

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

Tehnici de programare

Documentație

**QUEUES SIMULATOR**

Bartalus Izabella

Grupa 30221

**Cuprins**

1. Obiectivul temei

2**.** Analiza problemei, modelare, scenarii, cazuri de utilizare

3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete , relatii, packages, algoritmi, interfata utilizator)

4. Implementare

5. Rezultate

6. Concluzii

7. Bibliografie

1. Obiectivul temei

Obiectivul principal al acestui proiect este de a descrie o simulare in timp real al unui sistem care lucreaza cu cozi pentru a determina si a minimiza timpul de asteptare al clientilor . Datele pe care se bazeaza simularea sunt citite dintr-un fisier, iar la sfarsit trebuie specificate , intr-un alt fisier , ordinea in care au intrat in cozi clientii , iar la fiecare moment de timp este specificat timpul ramas pana acesta va parasi coada . De asemenea , la sfarsit , va fi calculata si afisata in fisier timpul mediu de asteptare al clientilor in coada .

Ca obiective secundare am putea mentiona :

* Descrierea unei clase “ Client “ ;
* Descrierea unei clase “ Queue ” , care contine un sir de client ;
* Descrierea unei clase “ SimulationManager “ , unde se citesc din fisier toate datele necesare implementarii aplicatiei ;
* Descrierea unei clase “ Server “ , unde sunt prelucrate cozile ;
* Crearea multi-threadurilor ( una pentru fiecare coada ) ;
* Simularea ( cat de real posibil ) a timpului de sosire si de asteptare a clientilor ;
* Afisarea rezultatului simularii cozilor intr-un fisier ;
* Calculul timpului mediu de asteptare al clientilor.

1. Analiza problemei, modelare, scenarii, cazuri de utilizare

* Analiza

Scopul principal al problemei este acela de a dezvolta un program original care realizeaza o simulare in timp real a unui sistem bazat pe cozi , utilizand paradigmele Programarii Orientate pe Obiect . In primul rand , trebuie sa gasim un mod de a reprezenta toate datele semnificative si , de asemenea , un mod de a afisa rezultat intr-un mod simplu , atat pentru programator , cat si pentru utilizator .Pe masura ce incepem sa implementam simularea sistemului bazat pe cozi , vom realiza ca va deveni mai clar ca ne confruntam cu o situatie unde vom avea nevoie sa realizam anumite actiuni care se intampla in parallel .

Asadar , ne confruntam cu concurenta . Concurenta e abilitatea de a rula anumite programe sau anumite parti dintr – un program in paralel . Daca un task care se executa intr – o anumita perioada de timp , poate fi realizat asincron , sau in paralel , vom putea imbunatati cu acesta randamentul si interactivitatea programului .

* Modelarea

Prima problema cu care ne care ne confruntam consta in stocarea informatit . Pentru aceasta , incepem cu cea mai mica parte a programului ,si anume , “ Clientul ” . Dupa ce am rezolvat aceasta problema , am creat clasa “ Queue ” , care are , printre atributele sale , o lista de clienti . Aici simulam , de fapt “ sosirea “ ,”asteptarea ” , “ eliminarea “ clientilor .

Ca o a doua problema , trebuie sa ne confruntam cu nevoia de concurenta a simularii programului . Pentru asta am utilizat threaduri ( fiecare coada are un thread ) .

Si , in final , a treia problema ar fi gestionarea tuturor cozilor ,si , pentru asta , am implementat clasa “ Server ” ( care este si ea un thread ) . Clasa “ Server “ apeleaza clasa “ SimulationManager “ ,unde sunt prelucrate din fisier si introduse in program informatiile necesare implementarii programului . In aceasta clasa , pe langa preluarea informaitiilor din fisier , vom genera clientii si vom initializa cozile .

* Scenarii

Daca programul ruleaza corect , utilizatorul ar trebui sa vada , intr – un fisier modul in care intra clientii in cozi , timpul petrecut in coada ,dar si clientii care asteapta . Iar , la finalul fisierului , va aparea timpul mediu de asteptare al clientilor .

