Tema 2: Simulator cozi de asteptare

Student: Campean Bogdan Mihai

Grupa: 30221

**1. Obiectivul temei**

Obiectivul principal al acestei teme este de a simula, printr-o aplicatie pe computer, in limbajul de programare Java si prin paradigmele programarii orientate pe obiect, o problema des intalnita in viata de zi cu zi, si anume distribuirea a n task-uri in m servere de procesare, unde m si n sunt variabile care depind de timp. Aceasta problema se aplica nu numai in domeniul it, dar acest domeniu ajuta la reducerea semnificativa a timpului de distribuire a acestor task-uri; se aplica si, de exemplu, la statul la rand la magazine pentru a plati cumparaturile, unde fiecare om reprezinta un task cu un anumit timp de procesare (cate cumparaturi se scaneaza), iar un server reprezinta numarul caselor.

Obiectivul secundar urmareste dezvoltarea corecta a aplicatiei folosind thread-uri (fire de executie) care apparent ruleaza in parallel, astfel fiind posibila calculul distribuirilor la fiecare

moment de timp t care poate tinde la 0.

2. Analiza problemei, asumptii, modelare, scenarii, cazuri de utilizare, erori

Incepem prin analizarea problemei, evidentiind cuvintele cheie ale aplicatiei si impartirea problemei in mai multe subprobleme specifice paradigmei programarii orientate pe obiect pentru usurarea rezolvarii acesteia.

In primul rand, observam ca exista un numar de servere (n) si un numar de task-uri (m). Pentru calculul starii fiecarui server la fiecare moment de timp t, putem considera un server ca fiin un thread, care executa aceleasi operatii asupra task-urilor la fiecare moment de timp t (noi vom considera t ca fiind 1 secunda). Apoi observam ca pentru procesarea fiecarui calcul din server (thread), avem nevoie de inca un thread principal care sa execute thread-urile de procesare a serverelor.

In al doilea rand, observam ca fiecare task pe care il primeste un server se poate define ca si o insiruire de task-uri, in care primul venit e primul servit, mai exact printr-o structura de tip coada. La fel se observa si in cazul serverelor.

In concluzie, problema impartita in subprobleme este urmatoarea: avem mai multe thread-uri care proceseaza un task dintr-un server si avem un thread principal care le executa pe toate si proceseaza informatii suplimentare. De asemenea, avem si 2 structuri de tip coada care ne ajuta la usurarea procesarii, pe rand, a fiecarui thread dintr-ul server.

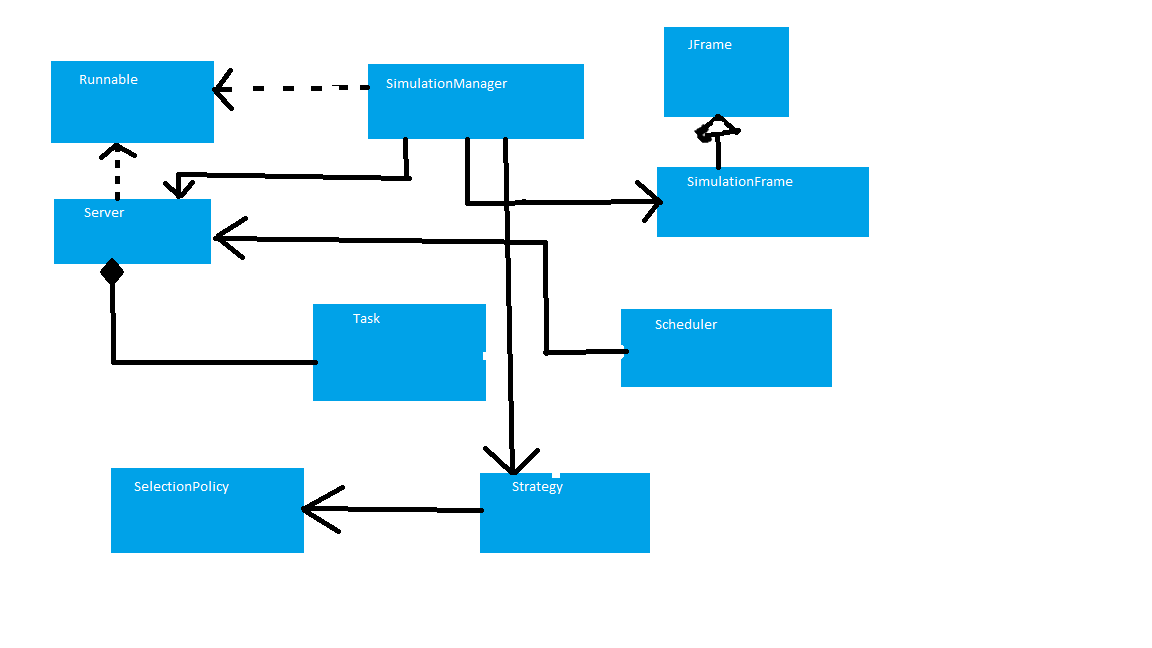
In acest capitol vom evidentia si asumptiile referitoare la aplicatia dezvoltata, si anume: vom presupune ca programul ruleaza o singura data de la pornire, fiind necesara pornirea ulterioara a sa pentru simularea altui caz descries mai sus. De asemenea, consideram o serializare a pasilor de urmat pentru folosirea acestui prototip, si anume: intai introducem datele cerute in mod corect (doar numere, fara spatii sau caractere), apoi putem porni simularea aplicatiei. Dupa acesti 2 pasi putem folosi celelalte comenzi in orice ordine vrea cumparatorul aplicatiei, si anume: introducerea unui nou server, introducerea random a unui task, stergerea unui server, schimbarea strategiei si oprirea simularii.

