

Simulator de cozi

1. Obiectivul temei

Scopul acestei teme este de a proiecta si implementa o aplicatie care sa simuleze intr-un timp definit procesul de servire a clientilor de catre servere(case de marcat) determinand si minimizand timpul de asteptare al clientilor. Serverele sunt bazate pe cozi care au ca si rol asigurarea unui loc pentru fiecare client unde sa astepte inainte de a fi servit.

Aplicatia simuleaza N client pregatiti pentru servire, fiecare la un moment de timp aleator, plasati in cele Q cozi pentru a astepta, urmand ca cel care este primul in coada sa fie servit si scos din coada.

Toti clientii sunt generati cand incepe simularea($t_{simulation}$). Un client este caracterizat de:

- ID: numar cuprins intre 1 si N
- $t_{arrival}$: momentul de timp la care este pregatit sa mearga la coada (ex: timpul la care isi termina cumparaturile)
- $t_{service}$: durata servirii acestuia (ex: durata de timp pentru care se afla primul in coada)

Aplicatia urmareste totalul timpului petrecut de fiecare client in coada si calculeaza timpul mediu de asteptare (cat timp asteapta un client pana sa fie servit, pana sa fie el in capul cozii). Fiecare client este adaugat la coada cu timpul minim de asteptare cand $t_{arrival} \geq t_{simulation}$.

Intrările urmatoare de date pentru aplicatie trebuie inserate de catre utilizator din interfata:

- Numarul de clienti: N
- Numarul de cozi: Q
- Intervalul de simulare ($t_{max_simulation}$)
- Minimul si maximul pentru $arrival_time$
- Minimul si maximul pentru $t_{service}$
- Numar maxim de clienti/coada

Sub-obiectivele temei sunt:

- Analizarea problemei si identificarea cerintelor: se iau in considerare scenariile de utilizare si cerinte functionale (2.)
- Proiectarea simulatorului de clienti si cozi: decizii de proiectare (3.)
- Implementarea simulatorului: prezentarea claselor cu metode si campuri importante (4.)
- Rezultate: testarea datelor introduse prin intermediul unui jurnal de evenimente, fisier text, in care este transpusa simularea in timp real (5.)

2. Analiza problemei

Utilizatorul poate alege între 2 strategii: clienții pot fi plasati la cea mai scurtă coadă sau pot fi plasati la coadă cu cel mai mic timp de așteptare.

Caz de utilizare: Utilizatorul alege strategia minimum waiting time

Scenariu principal reușit:

- Utilizatorul introduce în interfața grafică toate datele cerute: număr clienți, număr cozi, timpul de simulare, minim și maxim pentru timpul de sosire al clienților, minim și maxim pentru timpul de procesare al clienților.
- Utilizatorul alege din caseta drop-down strategia „minimum waiting time”.
- Utilizatorul apasă butonul „Start” pentru începerea simulării.
- Simulatorul va deschide o a doua fereastră unde este vizibilă simularea în timp real.
- Aceste rezultate sunt reținute și într-un jurnal de evenimente, în fișierul text „displayLogOfEvents.txt”

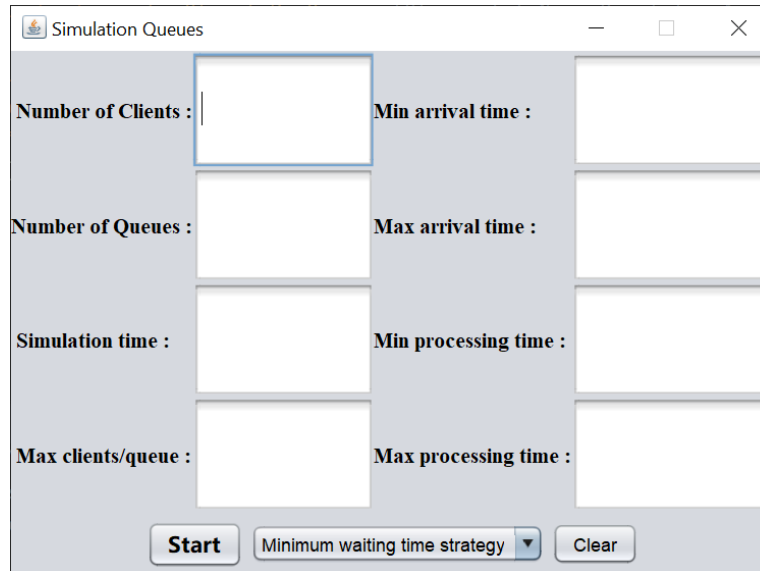
Scenariu alternativ:

- Utilizatorul omite completarea tuturor câmpurilor cerute.
- Procesul se reia de la primul pas.

