מערכות קבצים File Systems
נושאים

- מבוא
  - מטרת מערכת קבצים
- מנשק למערכת קבצים
  - פעולות על קבצים
  - סמנטיקה
  - ארגון קבצים
  - הגנה
- מבנה מערכות קבצים
  - תכונות בסיסיות של דיסקים
  - מיפויי קבצים בסיסיים
  - שמירה על אמינות הנתונים והתאוששות מנפילות
  - מבנים מתקדמים

All Rights Reserved - שאול קובל מערכות מחשבים
4

מושג הקובץ - File concept

- היחידה הלוגית של מערכת קבצים הוא הקובץ (file).
- קבצים לוגיים ממופעים בתוך ישויות פיזיות על ידי מערכת ההפעלה.
- במבט המשתמש, קובץ היא היחידה הקטנה ביותר הניתנת לשמירה בדיסק.
- התכונות שיכולות לכלל בקובץ הם:
  - שם (name): המספק מזהה ייחודי המוקצה לקובץ ומאפשר לתכנה לגשת אליו.
    - DOS (8 chars + 3 char extension), Windows (unlimited? length, 3 char extension)
    - UNIX (spaces tricky, no extension needed)
  - סוג (type): מסמן איך צריך לטפל בקובץ.
    - DOS/Windows rely on extension, can map extensions to programs
    - Mac associates creator attribute with each file
    - UNIX uses "magic number", first few bytes of file specify file type
  - הגנות (protections): הרשאות, מידע לגבי בקרת גישה (access control).
    - UNIX utilizes permission string: chmod 644 foo.txt → -rw-r--r--
    - owner & group: chown, chgrp
    - Windows utilizes file properties/attributes: NoAccess, List, Read, Read&Add, …
  - מיקום וגודל (location & size).
  - מידע ניהולית (accounting information).

All Rights Reserved - שאול קובל מערכות מחשבים
5

מטרת מערכת קבצים

- מטרה
  - שמירת מידע לטווח בינוני וארוך (non-volatile)
- תכונות רצויות
  - הפשטה מתכונות ספציפיות של ההתקן הפיזי
    - גישה דומה לדיסק, CD-ROM, DVD, טייפ, ...
  - זמן ביצוע סביר
  - הפרדה בין משתמשים שונים – protection
  - הגנה – security
  - ארגון נוח של הנתונים

All Rights Reserved - שאול קובל מערכות מחשבים
6

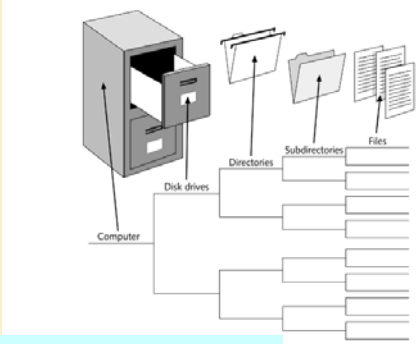
מבנה לוגי של מערכת קבצים

- אוסף של קבצים
  - כל קובץ מכיל מספר בתים/שורות/רשומות בגודל קבוע/משתנה
  - קובץ יכול להכיל מידע מטיפוס מסוים
    - בינרי,טקסט, ...

- Text
- Images
- Music

- Sounds
- Video
- Web Pages for the Internet

מבנה לוגי של מערכת קבצים



מטפורה של מערכת קבצים

פעולות בסיסיות עם קבצים

קובץ היא חבילה של סוג נתונים אבסטרקטי (Abstract Data Type - ADT) אז אנחנו מטפלים אותו דרך סדרת פעולות:

- מציאת מקום בדיסק ויצירה / השמדה של קובץ: create, delete
- קריאה/כתיבה: read, write (נדרש מיקום מדויק בתוך הקובץ)
- מיקום מחדש: repositioning
- פעולות נוספות
  - open, close
  - למה הן דרושות?
  - append, rename, copy, ...
  - ניתן לממשן בעזרת פקודות אחרות...
    - נעילה
    - עדכון תכונות הקובץ
- מערכת הפעלה חייבת לנהל מערכת מידע לגבי כל הקבצים ה"פתוחים". הטבלה עם המידע מאפשרת למערכת ההפעלה לבצע מדיניות כך שרק תהליך אחד יכול לכתבות לתוך קובץ ברגע נתון.

- file pointer : the current position of the read/write pointer in the file
- disk location : the location of the file on the disk
- file open count : keep track of number of processes currently accessing the file

תכונות קבצים (Attributes)

- לקובץ בד"כ מצמידים מספר תכונות
- **תכונות מערכת**
  - **גודל**
  - **זמן גישה אחרונה**
  - **זמן שינוי אחרון**
  - **זמן יצירה**
  - **שם (???)**
- **תכונות משתמש**
  - למשל, ב-OS/2 ניתן להצמיד לקובץ זוגות <attr_name, string> כאשר אורך המחרוזת יכול להגיע ל-2GB

מנשק למשתמש
גישה לקובץ

- גישה סדרתית
  - בד"כ מהירה יותר
  - לא מחיבת לציין מהיכן לקרוא
  - מאפשר למערכת לבצע prefetching
- גישה אקראית/ישירה
  - הגישה יכולה להיות אקראית לפי מיקום או לפי מפתח
  - לפעמים ניתן לזהות תבניות גישה

מבנה הקבצים - File structure

- **קבצים ניתנים לשמירה באופן פיזי כ:**
  - **בתים (bytes)**
  - **שורות (lines)**
  - **רקומות (records)**

- ללא חשיבות למיקום של הקובץ, מערכת הפעלה חייבת למפות לתוך אזור (sector) של הדיסק כי בדיסק פיזי, הסקטורים הם היחידה הקטנה ביותר הניתנת לכתיבה.

שיטות גישה לקבצים - access methods

- *sequential* (סדרתית): information in the file is accessed from first to last
  `readNext, writeNext, reset`
- *direct* (ישיר): possible to reposition read/write pointer to any position such files are generally made up of fixed-length records
  `readRecord N, writeRecord N, positionAt N, reset`
- *indexed* : built on top of direct access, but accesses records in file using a key each record has a key associated with it, an index of keys is stored with the file
  `readRecord KEY, writeRecord KEY, positionAt KEY, reset`

מנשק למשתמש - סמנטיקה

- מהי הסמנטיקה של פקודה?
  - מהן תוצאות הלוואי של פקודת read?
  - מתי תוצאות הכתיבה של תהליך יכולות להיראות אצל תהליכים אחרים?
  - מתי כתיבה הופכת יציבה (מגיעה לדיסק)?
  - האם מותר ליותר מתהליך אחד לכתוב בו-זמנית לאותו קובץ?
  - האם כתיבות אטומיות?
  - מה עם עדכון התכונות?
- נכונות של תכנית תלויה בהנחות על סמנטיקת מערכת הקבצים
  - סמנטיקות שונות מקשות על portability של תכניות

