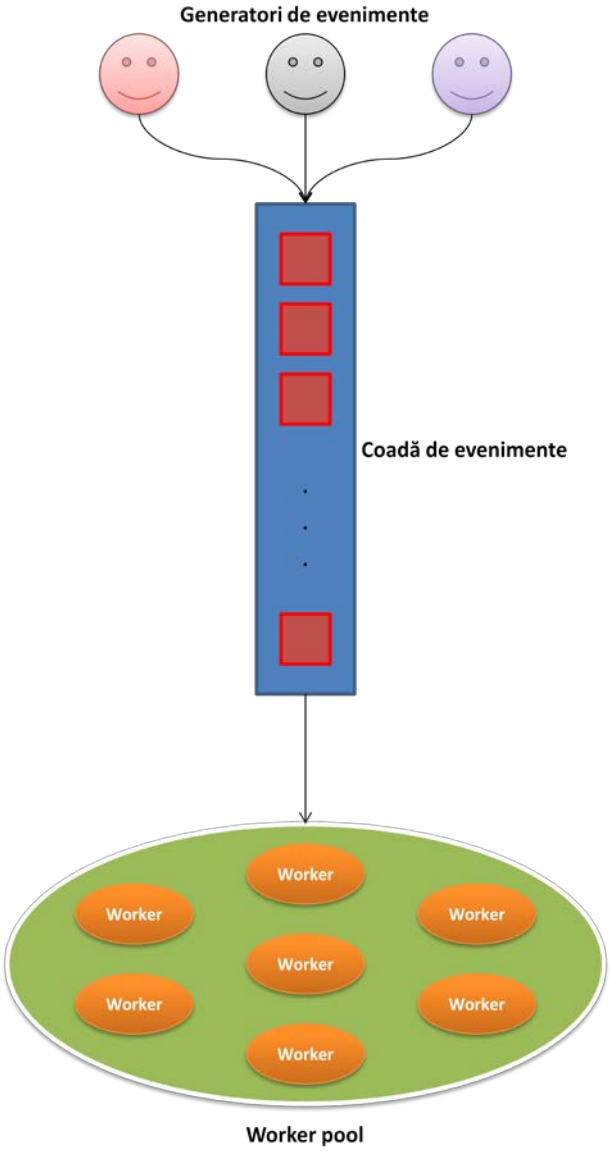
Tema 2 – Prelucrare paralelă de evenimente

Termen de predare: 4 decembrie 2016, ora 23:55
Responsabili temă: Radu-Ioan Ciobanu, Cristian Chilipirea, Valeriu Stanciu, Dragoş Comăneci

Scopul acestei teme este implementarea unui sistem de prelucrare paralelă a unor evenimente provenind de la mai multe surse, folosind Java Concurrent. Ca un exemplu, considerați că mai mulți producători pot produce date adnotate. Datele sunt necesare unor consumatori, însă fiecare dintre acești consumatori este interesat doar de anumite tipuri de date (considerând acele adnotări).

Ideea de funcționare a sistemului este prezentată în figura de mai jos. Există mai mulți generatori de evenimente de diferite tipuri (un eveniment necesitând calculul unei valori). La generarea unui eveniment, acesta este adăugat într-o coadă de evenimente. Evenimentele din coadă sunt apoi procesate de niște workeri aflați într-un pool (conform paradigmei "replicated workers" prezentată în laboratorul 7), rezultatele fiind puse într-o structură de date specifică tipului de eveniment. Cât timp sunt evenimente în coadă și workeri disponibili în pool, se dă un eveniment unui worker spre procesare. Când un worker termină de procesat un eveniment, revine în pool în așteptarea altui eveniment. La finalizarea procesării tuturor evenimentelor din sistem, structurile specifice lor sunt sortate în ordine crescătoare.



