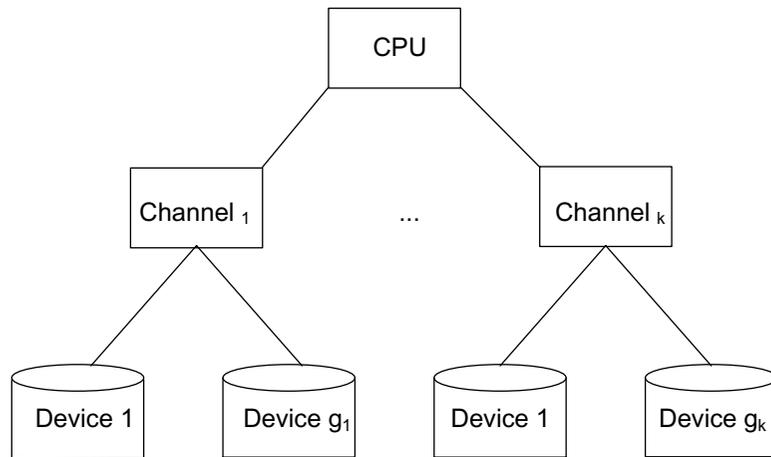


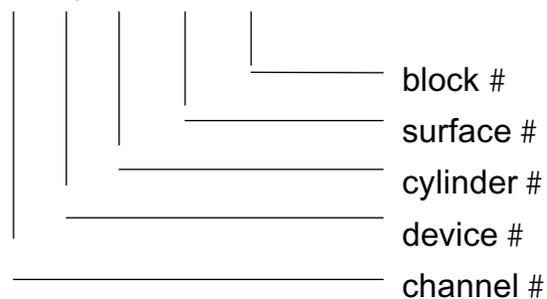
Chapter 1.2 File-System

Physical Disk Storage



Address of a disk block:

$(K_i, G_j, Z_k, O_e, B_m)$



i.e. interface **PPS** : physical peripheral storage with physical blocks (P-Block)

Operations of PPS:

„start channel programm“ in main frames
 „call disk driver“ in WS and PC

Abstract peripheral disk store: APS

- Objects: Sets of logical blocks
 e.g. Drive, Minidisk, Segment
 often: | Block | \neq | P-Block |
- Addressing: as uniformly as possible
 e.g. N x N
- Operations: **read** (Blockaddr, Bufferaddr)
write (Blockaddr, Bufferaddr)

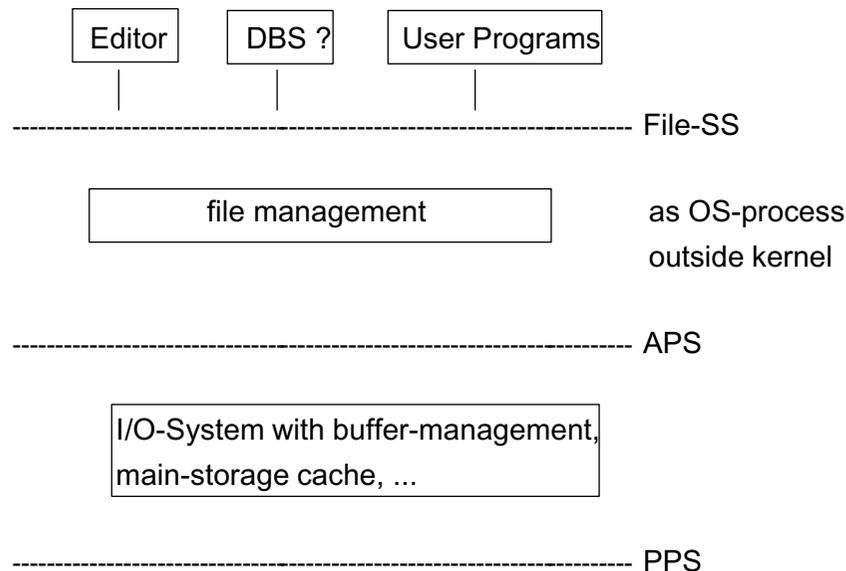
Motivation for APS:

- Segments are freely configurable
- Size of segment, | Block |
 change dynamically
- shadowing of defect P-Blocks
- simple addressing, management of free storage

File-Interface, File-System:

- Objects: Files of Records
 - rich structure of files and record
 - e.g. sequential, random,
 - index-sequential via record-keys, ...
- Addressing: various methods, depending on file and record type
- Operations: get, put, ...
 - open, close, EOF, ...

