ALGORITMI PARALELI ȘI DISTRIBUIȚI
Tema #1 Procesare paralelă folosind paradigma Map-Reduce

Responsabili: Radu-Ioan Ciobanu, Dorinel Filip

Termen de predare: 20-11-2022 23:59 (soft), 27-11-2022 23:59 (hard)
Ultima modificare: 28-10-2022 21:55

Cuprins

Cerință **2**

Paradigma Map-Reduce **2**

Detalii tehnice **2**
Operațiile de tip Map . . . 2
Operațiile de tip Reduce . . . 3

Execuție **3**
Exemplu . . . 4

Notare **7**
Bonus . . . 8

Testare **8**
Docker . . . 9

Recomandări de implementare și testare **9**

Link-uri utile **12**

Cerință

Să se implementeze un program paralel în Pthreads pentru găsirea numerelor mai mari decât 0 care sunt puteri perfecte dintr-un set de fișiere și numărarea valorilor unice pentru fiecare exponent.

Pentru paralelizarea procesării documentelor de intrare, se va folosi modelul Map-Reduce. Fișierele de intrare vor fi împărțite (în mod dinamic) cât mai echilibrat la niște thread-uri care le vor parsa și vor verifica ce numere mai mari decât 0 sunt puteri perfecte (operațiunea de **Map**), rezultând astfel liste parțiale pentru fiecare exponent (ex., liste pentru pătratele perfecte, liste pentru cuburile perfecte, etc.). Pasul următor îl reprezintă combinarea listelor parțiale (operațiunea de **Reduce**) în urma căreia se vor obține liste agregate pentru fiecare exponent în parte. Pentru fiecare astfel de listă, se vor număra în paralel valorile unice, rezultatele fiind apoi scrise în niște fișiere de ieșire.

Paradigma Map-Reduce

Pentru rezolvarea temei, se va folosi un model Map-Reduce similar cu cel folosit la Google pentru procesarea unor seturi mari de documente în sisteme distribuite. Acest articol prezintă modelul Map-Reduce folosit de Google și o parte dintre aplicațiile lui (mai importante pentru înțelegerea modelului sunt primele 4 pagini).

Map-Reduce este un model (și o implementare asociată) de programare paralelă pentru procesarea unor seturi imense de date, folosind sute sau mii de procesoare. În majoritatea cazurilor, Map-Reduce este folosit într-un context distribuit, fiind, de fapt, un model de programare care poate fi adaptat pentru ambele situații. Cea mai cunoscută implementare este Apache Hadoop, dezvoltat inițial de către Doug Cutting și Mike Cafarella. Modelul permite paralelizarea și distribuirea automată a task-urilor. Paradigma Map-Reduce se bazează pe existența a două funcții care îi dau și numele: Map și Reduce. Funcția Map primește ca input o funcție f și o listă de elemente, și returnează o nouă listă de elemente rezultată în urma aplicării funcției f asupra fiecărui element din lista inițială. Funcția Reduce combină rezultatele obținute anterior.

Mecanismul Map-Reduce funcționează în modul următor:

- utilizatorul cere procesarea unui set de documente
- această cerere este adresată unui proces (sau fir de execuție) coordonator
- coordonatorul asignează documentele unor procese (sau fire de execuție) de tip Mapper[1]
- un Mapper va analiza fișierele de care este responsabil și va genera niște rezultate parțiale, având în general forma unor perechi de tip $\{cheie, valoare\}$
- după ce operațiile Map au fost executate, alte procese (sau fire de execuție) de tip Reducer combină rezultatele parțiale și generează soluția finală.

Detalii tehnice

Dându-se un set de N documente, să se numere valorile unice mai mari decât 0 de tip putere perfectă pentru fiecare exponent E folosind Map-Reduce. În implementarea temei, vor exista thread-uri care pot fi de două tipuri, Mapper sau Reducer, toate fiind pornite împreună la începutul rulării.

Operațiile de tip Map

Pornind de la lista de documente de procesat ce va fi disponibilă în fișierul de intrare, fiecare Mapper va ajunge să proceseze niște documente. Alocarea de documente thread-urilor Mapper poate fi realizată static înainte de pornirea acestora (deși nu este recomandat acest lucru), sau poate fi realizată în mod dinamic pe

[1] Acest lucru nu trebuie neapărat realizat la început, el se poate face și în mod dinamic pe măsură ce fișierele sunt procesate.

măsură ce documentele sunt procesate (într-o astfel de situație, un Mapper care se mișcă mai repede decât ceilalți poate să "fure" muncă de la alt Mapper, lucru care va duce la o procesare mai eficientă). Fiecare Mapper va executa următoarele acțiuni, **pentru fiecare fișier de care este reponsabil**:

- deschide fișierul și îl parcurge linie cu linie (pe fiecare linie fiind câte o valoare numerică de tip întreg)
- pentru fiecare întreg citit, verifică dacă este o putere perfectă a lui 2, 3, 4, etc. mai mare decât 0 (exponentul maxim până la care verifică este dat de numărul de thread-uri Reducer, așa cum se va explica mai târziu)
- fiecare valoare mai mare decât 0 care este o putere perfectă cu un exponent E este salvată într-o listă parțială (dacă o valoare este putere perfectă pentru mai mulți exponenți E - de exemplu, 81 poate fi scris ca 3^4 sau 9^2 - aceasta va fi plasată în mai multe liste parțiale)
- când s-a terminat de procesat un fișier, Mapper-ul îl închide.

