UNIX

Unit 1

Q1. What do you mean by multiuser system?

Ans: Multi-user is a term that defines an operating system or application software that allows concurrent access by multiple users of a computer. Time-sharing systems are multi-user systems. Most batch processing systems for mainframe computers may also be considered "multi-user", to avoid leaving the CPU idle while it waits for I/O operations to complete. However, the term "multitasking" is more common in this context.

Q2. Explain soft link and hardlink.

Ans: Soft link:-This is a Symbolic link between files.The actual file or directory must be residing at any available partitions of the harddisk Soft Link is just a shortcut (in windows terms) or link created with a new file name at the working directory or at current working partition of hard disk.Even when you don't require it you can confidently delete this soft link as it doesn't remove the actual file or directory.The reason is the actual file or diretcory's inode is different from the softlink created file's inode in any unix system.

Hard link:-It is the replica of the actual file or directory which must be residing at any available partitions. This is a duplicate file copy of it's orginal which can be created at current working partition.When we remove or delete this hardlink it removes the original file or directory too.The reson is they share the same inode in any unix file system.

Q3. Describe features of Unix

Ans: The UNIX Operating System is available on machines with a wide range of computing power, from microcomputers to mainframes, and on different manufacture's machines. No other operating system can make this claim.

Features of UNIX

Portability:
The system is written in high-level language making it easier to read, understand, change and, therefore move to other machines. The code can be changed and complied on a new machine. Customers can then choose from a wide variety of hardware vendors without being locked in with a particular vendor.

Machine-independence:
The System hides the machine architecture from the user, making it easier to write applications that can run on micros, mins and mainframes.

Multi-User Operations:
UNIX is a multi-user system designed to support a group of users simultaneously. The system allows for the sharing of processing power and peripheral resources, white at the same time providing excellent security features.

Hierarchical File System:
UNIX uses a hierarchile file structure to store information. This struture has the maximum flexibility in grouping information in a way that reflects its natural state. It allows for easy maintenance and efficient implementation.

UNIX shell:
UNIX has a simple user interface called the shell that has the power to provide the services that the user wants. It protects the user from having to know the intricate hardware details.

Pipes and Filters:
UNIX has facilities called Pipes and Filters which permit the user to create complex programs from simple programs.

Utilities:
UNIX has over 200 utility programs for various functions. New utilities can be built effortlessly by combining existing utilities.

Software Development Tools:
UNIX offers an excellent variety of tools for software development for all phases, from program editing to maintenance of software,

Q4. Features and benefits of unix file system.
Ans: UNIX file systems offer the following features, and have for many years:

- Excellent expandability, and support for large storage devices.
- Directory-level and file-level security and access controls, including the ability to control which users or groups of users can read, write or execute a file.
- Very good performance and efficient operation.
- The ability to create "flexible" file systems containing many different devices, to combine devices and present them as a single file system, or to remotely mount other storage devices for local use.
- Facilities for effectively dealing with many users and programs in a multitasking environment, while requiring a minimum of administration.
- Ways to create special constructs such as logically linked files.
- Reliability and robustness features such as journaling and support for RAID.

Benefits :

- Full multitasking with protected memory. Multiple users can run multiple programs each at the same time without interfering with each other or crashing the system.
- Very efficient virtual memory, so many programs can run with a modest amount of physical memory.
- Access controls and security. All users must be authenticated by a valid account and password to use the system at all. All files are owned by particular accounts. The owner can decide whether others have read or write access to his files.
- A rich set of small commands and utilities that do specific tasks well -- not cluttered up with lots of special options. Unix is a well-stocked toolbox, not a giant do-it-all Swiss Army Knife.
- Ability to string commands and utilities together in unlimited ways to accomplish more complicated tasks -- not limited to preconfigured combinations or menus, as in personal computer systems.
- A powerfully unified file system. Everything is a file: data, programs, and all physical devices. Entire file system appears as a single large tree of nested directories, regardless of how many different physical devices (disks) are included.
- A lean kernel that does the basics for you but doesn't get in the way when you try to do the unusual.
- Available on a wide variety of machines - the most truly portable operating system.
- Optimized for program development, and thus for the unusual circumstances that are the rule in research.

