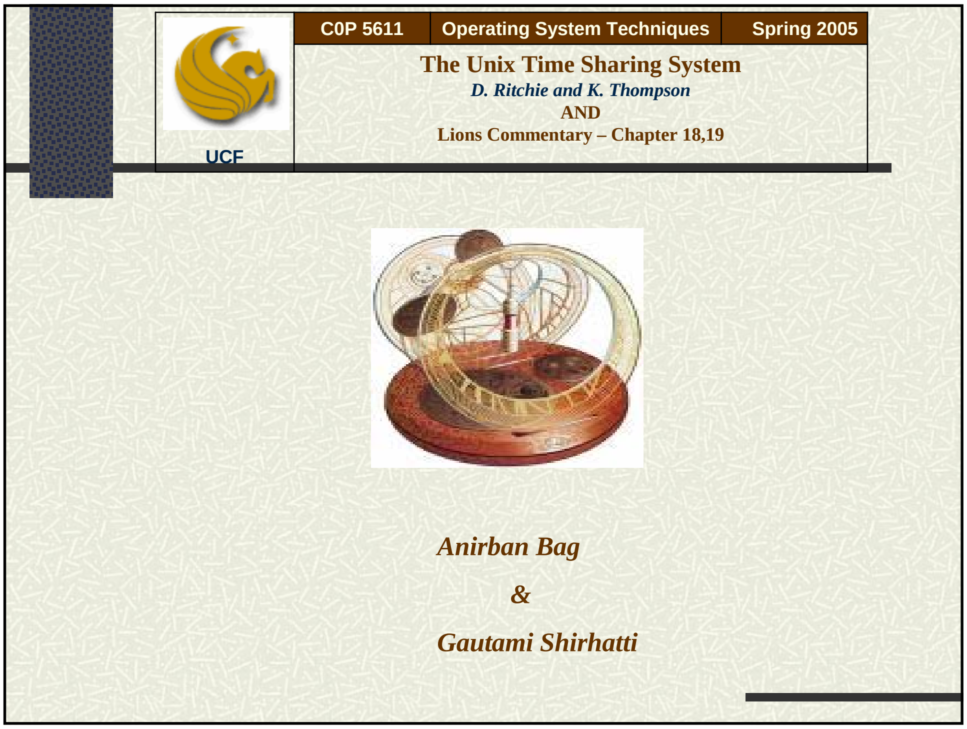


UCF

C0P 5611 | Operating System Techniques | Spring 2005

The Unix Time Sharing System

D. Ritchie and K. Thompson

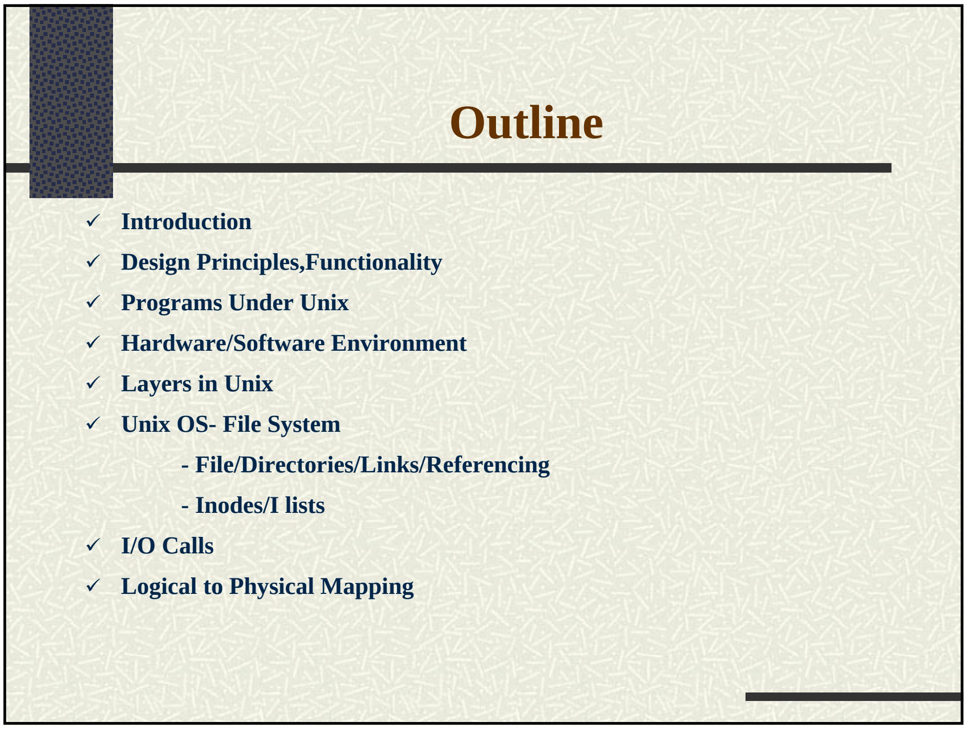
AND

Lions Commentary – Chapter 18,19

Anirban Bag

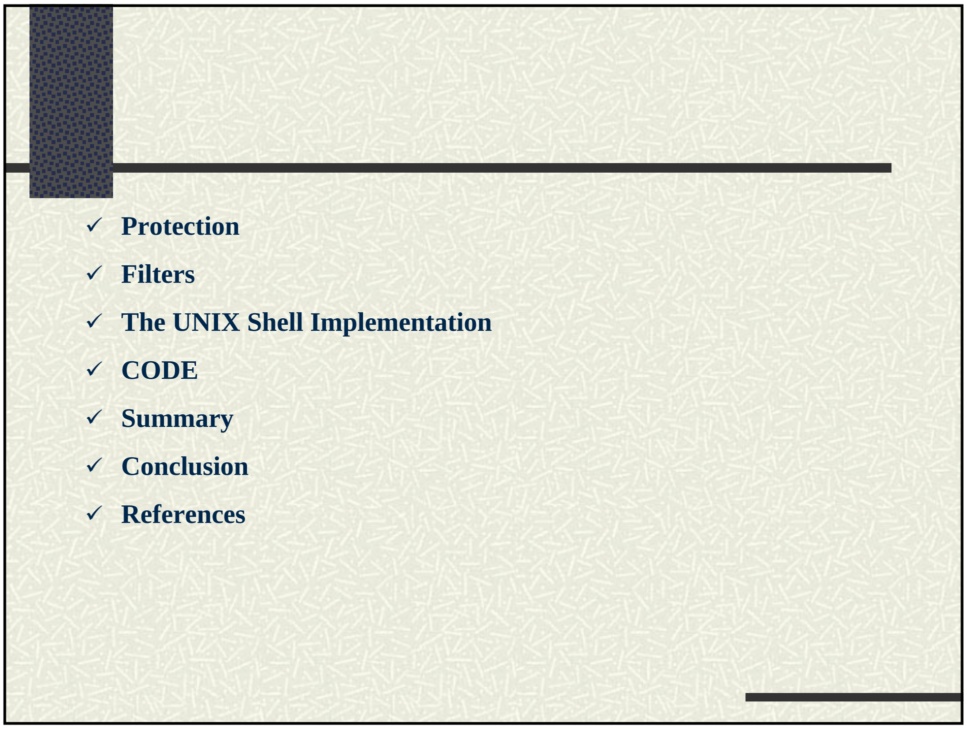
&

Gautami Shirhatti

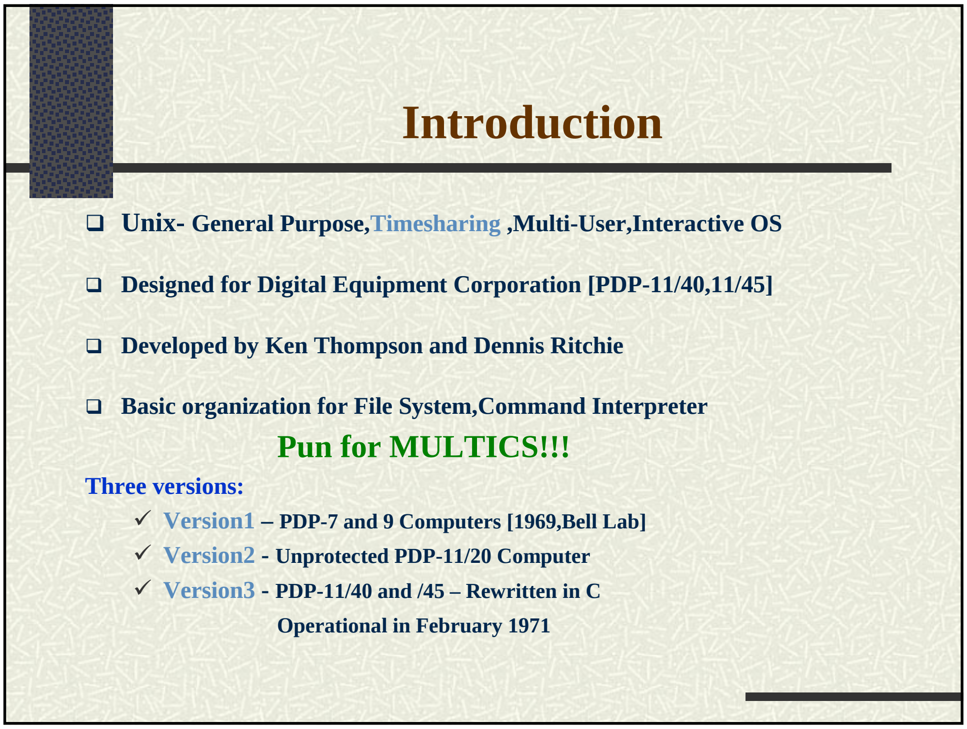


Outline

- ✓ Introduction
- ✓ Design Principles,Functionality
- ✓ Programs Under Unix
- ✓ Hardware/Software Environment
- ✓ Layers in Unix
- ✓ Unix OS- File System
  - - File/Directories/Links/Referencing
  - - Inodes/I lists
- ✓ I/O Calls
- ✓ Logical to Physical Mapping



- ✓ Protection
- ✓ Filters
- ✓ The UNIX Shell Implementation
- ✓ CODE
- ✓ Summary
- ✓ Conclusion
- ✓ References



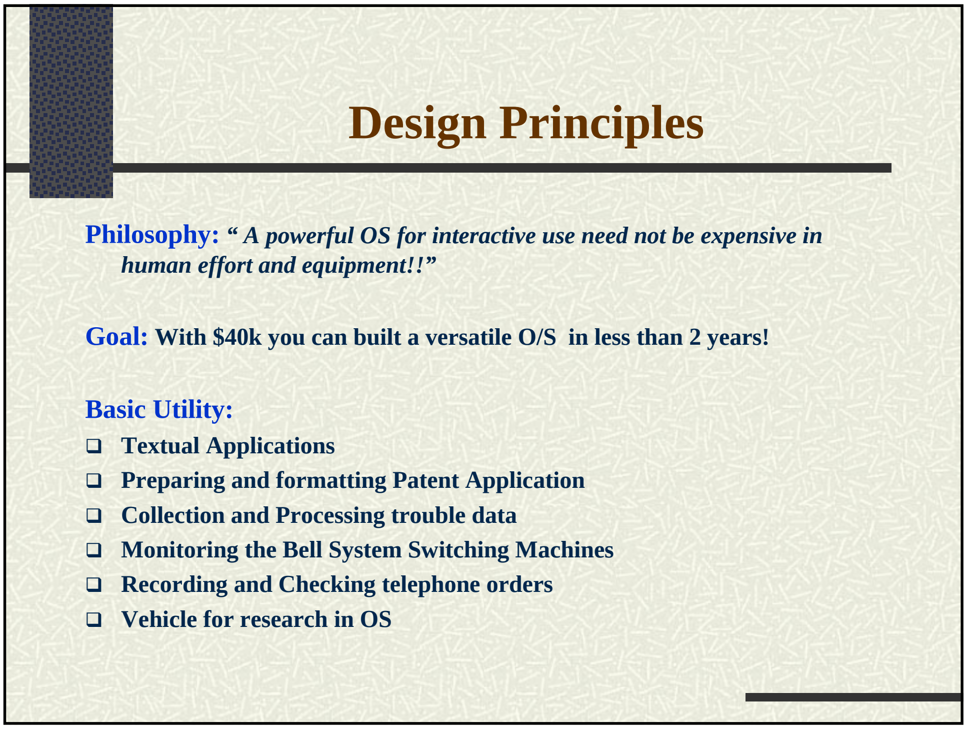
Introduction

- ❑ Unix- General Purpose,Timesharing ,Multi-User,Interactive OS
- ❑ Designed for Digital Equipment Corporation [PDP-11/40,11/45]
- ❑ Developed by Ken Thompson and Dennis Ritchie
- ❑ Basic organization for File System,Command Interpreter

Pun for MULTICS!!!

Three versions:

- ✓ Version1 – PDP-7 and 9 Computers [1969,Bell Lab]
- ✓ Version2 - Unprotected PDP-11/20 Computer
- ✓ Version3 - PDP-11/40 and /45 – Rewritten in C
  Operational in February 1971



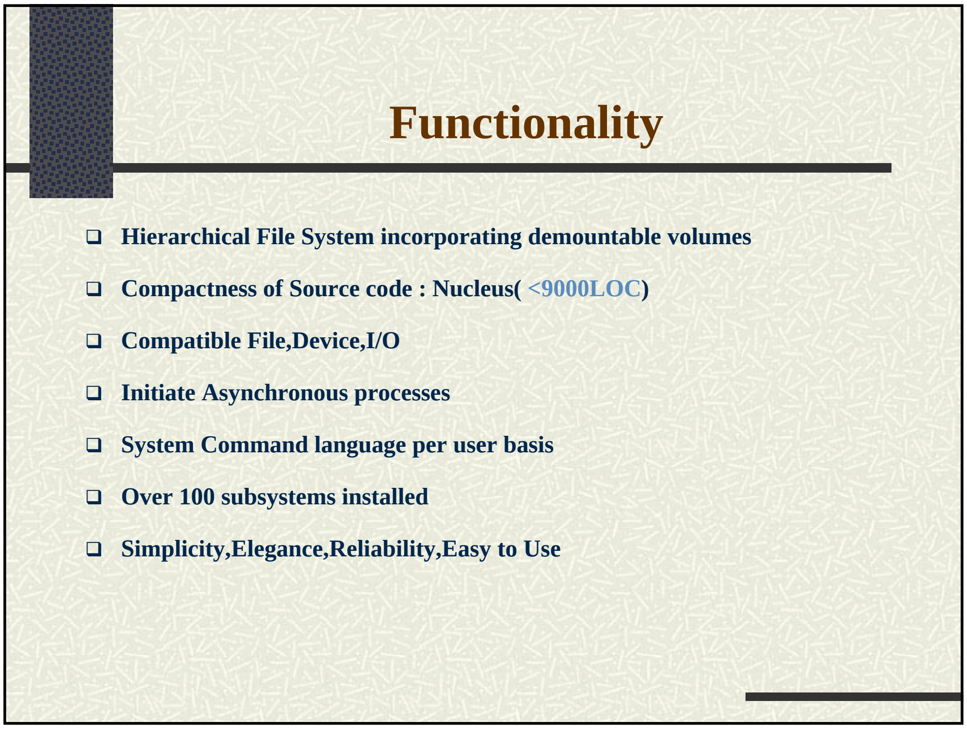
Design Principles

Philosophy: ***" A powerful OS for interactive use need not be expensive in human effort and equipment!!"***