Atenție! Pentru verificarea dacă un număr este putere perfectă, NU folosiți funcția *pow* (sau orice funcție similară de ridicat la putere) **cu exponent subunitar**. Din cauza operațiilor în virgulă mobilă, este posibil să ratați anumite valori sau să luați în considerare valori incorecte.

Operațiile de tip Reduce

Un fir de execuție de tip Reducer va fi responsabil pentru agregarea și numărarea valorilor puteri perfecte pentru un singur exponent (de exemplu, un Reducer se va ocupa de pătratele perfecte, altul de cuburile perfecte, etc.). Astfel, având rezultatele din cadrul operațiunii de Map, un Reducer va realiza următoarele acțiuni:

- combină listele parțiale pentru exponentul E de care este responsabil într-o listă agregată (etapa de combinare)
- numără valorile unice din lista agregată și scrie rezultatul într-un fișier (etapa de procesare).

Atenție! Înainte să începeți operațiile de tip Reduce, trebuie să vă asigurați că toate operațiile de tip Map s-au finalizat.

Execuție

Programul se va rula în felul următor:

```
./tema1 <numar_mapperi> <numar_reduceri> <fisier_intrare>
```

Atenție! Trebuie să porniți toate thread-urile (atât cele Mapper, cât și cele Reducer) într-o singură iterație a thread-ului principal. Nu se acceptă mai multe perechi de *pthread_create*/*pthread_join* în cod, acest lucru ducând la un punctaj de **0** pe întreaga temă.

Fișierul de intrare are următorul format:

```
numar_fisiere_de_procesat
fisier1
...
fisierN
```

Așadar, prima linie conține numărul de documente text de procesat, iar următoarele linii conțin numele documentelor, câte unul pe linie. Toate fișierele de intrare vor conține **doar** caractere ASCII și se pot considera valide.

Un fișier care trebuie procesat are următorul format:

```
numar_valori_de_verificat
valoare1
...
valoareN
```

Aceste fișiere trebuie împărțite cât mai echilibrat thread-urilor Mapper, fără să existe fișiere neprocesate de niciun Mapper sau fișiere procesate de mai multe thread-uri Mapper.

Atenție! O împărțire echilibrată a fișierelor de procesat nu înseamnă neapărat că fiecare thread Mapper va avea câte un număr aproximativ egal de fișiere, deoarece acestea pot avea dimensiuni foarte diferite. De exemplu, dacă există trei thread-uri Mapper care au de procesat 6 fișiere de dimensiuni 5 MB, 5 MB, 1 MB, 1 MB, 1 MB, 1 MB, este mai eficient ca primele două thread-uri Mapper să proceseze câte un fișier de 5 MB și al treilea Mapper să proceseze cele patru fișiere de câte 1 MB, decât să se aloce câte două fișiere fiecărui Mapper. Acest lucru se poate face static (înaintea de pornirea thread-urilor Mapper) sau dinamic (fiecare Mapper care termină de procesat un fișier vede dacă mai sunt fișiere disponibile). O împărțire ineficientă poate afecta scalabilitatea temei și implicit punctajul obținut.

În funcție de numărul de thread-uri Reducer, se va calcula și exponentul maxim E până la care se verifică puterile perfecte mai mari decât 0. De exemplu, dacă se rulează cu trei thread-uri Reducer, E va fi 4 (se verifică pătratele perfecte, cuburile perfecte, și numerele care sunt puteri perfecte cu exponentul 4). Programul va avea câte un fișier de ieșire pentru fiecare Reducer, care se va numi $out E.txt$, unde E este exponentul de care este responsabil fiecare Reducer. Un fișier de ieșire va conține o singură valoare de tip întreg, adică numărul de valori unice mai mari decât 0 care sunt puteri perfecte cu exponentul E.

Exemplu

Se dă următorul fișier de intrare:

```
$ cat test.txt
4
in1.txt
in2.txt
in3.txt
in4.txt
```

Cele patru documente de procesat arată în felul următor:

```
$ cat in1.txt
6
243
9
27
243
81
243

$ cat in2.txt
6
81
```

```
9
27
243
27
27

$ cat in3.txt
6
9
27
9
81
9
53

$ cat in4.txt
5
243
243
243
1
0
```

Tema se rulează cu următoarea comandă:

```
./tema1 3 5 test.txt
```

Așadar, există trei thread-uri Mapper (la care se împart cele patru fișiere) și cinci thread-uri Reducer (deci se vor verifica puterile perfecte pentru exponenții 2, 3, 4, 5 și 6).