Cerințe funcționale:

- Simulatorul trebuie să permită utilizatorului să introducă datele esențiale pentru executia simulării
- Simulatorul trebuie să permită utilizatorului să aleaga una din cele două strategii în funcție de necesități
- Simulatorul trebuie să facă vizibilă simularea în timp real cu toate detaliile: clienții neprocesati, fiecare coadă cu clienții săi în așteptare, momentul de timp și la final timpul mediu de așteptare, timpul mediu de servire și cel mai aglomerat moment de timp.

3.Proiectarea simulatorului de clienti si cozi



Pentru a simula in timp real procesarea clientilor de niste servere(cozi),este necesara introducerea datelor de mai sus de catre utilizator.

Problema plasarii clientilor la cozi in momente de timp diferite se rezolva cu ajutorul Thread-urilor.Am proiectat un thread pentru managerul care se ocupa de preluarea clientilor neprocesati la momentul de timp corespunzator si cate un thread pentru fiecare coada care se asigura de servirea clientilor sai.

A fost necesar campul „Max clients/queue”,deoarece proiectul are la baza lucrul cu Thread-uri,iar cozile le-am ales sa fie de tipul ArrayBlockingQueue pentru a fi sincronizate si acest tip necesita capacitatea unei cozi specificate.

Caseta drop-down permite utilizatorului sa aleaga una din cele doua strategii,in functie de necesitati:plasarea clientului in coada cea mai scurta sau in coada cu cel mai putin timp de asteptare.

3. Implementarea simulatorului:prezentarea claselor cu metode si campuri importante

Clasele componente proiectului sunt:

- a) Task: -aceasta clasa caracterizeaza clientul care are ca si attribute urmatoarele:

```
private int arrivalTime; //timpul cand clientul e gata sa mearga la coada
private int processingTime; //durata servirii clientului
private int ID;//numar ales random intre 1 si numarul total al clientilor
private int remainingProcessingTime;//timpul ramas de servire
```

-clasa nu are foarte multe metode,cele mai importante ar fi cea care ajuta la sortarea clientilor in functie de arrivalTime si decrementTimeReturnStatus() care decrementeaza remainigProcessingTime si returneaza true cand clientul este finalizat

- b) Server: -este un Thread care se ocupa de procesarea clientilor din coada

```
private BlockingQueue<Task> taskList; //lista de clienti din coada
private AtomicInteger waitingPeriod; //timpul de asteptare al cozii
private int maxClientsPerServer; //cati clienti pot astepta in acelasi timp la coada
private boolean running; //true daca thread-ul ruleaza, false daca este oprit
```

-metoda importanta din Server este run() care este executata automat ce thread-ul corespunzator serverului este pornit prin Thread.start(). Ea ruleaza cat timp atributul running este true si proceseaza fiecare client din coada, scazand waitingPeriod cu o unitate cu fiecare client finalizat si scos din coada. WaitingPeriod este crescut cu processingTime al fiecarui client adaugat de metoda addTask(Task t).

c) Scheduler- creeaza lista de cozi si porneste toate thread-urile corespunzatoare cozilor

```
private List<Server> serverList; //lista de cozi
private Strategy strategy; //strategia aleasa

public enum SelectionPolicy{
    SHORTEST_QUEUE, SHORTEST_TIME
}
```

-printre metodele planificatorului se numara changeStrategy(SelectionPolicy policy) care schimba strategia in cea aleasa de utilizator, iar metoda dispatchTask(Task t) trimite clientul la o coada in functie de strategie. De asemenea, clasa Scheduler opreste fortat toate thread-urile corespunzatoare serverelor prin metoda stopServers().