מנשק למשתמש
סמנטיקה – POSIX כדוגמא

- במשך השנים, התפתחו גרסאות שונות של Unix
  - לפעמים הבדלים גדולים בסמנטיקה
  - SunOS, AIX, BSD, FreeBSD, Linux, …
- גוף סטנדרטים של ה-IEEE יזם הגדרת סמנטיקה אחידה (סטנדרטית)
  - רחבה יותר מאשר מערכת הקבצים
  - כוללת ניהול תהליכים, חוטים, וכו'
  - יכולה להיתמך (חלקית) ע"י מערכות הפעלה לא Unix-יות
  - למשל Windows NT כוללת סביבה POSIX-compliant
  - חלקים של מערכת ההפעלה עשויים לא להיות compliant
  - למשל Network File System (NFS) version 3

ארגון מערכת קבצים

- מטרה
  - ברוב המערכות, מספר הקבצים מגיע לאלפים
    - ישנן מערכות שמכילות מיליארדי קבצים
  - נדרשת שיטה לארגן את הקבצים
- שיטות מקובלות
  - **מחיצות** (**partitions**) – בד"כ לפי התקנים
  - **מדריכים** (**directories**) – טבלאות הקבצים בתוך המחיצה

ארגון המדריכים - Directory structure

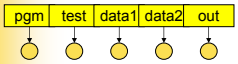
- **לניהול כמות גדול של קבצים, נחוץ מבנה מיוחד.**
- **הדיסקים הפיזיים מחולקים במחיצות (partitions).**
  - כל מחיצה משובצת (mounted) בנפרד.
  - כל מחיצה יכולה לעבוד על פי מבנה שונה (file system method) של מערכת הפעלה (FAT, NTFS, UFS, …)
  - בתוך כל מחיצה התקן של מדריכים (device directory) שומר את המידע לגבי הקבצים הנשמרים.
  - ניתן ליראות כטבלת של סימונים הממפה שמות קבצים לכניסות למדריכים.
  - פעולות המדריכים כוללים: חיפוש, יצירה, מחיקה, שינוי שם של קובץ, הצגת רשימת ממצאים, ...

פעולות על מדריכים - (directories)

- **ניתן להתייחס למדריך כמבנה נתונים מופשט עם הפעולות הבאות**
  - מציאת קובץ (לפי שם) (searching for a file).
  - יצירת כניסה (creating a file).
  - מחיקת כניסה (deleting a file).
  - קבלת רשימת הקבצים בתוך המדריך
  - החלפת שם של קובץ (renaming a file).
  - לעבור על כל קבצי המערכת (traversing the file system e.g., for backups).
  - וכו'
- **מהו המבנה של מדריך?**

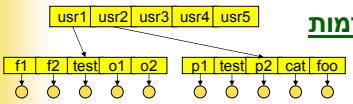
מבנה מדריכים - (directories)

- **מדריך בעל רמה אחת**



☑ מבנה פשוט וקל לביצוע
☒ מסורבל כאשר יש מספר רב של קבצים ו/או משתמשים
☒ מחייב שמות יחידיים, אפילו בין משתמשים שונים
☒ כל הקבצים שמורים באותו מדריך

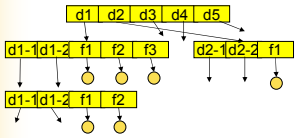
- **מדריך בעל שתי רמות**



- לכל משתמש, מדריך פרטי עם כל קבציו
  - ☑ מפריד בין משתמשים
  - ☒ מקשה אם רוצים שיתוף קבצים בין משתמשים
  - ☒ הופסק את השימוש של שיטה זו

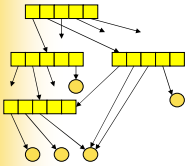
מבנה מדריכים - (directories)
עץ מכוון

- הרחבה לעץ בעל מספר רמות שרירותי (MS-DOS)
  - מלווה במושג מדריך נוכחי – current directory
  - קובץ מזוהה ע"י מסלול מהשורש (מוחלט) או מהמדריך הנוכחי (יחסי)



מבנה מדריכים - (directories)
גרף אציקלי

- בגרף אציקלי, מאפשרים למספר כניסות במדריכים (אולי שונים) להצביע לאותם העצמים (Unix)
  - קשר סימבולי (symbolic link) – כניסה שמכילה את המסלול לעצם
  - קשר חזק (hard link) – לא ניתן להבדיל בינה לבין הכניסה המקורית
- מתי נמחק קובץ?
  - קשר סימבולי אינו מונע מחיקת הקובץ
    - משאיר מצביע באוויר
  - קשר חזק מחייב מחיקת כל הכניסות
    - אבל גם זה לא מספיק...



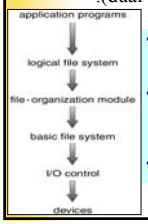
למה לא כדאי לאפשר גרף מסוג כלשהו?

מבנה מדריכים ב-Unix

- **אבחנות כלליות**
  - שם (מסלול מלא) אינו תכונה של קובץ
    - לאותו קובץ ניתן להגיע דרך מסלולים שונים
  - מחזיקים use-counter שמאפשר לדעת מתי למחוק קובץ
    - תהליך יכול ליצור קובץ, לפתוח אותו, למחוק את הכניסה (היחידה) שלו מהמדריך ולהמשיך לעבוד עליו
  - כל תהליך מחזיק מבנה נתונים לקבצים פתוחים
    - בד"כ כולל מידע על מיקום נוכחי בקובץ וכו' שאינו משותף עם פתיחות אחרות של אותו קובץ, כולל של אותו תהליך
  - ישנן תכונות גלובליות שמוחזקות בנפרד
    - זמן נגיעה אחרון, userid של בעל הקובץ, הרשאות, וכו'

הקמת מערכת קבצים - File system mounting

- *יש צורך של הקמת מערכת קבצים לפני שמערכת הפעלה יכולה לגשת לאותם קבצים*
  - מערכת הקבצים מצטרפת למערת מדריכים (directories).
  - יתרון: יש אפשרות של מספר מערכות הפעלה לסוגים שונים של קבצים כדוגמה: user accounts, system libraries, documentation, executables, ...
  - יתכנו מערכות הפעלה שונות ממוקמות במחיצות שונות (dual boot).



