Tema - Arhitectura Sistemelor de Calcul

Seriile 13, 14, 15

Noiembrie 2024
Ultima actualizare: 17.12.2024

Cuprins

1 Detalii administrative

1.1 Deadline

Puteți trimite solutiile pana **cel tarziu pe 5 Ianuarie 2025, ora 23:59**.

1.2 Reamintirea punctajului pe tema

Tema valoreaza 25% din nota la acest laborator (conform Cursului 0x00), si este necesara obtinerea notei 5 pentru promovare.

1.3 Transmitere

Veti trimite solutiile in urmatoarele formulare, in functie de grupa: *link-urile vor fi afisate in perioada urmatoare.*

1.4 Ce se va transmite

Se vor incarca in formular **doua surse** (cate una pentru fiecare cerinta) cu denumirile **grupa_nume_prenume_0.s**, respectiv **grupa_nume_prenume_1.s**. Daca aveti mai multe nume / prenume, veti incarca surse denumite, de exemplu, **172_GeorgescuXulescu_IonVasile_0.s**. Este **important** sa incarcati surse cu denumirea corecta, deoarece testarea va fi **automata**.

1.5 Cum se va face evaluarea

Exista doi pasi pentru obținerea notei:

- se verifica toate sursele sa nu fie cazuri de plagiat. In cazul in care se detecteaza plagiat, se face automat sesizare catre *Comisia de Etica a Universitatii din Bucuresti*;
- sursele care au trecut de verificarea anti-plagiat, vor fi testate automat.

Important! Studentii care au alte configurari fata de cele pe care lucram la laborator, trebuie sa precizeze acest lucru in formularul in care transmit tema, pentru a putea efectua evaluarea si pentru a nu primi 0 implicit.

1.6 Alte observatii

1. Nu va interzicem sa discutati idei intre voi, dar aveti grija, deoarece este o diferenta importanta intre a da o idee si a da codul direct.
2. Nu folositi convertoare automate din C/C++/alte limbaje in x86, le-am folosit si noi si recunoastem fara dificultate un cod care nu este scris de voi.

2 Formularea temei

Pentru implementarea acestei teme, presupunem ca faceti parte din echipa de dezvoltare a unui *sistem de operare* minimal - sistemul de operare fiind un produs software care se ocupa cu gestionarea si coordonarea activitatilor unui sistem de calcul, cu rol in medierea accesului programelor de aplicatie la resursele masinii. Sarcina care va revine este aceea de a implementa o componenta de gestiune a dispozitivului de stocare (*hard-disk* ori *SSD*), iar pentru ca proiectul este abia la inceput, avem multe presupuneri care simplifica dezvoltarea acestui produs.

2.1 Prezentarea cerintelor

Sunt luate in calcul doua moduri de functionare, un caz in care memoria este liniara, unidimensionala, respectiv un caz in care memoria este bidimensionala.

2.1.1 Implementarea sistemului cu memorie unidimensionala

In acest caz unidimensional, modul in care vi se cere sa functioneze dispozitivul de stocare este urmatorul:

- capacitatea de stocare a dispozitivului este data si fixata la 8MB;
- capacitatea de stocare a dispozitivului este impartita in blocuri de cate 8kB fiecare;
- intr-un bloc poate fi stocat continut dintr-un singur fisier;
- un fisier are nevoie de cel putin doua blocuri pentru stocare;
- se presupune ca un fisier stocat este stocat contiguu;
- daca un fisier nu se poate stoca contiguu atunci scrierea sa pe dispozitiv nu este posibila.

Sistemul de operare nu are o structura de directoare si fisiere, ci doar trebuie sa stocheze fisiere. In acest sens, fiecare fisier este identificat printr-un descriptor - ID unic (un numar natural intre 1 si 255); astfel, sistemul nostru poate stoca maximum 255 de fisiere diferite.

Ne intereseaza ca modulul de management al dispozitivului de stocare sa poate realiza urmatoarele operatii:

- dat un descriptor (ID de fisier), sa se returneze intervalul de blocuri `(start, end)` unde este stocat fisierul;
- dat un descriptor de fisier si dimensiunea sa in kB, sa se returneze intervalul de blocuri unde poate sa fie stocat fisierul. Se va returna primul interval liber, in parcurgerea de la stanga la dreapta. In cazul in care stocarea nu este posibila, se returneaza intervalul `(0, 0)`;
- dat un descriptor, sa se stearga fisierul respectiv (adica sa se elibereze blocurile unde continutul fisierul a fost salvat); consideram *stergere* operatia prin care blocurile primesc drept descriptor valoarea `0`;
- operatia de defragmentare: reordonati/recalculati blocurile in care sunt stocate fisierele, astfel incat acestea sa fie stocate compact (adica incepand cu blocul 0 si folosind toate blocurile consecutive, fara goluri).

