

Synchronization Mechanisms Lab Assignment

December 12, 2019

Student: Brinzan Florinel-Razvan

Calculatoare si tehnologia informatiei
(limba romana)

C.R 3.1 A

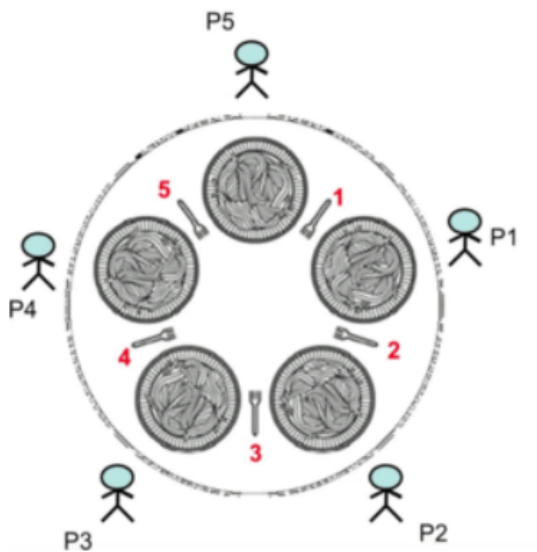
Anul 3

1 Problema 2 – Problema cinei filosofilor

1.1 Enuntul problemei

— Implementați problema cinei filosofilor folosind:

1. semafoare
2. monitoare
3. lacate



Pe scurt, la o masa sunt **5 filosofi** si **5 furculite**. Filosofi fac doua activitati separate: se gandesc si mananca.

Pentru a manca, un filosof are nevoie de doua furculite. Asta in-seamna ca pentru a manca toti cei 5 filosofi in acelasi timp ar fi nevoie de 10 furculite (ceea ce nu este posibil).

Doi filosofi alaturati impart aceeasi furculita.

Spre exemplu:

Filosofului 1 ii apartine:

- *furculita 1*
- *furculita 2*

Filosofului 2 ii apartine:

- *furculita 2*

- *furculita 3*

Filosofului 3 ii apartine:

- *furculita 3*

- *furculita 4*

Filosofului 4 ii apartine:

- *furculita 4*

- *furculita 5*

Filosofului 5 ii apartine:

- *furculita 5*

- *furculita 1*

Pentru a putea manca, un filosof trebuie sa achizitioneze ambele furculite. Dupa ce a terminat de mancat el va elibera furculitele pentru a putea fi folosite si de vecinii din imediata lui apropiere.

1.2 Implementarea problemei in Java

1.2.1 Utilizarea semafoarelor

— In implementarea acestei probleme am creat **3 clase** : *clasa Furculita*, *clasa Filosof* si *clasa MainClass*.

1. In *clasa Furculita* am instantiat un obiect de tip semafor cu o **permisiune** deoarece consider starea initiala a furculitei ca fiind ”**disponibila**”, deci oricare dintre filosofii care au primit-o ca parametru o pot achizitiona, dupa care va fi nevoie de apelarea metodei **release()** pentru a o elibera.

— In *clasa Furculita* sunt implementate **2 metode** :

- **prima metoda** (*boolean ridicaFurculita()*) **incearca** dobandirea permisiunii semaforului prin apelarea metodei *tryAcquire()*. Daca va reusi va returna **True** iar daca nu va reusi va returna **False**. In cazul in care va returna **True**, furculita va fi achizitionata de thread-ul (filosoful) care a facut apelul, iar daca va returna **False** inseamna ca **incercarea de achizitionare a furculitei a esuat** din cauza ca ea este detinuta de alt thread (filosof).

- a doua metoda (*void puneJosFurculita()*) elibereaza un permis, returnandu-l semaforului. Dupa apelarea acestei metode, semaforul poate acorda permisul altui fir de executie (daca i se va solicita acest lucru). Prin urmare, dupa apelarea acestei metode, oricare filosof care a primit aceasta furculita ca parametru o poate achizitiona.

