File Systems

Long-term Information Storage

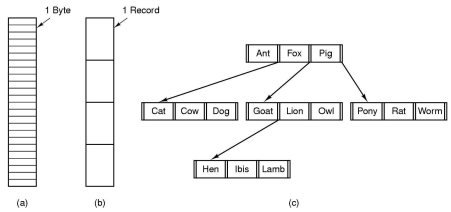
1. Must store large amounts of data
2. Information stored must survive the termination of the process using it
3. Multiple processes must be able to access the information concurrently

File Naming

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

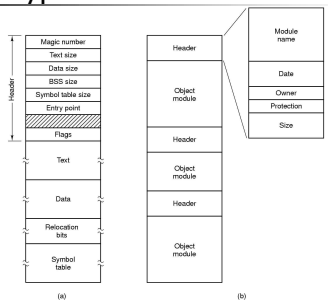
Typical file extensions.

File Structure



- Three kinds of files
  - byte sequence
  - record sequence
  - tree

File Types



(a) An executable file (b) An archive

File Access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - convenient when medium was mag tape
- Random access
  - bytes/records read in any order
  - essential for data base systems
  - read can be …
    - move file marker (seek), then read or …
    - read and then move file marker

File Attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Possible file attributes

File Operations

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set Attributes
11. Rename

An Example Program Using File System Calls (1/2)

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                  /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);       /* ANSI prototype */

#define BUF_SIZE 4096                   /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                /* protection bits for output file */

int main(int argc, char *argv[])
{
     int in_fd, out_fd, rd_count, wt_count;
     char buffer[BUF_SIZE];

     if (argc != 3) exit(1);            /* syntax error if argc is not 3 */
```

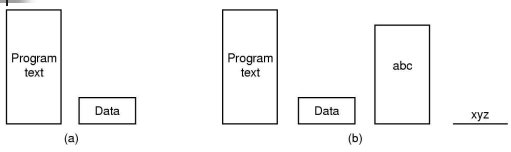

An Example Program Using File System Calls (2/2)

```
     /* Open the input file and create the output file */
     in_fd = open(argv[1], O_RDONLY);    /* open the source file */
     if (in_fd < 0) exit(2);             /* if it cannot be opened, exit */
     out_fd = creat(argv[2], OUTPUT_MODE);  /* create the destination file */
     if (out_fd < 0) exit(3);            /* if it cannot be created, exit */

     /* Copy loop */
     while (TRUE) {
          rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
          if (rd_count <= 0) break;      /* if end of file or error, exit loop */
          wt_count = write(out_fd, buffer, rd_count); /* write data */
          if (wt_count <= 0) exit(4);    /* wt_count <= 0 is an error */
     }