Architecture of File-Systems:



Tasks of the I/O-systems

1. Creation and management of block-sets

2. Increase, decrease block-sets
3. Management of P-blocks (free storage)
 - mapping: { blocks } → { P-blocks }
 - with | block | ≠ | P-block |
 - replace (shadow) P-blocks
 - ⇒ address book, dictionary
4. Execution of read / write operation calls
 - i.e. interpret parameters
 - evaluate (search) dictionary
 - start and manage PPs calls
 - e.g. start I/O
 - process interrupts
5. Buffer-Management: mainstore-cache
 - disk-caches of main frames in **Extended Storage System (ESS)**
 - i.e. semiconductor storage with block-interface

Tasks of the file-system

1. create, delete files
 - various file types

(large variety for mainframes)

2. manage file catalogues

catalogue = special type of file
embedding technique, like DB-Schema

3. Storage management for files

mapping: { Records } → { block areas }
e.g. for B-Trees
tree-node → block
internal node-structure?

4. Access rights: manage and enforce

UNIX: via catalogue
MVS: via RACF

5. Records: find, read, write

i.e. extract from blocks and insert into blocks (CPU intensive!!)

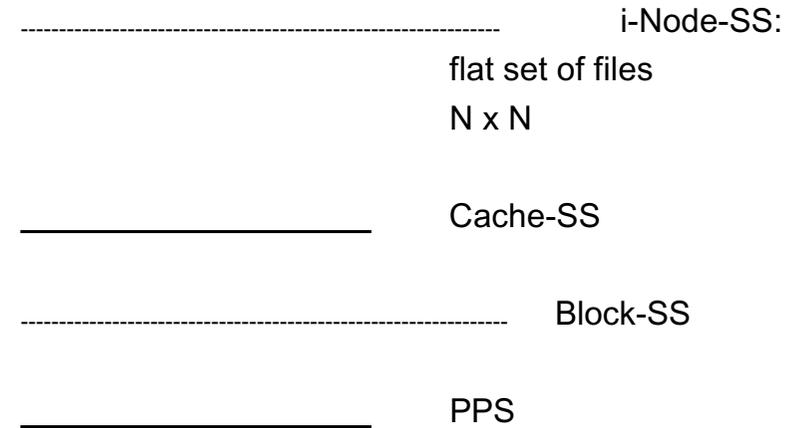
Chapter 1.3 UNIX File-System

Abstract file system

File-SS

UNIX

Directory-tree



1. PPS-Interface:

| P-block | = 512 Bytes

disk format is configurable, tendency towards ≥ 4 KB

diskread (P-block-Addr., Buffer-Addr.)

diskwrite (P-block-Addr., Buffer-Addr.)

Intermediate Consideration:

Performance and size of P-blocks

Transfer rate: 3 MB/s

average access time: 15 ms

pure transfer time for | P-block | = 512 B:

$$\frac{512 \text{ B/P-BI}}{3 \text{ MB/s}} = \frac{512 \text{ B} \cdot \text{s}}{3 \cdot 10^6 \text{ B} \cdot \text{P-BI}} \approx 0.2 \text{ ms/P-BI}$$

pure transfer time for | P-block | = 4 KB \approx 1.6 ms/P-BI

Effective performance with 512 B/P-BI:

$$\frac{512 \text{ B}}{15.2 \text{ ms}} = \frac{512 \text{ KB}}{15.2 \text{ s}} = \frac{\text{KB}}{33.7 \text{ s}} \approx 1.1\%$$

Effective performance with 4 KB:

$$\frac{4096 \text{ B}}{16.2 \text{ ms}} = \frac{4 \text{ MB}}{16.2 \text{ s}} = 252.8 \frac{\text{K}}{\text{s}} \approx 8.4\%$$

⇒ Effective performance of disks grows nearly linearly
with P-block size!!
(general formula?)

2. Block-SS:

- Objects: logical devices with dev #
~ block-sets
| block | = 512 B

- Addressing: (dev #, block #)
0, 1, 2, ...

- Operations: **strategy** for read and write
with **control-block** for parameters:

R/W-Flag

Addr. of a Cache-Block in AS

Block-Addr.

Error-Flags

Done-Flag (for asynchronous read, write: interrupt
triggered?)

open

close

mount

for exchanging of devices

3. Cache - SS

- Objects and addressing: (dev #, block #)
i.e. identical with Block-SS

- Operations:

READS : read synchronously

READA : read ahead, asynchronously
e.g. for Editor

WRITES : synchronous write, i.e. write immediately and wait

WRITEA : asynchronous write, i.e. write immediately, don't wait

WRITED : write deferred, leave open time of physical write, large
potential for optimization

FLUSH Buffer :
physical write through to disk of all pending WRITES, WRITEA, WRITED orders, which have not been completed yet.