Implementare

Exista 4 tipuri de evenimente în sistem:

- PRIME - se calculează cel mai mare număr prim mai mic sau egal cu un N dat
- FACT - se calculează cel mai mare număr care are factorialul mai mic sau egal cu un N dat
- SQUARE - se calculează cel mai mare număr care are pătratul perfect mai mic sau egal cu un N dat
- FIB - se calculează cel mai mare număr pentru care valoarea corespunzătoare din şirul Fibonacci este mai mică sau egală cu un N dat (unde F_0 este 0 si F_1 este 1)

Datele de intrare
Programul va primi următorii parametri în linia de comandă:

```
dim_coadă evenimente_fișier nume_fișier1 nume_fișier2 nume_fișier3 […]
```

După cum se observă, primul parametru este dimensiunea cozii (exprimată în număr maxim de evenimente care pot fi stocate simultan în coadă), apoi numărul de evenimente din fiecare fişier (toate fişierele au un număr egal de evenimente), iar în final numele unor fișiere care conțin evenimentele ce vor fi generate. Fiecare fișier corespunde câte unui generator de evenimente, și arată în felul următor:

```
interval_timp1,eveniment1,N1
interval_timp2,eveniment2,N2
[…]
```

Așadar, fiecare fișier va conține un număr de linii dat de parametrul `evenimente_fișier`, pe fiecare linie fiind o durată de timp (exprimată în milisecunde), un tip de eveniment, și un număr N > 1 pentru evenimentul respectiv. Exemplu de fișier:

```
100,PRIME,50
50,SQUARE,120
250,FIB,170
50,PRIME,20
200,FACT,250
[...]
```

Generatorii de evenimente
Entitățile care generează evenimente vor fi implementate ca niște threaduri Java, câte unul pentru fiecare fișier de intrare primit în linia de comandă. Un generator de evenimente face următorii pași, pentru fiecare linie din fișier:

1. citește linia
2. așteaptă durata (în milisecunde) specificată în primul parametru de pe linie
3. generează un eveniment de tipul specificat, cu N-ul specificat
4. introduce evenimentul generat în coada de evenimente

Evenimentele
Evenimentele vor fi implementate ca niște obiecte Java, care trebuie să conțină tipul evenimentului și N-ul specific evenimentului (de exemplu, pentru primul eveniment din fişierul dat mai sus, se va stoca tipul ca fiind PRIME, iar N-ul 50, adică evenimentul va necesita calculul celui mai mare număr prim mai mic sau egal 50).

Coada de evenimente
Odată generat, un eveniment este adăugat în coada de evenimente, o structură de tip FIFO. Coada de evenimente funcționează ca un buffer dintr-o problemă de tip producători-consumatori: nu se poate scrie în ea cat timp este plină, nu se poate citi din ea cât timp este goală, și nu se poate scrie și/sau citi din ea din mai multe threaduri simultan. Este permisă folosirea obiectelor de tip BlockingQueue din

Java pentru implementarea cozii de evenimente. Coada trebuie implementată explicit, chiar dacă se foloseşte ExecutorService pentru worker pool, deoarece trebuie să se poată controla dimensiunea ei.

Worker Pool

Un eveniment poate fi procesat de oricare worker din pool. Astfel, dacă sunt evenimente în coadă și workeri disponibili în pool, se va lua primul eveniment din coadă și va fi alocat unui worker. Workerul va procesa evenimentul respectiv calculând valoarea specificată de tipul evenimentului și de N-ul corespunzător, și va scrie acea valoare într-o structură (listă, array, etc.) specifică evenimentului respectiv (de exemplu, va exista o listă cu rezultatele evenimentelor de tip PRIME, una cu rezultatele evenimentelor de tip FACT, etc.). Pentru implementare, este permisă folosirea obiectelor de tip ExecutorService din Java.

Datele de ieșire

În momentul în care toate evenimentele au fost prelucrate și toate threadurile au terminat de rulat, threadul Main va sorta structurile de date cu rezultatele evenimentelor, și le va scrie pe fiecare în câte un fișier, câte o valoare pe o linie. Astfel, programul va avea 4 fișiere de ieșire:

- PRIME.out
- FACT.out
- SQUARE.out
- FIB.out

Exemplu

Fie următoarele două fişiere de intrare:

```
440,FIB,67
70,FIB,57
50,FIB,69
190,SQUARE,25
130,SQUARE,77
```

```
500,FIB,24
340,FIB,90
160,SQUARE,100
200,PRIME,67
280,FACT,27
```

Există 5 evenimente de tip FIB, deci fişierul FIB.out ar trebui să arate în felul următor:

```
8
10
10
10
11
```

Fişierul SQUARE.out va conţine rezultatele evenimentelor de tip SQUARE, şi va arăta astfel:

```
5
8
10
```

Există un singur eveniment de tip PRIME, deci fişierul de ieşire va conţine doar numărul 67. De asemenea, fiind un singur eveniment FACT, fişierul FACT.out va conţine numărul 4.

