Operating System

Nguyen Tri Thanh

ntthanh@vnu.edu.vn

File System Implementation

File-System Structure

File-System Implementation

Directory Implementation

Allocation Methods

Free-Space Management

Efficiency and Performance

Recovery

Objectives

- Introduce what a file is
- Introduce file system implementation
- Introduce directory implementation
- Introduce 3 allocation methods
- Introduce free space management

Reference

- Chapter 10, 11 of Operating System Concepts

File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
  - Operating system
  - Program

File Attributes

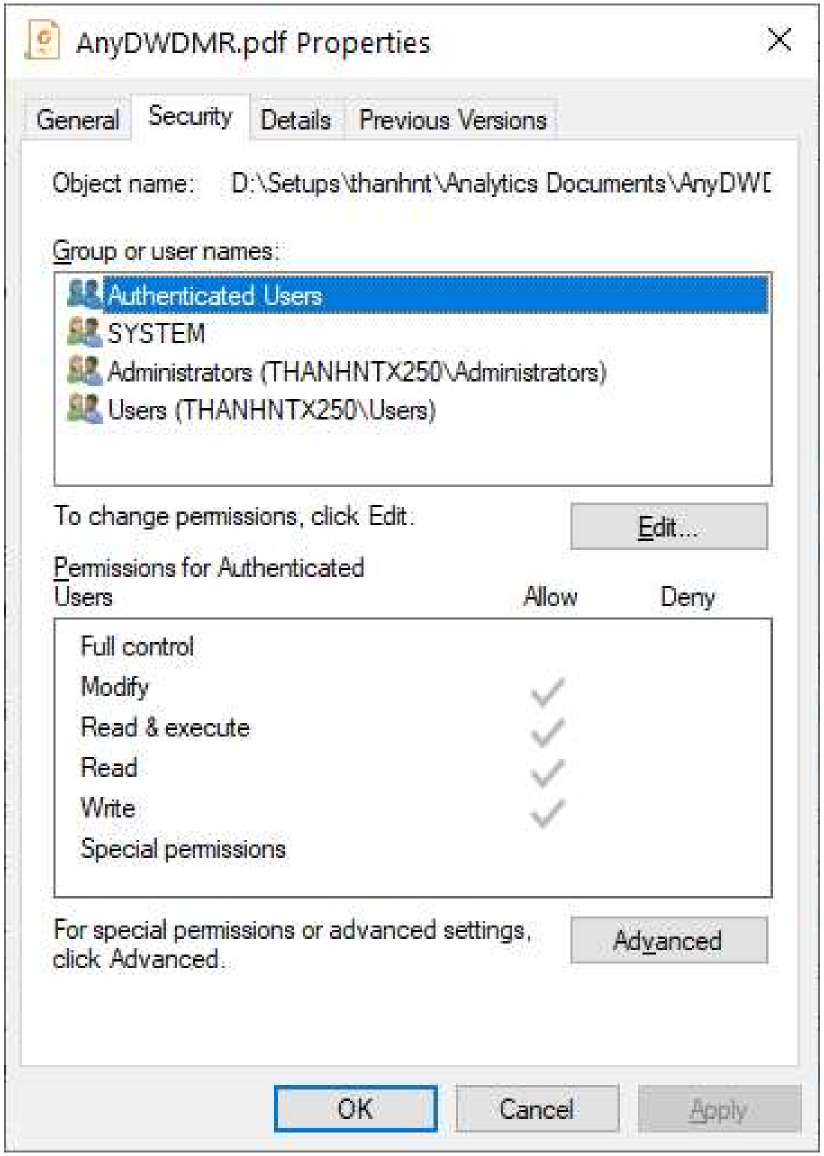
- Name
  - only information kept in human-readable form
- Identifier
  - unique (number) identifies file within file system
- Type
  - needed for systems that support different types
- Location
  - pointer to file location on storage device

File Attributes (cont’d)

- Size
  - current file size
- Protection
  - controls who can do reading, writing, executing
- Time, date, and user identification
  - data for protection, security, and usage monitoring

- Information about files are kept in the directory structure on the disk

File Attributes (cont’d)



Object name: D:\Setups\thanhnt\Analytics Documents\AnyDWI

Group or user names:

- Authenticated Users
- SYSTEM
- Administrators (THANHNTX250\Administrators)
- Users (THANHNTX250\Users)

To change permissions, click Edit.

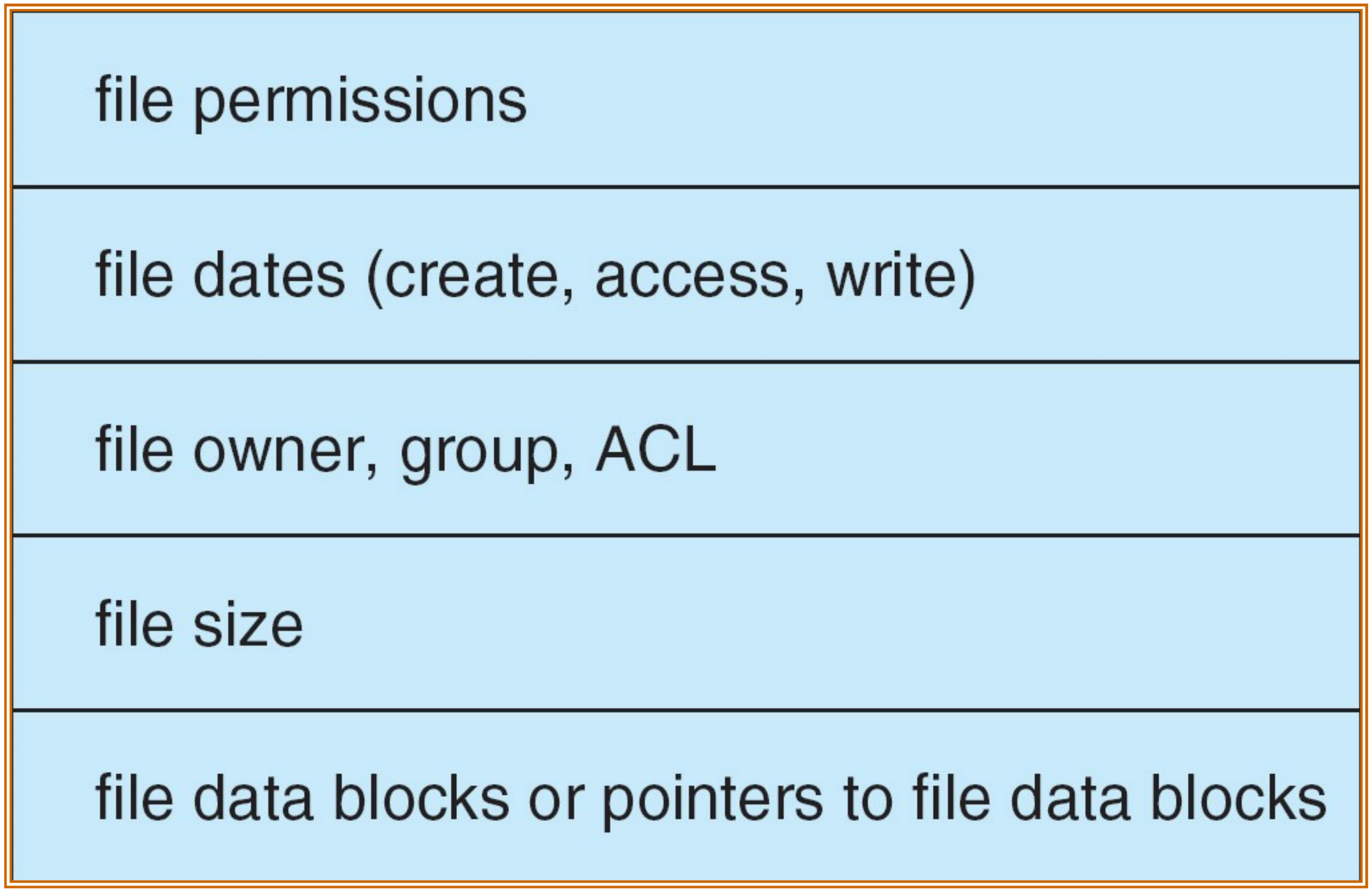
Permissions for Authenticated Users	Allow	Deny
Full control		
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced.

File System Structure

- File system resides on secondary storage
  - HDD disks, CD ROM, DVD, flash drive, SSD, …
- File system organized into layers
- File control block (FCB)
  - storage structure of information about a file
  - *inode* (index node) has partial information of FCB on Linux

A Typical File Control Block



file permissions

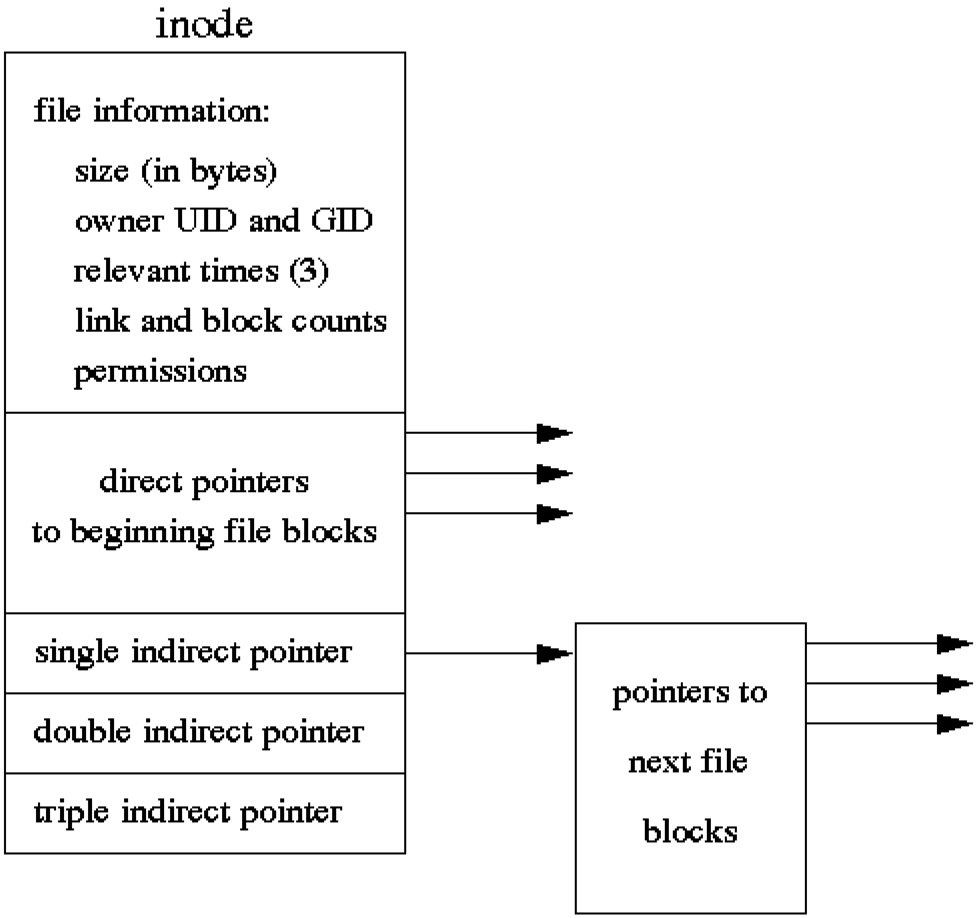
file dates (create, access, write)

file owner, group, ACL

file size

file data blocks or pointers to file data blocks

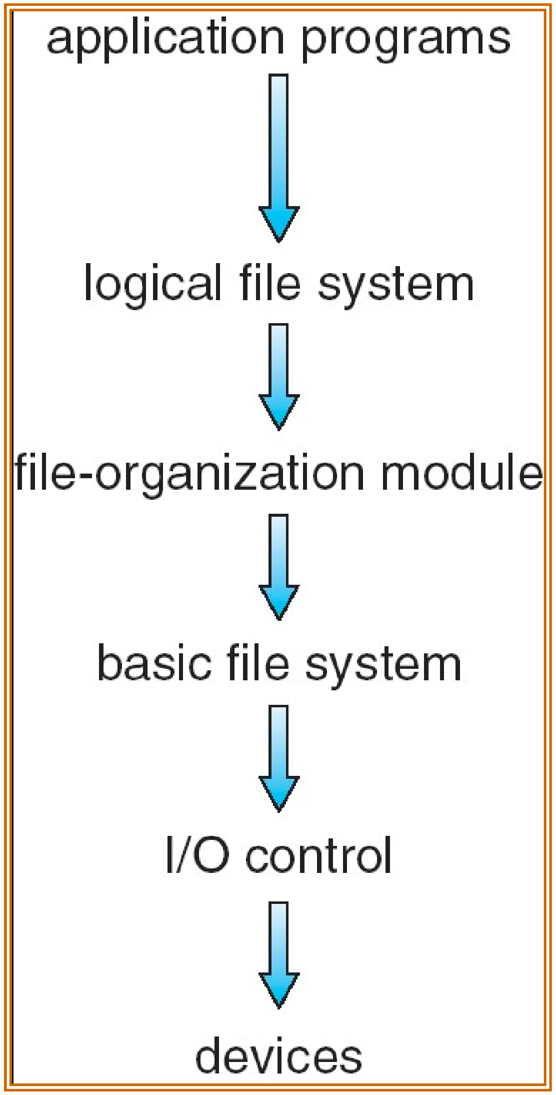
Inode on Linux



Question

- Which of the following is incorrect about file control block (FCB)?
  A. a data structure containing information of a file
  B. OS needs a FCB to access a file
  C. FCB of a file is updated when a file is accessed
  D. FCBs reside in memory

Layered File System



application programs

logical file system

file-organization module

basic file system

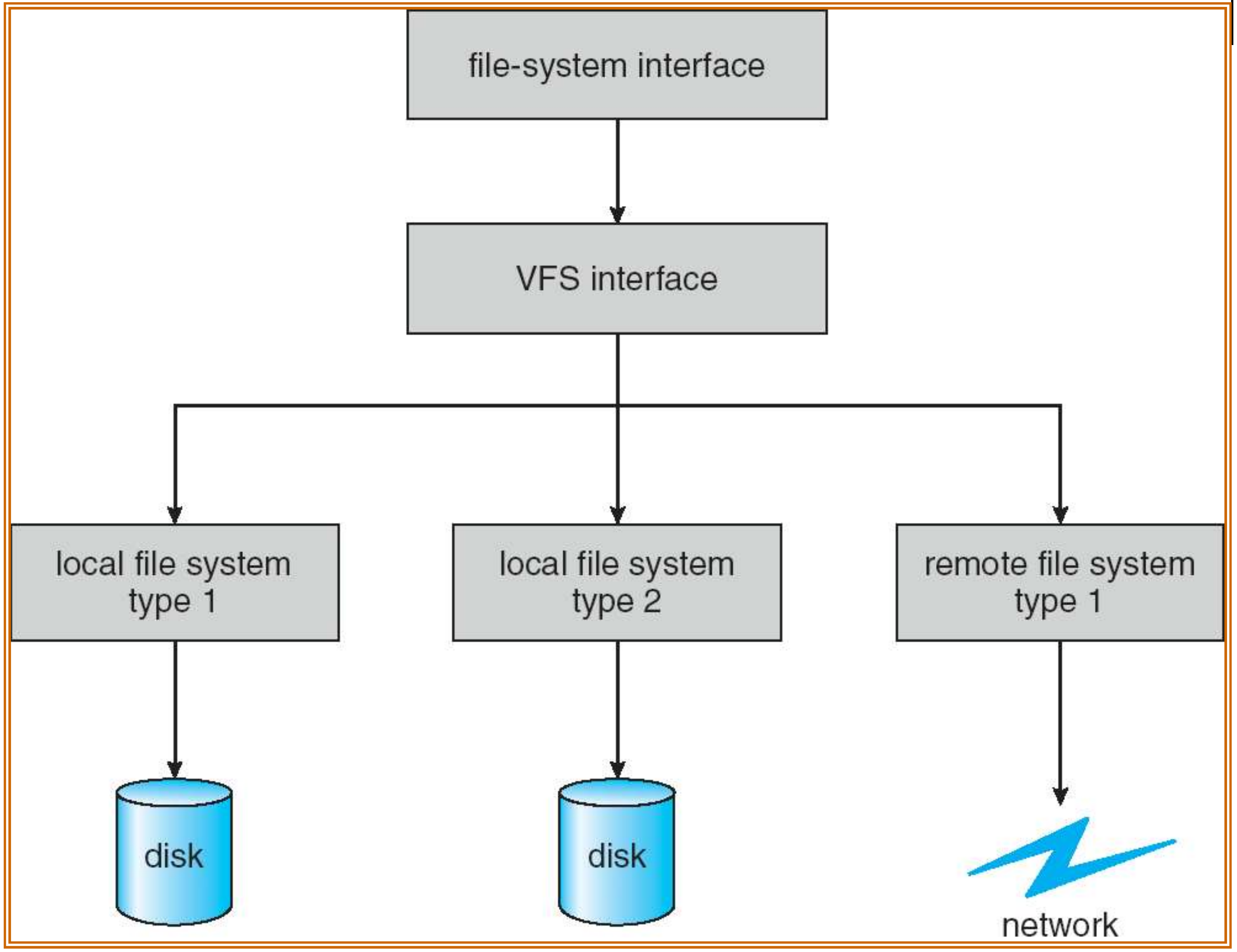
I/O control

devices

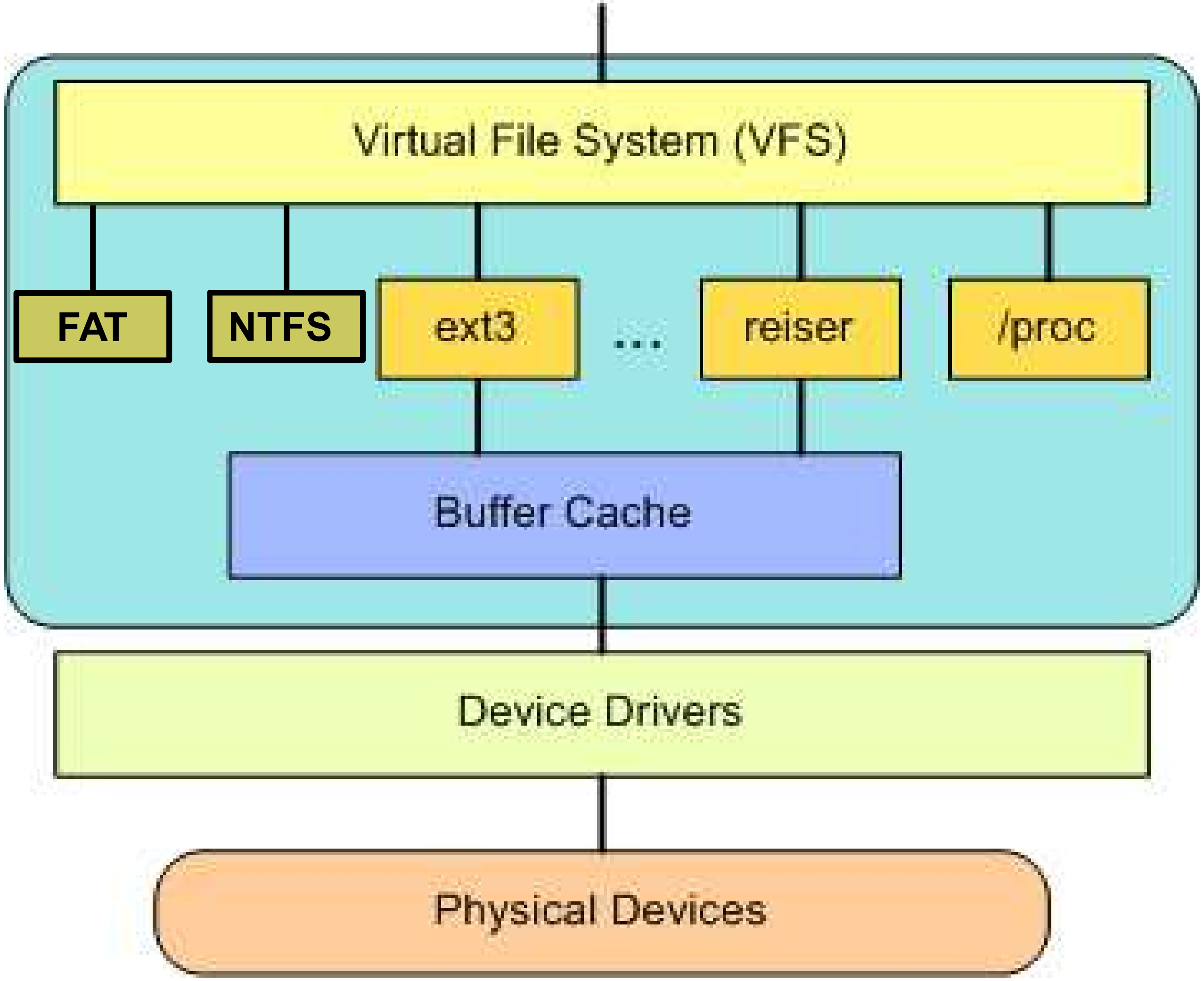
Virtual File Systems

- Virtual File Systems (VFS)
  - provide an object-oriented way of implementing file systems
  - allow the same system call interface (the API) to be used for different types of file systems
  - the API is to the VFS interface, rather than any specific type of file system.