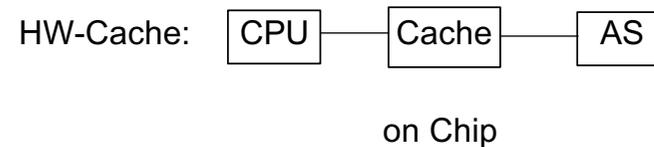
⇒ all physical operations with strategy
most frequently: READS; WRITED

Example for optimization:

Reduction of physical transports between disk and AS and of arm positioning

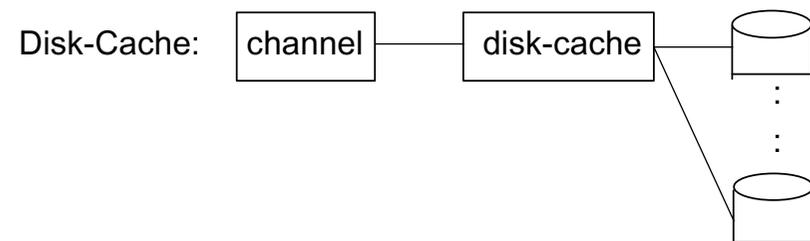
WRITED (3,17)
READS (3,17)
WRITED (3,17)
WRITEA (3,17)
WRITED (3,18)

Other Caches:



today > 95% of all storage accesses from Cache

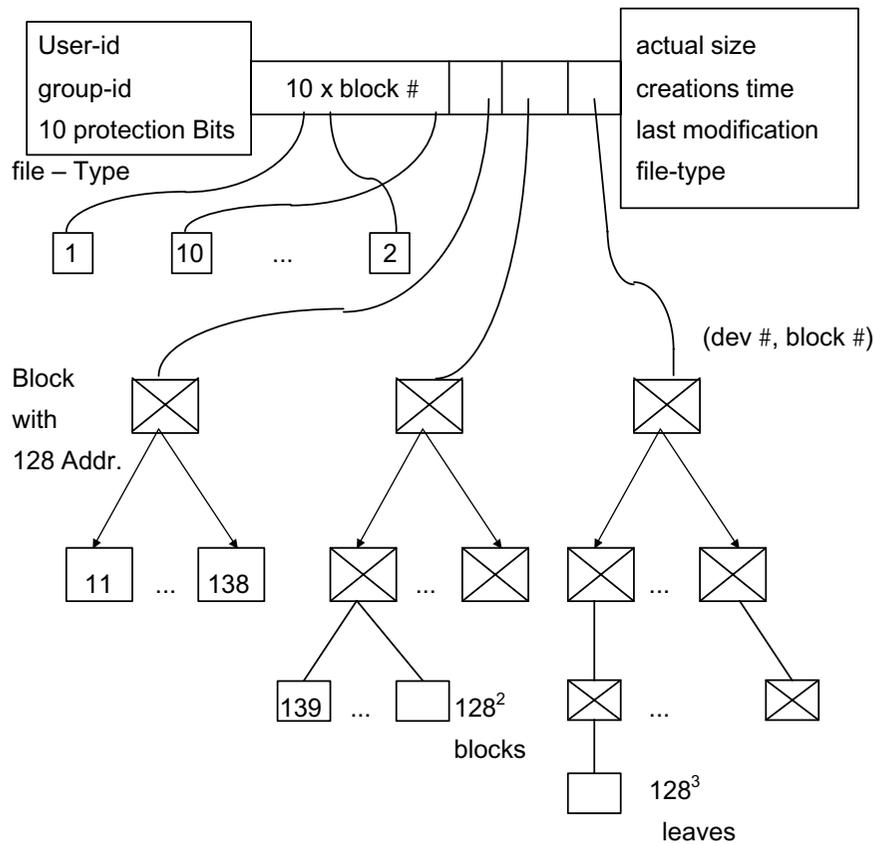
DB-Cache: see later, chapter 5.5



4. i-Node SS:

i ~ file # ~ Index of i-Node entry in special table = i-List (flat catalogue)

i - Node ~ file descriptor with fixed format:



⇒ 2 113 674 blocks
 maximal file size ~ 1 GB
 1.082.201.088 Bytes
 today by far too small e.g. for OMNIS and in general for DB applications

