The Producer Consumer problem

Diaconescu Bogdan-Florin
CR 3.2 A, Anul 3

December 12, 2019

1 Problem statement

Problema producator-consumator presupune crearea a 2 procese, producatorul si consumatorul care utilizeaza o resursa comuna, cu dimensiune fixa, numita buffer, care se comporta ca si o coada. Rolul producatorului este acelasi de a genera date si de a le introduce in buffer. In acelasi timp, rolul consumatorului este de a "consuma" datele din buffer una cate una.

De asemenea, trebuie evitata situatia cand producatorul incearca sa puna date in buffer desi acesta este plin, sau atunci cand consumatorul incearca sa "consume" date din buffer -ul gol.

2 Implementation/Solution

Solutia utilizand monitoare

Pentru a rezolva problema am creat 2 clase, Produse si Consumer care extind clasa Thread din java, respectiv clasa Market, care reprezinta de fapt buffer -ul.

In clasa Market am implementat cele 2 functii produce si consume, iar buffer -ul este reprezentat de un vector de numere intregi de dimensiunea N. Aceste 2 functii sunt de tipul synchronized pentru a ma asigura ca un singur thread poate executa metoda respectiva la un moment dat. In cazul in care buffer -ul este deja plin, in interiorul metodei produce se apeleaza metoda wait(). Aceasta metoda face ca celelalte threaduri care asteapta dupa blocajul aceluiasi obiect sa poata sa intre in lucru. Cu alte cuvinte, prin adaugarea acestei metode se stopeaza producatorul, dar se elibereaza accesul la resursa comuna, adica consumatorul poate sa "consume" date din buffer. Atunci cand este eliberat un loc in buffer, metoda produse este "trezita" prin apelarea metodei notifyAll() de catre consumator.

In clasele Producer si Consumer metoda run() implementeaza "producerea" si "consumul" datelor. Clasa Main contine functia main in care sunt create threadurile producator si consumator si este pornita executia acestora cu metoda start(). Am utilizat 2 variabile MINN si MAXN pentru a generare producatori si consumatori in intervalul 0-76. La final fiecare consumator si producator afiseaza numarul de elemente "consumate", respectiv "produse".

Solutia utilizand zavoare

Rezolvarea problemei utilizand zavoarele este asemanatoare ca cea cu monitoare. Daca la monitoare blocarea si deblocarea se faceau automat, la zavoare, atat deblocarea cat si blocarea zavorului se face utilizand functiile lock() si unlock(). Singura clasa schimbata este clasa Market, unde metodele produce si consume au fost implementate utilizand functii din biblioteca ReentrantLock din java. Am adaugat si 2 variabile boolene notFull si notEmpty pentru a verifica daca bufferul este plin sau gol, care initial au valorile true, respectiv false.

De asemenea, am creat si un zavor numit bufferLock, care la inceputul functiilor procese si consume blocheaza accesul la sectiunea critica prin apelarea functiei lock(). Oricare alt thread care incearca sa apeleze functia lock() va fi blocat pana cand threadul care a apelat functia lock() va debloca bufferLock prin apelul functiei unlock(). La finalul functiilor produce si consume, bufferLock deblocheaza accesul la sectiunea critica prin apelul functiei unlock().

Algorithm 1 Functia produce implementata cu zavoare

```
function PUBLIC VOID PRODUCE(int value)
bufferLock.lock();
  try
if notFull == true then
  $buffer[newest] \leftarrow value$
  $newest \leftarrow (newest + 1)\%N$
  $count + +$
end if
if count == N then
  $notFull \leftarrow false$
end if
$notEmpty \leftarrow true$
  finally
    bufferLock.unlock()
  end try
```

Algorithm 2 Functia consume implementata cu zavoare

```
0: function PUBLIC INT CONSUME
1: $value \leftarrow -1$
2: bufferLock.lock();
2:    try
3: if notEmpty == true then
4:    $value \leftarrow buffer[oldest]$
5:    $oldest \leftarrow (oldest + 1)\%N$
6:    $count - -$
7: end if
8: if count == 0 then
9:    $notEmpty \leftarrow false$
10: end if
11: $notFull \leftarrow true$
11:    finally
11:       bufferLock.unlock()
11:    end try
12: return value
```