Goal: **With $40k you can built a versatile O/S in less than 2 years!**

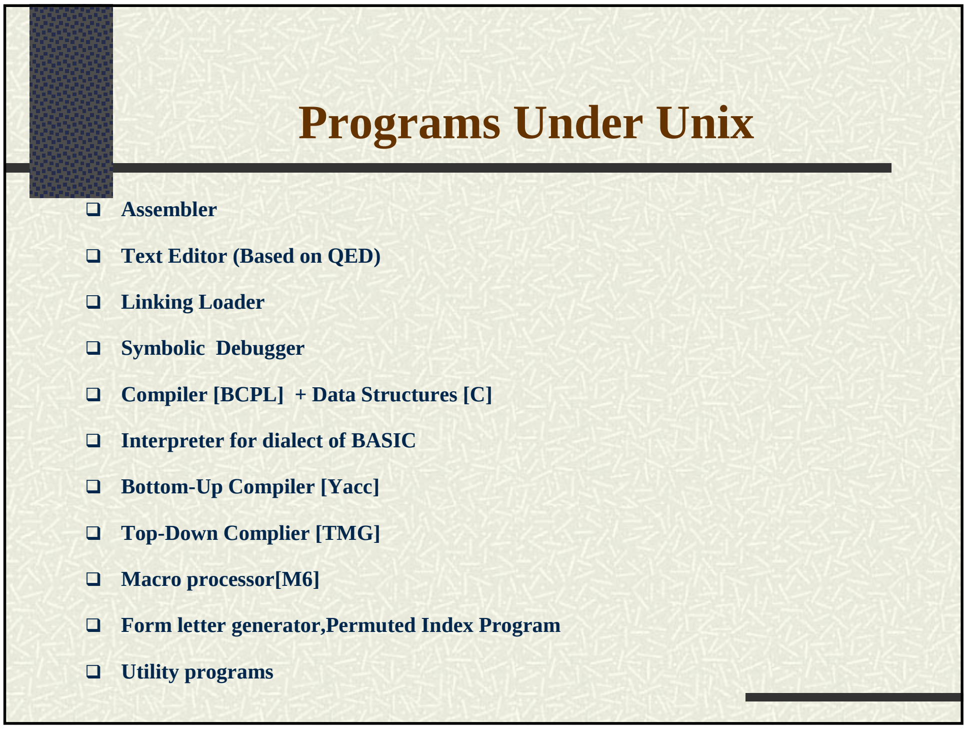
Basic Utility:

- ❑ **Textual Applications**
- ❑ **Preparing and formatting Patent Application**
- ❑ **Collection and Processing trouble data**
- ❑ **Monitoring the Bell System Switching Machines**
- ❑ **Recording and Checking telephone orders**
- ❑ **Vehicle for research in OS**



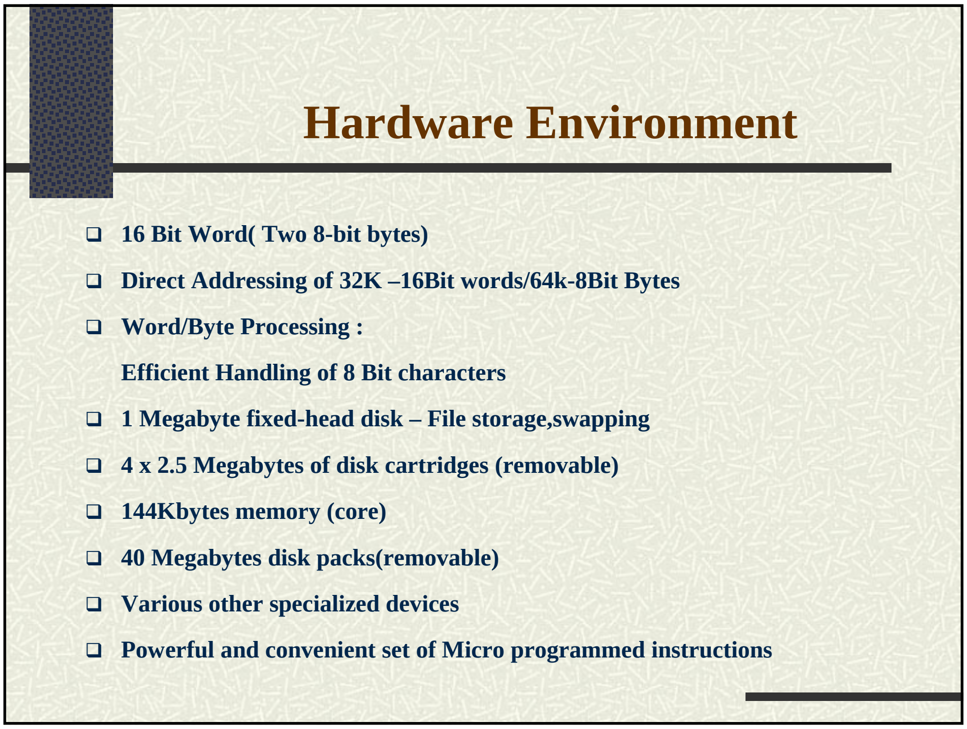
Functionality

- ❑ **Hierarchical File System incorporating demountable volumes**
- ❑ **Compactness of Source code : Nucleus(<9000LOC)**
- ❑ **Compatible File,Device,I/O**
- ❑ **Initiate Asynchronous processes**
- ❑ **System Command language per user basis**
- ❑ **Over 100 subsystems installed**
- ❑ **Simplicity,Elegance,Reliability,Easy to Use**



Programs Under Unix

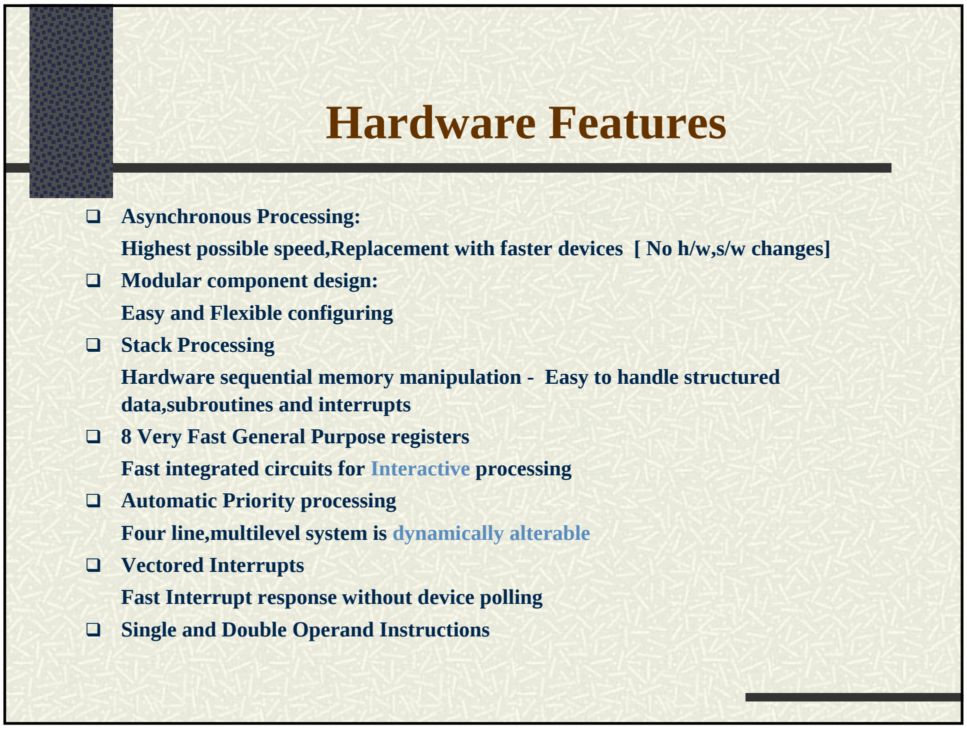
- Assembler
- Text Editor (Based on QED)
- Linking Loader
- Symbolic Debugger
- Compiler [BCPL] + Data Structures [C]
- Interpreter for dialect of BASIC
- Bottom-Up Compiler [Yacc]
- Top-Down Complier [TMG]
- Macro processor[M6]
- Form letter generator,Permuted Index Program
- Utility programs



Hardware Environment

- 16 Bit Word(Two 8-bit bytes)
- Direct Addressing of 32K –16Bit words/64k-8Bit Bytes
- Word/Byte Processing :

  Efficient Handling of 8 Bit characters
- 1 Megabyte fixed-head disk – File storage,swapping
- 4 x 2.5 Megabytes of disk cartridges (removable)
- 144Kbytes memory (core)
- 40 Megabytes disk packs(removable)
- Various other specialized devices
- Powerful and convenient set of Micro programmed instructions



Hardware Features

- ❑ Asynchronous Processing:

  Highest possible speed,Replacement with faster devices [No h/w,s/w changes]
- ❑ Modular component design:

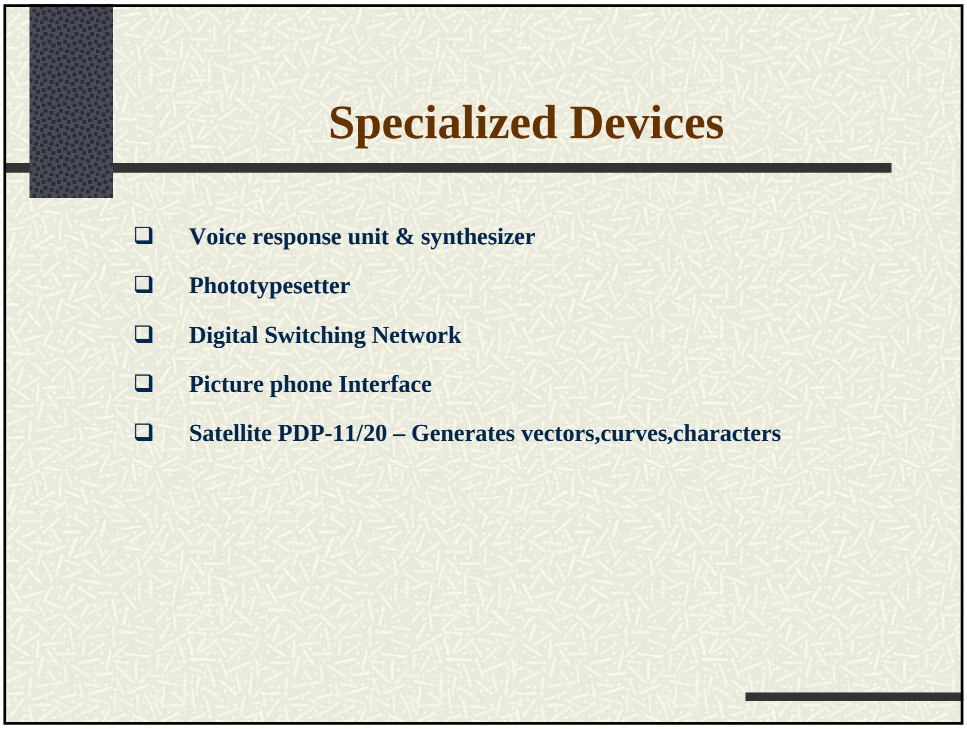
  Easy and Flexible configuring
- ❑ Stack Processing

  Hardware sequential memory manipulation - Easy to handle structured data,subroutines and interrupts
- ❑ 8 Very Fast General Purpose registers

  Fast integrated circuits for Interactive processing
- ❑ Automatic Priority processing

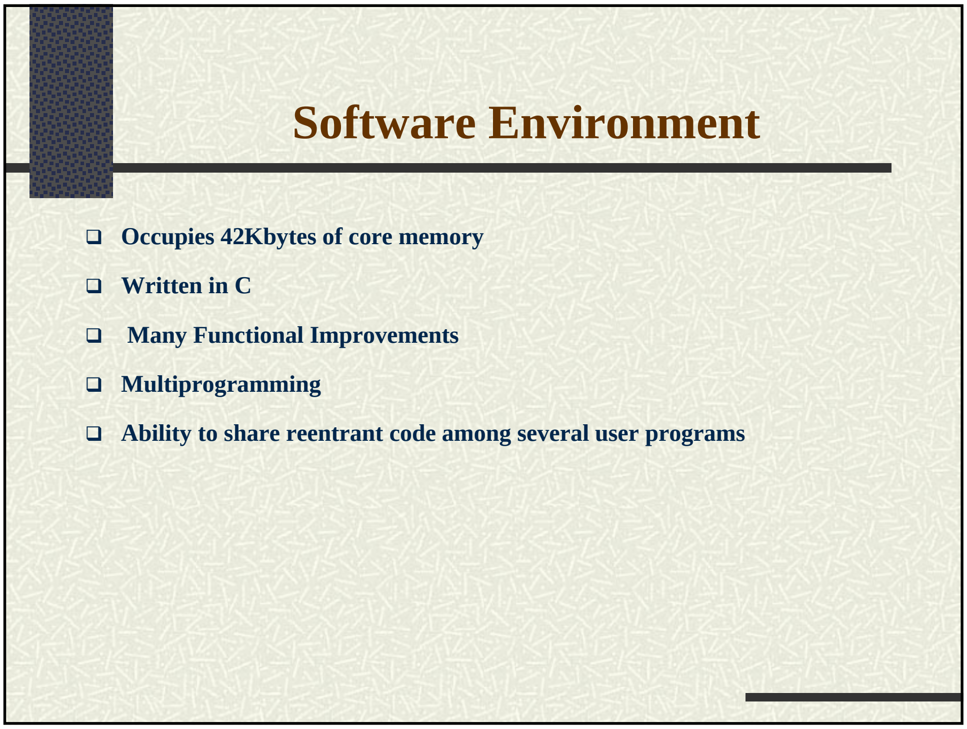
  Four line,multilevel system is dynamically alterable
- ❑ Vectored Interrupts

  Fast Interrupt response without device polling
- ❑ Single and Double Operand Instructions



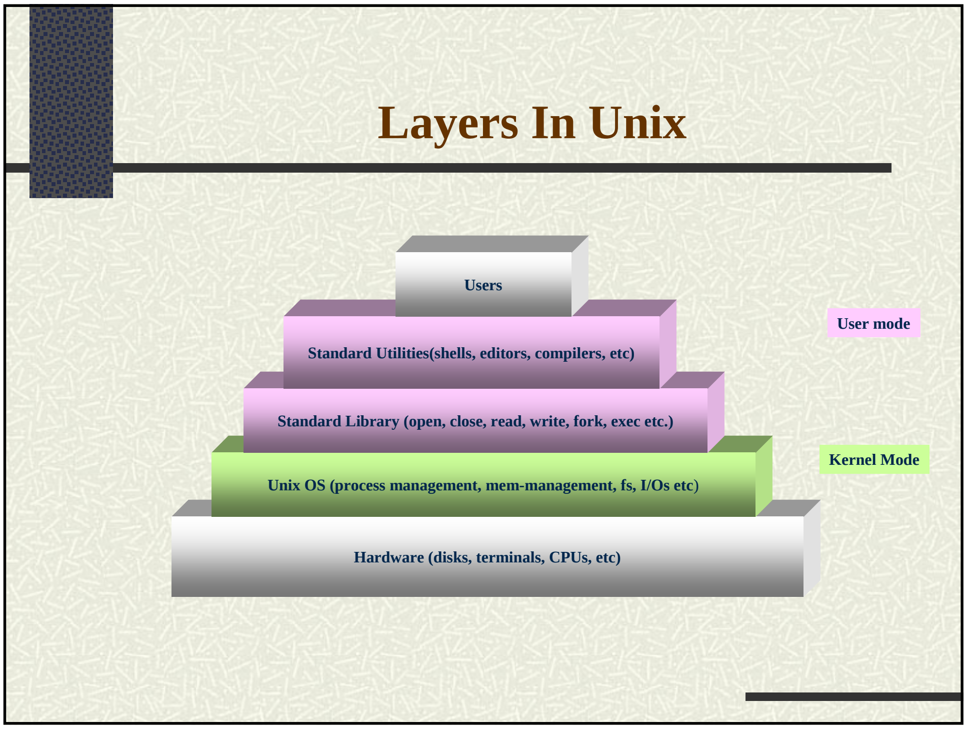
Specialized Devices

- ❑ Voice response unit & synthesizer
- ❑ Phototypesetter
- ❑ Digital Switching Network
- ❑ Picture phone Interface
- ❑ Satellite PDP-11/20 – Generates vectors,curves,characters



