

Tema 1 - Marketplace



- **Deadline:** 4 aprilie 2021, ora 23:55. Primiți un bonus de 10% pentru trimiterea temei cu 3 zile înainte acestui termen, adică înainte de 1 aprilie 2021, ora 23:55.
- **Deadline hard:** 11 aprilie 2021, ora 23:55. Veți primi o depunțare de 10% din punctajul maxim al temei pentru fiecare zi de întârziere, până la maxim 7 zile, adică până pe 11 aprilie 2021, ora 23:55.
- **Responsabili:** Tudor Antonio Barbu, Voichița Iancu, Eduard Stăniloiu, Loredana Soare, Giorgiana Vlăsceanu
- **Autori:** Luca Istrate, Adriana Draghici, Loredana Soare



- Dată publicare: 21 martie
- Dată actualizare enunț: 24 martie

Scopul temei

- Utilizarea eficientă a elementelor de sincronizare studiate la laborator
- Implementarea unei aplicații concurente utilizând o problemă clasică (Multi Producer, Multi Consumer)
- Aprofundarea anumitor elemente din Python (clase, elemente de sintaxă, thread-uri, sincronizare, precum și folosirea modulelor Python pentru lucrul cu thread-uri)

Enunț

În cadrul acestei teme veți avea de implementat un Marketplace prin intermediul căruia mai mulți **producători** își vor oferi produsele spre vânzare, iar mai mulți **cumpărători** vor achiziționa produsele puse la dispoziție.

Marketplace

Marketplace-ul este unul destul de simplu, cu **două tipuri de produse (ceai și cafea)** ce vor fi comercializate de către producători. Acesta va fi intermediarul dintre producători și consumatori, prin el realizându-se achiziția de produse: producătorul (producer) va produce o anumită cantitate de produse de un anumit tip / mai multe tipuri cumpărătorul (consumer) va cumpăra o anumită cantitate de produse de un tip / de mai multe tipuri. De asemenea, Marketplace-ul va pune la dispoziția fiecărui cumpărător câte un **coș de produse (cart)** (acesta va fi folosit pentru rezervarea produselor care se doresc a fi cumpărate).

Producător

Vor exista mai mulți producători ce vor produce obiectele de tip cafea / ceai. Fiecare produs va fi furnizat într-o anumită cantitate. Un producător poate produce atât obiecte de tip cafea, cât și de tip ceai.

Consumator

În momentul în care un client își dorește să cumpere anumite produse dintr-un magazin, acesta va avea nevoie de un coș de cumpărături pe care să îl folosească în scopul rezervării acestora. Astfel, de fiecare dată când un client își începe cumpărăturile, acesta va primi din partea Marketplace-ului un coș de cumpărături, căruia îi va fi asociat un *id*. Clientul poate:

- adăuga produse în coș ⇒ produsele respective devin indisponibile pentru ceilalți clienți
- șterge produse din coș ⇒ produsele respective devin disponibile pentru ceilalți clienți
- plasa o comandă

Descrierea implementării

Marketplace-ul ce va trebui implementat va simula problema **Multi Producer Multi Consumer (MPMC)**. Pentru rezolvarea acestei teme va trebui să completați clasele *MarketPlace*, *Producer*, și *Consumer* cu o implementare corectă a metodelor deja definite.

Rezolvarea temei va fi concentrată preponderent pe metodele clasei *MarketPlace*, metode ce vor fi apelate atât de producător, cât și de cumpărător în clasele aferente ale acestora.

Operația efectuată de către producător este cea de *publicare a produselor sale*. Implementarea metodei *publish* va fi făcută în clasa *MarketPlace*.

Vor exista doua tipuri de operații pe care clientul le poate efectua asupra coșului de cumpărături:

- *add_to_cart* ⇒ adaugă un produs în coș
- *remove_from_cart* ⇒ șterge un produs din coș

Ambele metode (*add_to_cart* și *remove_from_cart*) vor trebui implementate în clasa *MarketPlace*.

În momentul în care un consumator adaugă un produs în coșul pentru cumpărături, produsul respectiv va deveni indisponibil pentru ceilalți clienți ai Marketplace-ului. Clientul își va putea plasa comanda prin apelarea metodei *place_order* (din clasa *MarketPlace*). În cazul în care un produs este eliminat din coșul pentru cumpărături, acesta devine disponibil pentru ceilalți clienți ai Marketplace-ului.