Q5. Describe versions of unix operating system.
Ans:

- AIX by IBM
- BSD/OS (BSDi) by Wind River
- CLIX by Intergraph Corp.
- Debian GNU/Linux by Software in the Public Interest, Inc.
- Tru64 Unix (formerly Digital Unix) by Compaq Computer Corp.
- DYNIX/ptx by IBM (formerly by Sequent Computer Systems)
- Esix Unix Esix Systems
- FreeBSD by FreeBSD Group
- GNU Herd by GNU Organization
- HAL SPARC64/OS by HAL Computer Systems, Inc.
- HP-UX by Hewlett-Packard Company
- Irix by Silicon Graphics, Inc.
- Linux by several groups several
- LynxOS by Lynx Real-Time Systems, Inc.
- MacOS X Server by Apple Computer, Inc.
- NetBSD by NetBSD Group

- NonStop-UX by Compaq computer Corporation
- OpenBSD by OpenBSD Group
- OpenLinux by Caldera Systems, Inc.
- Openstep by Apple Computer, Inc.
- Red Hat Linux by Red Hat Software, Inc.
- Reliant Unix by Siemens AG
- SCO Unix by The Santa Cruz Operation Inc.
- Solaris by Sun Microsystems
- SuSE by S.u.S.E., Inc.
- UNICOS by Silicon Graphics, Inc.
- UTS by UTS Global, LLC

Q6. What is Vi editor ? what is the need of it ? explain various modes of its.
Ans: The vi editor (short for visual editor) is a screen editor which is available on almost all Unix systems. Once you have learned vi, you will find that it is a fast and powerful editor. vi has no menus but instead uses combinations of keystrokes in order to accomplish commands. If you are just beginning to learn Unix, you might find the Pico editor easier to use (most command options are displayed at the bottom of the screen). If you use the Pine email application and have composed or replied to a message you have probably already used Pico as it is used for text entry. For more information please refer to the Pine/Pico page.

There are two basic modes of vi:

- Command mode
  This is the default when you enter vi. In command mode, most letters, or short sequences of letters, that you type will be interpreted as commands, **without explicitly pressing Enter** . If you press **Esc** when you're in command mode, your terminal will beep at you. This is a very good way to tell when you're in command mode. (So is **:set** showmode -- see".exrc Profile and Miscellaneous Commands".)

- Insert mode
  In insert mode, whatever you type is inserted in the file at the cursor position. Type **a** (lowercase letter a, for append) to enter insert mode from command mode; press **Esc** to end insert mode, and return to command mode.

Need of Vi

- sometimes it's the only available editor
- when you log on remotely (ssh) to a Unix host from a Mac or PC, only a text editor (like VI or emacs or pico) can be used to edit files in a text-only terminal window
- mouse movements (e.g., menus, highlighting, clicking, scrolling) slow down the touch-typist. VI requires none
- as mentioned above, VI is the editor sure to be on **every** Unix computer in the world
- VI is a **very** powerful editor for those who learn more than just the beginner commands, and even more powerful for those who are familiar with Unix commands

Q7. Explain the commands:

- **ps**

The ps command displays information about programs (i.e., processes) that are currently running. Entered without arguments, it lists basic information about interactive processes you own. However, it also has many options for determining what processes to display, as well as the amount of information about each. Like lp and lpr, the options available differ between BSD and System V implementations. For example, to view detailed information about all running processes, in a BSD system, you would useps with the following arguments:

```
ps -alxww
```

To display similar information in System V, use the arguments:

```
ps -elf
```

- **pwd**

This command reports the current directory path. Enter the command by itself:

```
pwd
```

- **ls**

This command will list the files stored in a directory. To see a brief, multi-column list of the files in the current directory, enter:

```
ls
```

- **chmod**

This command changes the permission information associated with a file. Every file (including directories, which Unix treats as files) on a Unix system is stored with records indicating who has permission to read, write, or execute the file, abbreviated as r, w, and x. These permissions are broken down for three categories of user: first, the owner of the file; second, a group with which both the user and the file may be associated; and third, all other users. These categories are abbreviated as u for owner (or user), g for group, and o for other.