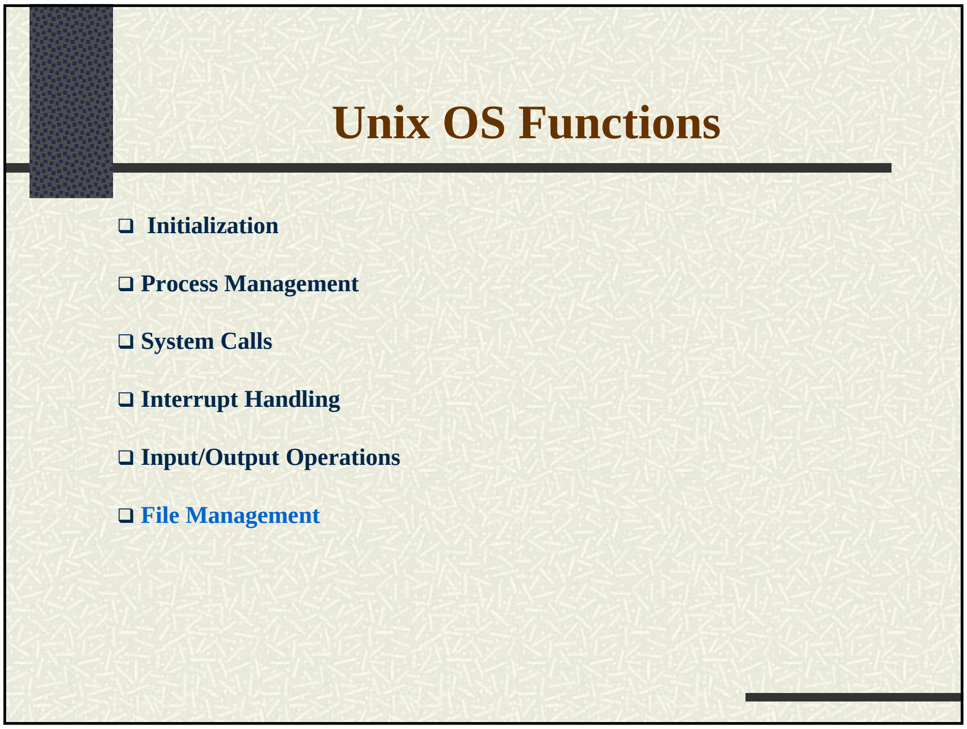
Software Environment

- Occupies 42Kbytes of core memory
- Written in C
- Many Functional Improvements
- Multiprogramming
- Ability to share reentrant code among several user programs



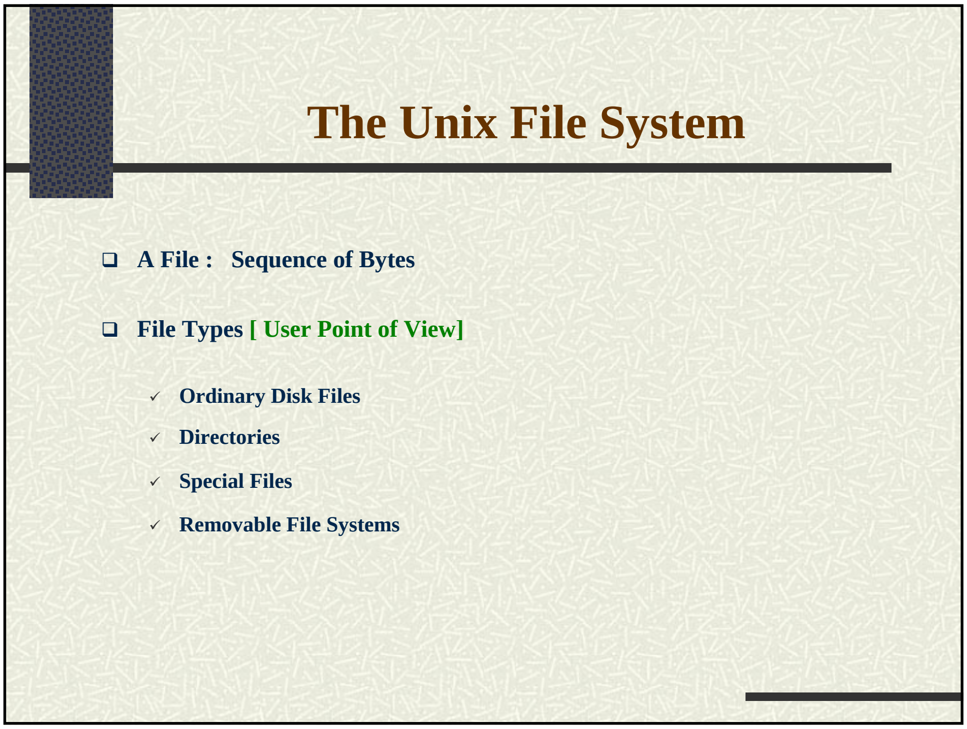
Layers In Unix

Users
Standard Utilities(shells, editors, compilers, etc)
Standard Library (open, close, read, write, fork, exec etc.)
Unix OS (process management, mem-management, fs, I/Os etc)
Hardware (disks, terminals, CPUs, etc)
User mode
Kernel Mode



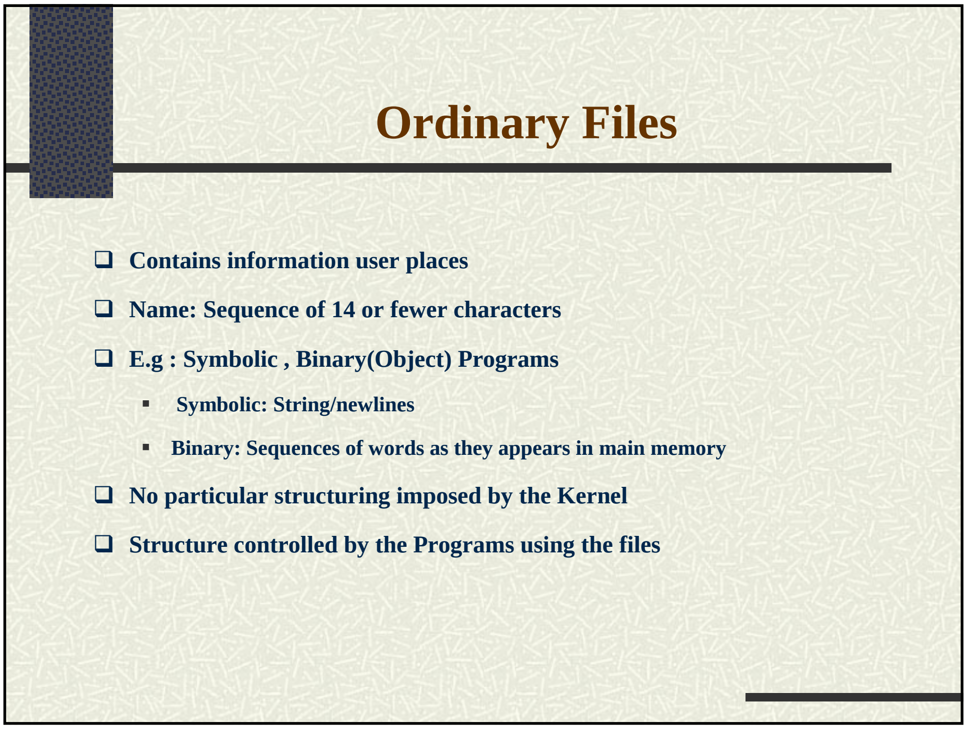
Unix OS Functions

- □ Initialization
- □ Process Management
- □ System Calls
- □ Interrupt Handling
- □ Input/Output Operations
- □ File Management



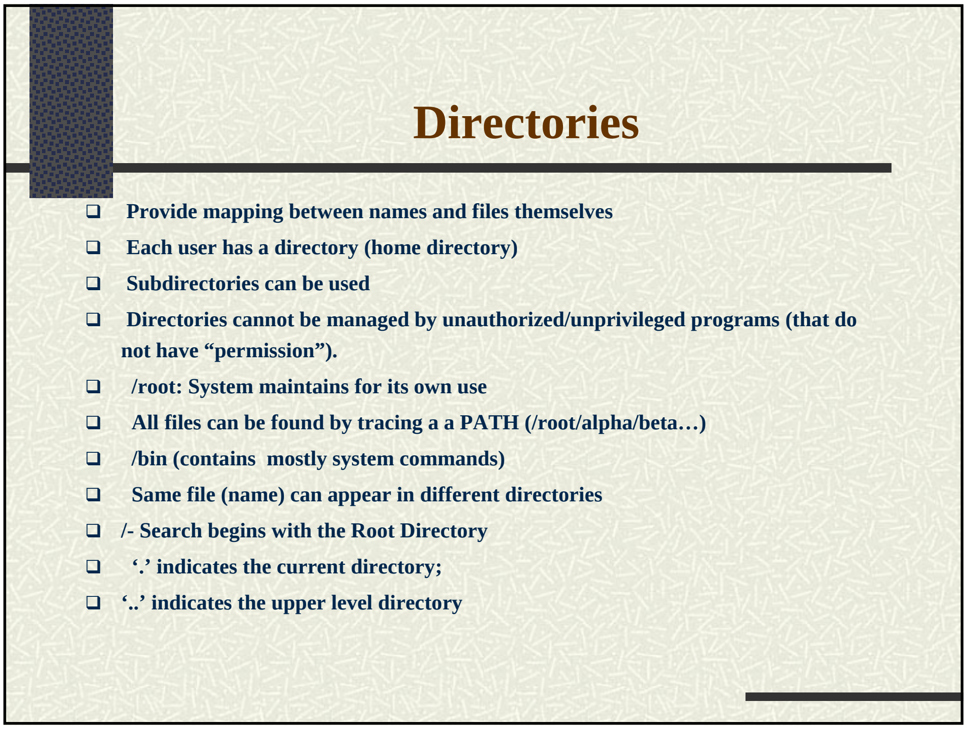
The Unix File System

- □ A File : Sequence of Bytes
- □ File Types [User Point of View]
  - ✓ Ordinary Disk Files
  - ✓ Directories
  - ✓ Special Files
  - ✓ Removable File Systems



Ordinary Files

- ❑ Contains information user places
- ❑ Name: Sequence of 14 or fewer characters
- ❑ E.g : Symbolic , Binary(Object) Programs
  - Symbolic: String/newlines
  - Binary: Sequences of words as they appears in main memory
- ❑ No particular structuring imposed by the Kernel
- ❑ Structure controlled by the Programs using the files



Directories

- ❑ Provide mapping between names and files themselves
- ❑ Each user has a directory (home directory)
- ❑ Subdirectories can be used
- ❑ Directories cannot be managed by unauthorized/unprivileged programs (that do not have “permission”).
- ❑ /root: System maintains for its own use
- ❑ All files can be found by tracing a a PATH (/root/alpha/beta…)
- ❑ /bin (contains mostly system commands)
- ❑ Same file (name) can appear in different directories
- ❑ /- Search begins with the Root Directory
- ❑ ‘.’ indicates the current directory;
- ❑ ‘..’ indicates the upper level directory

Names and Links

Absolute Path Names :

Start at Root of the file system

Relative Path Names:

Start at the current directory

Links: Multiple Names

- ❑ A directory entry for a file
- ❑ Same Non Directory File appears in several directories under different names
- ❑ All links to file have equal status
- ❑ File doesn’t exist within directory
- ❑ Entry contains File Name and Pointer

Symbolic Links[Soft]: Path name of another file

Hard Links: Don’t cross file-system boundaries

Special Files

- ❑ Most unique feature of UNIX
- ❑ Each I/O device is associated with a file
- ❑ Written into/ Read from as ordinary files
- ❑ Read/Write result in activation of assosciated device
- ❑ Directory “/dev” contains all special files in the system
- ❑ Link can be made link ordinary file

E.g: To punch a card then one has to write to /dev/ppt

Advantages: (I/O Devices treated as Files)

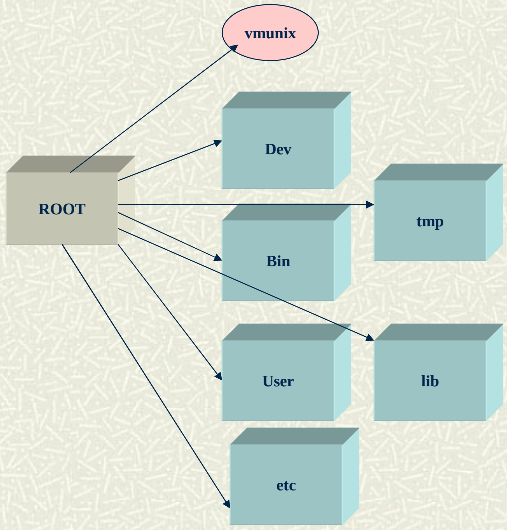
❑ Files and I/O Devices are similar in structure

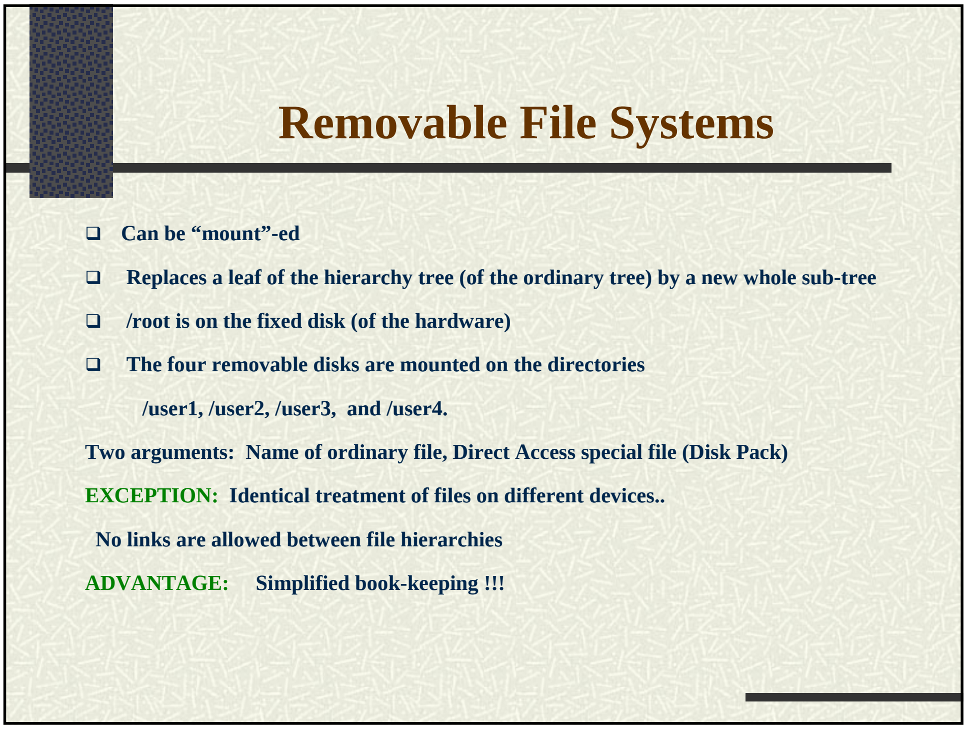
❑ File devices have the same naming convention,syntax