- UNIX: mounts are explicit – configuration file lists devices & mount points, other mounts can be executed manually
- Mac: when OS finds a disk (hard drives at boot, floppy when inserted), searches for file system and mounts at root level (& adds icon to desktop)
- Windows: maintains a 2-level directory structure – each device and partition is assigned a drive letter

מבנה פנימי של מערכת קבצים

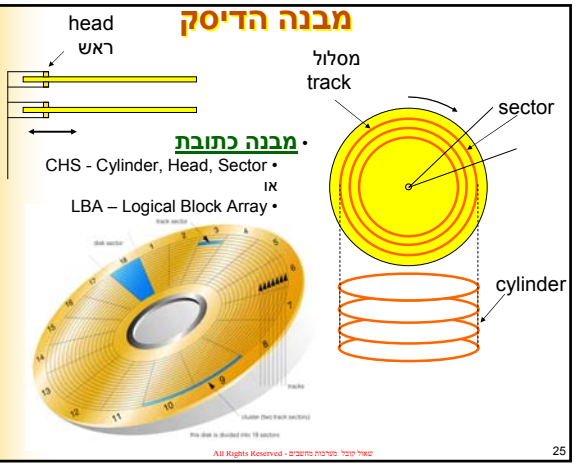
מבוא

- **נתרכז בדיסקים כהתקן המועדף לשמירת מידע online לאורך זמן**
  - גישה אקראית מהירה
  - קיבולת יחסית גבוהה
- **מערכת קבצים בד"כ שומרת מידע בבלוקים באורך קבוע**
  - כפולה של סקטורים – כאשר גודל סקטור נע בין 512B – 4KB

הגנה

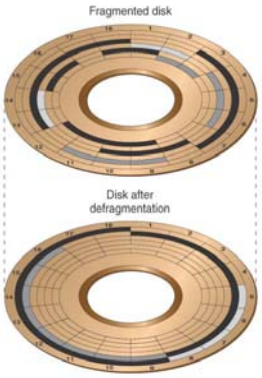
- **המטרה**
  - למנוע ממשתמש/ים מסויים/ים לבצע פעולות ספציפיות על הקובץ
    - למשל קריאה, כתיבה, ביצוע, שינוי שם, וכו'
- **רשימות גישה – Access List**
  - נרשום לכל אוביקט למי מותר לבצע מה
  - בעיה
    - הרשימות עלולות להפוך לארוכות מדי וקשות לתחזוקה
  - פיתרון
    - נקבץ משתמשים למחלקות שונות: למשל owner, group universe ומספר מוגבל של פעולות read, write, execute

מבנה הדיסק



- מבנה כתובת
  - CHS - Cylinder, Head, Sector
  - או
  - LBA – Logical Block Array

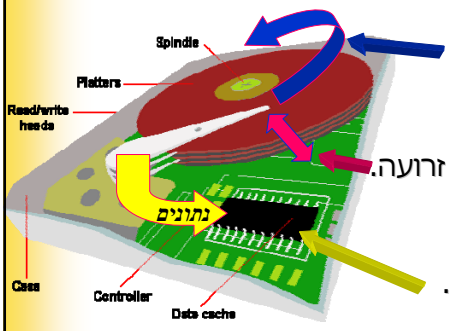
תוכניות שרות - Utilities: Service Programs



Fragmentation - the scattering of portions of files about the disk in nonadjacent areas, thus greatly slowing access to the files

Defragmenter utility - program that finds all the scattered files on your hard disk and reorganizes them as contiguous files

זמני גישה



- זמן סיבוב.
- זמן תנועת זרוע.
- זמן קריאה.

ביצועים של דיסק "שרת" דוגמא - Seagate

- r – סיבובים לדקה
- l – זמן ממוצע מהגעה ל-track עד להגעה לסקטור
- s – זמן ממוצע להזזת זרוע מ-track ל-track
- tr – קצב העברת נתונים פנימי

CHEETAH X15-36LP	36.7 GB
Model Number	ST336752LC/LW/FC
PERFORMANCE	
Spindle Speed (RPM)	15K
Latency, average (msec)	2.0
Seek Time	
Average read/write (msec)	3.6/4.2
Track-to-Track read/write (msec)	0.3/0.4
Transfer Rate	
Internal (Mbits/sec)	522–709

- שימו לב! $l = 1/(2 \cdot r)$
- זמן ממוצע לגישה לסקטור אקראי: $s + l = 3.6 + 2 = 5.6 msec$
- זמן העברה פנימי של סקטור 4KB: $4KB / tr = 45 \mu sec$

ביצועים של דיסק "מחשב שולחני" דוגמא - IBM

$s + l = 8.5 + 4.17 = 12.67 msec$

IBM Deskstar 120GXP at a glance	
Performance	
Data buffer	2MB
Rotational speed (rpm)	7,200
Latency average (ms)	4.17
Media transfer rate (max. Mbits/sec)	592
Interface transfer rate (max. MB/sec)	100
Sustained data rate (MB/sec)	48 to 23
Seek time (read, typical)[3]	
Average (ms)	8.5
Track to track (ms)	1.1
Full track (ms)	15

זמני גישה – דוגמה 1

ל- floppy disk 39 צילנדרים. זמן הסריקה הוא 6ms. זמן העברת בלוק הינו 25ms. זמן הסיבוב עד מציאת בלוק הינו 100ms. כמה זמן ייקח העברת קובץ בין 100 בלוקים המפוזר רנדומית על פני הדיסק?

100 (6 * 13 + 100 +25) = 20300ms

זמני גישה – דוגמה 2

דיסק בעל 131,072 bytes בכל מסילה. זמן הסיבוב הוא 8.33msec, זמן הסריקה הממוצע הוא 10msec. גודל בלוק הנו 2kb. מהו קצב העברת הנתונים מהדיסק?

זמני גישה – פתרון דוגמה 2

זמן קריאה של בלוק:

10 + 4.165 + (2048/131072 * 8.33) = 14.295msec

בשניה מועברים:

14.295^-1*1000 = 69.95 blocks

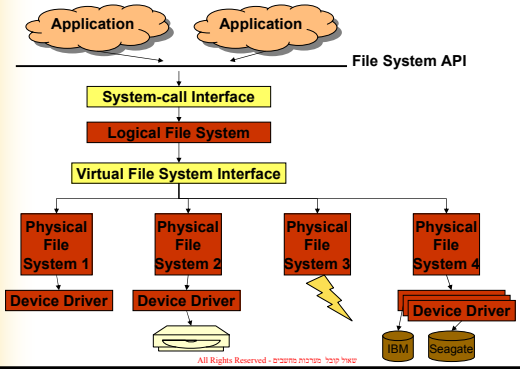
לכן קצב העברה:

69.95 * 2048 =~ 140 kb/sec

תכונות של דיסקים

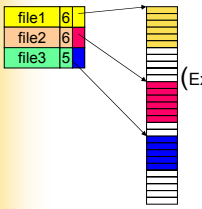
- כתובות LBA (Logical Block Array) מול CHS (Cylinder, Head, Sector)
  - ב-LBA ניתן להסתיר סקטורים לא תקינים ע"י הפניה כתיבות/קריאות לסקטורים רזרביים
- תוצאות כתיבה בעת נפילת מתח אינה אטומית (stability)
  - תוכן הסקטור שבכתיבה עלול להשתבש
- בקרים מתקדמים של דיסקים כוללים מטמון

מבנה מערכת קבצים דוגמא



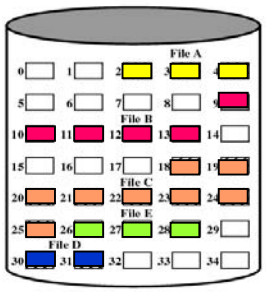
מיפוי קבצים לדיסקים הקצאה רציפה

- אפיונים
  - המשתמש מצהיר על גודל הקובץ בעת היצירה
  - מחפשים בלוקים רצופים שיכולים להכיל את הקובץ
  - הכניסה במדריך מצביעה לבלוק הראשון בקובץ
- יתרונות/חסרונות
  - גישה מהירה (סדרתית וישירה)
  - שיברור חיצוני (External fragmentation)
  - קשה להגדיל קובץ
  - יש צורך לבצע קיבוץ (compress)