Cele patru fișiere de intrare se pot împărți la cele trei thread-uri Mapper astfel:

- M0 → *in1.txt*
- M1 → *in2.txt*
- M2 → *in3.txt*, *in4.txt*

Conform operației de Map, fiecare Mapper verifică toate numerele din fișierele asignate lui și salvează liste parțiale cu puterile perfecte mai mari decât 0. Pentru exemplul de mai sus, operațiile de Map duc la următoarele rezultate (unde prima listă de la fiecare Mapper reprezintă pătratele perfecte, a doua listă cuburile perfecte, etc.):

- M0 → {9, 81}, {27}, {81}, {243, 243, 243}, {}
- M1 → {81, 9}, {27, 27, 27}, {81}, {243}, {}
- M2 → {9, 9, 81, 9, 1}, {27, 1}, {81, 1}, {243, 243, 243, 1}, {1}

Fiecare Mapper are câte cinci liste parțiale, una pentru fiecare exponent. Se poate observa că numerele care sunt puteri perfecte pentru mai mulți exponenți E apar în mai multe liste parțiale de la același Mapper. De asemenea, se poate observa și faptul că 1 este considerat putere perfectă pentru orice exponent, dar 0 nu se ia în considerare (pentru că se cer valorile perfecte mai mari decât 0).

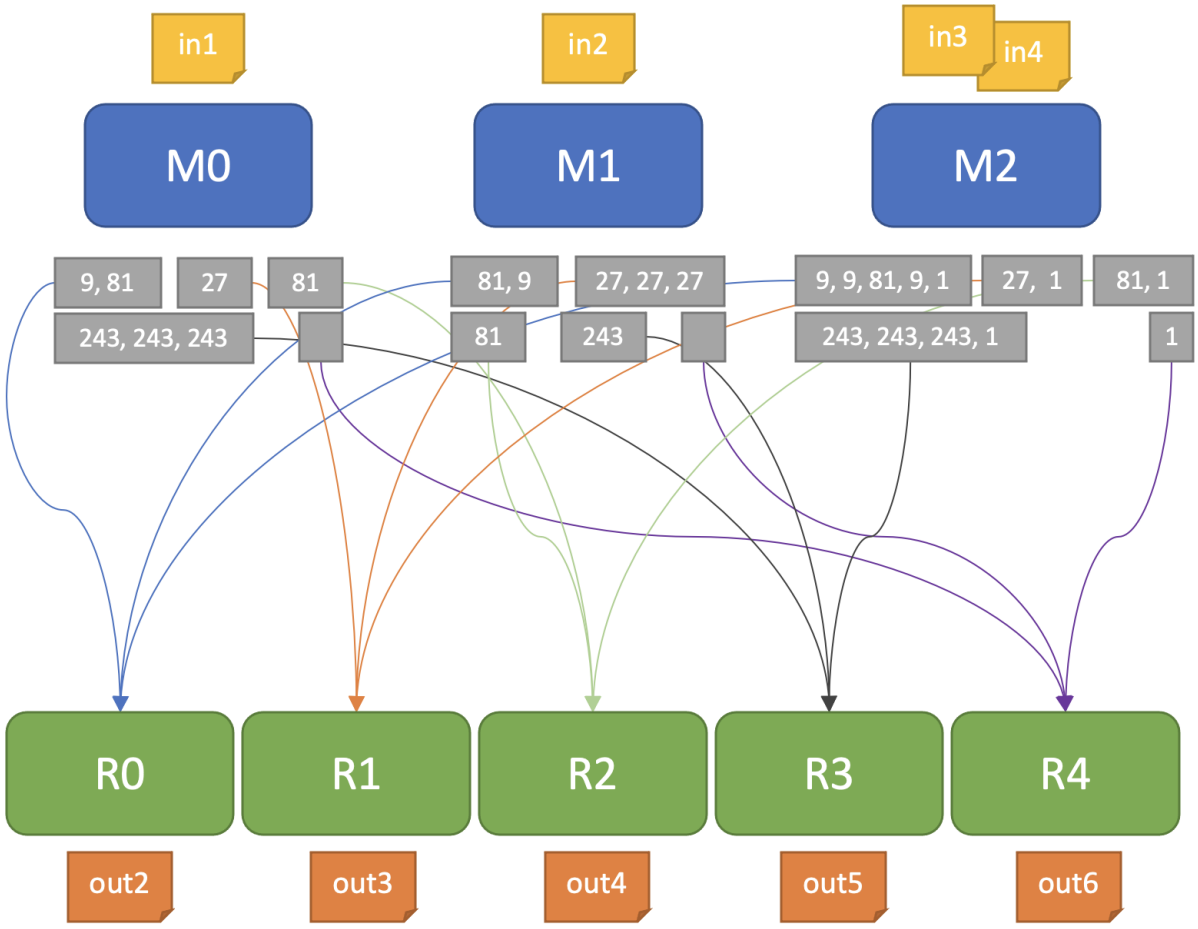
Odată ce toate operațiile de Map s-au terminat, thread-urile Reducer se apucă de combinare și procesare. Conform rulării, în acest exemplu sunt cinci astfel de thread-uri, fiecare din ele responsabil de un exponent. Astfel, fiecare Reducer ia toate listele parțiale pentru exponentul său (care vor fi în număr de trei, câte una per Mapper) și le agregă, ajungând la următorul set de liste agregate:

- R0 → {9, 81, 81, 9, 9, 9, 81, 9, 1}
- R1 → {27, 27, 27, 27, 27, 1}
- R2 → {81, 81, 81, 1}
- R3 → {243, 243, 243, 243, 243, 243, 243, 1}
- R4 → {1}

În final, fiecare Reducer numără valorile unice din lista sa finală, ajungând la următorul rezultat:

```
$ cat out2.txt
3

$ cat out3.txt
2

$ cat out4.txt
2

$ cat out5.txt
2

$ cat out6.txt
1
```

O reprezentare grafică a rulării de Map-Reduce pe exemplul precedent se poate observa în figura de mai jos.



Notare

În primă fază, tema se va testa local, după cum se explică mai jos. Atunci când va fi disponibil și checker-ul automat, vom oferi detalii. Tema se va încărca de asemenea și pe Moodle. Se va încărca o arhivă Zip care, pe lângă fișierele sursă C/C++, va trebui să conțină următoarele două (sau trei) fișiere **în rădăcina arhivei**:

- *Makefile* - cu directiva *build* care compilează tema voastră (**fără flag-uri de optimizare**) și generează un executabil numit *tema1* aflat în rădăcina arhivei, și directiva *clean* care șterge executabilul
- *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 (așa cum se detaliază mai jos).

Punctajul este divizat după cum urmează:

- **70p** - scalabilitatea soluției
- **50p** - corectitudinea rezultatelor (la rulări multiple pe aceleași date de intrare, trebuie obținute aceleași rezultate)[2]
- **30p** - claritatea codului și a explicațiilor din README.

Nerespectarea următoarelor cerințe va duce la depunctări:

- **-150p** - ne-urmărirea paradigmei Map-Reduce conform descrierii din enunț
- **-150p** - pseudo-sincronizarea firelor de execuție prin funcții cum ar fi `sleep`
- **-150p** - utilizarea altor implementări de thread-uri în afară de Pthreads (cum ar fi *std::thread* din C++11)
- **-150p** - crearea și oprirea de thread-uri în mod repetat (pentru a nu lua această depunctare, trebuie să creați un singur set de cel mult $M + R$ thread-uri la început, unde M e numărul de thread-uri Mapper, iar R numărul de thread-uri Reducer, așa cum sunt date în argumentele programului)
- **-150p** - pornirea a mai mult de $M + R$ thread-uri
- **-150p** - utilizarea de flag-uri de optimizare în Makefile
- **-150p** - utilizarea funcției *pow* (sau oricărei funcții similare de ridicat la putere) cu exponent subunitar (în rest, puteți folosi această funcție cât timp exponentul este mai mare decât 1)
- **-20p** - utilizarea variabilelor globale (soluția pentru a evita variabile globale este să trimiteți variabile și referințe la variabile prin argumentele funcției pe care o dați la crearea firelor de execuție).

Atenție! Checker-ul vă dă cele 120 de puncte pentru scalabilitate și corectitudinea rezultatelor, dar acela nu este punctajul final (dacă, de exemplu, temă vă scalează și dă rezultate corecte, dar nu ați urmat paradigma Map-Reduce, veți avea nota finală 0).

[2] Acest punctaj este condiționat de scalabilitate. O soluție secvențială, deși funcționează corect și dă rezultate bune, nu se va puncta.

Bonus

În cadrul acestei teme, puteți obține până la **20p** bonus la punctaj pentru stilul de programare. În general, programarea combină multe noțiuni, iar cunoștințele acumulate la alte materii pot fi folosite foarte bine în combinație cu programarea paralelă. Astfel, vom acorda acest punctaj bonus dacă:

- ați obținut punctajul maxim pentru rezolvarea problemei
- rezolvați tema în mod generic, adică dacă aveți o implementare de Map-Reduce pentru orice tip de date de intrare, orice tip de date de ieșire și orice tip de operații Map și Reduce, iar în final folosiți implementarea generică pe cazul particular al temei.

Adoptarea diverselor framework-uri de programare paralelă și distribuită cum sunt Apache Hadoop sau Apache Spark pe scară largă nu se datorează doar faptului că au o performanță bună, dar și pentru că interfețele expuse de acestea sunt generice și adesea funcționale din punct de vedere al paradigmei de programare. Pentru a vă reaminti ce înseamnă genericitatea în C/C++, puteți să vă uitați aici.

Atenție! Dacă implementați bonusul, este necesar să adăugați în arhivă și un fișier *README_BONUS* în care descrieți ce ați făcut.

Testare

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[3].