2. **Clasa Filosof extinde clasa Thread si suprascrisce metoda "run()".** In aceasta clasa sunt implementate metodele necesare simularii problemei filosofilor. Pe langa metodele necesare implementarii problemei filosofilor, clasa Filosof contine doua obiecte de tip Furculita (furculita stanga si furculita dreapta) cu ajutorul carora filosoful poate manca. Membrul privat de tip data - numarul intreg "id" este folosit pentru a identifica filosoful. Fiecare filosof are propriul id.

—Aceasta clasa contine implementarea constructorului cu parametri si 3 metode: *void Activitate(String)*, *void run()*, *void mananca()*.

- **Constructorul clasei Filosof** are rolul de a asigna membrilor sai privati de tip data valorile date ca parametri (cele doua furculite cu ajutorul carora poate manca filosoful si id-ul unic).

PSEUDOCOD

```
this.FurculitaStanga <- FurculitaStanga
this.FurculitaDreapta <- FurculitaDreapta
this.id <- id
```

- Metoda *Activitate(String)* are rolul de a afisa in consola activitatea curenta a firului de executie (filosofului) care o apeleaza. Prin intermediul obiectului "random" se genereaza un numar in intervalul [0, 1000) care va fi dat ca parametru functiei *sleep()* pentru a produce o intarziere.
- Metoda *run()* din clasa Filosof suprascrisce metoda "run()" din clasa Thread. Aceasta metoda contine o bucla infinita ce apeleaza la inceput metoda "Activitate(String)" pentru a semnala faptul ca filosoful respectiv se gandeste apoi apeleaza metoda "mananca()". Apeland metoda "mananca()" filosoful va incerca sa manance. Daca nu va avea disponibile ambele furculite nu va manca si va trece din nou la starea de gandire.
- Metoda *mananca()* testeaza initial daca filosoful poate achizitiona furculita stanga. Daca nu o poate achizitiona se va afisa mesajul corespunzator prin intermediul metodei "Activitate(String)" si se va iesi din functie. Daca va achizitiona furculita stanga va incerca sa

o achizitioneaza si pe cea dreapta. Daca nu va reusi se va afisa mesajul corespunzator prin intermediul metodei "**Activitate(String)**" si va elibera furculita stanga pentru a evita interblocajul (**deadlock**). **Daca filosoful va reusi sa achizitioneze albele furculite atunci va manca.** Se va afisa lucrul acesta prin intermediul metode "**Activitate(String)**" apoi **va elibera ambele furculite.**

PSEUDOCOD

```

daca furculita stanga nu e utilizata
    ridica furculita stanga
    afiseaza "Filosoful X ridica furculita din Stanga."

daca furculita dreapta nu e utilizata
    ridica furculita dreapta
    afiseaza "Filosoful X ridica furculita din
              Dreapta."
    afiseaza "Filosoful X MANANCA."
    afiseaza "Filosoful X a terminat de mancat !!
              Acum Pune jos furculita din Stanga si pe cea
              din Dreapta."

    elibereaza furculita stanga
    elibereaza furculita dreapta

altfel
    afiseaza "Filosoful X lasa furculita din
              Stanga jos din cauza ca furculita din
              Dreapta nu e disponibila, deci nu poate
              Manca."
    elibereaza furculita stanga

altfel
    afiseaza "Filosoful X a vrut sa Manance dar nu a
              reusit din cauza ca furculita din Stanga era
              luata de alt filosof.
```

3. Clasa **MainClass** contine metoda "**main()**" in care sunt instantiati filosofii si obiectele de tip **Furculita**.

- In aceasta clasa sunt create thread-urile (filosofii) si furculitele si este apelata metoda "**start()**" ce pune in executie thread-urile. Fiecarui filosof ii sunt asignate 2 furculite si un **id**. Fiecare furculita din cele 5 va fi data ca parametru pentru 2 filosofi (ex. filosoful 1 va avea furculitele 1 si 2 iar filosoful 2 va avea furculitele 2 si 3). Numarul filosofilor si furculitelor este dat prin intermediul variabilelor statice - finale "**NR-FILOSOFI**" si "**NR-FURCULITE**".