Funcționalitatea clasei *Producer* este să:

- furnizeze produselor pe care producătorul le pune la dispoziție



Producer produce secvențial numărul de produse și tipul din cadrul fișierului de intrare și așteaptă după realizarea fiecărui produs un număr de secunde specificat. Informațiile se preiau din fișierul de intrare și are următorul format pentru produse["id", cantitate, timp-așteptare].

Funcționalitatea clasei *Consumer* este să:

- primească id-ului coșului de cumpărături
- adauge / elimine din coșul de cumpărături anumite cantități de produse
- plaseze comenzi

Modulul **Product** conține reprezentările claselor **Coffee** și **Tea**.

Marketplace-ul limitează numărul de produse ce pot fi publicate de către un producător. În momentul în care s-a atins limita, producătorul nu mai poate publica altele până nu sunt cumpărate. El va reîncerca să publice după un timp definit în fișierul de test.

Dacă un cumpărător nu găsește un produs în marketplace, el va încerca mai târziu, după un timp definit în fișierul de test.

Se consideră timp de așteptare după:



- adăugarea unui produs
- semnalizarea că nu se găsește un produs
- semnalizarea faptului că este plină coada asociată producătorului

Formatul Testelor

Testarea se va face cu ajutorul a două tipuri de fișiere, cele de input și cele de output ({id}.in și {id}.out), primul fiind în format JSON. Fișierul **{id}.in** va reprezenta fișierul de intrare și va conține configurațiile necesare pentru fiecare clasă în parte, iar fișierul **{id}.out** va reprezenta fișierul de ieșire prin intermediul căruia se va verifica corectitudinea implementării temei.

Fișierele de input vor fi fișiere JSON ce vor conține următoarele chei:

- marketplace
- products
- producers
- consumers

Exemplu conținut fișier de intrare și fișierul corespunzător de ieșire:

Click pentru exemplu ↗



Atât conținutul fișierului de intrare, cât și conținutul fișierului de ieșire sunt descrise în README

Pentru a putea compara fișierele de ieșire obținute de voi cu cele de referință, scriptul de testare va ordona output-ul rezultat, întrucât avem de-a face cu multithreading.

Precizări încărcare / VMChecker

Arhiva temei va fi încărcată pe vmchecker.

Arhiva trebuie să conțină:

- director tema cu fișierele temei: marketplace.py, producer.py, consumer.py
- alte fișiere .py folosite în dezvoltare
- README
- director .git



Pentru a documenta realizarea temei, vă recomandăm să folosiți template-ul de aici

Punțare



Tema va fi verificată automat, folosind infrastructura de testare, pe baza unor teste definite în directorul tests.

Tema se va implementa **Python>=3.7**.

Notarea va consta în 100 pct acordate egale între teste. Depunțări posibile sunt:

- folosirea incorectă a variabilelor de sincronizare (ex: lock care nu protejează toate accesele la o variabilă partajată, notificări care se pot pierde) (-2 pct)
- prezența print-urilor de debug (maxim -10 pct în funcție de gravitate)
- folosirea lock-urilor globale (-10 pct)
- folosirea variabilelor globale/statice (-5 pct)
 - Variabilele statice pot fi folosite doar pentru constante
- folosirea inutilă a variabilelor de sincronizare (ex: se protejează operații care sunt deja thread-safe) (-5 pct)
- alte ineficiențe (ex: creare obiecte inutile, alocare obiecte mai mari decât e necesar, etc.) (-5 pct)
- lipsa organizării codului, implementare încălțită și nemodulară, cod duplicat, funcții foarte lungi (între -1pct și -5 pct în funcție de gravitate)
- cod înghesuit/ilizibil, inconsistența stilului - vedeți secțiunea Pylint
 - pentru code-style recomandăm ghidul oficial PEP-8
- cod comentat/nefolosit (-1 pct)
- lipsa comentariilor utile din cod (-5 pct)
- fișier README sumar (până la -5 pct)
- nerespectarea formatului .zip al arhivei (-2 pct)
- alte situații nespecificate, dar considerate inadecvate având în vedere obiectivele temei; în special situațiile de modificare a interfeței oferite

Se acordă bonus 5 pct pentru adăugarea directorului .git și utilizarea versionării în cadrul repository-ului.



Temele vor fi testate împotriva plagiatului. Orice tentativă de copiere va fi depunțată conform regulamentului.