⇒ block assignment to files dynamically, i. e. only little physical clustering or random accesses even for sequential reading of a file (compensation by caching techniques)

- Objects: files of D-blocks
 D-block = leaf of i-Node tree

- Addressing: (i, b #)
 Dictionary: $\{(i, b\#) \rightarrow \{(dev \#, block \#)\}\}$
 evaluation via i-Node tree

- Location of i-List: fixed, known area on Device

- Operations:
 i-Nodes: create, delete, read, write
 i-Node list: manage tree storage!
 Block-assignment to files dynamic, implicate
 Evaluation of dictionary:
 e.g. i-read (7, 19) → READS (2,326)
 (i, b #) → (dev #, block #)

5. Directory and File-SS

Directory-tree above flat i-Node files. Directory is special file with references

(pointers) to sons, father, itself.

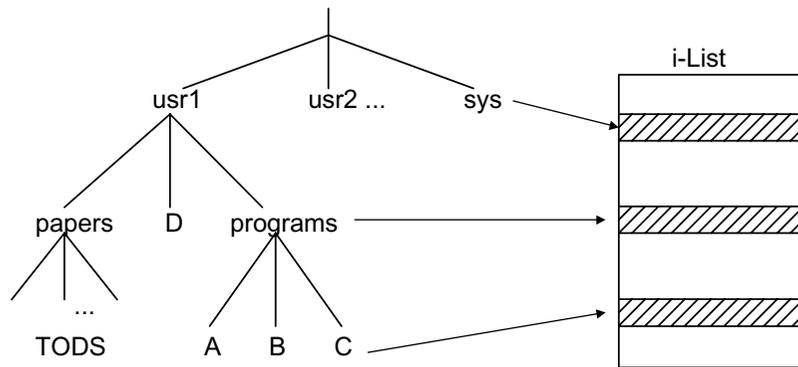
3 variants of files:

- directories
- „normal“ files
- „device files“ (devices are treated like files)

Directory Tree

Internal nodes are directories

Leaves are normal files or Devices



Path names: /usr1/papers/TODS

Current directory: e.g. /usr1

Identification of files:

- complete path names: /usr1/ papers/TODS
- rel. to current directory: papers/TODS
without / in front

Directory Structure: only references to i-Nodes, Info about file in i-Node:

| | | |
|----------|----------------|--------------------|
| • | i ₁ | { for usr1 itself} |
| • • | i ₂ | { for father-node} |
| papers | i ₃ | |
| D | i ₄ | |
| programs | i ₅ | |

content of usr1

this directory file „user 1“ has itself i-Node with # i₁

similar directories for /, usr2, sys, papers, programs

Note: i-Node tree is optimized for directories?

Sequence of Operations at Cache-SS

for /usr1/D

for READS (1,1) {read block from i-List with i-Node # 1
/-directory, location is known}
extract 1-node

→ READS {search D-blocks of /-directory for (usr1, j)}
└ READA
with Index j compute Block # J of that block of the i-List, which contains i-Node j for directory usr1.

READS (1, J)
extract i-Node with # j for usr1

→ READS {search D-blocks of usr1-directory for
└ READA entry (D, k)}
with Index k compute Block K, which contains i-Node k for File D

READS (1, K)
extract i-Node with # k for D
{up to here opening procedure for file D}

READS ... {read leaves of D, e.g. sequentially}

General Access Pattern:

for directories: 1. fetch i-Node (with direct access to i-list)
2. search leaves of directory file for desired entry

for normal files: 1. fetch i-Node
2. process leaves of i-Node tree

⇒ accesses to blocks of i-list are very frequent, caching!

⇒ i-Node itself is root of an access structure tree, needed very frequently,

where should it be stored?

- Cache
- file-management
- address space of the process who manages the file

Operations:

for dir files: <name>.dir

MKDIR

RMDIR

CHDIR

LS

for normal files:

CREATE

UNLINK update of the corresponding directories

LINK

OPEN fetch i-node

CLOSE

READ sequentially n bytes from file position,

WRITE position file show Info in i-list entry

SEEK

STAT

Access Control in UNIX:

- no passwords on file level, only via user-id and user group

- every file has owner

- 10 Protection Bits

9 are read/write/execute for owner/group/all others

1 bit for privileged programs (i.e. special rights during access to file,

only indirectly via authorized program)

Operations:

CHMOD: change access rights
(only for owner)

CHOWN: change owner

Note: DBS needs more refined access control, e.g.
single attributes
views, etc.