Solutia utilizand semafoare

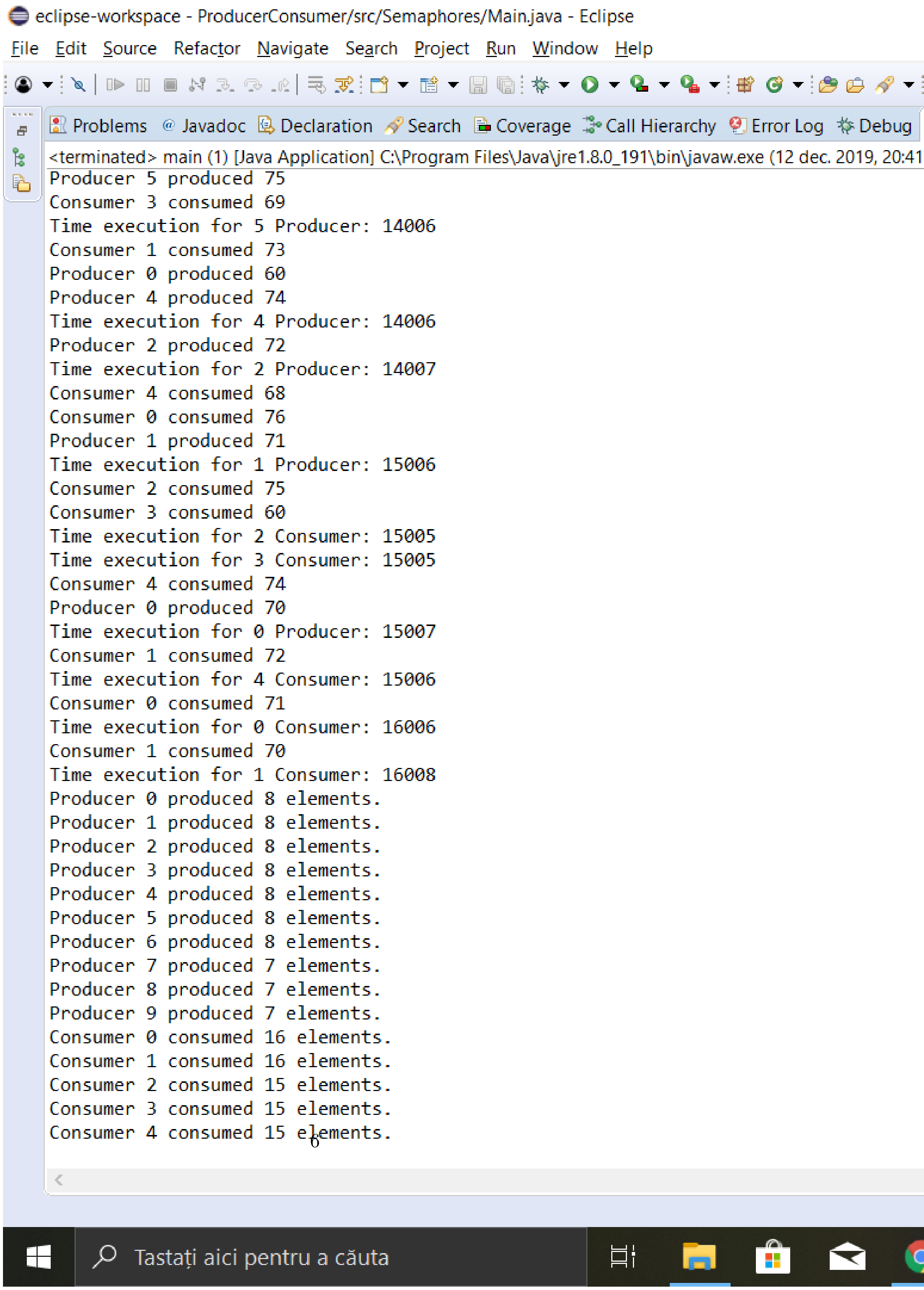
Am rezolvat problema utilizand biblioteca Semaphore din java pentru implementarea semafoarelor.In clasa Market am creat 4 semafoare: 2 semafoare pentru producator si consumator, producerSemaphore si consumerSemaphore, si 2 semafoare notFull si notEmpty pentru a verifica daca bufferul este plin sau gol. producerSemaphore si notFull primesc initial permisiunea 1, deoarece la inceput producatorul se apeleaza prima data, iar bufferul nu este plin.