PSEUDOCOD metoda main()

```
NR_FILOSOFI = 5;
NR_FURCULITE = 5;

Filosof[] filosofi <- new Filosof[NR_FILOSOFI]
Furculita[] furculite <- new Furculita[NR_FURCULITE]

pentru iterator de la 0 la NR_FURCULITE - 1 executa
    furculite[iterator] <- new Furculita()

pentru iterator de la 0 la NR_FILOSOFI - 1 executa
    Furculita FurculitaStanga <- furculite[iterator]
    Furculita FurculitaDreapta <- furculite[(iterator +
        1) mod NR_FURCULITE]

    filosofi[iterator] <- new Filosof(FurculitaStanga,
        FurculitaDreapta, iterator)
    start filosofi[iterator]
```

1.2.2 Utilizarea lacatelor

- In implementarea acestei probleme am creat **3 clase** : *clasa Furculita*, *clasa Filosof* si *clasa MainClass*.
- Clasele **Filosof** si **MainClass** au ramas neschimbate (descrierea lor este prezentata mai sus). **Modificarea a intervenit numai in clasa Furculita**.
- **Clasa Furculita** contine un obiect de tip **ReentrantLock** prin intermediul caruia furculita este blocata cat timp se afla in posesia unui filosof si deblocata in momentul in care in care filosoful o elibereaza. Metodele folosite pentru blocarea respectiv deblocarea furculitei sunt: *ridicaFurculita()* si *puneJosFurculita()*.
- **Metoda *ridicaFurculita()*** – aceasta metoda incearca achizitionarea furculitei prin apelul functiei *tryLock()*. **Daca furculita va fi disponibila atunci ea va fi achizitionata de filosoful care face apelul si se va bloca lacatul, returnandu-se *true*. Daca furculita este deja blocata (este folosita in acel moment de alt filosof) se va returna *false*.**
- **Metoda *puneJosFurculita()*** – aceasta metoda elibereaza furculita deblocand lacatul.

1.2.3 Utilizarea monitoarelor

- In implementarea acestei probleme am creat **3 clase** : *clasa Furculita*, *clasa Filosof* si *clasa MainClass*.
- Clasele **Filosof** si **MainClass** au ramas neschimbate (descrierea lor este prezentata mai sus). **Modificarea a intervenit numai in clasa Furculita.**
- **Clasa Furculita** contine o **variabila booleana** ce memoreaza starea furculitei (**false** - furculita disponibila, **true** - furculita indisponibila) si **3 metode sincronizate** : achizitionarea furculitei, eliberarea furculitei si returnarea starii furculitei.
- **Metoda *synchronized void ridicaFurculita()*** – aceasta metoda incerca achizitionarea furculitei. Daca furculita este indisponibila atunci firul curent de executie va astepta pana va fi trezit (notificat). Daca furculita este disponibila atunci ea va fi achizitionata de filosoful care face apelul.

PSEUDOCOD

```
cat timp stare == true
    asteapta
daca stare == false atunci
    stare <- true
```

- **Metoda *synchronized void puneJosFurculita()*** – Aceasta metoda elibereaza furculita, seteaza starea pe false si trezeste toate firele de executie care asteapta pe **monitorul** acestui obiect.

