Model testare - Implementarea Concurenței în Limbajele de Programare
Opţional 2024-2025

Materiale permise: materialele de curs și cele de laborator, inclusiv fișiere cod cu soluții ale problemelor rezolvate în curs și laborator; în plus, fiecare student poate avea o foaie (2 pagini) în format fizic (scris de mână/printat) cu elemente de limbaj (Haskell, Java).

În timpul examenuli NU AVEȚI VOIE SĂ AVEȚI BROWSERE DESCHISE, NU PUTEȚI CONSULTA MATERIALE ONLINE.

Fiecare student va avea un folder pe care îl va denumi "nume-student-ICLP-2025" în care va avea aceste materiale și în care va rezolva exercițiile. În acest folder, pe parcursul examenului NU pot să apară alte fișiere în afara celor menționate.

La finalul examenului veți face submisia subiectelor rezolvate NUMAI în prezența unui supraveghetor. Vă vom anunța în timpul examenului modul de submitere.

1 Partea 0x00 - Java (maxim 3 puncte)

Se pot alege si limbajele Python, respectiv Go pentru rezolvarea acestui subiect.

1.1 Testare 2024

Sa se scrie un program care citeste de la `STDIN` un *path* absolut, doua numere intregi `2 < N < 10` si `3 < M < 8` si implementeaza urmatorul scenariu: prin intermediul a `N` *thread*-uri cu rol dedicat, vor fi citite toate fisierele text continute in *path*-ul specificat, inclusiv in subfoldere (*vezi codul auxiliar de mai jos*), iar continutul lor va fi scris de alte `M` *thread*-uri la `STDOUT`, pastrand doar caracterele alfanumerice. Implementati si un monitor care sa retina toate *log*-urile, cu informatii de forma: `Thread Id {threadId} is currently working on file {filePath}` sau `Thread Id {threadId} is currently writing at standard output.` Implementati un mecanism astfel incat scrierea la `STDOUT` sa fie efectuata dupa prelucrarea a cel putin 2 fisiere text. Justificati, intr-un comentariu, modul in care ati decis sa implementati acest mecanism.

Cod auxilar.

```
class FileWalker {
    public ArrayList<String> walk(String path) {
        ArrayList<String> files = new ArrayList<>();

        File root = new File(path);
        File[] list = root.listFiles();

        if (list == null) {
            return files;
        }

        for (File f : list) {
            if (f.isDirectory()) {
                files.addAll(walk(f.getAbsolutePath()));
            }
            else if (f.getName().endsWith(".txt")) {
                files.add(f.getAbsolutePath());
            }
        }

        return files;
    }
}
```

1.2 Testare 2023

Folosind `ExecutorService`, implementati un `array` de actiuni care sa se execute asincron. In executarea propriu-zisa a acestor actiuni, logati, la `STDOUT`, si indexul actiunii care s-a executat, nu doar rezultatul.

a. Particularizati scenariul pentru calculul in paralel a functiei `fibonacci`. Cele `N` pozitii din `array`-ul `v` vor executa `fibonacci(v[i])`, $0 \leq \text{i} < N$.

b. Particularizati scenariul intr-un context `Reader-Writer`, astfel:

– Fiecare actiune executa un `Reader` si poate sa citeasca linii dintr-o baza de date, intre doua limite, `N` si `M`. Se garanteaza `N < M`.

– Dupa ce au citit din baza de date, peste liniile selectate se aplica o functie `computeResult`. Cu ajutorul `thread`-urilor `Writer` se updateaza sursa de date, etichetand fiecare linie cu scor-ul `batch`-ului. In acelasi timp, toate rezultatele se trimit sub forma `(index, result)` unui alt `thread`, cu rolul de monitor, care retine mereu doar perechea corespunzatoare celui mai mare scor obtinut.

– Dupa ce toate actiunile s-au executat, `thread`-ul monitor afiseaza la `STDOUT` indexul actiunii cu cel mai bun scor.

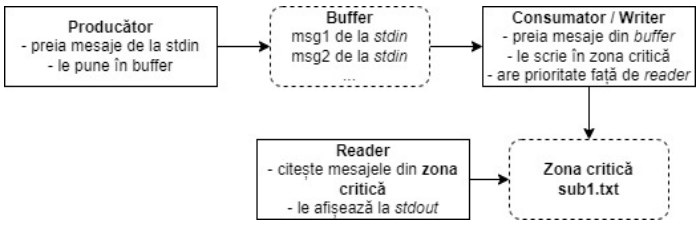
Observatie: Pentru implementare, citirea/scrierea/actualizarea in baza de date vor fi actiuni care doar vor dura un anumit timp (`random`) si vor respecta o signatura intuitiva. Trebuie doar sa implementati arhitectura acestui scenariu: modul de functionare pentru `readeri` si `writeri`, respectiv sistemul final de vot, care doar calculeaza maximul dintre rezultatele functiei pe `batch`-uri. Functia `computeResult` poate fi implementata sa returneze un `random`.

