ALGORITMI PARALELI ȘI DISTRIBUIȚI
Tema #2 Manager de comenzi de Black Friday în Java

Responsabili: Gabriel Guțu-Robu, George-Bogdan Oprea, Dorian-Alexandru Verna, Lucian-Marius Iliescu, Radu-Ioan Ciobanu

Termen de predare: 18-12-2022 23:59 (soft), 23-12-2023 23:59 (hard)
Ultima modificare: 01-12-2022 14:30

Cuprins

Cerință

Pentru această temă trebuie să implementați un procesator de comenzi de Black Friday în limbajul de programare Java, care să folosească mecanisme de paralelizare.

Ideea temei este acea de a prelucra comenzi în paralel, respectiv de a prelucra și fiecare produs în parte (chiar și din cadrul aceleiași comenzi) în mod paralel. Puteți face o analogie cu modul în care sunt trimise (sau cel puțin erau la un moment dat) produsele din comenzile eMAG: din cauza faptului că se pot afla în depozite diferite, de multe ori produse din aceeași comandă pot pleca în colete diferite prin curier pentru eficientizarea livrării. Scopul temei este acela de a simula acest produs. Astfel, la un moment de timp o comandă poate avea o parte produse dintre produse expediate (*shipped*), însă doar când toate produsele din cadrul său sunt trimise putem spune că întreaga comandă este *shipped*.

Programul trebuie să citească două fișiere de intrare (ce conțin comenzi, respectiv produsele conținute în cadrul acestora) și să creeze două fișiere de ieșire în care să scrie comenzile expediate, respectiv produsele expediate.

Datele de intrare

Fișierele de intrare sunt următoarele:

1. **orders.txt** cu liniide forma:

```
id_comanda,nr_produse
```

unde *id_comanda* este un idcomandă (prefixat cu o_).

Reguli fișier orders.txt:

- *id_comanda* este un id unic în fișier (nu pot exista id-uri duplicate de comenzi);
- *nr_produse* este numărul de produse conținute de acea comandă;
- o comandă din fișier poate să conțină *0* (zero) produse, caz în care nu va apărea în fișierul de ieșire (comenzi de tip *Empty Order*).

2. **order_products.txt** cu linii de forma:

```
id_comanda,id_produs
```

unde *id_comanda* este id-ul unei comenzi, iar *id_produs* este id-ul unui produs (prefixat cu p_),

Reguli fișier order_products.txt:

- același id de produs se poate regăsi de mai multe ori în fișier (în mai multe comenzi, dar și în cadrul aceleași comenzi - vezi mai jos);
- pot exista linii duplicat, caz în care se consideră că acel produs a fost comandat în cantitate multiplă pentru aceeași comandă (atâtea bucăți câte linii sunt pentru o anumită comandă);
- pot exista linii cu produse al căror id de comandă să NU existe în fișierul *orders.txt*; în acest caz se consideră că produsele sunt de tipul *Abandoned Cart* (au fost adăugat în coș, dar comanda nu a fost trimisă).

Atenție! NU vor exista headere (de tip CSV) în fișierele de intrare!

Datele de ieșire

După rularea programului trebuie să obțineți fișierele următoare:

1. **orders_out.txt** cu structura următoare (unde secțiunea *status* va conține *shipped* pentru acele comenzi ale căror toate produse au fost livrate):

```
id\_comanda,nr\_produse,status
```

2. **order_products_out.txt** cu structura următoare (unde coloana status va conține *shipped* pentru produsele livrate):

```
id\_comanda,id\_produs,status
```

Atenție! Atât în fișierul *orders_out.txt*, cât și în fișierul *order_products_out.txt*, ordinea în care scrieți liniile **NU** este relevantă (le puteți scrie în ce ordine doriți). Script-urile de testare sortează liniile alfabetic (folosind comanda *sort* din bash) pentru a compara rezultatele voastre. **NU** trebuie să includeți headere (de tip CSV) în fișierele de ieșire!

Workflow

Programul vostru trebuie să primească ca parametri de intrare:

- folder-ul ce conține fișierele de intrare (de unde vor citi cele două fișiere, *orders.txt* și *order_products.txt*);
- numărul de thread-uri pentru paralelizare P.

Programul va trebui să implementeze mai multe categorii de thread-uri, astfel încât să respectați pașii specificați mai jos. Se dorește paralelizare la fiecare pas, însă vor exista un număr maxim P de thread-uri la fiecare nivel de paralelizare.