PSEUDOCOD

```
stare <- false
trezeste thread-urile
```

- **Metoda *synchronized Boolean getState()*** – Aceasta metoda returneaza starea furculitei. Returneaza **TRUE** daca furculita este indisponibila si **FALSE** in cazul in care este disponibila

1.3 Rezultate, Scenarii posibile si descrierea output-ului

- Principala problema ce poate interveni (chiar daca implementarea semafoarelor, lacatelor sau monitoarelor este facuta corect) este *deadlock-ul*.
- *Deadlock-ul* apare in momentul in care toti cei 5 filosofi ridica furculita stanga si dorind sa o ridice si pe cea dreapta nu reusesc.
- Pentru a scapa de aceasta problema, am pus conditia ca daca un filosof detine furculita stanga, incearca sa o achizitioneze si pe cea dreapta si nu reuseste, atunci el va trebui sa elibereze furculita stanga, astfel, putand fi luata de alt filosof, sau chiar de el la un moment ulterior.
- Implementarea acestei solutii se poate observa in figura de mai jos.

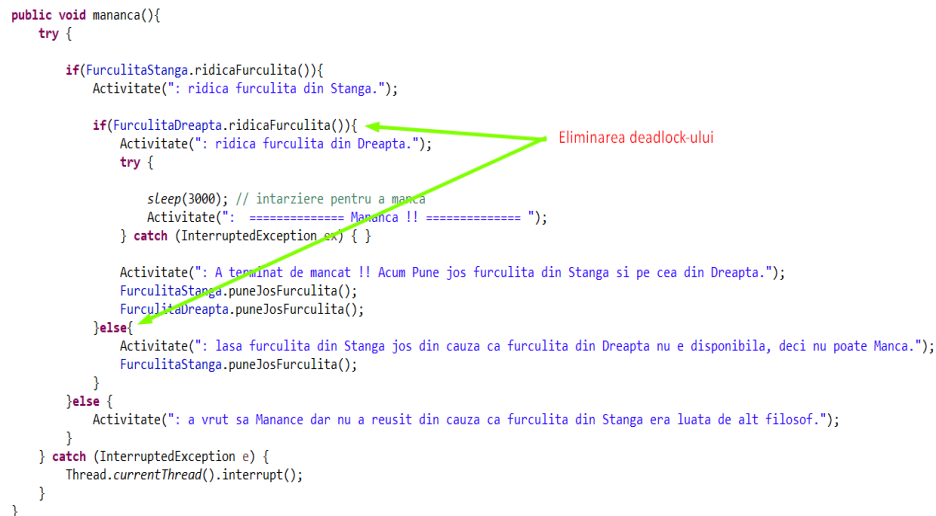
```
public void mananca(){
    try {

        if(FurculitaStanga.ridicaFurculita()){
            Activitate(": ridica furculita din Stanga.");

            if(FurculitaDreapta.ridicaFurculita()){
                Activitate(": ridica furculita din Dreapta.");
                try {

                    sleep(3000); // intarziere pentru a manca
                    Activitate(": ===== Mananca !! ===== ");
                } catch (InterruptedException ex) { }

                Activitate(": A terminat de mancat !! Acum Pune jos furculita din Stanga si pe cea din Dreapta.");
                FurculitaStanga.puneJosFurculita();
                FurculitaDreapta.puneJosFurculita();
            } else {
                Activitate(": lasa furculita din Stanga jos din cauza ca furculita din Dreapta nu e disponibila, deci nu poate Manca.");
                FurculitaStanga.puneJosFurculita();
            }
        } else {
            Activitate(": a vrut sa Manance dar nu a reusit din cauza ca furculita din Stanga era luata de alt filosof.");
        }
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
}
```