To allow yourself to execute a file that you own named myfile, enter:

```
chmod u+x myfile
```

To allow anyone who has access to the directory in which myfile is stored to read or execute myfile, enter:

```
chmod o+rx myfile
```

- **mv**

This command will move a file. You can use mv not only to change the directory location of a file, but also to rename files. Unlike the cp command, mv will not preserve the original file.

Note: As with the cp command, you should always use -i to make sure you do not overwrite an existing file.

To rename a file named old name in the current directory to the new name new name, enter:

```
mv –i old name new name
```

- **find**

The find command lists all of the files within a directory and its subdirectories that match a set of conditions. This command is most commonly used to find all of the files that have a certain name.
To find all of the files named myfile.txt in your current directory and all of its subdirectories, enter:

```
find . –name myfile.txt –print
```

To look in your current directory and its subdirectories for all of the files that end in the extension .txt , enter:

```
find . –name “*.txt” –print
```

UNIT- 3

1 Define buffer header ?

Ans:-

- A buffer consists of two parts
  - a memory array
  - buffer header
- disk block : buffer = 1 : 1

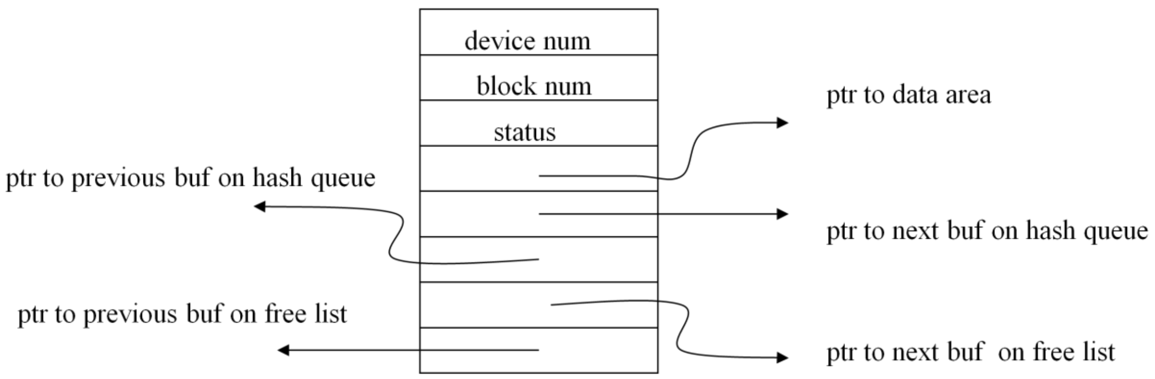

device num
block num
status
ptr to data area
ptr to previous buf on hash queue
ptr to next buf on hash queue
ptr to previous buf on free list
ptr to next buf on free list

Figure 3.1 Buffer Header

1 Explain structure of buffer pool ?

- Buffer pool according to LRU
- The kernel maintains a free list of buffer
  - doubly linked list
  - take a buffer from the head of the free list.
  - When returning a buffer, attaches the buffer to the tail.