d) SimulationManager-proceseaza clientii si executa simularea pana cand timpul dat se termina sau toti clientii au fost procesati si cozile sunt goale. Tot aici se genereaza random clientii.

```
private int timeSimulation;
private int numberOfClients; //numar clienti
private int numberOfServers; //numar cozi
private int maxProcessingTime; //timp max servire
private int minProcessingTime; //timp min servire
private int maxArrivalTime; //timp max sosire
private int minArrivalTime; //timp min sosire
private Scheduler scheduler;
private List<Task> generatedTasks; //lista de clienti in care se genereaza random arrivalTime, id si processingTime
private Scheduler.SelectionPolicy selectionPolicy;
private DisplayLogOfEvents frame;
private double averageWaitingTime; //timp mediu de asteptare la cozi
private double averageServiceTime; //timp mediu de servire
private int peakHour; //moment de timp cel mai aglomerat
```

-metoda run() este cea pe care o executa Thread-ul managerului, ruland pana cand timpul de simulare se termina sau nu mai sunt clienti de servit si toate cozile sunt goale. Cat timp mai sunt clienti de procesat, se preia la fiecare moment de timp clientul cu t_arrival = t_simulation si se paseaza clientul metodei dispatchTask din Scheduler care il adauga in coada potrivita strategiei alese. Dupa ce timpul de simulare s-a terminat, thread-urile pentru cozi se opresc fortat de catre Scheduler.

Average waiting time: cat timp trebuie sa astepte un client din momentul plasarii la coada pentru a fi servit (pana sa ajunga primul in coada)

Peak hour: momentul de timp cu cei mai multi clienti in cozile existente

e) ShortestTimeStrategy-clasa care implementeaza interfata Strategy si metoda sa addTask.

Metoda addTask cauta coada(serverul)care are cel mai mic timp de asteptare.Se retine waitingPeriod al cozii alese si apoi se adauga clientul in coada respectiva.WaitingPeriod se retine inainte de a adauga clientul in coada,deoarece dupa adaugare el va creste cu timpul de procesare al clientului.

- f) ShortestQueueStrategy-si aceasta clasa implementeaza interfata Strategy si metoda sa addTask.

Metoda addTask cauta coada(serverul)care are cei mai putini clienti in asteptare. Se retine waitingPeriod al cozii alese si apoi se adauga clientul in coada respectiva.WaitingPeriod se retine inainte de a adauga clientul in coada,deoarece dupa adaugare el va creste cu timpul de procesare al clientului.De aceea,daca coada este goala si un client este adaugat,waitingPeriod al cozii in acel moment este 0,clientul fiind servit de indata ce a ajuns la coada.

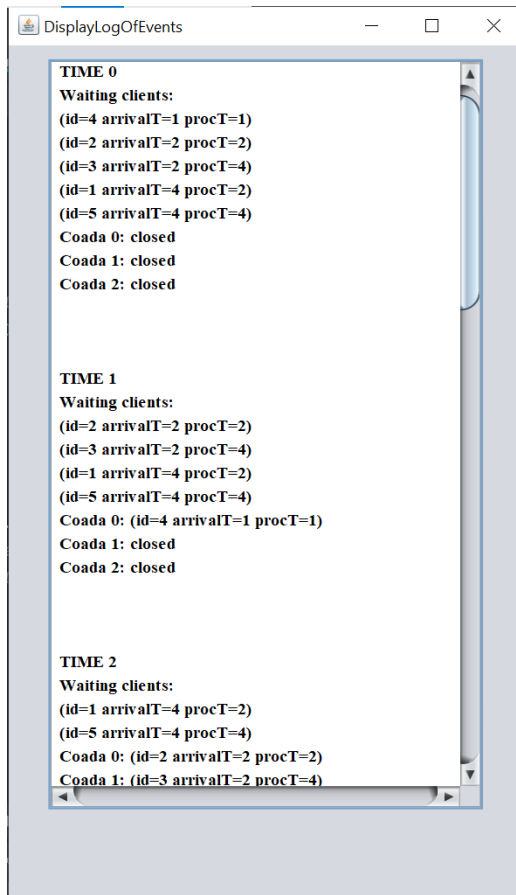
- g) StartSimulation-clasa reprezinta interfata grafica de mai jos.In momentul apasarii butonului „Start”,dupa introducerea datelor evident,se creaza managerul din clasa SimulationManager care preia strategia aleasa de utilizator si se porneste Thread-ul sau.Odata ce managerul este creat,el va crea la randul sau scheduler-ul din Clasa Scheduler care creaza cozile si Thread-urile lor,pornind-ule spre executie.Datorita faptului ca managerul are ca si camp frame-ul din clasa DisplayLogOfEvents,atunci cand acesta este creat(cand se apasa butonul „Start”) apare o noua fereastră creata in clasa DisplayLogOfEvents.