De asemenea, aplicatia vine cu un modul de a calcula in timp real a orei de varf si a timpului mediu de asteptare la acest “magazin”.

Detaliile de implementare vor fi descrise in detaliu in capitolele urmatoare.

3. Proiectarea

In acest capitol vom intra in detaliile de impartire in clase a aplicatiei si vom incepe prin realizarea diagramei UML pentru o usoara intelgere a acestora si ce problema rezolva fiecare in parte.



Dupa caum vedem in diagrama de clase UML, reprezentam Serverul ca fiind un thread separat de thread-ul principal, cel din calasa SimulationManager, de unde controlam toata aplicatia. Task-ul va fi o clasa in relatie de compozitie cu clasa Server, deoarece un Server poate exista si daca nu are niciun task activ il el care trebuie procesat. In clasa Scheduler impartim toate operatiile ce trebuie effectuate asupra unui server si a task-urilor sale, cum ar fi calculul mediei de asteptare sau calclulul orei de varf din fiecare simulare. Clasa Strategy impreuna cu clasa SelectionPolicy determina strategia pe care utilizatorul o alege pe parcursul simularii. Poate alege una dintre strategiile SHORTEST QUEUE sau SHORTEST TIME, care semnifica urmatoarele: daca este active opriunea SHORTEST QUEUE, fiecare task nou introdus va fi repartizat unde sunt mai putini oameni la coada, indifferent de timpul de asteptare de pe acel server; daca este active optiunea SHORTEST TIME, atunci fiecare task nou introus va fi repartizat la serverul cu cel mai scurt timp total de asteptare, indifferent de numarul task-urilor de pe celelalte servere. De asemenea, pentru interfata fomosim clasa SimulationFrame care introduce cele 7 butoane pentru selectia comenzilor cerute de utilizator, un ecran cu coada de asteptare in care se gasesc, puse pe rand, toate task-urile active si timpul in care fiecare task isi va termina executia, si un panou unde vor fi reprezentate calculele effectuate la fiecare secunda (timpul mediu de asteptare, ora de varf si timpul care s-a scurs din momentul inceperii simularii).

4. Implementare

In acest capitol vor fi prezentate detaliile fiecarei clase si anume: metode, variabile de clasa si utilizarea fiecarei metode importante din fiecare clasa.