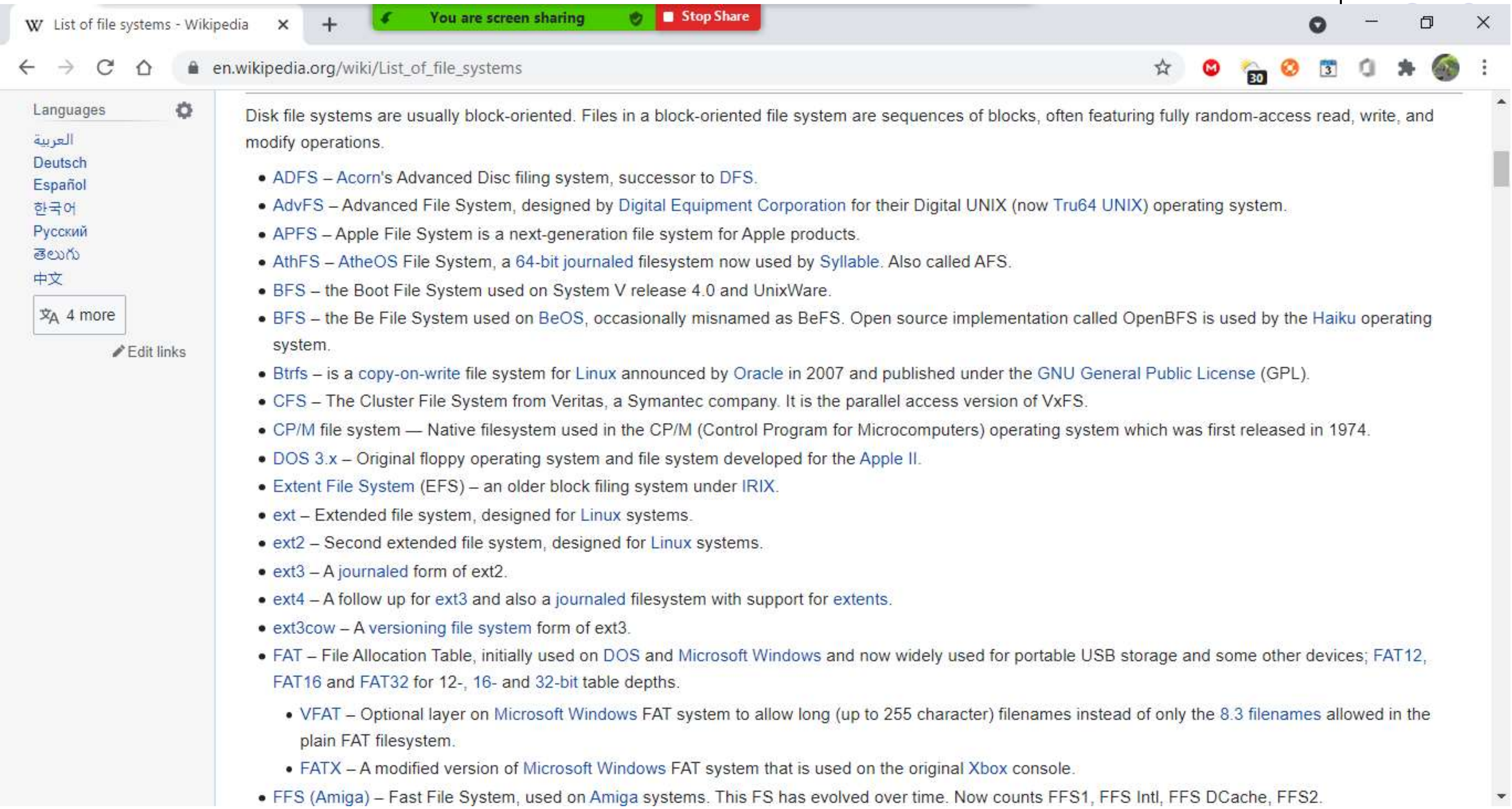
Schematic View of Virtual File System



Linux VFS


Virtual File System (VFS)
FAT
NTFS
ext3
...
reiser
/proc
Buffer Cache
Device Drivers
Physical Devices

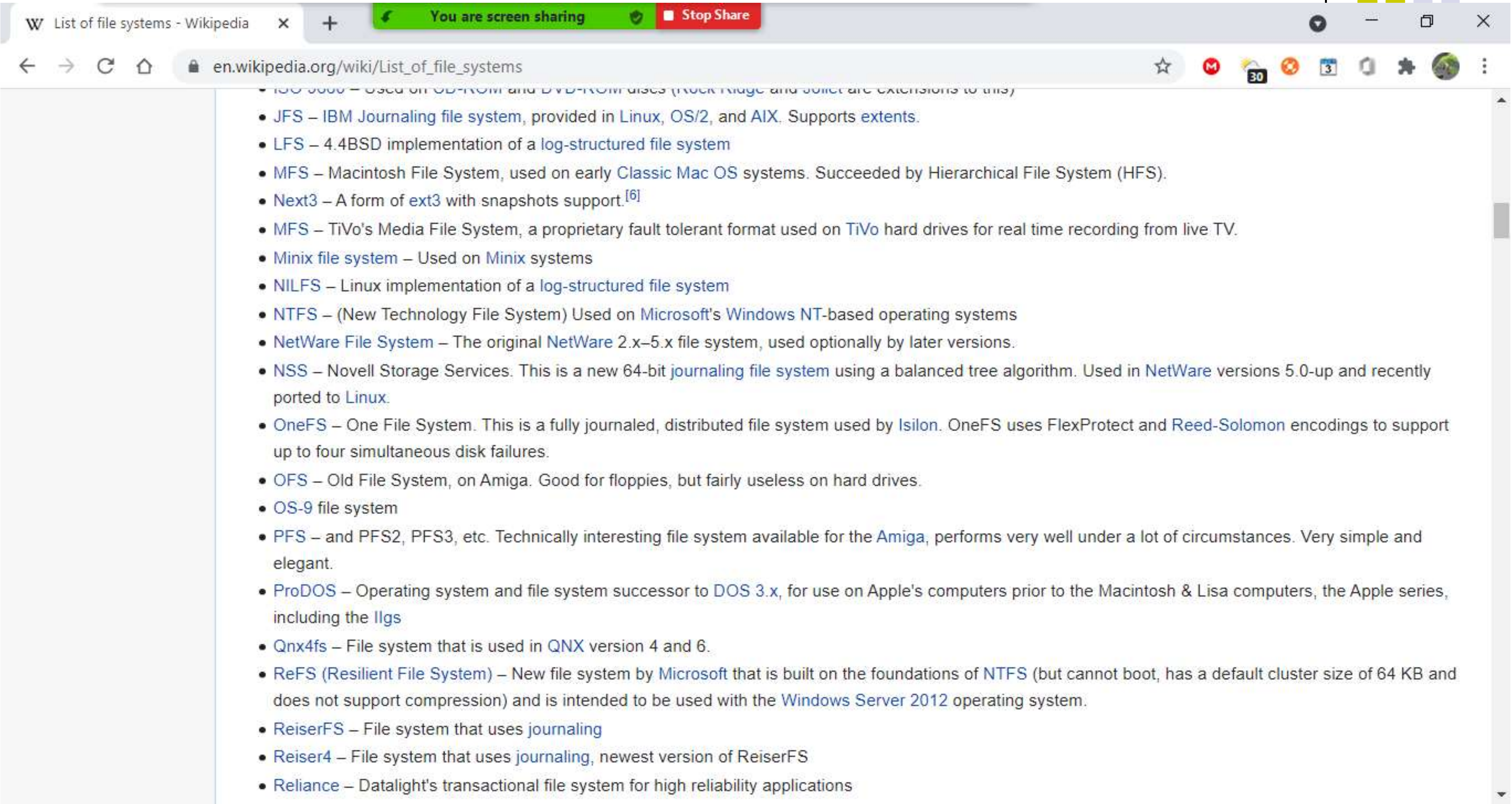
File system list



Disk file systems are usually block-oriented. Files in a block-oriented file system are sequences of blocks, often featuring fully random-access read, write, and modify operations.

- ADFS – Acorn's Advanced Disc filing system, successor to DFS.
- AdvFS – Advanced File System, designed by Digital Equipment Corporation for their Digital UNIX (now Tru64 UNIX) operating system.
- APFS – Apple File System is a next-generation file system for Apple products.
- AthFS – AtheOS File System, a 64-bit journaled filesystem now used by Syllable. Also called AFS.
- BFS – the Boot File System used on System V release 4.0 and UnixWare.
- BFS – the Be File System used on BeOS, occasionally misnamed as BeFS. Open source implementation called OpenBFS is used by the Haiku operating system.
- Btrfs – is a copy-on-write file system for Linux announced by Oracle in 2007 and published under the GNU General Public License (GPL).
- CFS – The Cluster File System from Veritas, a Symantec company. It is the parallel access version of VxFS.
- CP/M file system — Native filesystem used in the CP/M (Control Program for Microcomputers) operating system which was first released in 1974.
- DOS 3.x – Original floppy operating system and file system developed for the Apple II.
- Extent File System (EFS) – an older block filing system under IRIX.
- ext – Extended file system, designed for Linux systems.
- ext2 – Second extended file system, designed for Linux systems.
- ext3 – A journaled form of ext2.
- ext4 – A follow up for ext3 and also a journaled filesystem with support for extents.
- ext3cow – A versioning file system form of ext3.
- FAT – File Allocation Table, initially used on DOS and Microsoft Windows and now widely used for portable USB storage and some other devices; FAT12, FAT16 and FAT32 for 12-, 16- and 32-bit table depths.
    - VFAT – Optional layer on Microsoft Windows FAT system to allow long (up to 255 character) filenames instead of only the 8.3 filenames allowed in the plain FAT filesystem.
    - FATX – A modified version of Microsoft Windows FAT system that is used on the original Xbox console.
- FFS (Amiga) – Fast File System, used on Amiga systems. This FS has evolved over time. Now counts FFS1, FFS Intl, FFS DCache, FFS2.

File system list


List of file systems - Wikipedia
You are screen sharing
Stop Share
en.wikipedia.org/wiki/List_of_file_systems

- ISO 9660 – Used on CD-ROM and DVD-ROM discs (Rock Ridge and Joliet are extensions to this)
- JFS – IBM Journaling file system, provided in Linux, OS/2, and AIX. Supports extents.
- LFS – 4.4BSD implementation of a log-structured file system
- MFS – Macintosh File System, used on early Classic Mac OS systems. Succeeded by Hierarchical File System (HFS).
- Next3 – A form of ext3 with snapshots support.[6]
- MFS – TiVo's Media File System, a proprietary fault tolerant format used on TiVo hard drives for real time recording from live TV.
- Minix file system – Used on Minix systems
- NILFS – Linux implementation of a log-structured file system
- NTFS – (New Technology File System) Used on Microsoft's Windows NT-based operating systems
- NetWare File System – The original NetWare 2.x–5.x file system, used optionally by later versions.
- NSS – Novell Storage Services. This is a new 64-bit journaling file system using a balanced tree algorithm. Used in NetWare versions 5.0-up and recently ported to Linux.
- OneFS – One File System. This is a fully journaled, distributed file system used by Isilon. OneFS uses FlexProtect and Reed-Solomon encodings to support up to four simultaneous disk failures.
- OFS – Old File System, on Amiga. Good for floppies, but fairly useless on hard drives.
- OS-9 file system
- PFS – and PFS2, PFS3, etc. Technically interesting file system available for the Amiga, performs very well under a lot of circumstances. Very simple and elegant.
- ProDOS – Operating system and file system successor to DOS 3.x, for use on Apple's computers prior to the Macintosh & Lisa computers, the Apple series, including the IIgs
- Qnx4fs – File system that is used in QNX version 4 and 6.
- ReFS (Resilient File System) – New file system by Microsoft that is built on the foundations of NTFS (but cannot boot, has a default cluster size of 64 KB and does not support compression) and is intended to be used with the Windows Server 2012 operating system.
- ReiserFS – File system that uses journaling
- Reiser4 – File system that uses journaling, newest version of ReiserFS
- Reliance – Datalight's transactional file system for high reliability applications

Question

- Which of the following is incorrect about VFS?
  A. VFS allows an OS to support many different file systems
  B. VFS provides the same API for all file systems
  C. VFS is available in all OSes
  D. VFS hides the detailed implementation of each file system from programmers

Question

- Which of the following is a correct description of a directory?

  A. a directory is a disk partition to contain files

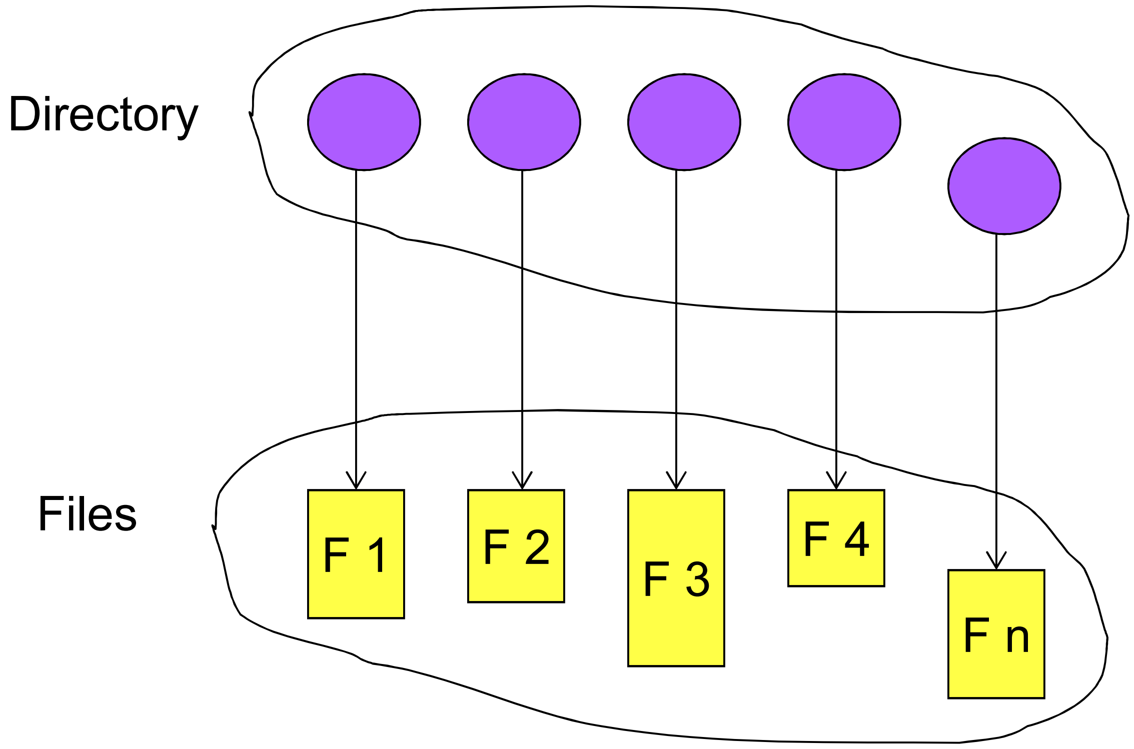
  B. a directory is actually a file containing partial information about its files

  C. a directory is a container of files' data

  D. a directory contains the FCB and data of files

Directory Structure

- A collection of nodes containing information about all files



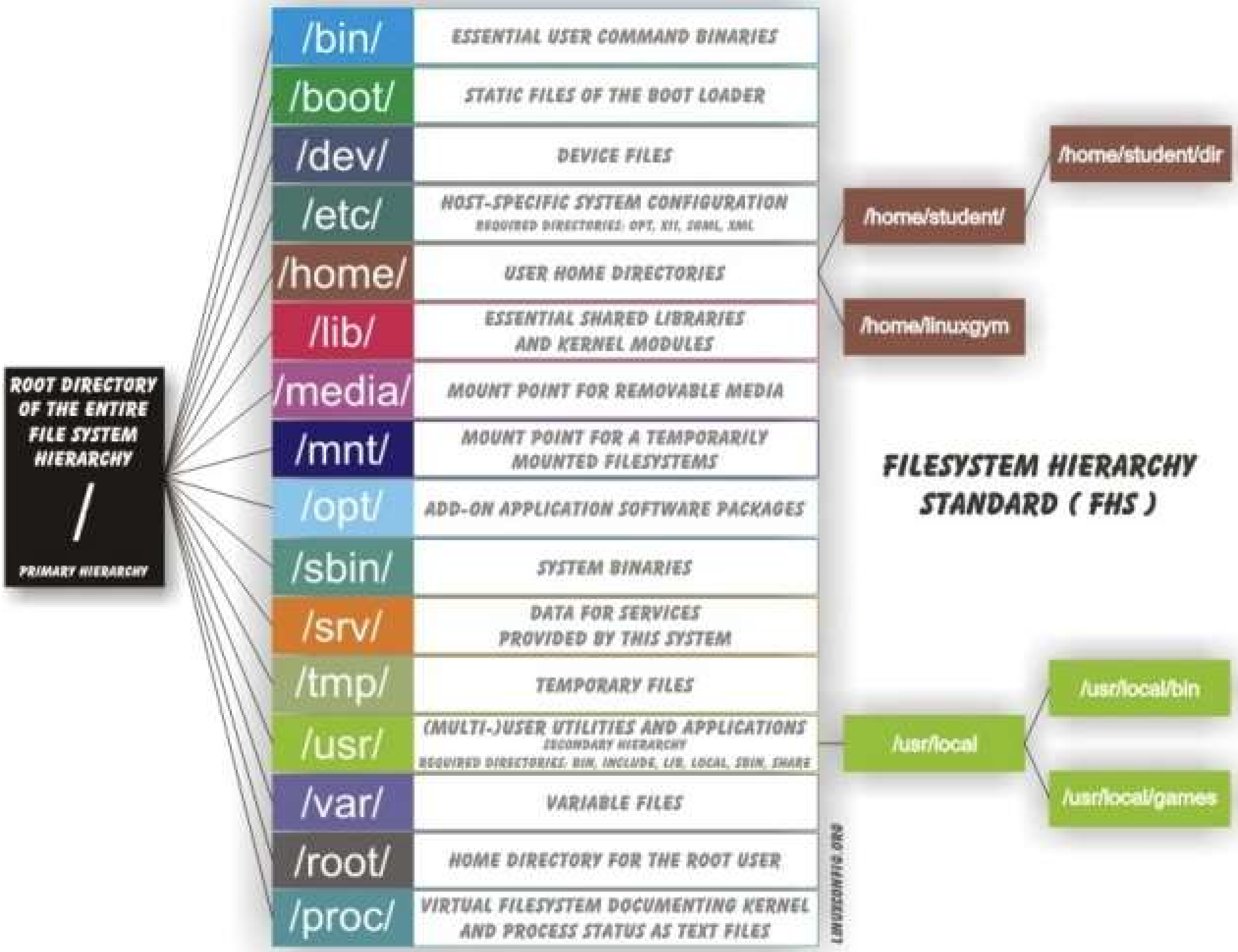
Directory Implementation

- **Linear list**
  - list of file names with pointer to the data blocks
  - simple to program
  - time-consuming to execute
  - FAT http://en.wikipedia.org/wiki/File_Allocation_Table
  - Linux FS (Ext3)
- **Hash Table**
  - linear list with hash data structure
  - decreases directory search time
  - **collisions** resolution
  - fixed size

Directory Implementation

- **Balanced binary tree**
  - RAISERFS http://en.wikipedia.org/wiki/ReiserFS

Directory in Linux



ROOT DIRECTORY OF THE ENTIRE FILE SYSTEM HIERARCHY

/

PRIMARY HIERARCHY

/bin/	ESSENTIAL USER COMMAND BINARIES
/boot/	STATIC FILES OF THE BOOT LOADER
/dev/	DEVICE FILES
/etc/	HOST-SPECIFIC SYSTEM CONFIGURATION REQUIRED DIRECTORIES: OPT, X11, SGML, XML
/home/	USER HOME DIRECTORIES
/lib/	ESSENTIAL SHARED LIBRARIES AND KERNEL MODULES
/media/	MOUNT POINT FOR REMOVABLE MEDIA
/mnt/	MOUNT POINT FOR A TEMPORARILY MOUNTED FILESYSTEMS
/opt/	ADD-ON APPLICATION SOFTWARE PACKAGES
/sbin/	SYSTEM BINARIES
/srv/	DATA FOR SERVICES PROVIDED BY THIS SYSTEM
/tmp/	TEMPORARY FILES
/usr/	(MULTI-)USER UTILITIES AND APPLICATIONS SECONDARY HIERARCHY REQUIRED DIRECTORIES: BIN, INCLUDE, LIB, LOCAL, SBIN, SHARE
/var/	VARIABLE FILES
/root/	HOME DIRECTORY FOR THE ROOT USER
/proc/	VIRTUAL FILESYSTEM DOCUMENTING KERNEL AND PROCESS STATUS AS TEXT FILES

/home/student/

/home/student/dir

/home/linuxgym

FILESYSTEM HIERARCHY STANDARD (FHS)