מבנה מערכת קבצים דוגמא

- Logical File System
  - קטע קוד כללי, בלתי תלוי במערכת קבצים ספציפית
    - קוד שמקבל מסלול, ומחזיר את קובץ היעד
    - ניהול מידע כללי עבור קבצים פתוחים, כמו מיקום בקובץ, וכד'
- Virtual File System interface
  - מנשק אחיד לכל מערכות הקבצים הספציפיות
    - כולל ... ,vfs_read, vfs_write, vfs_seek
- Physical File System
  - מימוש מנשק ה-VFS עבור מערכת קבצים ספציפית
    - למשל, מעל דיסק, דיסקט, RAM, CD-ROM, וכו'
  - בהמשך, נתרכז בו
- Device Driver
  - קוד שיודע איך לפנות להתקן ספציפי

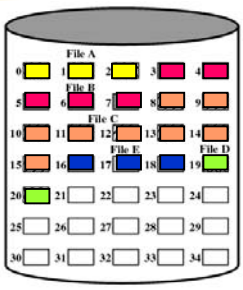


File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

הקצאה רציפה

Contiguous File Allocation



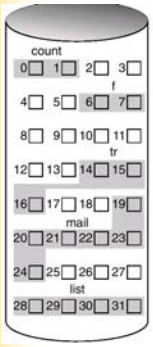
File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

הקצאה רציפה לאחר קיבוץ

Contiguous File Allocation (After Compaction)

Directory implementation: contiguous allocation



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

each file occupies a contiguous set of blocks on the disk

problems:

- managing free space is complex, external fragmentation is possible
- total space needed for file is often unknown at creation
- over-allocation leads to internal fragmentation

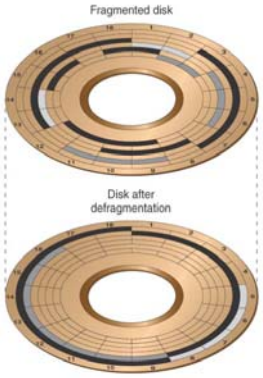
used by the Veritas File System

Contiguous Allocation

- הדירקטורי מצביע לבלוק הראשון
  Directory points contains number of first block.
- הקצעת הבלוקים:
  Block allocation
  - First fit - המתאים הראשון
  - Best fit - המתאים ביותר
  - Worst fit - הפחות מתאים

- For:
  - Fast access of files
    - File stored in contiguous blocks, minimum head movement
    - Direct access easy - easy to calculate block containing record 'n'
- Against:
  - Have to know size of file before creation
  - Fragmentation
    - compaction required
  - Extending files

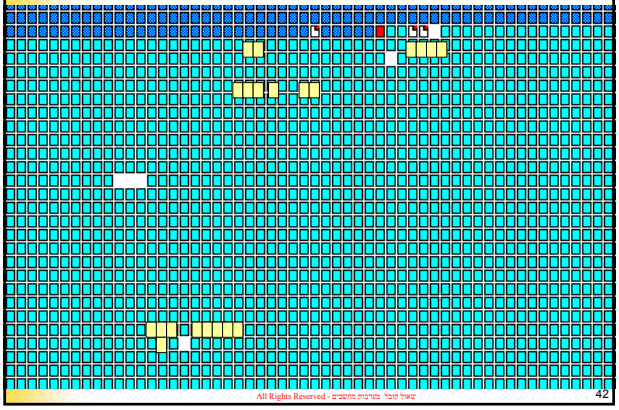
Utilities: Service Programs - תוכניות שרות



Fragmentation - the scattering of portions of files about the disk in nonadjacent areas, thus greatly slowing access to the files

Defragmenter utility - program that finds all the scattered files on your hard disk and reorganizes them as contiguous files

תפוסה של הדיסק



למה יש חלוקה רבה בין חלקים שונים של קובץ השמור בדיסק ??

How disks become Fragmented

ולמה יש צורך לבצע איחוי של אותם חלקים ??

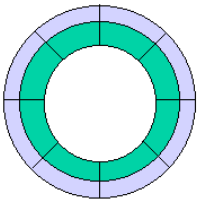
why you should periodically Defragment

43

טבלה לניהול מקום בדיסק או דיסקט

File Allocation Table

Filename	Start	End	Extent

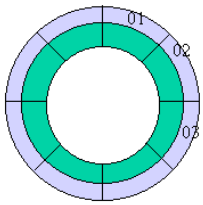


44

טבלה לניהול מקום בדיסק או דיסקט

File Allocation Table

Filename	Start	End	Extent
WORD.EXE	01	03	

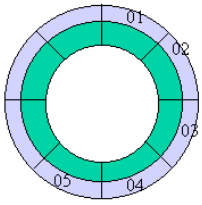


45

טבלה לניהול מקום בדיסק או דיסקט

File Allocation Table

Filename	Start	End	Extent
WORD.EXE	01	03	
CLA1-3.DOC	04	05	

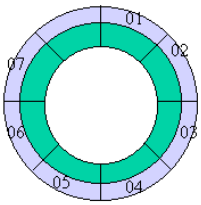


46

טבלה לניהול מקום בדיסק או דיסקט

File Allocation Table

Filename	Start	End	Extent
WORD.EXE	01	03	
CLA1-3.DOC	04	05	
CLA1-4.DOC	06	07	

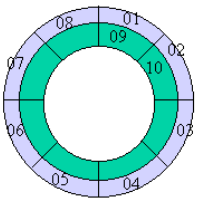


47

טבלה לניהול מקום בדיסק או דיסקט

File Allocation Table

Filename	Start	End	Extent
WORD.EXE	01	03	
CLA1-3.DOC	04	05	
CLA1-4.DOC	06	07	
CLA2-4.XLS	08	10	

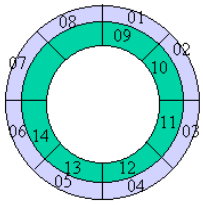


48

טבלה לניהול מקום בדיסק או דיסקט

File Allocation Table

Filename	Start	End	Extent
WORD.EXE	**01**	**03**	
CLA1-3.DOC	04	05	
CLA1-4.DOC	06	07	
CLA2-4.XLS	08	10	
POWERPT.EXE	11	14	



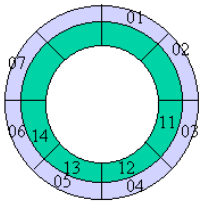
49

טבלה לניהול מקום בדיסק או דיסקט

File Allocation Table

Delete file CLA2-4.XLS

Filename	Start	End	Extent
WORD.EXE	**01**	**03**	
CLA1-3.DOC	04	05	
CLA1-4.DOC	06	07	
	08	10	
POWERPT.EXE	11	14	