2.1.2 Implementarea sistemului cu memorie bidimensionala

Dupa ce ati finalizat implementarea in cazul unidimensional, observati ca spatiul pe care il aveti se umple destul de repede, asa ca va ganditi la o extindere naturala, pe doua dimensiuni de aceasta data, cu o dimensiune de 8MB in ambele sensuri. Astfel, presupunem acum ca dispozitivul de stocare este bidimensional, adica avem o matrice de blocuri, iar o sectiune contigua este considerata pe linii.

- dat un descriptor de fisier, sa se returneze intervalul de blocuri `((startX, startY), (endX, endY))` unde este stocat fisierul;
- dat un descriptor de fisier si dimensiunea sa in kB, sa se returneze ID-urile blocurilor unde poate sa fie stocat fisierul. Se va intoarce primul interval in care fisierul poate fi pozitionat. Daca nu se poate stoca atunci se returneaza intervalul `((0,0), (0,0))`
- dat un descriptor de fisier, sa se stearga fisierul respectiv (adica sa se elibereze blocurile unde continutul fisierului a fost salvat); exact ca in cazul unidimensional, consideram *stergere* operatia prin care blocurile primesc drept descriptor valoarea 0;
- operatia de defragmentare bidimensionala; reordonati blocurile in care sunt stocate fisierele, astfel incat acestea sa fie stocate compact in matrice (mutam golurile in dreapta-jos in matrice).

2.2 Exemplu de functionare

Pentru implementarea temei, veti considera inclusiv o reprezentare simplificata a dimensiunii memoriei, doar pentru a putea demonstra ca gestiunea sistemului de fisiere functioneaza. Astfel, veti reduce cei 8kB ai unui bloc la 8b (1 Byte), si veti pastra aceeasi conversie si pentru dimensiunea memoriei.

2.3 Exemplu pentru cazul unidimensional

Initial, memoria voastra este 0 in toate blocurile:

```
0, 0, 0, 0, 0, 0, ..., 0, 0, 0
```

ADD Daca se adauga un fisier cu descriptorul 5 si o dimensiune de 20kB, modificarea va fi astfel:

- descriptorul 5 ne spune ca vom modifica blocurile cu valoarea 5;
- dimensiunea de 20kB se reduce in reprezentare la 20b, ceea ce inseamna ca avem nevoie de 3B (aproximarea superioara) pentru reprezentare - deci avem nevoie de 3 blocuri;
- vom aloca aceste blocuri in ordine.

```
5, 5, 5, 0, 0, 0, ..., 0, 0, 0
```

ADD Presupunem ca mai adaugam un fisier cu descriptorul 143 si o dimensiune de 14kB. Atunci, pentru el avem nevoie de doua blocuri, iar memoria va deveni, acum:

```
5, 5, 5, 143, 143, 0, ..., 0, 0, 0
```

GET Daca vrem sa interogam sistemul de operare si sa ne spuna unde gasim fisierul cu descriptorul 143, ne va intoarce intervalul `(3, 4)`, iar daca cerem fisierul cu descriptorul 10, ne va intoarce intervalul `(0, 0)`, intrucat acest fisier nu exista.

DELETE Presupunem ca stergem fisierul cu descriptorul 5. In acest caz, memoria rezultata arata astfel:

```
0, 0, 0, 143, 143, 0, ..., 0, 0, 0
```

ADD Presupunem ca adaugam iar un fisier, de data aceasta de 73kB, cu descriptorul 25. Pentru stocarea lui consideram 73b, ceea ce inseamna ca aproximare superioara 10B, deci 10 blocuri. Ele se aloca, in ordine, dupa ultimul descriptor pe care l-am completat, deci dupa descriptorul 143. Memoria va arata astfel:

```
0, 0, 0, 143, 143, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 0, ..., 0, 0, 0
```

ADD Presupunem ca adaugam un fisier cu descriptorul 251 si dimensiune de 12kB. In reprezentare avem 12b, deci 2B, corespunzatori pentru doua blocuri. Intrucat avem deja trei blocuri libere fix la inceput, fisierul va incapea in primele doua dintre ele.

```
251, 251, 0, 143, 143, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 0, ..., 0, 0, 0
```

DELETE Presupunem ca stergem fisierul cu descriptorul 25. In acest caz, memoria va arata astfel:

```
251, 251, 0, 143, 143, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..., 0, 0, 0
```