In functiile produce si consume mai intai se incearca intr-un bloc try catch obtinerea permisiunii de la semafoare. In cazul functiei produse, se cere permisunea de la producerSemaphore si notFull. Daca se primesc permisiunile se trece la adugarea elementului in buffer, in caz contrat se genereaza o exceptie. La final se verifica daca bufferul este plin sau gol si se incrementeaza semafoarele. In mod asemanator este construita si functia consume(). Clasele Producer, Consumer si Main au ramas neschimbate.

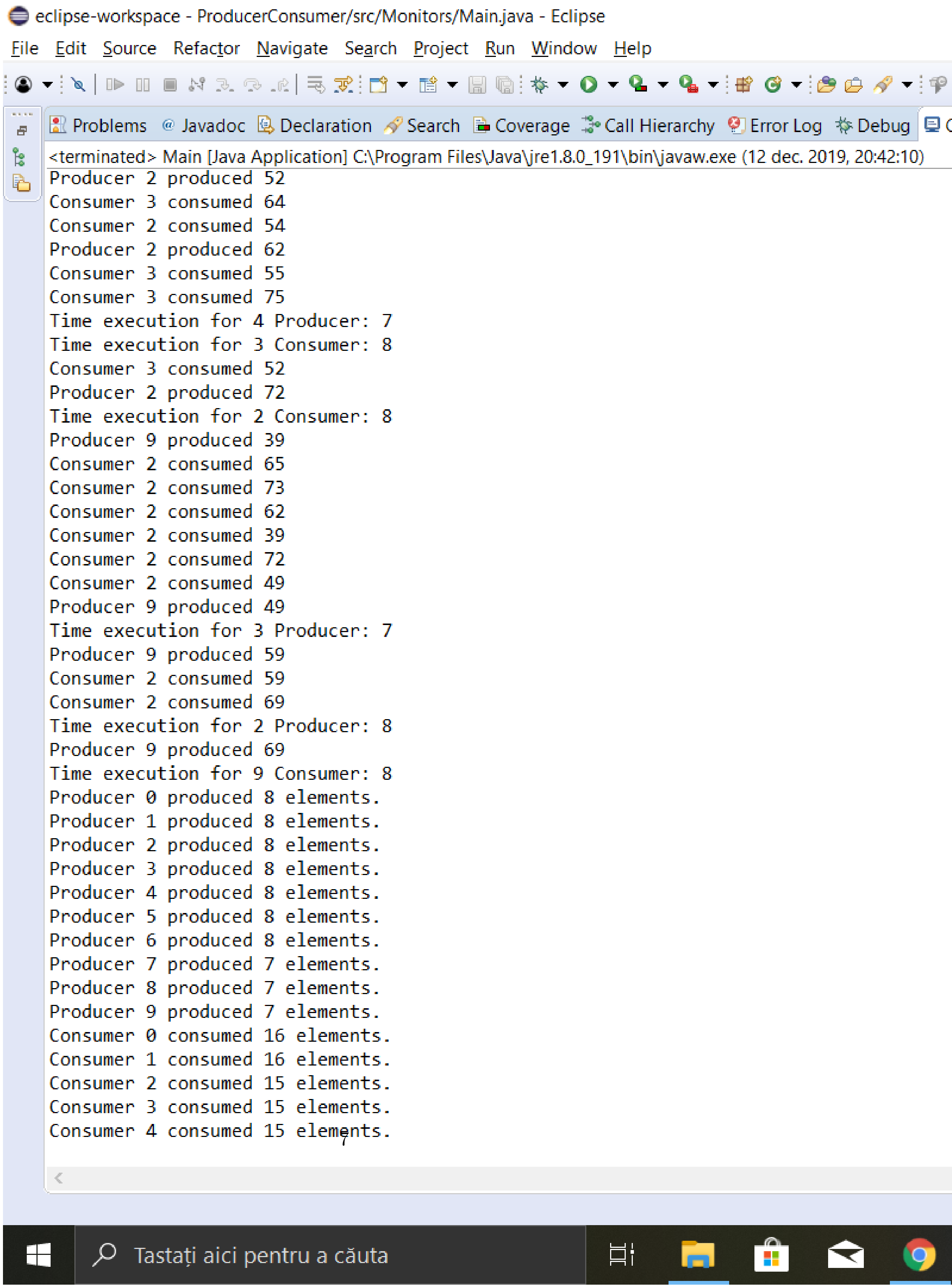
3 Experimental data

In rezolvarea problemei utilizand monitoare am folosit, initial, in loc de functia wait(), apelul Thread.currentThread().sleep(6). Insa, in timpul executiei, s-a intamplat fenomenul deadlock, adica 