❑ Program can pass Device Name as parameter

❑ Similar protection options can be applied uniformly – Regular files

UNIX Directory Structure





Removable File Systems

- Can be “mount”-ed
- Replaces a leaf of the hierarchy tree (of the ordinary tree) by a new whole sub-tree
- /root is on the fixed disk (of the hardware)
- The four removable disks are mounted on the directories

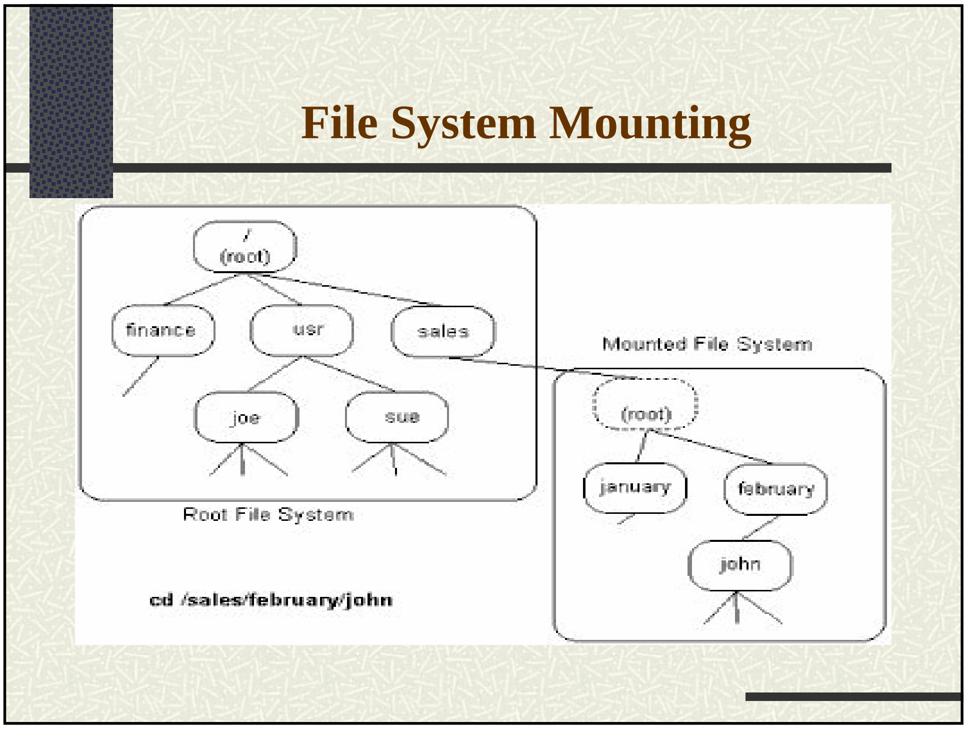
  /user1, /user2, /user3, and /user4.

Two arguments: Name of ordinary file, Direct Access special file (Disk Pack)

EXCEPTION: Identical treatment of files on different devices..

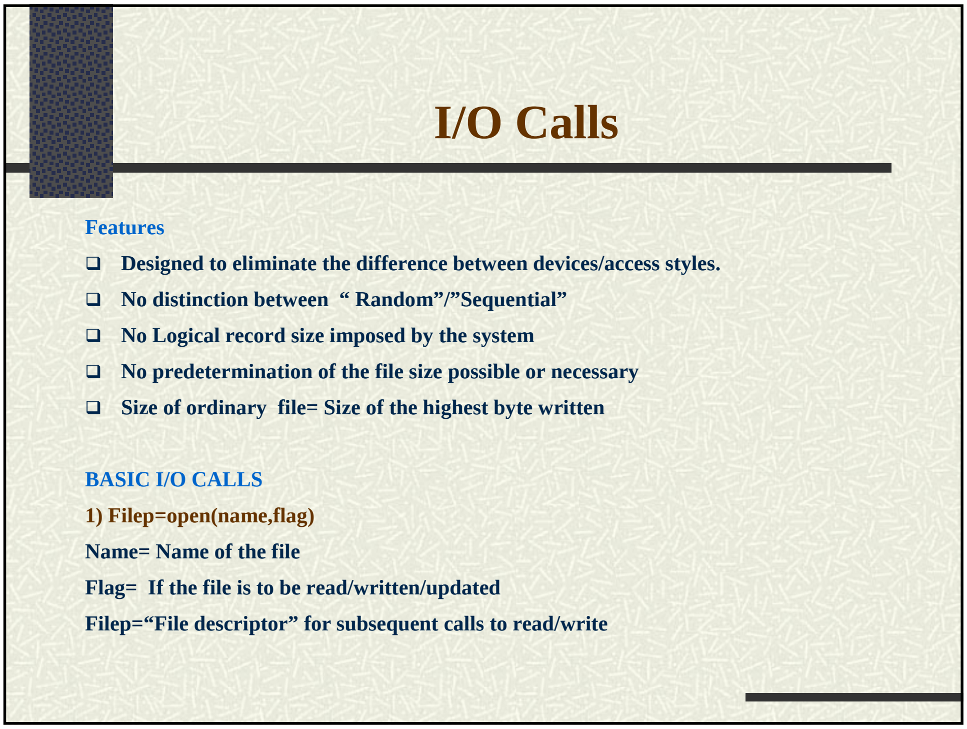
No links are allowed between file hierarchies

ADVANTAGE: Simplified book-keeping !!!



File System Mounting

cd /sales/february/john



I/O Calls

Features

- ❑ Designed to eliminate the difference between devices/access styles.
- ❑ No distinction between “ Random”/”Sequential”
- ❑ No Logical record size imposed by the system
- ❑ No predetermination of the file size possible or necessary
- ❑ Size of ordinary file= Size of the highest byte written

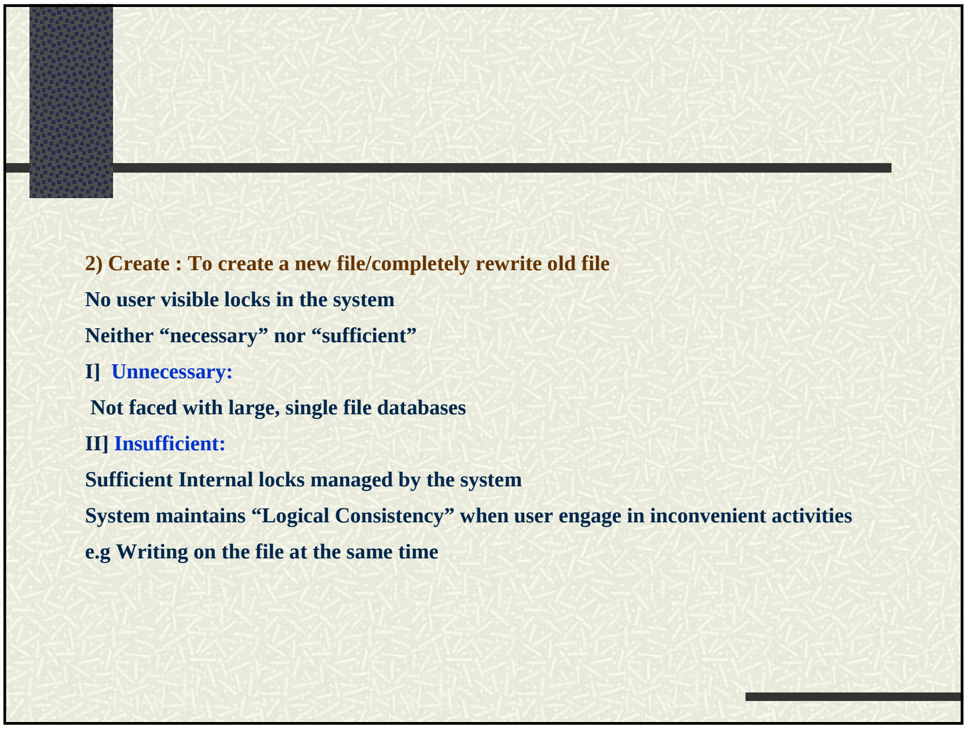
BASIC I/O CALLS

1) Filep=open(name,flag)

Name= Name of the file

Flag= If the file is to be read/written/updated

Filep=“File descriptor” for subsequent calls to read/write



2) Create : To create a new file/completely rewrite old file

No user visible locks in the system

Neither “necessary” nor “sufficient”

I] Unnecessary:

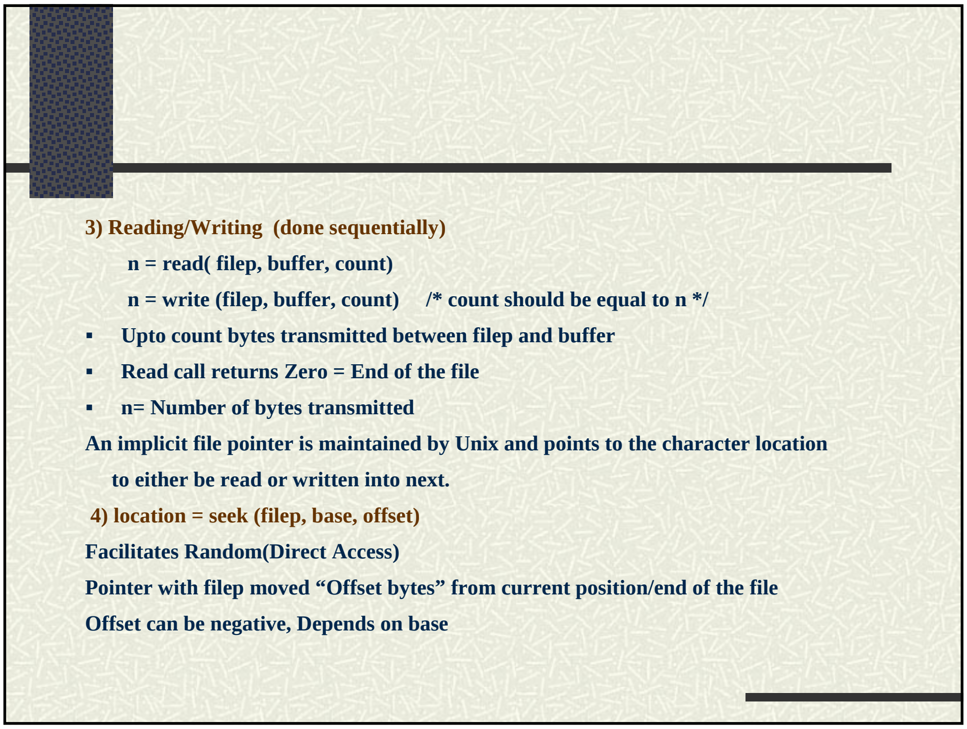
Not faced with large, single file databases

II] Insufficient:

Sufficient Internal locks managed by the system

System maintains “Logical Consistency” when user engage in inconvenient activities

e.g Writing on the file at the same time



3) Reading/Writing (done sequentially)

n = read(filep, buffer, count)

n = write (filep, buffer, count)   /* count should be equal to n */

- Upto count bytes transmitted between filep and buffer
- Read call returns Zero = End of the file
- n= Number of bytes transmitted

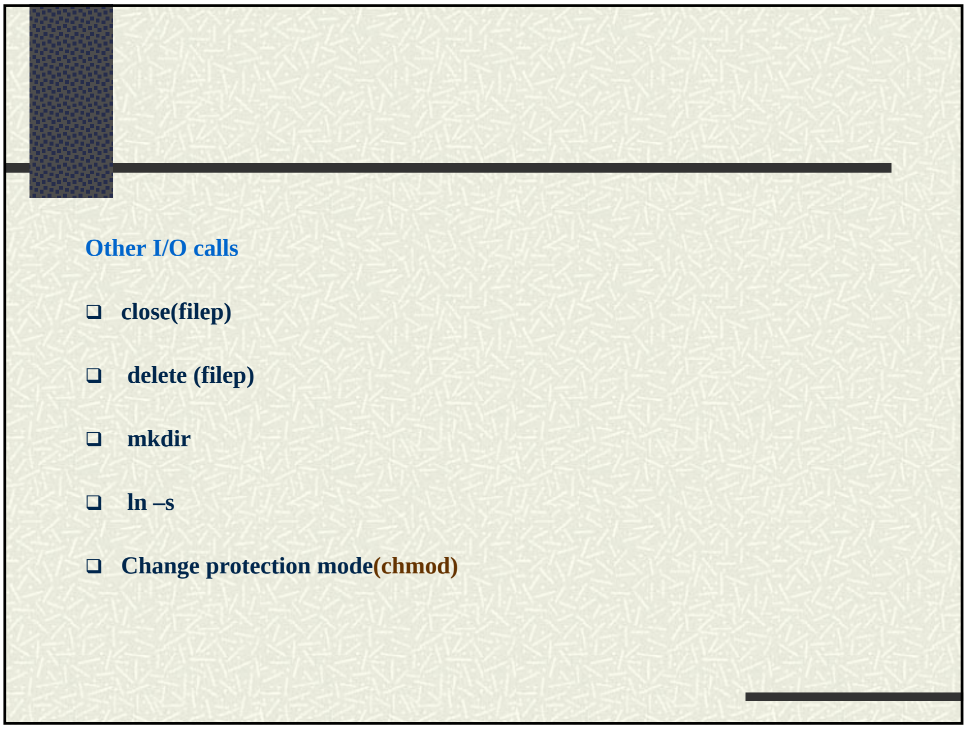
An implicit file pointer is maintained by Unix and points to the character location to either be read or written into next.