Clasa Task are ca variabile de clasa urmatoarele: arrivalTime, processingTime, finishTime, number si taskNumber. Variabila arrivalTime este utilizata pentru timpul de sosire al fiecarui task pe server. Acesta va fi preluat din timpul simulat de aplicatie care incepe de la 0 cand incepe simularea. Variabila processingTime este folosita pentru descrierea timpului de procesare al fiecarui task, care poate varia intre un minim si un maxim alese de utilizator, fiind luat ca un numar aleator in acest interval. Variabila finishTime este suma variabilelor descrise recent; aceasta va fi preluata de clasa SimulationFrame pentru asfisarea timpului de terminare al fiecarui task. Variabilele number si taskNumber se folosesc pentru a atribui un numar unic fiecarui task introdus, deci variabila number va fi numarul total de task-uri existente la un moment dat in aplicatie.

De asemenea, clasa Task are urmatoarele metode: seteri, geteri pentru fiecare variabila de clasa, metoda setTaskFinishTime care calculeaza timpul final al fiecarui task. Metoda getAverageTask este folosita ulterior de clasa Server pentru calcularea tmipului mediu de asteptare; face diferenta intre timpul de procesare si al catelea task este pe acel server.

Clasa Server are urmatoarele variabile de calasa: tasks, totalWaitingPeriodOnServer, queueIndex, serverNumber, number, average. Variabila tasks este o variabila de timp lista inlantuita care se actualizeaza cu cate un task cand acesta este introdus. Variabila totalWaitingPeriodOnServer este o variabila de tip AtomicInteger care este folosita de fiecare thread din server pentru calcularea in timp real a timpului total de asteptare pe server. Variabila queueIndex de tip in teste utilizata la retinerea numarului de task-uri existente la un anumt moment in server; aceasta va ajuta la calcularea timpului mediu de asteptare. Variabilele number si serverNumber de tip int se folosesc pentru atribuirea unui numar unic la fiecare server.

De asemenea, clasa Server are urmatoarele metode: seteri, geteri pentru fiecare variabila de clasa si metodele: decrementServerQueue este utilizata in SimulationManager pentru decrementarea numarului de task-uri existente pe fiecare server in momentul in care un task si-a terminat executia. Metoda getAllWaitingPeriod returneaza timpul total de asteptare pe un server. Metoda getPeakHour returneaza un numar al task-urilor care va fi utilizat ulterior in SimulationManager pentru calculul orei de varf. Metoda addTask este utilizata pentru introducerea in coada a fiecarui task nou si pentru calcularea timpilor care definesc serverul. Metoda run() defineste urmatorul algoritm: intr-ul while va lua fiecare task din coada, pe rand, iar in alt while ii va procesa timpul de procesare al fiecarui task prin scaderea acestuia cu 1 si a timpului total de asteptare pe server tot cu 1 si obligand thread-ul sa astepte 1 secunda pentru timpul real al unui om; cand timpul a ajuns la 0, task-ul va fi eliminat si se va trece la urmatorul. Aceasta bucla se va efectua pana cand exista task-uri in server. Aceste thread-uri se vor executa apparent in parallel pentru fiecare server existent. Metoda getAverageWaitingTimePerServer calculeaza timpul mediu de asteptare pe fiecare sever, care este utilizat ulterior in SimulationManager pentru calcularea timpului final mediu de asteptare pe simulator.

Clasa Strategy nu are variabile de clasa, deoarece este folosita pentru calcularea indexului serverului unde se va introduce urmatorul task. Clasa are o metoda care returneaza indexul acelui server prin urmatorul algoritm: daca este aleasa optiunea SHORTEST QUEUE, atunci metoda va lua fiecare server la rand si va calcula minimul dintre numarul task-urilor pe fiecare server, dupa care il va returna; daca este aleasa optiunea SHORTEST TIME, atunci metoda va trece prin toate serverele si va calcula minimul dintre timpii de asteptare pe fiecare server, iar apoi va returna pozitia acelui server.