Atenție! În implementarea voastră poate fi nevoie să existe două niveluri de paralelizare (anumite thread-uri să creeze alte thread-uri). Suma numărului de thread-uri care rulează la un moment dat la un anumit nivel trebuie să fie maxim P!

1. Se consideră faptul că toate comenzile plasate către eMAG sunt salvate în fișierul *orders.txt*. Citirea fișierului *orders.txt* trebuie să se facă în mod paralel - mai mulți angajați (ideal P) citesc comenzile primite în mod paralel. Împărțiți sarcinile de lucru angajaților în mod aproximativ egal. Imaginați-vă că un agent eMAG prelucrează câte o comandă din fișier.
2. Fișierul *orders.txt* conține inclusiv numărul de produse din cadrul unei comenzi (practic se consideră suma cantităților tuturor produselor). Pentru fiecare comandă citită din fișierul *orders.txt* se vor căuta produsele care sunt în acea comandă (atâtea produse cât specifică numărul *nr_produse* din fișierul *orders.txt*) în fișierul *order_products.txt*. Puteți face analogia cu faptul că agentul eMAG care prelucrează comanda creează sarcini de lucru pentru persoane care se vor ocupa de prelucrarea produselor.
3. Thread-urile care se vor ocupa de produse vor adăuga textul *,shipped* la finalul liniei corespunzatoare produsului pe care l-au prelucrat în fișierul *order_products_out.txt*. Considerați faptul că agenții eMAG care se ocupă de prelucrarea unui produs vor realiza o acțiunie dummy, după care se consideră că au prelucrat produsul și-l vor marca expediat în fișierul de ieșire.
4. Când toate cele *nr_produse* au fost livrate, thread-ul care a procesat comanda respectivă va scrie în fișierul *orders_out.txt* textul *,shipped* la finalul linei corespunzatoare comenzii de care s-a ocupat. O comandă va primi *,shipped* doar după ce toate produsele din cadrul său au primit *,shipped*! Astfel, agentul eMAG care s-a ocupat de o comandă o va marca expediată după ce toate produsele din cadrul său au fost marcate ca expediate.

Atenție! Într-o implmentare corectă, thread-urile care vor scrie în fișierul *order_products_out.txt* **NU** vor verifica dacă textul *,shipped* a fost adăugat anterior (îl vor scrie direct). Justificarea este aceea că workflow-ul ar trebui să funcționeze în așa fel încât împărțirea task-urilor să se facă fără ca un alt thread să se fi ocupat de produsul unui anumit thread.

Programul trebuie să genereze fișierele de ieșire *orders_out.txt* și *order_products_out.txt* direct în folder-ul unde se află "executabilul".

Arhiva voastră trebuie să conțină un fișier *Makefile* care să genereze fișierul **Tema2.class** la rularea comenzii make. Sursele voastre (fișierele Java) NU trebuie să conțină pachete!

Mecanisme de paralelism

Puteți folosi orice mecanism de sincronizare învățat: lock-uri, bariere, work pool, wait/notify, etc.

Bonus

Pentru bonus se dorește o implementare în care fiecare thread care citește din fișierul *orders.txt* să se ocupe doar de o secțiune din fișier (nu să îl citească pe tot), astfel încât să "își dea seama" de secțiunea de care trebuie să se ocupe. Bonusul valoarează **15 puncte**.

Exemplu

Programul se va rula în felul următor:

```
java Tema2 <folder_input> <nr_max_threads>
```

Se dau următoarele fișiere de intrare:

```
$ cat sample\_files/orders.txt
o_z1educ4,6
o_lovfpw0,3
o_dhms1d5,7

$ cat sample\_files/order\_products.txt
o_z1educ4,p_fdj3mele6v
o_dhms1d5,p_f4wa9o18ac
o_z1educ4,p_l2rxrmr3iu
o_dhms1d5,p_17u2b4v409
o_lovfpw0,p_3jtg509396
o_z1educ4,p_fdj3mele6v
o_dhms1d5,p_9kbicwfs64
o_z1educ4,p_rewpkndf37
o_lovfpw0,p_vh6bwlkpd4
o_z1educ4,p_w8veo5jlz5
o_dhms1d5,p_w8veo5jlz5
o_z1educ4,p_f4wa9o18ac
o_lovfpw0,p_fdj3mele6v
o_dhms1d5,p_4k1mnu1dvz
o_dhms1d5,p_vh6bwlkpd4
o_dhms1d5,p_vh6bwlkpd4
```