4) location = seek (filep, base, offset)

Facilitates Random(Direct Access)

Pointer with filep moved “Offset bytes” from current position/end of the file

Offset can be negative, Depends on base



Other I/O calls

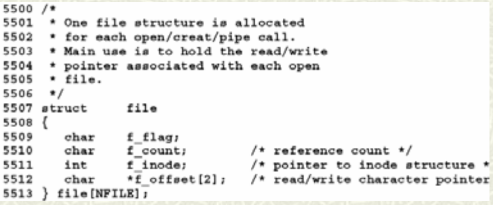
❑ close(filep)

❑ delete (filep)

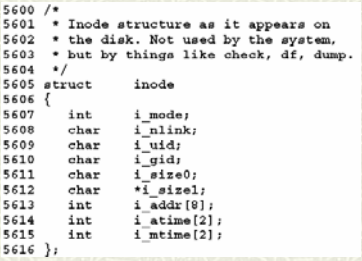
❑ mkdir

❑ ln –s

❑ Change protection mode(chmod)

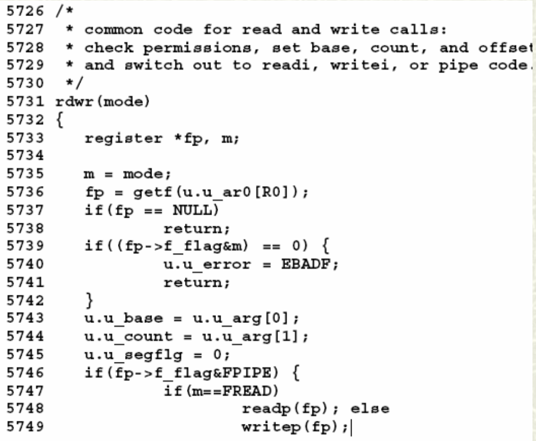


```
/*
 * One file structure is allocated
 * for each open/creat/pipe call.
 * Main use is to hold the read/write
 * pointer associated with each open
 * file.
 */
struct     file
{
    char    f_flag;
    char    f_count;        /* reference count */
    int     f_inode;        /* pointer to inode structure *
    char    *f_offset[2];   /* read/write character pointer
} file[NFILE];
```

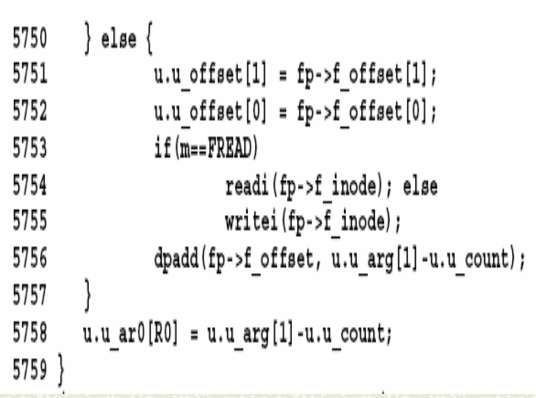


```
/*
 * Inode structure as it appears on
 * the disk. Not used by the system,
 * but by things like check, df, dump.
 */
struct     inode
{
    int     i_mode;
    char    i_nlink;
    char    i_uid;
    char    i_gid;
    char    i_size0;
    char    *i_size1;
    int     i_addr[8];
    int     i_atime[2];
    int     i_mtime[2];
};
```

Code- Read/Write



```
/*
 * common code for read and write calls:
 * check permissions, set base, count, and offset
 * and switch out to readi, writei, or pipe code.
 */
rdwr(mode)
{
    register *fp, m;

    m = mode;
    fp = getf(u.u_ar0[R0]);
    if(fp == NULL)
            return;
    if((fp->f_flag&m) == 0) {
            u.u_error = EBADF;
            return;
    }
    u.u_base = u.u_arg[0];
    u.u_count = u.u_arg[1];
    u.u_segflg = 0;
    if(fp->f_flag&FPIPE) {
            if(m==FREAD)
                    readp(fp); else
                    writep(fp);
```

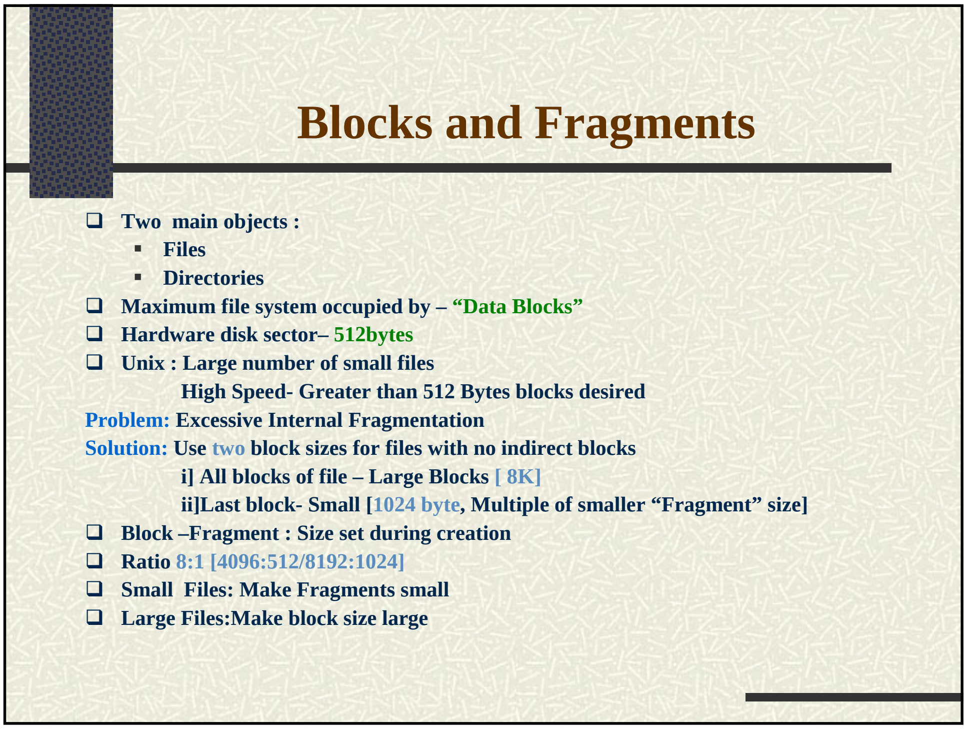


```
        } else {
                u.u_offset[1] = fp->f_offset[1];
                u.u_offset[0] = fp->f_offset[0];
                if(m==FREAD)
                        readi(fp->f_inode); else
                        writei(fp->f_inode);
                dpadd(fp->f_offset, u.u_arg[1]-u.u_count);
        }
        u.u_ar0[R0] = u.u_arg[1]-u.u_count;
}
```

Protection

- ❑ Each user is issued an "userid"
- ❑ At creation, any file is marked with the userid of the owner.
- ❑ 7 Bits are provided for protection
- ❑ Six bits designate who (user, group, others) has what access (w/r/x) on the file.
- ❑ Super user is the "reigning" user who is not restricted in any way
- ❑ Set-user-Id – Privileged programs using files inaccessible to others
- ❑ Avoid Intervention by the OS

e.g Accounting file shouldn't be read/written/changed by other programs



Blocks and Fragments

- Two main objects :
  - Files
  - Directories
- Maximum file system occupied by – “Data Blocks”
- Hardware disk sector– 512bytes
- Unix : Large number of small files

  High Speed- Greater than 512 Bytes blocks desired

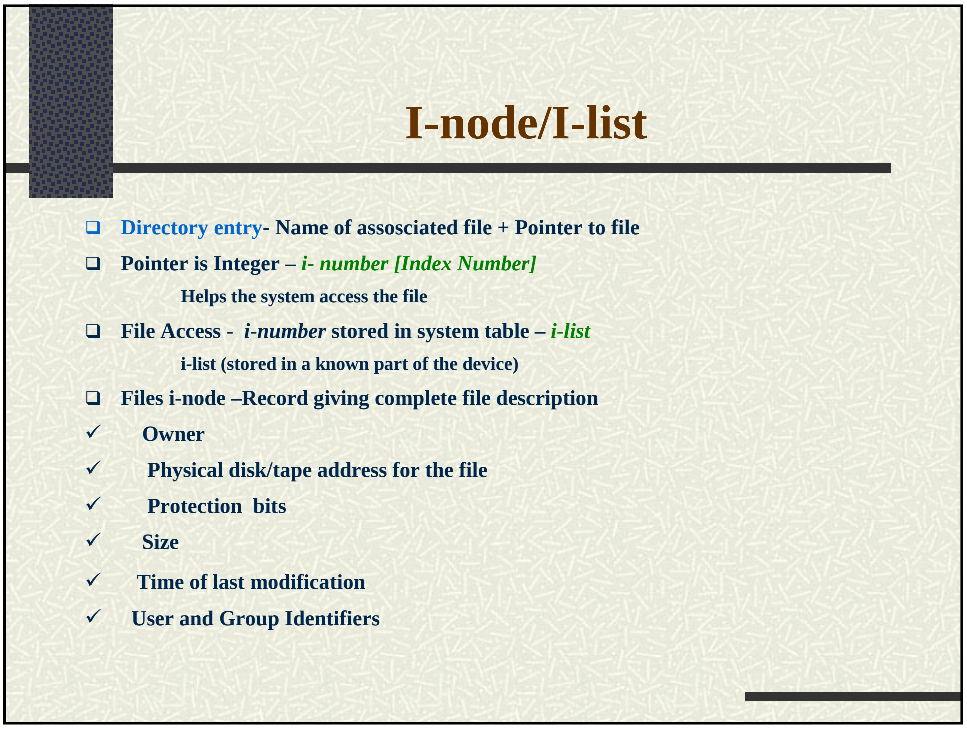
Problem: Excessive Internal Fragmentation

Solution: Use two block sizes for files with no indirect blocks

i] All blocks of file – Large Blocks [8K]

ii]Last block- Small [1024 byte, Multiple of smaller “Fragment” size]

- Block –Fragment : Size set during creation
- Ratio 8:1 [4096:512/8192:1024]
- Small Files: Make Fragments small
- Large Files:Make block size large



I-node/I-list

- Directory entry- Name of assosciated file + Pointer to file
- Pointer is Integer – *i- number [Index Number]*

  Helps the system access the file
- File Access - *i-number* stored in system table – *i-list*

  i-list (stored in a known part of the device)
- Files i-node –Record giving complete file description

✓ Owner

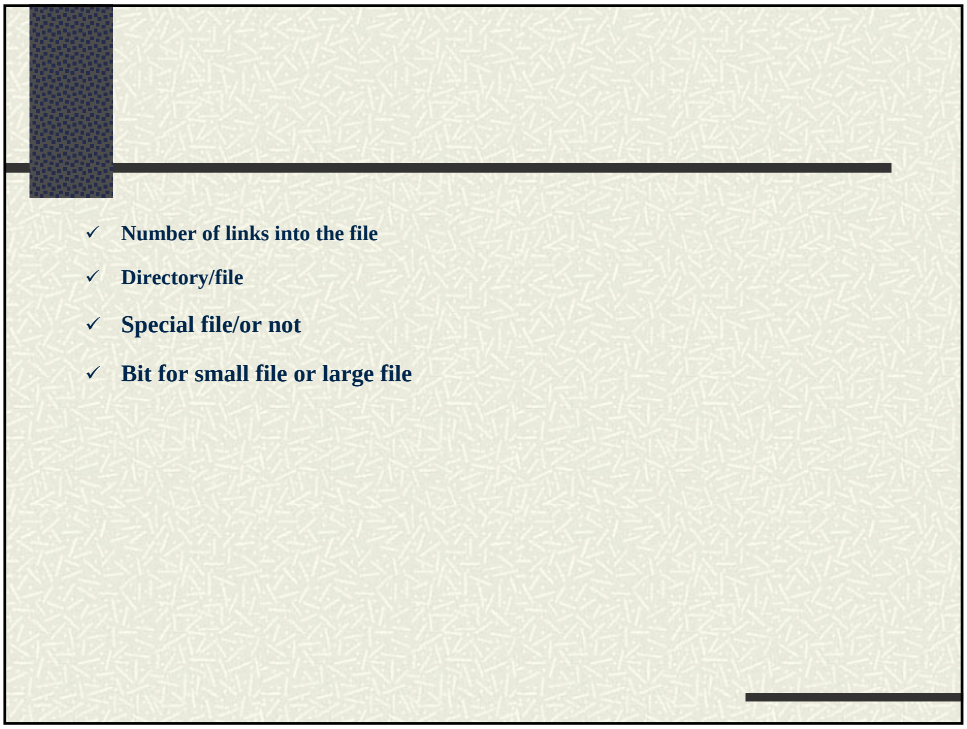
✓ Physical disk/tape address for the file

✓ Protection bits

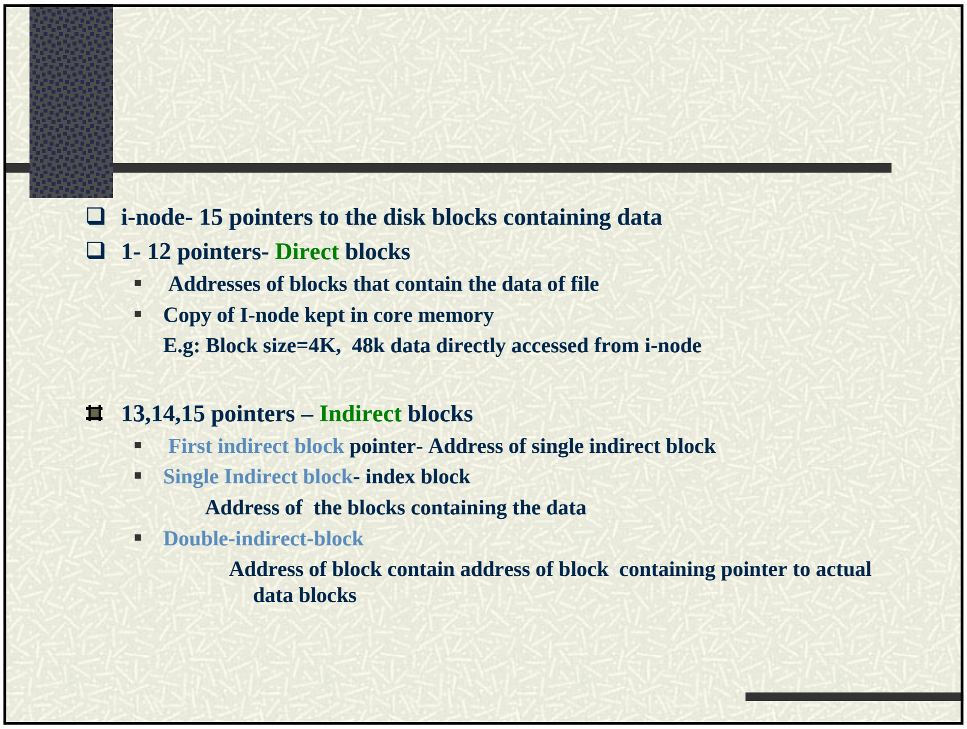
✓ Size

✓ Time of last modification

✓ User and Group Identifiers



- ✓ Number of links into the file
- ✓ Directory/file
- ✓ Special file/or not
- ✓ Bit for small file or large file



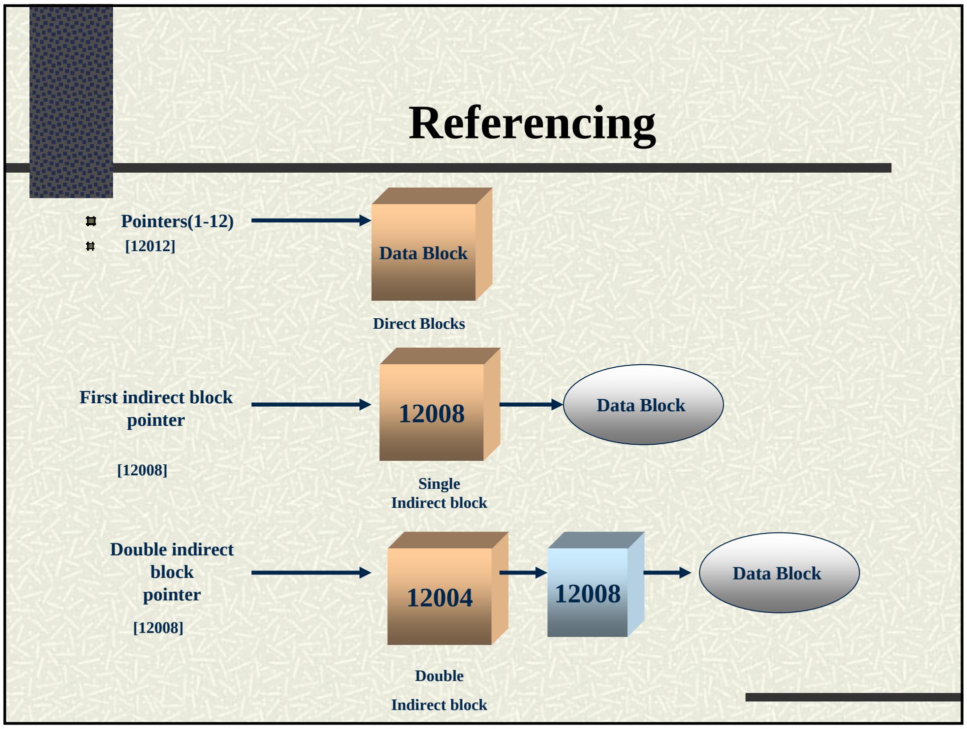
- ❑ i-node- 15 pointers to the disk blocks containing data
- ❑ 1- 12 pointers- Direct blocks
  - ▪ Addresses of blocks that contain the data of file
  - ▪ Copy of I-node kept in core memory