- Implementarea de mai sus are urmatoarele rezultate:

```
Filosoful 5 : ----> Se gandeste !!
Filosoful 3 : ----> Se gandeste !!
Filosoful 2 : ----> Se gandeste !!
Filosoful 1 : ----> Se gandeste !!
Filosoful 4 : ----> Se gandeste !!
Filosoful 2 : ridica furculita din Stanga.
Filosoful 3 : ridica furculita din Stanga.
Filosoful 5 : ridica furculita din Stanga.
Filosoful 4 : ridica furculita din Stanga.
Filosoful 1 : ridica furculita din Stanga.
Filosoful 5 : lasa furculita din Stanga jos din cauza ca furculita din Dreapta nu e disponibila, deci nu poate Manca.
Filosoful 5 : ----> Se gandeste !!
Filosoful 2 : lasa furculita din Stanga jos din cauza ca furculita din Dreapta nu e disponibila, deci nu poate Manca.
Filosoful 1 : lasa furculita din Stanga jos din cauza ca furculita din Dreapta nu e disponibila, deci nu poate Manca.
Filosoful 4 : ridica furculita din Dreapta.
Filosoful 3 : lasa furculita din Stanga jos din cauza ca furculita din Dreapta nu e disponibila, deci nu poate Manca.
Filosoful 5 : a vrut sa Manance dar nu a reusit din cauza ca furculita din Stanga era luata de alt filosof.
Filosoful 2 : ----> Se gandeste !!
Filosoful 1 : ----> Se gandeste !!
Filosoful 3 : ----> Se gandeste !!
Filosoful 5 : ----> Se gandeste !!
Filosoful 2 : ridica furculita din Stanga.
Filosoful 3 : ridica furculita din Stanga.
Filosoful 3 : lasa furculita din Stanga jos din cauza ca furculita din Dreapta nu e disponibila, deci nu poate Manca.
Filosoful 1 : ridica furculita din Stanga.
Filosoful 2 : lasa furculita din Stanga jos din cauza ca furculita din Dreapta nu e disponibila, deci nu poate Manca.
Filosoful 5 : a vrut sa Manance dar nu a reusit din cauza ca furculita din Stanga era luata de alt filosof.
Filosoful 2 : ----> Se gandeste !!
Filosoful 2 : ridica furculita din Stanga.
Filosoful 2 : lasa furculita din Stanga jos din cauza ca furculita din Dreapta nu e disponibila, deci nu poate Manca.
Filosoful 3 : ----> Se gandeste !!
Filosoful 2 : ----> Se gandeste !!
Filosoful 1 : ridica furculita din Dreapta.
Filosoful 5 : ----> Se gandeste !!
Filosoful 3 : ridica furculita din Stanga.
Filosoful 5 : a vrut sa Manance dar nu a reusit din cauza ca furculita din Stanga era luata de alt filosof.
Filosoful 2 : a vrut sa Manance dar nu a reusit din cauza ca furculita din Stanga era luata de alt filosof.
Filosoful 4 : ===== Mananca !! =====
Filosoful 5 : ----> Se gandeste !!
Filosoful 3 : lasa furculita din Stanga jos din cauza ca furculita din Dreapta nu e disponibila, deci nu poate Manca.
Filosoful 5 : a vrut sa Manance dar nu a reusit din cauza ca furculita din Stanga era luata de alt filosof.
Filosoful 3 : ----> Se gandeste !!
```

Filosoful 4 ridica furculita eliberata de filosoful 5 astfel eliminandu-se deadlock-ul

- Mai multe cazuri de test sunt prezentate in directorul "Date experimentale P2".

1.4 Concluzii

- Implementand aceasta problema mi-am fixat mai bine cunostintele legate de programarea concurenta, am invatat lucruri noi despre comportamentul firelor de executie cat si despre restrictiile ce ar putea fi aplicate asupra firelor astfel incat rezultatul sa fie cel dorit.
- Utilizarea mecanismelor de sincronizare este extrem de importanta in programarea concurenta. Studiind aceste mecanisme mi-am imbunatatit abilitatile de programare concurenta.
- Consider ca aceste tipuri de exercitii au fost foarte bine alese pentru partea aceasta de studiu a mecanismelor de sincronizare ale firelor de executie deoarece implementand solutiile unor cerinte de acest tip ajungi sa intelegi cu adevarat riscurile executiei unui program pe mai multe fire, stii la ce sa te astepti si cum ai putea rezolva anumite "conflicte" dintre firele de executie.