Teste şi punctare

Pentru a verifica funcţionalitatea temei, puteţi folosi testele din arhiva de pe site-ul de curs. Pentru rularea testelor, puteţi folosi scriptul Bash din arhivă, care rulează jar-ul vostru pentru un set de fişiere de input, şi compară fişierele rezultate cu un set de rezultate corecte. Pentru a rula scriptul, este necesar ca jar-ul să aibă numele **eventqueue.jar** şi să se afle în acelaşi director cu scriptul şi cu directoarele cu fişierele de input şi output de test.

Atenţie! Deoarece o parte din teste sunt mari, rularea scriptului pentru toate testele poate lua chiar câteva minute. Pentru verificarea funcționalității, puteți rula doar primele teste (care au dimensiuni mai mici), dar la corectare se vor rula toate testele.

Punctajul temei va fi distribuit astfel:

- implementarea paşilor conform specificaţiei temei - **60 puncte**
- consistenţa rezultatelor şi lipsa deadlock-urilor (la rulări multiple, trebuie obţinute aceleaşi rezultate, iar programul nu trebuie să se blocheze) - **40 puncte**

Conţinutul arhivei temei

Arhiva temei va conţine codul sursă pentru temă, împreună cu un fişier **build.xml** reprezentând paşii de compilare şi împachetare pentru aplicaţie, pentru sistemul de build Ant (tutorial pentru Ant găsiţi aici). De asemenea, arhiva trebuie să mai conțină un fișier README (în orice format text) în care să descrieți pe scurt implementarea.

În urma rulării comenzii **ant compile jar**, va trebui să rezulte un fişier numit **eventqueue.jar**, care va putea fi rulat cu comanda:

```
java -jar eventqueue.jar dim_coadă evenimente_fișier nume_fișier1 nume_fișier2 […]
```

De ce folosim o coadă de evenimente?

Există mai multe motive pentru care se pot folosi cozi de evenimente:

- într-un GUI, atunci când un utilizator apasă un buton sau face un gest cu mouse-ul, trebuie realizată o acţiune (de exemplu, apăsarea unui buton poate duce la apariţia unui meniu); deoarece acţiunea care trebuie făcută poate dura mult, se doreşte ca GUI-ul să nu fie blocat până când are loc acţiunea (vrem să putem să mişcăm mouse-ul în continuare pe ecran cât timp aşteptăm să apară meniul); din acest motiv, evenimentele de GUI sunt adăugate într-o coadă de evenimente, realizându-se astfel decuplarea generării de evenimente de acţiunile care se iau ca urmare a evenimentelor, precum şi o prelucrare asincronă
- în sisteme care lucrează cu zeci de mii de evenimente pe secundă, prelucrarea lor în timp real este crucială; din acest motiv, ele sunt trimise pe coada (sau cozile) de evenimente pentru procesare paralelă, ajungându-se astfel la o disponibilitate mai mare; mai mult, utilizarea cozilor de evenimente ajută la asigurarea robusteţii în cazul în care unele componente eşuează (evenimentele generate nu se pierd, ci sunt stocate în coadă)
- folosirea cozilor de evenimente ajută la uşurarea comunicării între diferite procese sau componente ale unui sistem distribuit, oferind totodată şi flexibilitate: tehnologii diferite pot comunica foarte uşor printr-o coadă de evenimente, fără cuplare directă

Resurse utile

http://gameprogrammingpatterns.com/event-queue.html

https://www.techopedia.com/definition/24963/event-queue
https://en.wikipedia.org/wiki/Message_queue