Curs 11

Fisiere

Abstractizare pentru stocarea persistenta a datelor. (stocarea persistenta se face pe discuri da si pe memorii flash, SSD, NVRAM, etc)
Sistemele de fisiere = componenta a sistemului de operare (parte a kernelului)
= gestioneaza mediul de stocare persistenta a datelor
=> abstractia de fisier si apeluir sistem corespunzatoare

Fisierele = containere pentru stocarea persistenta a datelor
= paradigma de folosire = open-read/write-close

File Concept = spatiu contiguu de adrese logice
= data (numerice, caractere, binare)
= program
= Definite de user = text file, source file, exe file, etc

Atributele unui fisier sunt: nume, identificator (ID pt fisier in file system), tip, locatie, dimensiune, protectie, time, data, uuser id

Operatiile disponibile pentru un fisier:
Create, write, read, reposition within file, delete, truncate, open, close

Open Files

Open-file table = monitorizeaza fisierelele deschise
= file pointer = pointer catre locatia ultimei citiri/scrieri (pt fiecare proce care a deschis fisierul)

File-open count = numarul de fisiere deschise

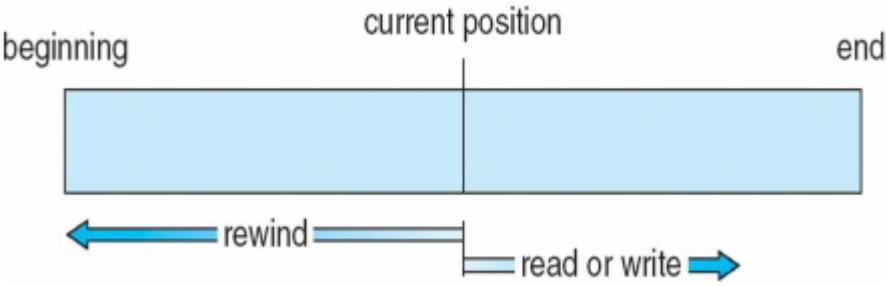
File Locking

Similar cu reader-writer lock
Shared lock = reader lock = mai multe procese pot face citi concurent
Exclusive lock = writer lock

Metode de acces: secvential, direct, alte metode

Acces secvential

Operatii: read next, write next, reset, rewrite



Acces Direct

Operatii: read n, write n, position to n (read next, write next, rewrite n)
N = numar relativ al unui bloc

Alte metode de acces

Pot fi prelungiri ale celorlalte doua metode.
In general, presupun crearea unui index pentru gasirea eficienta a fisierelor. (index prea mare => se creaza un memory-index care e un index al indexului din disc)

Fisiere in Unix

Tipuri de fisiere:

1. Obisnuite (stocheaza datelele utilizator, implementate ca un stream/sir de bytes)
2. Fisiere director(directoare) (contin nume de fisiere obisnuite, instituie o structura ierarhica a spatiului de nume pt fisiere)
3. Fisiere speciale (interfata catre driverele de echipamente sau structuri de date speciale ale kernelului (named pipes pt IPC))

Toate fisierele se accesau folosind aceleasi apeluri sistem + lucrul cu directoare.

Fisiere Unix la nivel aplicatie

- ➔ Inainte de accesul la date, un fisier trebuie deschis (apel de sistem open)
- ➔ Kernelul aloca un descriptor de fisier
- ➔ Intern kernelul aloca si un obiect corespunzator fisierului ce contine file pointer care reflecta offsetul in cardul stream-ului de bytes
- ➔ Pozitia file pointer-ului se poate schimba cu seek
- ➔ Read/write lucreaza cu byte-ul referit de catre file pointerul curent

➔ Daca doua procese deschid acelasi fisier folosesc offseturi diferite in fisier
➔ Descriptorii de fisiere se pot duplica folosind dup/dup2, situatie in care pointer-ul este partajat
➔ Fisierele sunt organizate intr-o ierarhie arborescenta cu un root unic

Arhitectura discurilor

Mediul de stocare e format prin stivuirea unor **platane** pe un ax care formeaza un **pachet** ce se invarte in jurul axului cu 5k-10krpm.
Blocurile de disc (sectoare) situate la ac. distanta de centrul platanului = **pista**
Cilindru = setul de piste la aceeasi distanta fata de axul platanelor
= toate datele dintr-un cilindru pot fi accesate simultan fara miscarea capului de citire (datele se citesc in multipli de dimensiunea blocului)

Citirea blocurilor

1. Se muta capul de citire deasupra cilindrului ce are blocul de date. 5ms
2. Se asteapta pana cand discul se roteste astfel incat datele sa ajunga sup capul de citire. 4ms pt un disc cu 7200 rpm
3. Se transfera datele prin alegerea capului de citire de deasupra pistei pe care se afla datele. 1ms pt 1KB

Algoritmi de disk scheduling

Incearca sa minimizeze timpul de cautare (seek time)

FCFS – simplu, dar inefficient

- ex: coada de cereri blocuri aflate pe cilindrii 53, 98, 183, 37, 122, 14, 124, 65, 67 => capetele de citire se vor muta peste 640 de cilindri

SSTF (shortest seek time first)

- serveste cererile de pe cilindrii cei mai apropiati de pozitia curenta a capetelor de citire
- ex anterior: capetele de citire se muta peste 236 de cilindri
- nu e optimal, ex: 53, 37, 14, 65, 67, 98, 122, 124, 183 => 208 cilindri parcursi
- sufera de starvation: daca apar in permanenta cereri in aproprierea capetelor de citire, cererile “indepartate” sunt intarziate indefinit