2 threaduri asteptau la infinit unul dupa celalalt, niciunul neputand sa-si continue executia. Aceasta problema a fost rezolvata prin folosirea functiei wait(), care elibereaza blocajul pe un thread, astfel celelalte threaduri care asteapta dupa el putand sa-si continue executia.

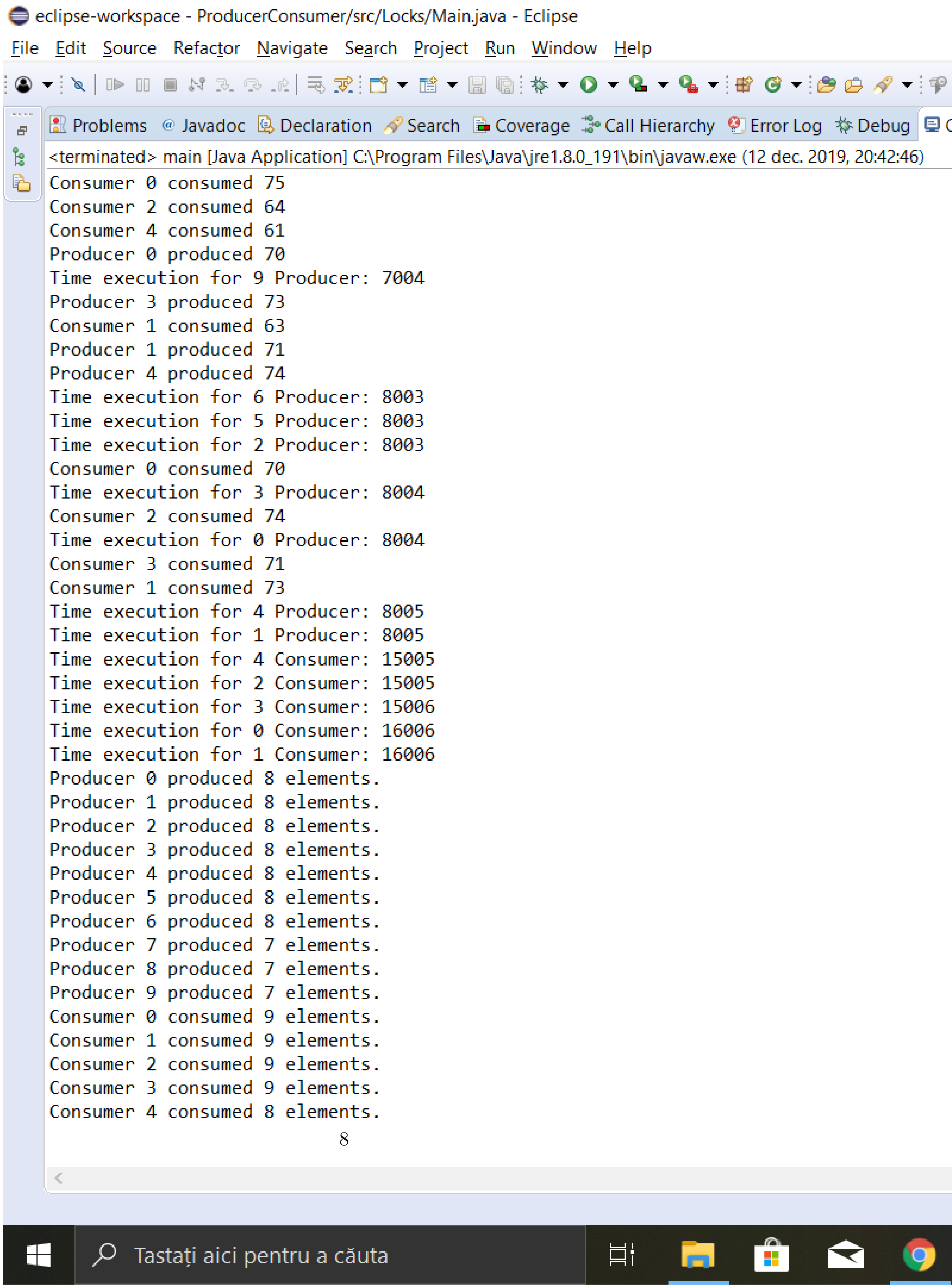
Pentru fiecare mod de rezolvare al problemei am calculat timpul de executie pentru fiecare thread producator si consumator.Timpul mediu cel mai scurt in care toate threadurile si-au incheiat executia a fost obtinut in cazul rezolvarii utilizand monitoarele: 6 milisecunde. Pentru celelalte 2 metode am obtinut urmatorii timpi medii de executie: 10290 milisecunde pentru implementarea utilizand zavoare si 14166 pentru cea cu semafoare.