DEFRAGMENTATION Acum, presupunem ca vrem sa facem o defragmentare a memoriei, astfel incat stocarea sa fie din nou compacta si sa permitem ca noile fisiere sa fie stocate la adrese mai mici de memorie, nu abia dupa fisierul cu descriptorul 251. In urma aplicarii acestei operatii, fisierele cu descriptorii 143 si 255 vor fi la adrese mici de memorie, stocate consecutiv:

```
251, 251, 143, 143, 0, 0, 0, ..., 0, 0, 0
```

ADD Acum, daca presupunem adaugarea unui fisier cu descriptorul 4, care ocupa 22kB, memoria va fi modificata astfel:

```
251, 251, 143, 143, 4, 4, 4, 0, 0, 0, ..., 0, 0, 0
```

Observatie. Daca acum stergem fisierul cu descriptorul 4, si adaugam un altul nou, cu descriptorul 29, intrucat fisierul cu descriptorul 4 era ultimul alocat, cel cu 29 va fi pus imediat dupa 143.

2.4 Exemplu pentru cazul bidimensional

Initial, memoria voastra este 0 in toate blocurile:

```
0, 0, 0, 0, 0, ..., 0, 0, 0
0, 0, 0, 0, 0, ..., 0, 0, 0
...
0, 0, 0, 0, 0, ..., 0, 0, 0
0, 0, 0, 0, 0, ..., 0, 0, 0
```

Pentru simplitate, vom considera o matrice completa, dar mai restransa, pentru a ilustra operatiile (nu vom considera memoria de 8MB x 8MB in acest exemplu, dar trebuie sa o considerati astfel in tema). In exemplul urmator, vom considera o memorie de 64kB pe 64kB.

```
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
```

ADD Adaugam un fisier cu descriptorul 5, respectiv o dimensiune de 50kB. (ocupa 7 blocuri)

```
5, 5, 5, 5, 5, 5, 5, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
```

ADD Adaugam un fisier cu descriptorul 2, respectiv o dimensiune de 52kB. (ocupa tot 7 blocuri)

```
5, 5, 5, 5, 5, 5, 5, 0
2, 2, 2, 2, 2, 2, 2, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
```

ADD Procedam intocmai si adaugam fisierele cu descriptorii 1, 3, 4, 6, 7, 8, de dimensiuni care ocupa 2, 4, 3, 5, 2, respectiv 8 blocuri.

```
5, 5, 5, 5, 5, 5, 5, 0
2, 2, 2, 2, 2, 2, 2, 0
1, 1, 3, 3, 3, 3, 7, 7
4, 4, 4, 6, 6, 6, 6, 6
8, 8, 8, 8, 8, 8, 8, 8
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
```

Observatie. Fisierul cu descriptorul 7 a fost pus imediat dupa zona pentru fisierul cu descriptorul 3, deoarece erau deja doua blocuri disponibile.

GET Daca vrem sa interogam sistemul de operare si sa ne spuna unde gasim fisierul cu descriptorul 3, ne intoarce intervalul `((2, 2), (2, 5))`.

DELETE Presupunem ca vrem sa stergem fisierele cu descriptorii 1, 4, 8. Rezultatul in memorie este:

```
5, 5, 5, 5, 5, 5, 5, 0
2, 2, 2, 2, 2, 2, 2, 0
0, 0, 3, 3, 3, 3, 7, 7
0, 0, 0, 6, 6, 6, 6, 6
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
```

DEFRAGMENTATION Acum, presupunem ca vrem sa facem o defragmentare a memoriei, incat stocarea sa fie din nou compacta. Astfel, vrem ca 0-urile sa ajunga in coltul din dreapta jos.

```
5, 5, 5, 5, 5, 5, 5, 0
2, 2, 2, 2, 2, 2, 2, 0
3, 3, 3, 3, 7, 7, 0, 0
6, 6, 6, 6, 6, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0
```

Observatie. Stocarea fisierelor se face pe linii, iar in defragmentare pastram ordinea fisierelor, incercand sa umplem golurile, acolo unde este posibil.

3 Cerinte

Observatie! In tema consideram `1 MB = 1024 kB`, respectiv `1kB = 1024 B`.