    E.g: Block size=4K, 48k data directly accessed from i-node

- ⌗ 13,14,15 pointers – Indirect blocks
  - ▪ First indirect block pointer- Address of single indirect block
  - ▪ Single Indirect block- index block

    Address of the blocks containing the data
  - ▪ Double-indirect-block

    Address of block contain address of block containing pointer to actual data blocks



Referencing

- Pointers(1-12)
- [12012]

Data Block

Direct Blocks

First indirect block pointer

[12008]

12008

Data Block

Single Indirect block

Double indirect block pointer

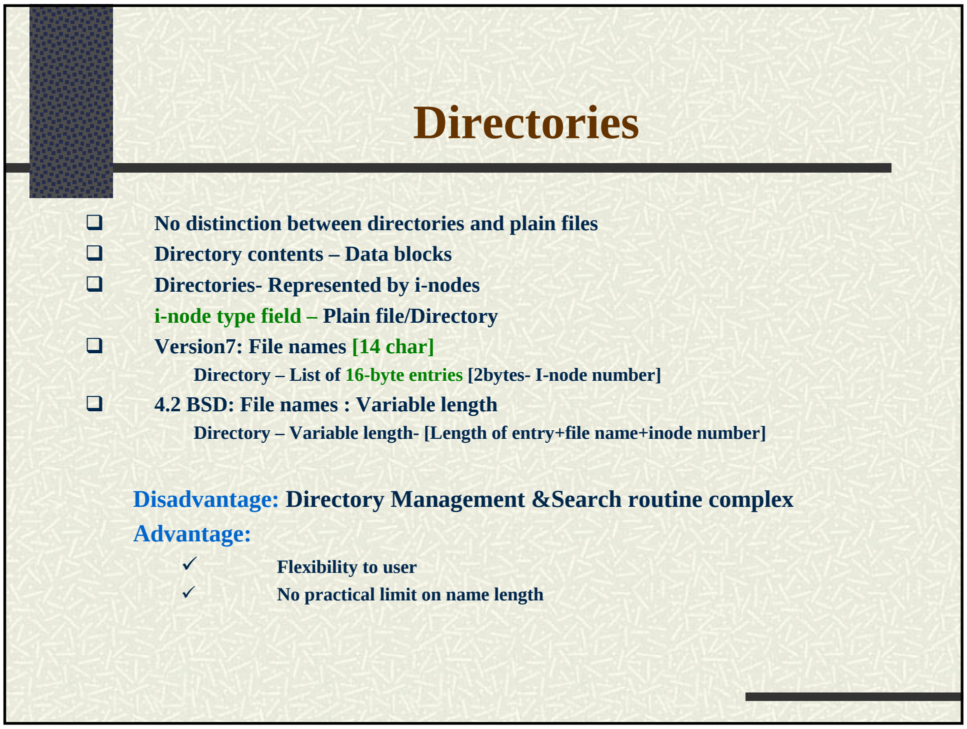
[12008]

12004

12008

Data Block

Double Indirect block



Directories

- ❑ No distinction between directories and plain files
- ❑ Directory contents – Data blocks
- ❑ Directories- Represented by i-nodes

  i-node type field – Plain file/Directory
- ❑ Version7: File names [14 char]

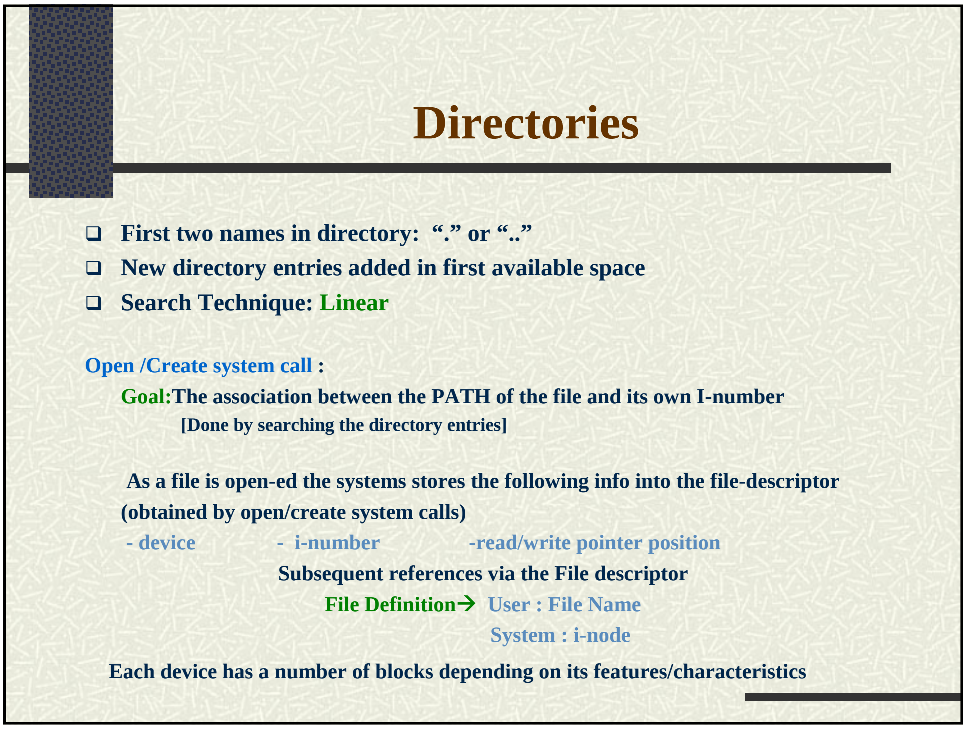
  Directory – List of 16-byte entries [2bytes- I-node number]
- ❑ 4.2 BSD: File names : Variable length

  Directory – Variable length- [Length of entry+file name+inode number]

Disadvantage: Directory Management &Search routine complex

Advantage:

- ✓ Flexibility to user
- ✓ No practical limit on name length



Directories

- ❑ First two names in directory: “.” or “..”
- ❑ New directory entries added in first available space
- ❑ Search Technique: Linear

Open /Create system call :

Goal:The association between the PATH of the file and its own I-number

[Done by searching the directory entries]

As a file is open-ed the systems stores the following info into the file-descriptor (obtained by open/create system calls)

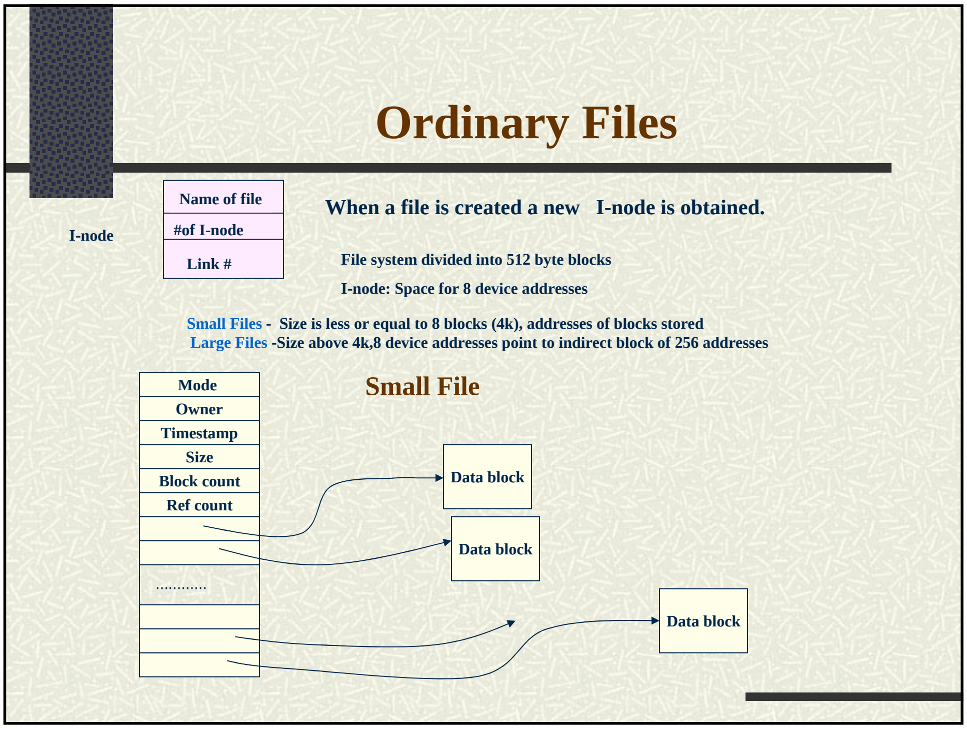
- device - i-number -read/write pointer position

Subsequent references via the File descriptor

File Definition→ User : File Name

System : i-node

Each device has a number of blocks depending on its features/characteristics



Ordinary Files

I-node

Name of file
#of I-node
Link #

When a file is created a new I-node is obtained.

File system divided into 512 byte blocks

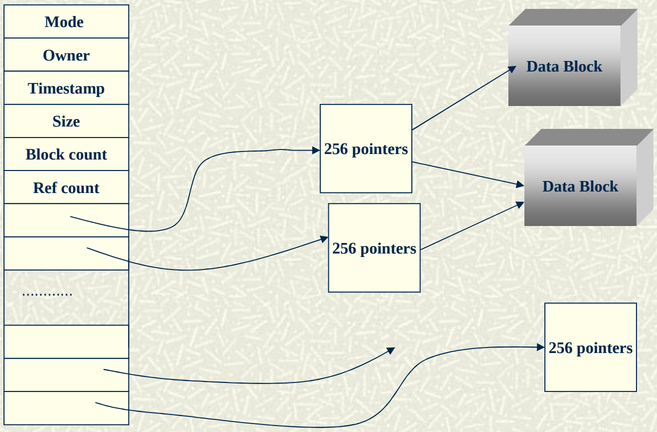
I-node: Space for 8 device addresses

Small Files - Size is less or equal to 8 blocks (4k), addresses of blocks stored

Large Files -Size above 4k,8 device addresses point to indirect block of 256 addresses

Small File

Large Files



Mode

Owner

Timestamp

Size

Block count

Ref count

…………

256 pointers

256 pointers

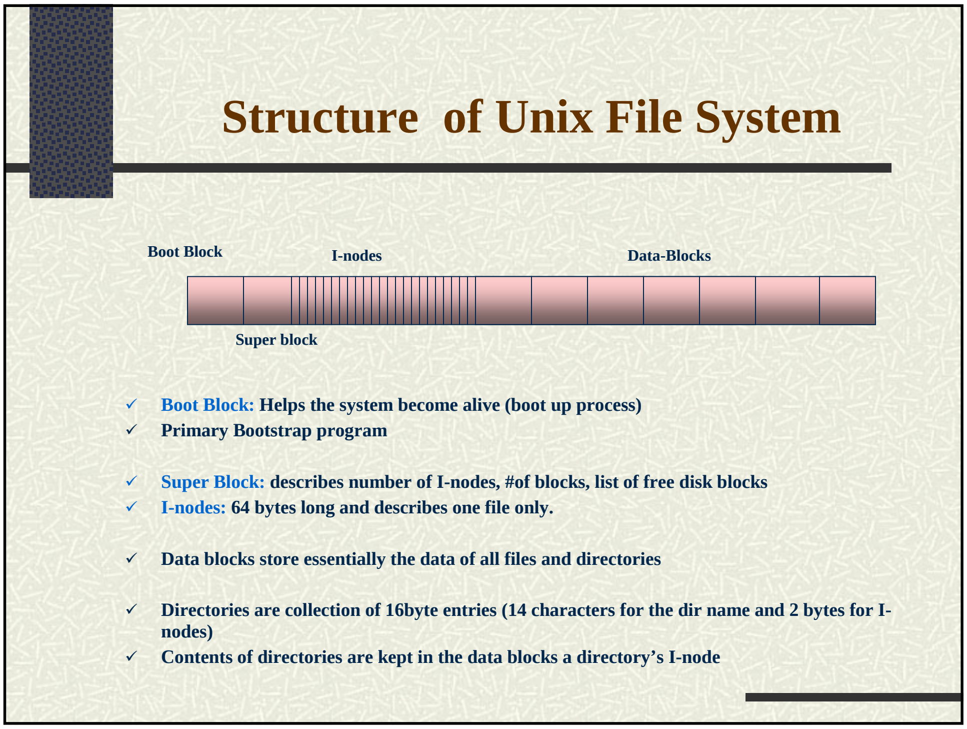
256 pointers

Data Block

Data Block

Link Operations

- ❑ **Adding Link:**
  - ❑ Create directory entry with new name
  - ❑ Copy the I-number from original file entry
  - ❑ Increment the link count filed of I-node
- ❑ **Deleting Link:**
  - ■ Decrement the link count of I-node
  - ■ Erase the directory entry
  - ■ If link count==0
    - ▪ Free Disk Blocks
    - ▪ Deallocate I-node



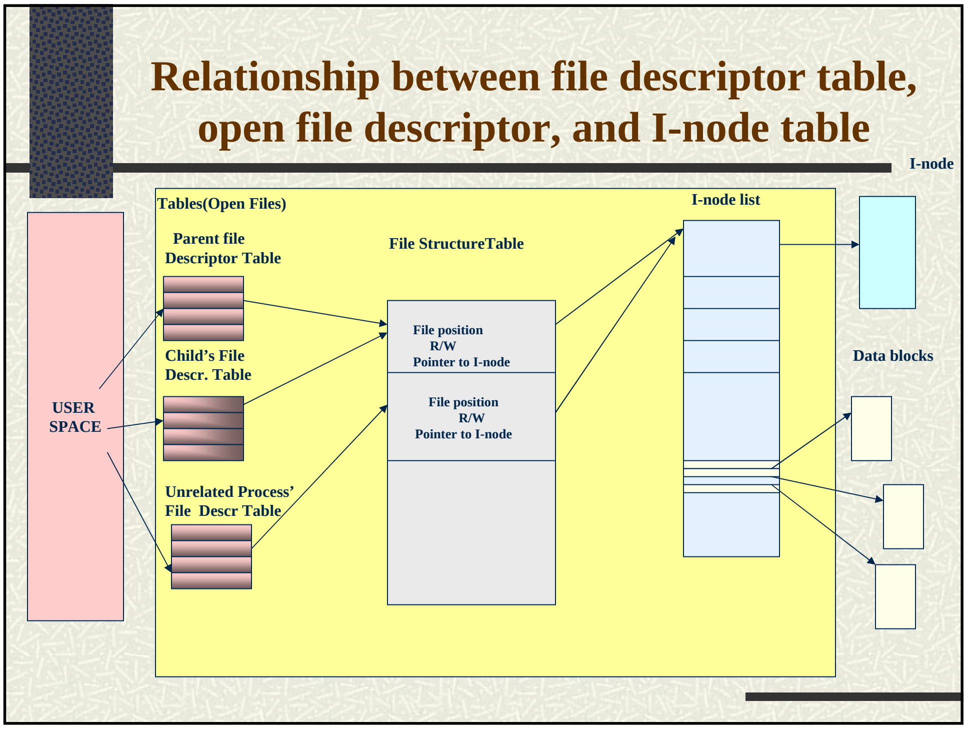
Structure of Unix File System

- ✓ Boot Block: Helps the system become alive (boot up process)
- ✓ Primary Bootstrap program

- ✓ Super Block: describes number of I-nodes, #of blocks, list of free disk blocks
- ✓ I-nodes: 64 bytes long and describes one file only.