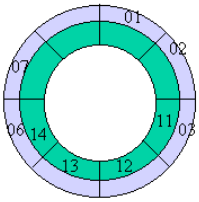
50

טבלה לניהול מקום בדיסק או דיסקט

File Allocation Table

Delete file CLA1-3.DOC

Filename	Start	End	Extent
WORD.EXE	**01**	**03**	
	04	05	
CLA1-4.DOC	06	07	
	08	10	
POWERPT.EXE	11	14	



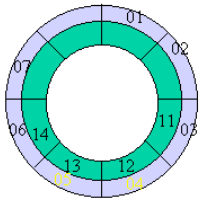
51

טבלה לניהול מקום בדיסק או דיסקט

File Allocation Table

Save ZIP.EXE 6 sectors required

Filename	Start	End	Extent
WORD.EXE	**01**	**03**	
ZIP.EXE	04	05	
CLA1-4.DOC	06	07	
	08	10	
POWERPT.EXE	11	14	



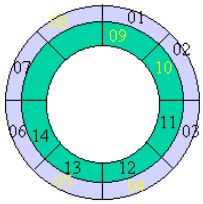
52

טבלה לניהול מקום בדיסק או דיסקט

File Allocation Table

Save ZIP.EXE 6 sectors required

Filename	Start	End	Extent
WORD.EXE	**01**	**03**	
ZIP.EXE	04	05	YES
CLA1-4.DOC	06	07	
ZIP.EXE	08	10	
POWERPT.EXE	11	14	



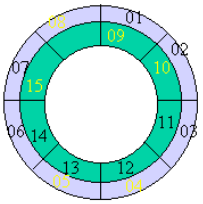
53

טבלה לניהול מקום בדיסק או דיסקט

File Allocation Table

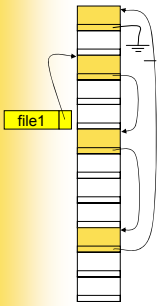
Save ZIP.EXE 6 sectors required

Filename	Start	End	Extent
WORD.EXE	**01**	**03**	
ZIP.EXE	04	05	YES
CLA1-4.DOC	06	07	
ZIP.EXE	08	10	YES
POWERPT.EXE	11	14	
ZIP.EXE	15	15	

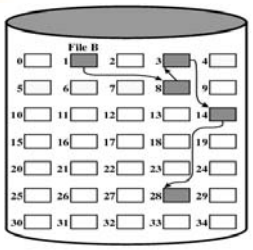


54

מיפוי קבצים לדיסקים
הקצאה משורשרת

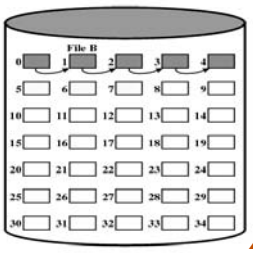


- איפיונים
  - כל בלוק מצביע לבלוק הבא
  - הכניסה במדריך מצביעה לבלוק הראשון בקובץ
- יתרונות/חסרונות
  - קל להגדיל קובץ
  - גישה איטית, בעיקר לגישה ישירה
  - שימוש בבלוקים גדולים עשוי להקל, אך יגרום לשיברור פנימי
  - שיבוש מצביע בבלוק יגרום לאיבוד חלקים רחבים של קובץ
- וריאציה
  - החזקת שרשרת המצביעים בנפרד כמו ב-MS-DOS
  - File Allocation Table (FAT)



הקצאה משורשרת

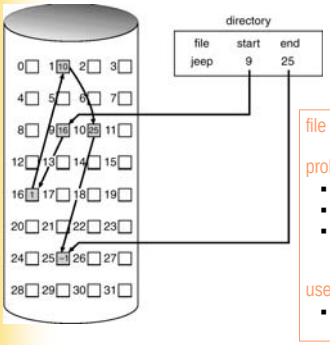
Figure 12.9 Chained Allocation



הקצאה משורשרת
לאחר קיבוץ

Chained Allocation (After Consolidation)

Directory implementation: linked allocation



file is a linked list of disk blocks

problems:

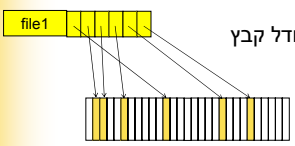
- only supports sequential access
- extra space is needed for pointers
- a single bad sector can corrupt an entire file

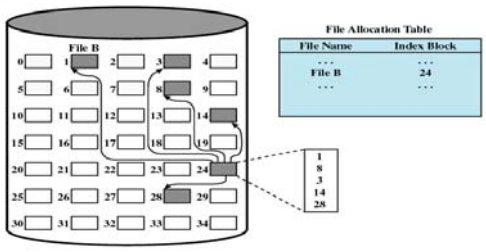
used in MS-DOS & Windows

- File-Allocation Table (FAT) is stored in first block of the partition

מיפוי קבצים לדיסקים
שימוש באינדקס

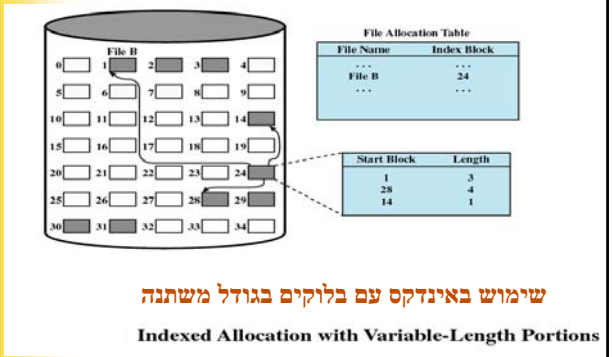
- איפיונים
  - המשתמש מצהיר על גודל קובץ מקסימלי
  - המצביעים לבלוקים מרוכזים בשטח רצוף
- יתרונות/חסרונות
  - דומה להקצאה משורשרת, אך משפר את זמן הגישה הישירה
  - צריך להצהיר מראש על גודל קבץ





שימוש באינדקס עם בלוקים בגודל קבוע

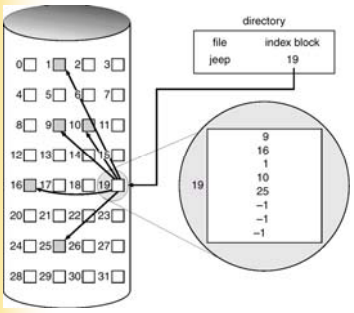
Indexed Allocation with Block Portions



שימוש באינדקס עם בלוקים בגודל משתנה

Indexed Allocation with Variable-Length Portions

Directory implementation: indexed allocation



for each file, an index block stores pointers to all the blocks

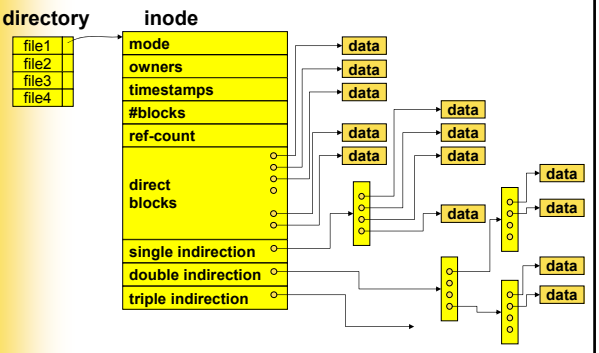
problems:

- index block requires space
- large file may need several index blocks
- growing files may need index blocks dynamically extended

used in Unix File System

- uses multiple levels of indirection

מיפוי קבצים לדיסקים
שימוש באינדקס – Combined Multilevel



מיפוי קבצים לדיסקים
Unix - SOLARIS

- מאפיינים
  - Combined Multilevel index
  - בלוקים פנויים – מחזיקים free block bitmap
- מאפשר לזהות בקלות בלוקים פנויים רצופים

ניהול בלוקים פנויים - SOLARIS

- **רשימה מקושרת**
  - מציאת בלוק פנוי מהירה
  - מבנה נתונים שברירי אם בלוק באמצע הרשימה השתבש
  - הקצאת בלוקים המרוחקים זה מזה לאותו קובץ
- **bitmap עם סיבית לכל בלוק**
  - 0 – הבלוק תפוס; אחרת הוא פנוי
  - מקל על מציאת מספר בלוקים פנויים רצופים
- **הקבצה**
  - שימוש ברשימה מקושרת של בלוקים, כאשר כל בלוק מכיל טבלה של מצביעים לבלוקים הפנויים
  - מאפשרת למצוא בצורה יעילה מספר גדול של בלוקים פנויים ורציפים

אמינות - SOLARIS

- **המידע בדיסק מתחלק ל-**
  - user data – נתוני המשתמש (בתוך הקבצים)
  - metadata – נתונים על ארגון הקבצים
    - inode, block tables, *etc.*
- **איבוד/שיבוש ה-metadata** עלול לגרום לאיבוד user data רב
  - זיכרו! נפילת חשמל באמצע כתיבה עלולה לשבש את הסקטור שכעת נכתב
- **המצב מחמיר בגלל שימוש במטמון במספר רמות**
  - בקר הדיסק, מטמון סקטורים/בלוקים בזיכרון הראשי
- **מתי כתיבות יורדות מזיכרון מטמון לדיסק?**
  - write-through – כל כתיבה יורדת מיידית לדיסק
  - write-back – עדכון בדיסק נעשה אסינכרונית, ללא שמירה על סדר בהכרח

אמינות נתוני המשתמש
מערכות Unix - SOLARIS

- במערכות Unix משתמשים במדיניות write-back
  - נתוני המשתמש יורדים באופן מחזורי לדיסק
  - תואם את סמנטיקת POSIX
  - קיימות פקודות fsync, sync אשר מאלצות לכתוב את הבלוקים המלוכלכים לדיסק

אמינות ה-metadata

- משתמשים במדיניות write-through
  - עדכונים יורדים מידית לדיסק
- לגבי נתונים מסויימים, לא נבצע כתיבה in-place
  - מבטיח קיום גרסא במקרה של נפילה
  - version number מאפשר לדעת מהי הגרסא העדכנית
- נתונים מסויימים נשמרים במספר עותקים
  - למשל, שורש ה-file system
- כאשר מערכת הקבצים עולה אחרי נפילה, מתקנים את מבני הנתונים
  - במערכות קבצים מסויימות, זה דורש מעבר על כל הדיסק
    - ScanDisk, fsck (file system check)

אמינות ה-metadata
שימוש ב-logging

- שיטה יעילה לתחזוקת ה-metadata
  - מוכרת גם כ-journaling
    - Journaled File System
- מאפיינים
  - רושמים ב-log סדרתי את העדכונים על ה-metadata לפני שהם נכתבים לדיסק
    - Write-ahead logging
  - לאחר מכן, הבלוקים שהשתנו יכולים להיכתב לדיסק
    - ייתכן אפילו לא לפי הסדר
  - ניתן לקבץ מספר שינויים ולכתוב אותם בכתיבה אחת
  - עדכונים שירדו לדיסק ניתן למחוק אותם מה-log

אמינות ה-metadata
שימוש ב-logging *(המשך)*

- התאוששות מנפילה
  - עוברים על כל כניסה ב-log
  - בצע את הפעולה
    - ביצוע נוסף של פקודה שהתבצעה במלואה או חלקית נותן תוצאה שקולה לביצוע הפעולה המקורית (idempotent)
  - אם הכניסה האחרונה ב-log אינה שלמה, מתעלמים ממנה

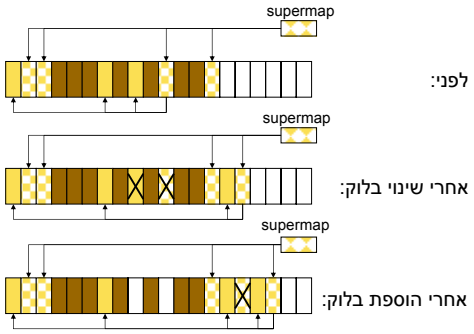
אמינות ה-metadata
שימוש ב-logging *(המשך)*

- יתרונות
  - Metadata יכול להכתב בצורה אסינכרונית - יעיל
  - התאוששות יעילה
    - תלויה במספר השינויים שלא ירדו לדיסק, ולא בגודל מערכת הקבצים
- חסרונות
  - דורש כתיבות נוספות

מערכת קבצים log-structured

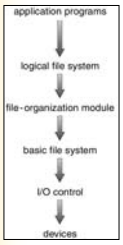
- מאפיינים
  - מתייחסים ל-data blocks כ-log סדרתי של כתיבות
  - כתיבה של data block (חדש או ישן) נעשה תמיד בסוף ה-log
    - ולכן העותק הישן של הבלוק לא תקף
  - ניתן גם לכתוב metadata blocks ב-log
- יתרונות
  - הכתיבות הן סדרתיות (ולכן יעילות יותר)
  - בכתיבה סדרתית של קובץ, סיכוי טוב שהבלוקים של הקובץ יהיו רצופים
- חסרונות
  - איסוף הבלוקים אשר מתפנים
  - הזזת הבלוקים התפוסים לתחילת ה-log

מערכת קבצים log-structured



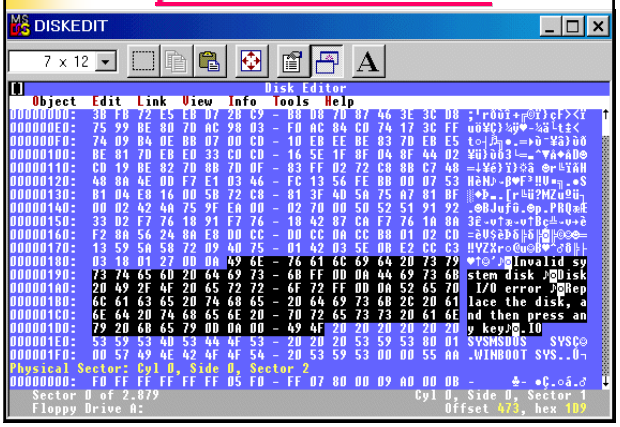
יישומים נוספים במערכת הקבצים
File system implementation

file system is generally composed of many levels



- *logical file system* : manages directories, manipulates files as ADT's
- *file organization module* : maps logical block addresses to physical blocks, manages free space
- *basic file system* : issues generic commands to read/write physical blocks
- *I/O control* : consists of device drivers and interrupt handlers to transfer info between RAM and disk