In cadrul acestei teme, aveti doua cerinte: o cerinta pentru abordarea unidimensionala, pentru care veti primi 50p, respectiv o cerinta pentru partea bidimensionala, pentru care veti primi 40p. **Important! NU** introduceti inputul manual la fiecare retestare a programului! Sunt inputuri lungi, care va vor costa timp. Creati-va un fisier, de exemplu `input.txt`, in care scrieti inputul dorit, iar dupa ce aveti un executabil, de exemplu `task00`, pe care in mod normal l-ati fi rulat cu `./task00`, rulati comanda `./task00 < input.txt`. Astfel, continutul din fisier va fi redirectat la **STDIN**, exact ca atunci cand ati fi introdus manual valorile. Folositi aceasta informatie si pentru a va testa mai multe inputuri, creandu-va fisiere `input0.txt`, `input1.txt` etc., si testandu-le cu `./task00 < input0.txt`, `./task00 < input1.txt` etc.

Important! Toate sirurile de caractere (utilizate pentru afisare) pe care le definiti in sectiunea `.data` vor avea, la final, caracterul '\n'!

3.1 Cerinta 0x00 - cazul unidimensional

Aceasta cerinta va permite obtinerea a 50p, dintre care:

- 10p obtineti doar pentru implementarea functionalitatii ADD;
- 10p obtineti daca ati implementat ADD si GET;
- 10p obtineti daca ati implementat ADD si DELETE;
- 5p obtineti daca ati implementat ADD, DELETE, GET;
- 15p obtineti daca ati implementat ADD, DELETE, GET, DEFRAGMENTATION.

(se obtin 10p pentru ADD, 20p pentru implementarea GET sau DELETE, 35p daca sunt si ADD, si GET, si DELETE, respectiv 50p daca sunt toate)

In implementarea primei cerinte, aveti urmatorul input:

- prima linie a inputului contine numarul O de operatii care vor fi efectuate;
- incepand cu a doua linie, veti primi fiecare operatie, codificarea fiecareia fiind:
  - 1 - ADD
  - 2 - GET
  - 3 - DELETE
  - 4 - DEFRAGMENTATION
- daca operatia este ADD, atunci:
  - urmatoarea linie contine un numar N, reprezentand numarul fisierelor care vor fi adaugate;
  - pe urmatoarele 2N linii veti primi, succesiv, descriptorul de fisier, respectiv dimensiunea in kB a fisierului respectiv;

  - veti returna **intervalele** (reprezentate de ID-urile blocurilor) in care au fost inserate fisierele date, sub forma `"%d: (%d, %d)\n"`, unde primul intreg este descriptorul, al doilea este capatul din stanga, al treilea este capatul din dreapta. Intervalele se considera intervale **inchise**;
  - daca un fisier nu poate fi adaugat, se afiseaza `fd: (0, 0)` (unde `fd` este descriptorul pe care trebuia sa-l adaugati)

- daca operatia este GET, atunci veti primi doar un descriptor de fisier pe linia urmatoare si se va intoarce intervalul in care fisierul se gaseste, respectiv intervalul `(0, 0)` daca fisierul nu exista; Formatul de afisare trebuie sa fie sub forma `"(%d, %d)\n"`

- daca operatia este DELETE, atunci veti primi doar un descriptor de fisier pe linia urmatoare, si se va intoarce memoria actualizata; daca fisierul de sters exista, atunci el nu va mai fi continut in memorie (blocurile vor fi inlocuite cu 0), altfel memoria va fi afisata fara nicio modificare;

- daca operatia este DEFRAGMENTATION, se va afisa memoria defragmentata, sub acelasi format ca in urma efectuarii operatiei **ADD**, conform explicatiilor din sectiunea anterioara (se va asigura continuitatea blocurilor care stocheaza fisierele).

Input	**Explicatie input**	**Output**	**Explicatie output**
4 1 5 1 124 4 350 121 75 254 1024 70 30	se efectueaza 4 operatii se efectueaza ADD numarul de fisiere descriptorul primului fisier dimensiunea in kB descriptorul celui de-al doilea fisier dimensiunea in kB descriptorul celui de-al treilea fisier dimensiunea in kB descriptorul celui de-al patrule fisier dimensiunea in kB descriptorul celui de-al cincilea dimensiunea in kB	1: (0, 15) 4: (16, 59) 121: (60, 69) 254: (70, 197) 70: (198, 201)	output prima operatie
2 121	se efectueaza GET descriptorul de fisier peste care facem GET	(60, 69)	output a doua operatie
3 4	se efectueaza DELETE descriptorul de fisier peste care facem DELETE	1: (0, 15) 121: (60, 69) 254: (70, 197) 70: (198, 201)	output a treia operatie
4	se efectueaza DEFRAGMENTATION	1: (0, 15) 121: (16, 25) 254: (26, 153) 70: (154, 157)	output a patra operatie

3.2 Cerinta 0x01 - cazul bidimensional

Aceasta cerinta va permite obtinerea a 50p, dintre care

- 10p se obtin doar din implementarea metodei ADD;

- inca 10p se obtin doar daca sunt implementate ADD, GET, DELETE;
- inca 10p se obtin doar daca sunt implementate ADD, GET, DETELE, DEFRAGMENTATION;
- 20p se obtin daca se implementeaza complet tema, dar fisierele utilizate sunt fisiere existente pe disc.