Ans:-

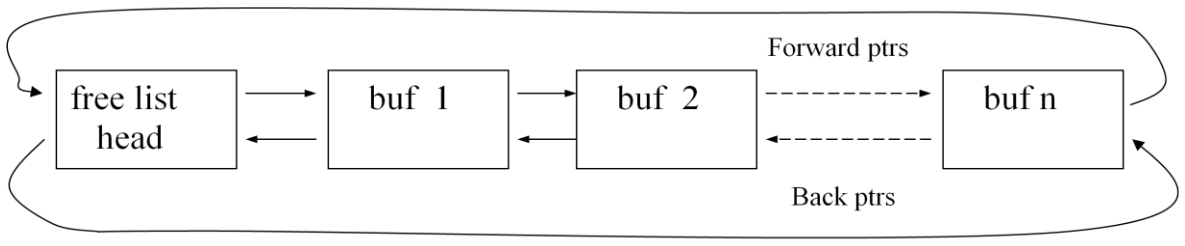


Figure 3.2 Free list of Buffers

- When the kernel accesses a disk block
  - separate queue (doublely linked circular list)
  - hashed as a function of the device and block num
  - Every disk block exists on one and only on hash queue and only once on the queue

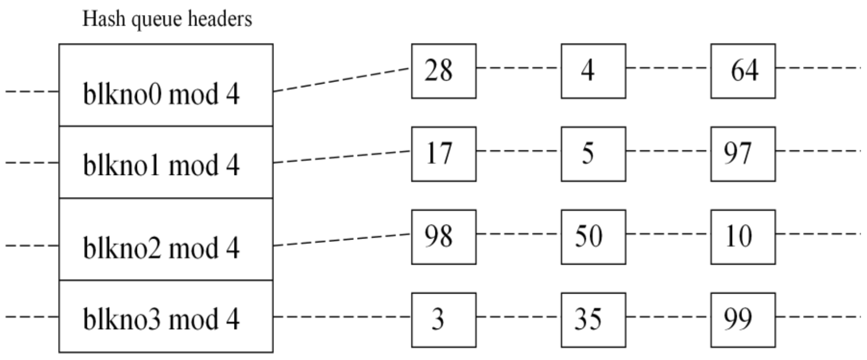


Figure 3.3 Buffers on the Hash Queues

2 What is buffer cache ? Explain advantage and disadvantages .

Ans:

Buffer Cache

-

- The kernel attempts to minimize the frequency of disk access
- Buffer cache == pool of buffers
- Buffer cache contains the data in recently used disk blocks
- Higher-level kernel algorithms instruct the buffer cache modules to pre-cache data or to delay-write data to maximize caching effect

Advantages of the buffer cache

- Uniform disk access => system design simpler
- Copying data from user buffers to system buffers => eliminates the need for special alignment of user buffers.
- Use of the buffer cache can reduce the amount of disk traffic.
- Single image of of disk blocks contained in the cache => helps insure file system integrity

