1 Contextul temei

Ca ingineri ai unei companii de calculatoare (voi aţi vrut…) vă confruntaţi cu următoarea situaţie:

Compania tocmai a lansat pe piaţă un nou tip de hard disk. Imediat după lansare șefii companiei au primit o sesizare că acest produs va avea o durată de funcţionare mai mică decât precizează garanţia. Pentru a evita o lovitură financiară serioasă, consiliul directorilor a decis retragerea de pe piaţă a produsului.

Observând că eroarea sesizată nu există în practică, voi aţi sugerat păstrarea pe piaţă a produsului. Consiliul directorilor a amânat retragerea produsului cu 3 săptămâni, însă vi s-a cerut să veniţi cu o dovadă clară că eroarea sesizată nu prezintă un pericol. Astfel va trebui să construiţi o simulare aproximativă a dispozitivului în C sau C++.

2 Descriere dispozitiv

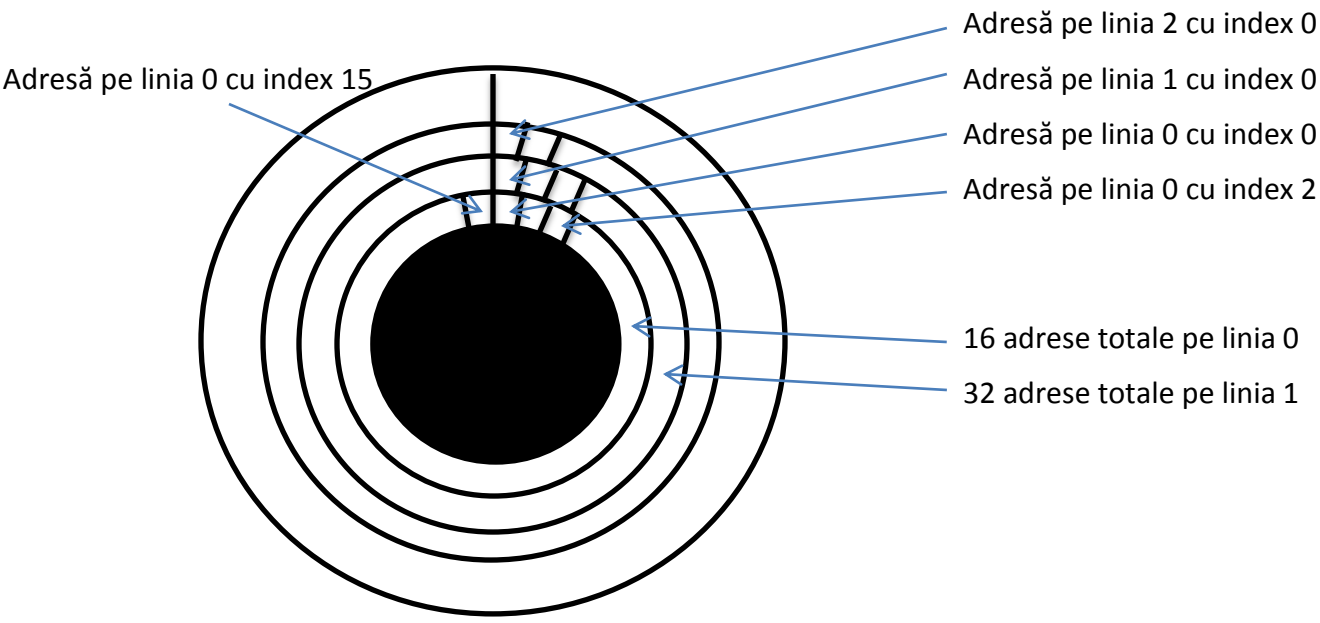
Hard disk-ul este compus dintr-un disk pe care se scrie informaţia și dintr-un cursor ce execută operaţii pe disk. Informaţia este stocată în grupări de 2 octeţi în adrese de memorie. Astfel o adresă va conţine:

{ xxxx | x ∈ {0, 1, 2, … , 9, A, B, C, D, E, F } }, deci o adresă va putea stoca **{ 0000, 0001, …, FFFF }.**

Adresele sunt grupate pe linii astfel:

- **Prima linie va conţine 16 adrese** (una lângă alta).
- Fiecare linie dinspre exterior va conţine **de 2 ori mai multe adrese decât linia precedentă** din interior.

Atăt adresele cât și liniile sunt **numerotate de la 0**. Cursorul se poate deplasa **doar în sensul acelor de ceasornic** și doar de la o adresă la altă adresă vecină, deci **nu se poate face salt peste adrese**. Pentru a schimba linia, **cursorul se poate muta sus sau jos DOAR prin adresele cu index-ul 0** al fiecărei linii și nici aici nu se poate sări peste adrese ci se parcurge tot din aproape în aproape.



Se va lucra pe timpi. Pe fiecare timp cursorul poate să facă una din următoarele operaţii:

- **Execută** o comandă pe adresa la care se află, sau
- **Se mută** cu o adresă sau
- **Rămâne pe loc** (daca nu are operaţii de executat)

Toate operaţiile aplică o uzură pe acea adresă de memorie (**DAMAGE**) pe care noi trebuie să o reţinem deoarece reprezintă o parte crucială a demonstraţiei noastre.

3 Ce vrem să demonstrăm:

Sesizarea: “Deoarece pentru a ajunge la o adresă aflată la finalul unei linii trebuie să trecem prin toate cele de la început, uzura pe primul sfert din disk va fi mai mare decât pe restul disk-ului și va ceda în acea zona.”