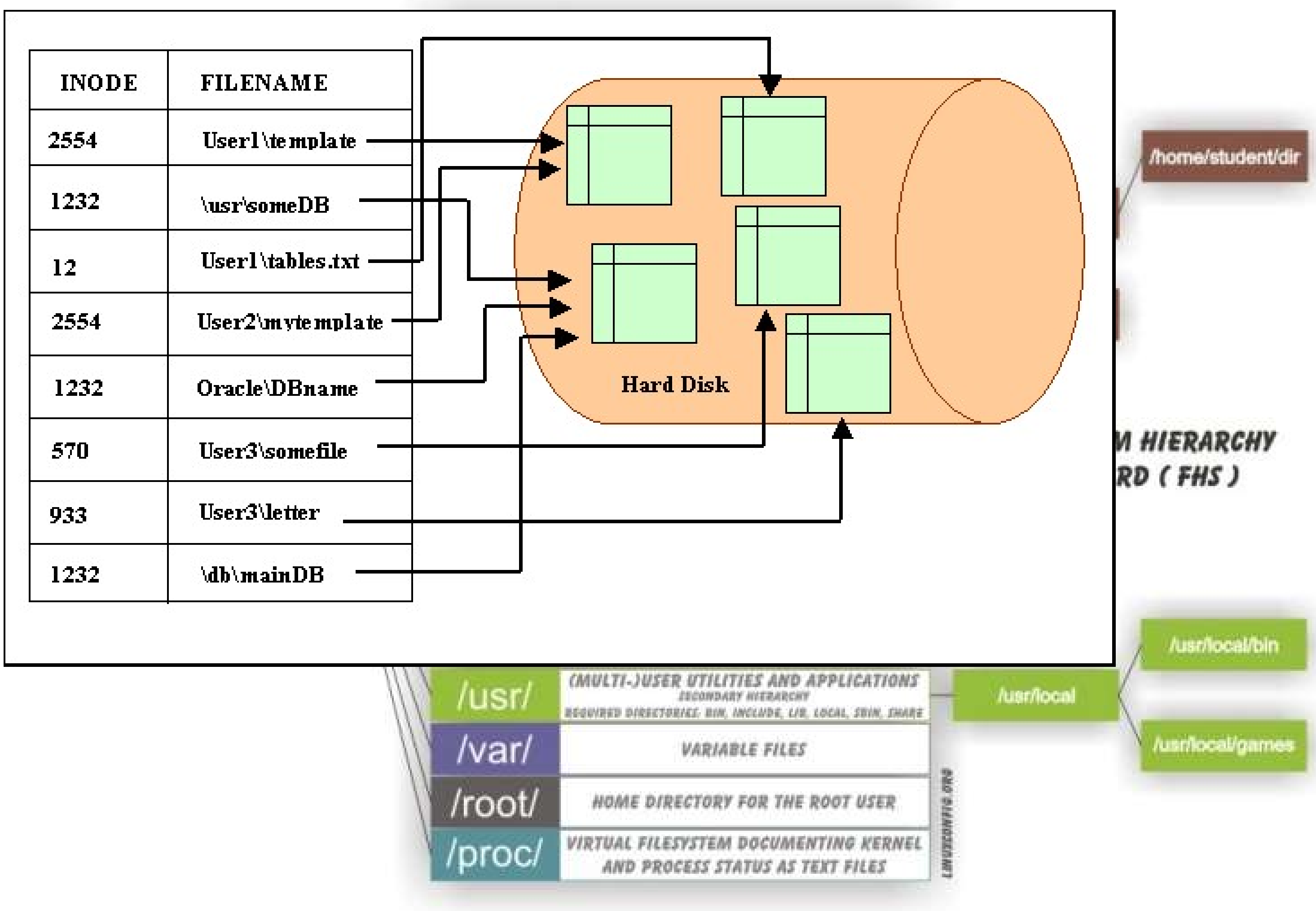
/usr/local

/usr/local/bin

/usr/local/games

LINUXCONFIG.ORG

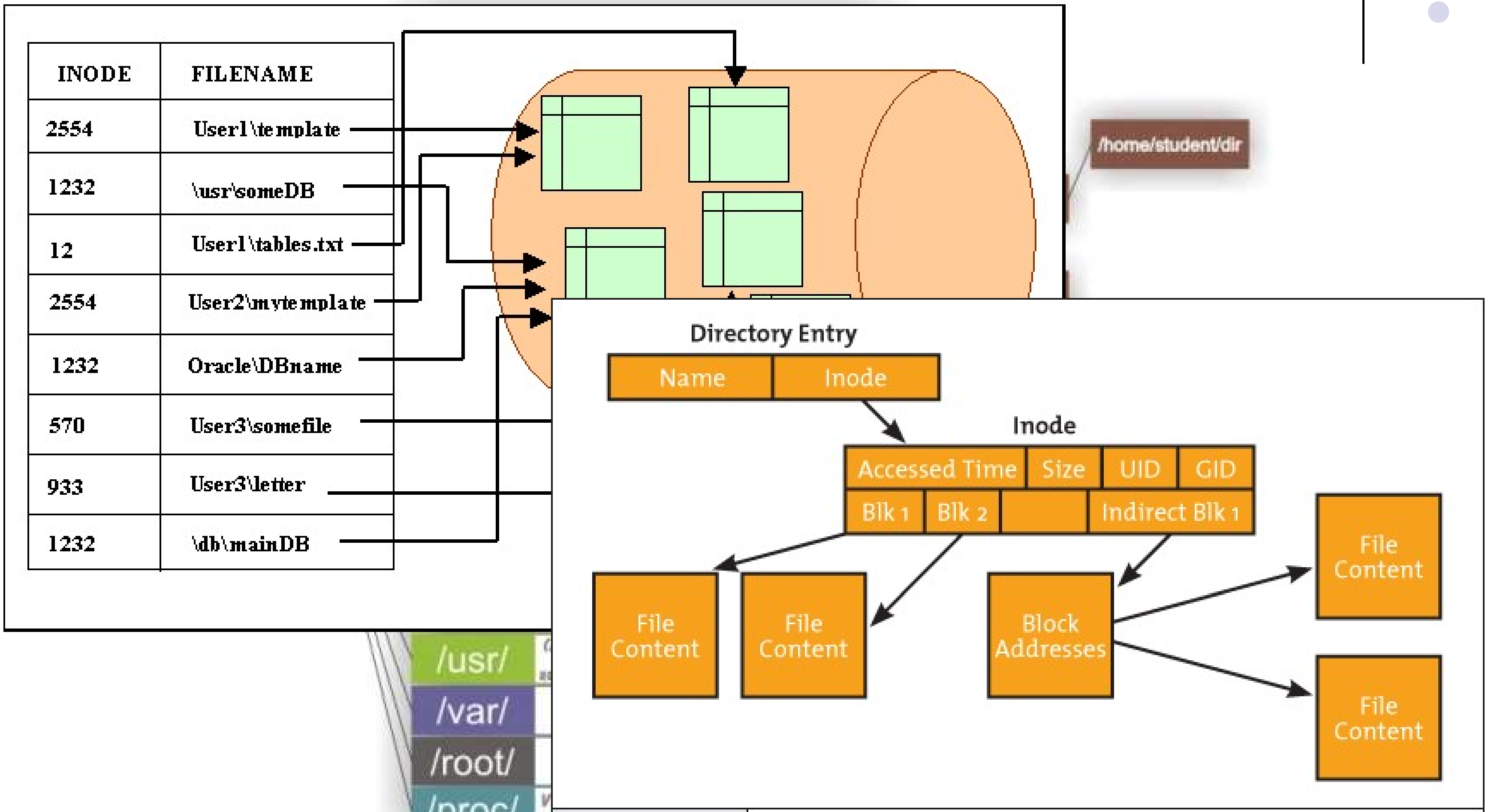
Directory in Linux



INODE	FILENAME
2554	User1\template
1232	\usr\someDB
12	User1\tables.txt
2554	User2\mytemplate
1232	Oracle\DBname
570	User3\somefile
933	User3\letter
1232	\db\mainDB

Hard Disk
/home/student/dir
M HIERARCHY
RD (FHS)
/usr/local/bin
/usr/local
/usr/local/games
/usr/
(MULTI-)USER UTILITIES AND APPLICATIONS
SECONDARY HIERARCHY
REQUIRED DIRECTORIES: BIN, INCLUDE, LIB, LOCAL, SBIN, SHARE
/var/
VARIABLE FILES
/root/
HOME DIRECTORY FOR THE ROOT USER
/proc/
VIRTUAL FILESYSTEM DOCUMENTING KERNEL AND PROCESS STATUS AS TEXT FILES

Directory in Linux



INODE	FILENAME
2554	User1\template
1232	\usr\someDB
12	User1\tables.txt
2554	User2\mytemplate
1232	Oracle\DBname
570	User3\somefile
933	User3\letter
1232	\db\mainDB

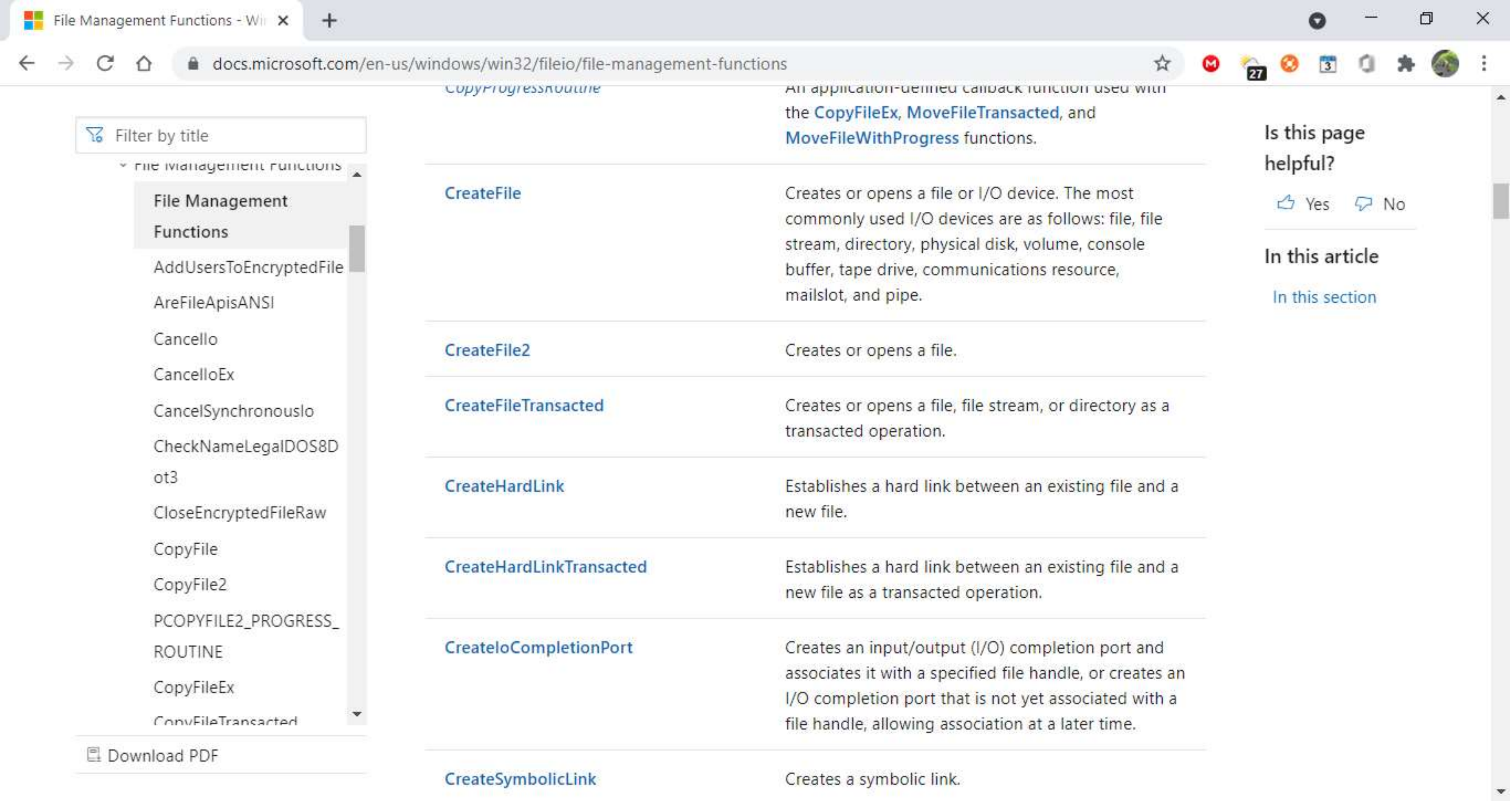
FIGURE 1 RELATIONSHIP BETWEEN THE DIRECTORY ENTRY, AN INODE, AND BLOCKS OF AN ALLOCATED FILE

File management API

File Operations

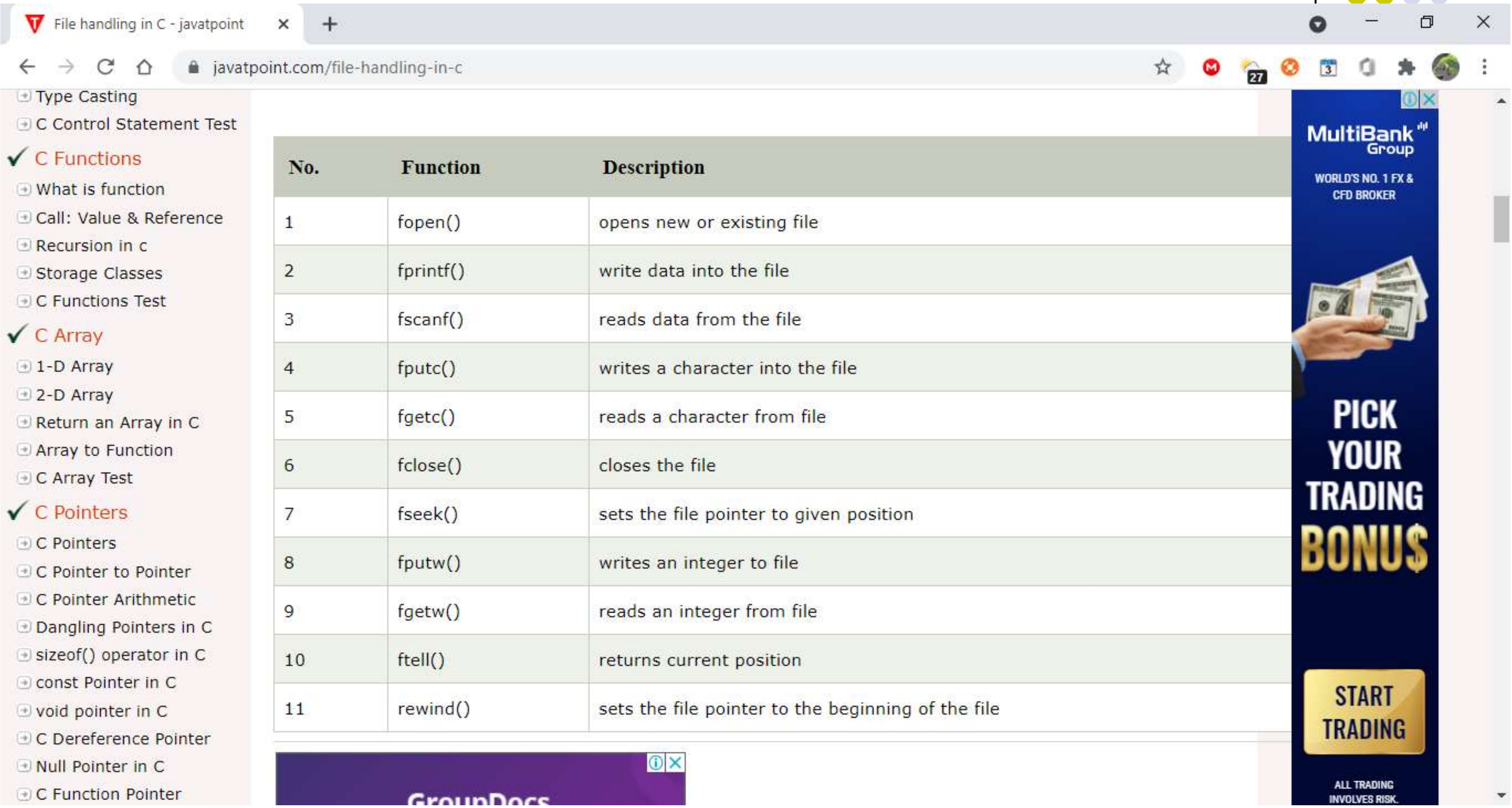
- Create
- Write
- Read
- Reposition within file
- Delete
- Truncate
- *Open(F_i)*
  - search the directory structure on disk for entry F_i, and move the content of entry to memory
- *Close (F_i)*
  - move the content of entry F_i in memory to directory structure on disk

File Operations



CopyProgressRoutine	An application-defined callback function used with the CopyFileEx, MoveFileTransacted, and MoveFileWithProgress functions.
CreateFile	Creates or opens a file or I/O device. The most commonly used I/O devices are as follows: file, file stream, directory, physical disk, volume, console buffer, tape drive, communications resource, mailslot, and pipe.
CreateFile2	Creates or opens a file.
CreateFileTransacted	Creates or opens a file, file stream, or directory as a transacted operation.
CreateHardLink	Establishes a hard link between an existing file and a new file.
CreateHardLinkTransacted	Establishes a hard link between an existing file and a new file as a transacted operation.
CreateIoCompletionPort	Creates an input/output (I/O) completion port and associates it with a specified file handle, or creates an I/O completion port that is not yet associated with a file handle, allowing association at a later time.
CreateSymbolicLink	Creates a symbolic link.

File Operations



File handling in C - javatpoint
javatpoint.com/file-handling-in-c
27

- Type Casting
- C Control Statement Test

✓ C Functions

- What is function
- Call: Value & Reference
- Recursion in c
- Storage Classes
- C Functions Test

✓ C Array

- 1-D Array
- 2-D Array
- Return an Array in C
- Array to Function
- C Array Test

✓ C Pointers

- C Pointers
- C Pointer to Pointer
- C Pointer Arithmetic
- Dangling Pointers in C
- sizeof() operator in C
- const Pointer in C
- void pointer in C
- C Dereference Pointer
- Null Pointer in C
- C Function Pointer

No.	Function	Description
1	fopen()	opens new or existing file
2	fprintf()	write data into the file
3	fscanf()	reads data from the file
4	fputc()	writes a character into the file
5	fgetc()	reads a character from file
6	fclose()	closes the file
7	fseek()	sets the file pointer to given position
8	fputw()	writes an integer to file
9	fgetw()	reads an integer from file
10	ftell()	returns current position
11	rewind()	sets the file pointer to the beginning of the file

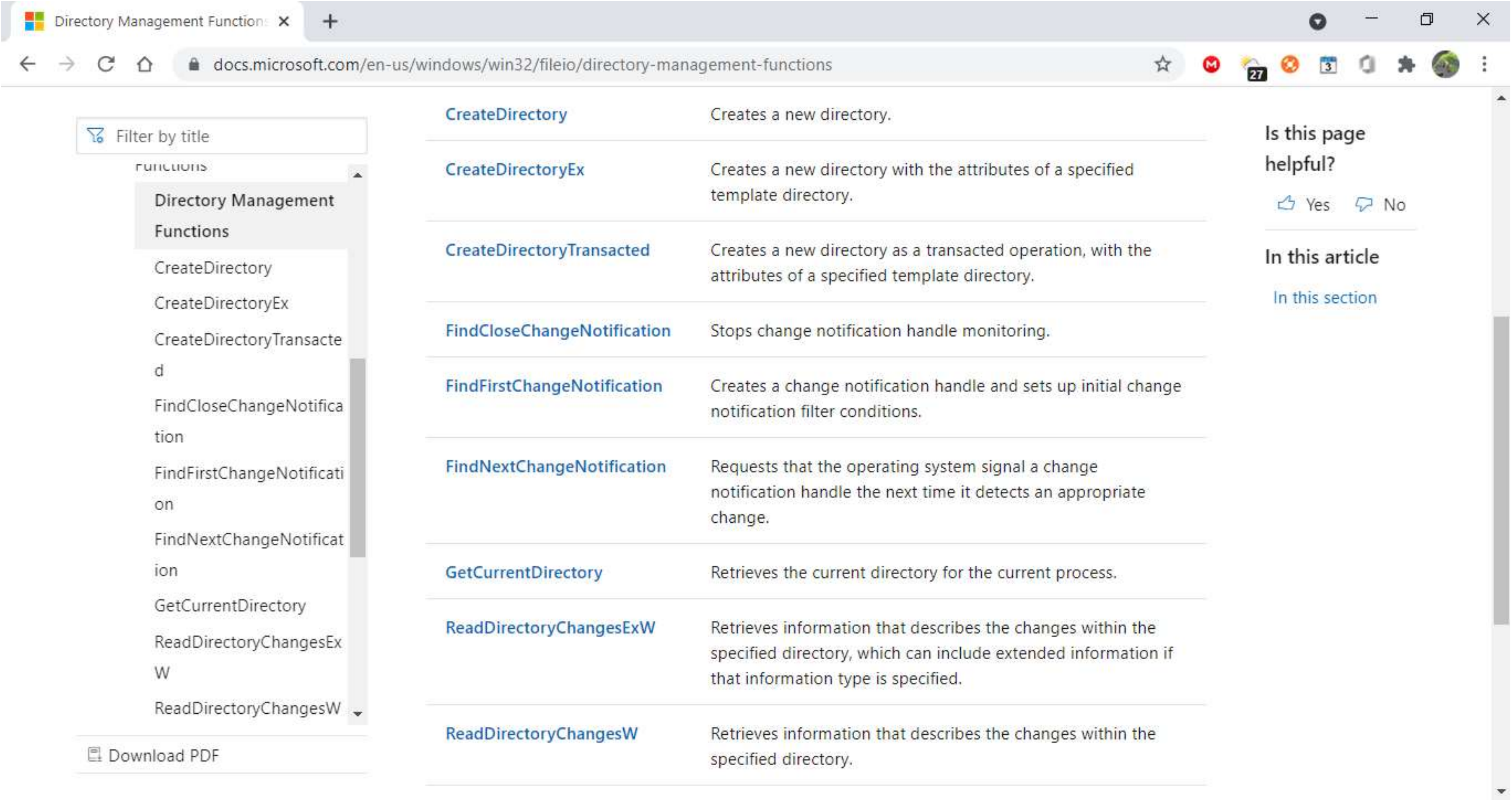
GroupDocs

MultiBank Group
WORLD'S NO. 1 FX & CFD BROKER
PICK YOUR TRADING BONU$
START TRADING
ALL TRADING INVOLVES RISK.