נתונים באזורים בדיסק



Implementing the directory

- need to store a list of directories/files and their locations in memory

linear list
- simple implementation, but requires sequential search
- if contiguous (e.g., array/vector), deletions require shifting
- can improve efficiency with caching

hash table
- more complex, but can improve performance

Performance - שיפור בביצוע

- performance depends greatly on how the system is used
- contiguous allocation
  - only one access required since can calculate where any particular block is
- linked allocation
  - since sequential, accessing Nth block requires N accesses
- indexed allocation
  - if index is in memory, 2 accesses required (but index might be very large)
- some systems optimize by using different schemes for different file types
  - direct access files use contiguous; sequential access files use linked
  - small files use contiguous; large files use indexed

Free space management

- the OS needs to keep track of free blocks on the disk
  - bit vector
    - use a bit string (Nth block is free/allocated, Nth bit is 1/0)
    - e.g., blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 and 27 are free
      `00111100111111000110000001110000 0`
    - simple, but bit vector can be very large & needs to be in memory
  - linked list
    - use a linked list of free blocks
    - slow and inefficient since it can't be easily cached
  - grouping
    - use a free block as an index to a group of free blocks

Efficiency and performance

- efficient use of disk space is dependent on design choices, actual data
  e.g., consider the choice of pointer size in directories, most use 16-bit or 32-bit pointers → file size limited to 2^{16} (64K) or 2^{32} (4G) blocks
  64-bit pointers can be used to handle larger files, but take up more space
- caching can improve performance
  - cache an entire track, then read sectors from RAM
  - cache requires effort to maintain consistency
- hidden inefficiencies
  - maintaining "last access time" requires that every access to the file also requires an update to the directory, requiring an extra disk access

Recovery

- all cached information needs to be refreshed on the disk to avoid losing data during crashes
  - when crash occurs, the table of opened files may be lost
  - directory may no longer accurately reflect the file states
- a *consistency checker* compares the directory with the actual blocks
  - if inconsistent, tries to fix it (e.g., if linked allocation, traverse to reconstruct)
  - UNIX runs `fsck` to do a file system check
- some steps can be taken to avoid consistency problems
  e.g., UNIX caches directory entries for reads, but writes are updated synchronously

Case study: Windows XP file system

- on readable/writable disks, XP supports FAT and NTFS
  - File Allocation Table (FAT) is simpler, backward compatible
  - NTFS supports large drives (> 32 GB)
    - security features: encryption, file/directory protections, transaction logging, …
    - advanced features: quotas, compression, mounted drives, …

operating system and file system compatibility

Operating System	FAT16	FAT32	NTFS
Windows XP	•	•	•
Windows 2000	•	•	•
Windows NT 4.0[1]	•		•
Windows 95 OSR2, Windows 98, and Windows Me	•	•	
Windows 95 (prior to OSR2)	•		
MS-DOS	•		

Windows XP: clustering

- **The FAT disk format is organized into sectors.Each sector can store 512 bytes.**
- **A sector is the smallest unit used when transferring data.**
- **a *cluster*** is the smallest amount of disk space that can be allocated to a file
  smaller cluster size → more efficient use of disk (less fragmentation)
  more storage for directory (more, longer addresses)
  administrator can specify cluster size when volume/partition is formatted
  - otherwise, default cluster sizes are used

Volume Size	FAT16 Cluster Size	FAT32 Cluster Size	NTFS Cluster Size
7 MB–16 MB	2 KB	Not supported	512 bytes
17 MB–32 MB	512 bytes	Not supported	512 bytes
33 MB–64 MB	1 KB	512 bytes	512 bytes
65 MB–128 MB	2 KB	1 KB	512 bytes
129 MB–256 MB	4 KB	2 KB	512 bytes
257 MB–512 MB	8 KB	4 KB	512 bytes
513 MB–1,024 MB	16 KB	4 KB	1 KB
1,025 MB–2 GB	32 KB	4 KB	2 KB
2 GB–4 GB	64 KB	4 KB	4 KB
4 GB–8 GB	Not supported	4 KB	4 KB
8 GB–16 GB	Not supported	8 KB	4 KB
16 GB–32 GB	Not supported	16 KB	4 KB
32 GB–2 terabytes	Not supported	Not supported[1]	4 KB

Windows XP: size limits

FAT16

Description	Limit
Maximum file size	4 GB minus 1 byte (2^{32} bytes minus 1 byte)
Maximum volume size	4 GB
Files per volume	Approximately 65,536 (2^{16} files)
Maximum number of files and folders within the root folder	512 (Long file names can reduce the number of available files and folders in the root folder.)

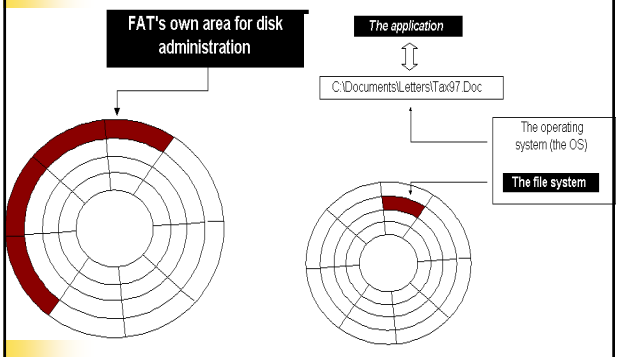
FAT32

Description	Limit
Maximum file size	4 GB minus 1 byte (2^{32} bytes minus 1 byte)
Maximum volume size	32 GB (implementation)
Files per volume	4,177,920
Maximum number of files and subfolders within a single folder	65,534 (The use of long file names can significantly reduce the number of available files and subfolders within a folder.)

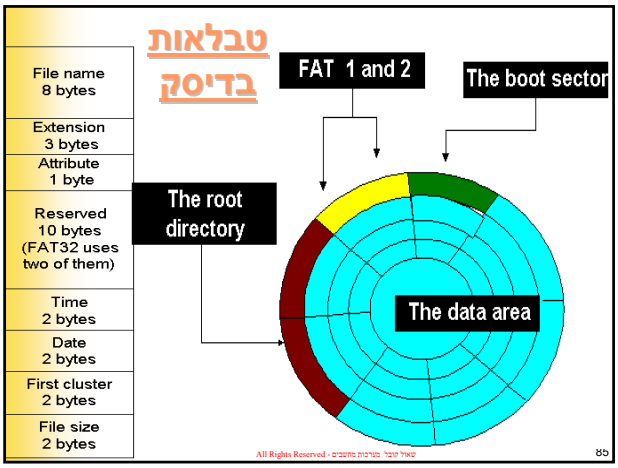
NTFS

Description	Limit
Maximum file size	Theory: 16 exabytes minus 1 KB (2^{64} bytes minus 1 KB) Implementation: 16 terabytes minus 64 KB (2^{44} bytes minus 64 KB)
Maximum volume size	Theory: 2^{64} clusters minus 1 cluster Implementation: 256 terabytes minus 64 KB (2^{32} clusters minus 1 cluster)
Files per volume	4,294,967,295 (2^{32} minus 1 file)