Pylint

Vom testa sursele voastre cu pylint configurat conform fișierului *pylintrc* din cadrul repo-ului dedicat temei. Atenție, *nu!lăm pylint doar ne modulele completate și adăugate de voi*, nu și pe cele ale testerului.

Deoarece apar diferențe de scor între versiuni diferite de pylint, vom testa temele doar cu ultima versiune. Vă recomandăm să o folosiți și voi tot pe aceasta.

Vom face depunțări de până la -5pct dacă verificarea făcută cu pylint vă dă un scor mai mic de 8.

Observații

- Pot exista depunțări mai mari decât este specificat în secțiunea *Notare* pentru implementări care nu respectă obiectivele temei și pentru situații care nu sunt acoperite în mod automat de către sistemul de testare
- Implementarea și folosirea metodelor oferite în schelet este obligatorie
- Puteți adăuga variabile/metode/clase, însă nu puteți schimba antetul metodelor oferite în schelet
- Bug-urile de sincronizare, prin natura lor sunt nedeterminate; o temă care conține astfel de bug-uri poate obține punctaje diferite la rulări succesive; în acest caz punctajul temei va fi cel dat de tester în momentul corectării
- Recomandăm testarea temei în cât mai multe situații de load al sistemului și pe cât mai multe sisteme pentru a descoperi bug-urile de sincronizare

Resurse necesare realizării temei

Pentru a clona repo-ul și a accesa resursele temei 1:

```
student@asc:~$ git clone https://bitbucket.org/ASC-admin/asc.git
student@asc:~$ cd asc/assignments
student@asc:~/assignments$ cd 1-marketplace
```

Suport, întrebări și clarificări

Pentru întrebări sau nelămuriri legate de temă folosiți forumul temei.



Orice întrebare e recomandat să conțină o descriere cât mai clară a eventualei probleme. Întrebări de forma: "Nu merge X. De ce?" fără o descriere mai amănunțită vor primi un răspuns mai greu.

ATENȚIE să nu postați imagini cu părți din soluția voastră pe forumul pus la dispoziție sau orice alt canal public de comunicație. Dacă veți face acest lucru, vă asumăm răspunderea dacă veți primi copiat pe temă.

Search

Navigare

- Anunțuri
- Regulament
- Echipă
- Orar

Laboratoare

- Laboratorul 01 - Introducere în limbajul Python
- Laboratorul 02 - Fire de execuție în Python
- Laboratorul 03 - Programare concurentă în Python (continuare)
- Laboratorul 04 - Arhitecturi de Microprocesoare și Sisteme de Calcul
- Laboratorul 05 - Tehnici de Optimizare de Cod - Înmultirea Matricelor
- Laboratorul 06 - Analiza Performanței Programelor
- Laboratorul 07 - Arhitecturi de tip GPGPU
- Laboratorul 08 - Arhitectura GPU NVIDIA CUDA
- Laboratorul 09 - Advanced CUDA
- Exerciții din alți ani

Teme

- Tema 1
- Tema 2
- Tema 3

Resurse

- Cluster cheat-sheet
- Tutorial video rulare task-uri cluster
- Tutorial video rulare programe MPI pe cluster

GPU related

- CUDA C Programming
- CUDA NVCC compiler
- Visual Profiler
- CUDA 9.1 Toolkit
- NVIDIA Tesla K40M
- NVIDIA Tesla C2070
- Nvidia Tesla 2050/2070
- Nvidia CUDA Fermi/Tesla

Lecture related

- Comutatoare
- Taxonomia Flynn
- Single Board Computers
- Explicitly Parallel Instruction Computing
- Intel Parallel Studio

Utilitare

- Dinero cache simulator
- Python Visual Interpreter

Older Labs & Resources

- Cell - Rulare pe CLUSTER
- GDB on Cell BE
- Mailbox Hands-On
- Arhitectura Cell BE
- Kickstart Cell BE
- DMA 101
- Reference Manuals
- Folosirea simulatorului
- Branch Prediction
- Cell Profiler
- Tutorial Cell - Eclipse
- Software Managed Cache
- Liste DMA
- Continut

Table of Contents

- Tema 1 - Marketplace
 - Scopul temei
 - Enunț
 - Marketplace
 - Producător
 - Consumator
 - Descrierea implementării
 - Formatul Testelor
 - Precizări încărcare / VMChecker
 - Punțare
 - Pylint
 - Observații
 - Resurse necesare realizării temei
 - Suport, întrebări și clarificări