Operations Performed on Directory

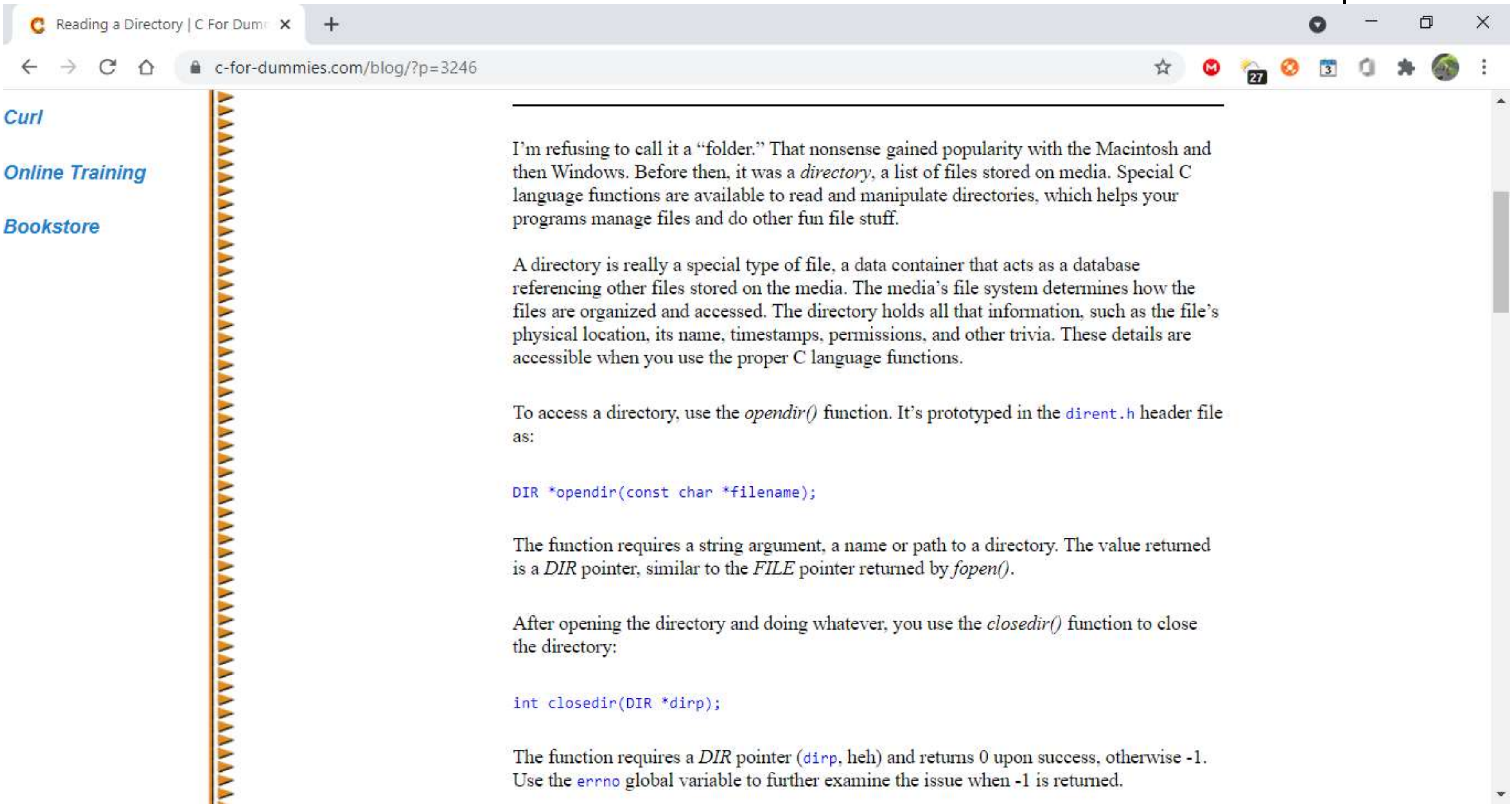
- Search for a file
- Create a directory
- Delete a directory
- List a directory
- Rename a directory
- Traverse the file system

Operations Performed on Directory


Directory Management Function
docs.microsoft.com/en-us/windows/win32/fileio/directory-management-functions
Filter by title
Functions
Directory Management Functions
CreateDirectory
CreateDirectoryEx
CreateDirectoryTransacted
FindCloseChangeNotification
FindFirstChangeNotification
FindNextChangeNotification
GetCurrentDirectory
ReadDirectoryChangesExW
ReadDirectoryChangesW
Download PDF
Is this page helpful?
Yes
No
In this article
In this section

CreateDirectory	Creates a new directory.
CreateDirectoryEx	Creates a new directory with the attributes of a specified template directory.
CreateDirectoryTransacted	Creates a new directory as a transacted operation, with the attributes of a specified template directory.
FindCloseChangeNotification	Stops change notification handle monitoring.
FindFirstChangeNotification	Creates a change notification handle and sets up initial change notification filter conditions.
FindNextChangeNotification	Requests that the operating system signal a change notification handle the next time it detects an appropriate change.
GetCurrentDirectory	Retrieves the current directory for the current process.
ReadDirectoryChangesExW	Retrieves information that describes the changes within the specified directory, which can include extended information if that information type is specified.
ReadDirectoryChangesW	Retrieves information that describes the changes within the specified directory.

Operations Performed on Directory


Reading a Directory | C For Dum
c-for-dummies.com/blog/?p=3246
Curl
Online Training
Bookstore

I'm refusing to call it a "folder." That nonsense gained popularity with the Macintosh and then Windows. Before then, it was a *directory*, a list of files stored on media. Special C language functions are available to read and manipulate directories, which helps your programs manage files and do other fun file stuff.

A directory is really a special type of file, a data container that acts as a database referencing other files stored on the media. The media's file system determines how the files are organized and accessed. The directory holds all that information, such as the file's physical location, its name, timestamps, permissions, and other trivia. These details are accessible when you use the proper C language functions.

To access a directory, use the *opendir()* function. It's prototyped in the `dirent.h` header file as:

```
DIR *opendir(const char *filename);
```

The function requires a string argument, a name or path to a directory. The value returned is a *DIR* pointer, similar to the *FILE* pointer returned by *fopen()*.

After opening the directory and doing whatever, you use the *closedir()* function to close the directory:

```
int closedir(DIR *dirp);
```

The function requires a *DIR* pointer (`dirp`, heh) and returns 0 upon success, otherwise -1. Use the `errno` global variable to further examine the issue when -1 is returned.

File/directory protection

Protection

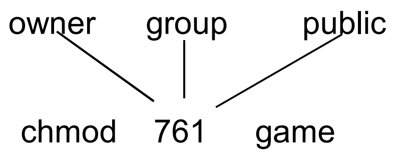
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

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

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



chmod 761 game

Attach a group to a file

chgrp G game

Question

- Suppose a file has access mode 664. Which is correct ?

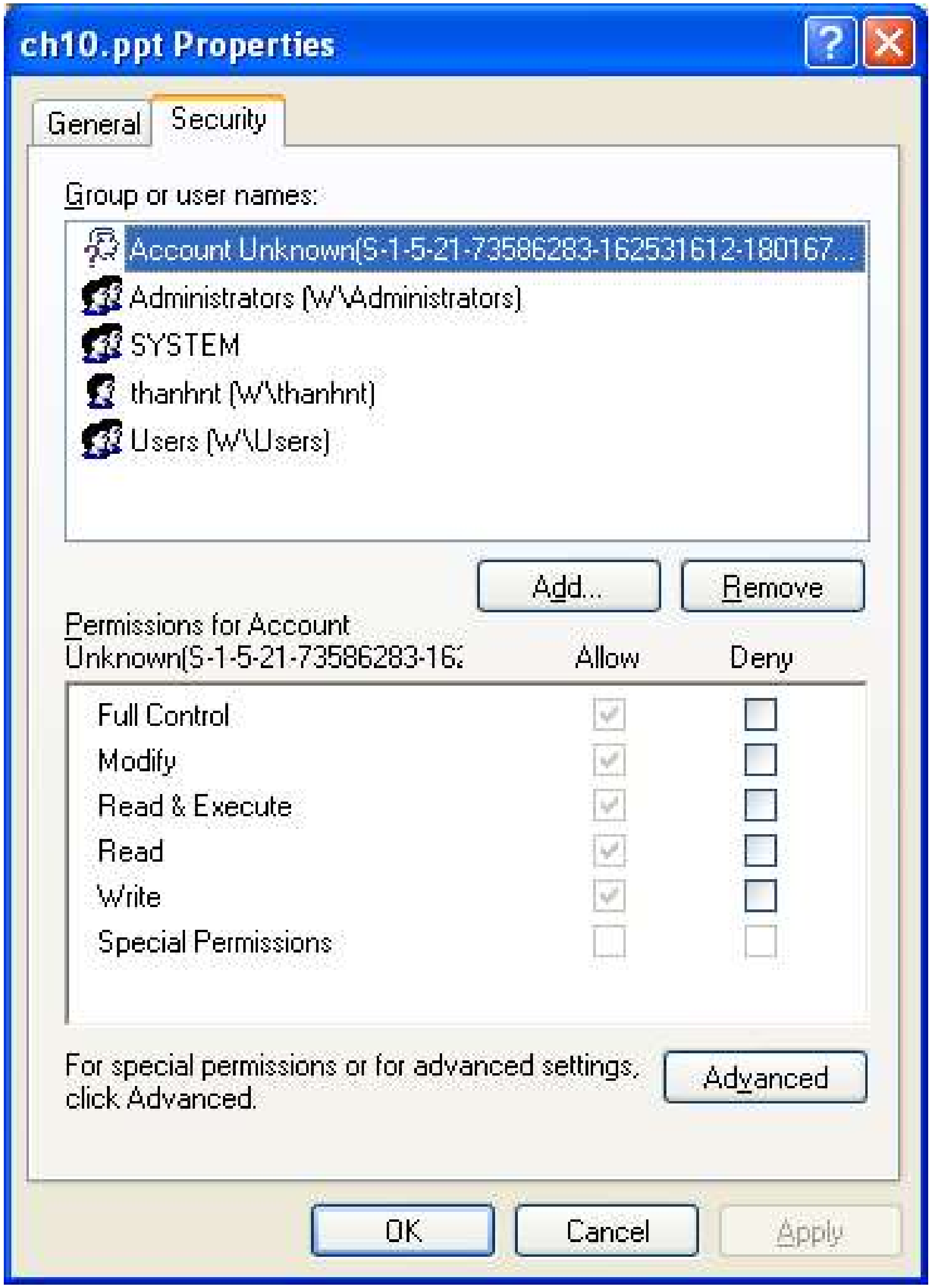
A. Any user can execute the file

B. Users of the owner group can execute the file

C. Any user can read the file

D. The owner cannot write the file

Windows XP Access-control List Management



ch10.ppt Properties

Group or user names:

- Account Unknown(S-1-5-21-73586283-162531612-180167...
- Administrators (W\Administrators)
- SYSTEM
- thanhnt (W\thanhnt)
- Users (W\Users)

Permissions for Account Unknown(S-1-5-21-73586283-16:	Allow	Deny
Full Control	☑	□
Modify	☑	□
Read & Execute	☑	□
Read	☑	□
Write	☑	□
Special Permissions	□	□

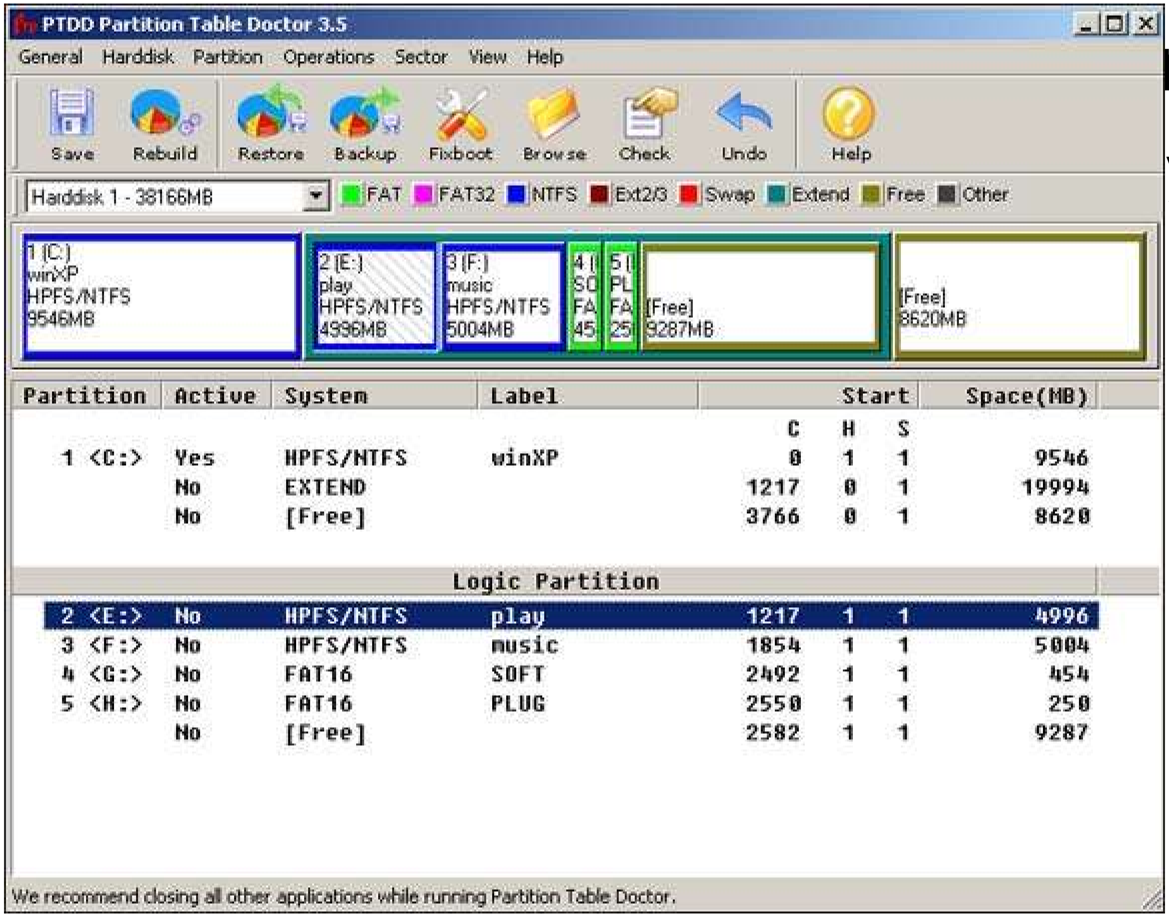
For special permissions or for advanced settings, click Advanced.

Storage Allocation Methods

Disk partitions

- A disk can be split into partitions
  - each is a consecutive range of cylinders
  - each is also called logic disk
  - e.g., Windows called: C, D, E drives

Disk partitions


PTDD Partition Table Doctor 3.5
General Harddisk Partition Operations Sector View Help
Save Rebuild Restore Backup Fixboot Browse Check Undo Help
🟩 FAT 🟪 FAT32 🟦 NTFS 🟥 Ext2/3 🟥 Swap 🟩 Extend 🟨 Free ⬛ Other
1 (C:) winXP HPFS/NTFS 9546MB
2 (E:) play HPFS/NTFS 4996MB
3 (F:) music HPFS/NTFS 5004MB
4 (SO FA 45
5 (PL FA 25
[Free] 9287MB
[Free] 8620MB

Harddisk 1 - 38166MB

Partition	Active	System	Label	Start C	Start H	Start S	Space(MB)
1 <C:>	Yes	HPFS/NTFS	winXP	0	1	1	9546
	No	EXTEND		1217	0	1	19994
	No	[Free]		3766	0	1	8620
Logic Partition							
2 <E:>	No	HPFS/NTFS	play	1217	1	1	4996
3 <F:>	No	HPFS/NTFS	music	1854	1	1	5004
4 <G:>	No	FAT16	SOFT	2492	1	1	454
5 <H:>	No	FAT16	PLUG	2550	1	1	250
	No	[Free]		2582	1	1	9287

We recommend closing all other applications while running Partition Table Doctor.

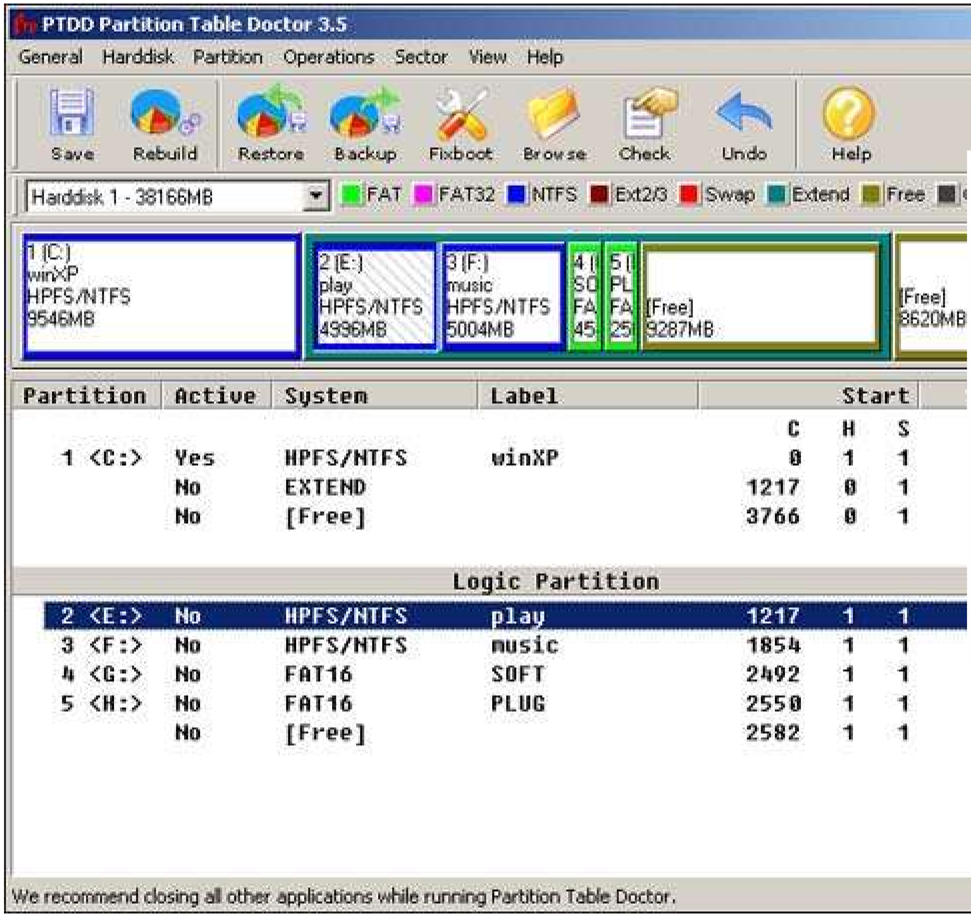
ns

ylinders

ves

Disk partitions

ns

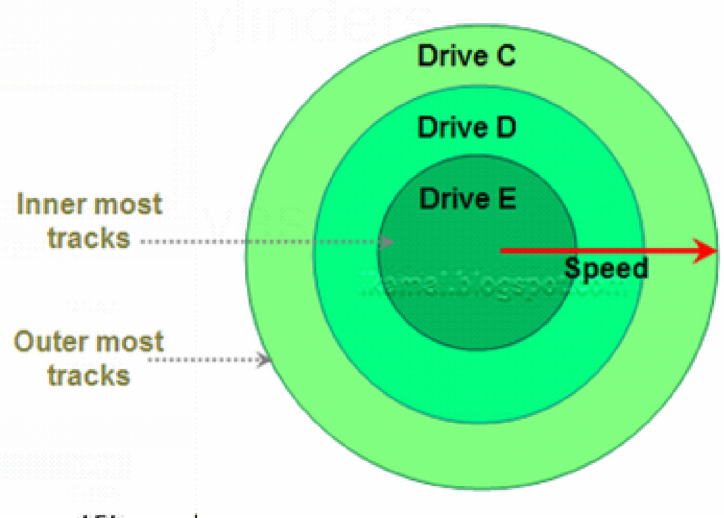


PTDD Partition Table Doctor 3.5
General Harddisk Partition Operations Sector View Help
Save Rebuild Restore Backup Fixboot Browse Check Undo Help
Harddisk 1 - 38166MB
🟩 FAT 🟪 FAT32 🟦 NTFS 🟥 Ext2/3 🟥 Swap 🟦 Extend 🟫 Free ⬛
1 (C:) winXP HPFS/NTFS 9546MB
2 (E:) play HPFS/NTFS 4996MB
3 (F:) music HPFS/NTFS 5004MB
4 (SO FA 45
5 (PL FA 25
[Free] 9287MB
[Free] 8620MB

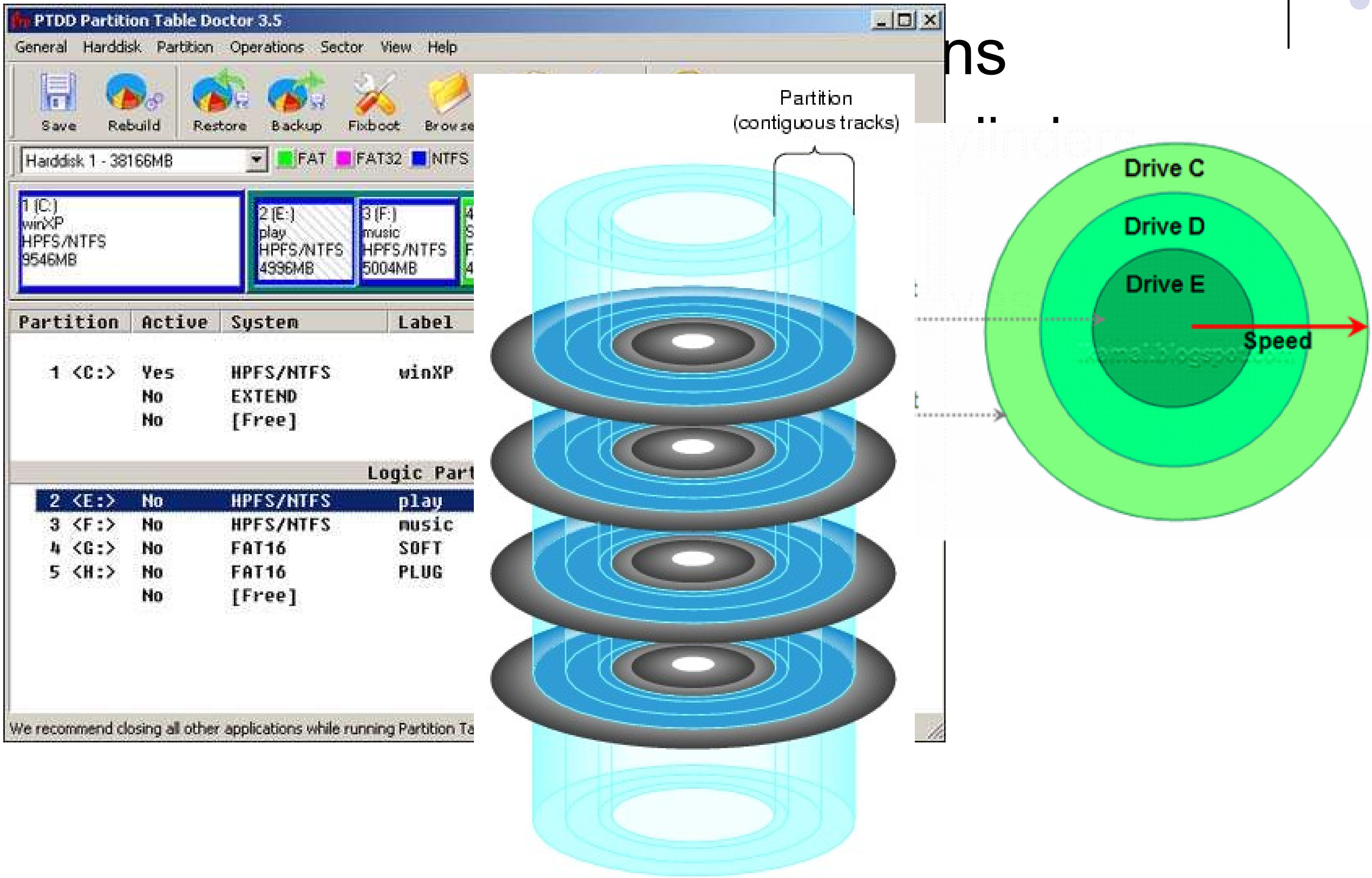
```
Partition  Active  System       Label                 Start
                                                  C    H    S
  1 <C:>   Yes     HPFS/NTFS    winXP             0    1    1
           No      EXTEND                      1217    0    1
           No      [Free]                      3766    0    1

                           Logic Partition
  2 <E:>   No      HPFS/NTFS    play           1217    1    1
  3 <F:>   No      HPFS/NTFS    music          1854    1    1
  4 <G:>   No      FAT16        SOFT           2492    1    1       454
  5 <H:>   No      FAT16        PLUG           2550    1    1       250
           No      [Free]                      2582    1    1      9287
```

We recommend closing all other applications while running Partition Table Doctor.