eclipse-workspace - ProducerConsumer/src/Semaphores/Main.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Coverage Call Hierarchy Error Log Debug
Tastați aici pentru a căuta

```
<terminated> main (1) [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (12 dec. 2019, 20:41
Producer 5 produced 75
Consumer 3 consumed 69
Time execution for 5 Producer: 14006
Consumer 1 consumed 73
Producer 0 produced 60
Producer 4 produced 74
Time execution for 4 Producer: 14006
Producer 2 produced 72
Time execution for 2 Producer: 14007
Consumer 4 consumed 68
Consumer 0 consumed 76
Producer 1 produced 71
Time execution for 1 Producer: 15006
Consumer 2 consumed 75
Consumer 3 consumed 60
Time execution for 2 Consumer: 15005
Time execution for 3 Consumer: 15005
Consumer 4 consumed 74
Producer 0 produced 70
Time execution for 0 Producer: 15007
Consumer 1 consumed 72
Time execution for 4 Consumer: 15006
Consumer 0 consumed 71
Time execution for 0 Consumer: 16006
Consumer 1 consumed 70
Time execution for 1 Consumer: 16008
Producer 0 produced 8 elements.
Producer 1 produced 8 elements.
Producer 2 produced 8 elements.
Producer 3 produced 8 elements.
Producer 4 produced 8 elements.
Producer 5 produced 8 elements.
Producer 6 produced 8 elements.
Producer 7 produced 7 elements.
Producer 8 produced 7 elements.
Producer 9 produced 7 elements.
Consumer 0 consumed 16 elements.
Consumer 1 consumed 16 elements.
Consumer 2 consumed 15 elements.
Consumer 3 consumed 15 elements.
Consumer 4 consumed 15 elements.
```