Un exemplu de fișiere de ieșire este prezentat în continuare:

```
$ cat orders\_out.txt
o_lovfpw0,3,shipped
o_z1educ4,6,shipped
o_dhms1d5,7,shipped

$ cat order\_products\_out.txt
o_z1educ4,p_fdj3mele6v,shipped
o_z1educ4,p_f4wa9o18ac,shipped
o_z1educ4,p_fdj3mele6v,shipped
o_dhms1d5,p_vh6bwlkpd4,shipped
o_dhms1d5,p_17u2b4v409,shipped
o_z1educ4,p_w8veo5jlz5,shipped
o_lovfpw0,p_vh6bwlkpd4,shipped
o_z1educ4,p_rewpkndf37,shipped
o_z1educ4,p_l2rxrmr3iu,shipped
o_dhms1d5,p_9kbicwfs64,shipped
o_lovfpw0,p_3jtg509396,shipped
o_dhms1d5,p_f4wa9o18ac,shipped
o_dhms1d5,p_vh6bwlkpd4,shipped
o_lovfpw0,p_fdj3mele6v,shipped
o_dhms1d5,p_4k1mnu1dvz,shipped
o_dhms1d5,p_w8veo5jlz5,shipped
```

Notare

Tema se poate testa local, după cum se explică mai jos, sau automat. Checker-ul automat se utilizează după informațiile din acest document. Tema se va încărca pe Moodle. Se va încărca o arhivă Zip care, pe lângă fișierele sursă Java, va trebui să conțină următoarele două (sau trei) fișiere **în rădăcina arhivei**:

- *Makefile* - cu directiva *build* care compilează tema voastră și generează un fișier numit *Tema2.class* aflat în rădăcina arhivei, și directiva *clean* care șterge fișierul .class
- *README* - fișier text în care să se descrie pe scurt implementarea temei
- *README_BONUS* (opțional) - fișier text în care să se descrie pe scurt implementarea bonusului, doar dacă l-ați realizat.

Punctajul este divizat după cum urmează:
To be updated
Atenție! Dacă implementați bonusul, este necesar să adăugați în arhivă și un fișier *README_BONUS* în care descrieți ce ați făcut.

Testare

Pentru testarea temei puteți rula script-ul *test.sh*, pe care îl veți regăsi în repository-ul temei. Acesta va rula teste (variind numărul de thread-uri folosind ca date de intrare fișierele aflate în folder-ul input (unde se regăsesc subfolder-ele *input_0* ... *input_9*) și va verifica fișierele *orders_out.txt* și *order_products_out.txt* comparativ cu fișierele de output din folder-ul *output*.

Pentru a vă putea testa tema, găsiți în repository-ul temei un set de fișiere de intrare de test, precum și un script Bash (numit *test.sh*) pe care îl puteți rula pentru a vă verifica implementarea. Acest script va fi folosit și pentru testarea automată[1].

[1] Nota obținută în urma rulării automate poate fi scăzută pe baza elementelor de depunctare descrise mai sus.

Pentru a putea rula scriptul așa cum este, trebuie să aveți următoatoarea structură de fișiere:

```
$ tree
.
+-- Makefile
+-- [...] (sursele voastre)
+-- README
+-- test.sh
+-- input
|   +-- input_0
|      +-- [...] (fisierele testului)
|   +-- input_1
|      +-- [...] (fisierele testului)
|   +-- input_2
|      +-- [...] (fisierele testului)
|   +-- input_3
|      +-- [...] (fisierele testului)
|   +-- input_4
|      +-- [...] (fisierele testului)
|   +-- input_5
|      +-- [...] (fisierele testului)
|   +-- input_6
|      +-- [...] (fisierele testului)
|   +-- input_7
|      +-- [...] (fisierele testului)
|   +-- input_8
|      +-- [...] (fisierele testului)
|   +-- input_9
```

La rulare, scriptul execută următorii pași:

1. compilează programul
2. rulează programul de 10 ori, de fiecare dată dând ca parametru unul dintre folder-ele de intrare și un număr maxim de thread-uri (parametrul P
3. sortează crescător alfabetic liniile din fișierele *orders.txt* și *order_products.txt*
4. compară fișierele de ieșire sortare cu cele din folder-ul *output*
5. calculează punctajul final din cele maxim 100 de puncte