- ✓ Data blocks store essentially the data of all files and directories

- ✓ Directories are collection of 16byte entries (14 characters for the dir name and 2 bytes for I-nodes)
- ✓ Contents of directories are kept in the data blocks a directory's I-node



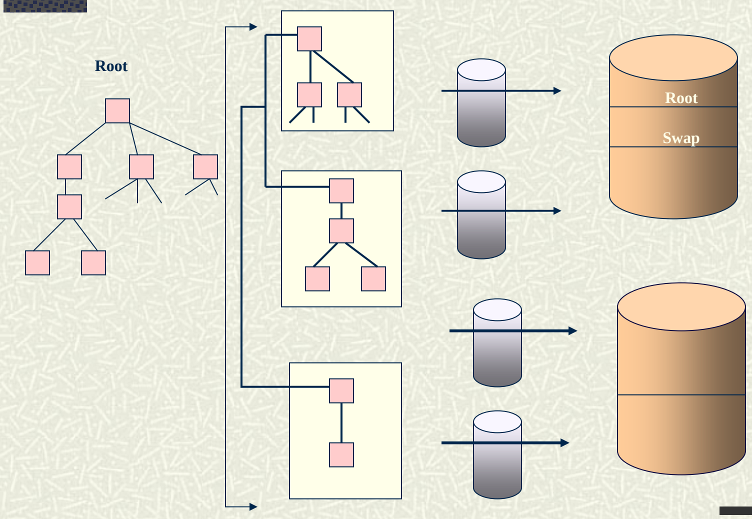
Relationship between file descriptor table, open file descriptor, and I-node table

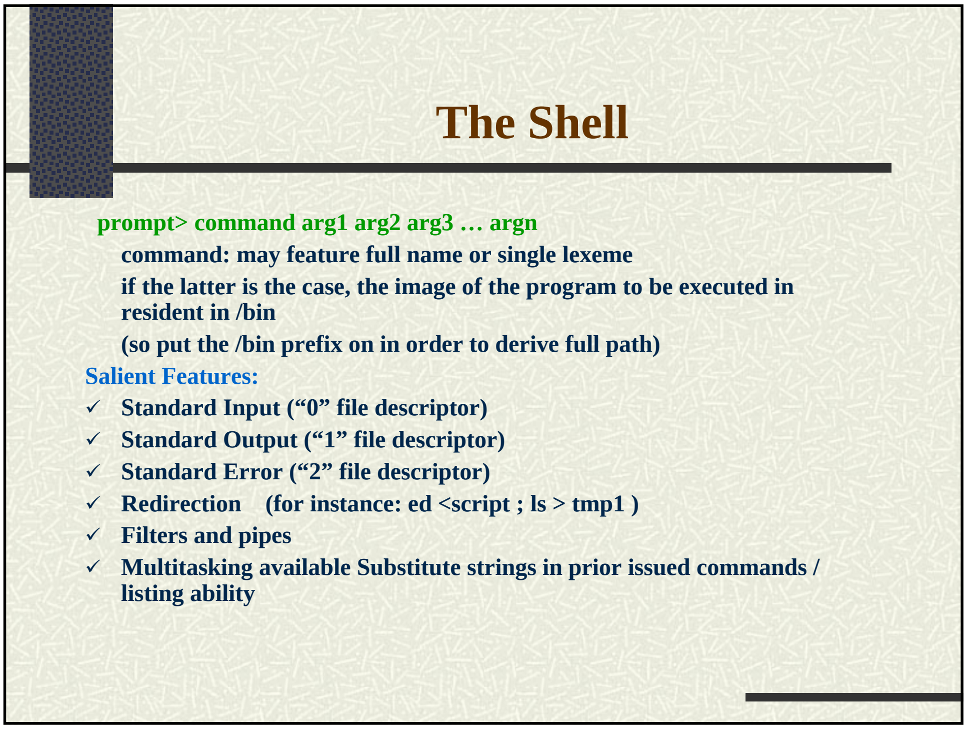
Physical Into Logical

Advantages

- Different file system –Uses/Swap Area
- Reliability
- Efficiency - Block and Fragment sizes
- Prevent a program from using all available space
- Disk Backups done from partition- Easy Search

Mapping of Logical File System to Physical devices





The Shell

prompt> command arg1 arg2 arg3 … argn

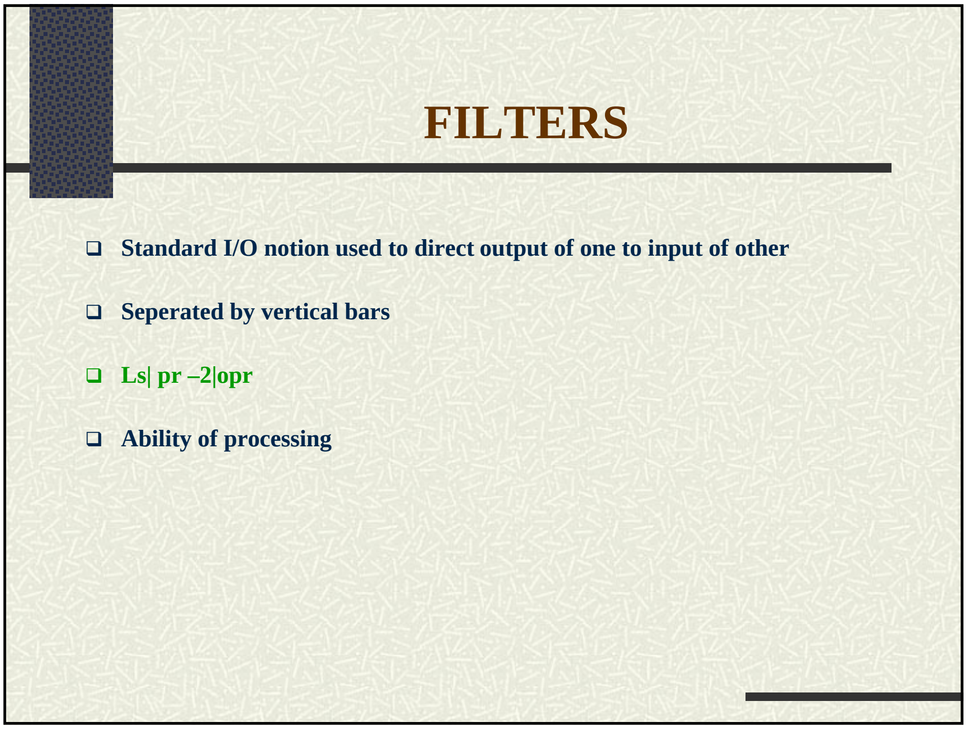
command: may feature full name or single lexeme

if the latter is the case, the image of the program to be executed in resident in /bin

(so put the /bin prefix on in order to derive full path)

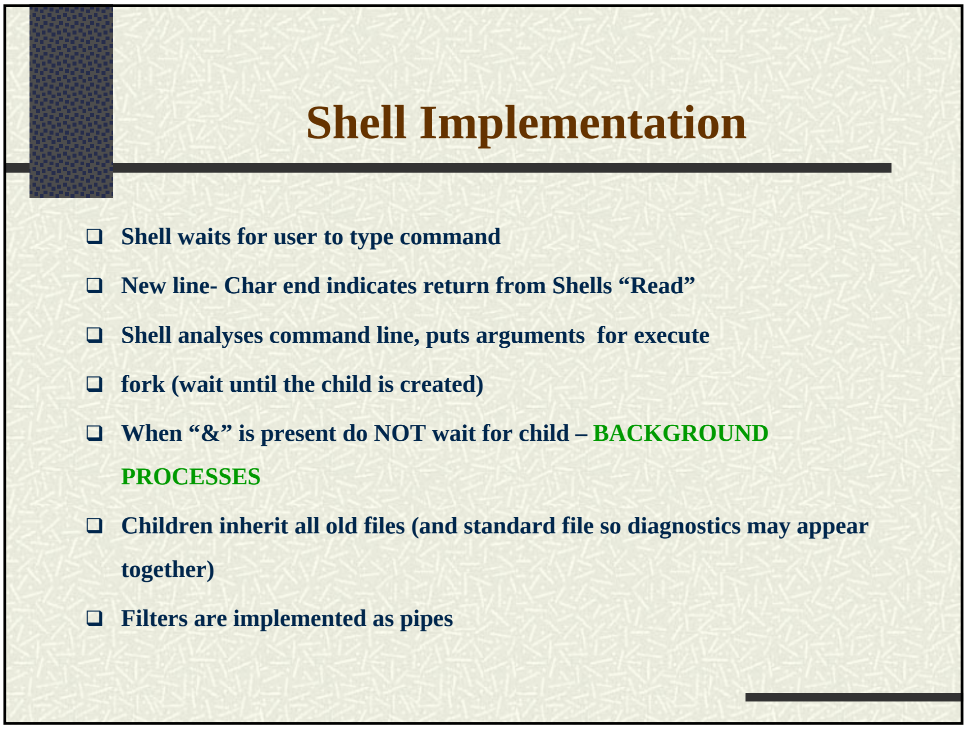
Salient Features:

- ✓ Standard Input (“0” file descriptor)
- ✓ Standard Output (“1” file descriptor)
- ✓ Standard Error (“2” file descriptor)
- ✓ Redirection (for instance: ed <script ; ls > tmp1)
- ✓ Filters and pipes
- ✓ Multitasking available Substitute strings in prior issued commands / listing ability



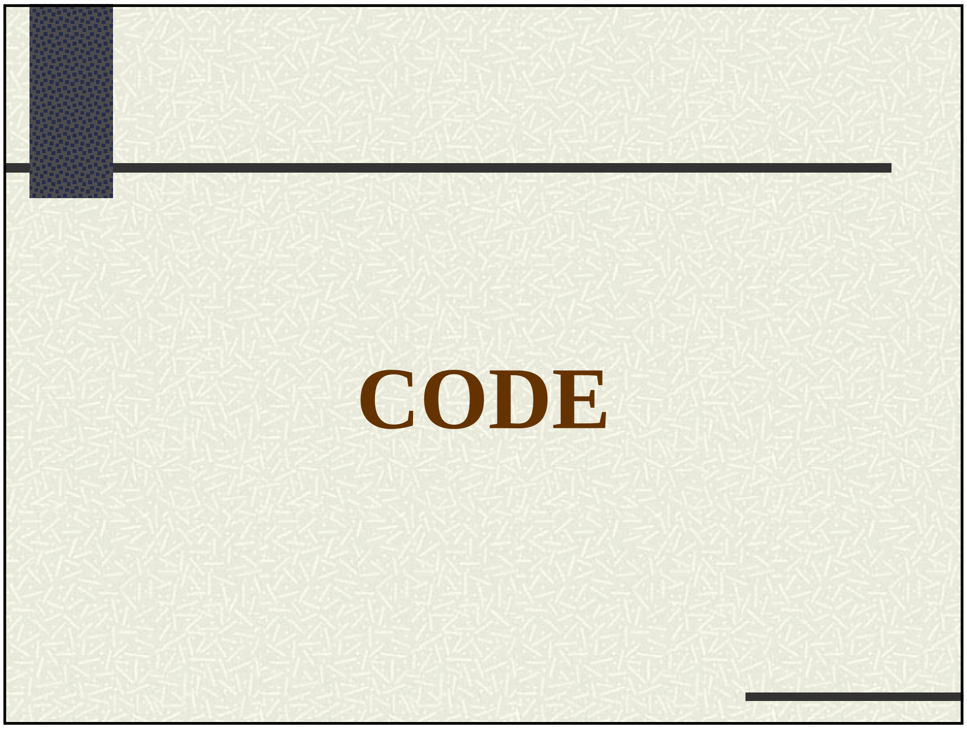
FILTERS

- ❑ Standard I/O notion used to direct output of one to input of other
- ❑ Seperated by vertical bars
- ❑ Ls| pr –2|opr
- ❑ Ability of processing



Shell Implementation

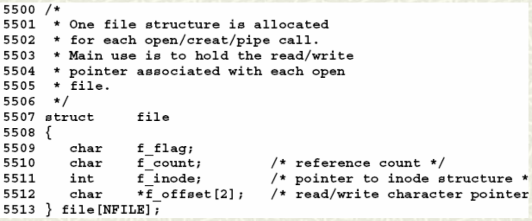
- Shell waits for user to type command
- New line- Char end indicates return from Shells “Read”
- Shell analyses command line, puts arguments for execute
- fork (wait until the child is created)
- When “&” is present do NOT wait for child – BACKGROUND PROCESSES
- Children inherit all old files (and standard file so diagnostics may appear together)
- Filters are implemented as pipes



CODE

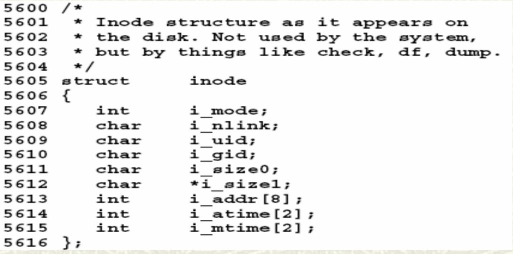
FILE & INODE Structures

FILE STRUCTURE



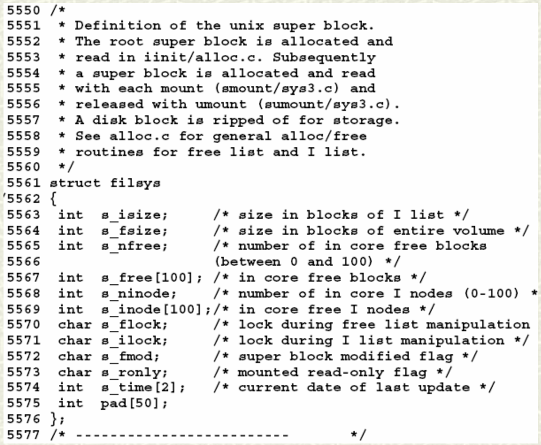
```
/*
 * One file structure is allocated
 * for each open/creat/pipe call.
 * Main use is to hold the read/write
 * pointer associated with each open
 * file.
 */
struct     file
{
   char    f_flag;
   char    f_count;         /* reference count */
   int     f_inode;         /* pointer to inode structure *
   char    *f_offset[2];    /* read/write character pointer
} file[NFILE];
```