eclipse-workspace - ProducerConsumer/src/Monitors/Main.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Coverage Call Hierarchy Error Log Debug
Tastați aici pentru a căuta

```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (12 dec. 2019, 20:42:10)
Producer 2 produced 52
Consumer 3 consumed 64
Consumer 2 consumed 54
Producer 2 produced 62
Consumer 3 consumed 55
Consumer 3 consumed 75
Time execution for 4 Producer: 7
Time execution for 3 Consumer: 8
Consumer 3 consumed 52
Producer 2 produced 72
Time execution for 2 Consumer: 8
Producer 9 produced 39
Consumer 2 consumed 65
Consumer 2 consumed 73
Consumer 2 consumed 62
Consumer 2 consumed 39
Consumer 2 consumed 72
Consumer 2 consumed 49
Producer 9 produced 49
Time execution for 3 Producer: 7
Producer 9 produced 59
Consumer 2 consumed 59
Consumer 2 consumed 69
Time execution for 2 Producer: 8
Producer 9 produced 69
Time execution for 9 Consumer: 8
Producer 0 produced 8 elements.
Producer 1 produced 8 elements.
Producer 2 produced 8 elements.
Producer 3 produced 8 elements.
Producer 4 produced 8 elements.
Producer 5 produced 8 elements.
Producer 6 produced 8 elements.
Producer 7 produced 7 elements.
Producer 8 produced 7 elements.
Producer 9 produced 7 elements.
Consumer 0 consumed 16 elements.
Consumer 1 consumed 16 elements.
Consumer 2 consumed 15 elements.
Consumer 3 consumed 15 elements.
Consumer 4 consumed 15 elements.
```


eclipse-workspace - ProducerConsumer/src/Locks/Main.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Problems
Javadoc
Declaration
Search
Coverage
Call Hierarchy
Error Log
Debug
Tastați aici pentru a căuta

```
<terminated> main [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (12 dec. 2019, 20:42:46)
Consumer 0 consumed 75
Consumer 2 consumed 64
Consumer 4 consumed 61
Producer 0 produced 70
Time execution for 9 Producer: 7004
Producer 3 produced 73
Consumer 1 consumed 63
Producer 1 produced 71
Producer 4 produced 74
Time execution for 6 Producer: 8003
Time execution for 5 Producer: 8003
Time execution for 2 Producer: 8003
Consumer 0 consumed 70
Time execution for 3 Producer: 8004
Consumer 2 consumed 74
Time execution for 0 Producer: 8004
Consumer 3 consumed 71
Consumer 1 consumed 73
Time execution for 4 Producer: 8005
Time execution for 1 Producer: 8005
Time execution for 4 Consumer: 15005
Time execution for 2 Consumer: 15005
Time execution for 3 Consumer: 15006
Time execution for 0 Consumer: 16006
Time execution for 1 Consumer: 16006
Producer 0 produced 8 elements.
Producer 1 produced 8 elements.
Producer 2 produced 8 elements.
Producer 3 produced 8 elements.
Producer 4 produced 8 elements.
Producer 5 produced 8 elements.
Producer 6 produced 8 elements.
Producer 7 produced 7 elements.
Producer 8 produced 7 elements.
Producer 9 produced 7 elements.
Consumer 0 consumed 9 elements.
Consumer 1 consumed 9 elements.
Consumer 2 consumed 9 elements.
Consumer 3 consumed 9 elements.
Consumer 4 consumed 8 elements.
```

4 Results & Conclusions

Cea mai eficienta metoda de rezolvare a problemei Producator-Consumator este cea utilizand monitoarele, prin folosirea mecanismului synchronized pentru functiile produce si consume. Rezolvarea folosind zavoare este asemanatoare cu cea cu monitoare, diferenta consta in faptul ca fiecare thread se blocheaza si se deblocheaza prin apelul functiilor lock() si unlock(), lucru care la monitoare se face automat in timpul executiei. Cea de-a treia metoda de rezolvare, cea in care se utilizeaza semafoare este apropiata ca timp de executie de cea cu zavoare, dar are cel mai mare timp de executie.