Disk partitions



PTDD Partition Table Doctor 3.5

Harddisk 1 - 38166MB 🟩 FAT 🟪 FAT32 🟦 NTFS

1 (C:) winXP HPFS/NTFS 9546MB	2 (E:) play HPFS/NTFS 4996MB	3 (F:) music HPFS/NTFS 5004MB	4 S F 4

```
Partition  Active  System      Label
  1 <C:>   Yes     HPFS/NTFS   winXP
           No      EXTEND
           No      [Free]
                            Logic Part
  2 <E:>   No      HPFS/NTFS   play
  3 <F:>   No      HPFS/NTFS   music
  4 <G:>   No      FAT16       SOFT
  5 <H:>   No      FAT16       PLUG
           No      [Free]
```

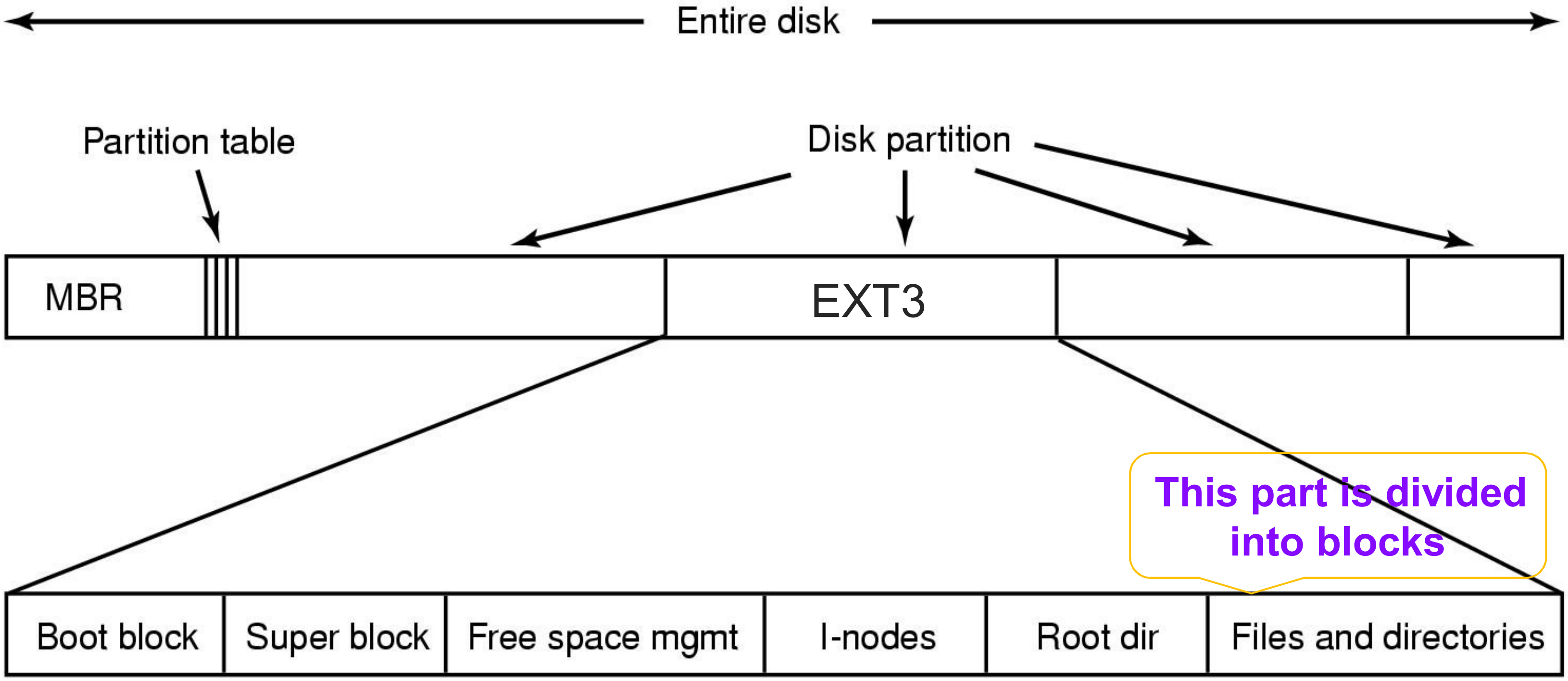
We recommend closing all other applications while running Partition Ta

Partition organization

- Organization is specific to each OS
  - region for storing data is divided into equal **blocks**
  - block is a read/write unit of OS

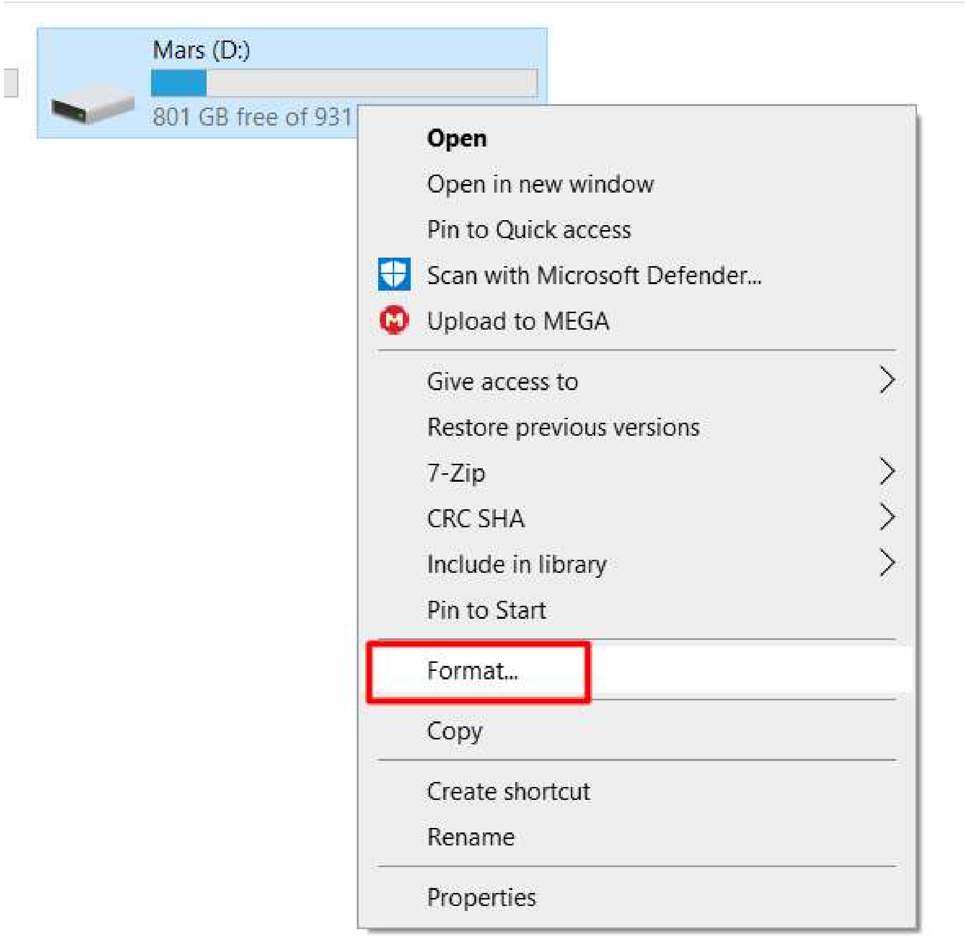
Partition organization

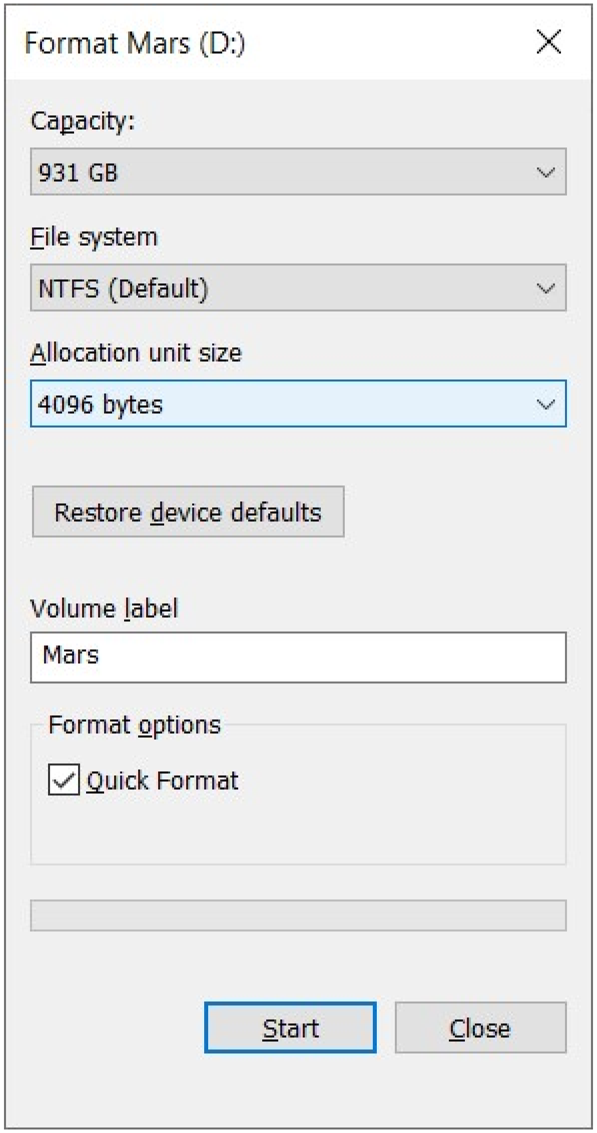
- Organization is specific to each OS
  - region for storing data is divided into equal **blocks**



This part is divided into blocks

Format logical volume




Format Mars (D:)

Capacity: 931 GB

File system: NTFS (Default)

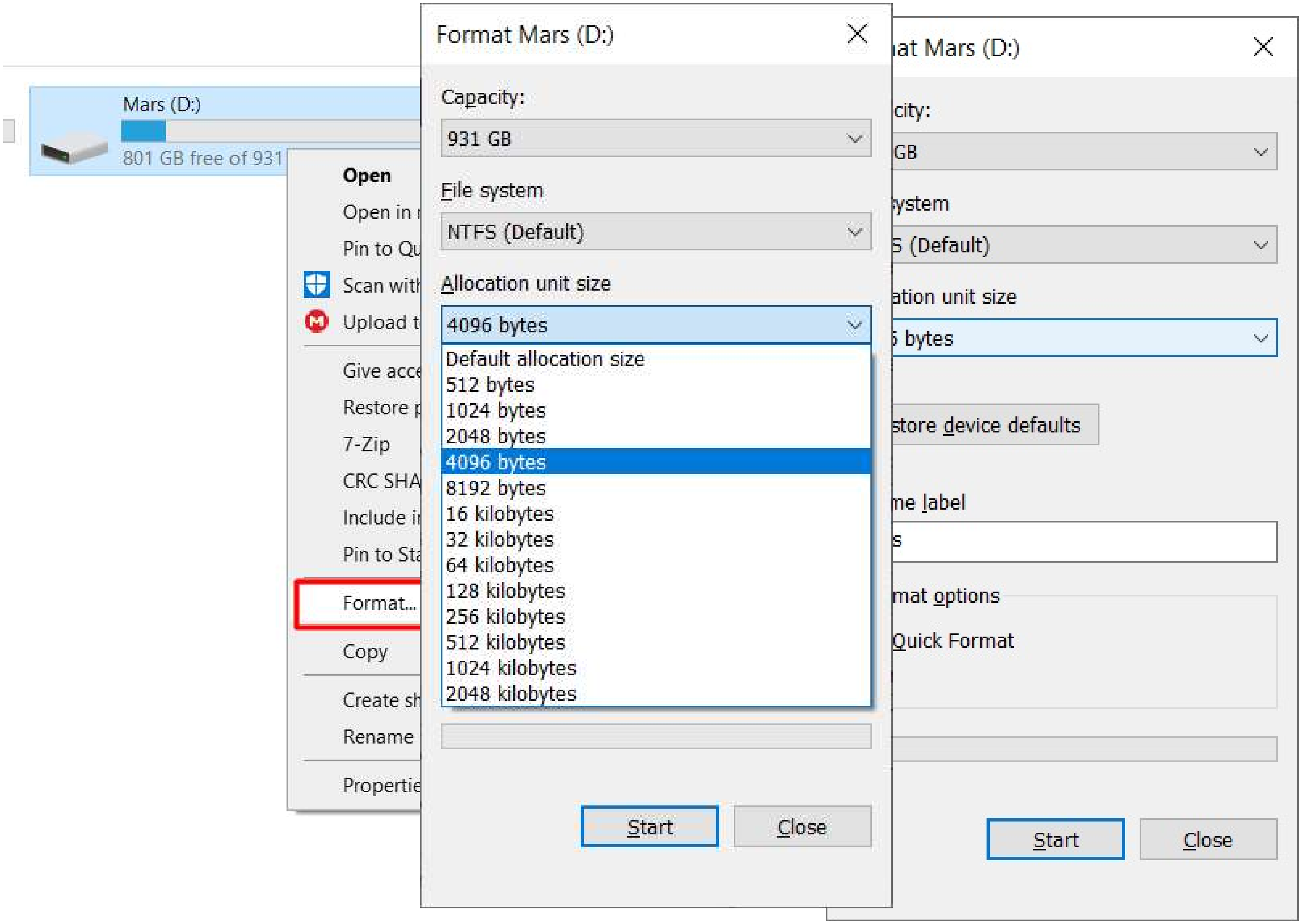
Allocation unit size: 4096 bytes

Volume label: Mars

Format options

☑ Quick Format

Format logical volume



Format Mars (D:)

Capacity:

931 GB

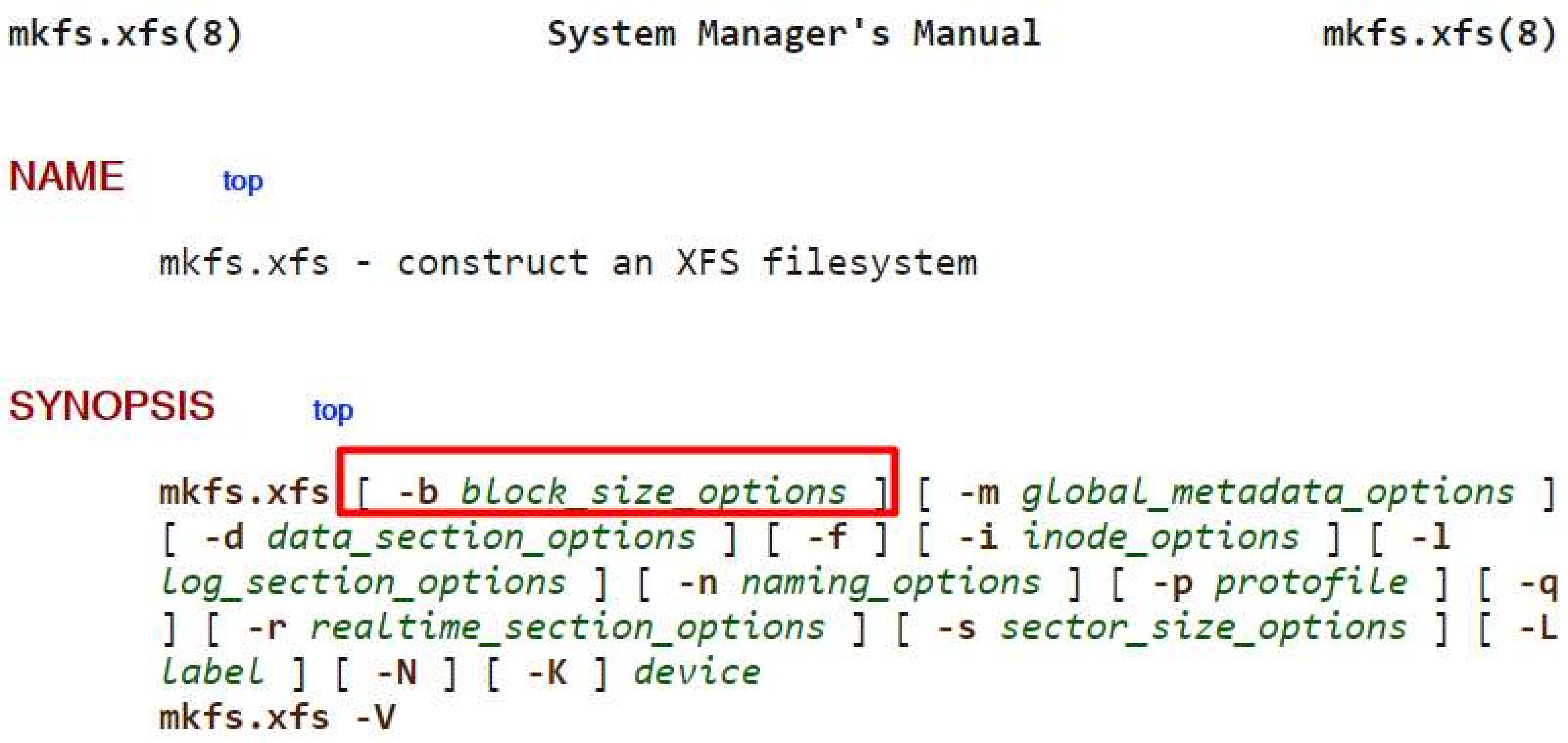
File system

NTFS (Default)

Allocation unit size

4096 bytes

xfs format



```
mkfs.xfs(8)              System Manager's Manual              mkfs.xfs(8)
```