The screenshot shows a window titled "Simulation Queues" with a light blue background. It contains several input fields arranged in a grid, each with a label and a text box. At the bottom, there are three buttons: "Start", "Minimum waiting time strategy" (which is a dropdown menu), and "Clear".

Number of Clients :	5	Min arrival time :	1
Number of Queues :	3	Max arrival time :	5
Simulation time :	9	Min processing time :	1
Max clients/queue :	3	Max processing time :	4

Buttons: Start, Minimum waiting time strategy, Clear

- h) DisplayLogOfEvents-clasa care se ocupa de afisarea simulării in timp real.Are o singura metoda,setTextArea(String s) care adauga in continuarea zonei de text din interfata informatiile preluate din manager(construite prin intermediul unui StringBuilder care face append fiecarui sir care doreste a fi afisat).



- i) SimulationArea-construieste atributul `StringBuilder simulationTextArea` care va fi afisat in interfata.Cu ajutorul metodei `addText(String s)` se adauga la `StringBuilder` siruri de caractere care formeaza ,bucata' de text cu continutul timpului curent,clientii neprocesati si cozile cu clientii lor.

5.Rezultate obtinute

1.Pentru primul test s-au introdus datele:

Numar clienti=4

Numar cozi=2

Timp simulare=60 secunde

`[T_min_arrival,T_max_arrival]=[2,30]`

`[T_min_processing,T_max_processing]=[2,4]`

S-au generat random urmatorii clienti:

TIME 0

Waiting clients:

(id=2 arrivalT=3 procT=4)

(id=4 arrivalT=8 procT=4)

(id=1 arrivalT=9 procT=4)

(id=3 arrivalT=24 procT=2)

Coadă 0: closed

Coadă 1: closed

Am obtinut:

Average waiting time=0.0

Average service time=3.5

Peak hour=9

2. Pentru al doilea test s-au introdus datele:

Numar clienti=50

Numar cozi=5

Timp simulare=60 secunde

[T_min_arrival,T_max_arrival]=[2,40]

[T_min_processing,T_max_processing]=[1,7]

Am obtinut:

Average waiting time=1.04

Average service time=3.7

Peak hour=30

3. Pentru ultimul test s-au introdus datele:

Numar clienti=1000

Numar cozi=20

Timp simulare=200 secunde

[T_min_arrival,T_max_arrival]=[10,100]

[T_min_processing,T_max_processing]=[3,9]

Am obtinut:

Average waiting time=80.453

Average service time=6.033

Peak hour=100

6.Concluzii

Din aceasta tema am inteles cat este de importanta impartirea sarcinilor diferitelor clase si mai mult decat atat, comunicarea dintre ele care usureaza efortul de a urmari codul scris. Desi a fost o noutate pentru mine notiunea de Thread, mi-am dat seama in cele din urma ca e mai eficient sa se efectueze mai multe sarcini in acelasi timp pentru a putea gestiona cantitati mai mari task-uri.

Ca si dezvoltari ulterioare, simulatorului proiectat i s-ar putea modifica metoda de scriere in interfata grafica a simularii in timp real, anume sa afiseze doar momentul de timp curent, nu sa se afiseze toata simularea de la timpul 0 pana la cel setat, deoarece oricum toata simularea se gaseste deja in fisierul text creat.

7.Bibliografie

StackOverFlow

Support Presentation