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

```
$ tree
.
+-- Makefile
+-- [...] (sursele voastre)
+-- README
+-- test.sh
+-- test0
|   +-- [...] (fisierele testului)
+-- test1
|   +-- [...] (fisierele testului)
+-- test2
|   +-- [...] (fisierele testului)
+-- test3
|   +-- [...] (fisierele testului)
+-- test4
    +-- [...] (fisierele testului)
```

La rulare, scriptul execută următorii pași:

1. compilează programul
2. pentru cinci seturi de teste, execută următoarele operații:
   (a) rulează varianta etalon cu 1 thread Mapper și 4 thread-uri Reducer și verifică dacă rezultatele sunt corecte
   (b) rulează variante cu 2/4 thread-uri Mapper și 4 thread-uri Reducer și verifică dacă rezultatele sunt corecte

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

(c) pentru toate seturile de fișiere de test în afară de primul și ultimul, calculează accelerația și verifică dacă este peste niște limite date

3. se calculează punctajul final din cele 120 de puncte alocate testelor automate (30 de puncte fiind rezervate pentru claritatea codului și a explicațiilor, așa cum se specifică mai sus).

Scriptul necesită existența următoarelor utilitare: *awk*, *bc*, *diff*, *sed*, *time*, *timeout*, *xargs*.

Atenție! Dacă aveți un calculator cu două core-uri fizice (cu sau fără hyper-threading), va trebui să modificați măsurarea accelerației să nu ia în considerare și testele cu 4 thread-uri, pentru că implementarea paralelă nu va scala.

Vă recomandăm să vă apucați de implementat tema și să încercați să o faceți să funcționeze pe fișierele de input din directorul *test0*, care este fix exemplul prezentat mai sus. Când ați obținut rezultate corecte pe acest test, puteți încerca să rulați și celelalte teste și să verificați scalabilitatea.

Docker

Pentru a avea un mediu uniform de testare, sau pentru a putea rula scriptul de test mai ușor de pe Windows sau MacOS (și cu mai puține resurse consumate decât dintr-o mașină virtuală), vă sugerăm să folosiți Docker. În repository-ul temei, găsiți un director numit *Docker*. Dacă aveți Docker instalat și rulați scriptul *start.sh*, vi se va da acces de shell într-un container care are montat sistemul local de fișiere la calea */tema1/*, de unde puteți apoi rula scriptul de testare automată. De menționat că, dacă descărcați de pe Git pe Windows, va trebui să schimbați sfârșitul liniilor din "CRLF" în "LF" și să mai rulați următoarea comandă în container, pentru a da drepturi de execuție pe script: `chmod a+x /tema1/test.sh`.

Recomandări de implementare și testare

Găsiți aici o serie de recomandări legate de implementarea și testarea temei:

1. la compilare, folosiți flag-urile `-Wall` și `-Werror`; `-Wall` vă afișează toate avertismentele la compilare, iar `-Werror` vă tratează avertismentele ca erori; aceste flag-uri vă ajută să preîntâmpinați erori cum ar fi variabile neinițializate, care se pot propaga ușor

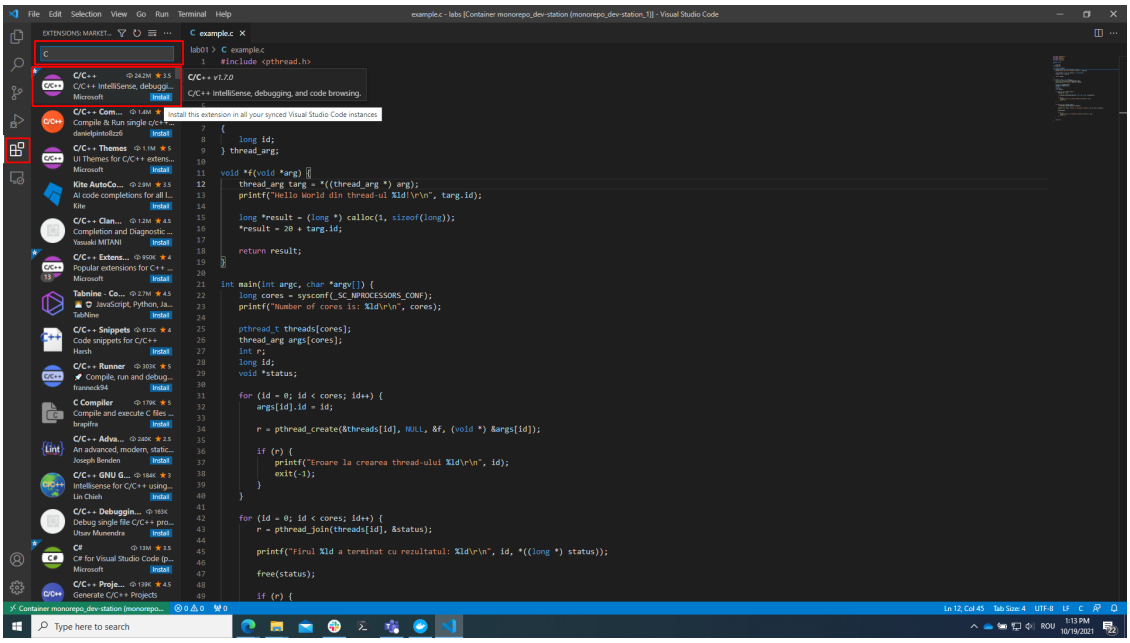
2. folosiți un repository **privat** de Git și dați commit-uri frecvent; vă ajută să vedeți diferențe între iterații de scris cod și vă asigură că aveți undeva sigur codul ca să-l puteți recupera în cazul în care l-ați șters din greșeală sau mașina pe care lucrați nu mai funcționează