     /* Close the files */
     close(in_fd);
     close(out_fd);
     if (rd_count == 0)                  /* no error on last read */
          exit(0);
     else
          exit(5);                       /* error on last read */
}
```

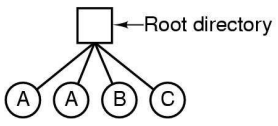

Memory-Mapped Files



(a) Segmented process before mapping files into its address space

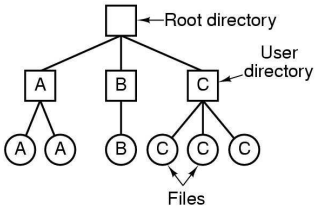
(b) Process after mapping
existing file *abc* into one segment
creating new segment for *xyz*

Directories: Single-Level Directory Systems



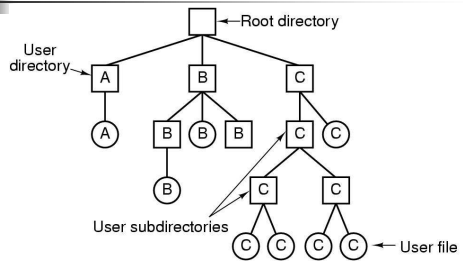
- A single level directory system
  - contains 4 files
  - owned by 3 different people, A, B, and C

Two-level Directory Systems



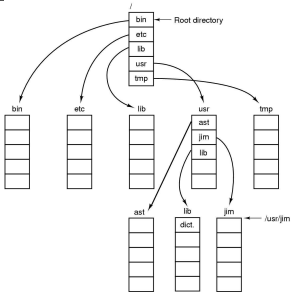
Letters indicate *owners* of the directories and files

Hierarchical Directory Systems



A hierarchical directory system

Path Names

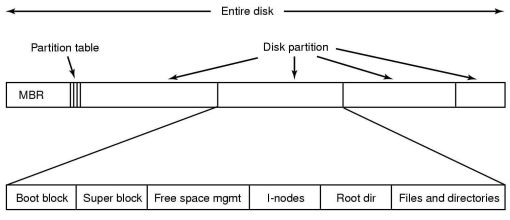


A UNIX directory tree

Directory Operations

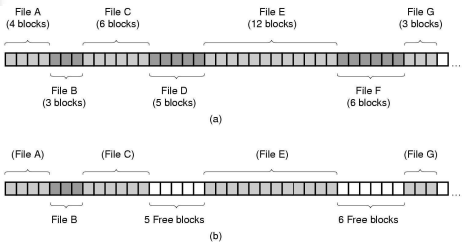
1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename
7. Link
8. Unlink

File System Implementation



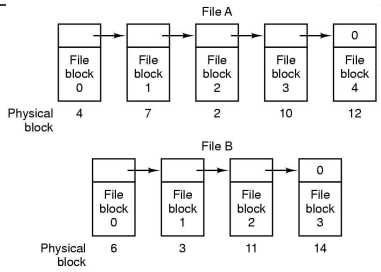
A possible file system layout

Implementing Files (1)



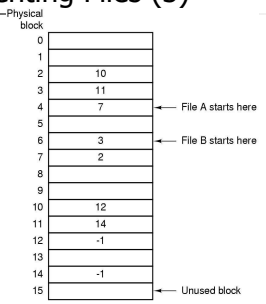
(a) Contiguous allocation of disk space for 7 files
(b) State of the disk after files *D* and *E* have been removed

Implementing Files (2)



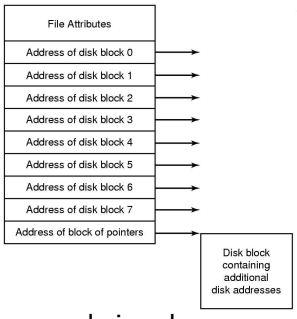
Storing a file as a linked list of disk blocks

Implementing Files (3)



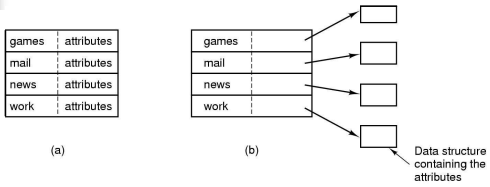
Linked list allocation using a file allocation table in RAM

Implementing Files (4)



An example i-node

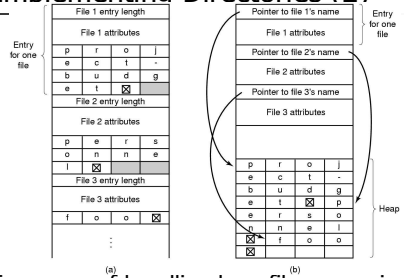
Implementing Directories (1)



(a) A simple directory
- fixed size entries
- disk addresses and attributes in directory entry

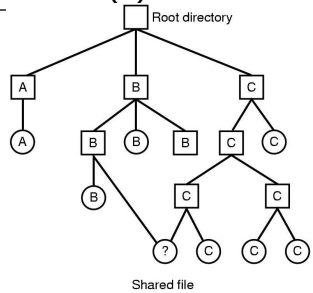