1.3 Testare 2022

Se citesc mesaje de la *standard input*. Să se implementeze un scenariu concurent care efectuează următoarele:

1. mesajele vor fi citite într-un model *Producer-Consumer*;
2. *consumatorul* din acest model va fi un *Writer* într-un model *Reader-Writer*. În cadrul acestuia, *thread*-urile *Writer* vor avea prioritate față de *thread*-urile *Reader*. Mesajele citite vor fi scrise de *writeri* într-un fișier *sub1.txt* (fiecare *writer* va adăuga mesajul său în această zonă critică), iar *readerii* vor afișa, la *standard output*, conținutul respectivului fișier.

Important! *Thread*-urile *Reader* nu pot porni până ce nu avem garanția că a fost cel puțin mesaj scris în zona critică.



2 Partea 0x01 - Haskell (maxim 3 puncte)

2.1 Testare 2024

O companie cu 50 de angajaţi şi 5 secretare are un spaţiu care conţine o zonă de conferinţe şi o zonă de printare. Zona de printare este folosită de una din cele 5 secretare ale companiei, dar secretarele nu pot folosi zona simultan. Zona de conferinţe are 10 locuri, accesul participantilor fiind permis numai cand grupul de angajaţi este complet. Angajaţii încearcă să participe la o conferinţe de mai multe ori pe parcursul unei zile. Fiecare secretară trebuie să printeze de mai multe ori pe parcursul unei zile. Accesul la cele două zone este gestionat de un administrator, care nu lasa angajaţii şi secretarele să desfăşoare activităţi simultan. Scrieţi un program în Haskell care să implementeze accesul la spaţiul descris anterior.

2.2 Testare 2023

Intr-o intersectie aglomerata, este montat un semafor inteligent care sa ajute la eficientizarea traficului. Semaforul are montata o camera prin care poate recunoaste masini, pietoni si animale. Modul in care semaforul permite trecerea prin intersectie este urmatorul:

- va permite trecerea grupurilor de pietoni, cand exista 5 pietoni in asteptare sa traverseze (semaforul va deveni rosu pentru masini);
- va permite trecerea masinilor, cand exista 10 masini in asteptare sa treaca prin intersectie (semaforul va deveni verde pentru masini);
- va opri trecerea masinilor de fiecare data cand va recunoaste un animal ca vrea sa treaca strada (semaforul va deveni rosu pentru masini).

Semaforul va functiona cu urmatoarea restrictie: animalele si pietonii au prioritate in fata masinilor. In scenariul prezentat, grupurile de animale, pietoni si masini vor veni la infinit.

2.3 Testare 2022

Definim un canal de comunicare printr-o variabilă `MVar` care conține o listă de întregi. Mesajele care se pot scrie / citi folosind acest canal vor fi numere întregi.

```
import Control.Concurrent
data MList = ML (MVar [Int])
```

Scrieți un program concurent care lansează două *thread*-uri, unul care citește, respectiv unul care scrie, astfel:

- *thread*-ul care scrie, va scrie începând din stânga canalului întregii introduși la *standard input*;
- *thread*-ul care citește, va citi din dreapta canalului, va afișa la *standard output* dublul numărului citit, și va elimina informația prelucrată.

Programul își va înceta executarea în momentul în care se va citi întregul 0. Nu se vor face verificări suplimentare, presupunem că utilizatorul introduce date de tipul așteptat (`Int`). Reamintim că aceste date sunt citite de la *standard input* ca șiruri de caractere.

3 Partea 0x02 - Erlang (maxim 3 puncte)

Modelati un scenariu in care avem o cladire cu doua lifturi si mai multi oameni care interactioneaza cu cele doua lifturi. Cineva poate sa solicite un lift pentru a merge la un anumit etaj, iar lifturile trebuie sa raspunda cererilor. Consideram, pentru lifturi, starile `stationary(Floor)` (liftul stationeaza la un anumit etaj), `moving(Direction, Floor)` (liftul se deplaseaza spre un anumit etaj, iar `Direction` poate fi `up` ori `down`). Pentru fiecare persoana, avem starile `waiting(Floor, Destination)` (asteapta la un anumit etaj, pentru a ajunge la un altul), respectiv `in_lift(Lift, Destination)` (in ce lift se afla, si la ce etaj vrea sa ajunga) Fiecare om trimite cererea catre ambele lifturi, si va fi prioritar liftul care e cel mai apropiat. Lifturile trebuie sa termine cate un transport pentru a putea prelua noi cereri. Lifturile au, initial, o capacitate maxima.