3. citiți paginile de manual pentru funcțile din biblioteca Pthreads și verificați valorile returnate de acestea; dacă aceste valori corespund unor erori descrise acolo, vă puteți da seama ușor dacă folosiți greșit funcțiile sau structurile din bibliotecă

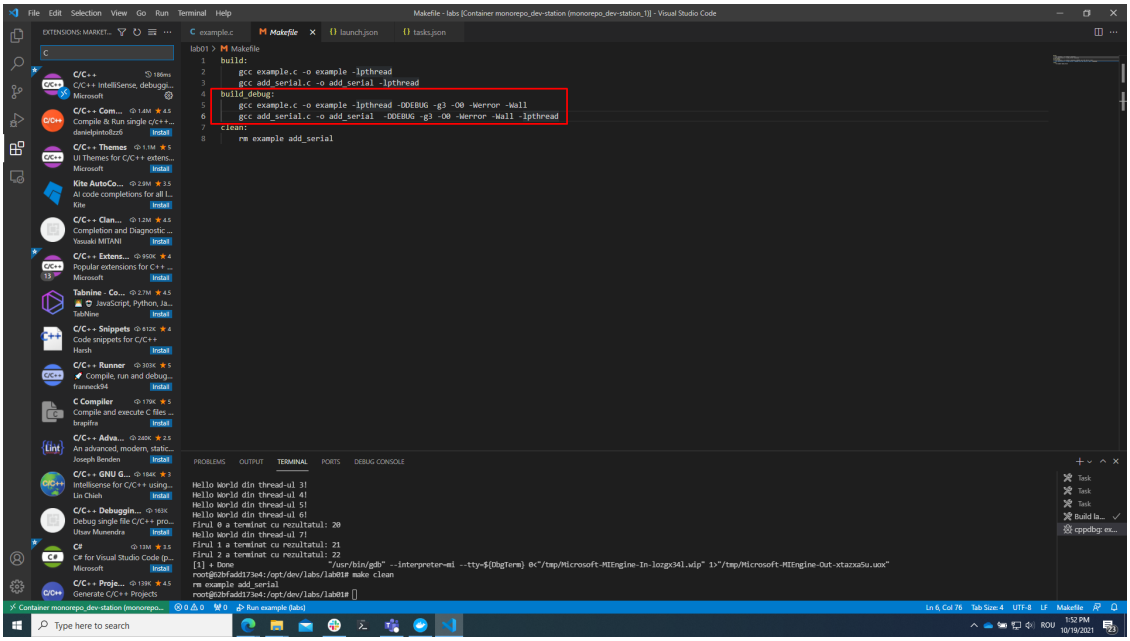
4. dacă lucrați pe Windows, vă recomandăm să folosiți WSL2 sau Docker (în loc de o mașină virtuală) pentru dezvoltare în Linux, deoarece sunt soluții care necesită mai puține resurse, iar performanța e mai aproape de cea a unui sistem de operare Linux care rulează direct pe mașinile voastre

5. folosiți un IDE precum CLion sau Visual Studio Code; pe lângă completarea automată a codului, aveți posibiliatea de a rula și a face debug pe cod.

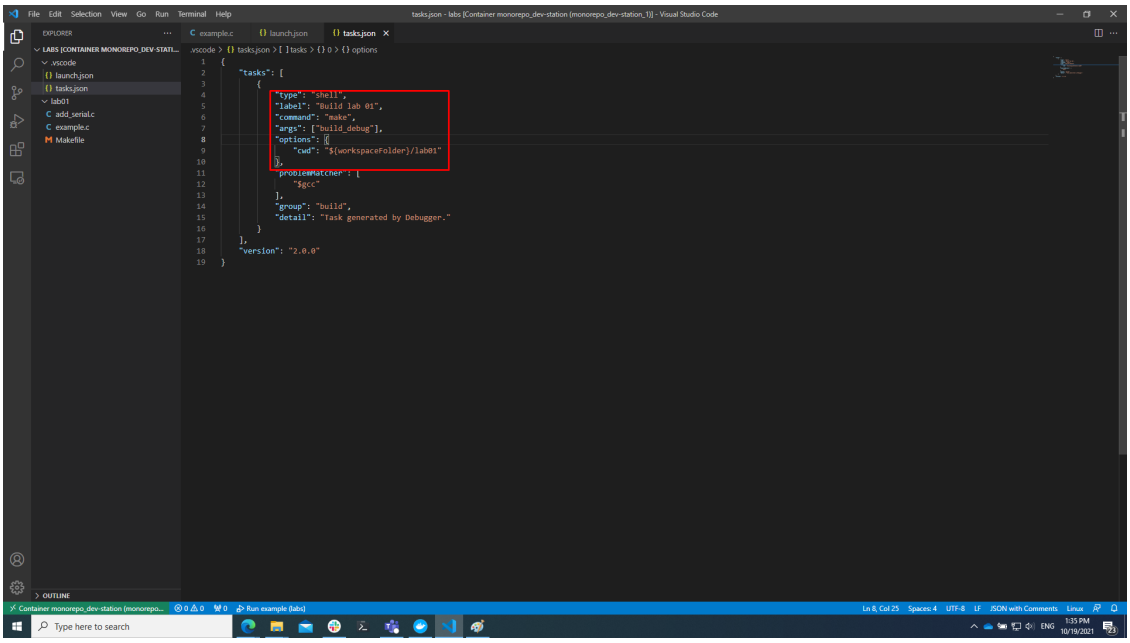
În continure, aveți un exemplu de cum puteți să faceți debug în Visual Studio Code. În primul rând, trebuie să instalați plugin-ul pentru C/C++.