(b) Directory in which each entry just refers to an i-node

Implementing Directories (2)



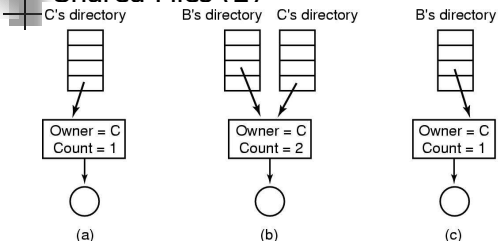
- Two ways of handling long file names in directory
  - (a) In-line
  - (b) In a heap

Shared Files (1)



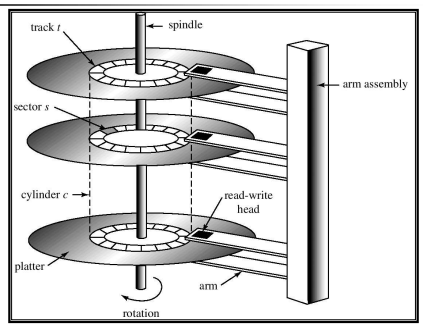
File system containing a shared file

Shared Files (2)

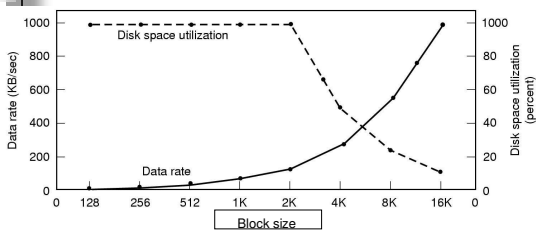


(a) Situation prior to linking
(b) After the link is created
(c) After the original owner removes the file

Disk structure

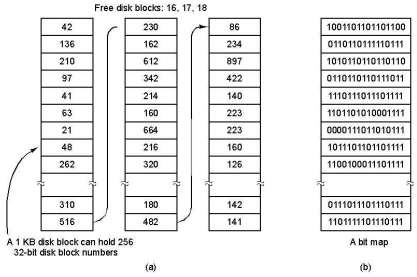


Disk Space Management (1)



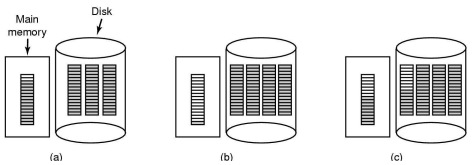
- Dark line (left hand scale) gives data rate of a disk
- Dotted line (right hand scale) gives disk space efficiency
- All files 2KB

Disk Space Management (2)



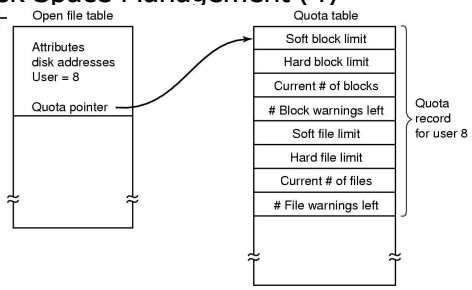
(a) Storing the free list on a linked list
(b) A bit map

Disk Space Management (3)



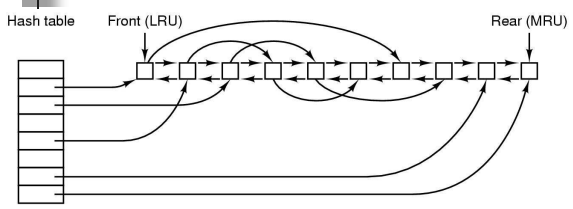
1. Almost-full block of pointers to free disk blocks in RAM
   - three blocks of pointers on disk
2. Result of freeing a 3-block file
3. Alternative strategy for handling 3 free blocks
   - - shaded entries are pointers to free disk blocks

Disk Space Management (4)



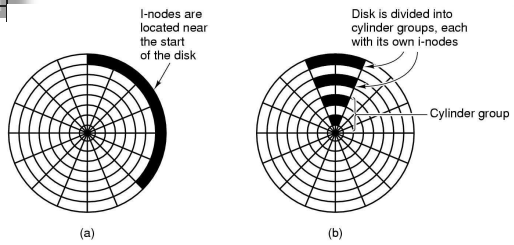
Quotas for keeping track of each user's disk use

File System Performance (1)



The block cache data structures

File System Performance (2)



- I-nodes placed at the start of the disk
- Disk divided into cylinder groups
  - each with its own blocks and i-nodes

Log-Structured File Systems

- With CPUs faster, memory larger
  - disk caches can also be larger
  - increasing number of read requests can come from cache
  - thus, most disk accesses will be writes
- LFS Strategy structures entire disk as a log
  - have all writes initially buffered in memory
  - periodically write these to the end of the disk log
  - when file opened, locate i-node, then find blocks