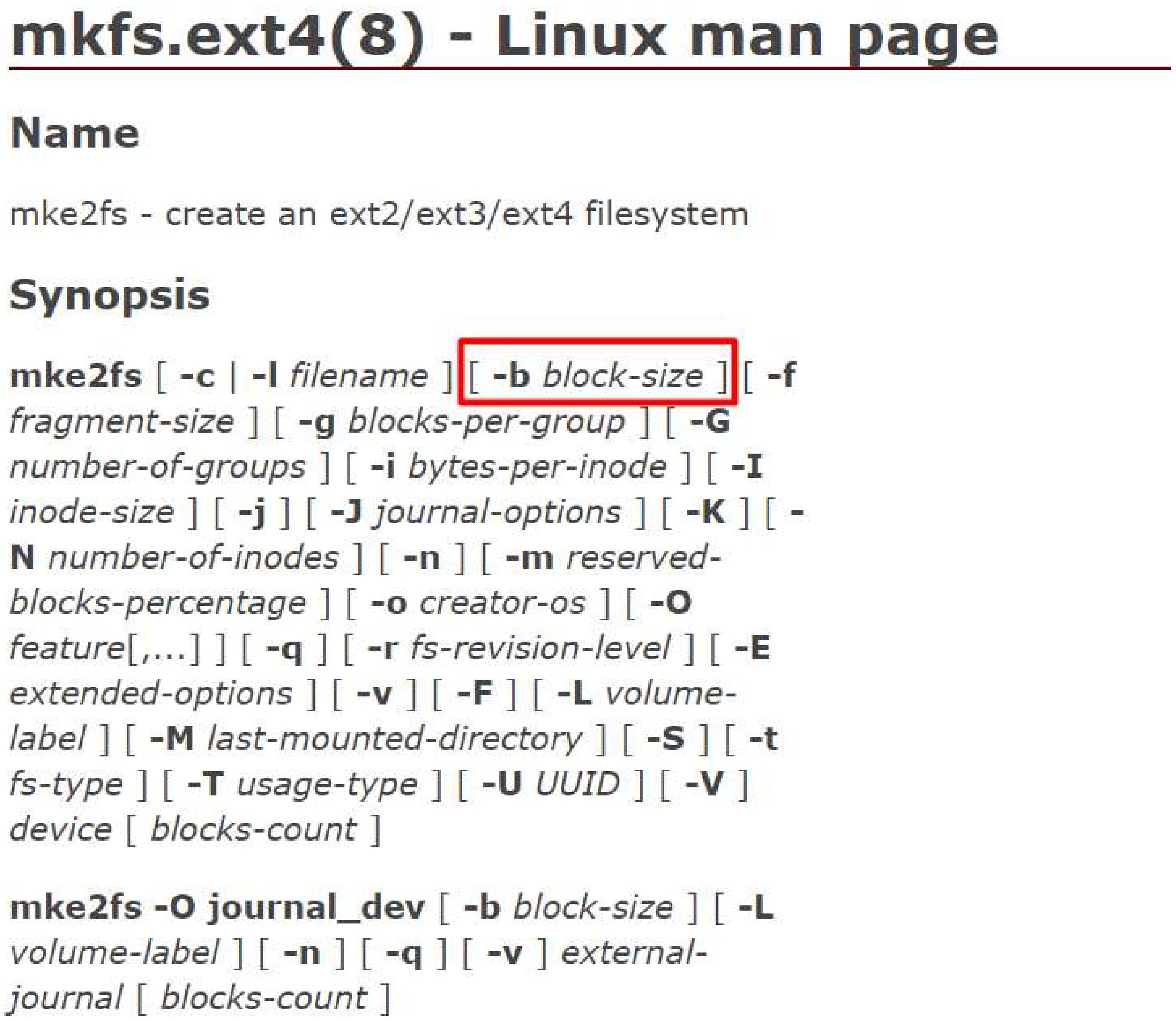
NAME top

```
       mkfs.xfs - construct an XFS filesystem
```

SYNOPSIS top

```
       mkfs.xfs [ -b block_size_options ] [ -m global_metadata_options ]
       [ -d data_section_options ] [ -f ] [ -i inode_options ] [ -l
       log_section_options ] [ -n naming_options ] [ -p protofile ] [ -q
       ] [ -r realtime_section_options ] [ -s sector_size_options ] [ -L
       label ] [ -N ] [ -K ] device
       mkfs.xfs -V
```

ext4 format



mkfs.ext4(8) - Linux man page

Name

mke2fs - create an ext2/ext3/ext4 filesystem

Synopsis

mke2fs [**-c** | **-l** *filename*] [**-b** *block-size*] [**-f** *fragment-size*] [**-g** *blocks-per-group*] [**-G** *number-of-groups*] [**-i** *bytes-per-inode*] [**-I** *inode-size*] [**-j**] [**-J** *journal-options*] [**-K**] [**-N** *number-of-inodes*] [**-n**] [**-m** *reserved-blocks-percentage*] [**-o** *creator-os*] [**-O** *feature*[,...]] [**-q**] [**-r** *fs-revision-level*] [**-E** *extended-options*] [**-v**] [**-F**] [**-L** *volume-label*] [**-M** *last-mounted-directory*] [**-S**] [**-t** *fs-type*] [**-T** *usage-type*] [**-U** *UUID*] [**-V**] *device* [*blocks-count*]

mke2fs -O journal_dev [**-b** *block-size*] [**-L** *volume-label*] [**-n**] [**-q**] [**-v**] *external-journal* [*blocks-count*]

Allocation Methods

- An allocation method refers to how disk blocks are allocated for files
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation
- Each method needs an appropriate way to access file

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Simple
  - only starting location (block #) and length (number of blocks) are required
- Random access
- Wasteful of space
  - dynamic storage-allocation problem
- Files cannot grow

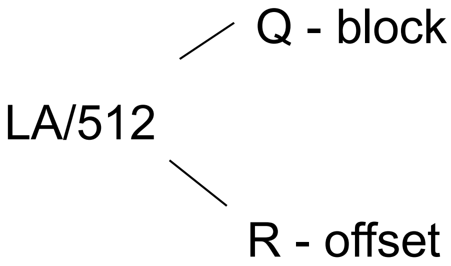
Mapping from logical add into block/offset

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	…
0				1				2				3				

- Logical address = x
- Q (blockid) = x / block_size
- R (offset) = x % block_size
- X = 9 => Q= 9/4 = 2; R= 9%2 =1

Contiguous Allocation

- Mapping from logical to physical
  - LA is position to access
  -



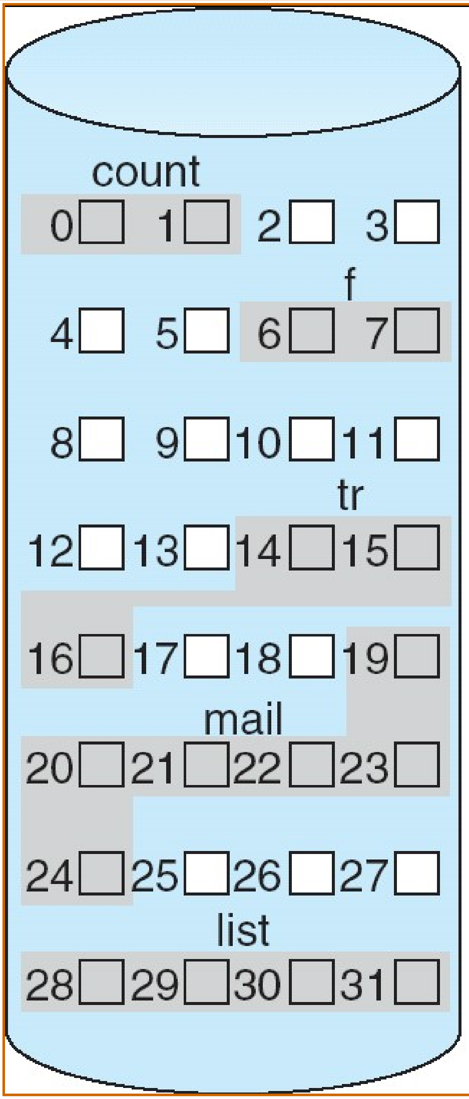
Q - block

LA/512

R - offset

Suppose block size is 512KB
Block to be accessed = Q + starting address
Displacement/offset into block = R

Contiguous Allocation of Disk Space



directory

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

Question

- A system using contiguous allocation
  - block size is 2KB
  - a file has the length of 12.5MB
- Which of the following is the correct location of file at position 50.5KB
  A. block 25, offset 512
  B. block 24, offset 1024
  C. block 25, offset 510
  D. block 24, offset 2512

Extent-Based Systems

- Many newer file systems (I.e. Veritas File System)
  - use a modified contiguous allocation scheme
- Extent-based file systems
  - allocate disk blocks in **extents**
- An **extent** is a contiguous blocks of disk
  - Extents are allocated for file allocation
  - A file consists of one or more extents
  - extents of a file are not necessarily contiguous

Question

- An extent-based file system
  - an extent has *100* blocks
  - a block has size *2KB*
  - a file has size 25.3MB
- Which of the following is the correct extent number (started by 0) of file at position 15MB?
  A. 74
  B. 75
  C. 76
  D. 77

Question

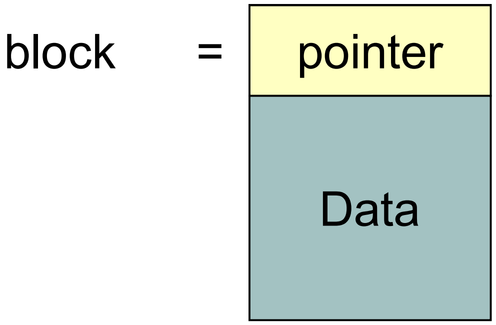
- An extent-based file system
  - an extent has *100* blocks
  - a block has size *2KB*
  - a file has size 25.3MB
- Which of the following is the correct block (started by 0) number in the extent containing the data of file at position 15MB?
  - A. 77
  - B. 78
  - C. 79
  - D. 80

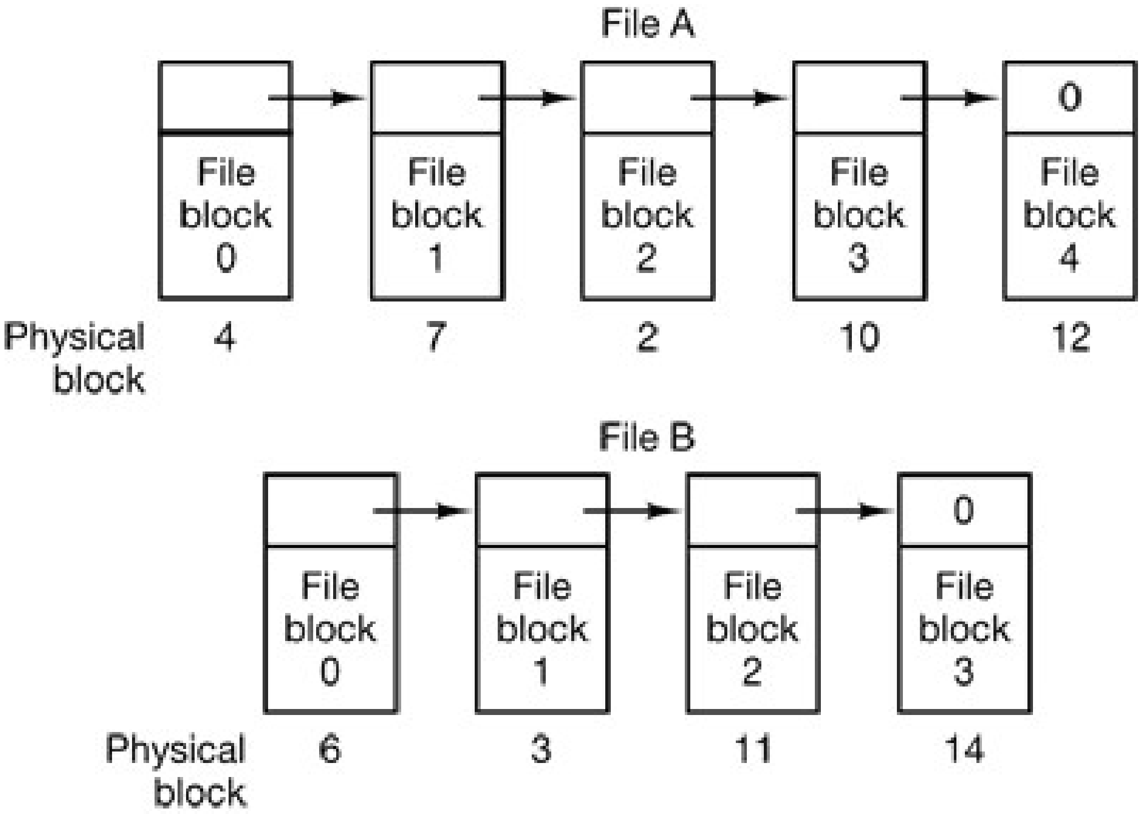
Question

- An extent-based file system
  - an extent has *100* blocks
  - a block has size *2KB*
  - a file has size 25.3MB
- Which of the following is the correct offset of the of file at position 15MB in the block containing data?
  A. 0
  B. 2047
  C. 2048
  D. 512

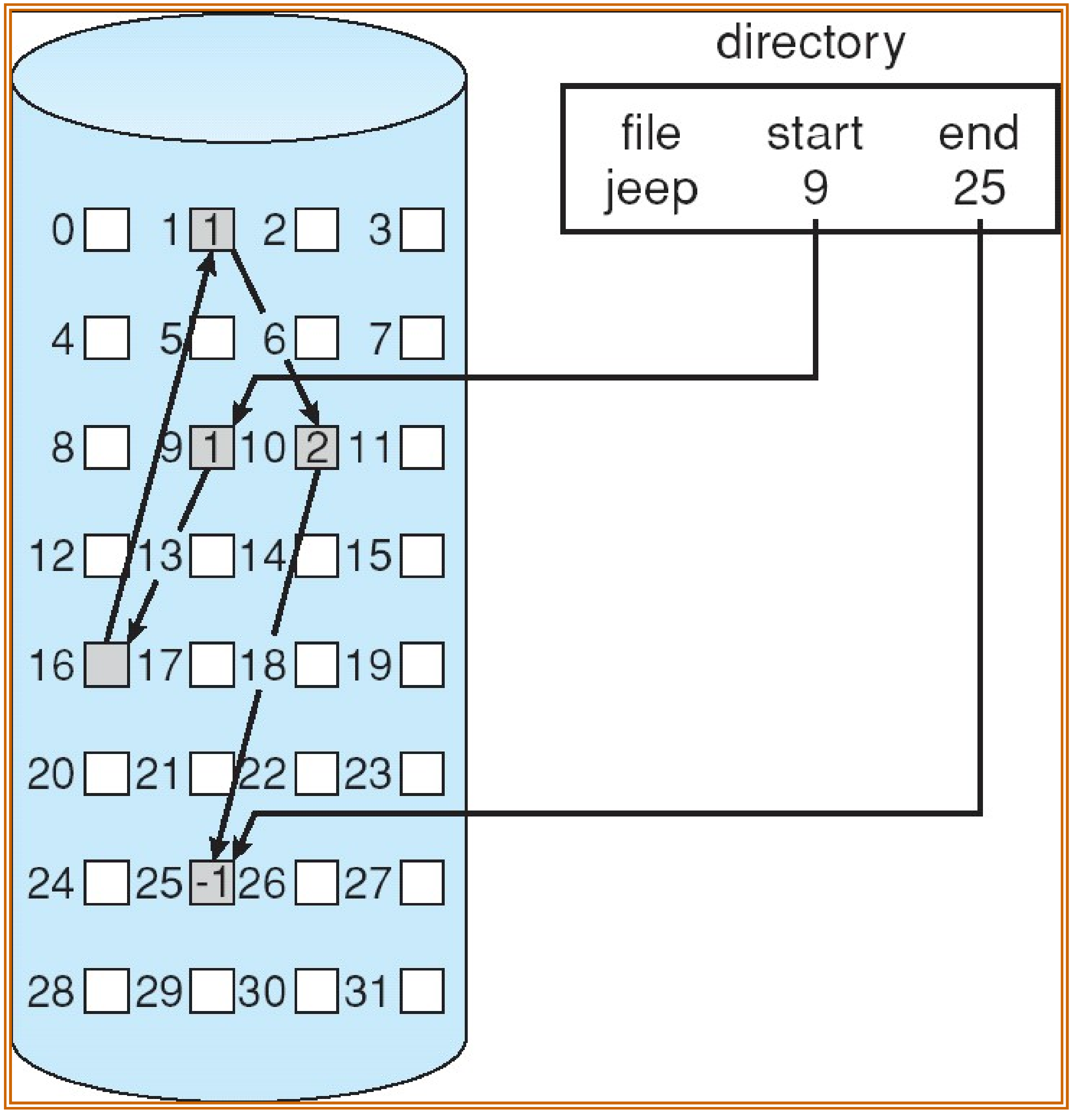
Linked Allocation

- Each file is a linked list of disk blocks:
  - blocks may be scattered anywhere on the disk.





Linked Allocation



Question

- A system uses linked list allocation
  - partition size 500GB (1GB=1024MB,1MB=1024KB,…)
  - block size *1*KB
- Which of the following is the correct size of the pointer of each block?
  - A. short (2 bytes)
  - B. int (4 bytes)
  - C. float (4 bytes)
  - D. long (8 bytes)

Question

- A system uses linked list allocation
- Which of the following is the correct reason for no direct access?

A. because we don't know the location of block *n* directly

B. because data blocks of a file may be scattered

C. because of security reason

D. because the information of data block location is hidden

Question

- A system uses linked list allocation
- Which of the following is the correct way to access block *n* of a file?

A. read the first block to find the location of block *n*

B. look up the location of block *n* from a table

C. recursively read block *n*-1 to find out the location of block *n*

D. there is no way to find the location of block *n*

Linked Allocation (Cont.)

- Simple
  - need only starting address
- Free-space management system
  - no waste of space
- No random access
  - File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

Question

- A system uses linked list allocation
  - block size 2KB
  - pointer size 4 bytes
  - file size 15.4MB
- Which of the following is the correct block of file at position 15.25KB?

A. 7

B. 8

C. 9

D. 10

Question

- A system uses linked list allocation
  - block size 2KB
  - pointer size 4 bytes
  - file size 15.4MB
- Which of the following is the correct offset in the block of file at position 15.25KB?

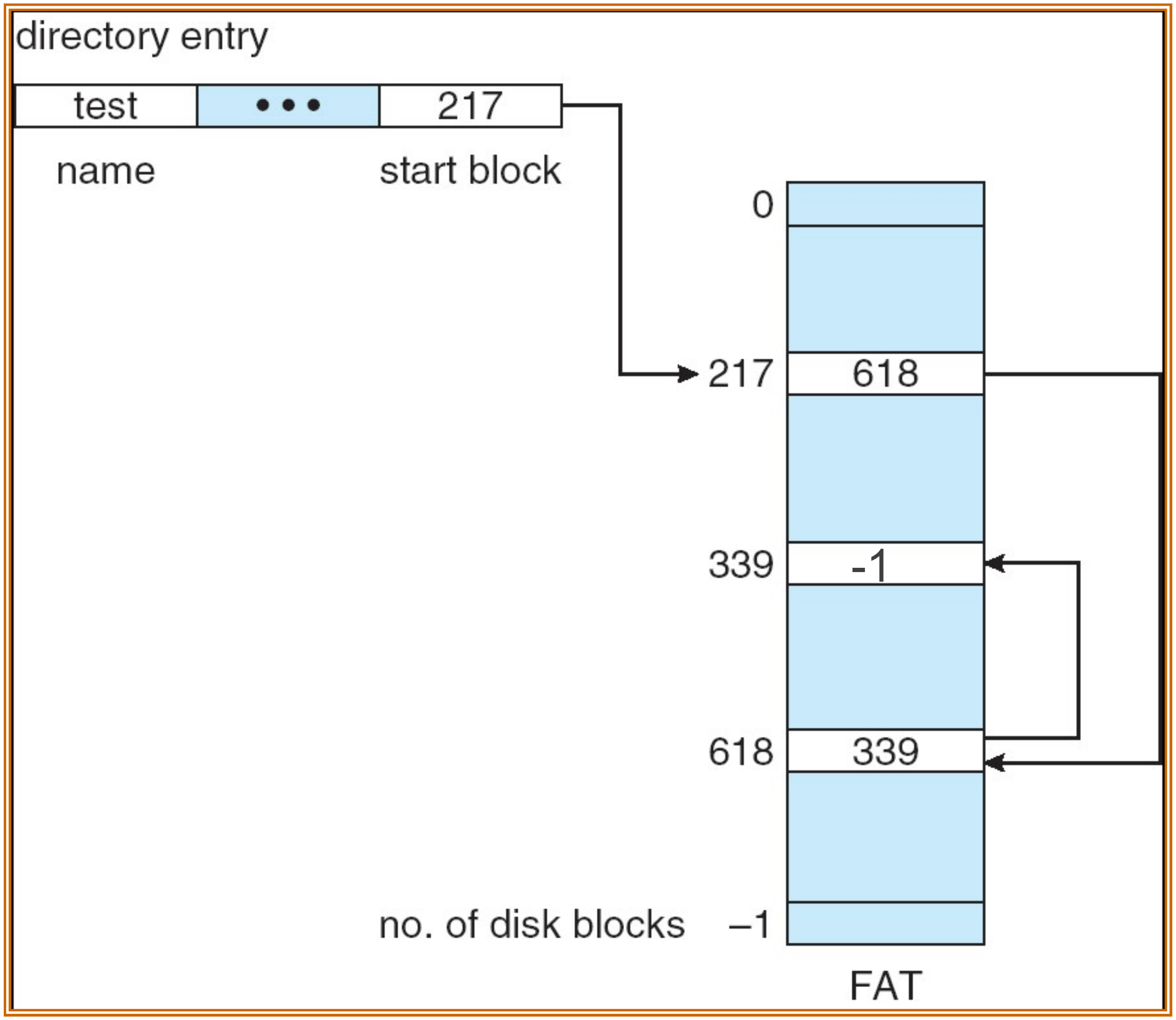
  A. 1311

  B. 1312

  C. 1313

  D. 1314

File-Allocation Table (DOS)



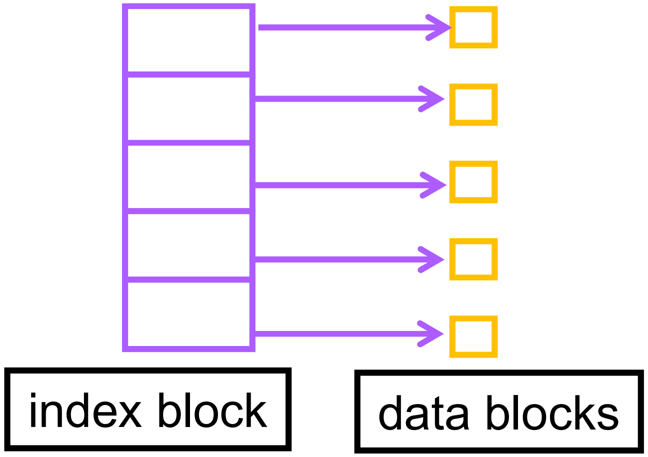
- Separate the pointers from data
- pointers are stored in File Allocation Table (FAT)
- The system stores two identical copies of FAT