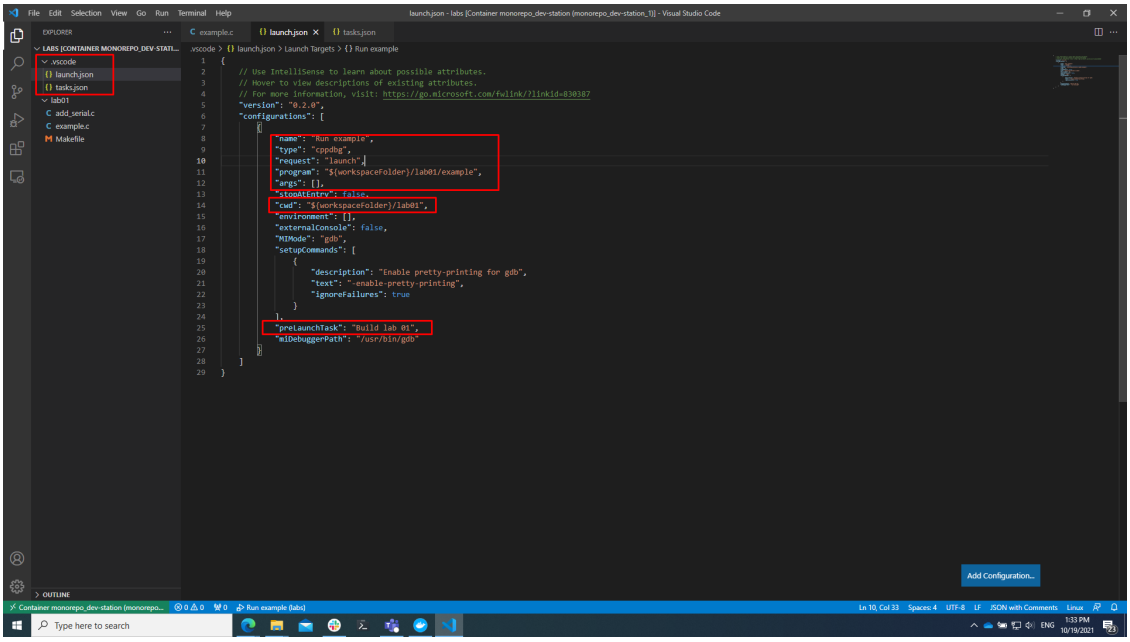
Apoi, trebuie să vă faceți în *Makefile* un build de debug care să aibă flag-urile `-O0 -g3 -DDEBUG`.