* Cazuri de utilizare

La deschiderea fisierului unde sunt sunt trecute rezultatele simularii aplicatiei , va aparea , la fiecare moment de timp ( de la 0 pana la timpul maxim de simulare , introdus in fisierul de intrare ) lista clientilor ( al caror timp de sosire nu corespunde cu timpul curent ) iar ,apoi , apar toate cozile , urmate de clientii care au intrat in coada . In cazul in care o coada este goala , langa aceasta va aparea mesajul “ CLOSED ” .

1. Proiectarea ( decizii de proiectare , diagrama UML , structuri de date , proiectare clase , interfete , relatii , packages , algoritmi , interfata utilizator )

Pentru a respecta paradigmele POO , programul a fost impartit in mai multe pachete cu nume sugestive care contin diferite clase :

* Pachetul controller contine clasa :
* App – de aici se porneste simularea;
* Pachetul model contine clasele
* Client – contine:
* 3 atribute ( id , arrivingTime , servingTime ) ; Cele 3 atribute au fiecare setters si getters ;
* Doi constructor ;
* Queue – creeaza un thread , astfel , implementeaza interfata Runnable ; Contine :
* O lista “ synchronizedList “ de clienti ( am ales aceasta lista , deoarece este sincrona si avem nevoie de o lista sincrona pentru a putea lucra cu threaduri ) ;
* Un “ waitingTime “ ( care este AtomicInteger , din acelasi motiv ca si mai sus – ca un contor atomic ( incrementAndGet ( ) , etc . ) care poate fi folosit de mai multe threaduri concurent ;
* Un atribut – “ closingTime “ – timpul fata de care ne raportam cand parcurgem coada ;
* Un atribut “ timer “ – care tine evidenta timpului in coada ;
* Un atribut numberOfClients , care stocheaza numarul de client al cozii ;
* Avem , de asemenea , un constructor care are ca parametru “ closingTime “ , si , care seteaza de asemenea , “waiting Time “ - ul pe 0 ;
* Ca metode avem :
* “ removeClient ( ) “ – pentru stergerea unui client din coada ;
* “ addClient ( ) ” – pentru adaugarea unui client in coada ;
* “ returnQueue ( ) “ – care ne returneaza coada ;
* “ getSize ( ) “ – care ne returneaza dimensiunea cozii ;
* “ run ( )” – care porneste threadul din coada ;
* “ startQueue ( ) “ – care creeaza un thread nou si il porneste ;
* Server – care contine atributele:
* “ currentTime “ : care retine timpul curent ( de la 0 la timpul maxim de simulare a aplicatiei ) ;
* “ thread “ ;
* “ avgWaitingTime “ : pentru a calcula timpul mediu de asteptare al clientilor ;
* “ sem “ ;
* Ca metode avem:
* “ addTask ( ) “ ( metoda folosita pentru a adauga un client intr –o coada – se alege coada minima prin apelarea metodei “ getMinQueue “ )
* O metoda “ printResult ( ) “ : unde se va scris in fisier simularea aplicatiei
* O metoda “ run ( )
* SimulationManager – care contine :
* atribute , pe care urmeaza sa le citim din fisier :
* nrClienti : numarul de clienti pe care ii vom repartiza in cozi ;
* nrQueues : numarul de cozi disponibile ;
* simulationInterval : timpul de rulare al simularii ;
* min/maxArrivalTime : timpul minim , respectiv maxim intre care vor fi generate valorile de sosire ale clientilor ;
* min/maxWaitingTime : timpul minimi , respectiv maxim intre care vor fi generate valorile de asteptare ale clientilor ;
* lista de clienti ,care sunt generati aleator ;
* o lista de cozi , sincrona , pentru siguranta threadurilor ;
* Ca metode avem:
* un constructor : care citeste valorile din fisier
* “ getMaxWaitingTime “ – care da valoarea maxima a timpului de asteptare pe care il pot avea clientii
* “ getSimulationInterval “ – care da timpul maxim de similar
* “ getNrOfClients “ – care returneaza numarul total de clienti
* “ getWaitingClients “ – care returneaza lista cu clientii care asteapta sa intre in cozi
* “ getQueues “ – care returneaza cozile
* “ getMinQueue “ – care returneaza coada minima din lista de cozi
* SortByArrivingTime ,
* SortByWaitingTime ;
* Pachetul view contine clasele View si ViewSimulator ;