Question

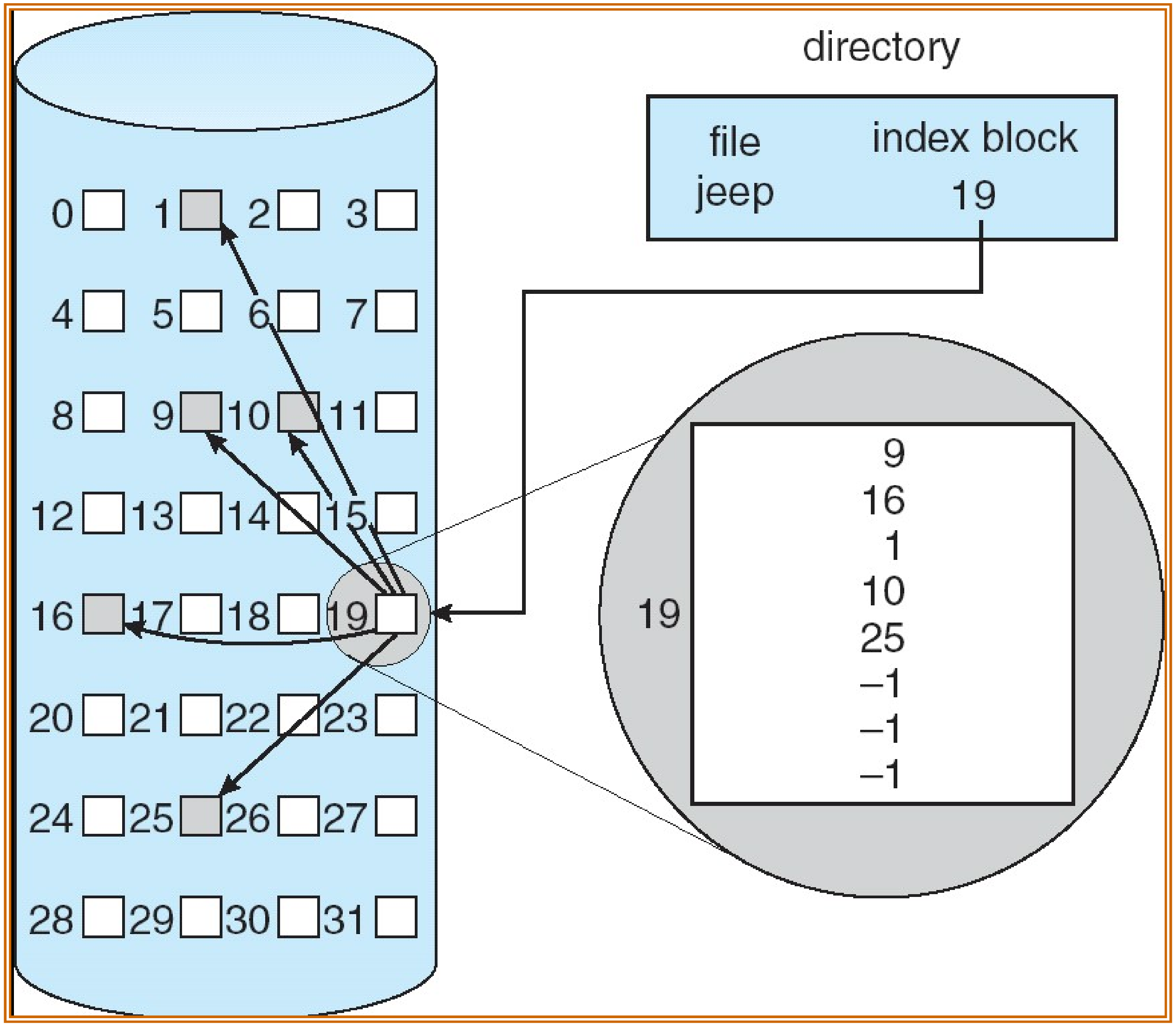
- Which of the following is incorrect about FAT?
  A. it is fast to access the block *n* of a file
  B. it is slow to access the block *n* of a file
  C. if FAT is corrupted the whole partition is corrupted
  D. the system keeps two copies of FAT in order to reduce the risk of FAT corruption

Indexed Allocation

- Brings all pointers together into the *index block*
- Logical view



Example of Indexed Allocation



Indexed Allocation (Cont.)

- Need index block
- Random access
- Dynamic allocation without external fragmentation
  - have overhead of index block

Question

- System uses indexed allocation
  - block size 4KB
  - pointer size 4 bytes
- Which of the following is the correctly maximum file size?
  A. 4MB
  B. 8MB
  C. 16MB
  D. 32MB

Question

- System uses indexed allocation
  - block size 4KB
  - pointer size 4 bytes
  - file size 3MB
- Which of the following is the correct block (starting from 0) and offset at file position 35KB?

A. (*block*, *offset*)=(9, 3071)

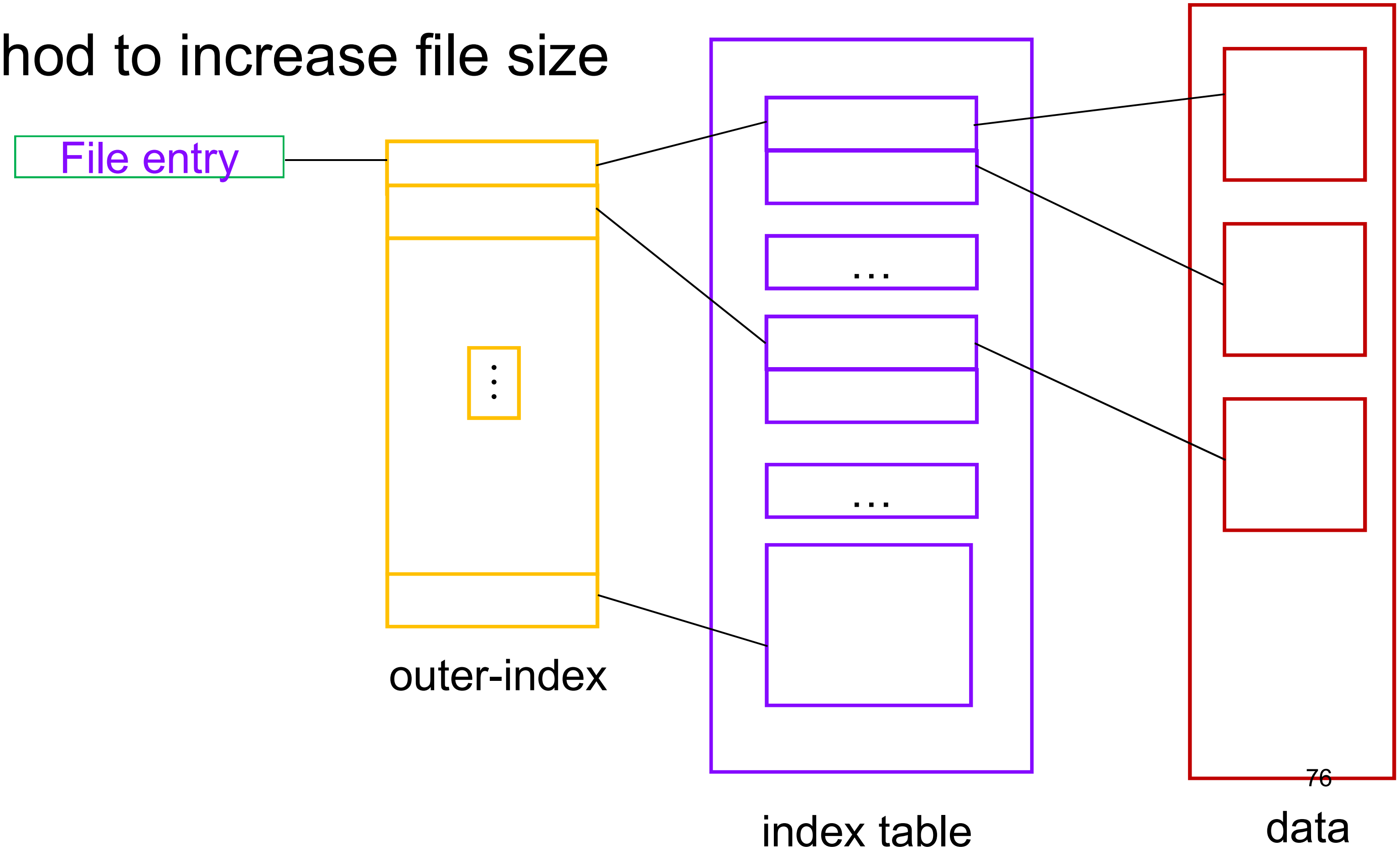
B. (*block*, *offset*)=(9, 3070)

C. (*block*, *offset*)=(8, 3072)

D. (*block*, *offset*)=(8, 3070)

Indexed Allocation (cont'd)

- Two-level index
  - Two level of index block
  - Method to increase file size



Address locating

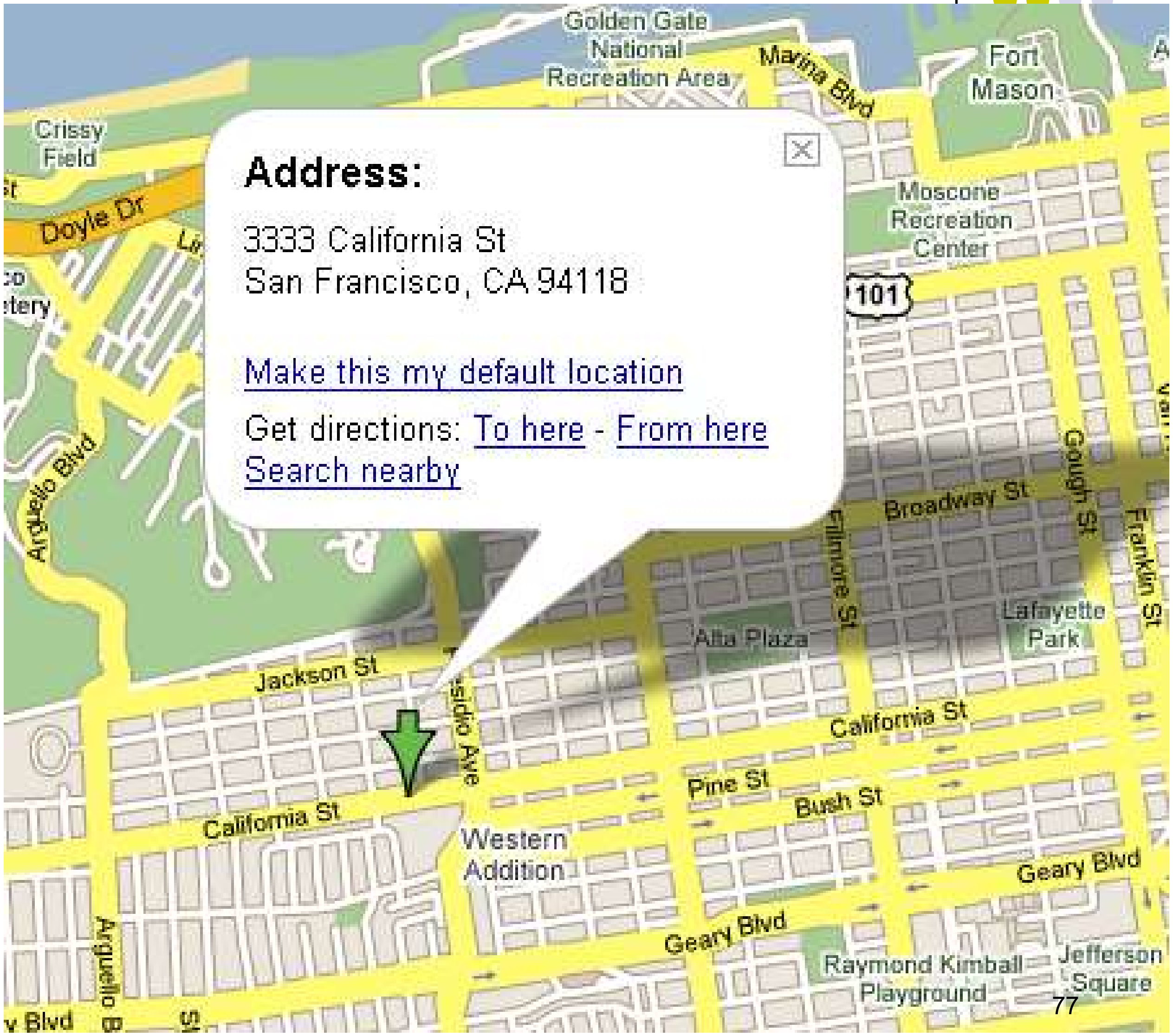
Go to USA →

San Francisco

→

California St

→ 3333



Address:

3333 California St
San Francisco, CA 94118

Make this my default location

Get directions: To here - From here

Search nearby

Indexed Allocation (cont'd)

- A system uses two-level index
  - block size 4KB
  - pointer size 4 Bytes
- Which of the following is the correct maximum file size?
  - A. 4GB
  - B. 1GB
  - C. 8GB
  - D. 2GB

Indexed Allocation (cont'd)

- A system uses two-level index
- Which of the following is the correct steps to locate the data at file position *n*?

A. Identify block number → block number in block table → offset

B. Identify block number in outer index block → block number → offset

C. Identify offset → block number

D. none of the above

Indexed Allocation (cont’d)

- A system uses two-level index
  - block size 4KB
  - pointer size 4 Bytes
  - File size 20MB
- Which is the correct address of file at position 15MB?

A. (4,3072,0)

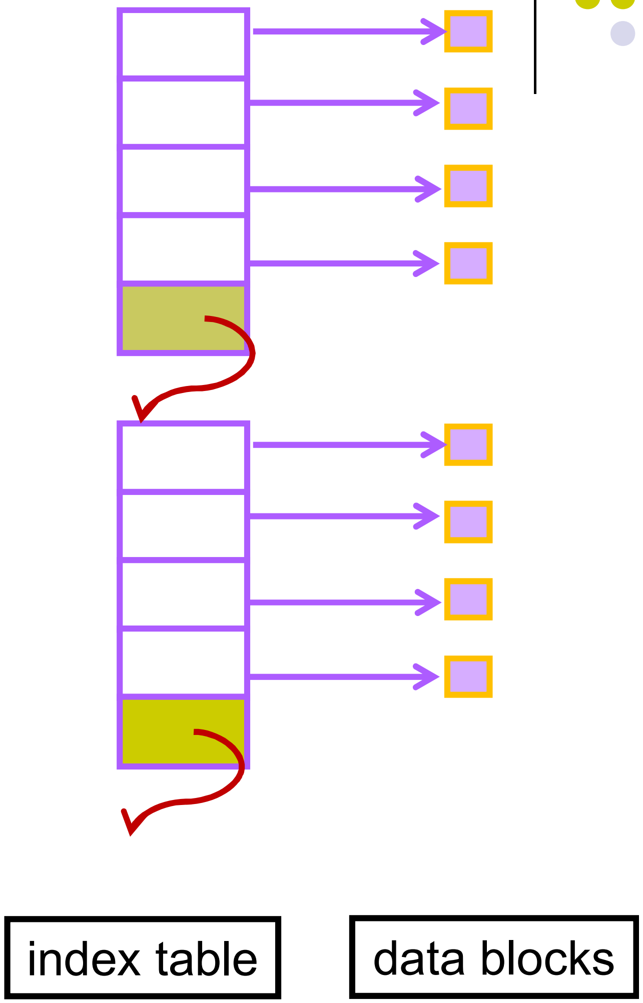
B. (3,768,1023)

C. (3,768,0)

D. (3,3072,0)

Indexed Allocation (Cont.)

- Linked scheme
  - no limit on file size
- Combine linked list with index block
  - index blocks are linked
  - last pointer of a index block is the address of the next one



Question

- Linked index block allocation
  - block size 2KB
  - pointer size 4 bytes
  - File size 20MB
- Which of the following is the correct steps to read file at position 15.5MB

  A. identify index block → offset → block number → read

  B. identify offset → block number → index block → read

  C. identify offset → index block → block number → read

  D. identify index block → block number →offset → read

Question

- Linked index block allocation
  - block size 2KB
  - pointer size 4 bytes
  - File size 20MB
- Which of the following is the correct index block number at file position 15.5MB (start from 0)

Question

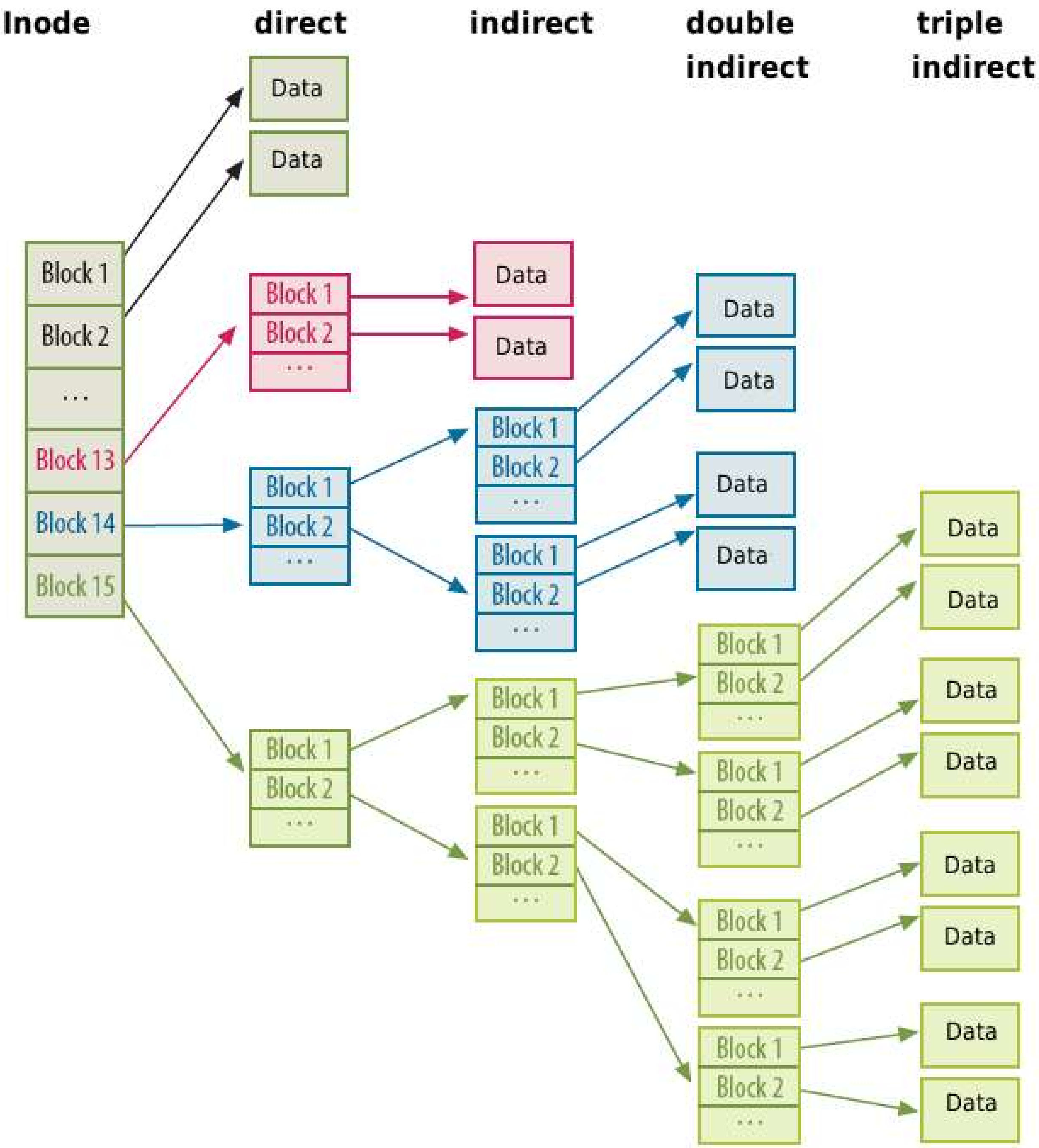
- Linked index block allocation
  - block size 2KB
  - pointer size 4 bytes
  - File size 20MB
- Which of the following is the correct index block number at file position 15.5MB (start from 0)

A. 13
B. 14
C. 15
D. 16

Question

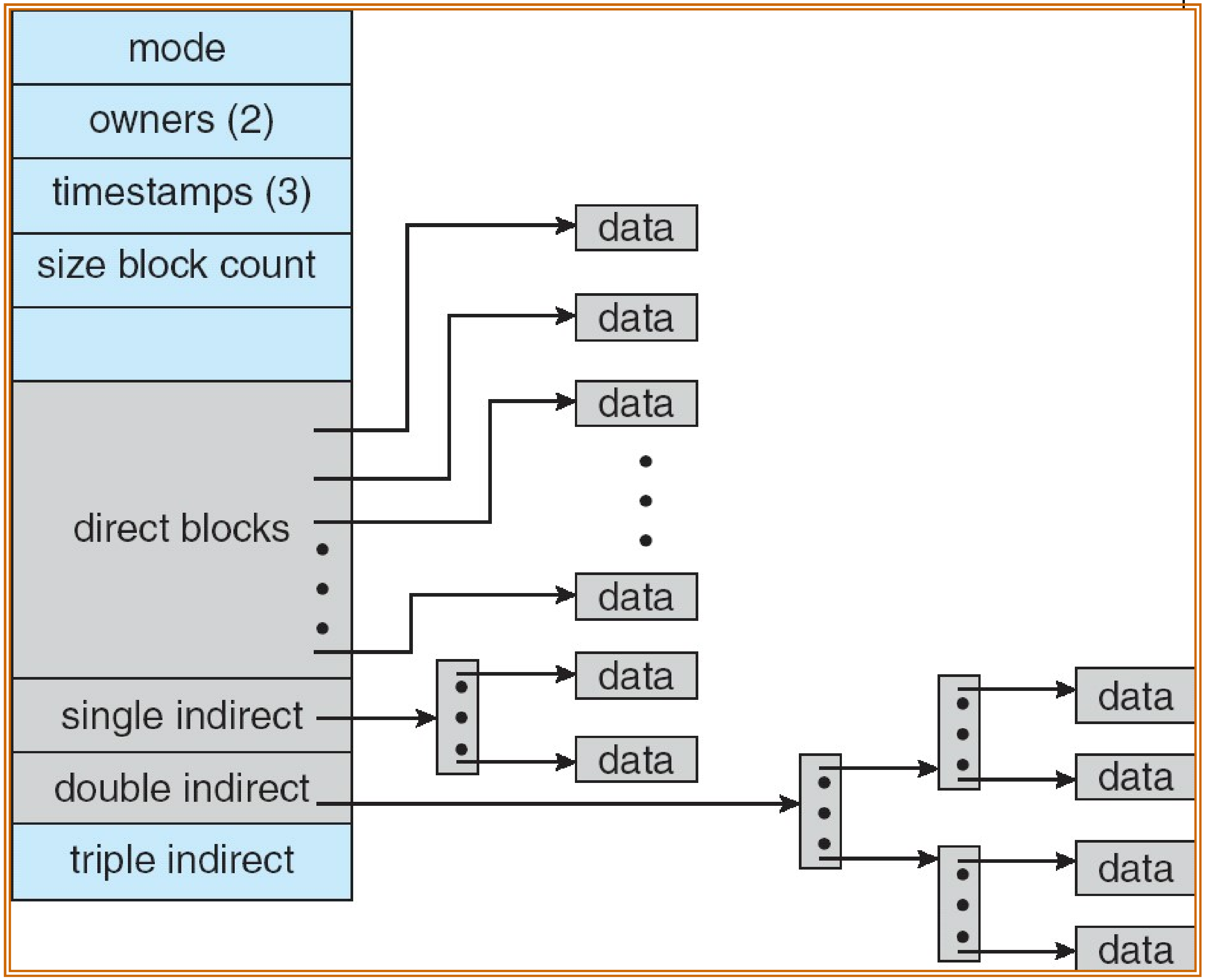
- Linked index block allocation
  - block size 2KB
  - pointer size 4 bytes
  - File size 20MB
- Which of the following is the correct block number and offset at file position 15.5MB
  - A. (*block*, *offset*)=(271, 2047)
  - B. (*block*, *offset*)=(271, 0)
  - C. (*block*, *offset*)=(270, 2047)
  - D. (*block*, *offset*)=(270, 0)

Combined Scheme: UNIX (4K bytes per block)



LINUX allocation

(10 direct pointers)



Question

- A UNIX system
  - pointer size *4* bytes
  - block size *4* KB
  - 12 direct pointers, 1 single indirect, 1 double indirect, 1 triple indirect pointers
- Which of the following is the correct maximum file size?