INODE STRUCTURE



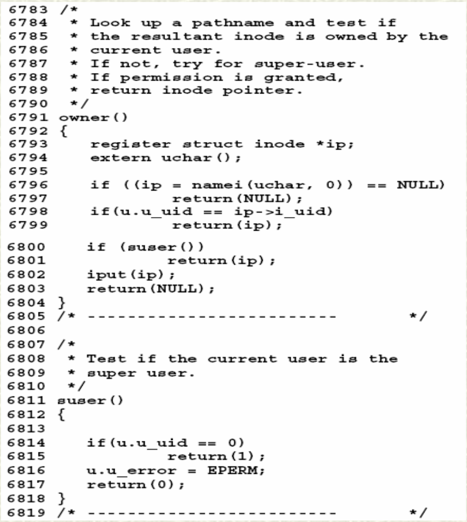
```
/*
 * Inode structure as it appears on
 * the disk. Not used by the system,
 * but by things like check, df, dump.
 */
struct      inode
{
    int      i_mode;
    char     i_nlink;
    char     i_uid;
    char     i_gid;
    char     i_size0;
    char     *i_size1;
    int      i_addr[8];
    int      i_atime[2];
    int      i_mtime[2];
};
```

Unix Super Block



```
/*
 * Definition of the unix super block.
 * The root super block is allocated and
 * read in iinit/alloc.c. Subsequently
 * a super block is allocated and read
 * with each mount (smount/sys3.c) and
 * released with umount (sumount/sys3.c).
 * A disk block is ripped of for storage.
 * See alloc.c for general alloc/free
 * routines for free list and I list.
 */
struct filsys
{
 int  s_isize;      /* size in blocks of I list */
 int  s_fsize;      /* size in blocks of entire volume */
 int  s_nfree;      /* number of in core free blocks
                    (between 0 and 100) */
 int  s_free[100]; /* in core free blocks */
 int  s_ninode;     /* number of in core I nodes (0-100) *
 int  s_inode[100];/* in core free I nodes */
 char s_flock;      /* lock during free list manipulation
 char s_ilock;      /* lock during I list manipulation */
 char s_fmod;       /* super block modified flag */
 char s_ronly;      /* mounted read-only flag */
 int  s_time[2];    /* current date of last update */
 int  pad[50];
};
/* ------------------------        */
```

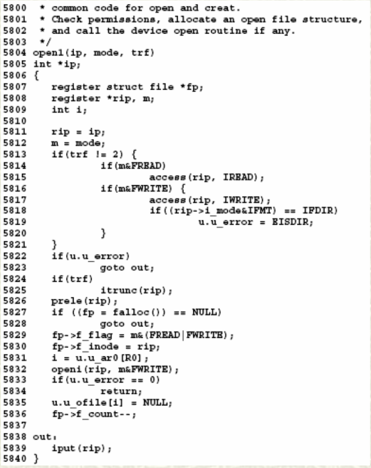
Protection-User/Super User



```
/*
 * Look up a pathname and test if
 * the resultant inode is owned by the
 * current user.
 * If not, try for super-user.
 * If permission is granted,
 * return inode pointer.
 */
owner()
{
    register struct inode *ip;
    extern uchar();

    if ((ip = namei(uchar, 0)) == NULL)
            return(NULL);
    if(u.u_uid == ip->i_uid)
            return(ip);
    if (suser())
            return(ip);
    iput(ip);
    return(NULL);
}
/* -------------------------        */

/*
 * Test if the current user is the
 * super user.
 */
suser()
{

    if(u.u_uid == 0)
            return(1);
    u.u_error = EPERM;
    return(0);
}
/* -------------------------        */
```

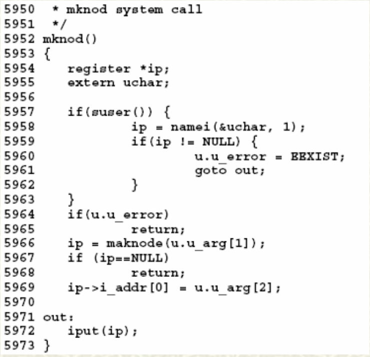
SYSTEM CALLS

Open/Create

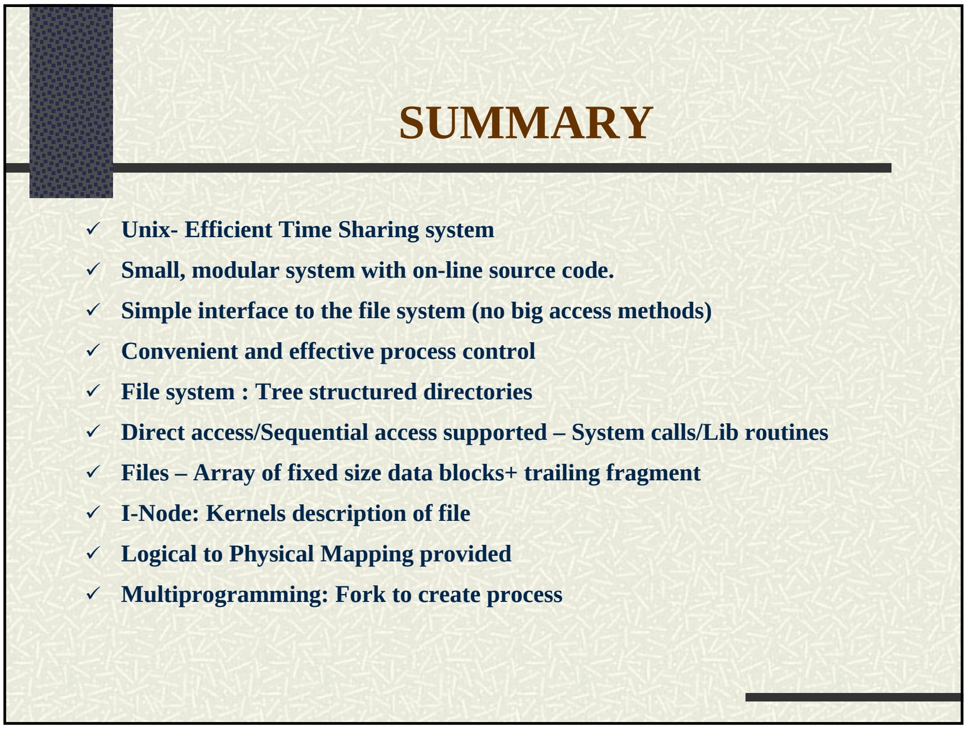


```
 * common code for open and creat.
 * Check permissions, allocate an open file structure,
 * and call the device open routine if any.
 */
open1(ip, mode, trf)
int *ip;
{
    register struct file *fp;
    register *rip, m;
    int i;

    rip = ip;
    m = mode;
    if(trf != 2) {
            if(m&FREAD)
                    access(rip, IREAD);
            if(m&FWRITE) {
                    access(rip, IWRITE);
                    if((rip->i_mode&IFMT) == IFDIR)
                            u.u_error = EISDIR;
            }
    }
    if(u.u_error)
            goto out;
    if(trf)
            itrunc(rip);
    prele(rip);
    if ((fp = falloc()) == NULL)
            goto out;
    fp->f_flag = m&(FREAD|FWRITE);
    fp->f_inode = rip;
    i = u.u_ar0[R0];
    openi(rip, m&FWRITE);
    if(u.u_error == 0)
            return;
    u.u_ofile[i] = NULL;
    fp->f_count--;

out:
    iput(rip);
}
```

mkNode



```
 * mknod system call
 */
mknod()
{
    register *ip;
    extern uchar;

    if(suser()) {
            ip = namei(&uchar, 1);
            if(ip != NULL) {
                    u.u_error = EEXIST;
                    goto out;
            }
    }
    if(u.u_error)
            return;
    ip = maknode(u.u_arg[1]);
    if (ip==NULL)
            return;
    ip->i_addr[0] = u.u_arg[2];

out:
    iput(ip);
}
```



SUMMARY

✓ Unix- Efficient Time Sharing system

✓ Small, modular system with on-line source code.

✓ Simple interface to the file system (no big access methods)

✓ Convenient and effective process control

✓ File system : Tree structured directories

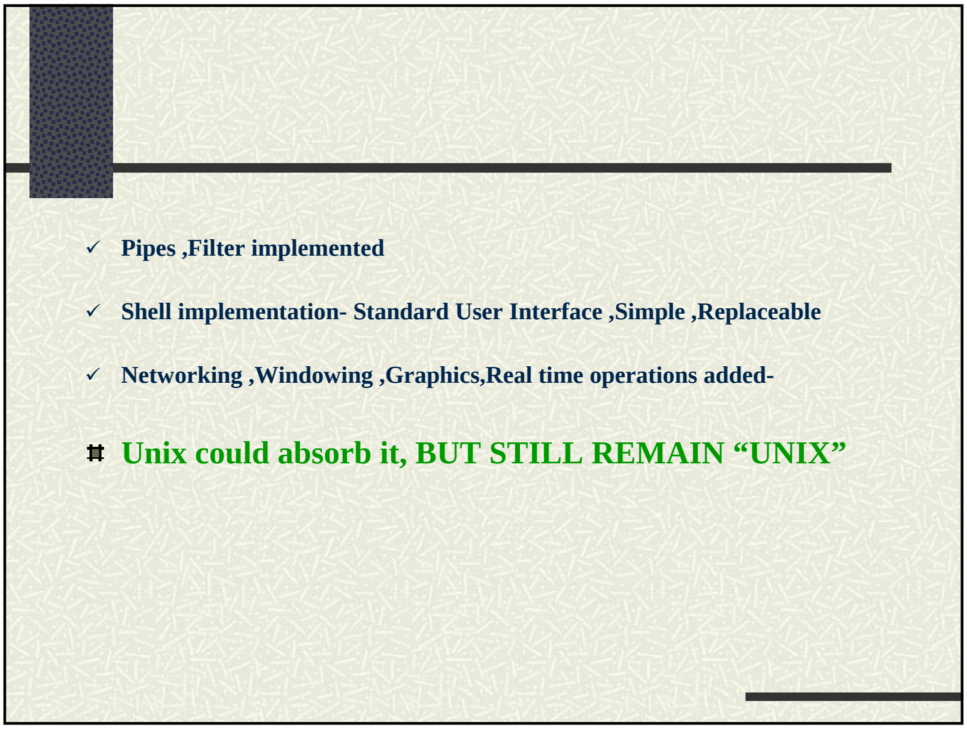
✓ Direct access/Sequential access supported – System calls/Lib routines

✓ Files – Array of fixed size data blocks+ trailing fragment

✓ I-Node: Kernels description of file

✓ Logical to Physical Mapping provided

✓ Multiprogramming: Fork to create process

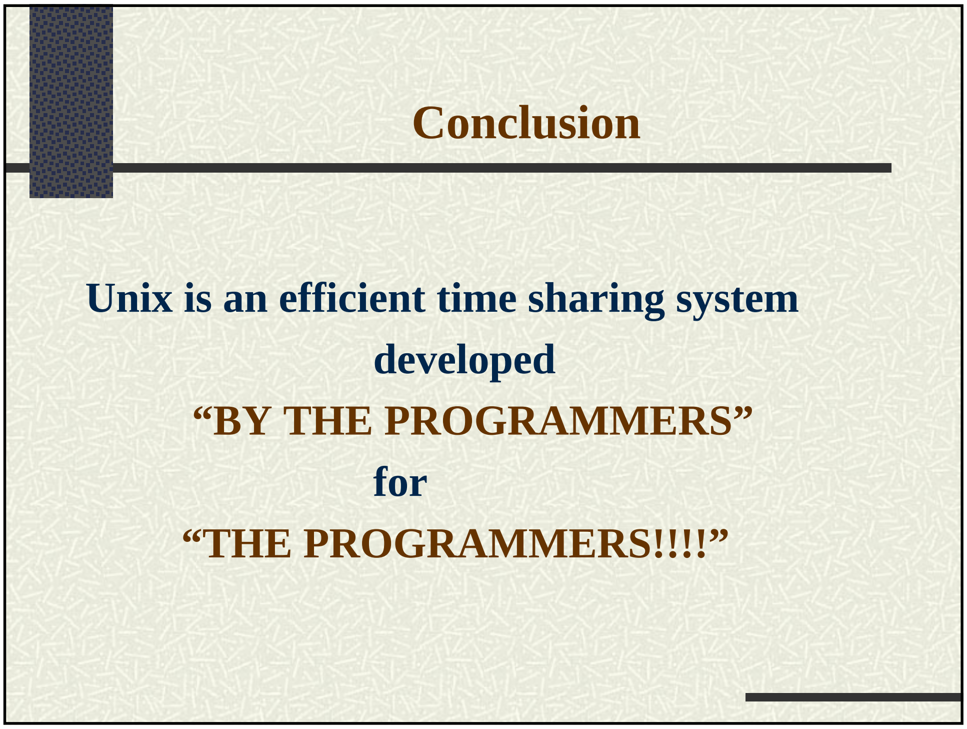


✓ Pipes ,Filter implemented

✓ Shell implementation- Standard User Interface ,Simple ,Replaceable

✓ Networking ,Windowing ,Graphics,Real time operations added-

⌗ Unix could absorb it, BUT STILL REMAIN “UNIX”



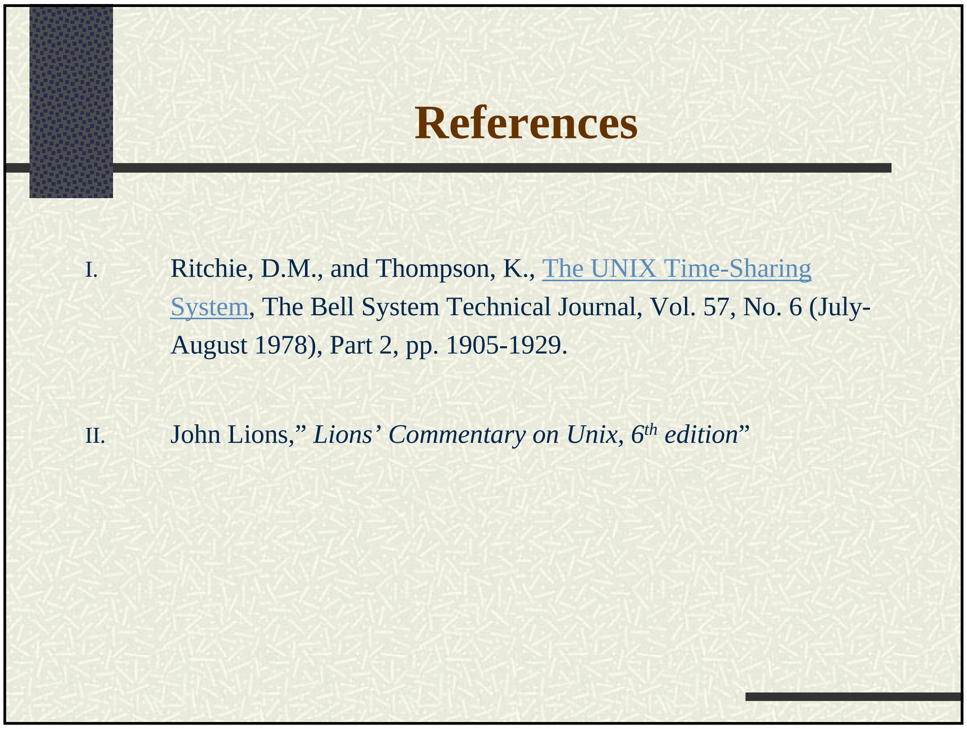
Conclusion

Unix is an efficient time sharing system developed

“BY THE PROGRAMMERS”

for

“THE PROGRAMMERS!!!!”



References

I. Ritchie, D.M., and Thompson, K., The UNIX Time-Sharing System, The Bell System Technical Journal, Vol. 57, No. 6 (July-August 1978), Part 2, pp. 1905-1929.

II. John Lions,” *Lions’ Commentary on Unix, 6th edition*”