In implementarea acestei cerinte, aveti urmatorul input:

- prima linie a inputului contine numarul O de operatii care vor fi efectuate;
- incepand cu a doua linie, veti primi fiecare operatie, codificarea fiecareia fiind:
  - 1 - ADD
  - 2 - GET
  - 3 - DELETE
  - 4 - DEFRAGMENTATION
- daca operatia este ADD, atunci:
  - urmatoarea linie contine un numar N, reprezentand numarul fisierelor care vor fi adaugate;
  - pe urmatoarele 2N linii veti primi, succesiv, descriptorul de fisier, respectiv dimensiunea in kB a fisierului respectiv;
  - veti returna **intervalele** (reprezentate de ID-urile blocurilor) in care au fost inserate fisierele date, sub forma `"%d: ((%d, %d), (%d, %d))\n"`, unde primul intreg este descriptorul, al doilea este capatul din stanga, al treilea este capatul din dreapta. Intervalele se considera intervale **inchise**;
  - daca un fisier nu poate fi adaugat, se afiseaza `fd: ((0, 0), (0, 0))` (unde `fd` este descriptorul pe care trebuia sa-l adaugati)
- daca operatia este GET, atunci veti primi doar un descriptor de fisier pe linia urmatoare si se va intoarce intervalul in care fisierul se gaseste, respectiv intervalul `((0, 0), (0, 0))` daca fisierul nu exista; Formatul de afisare trebuie sa fie sub forma `"((%d, %d), (%d, %d))\n"`
- daca operatia este DELETE, atunci veti primi doar un descriptor de fisier pe linia urmatoare, si se va intoarce memoria actualizata; daca fisierul de sters exista, atunci el nu va mai fi continut in memorie (blocurile vor fi inlocuite cu 0), altfel memoria va fi afisata fara nicio modificare;
- daca operatia este DEFRAGMENTATION, se va afisa memoria defragmentata, sub acelasi format ca in urma efectuarii operatiei **ADD**, conform explicatiilor din sectiunea anterioara (se va asigura continuitatea blocurilor care stocheaza fisierele).

In plus, se considera si operatia 5 - CONCRETE, caz in care se primeste un *file path* absolut, si se utilizeaza operatia ADD deja implementata, astfel incat memoria bidirectionala sa fie completata in functie de fisierele deja existente in *path*-ul specificat. Pentru a rezolva aceast sarcina, din calea respectiva, pentru fiecare fisier, vor fi determinate descriptorul (modulo 255 + 1), respectiv dimensiunea in kB a fisierului.

Operatia CONCRETE functioneaza in felul urmator:

- in input se primesc 5 si un *path* absolut catre folderul in care avem fisierele;
- fisierele sunt fisiere text, puteti considera ca sunt denumite de forma `file0.txt`, `file1.txt`, ..., `fileN.txt`;
- programul vostru trebuie sa:
  - determine un *file descriptor* pentru fiecare fisier; veti face acest lucru printr-un apel de sistem `open`. Atentie ca `open` va returneaza un descriptor real, voi trebuie sa transformati acel descriptor intr-o masura intre 1 si 255 (puteti folosi ca `int fd = (fds % 255) + 1`;
  - determine dimensiunea in kB a fisierului deschis.
- intrucat pentru fiecare fisier vom sti o pereche de forma `(file descriptor, size in kB)`, operatia CONCRETE va fi tratata exact ca un ADD;
- pe masura ce gasiti fisierele, afisati pe ecran aceste informatii, descriptorul **calculat** (cel modulo 255 + 1), respectiv dimensiunea in kB, alaturi de intervalul unde poate fi completat in memorie;
- daca un descriptor calculat se repeta, il afisati (tot sub forma ca scrieti la STDOUT descriptorul, respectiv dimensiunea), afisati `fd:  ((0, 0), (0, 0))` dar NU il mai adaugati in memoria voastra bidimensionala.

De exemplu, in limbajul C, ati avea de scris urmatoarele:

```
int fds = open(filepath, O_RDONLY);
int fd = (fds % 255) + 1;

struct stat fileStat;
fstat(fds, &fileStat);
long size = fileStat.st_size / 1024;
close (fds);
```