A. $(12+2^{10}+2^{20}+2^{30})$KB

B. $4*(2^{10}+2^{20}+2^{30})$KB

C. 2^{32}KB

D. $4*(12+2^{10}+2^{20}+2^{30})$KB

Question

- A UNIX system
  - pointer size *4* bytes
  - block size *4* KB
  - 12 direct pointer, 1 single indirect, 1 double indirect, 1 triple indirect pointers
- Which of the following is the correct maximum number of indexed blocks?

A. $(1+2^{10}+2^{20})$

B. $(2+2^{10}+2^{20})$

C. $(3+2^{11}+2^{20})$

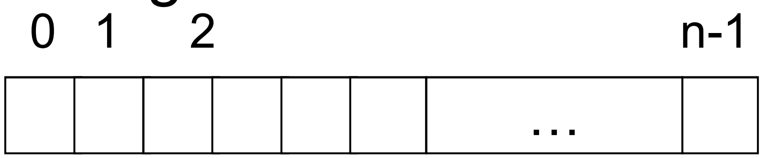
D. 2^{20}

Question

- A UNIX system
  - pointer size *4* bytes
  - block size *4* KB
  - 12 direct pointer, 1 single indirect, 1 double indirect, 1 triple indirect pointers
  - File size 78MB
- Which of the following is the correct location at file position 95KB?
  - A. triple indirect block, (*block*, *offset*)=(12,3071)
  - B. double indirect block, (*block*, *offset*)=(12,3071)
  - C. single indirect block, (*block*, *offset*)=(11,3071)
  - D. single indirect block, (*block*, *offset*)=(11,3072)

Free-Space Management

- Bit vector (*n* blocks), e.g. Linux (ext3)
  - Easy to get contiguous blocks



$$\text{bit}[i] = \begin{cases} 0 \Rightarrow \text{block}[i] \text{ free} \\ 1 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

- Bit map requires extra space

  Example:

  block size = 2^{12} bytes

  disk size = 2^{30} bytes (1 gigabyte)

  $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

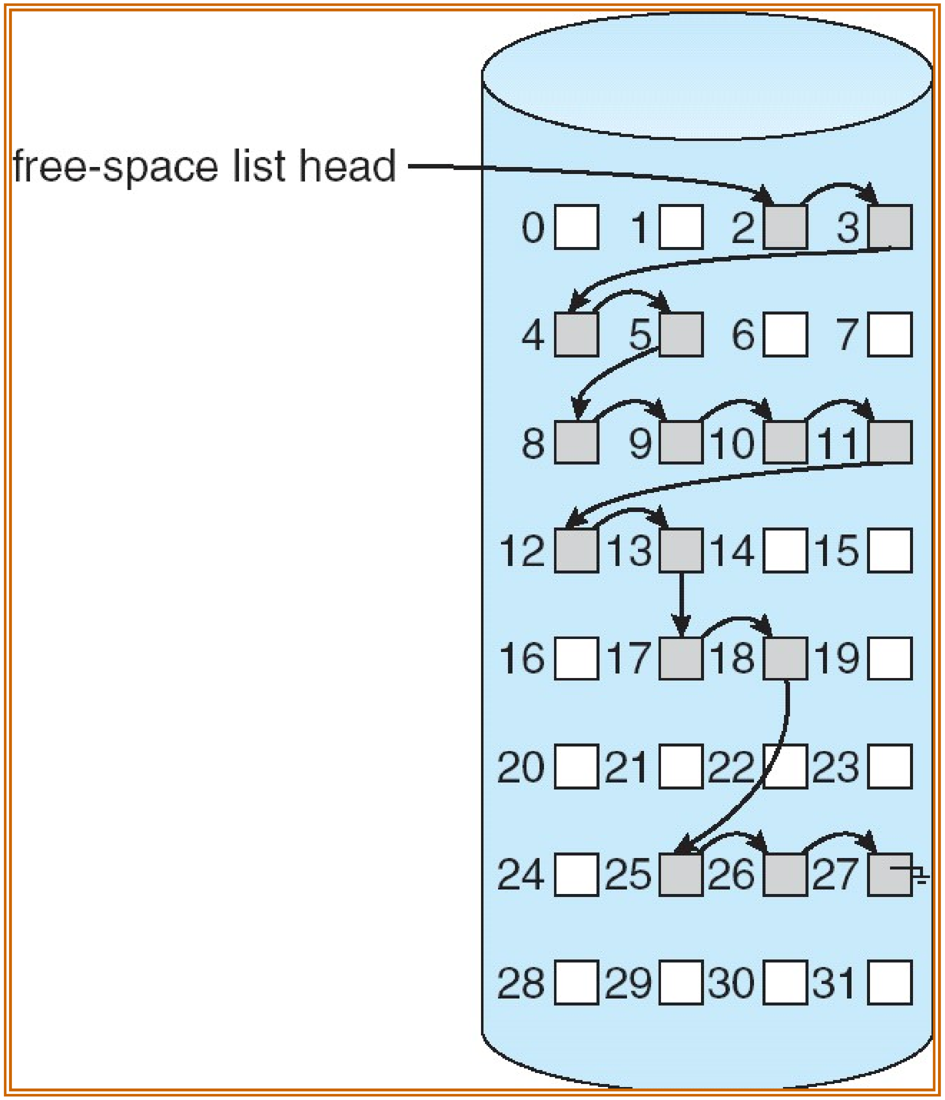
Free-Space Management (Cont.)

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
- Grouping
  - Use linked blocks to store pointers to free blocks
  - Last pointer in a block is the address of the next one
- Counting
  - several contiguous blocks are freed/allocated for a file
  - each entry has
    - address of the first free block
    - number of contiguously free blocks

Free-Space Management (Cont.)

- Need to protect:
  - Pointer to free list
  - Bit map
    - Must be kept on disk
    - Copy in memory and disk may differ
    - Cannot allow for block[*i*] to have a situation where bit[*i*] = 1 in memory and bit[*i*] = 0 on disk
  - Solution:
    - Set bit[*i*] = 1 in disk
    - Allocate block[*i*]
    - Set bit[*i*] = 1 in memory

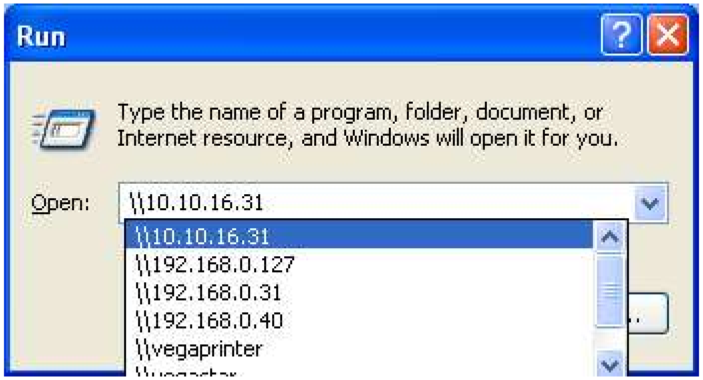
Linked Free Space List on Disk



Shared file systems

- A file system is shared to other machines
  - the file system may be large and powerful
  - file sharing is needed in many applications
  - available in many systems, e.g., Windows, Linux (NFS)

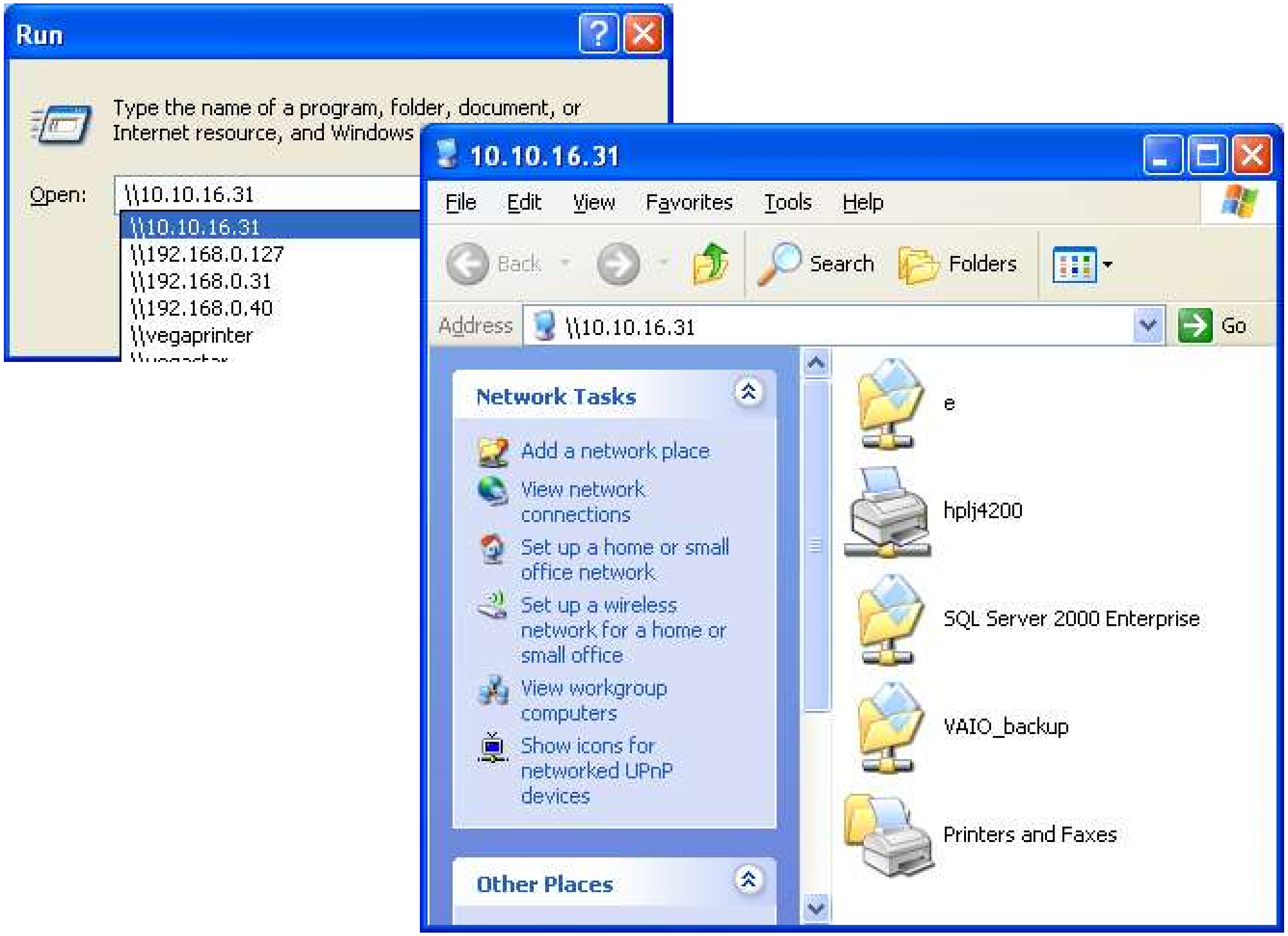
Shared file systems



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

```
\\10.10.16.31
\\10.10.16.31
\\192.168.0.127
\\192.168.0.31
\\192.168.0.40
\\vegaprinter
```

Shared file systems



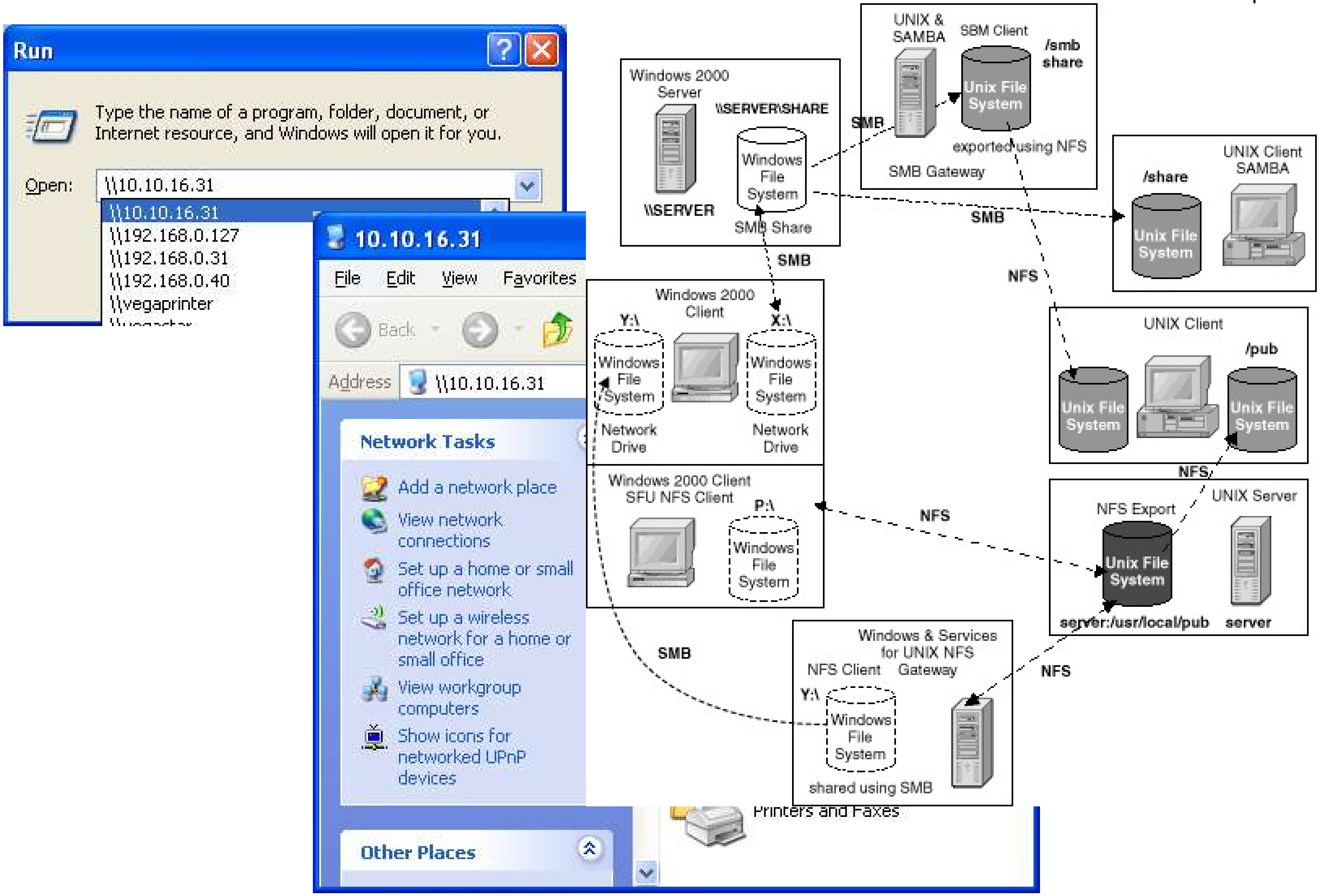
Type the name of a program, folder, document, or Internet resource, and Windows

Open: \\10.10.16.31

\\10.10.16.31
\\192.168.0.127
\\192.168.0.31
\\192.168.0.40
\\vegaprinter

Address \\10.10.16.31

Shared file systems



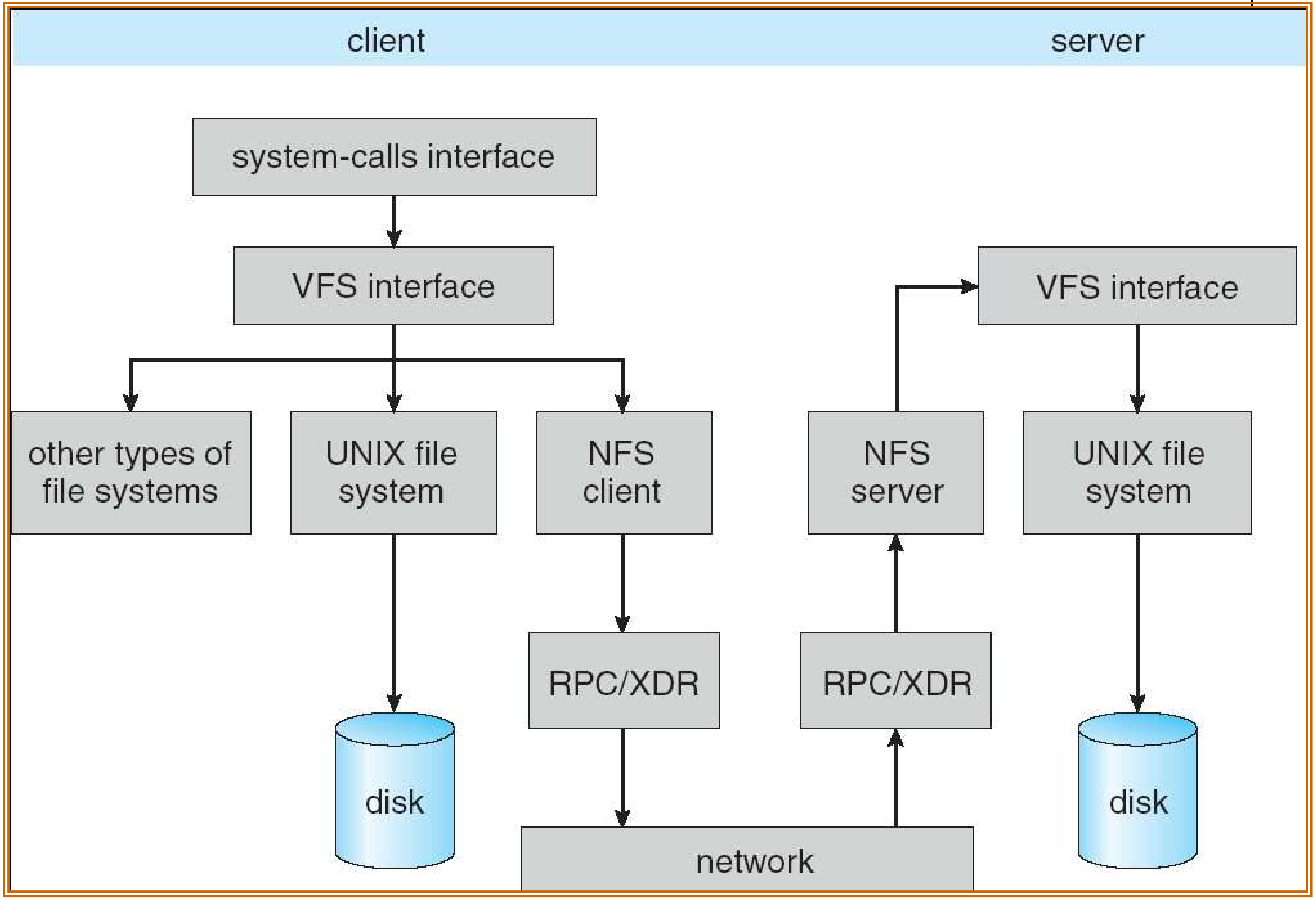
Run

Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: \\10.10.16.31

\\10.10.16.31
\\192.168.0.127
\\192.168.0.31
\\192.168.0.40
\\vegaprinter

Network File System (NFS) Architecture



Question?