Este evident de ce afirmaţia de mai sus este eronată. Ce trebuie făcut acum este să simulăm comenzi pe dispozitiv după care, împărţind disk-ul în 4 zone, să calculăm media uzurii pe fiecare zonă. După multe comenzi mediile vor fi apropiate.

Roșu – prima zonă, începe de la adresa 0 a fiecărei linii (include adresa 0)

Galben –zona 2 ; Verde – zona 3 ; Albastru – zona 4.

Notă: Toate zonele conţin un număr egal de adrese de memorie

4 Comenzi

Se trimite o comandă urmată de un timp de pauză ce reprezintă timpul până se va citi următoarea comandă. Cursorul trebuie să se mute timp cu timp către adresa de memorie precizată în comandă, după care să execute comanda. Cursorul va fi iniţial pe linia 0, index 0.

Comenzi de bază:

- **Read data:**

::r [linie] [index] , unde linie și index sunt de tip “int”

Se vor afișa datele stocate la adresa de memorie de pe linia și indexul precizate (newline)

- **Write data:**

::w [linie] [index] [data] , unde linie și index sunt de tip “int”, iar data este un string de 4 caractere

Se va scrie data în adresa de memorie de pe linia și indexul precizate

- **Read damage:**

::d [linie] [index] , unde linie și index sunt de tip “int”

Se va afișa DAMAGE-ul adresei de memorie de pe linia si indexul precizate (newline)

- **End:**

::e

Se vor neglija uzurile produse de și după această comandă. Se vor afișa, în ordine, separate prin spaţiu, mediile celor 4 zone de memorie: CU 2 ZECIMALE, APROXIMATE PRIN LIPSA. Programul se va termina IMEDIAT după această comandă (**cursorul va rămâne pe linia următoare**).

Comenzi BONUS:

- **Multiple Read data:**

::mr [linie] [index] [no] , unde linie, index și no sunt de tip “int”

Se va executa “Read data” pe ‘no’ adrese de memorie începând cu cea de la linia si indexul precizate. Datele citite se vor afișa câte una pe un rând. Dacă se termină linia, se continuă cu linia superioară.

- **Multiple Write data:**

::mw [linie] [index] [data] [data] [data] [data] … [data] . ,unde linie și index sunt de tip “int”, iar data este un string de 4 caractere; comanda se termină cu “.”Dacă se termină linia, se continuă cu începutul liniei superioare. Se va scrie de la adresa de memorie de pe linia și indexul precizate mai departe datele precizate.

DAMAGE

Read data – 5 Damage pe acea adresă de memorie după ce a citit datele

Write data – 30 Damage pe acea adresă de memorie după ce a scris datele

Read damage – 2 Damage pe acea adresă de memorie după ce a citit damage-ul

La finalul fiecărui timp cursorul aplică **1 Damage** pe adresa de memorie pe care se află, indiferent dacă s-a mutat pe adresă, a rămas pe adresă sau a executat o comandă la acea adresă.

5 Cerinţe și Punctaj

- Se construiește disk-ul. Modul de primire al comenzilor și numărul de linii se citesc din fișierul de intrare de pe prima linie. Toate adresele de memorie au inițial stocat "**0000**".

- **(QUEUE)** Pe prima linie a fișierului de intrare se va afla cifra 1 urmată de un spaţiu și numărul de linii conţinute de disk. Următoarele linii vor conţine o comandă, iar pe rândul următor timpul de asteptare până când se citește următoarea comandă. Fișierul se termină cu comanda **::e** .
  Să se implementeze un algoritm (C / C++) ce simulează acest dispozitiv pe care se pot executa comenzile de bază. Comenzile vor fi **executate în stil coadă** (first in – first out), astfel se vor executa în ordinea citirii. Se va construi fișierul de output.
  3p comenzile de bază + 1p comanda end = 4p

- **(STACK)** Pe prima linie a fisierului de intrare se va afla cifra 2 urmată de un spaţiu si numărul de linii conţinute de disk. Următoarele linii vor conţine o comandă, iar pe randul următor timpul de așteptare până când se citește următoarea comandă. Fișierul se termină cu comanda **::e** .
  Să se implementeze un algoritm (C / C++) ce simulează acest dispozitiv pe care se pot executa comenzile de bază. Comenzile vor fi **executate în stil stivă** (last in – first out), astfel comenzile citite ultimele vor avea prioritate. Se va construi fișierul de output.
  4p comenzile de baza + 1p comanda end = 5p

- **Coding Style** **1p**

- **BONUS 1** – pt cazul QUEUE să se implementeze și comenzile de bonus. **1p**

- **BONUS 2** – pt cazul STACK să se implementeze si comenzile de bonus. Fiecare comanda de bonus este văzută ca o serie de comenzi individuale, deci, dacă se citește o comandă nouă în timpul executării unei comenzi de bonus, se suspendă executarea celei de bonus și se trece la comanda nouă. Ulterior se poate relua comanda suspendată când redevine prioritară. (ex. la (6) hints) **1p**

FISIERELE SE CITESC CA ARGUMENTE: nume_fisier_de_intrare urmat de nume_fisier_de_iesire

Deoarece o implementare profesionistă ar fi cu alocare dinamică și prezentarea soluţiei trebuie sa fie cât mai profesionistă, **nu se vor folosi vectori / alocare statică** (depunctare consistentă).

IMPORTANT: **NU se vor folosi biblioteci cu liste, cozi, stive deja implementate!**

Se vor depuncta pierderile de memorie (1p).