SCAN (algortimul liftului)

- bratul discului porneste de la un capat al acestuia catre celalalt si serveste cererile intalnite in cale
- ajuns la capatul discului o ia in sens invers
- ex. anterior: 203 salturi de cilindri
- o cerere aparuta chiar inaintea capului de citire e servita imediat
- o cerere aparuta imediat in spatele capului de citire e intarziata pana se intoace capul de citire in sens contrar

C-SCAN (circular SCAN)

- pt o distributie uniforma a cererilor, cand bratul ajunge la capatul discului si se intoarce, exista relativ putine cereri in fata capului pt ca acestea tocmai au fost tratate
- densitatea mare de cereri noi e la celalalt capat al discului unde cererile au asteptat cel mai mult
- ofera timp de asteptare mai uniform (la momentul atingerii capatului discului, capul de citire se intoarce la celalalt capat fara a mai trata cererile din fata capului de citire)
- trateaza cilindrii discului ca pe o lista circulara
- ex anterior: bratul se muta peste 167 de cilindri (se considera ca mutarea capului de citire la inceputul discului e o operatie f. rapida)

LOOK

- SCAN si C-SCAN se muta capul de citire peste tot discul
- o implementare practica ia in calcul doar cererile pt cilindrii situati intre nr minim, respectiv maxim
- algoritmii respectivi s.n. LOOK si respectiv C-LOOK
- ex anterior: la cilindrul 183 bratul se intoarce imediat la cilindrul 14

Factori care influenteaza performanta

1. Numarul de cereri si tipul lor (daca exista o singura cerere in coada, toti algoritmii se comporta ca FCFS)
2. Metoda de alocare a fisierului (contigua cu miscare limitata a capetelor de citire sau indexata blocurile pot fi imprastiate pe disc, deci deplasarile sunt mai mari ale capetelor de citire)
3. Plasamentul directoarelor si indexarea blocurilor

Structura Diskurilor

Diskurile pot fi divizate in partitii si pot fi protejate RAID.
Diskurule sau partitiile pot fi utilizate raw (fara file system) sau formatate (cu file system).
O entitate ce contine file system = **volum**
Fiecare volum monitorizeaza file systemul in **device directory**

Liste de acces si Grupuri in UNIX

Mode of access: read, write, execute
Three classes of users on Unix / Linux

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 0 0 1

File-System Structure

File System block (FCB) = structura de stocare formata din informatiile fisierului

= organizat in layere:

1. Application program
2. Logical file system
3. File-organization module
4. Basic file system
5. I/O control
6. Devices

Device driver = controleaza device-ul fizic
= gestioneaza I/O devices din I/O control layer

Basic file system = traduce comenzi pentru device driver si gestioneaza memory buffers si caches

File organization module = intelege fisiere, adrese logice si blocuri fizice
Logical file system = gestioneaza informatia metadata
= traduce un nume de fisier intr-un numar de fisier, handle de fisier, locatie mentinand FCB (inodes in unix)
= directory management + protectie

Unix System V

Primul bloc al sistemului de fisiere e rezervat codului de boot
Superblock-ul contine:

1. dimensiunea in blocuri a sistemului de fisiere si a listei de inode-uri
2. nr de blocuri si inode-uri libere
3. lista partiala cu blocuri libere
4. lista partiala cu inode-uri libere
5. in general, cele 2 liste sunt prea mari pt a fi tinute integral in superblock

Superblock-ul contine vectori care identifica primele inoduri, respectiv blocuri de date libere.
! daca lista de inode uri libere e goala, kernelul scaneaza discul pentru a gasi inode-uri libere si a reumple lista partiala
! pt lista de blocuri libere se folosesc liste de blocuri libere inlanuite folosind primul element al listei partiale de blocuri libere din superblock

Inode-uri Unix

Fiecare fisier e rerezentat prin i-node (index node), iar lista de inode uri de dupa superblock are lungimea fixa si e alocata la crearea sistemului de fisiere.

i-node contine: tipul fisierului, drepturile asupra lui, numarul de linkuri ale fisierului, dimensiunea in bytes, timpul ultimului acces, timpul ultimei modificari, timpul ultimei schimbari ale i-nodeului si vectorul de adrese de disc.

Virtual Filesystem Switch (VFS)

VFS defineste abstractia de v-node

1. la instalarea unui sistem de fisiere, kernelul aloca o structura VFS care inregistreaza in campul v_op pointeri catre functiile necesare implementarii apelurilor de sistem (open, close, read, write, etc) pt acel tip de sistem de fisiere
2. de fapt, un fisier deschis e reprezentat in kernel de un v-node (nu inode), eventual proaspat alocat, daca fisierul nu exista
3. restul kernelului invoca functiile din v_op prin functii generice care redirectioneaza executia catre operatiile specifice sistemului de fisiere respective

! Directoare Unix sunt implementate ca perechi (nr inode, nume)

Metode de alocare

Contigua

An allocation method refers to how disk blocks are allocated for files:

Each file occupies set of contiguous blocks

- Best performance in most cases
- Simple – only starting location (block #) and length (number of blocks) are required
- Problems include:
  - Finding space on the disk for a file,
  - Knowing file size,
  - External fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

Linkuita

- Each file is a linked list of blocks
- File ends at nil pointer
- No external fragmentation
- Each block contains pointer to next block
- No compaction, external fragmentation
- Free space management system called when new block needed
- Improve efficiency by clustering blocks into groups but increases internal fragmentation
- Reliability can be a problem
- Locating a block can take many I/Os and disk seeks

Indexata

- Each file has its own **index block**(s) of pointers to its data blocks
- Logical view