Graphical user interface, text, application, chat or text message

Description automatically generated

Diagrama UML a proiectului :

Diagram, engineering drawing

Description automatically generated

4. Implementare

Clasele existente sunt :

* Clasa “ Client “ - aceasta clasa are doar 3 atribute , un constructor , 3 setters si 3 getters pentru fiercare atribut ( pentru a putea accesa si schimba atributele clientilor )
* Clasa “ Queue “ - metodele principale ale clasei “ Queue “ :
* addClient ( Client c ) : - in cazul in care clientul nu e primul client din lista , trebuie sa ii scadem timpul de asteptare cu 1

- crestem numarul de clienti ale cozii respective

- adaugam timpul de asteptare al clientului la timpul de asteptare global

* removeClient ( ) : - daca coada vrea sa stergem un client , insa coada este goala , trebuie sa asteptam ; daca nu este goala , pur si simplu stergem clientul din varful cozii si decrementam numarul de clienti .
* run ( ) : cat timp inca este timp sau inca avem client in lista de asteptare luam primul client din coada si , intr – un “ for “ ii decrementam timpul de asteptare , si , de asemenea , decrementam timpul de asteptare al cozii si asteptam o secunda . Cand timpul de asteptare s-a scurs , stergem clientul .
* Clasa “ SimulationManager “ – metodele principale ale acestei clase :
* Constructorul “ SimulationManager ( ) “ , in care , deschidem un fisier , din care vom citi valorile cu care urmeaza sa lucram pe tot parcursul implementarii programului .
* Metoda “ generateClients ( … ) “ , unde , se genereaza aleator un timp de sosire – cuprins in acel interval pe care l-am citit din fisier , un timp de astepare – cuprins in acel interval pe care l-am citit din fisier . Prin intermediul acestor valori , o sa creem un client , caruia ii atribuim aceste atribute , impreuna cu id – ul unic .
* Metoda “ getMinQueue ( ) “ : care returneaza pozitia cozii minime din lista de cozi . In cazul in care gaseste o coada goala , returneaza pozitia acesteia , in caz contrar , afla care e coada minima , prin aflarea cozii cu dimensiune minima . In caz ca sunt mai multe cozi de dimensiune minima , vom returna coada cu timpul de asteptare mai mic .
* Metoda “ InitQueues ( . . . ) “ : parcurgem un ciclu de la 0 la numarul de cozi pe care l-am citit din fisier si adaugam cate o coada in lista de cozi , iar apoi , pornim threadul acestei cozi , prin metoda startQueue ( ) .
* Clasa “ Server “ – metodele principale ale aceste clase sunt :
* Metoda “ printResult ( . . . ) “ : care afiseaza rezultatele simulatii sistemului pe baza de cozi intr-un fisier ;
* Ne creem un Obiect SimulationManager ,pentru a putea apela valorile pe care le-am citit din fisier . Intr-un “while“ :

- adaugam la coada clientii ce au timpul de sosire egal cu timpul current ;

* Dupa ce am facut asta, thread-ul trebuie sa “doarma “ pentru o secunda ;
* Incrementam timpul current ;
* Calculam timpul mediu de asteptare al clientilor .
* Clasa “ App “ : aici este instantiate un obiect de tip “ Server “ , si creem un nou thread cu acest obiect , pentru a-l putea porni , urmand apoi sa porneasca simularea aplicatiei .

5. Rezultate

Pentru a demonstra functionalitatea programului, am creat 3 fisiere text pe care le – am rulat pe rand si asa am obtinut 3 fisiere de iesire unde avem rezultatul fiecarui fisier de intrare (in\_1.txt -> out\_1.txt ; in\_2.txt –> out\_2.txt ; in\_3.txt -> out\_3.txt).

Text

Description automatically generated

De asemenea, pentru a fi mai usor de folosit programul, utilizatorul poate introduce datele de intrare din interfata unde se vor afisa si rezultatele.

Graphical user interface, application

Description automatically generated

6. Concluzii

Prin rezolvarea acestei cerinte m-am familiarizat cu thread-urile iar pe viitor imi va fi mult mai usor sa lucrez cu ele.

Imbunatatiri care pot fi aduse programului ar fi o mai buna sincronizare.

7. Bibliografie

1. Wikipedia

2. StackOverflow