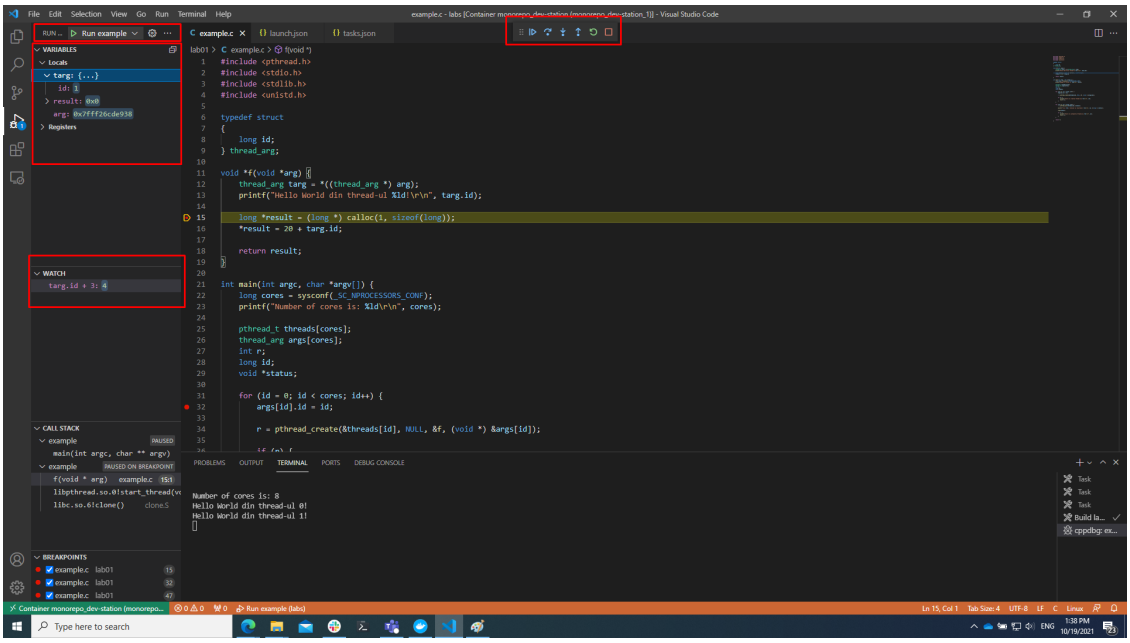
Ca să faceți build și să rulați codul, trebuie să aveți un director *.vscode* în care să aveți două fișiere, *launch.json* și *tasks.json*. În *tasks.json*, definiți ce acțiuni trebuie realizate pentru a se face build. În imaginea de mai jos, aveți un exemplu de cum trebuie să arate acest fișier. Trebuie să specificați unde se face build, care sunt argumentele pentru *make*, precum și un label pentru a identifica acțiunea de build.



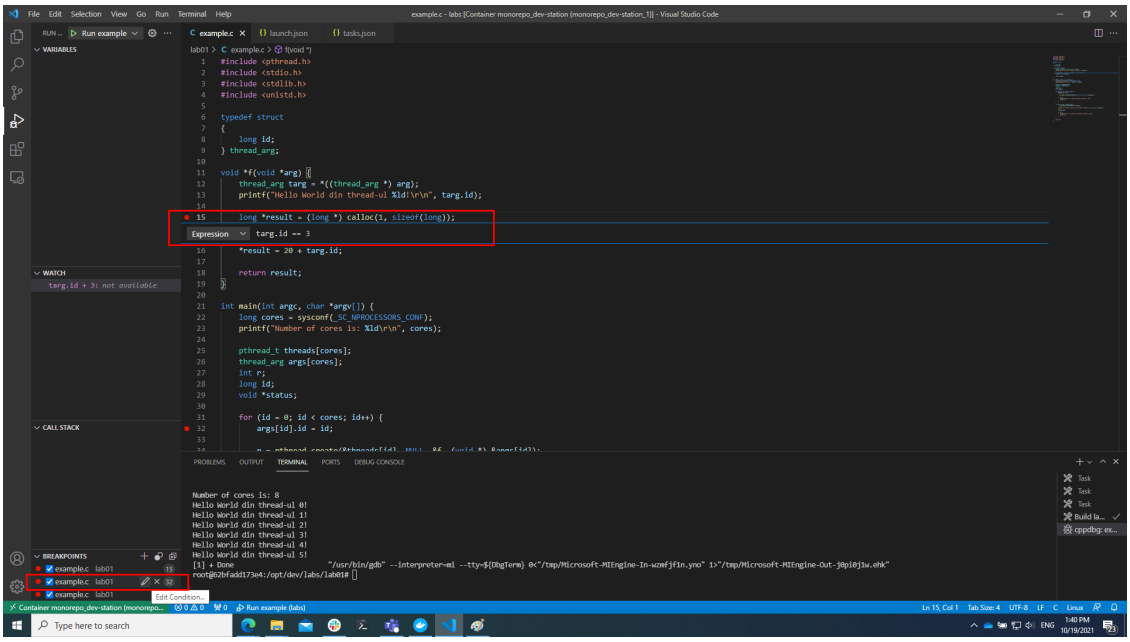
Pentru fișierul *launch.json*, trebuie să specificați programul de debug care trebuie rulat, directorul unde se rulează, precum și argumentele. De asemenea, este necesar să puneți label-ul pentru acțiunea de build definit precedent. Pentru rulare, trebuie să aveți instalat și utilitarul *gdb*.



După ce au fost create aceste fișiere, puteți să rulați executabilul de debug, să adăugați breakpoints și să navigați prin execuția programului. Puteți să mai vedeți variabilele locale și să evaluați expresii simple.



Un lucru foarte util pentru a face debug pe programe paralele este posibilitatea de a edita un breakpoint astfel încât să se declanșeze doar atunci când anumite condiții se îndeplinesc (de exemplu, dacă un fir de execuție are un anumit identificator).



Link-uri utile

1. The Anatomy of a Large-Scale Hypertextual Web Search Engine
2. Can Your Programming Language Do This?
3. MapReduce (Wikipedia)
4. MapReduce: Simplified Data Processing on Large Clusters

5. Apache Hadoop
6. Apache Spark
7. Genericity.