טבלאות FAT בדיסק



טבלאות בדיסק



וקלוסטרים בדיסק טבלאות, סקטורים

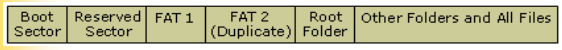
0	1 - 168	169 - 316	317 - 348	349 - 323,548	Sector number
Boot record	FAT number 1	FAT number 2	Root directory with 512 entries of 32 bytes	Data area divided into 40,400 clusters, each of 8 sectors. First cluster is number 2	Containing

CLUSTERS

Sector number:	349 - 356	357 - 364	365 - 372	...	323,541 - 323,548
Contains:	Cluster 2	Cluster 3	Cluster 4	...	Cluster 40,400

Windows XP: FAT

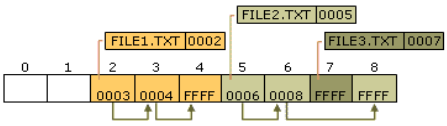
- File Allocation Table (FAT) file systems
  - locates the file allocation table near the beginning of the volume
  - the location of the FAT is specified in the boot sector (BIOS Parameter Block)
  - actually, 2 copies of the FAT are stored for redundancy
  - the FAT number refers to the number of bits per table entry
    - FAT12 → 2^{12} = 4M different clusters can be addressed (used for floppy disks)
    - FAT16 → 2^{16} = 64M different clusters can be addressed (MS-DOS compatible)
    - FAT32 → 2^{28} = 256G different clusters (4 bits are reserved)



- organization of a FAT partition
  each FAT entry identifies a cluster as BAD, UNUSED, USED-IN-FILE, END-OF-FILE

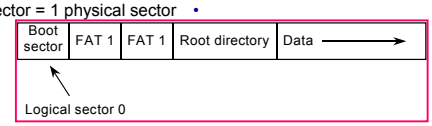
Windows XP: FAT (cont.)

- FAT utilizes a lined allocation scheme
  - each cluster contains a pointer to the next cluster in the file
  - a pointer with value (0xFFFF) marks the last cluster in the file
  - FAT stores the file attributes (name, date, type, …) & pointer to first cluster on disk



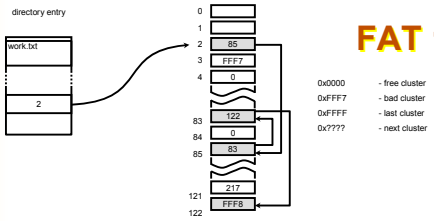
FAT 16

MS-DOS, Win95/98/NT/2000
Partition layout - set of logical sectors
1 logical sector = 1 physical sector



- Disc space allocation unit = cluster
  - cluster = 2^n sectors
- Disc allocation/management = variant of linked allocation - file allocation table
- FAT
  - array of cluster numbers
  - one entry per cluster
  - entry = 0 - cluster free
- FAT cached in memory

FAT 16



- Cluster number = 16 bits - problems? **FAT 16 Limitations?**
  - Limits number of clusters
  - Limits number of files - >= 1 cluster/file
  - As partition gets bigger so does cluster size leading to wasted space in clusters
- Limited root directory - 512 entries
- No security - user access
- No error recovery - extra FAT not used

מבנה של FAT16

Boot sector
on system (active) partition

File Allocation Table (FAT)
primary

File Allocation Table (FAT)
copy for fault tolerance

Root folder
fixed location and length (512 entries long)

Other folders and all files

FAT 32

- Win95 (OSR 2), NT + driver, Win2000
- 32 bit cluster numbers
  - more files
  - smaller clusters - less wasted space
  - bigger partitions
- No security - user access
- Root directory ordinary cluster chain - no limit on size
- Limited error recovery
  - can use backup copy of FAT

מבנה של FAT32

Boot sector points to
the first cluster of the root folder.

Root folder can be located anywhere on disk,
boot sector points to it. Limit to 65,535 entries.

File Allocation Table (FAT)
primary

File Allocation Table (FAT)
secondary–mirroring of primary can be disabled for performance.

Other folders and all files
varies

Windows XP: NTFS

- NTFS is the preferred file system for Windows XP
  - can set permissions to allow users/groups to share files, control types of actions
  - can encrypt a file/folder using private and public key encryption
  - can compress a file/folder, any Windows app will automatically expand as needed
  - can enforce quotas on disk usage
  - can mount new drives; create hard links to shared files/folders
  - recovery features: each file operations broken down into atomic transaction
    - maintains a transaction log – updates disk after each transaction
    - if failure occurs during a transaction, info is sufficient to complete or rollback
    - if a bad sector is found when writing, will automatically map to a different sector
- NTFS utilizes
  Master File Table (MFT): a relational database that stores file records & attributes
  metadata files: are contained within the MFT to store relevant info and attributes

Windows XP: metadata files in the MFT

System File	File Name	MFT Record	Purpose of the File
Master file table	$Mft	0	Contains one base file record for each file and folder on an NTFS volume. If the allocation information for a file or folder is too large to fit within a single record, other file records are allocated as well.
Master file table mirror	$MftMirr	1	Guarantees access to the MFT in case of a single-sector failure. It is a duplicate image of the first four records of the MFT.
Log file	$LogFile	2	Contains a list of transaction steps used for NTFS recoverability. The log file is used by Windows XP Professional to restore consistency to NTFS after a system failure. The size of the log file depends on the size of the volume, but you can increase the size of the log file by using the Chkdsk command. For more information about the log file, see "NTFS Recoverability" earlier in this chapter. For more information about Chkdsk, see "Troubleshooting Disks and File Systems" in this book.
Volume	$Volume	3	Contains information about the volume, such as the volume label and the volume version.
Attribute definitions	$AttrDef	4	Lists attribute names, numbers, and descriptions.
Root file name index	.	5	The root folder.
Cluster bitmap	$Bitmap	6	Represents the volume by showing free and unused clusters.
Boot sector	$Boot	7	Includes the BPB used to mount the volume and additional bootstrap loader code used if the volume is bootable.
Bad cluster file	$BadClus	8	Contains bad clusters for a volume.
Security file	$Secure	9	Contains unique security descriptors for all files within a volume.
Upcase table	$Upcase	10	Converts lowercase characters to matching Unicode uppercase characters.
NTFS extension file	$Extend	11	Used for various optional extensions such as quotas, reparse point data, and object identifiers.
		12–15	Reserved for future use.

Windows XP: NTFS (cont.)

- similar to FAT, the location of the MFT and backup MFT are stored in the boot sector
  - the MFT contains a record for each file
  - record contains links to cluster groupings for that file
  - if too large & fragmented, utilizes indexing (record points to external index block)
- for large folders, contents are organized into a B-tree (balanced tree structure) to optimize searches