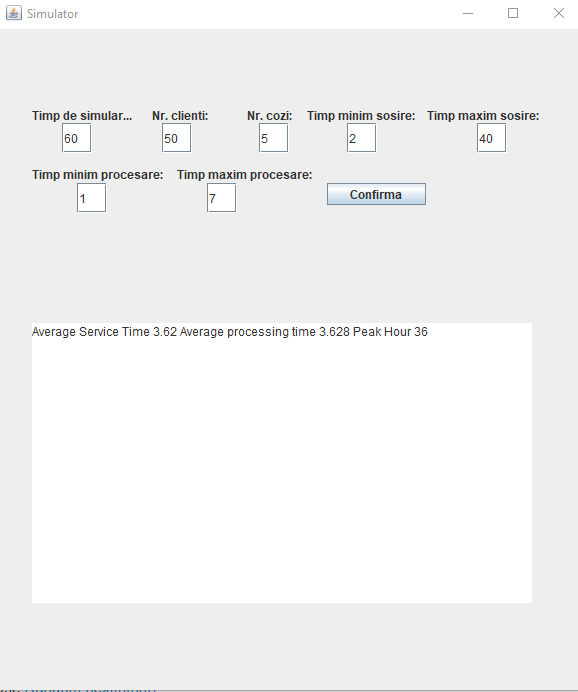
Testarea aplicației s-a făcut pe mai multe cazuri, pe mai multe seturi de parametri diferiti. Pentru a testa acuratețea, m-am asigurat că, clienții sunt puși in coada când momentul simulării corespunde cu timpul lor de sosire, și scoși din coada atunci cand timpul lor de procesare s-a finalizat. Analizând aceste aspecte, am dedus că simularea se face intr-un mod corect.



Test1



Test2



Test 3

# Concluzii

**Implementarea acestei aplicații a fost foarte utilă, reușind astfel să ințeleg si mai mult conceptele învățate până acum, ba mai mult descoperind noi concepte și noi modalităti de funcționare.Faptul că se poate observa simularea în timp real, secundă cu secundă ajută mult la înțelegerea algoritmului din spatele simulatorlui. Programarea cu thread-uri a fost un concept nou, foarte interesant de pus în practică și care m-a ajutat să ințeleg mult mai bine cum funcționeaza lucrurile și cum pot face ca bucăți diferite din codul meu să se execute în parallel, dar cu toate acestea să facă lucruri diferite.**