Clasa SimulationFrame are urmatoarele variabile de clasa: fereastra, label-uri, textField-uri si String-uri. Acestea se utilizeaza pentru afisarea rezultatelor in timp real, in timp ce ruleaza aplicatia. Ideea pe care se bazeaza afisarea timpului real este urmatoarea: realizam o matrice de JLabel-uri, in care vom afisa fiecare server cu task-urile sale existente, de sus in jos. Pastram rezultatele returnate de clasele Server, Task, Strategy intr-o matrice de String-uri, fiecare String fiind urmatorul rezutat al fiecarui JLabel, astfel facandu-se operatiile pe string-uri, facand afisarea si lucrul mai usor. Metodele scrise, combinate, fac urmatoarele lucruri: cand primul task se termina (timpul de procesare al lui ajunge la 0) acesta este scos din server, dar si din JLabel, facand o mutare la stanga cu o pozitie a fiecarui element din matricea de string-uri si completarea ultimului element cu “-“, character definitoriu pentru reprezentarea spatiului gol dintr-o coada. Aceste operatii se fac o data la fiecare secunda, scazandu-se la fiecare secunda numarul pe care il tine un String. Aceasta operatie se face prin intermediul unei variabile de tip Integer (se transforma din String in Integer, se face scaderea pe date de tim Integer, apoi transformandu-se inapoi in String) si afisarea lor pe JLabel. Metoda createSimulationFrame creeaza fereastra si pozitioneaza pe aceasta fiecare mesaj, buton si textfield. De asemenea, pe langa aceste metode, exista si seteri si geteri, care returneaza butoane, textfield-uri si String-uri pentru transmiterea valorilor de afisare din SimulationManager in SimulationFrame.

Clasa SimulationManager are urmatoarele variabile de clasa: timeLimit, variabila de tip int care retine timpul limita (in secunde) pe care il introduce utilizatorul si care defineste timpul cat va rula aplicatia, in caz ca nu este intrerupta de utilizator la un moment dat. Variabila maxProcessingTime, minProcessingTime defines un interval in care va fi luata o valoare aleatoare pentru fiecare task adaugat (se foloseste aceasta metoda pentru simplificarea utilizarii aplicatiei). Variabila numberOfServers defineste numarul de servere existent la inceputul aplicatiei; acest numar va fi ales de catre utilizatorul aplicatiei. Variabila timpScurs este utilizat pentru afisarea timpului real care este in secunde, dar ulterior va fi transformat in format orar, de la inceperea simularii; la fiecare pas va fi comparat cu timeLimit. Variabila averageWaitingTime si peakHour sunt variabile ce definesc raspunsul, la fiecare secunda, a timpului mediu de asteptare, respectiv al orei de varf. DE asemenea, sunt definite si variabile clasa pentru utilizarea si legatura logica dintre clase.

Metodele clasei SimulationManager sunt: generateRandomNumber, care va returna un obiect de tip Task cu toate atributele instantiate (timpul de sosire luat din timpul scurs al aplicatiei, iar timpul de procesare va fi un numar aleatoriu din interval pentru simplificarea utilizarii aplicatiei). Acest task va fi introdus in serverul corect, luat dupa strategia aleasa. Se va define, de asemenea, si un constructor pentru aceasta clasa. Metoda principala, run(), thread-ul principal al aplicatiei implementeaza urmatorul algoritm: intai va da spre executie paralela fiecare thread, apoi, intr-un while, va compara timpul scurs cu timpul limita ales; va astepta o secunda dupa care, va apela metodele ce s-au definit in celelalte clase si refresh-ul ferestrei la fiecare secunda pentru afisarea rezultatelor in timp real. De asemena, se va calcula ora de varf si timpul mediu de asteptare la fiecare secunda. In metoda principala main se definesc si se intstantiaza fiecare clasa, apoi se fac legaturile si calculul intre acestea la fiecare apasare de buton, depinzand in ce ordine si care buton va fi apasat.

5. Testare

6. Rezultate

7. Concluzii

In acest capitol vom trage concluziile referitoare la dezvoltarea aplicatiei.

In primul rand, se poate afirma ca s-a invatat folosirea thread-urilor, gestionarea, crearea si multithreadingul in limbajul de programare Java, dar si gestionarea gandirii in parallelism de calcul utilizand fire de executie.

In al doilea rand, aplicatia se poate dezvolta ulterior prin adaugarea unei interfete grafice mai apropiate de un utilizator obisnuit, cu forme geometrice reprezentand fiecare task (sau figuri umane), in locul timpului afisat la fiecare coada. De asemenea, se pot introduce mai multe calculi statistice facute in timpul unei simulari, in afata de acestea 2 implementate deja.