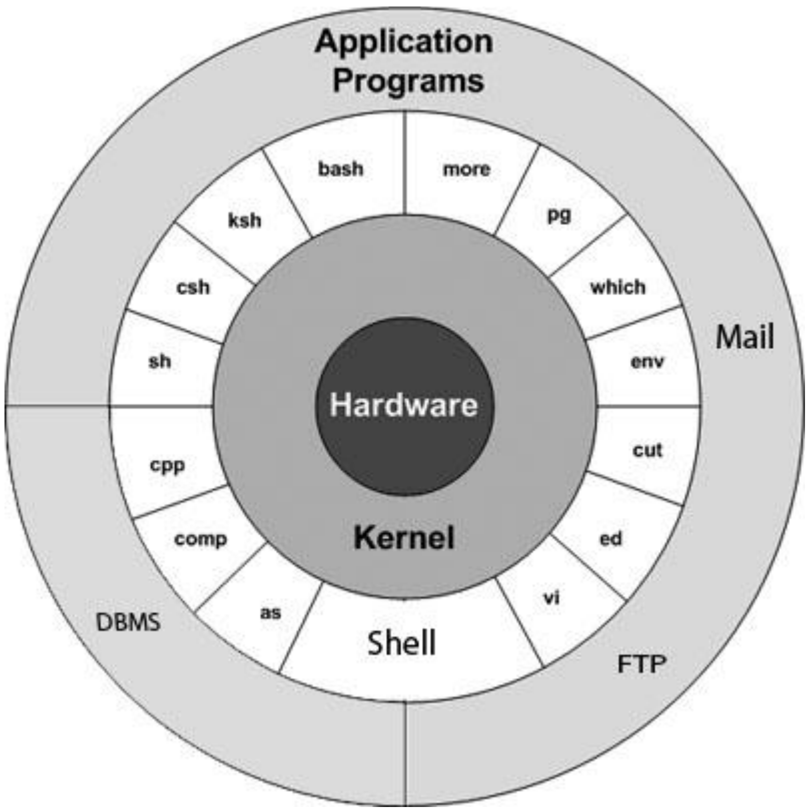
Disadvantages of the buffer cache

- Delayed write => vulnerable to crashes that leave disk data in incorrect state
- An extra data copy when reading and writing to and from user processes => slow down when transmitting large data

3 Explain architecture of UNIX O.S. with the block diagram ?

Ans:- Unix Architecture:

Here is a basic block diagram of a UNIX system:



The main concept that unites all versions of UNIX is the following four basics:

- **Kernel:** The kernel is the heart of the operating system. It interacts with hardware and most of the tasks like memory management, tash scheduling and file management.
- **Shell:** The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are most famous shells which are available with most of the Unix variants.
- **Commands and Utilities:** There are various command and utilities which you would use in your day to day activities. **cp, mv, cat** and **grep** etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various optional options.

- **Files and Directories:** All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

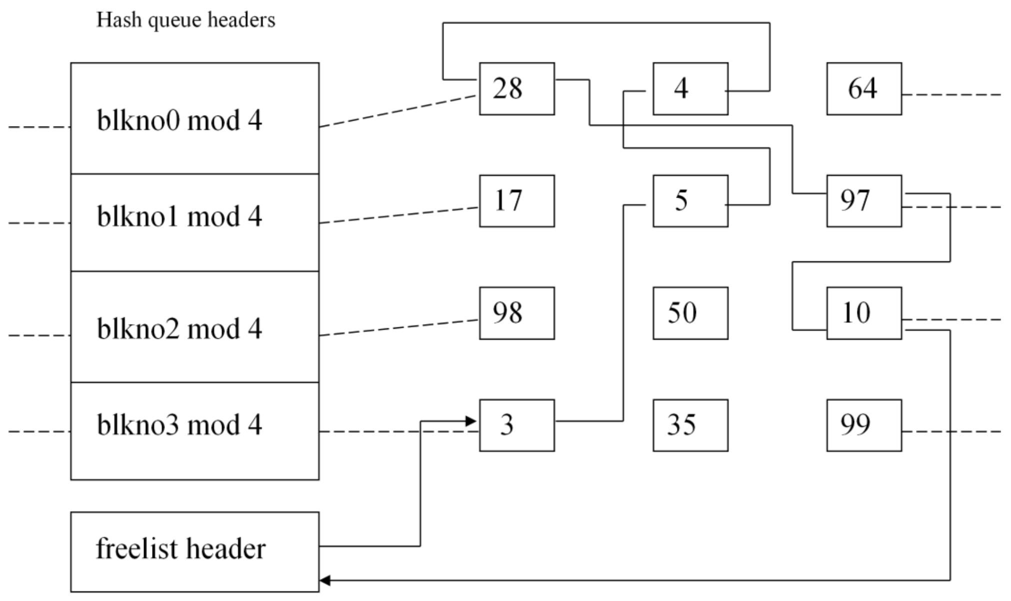
4 Explain the various scenario for retrieval of the buffer ?

Ans:-

Scenarios for retrieval of a buffer

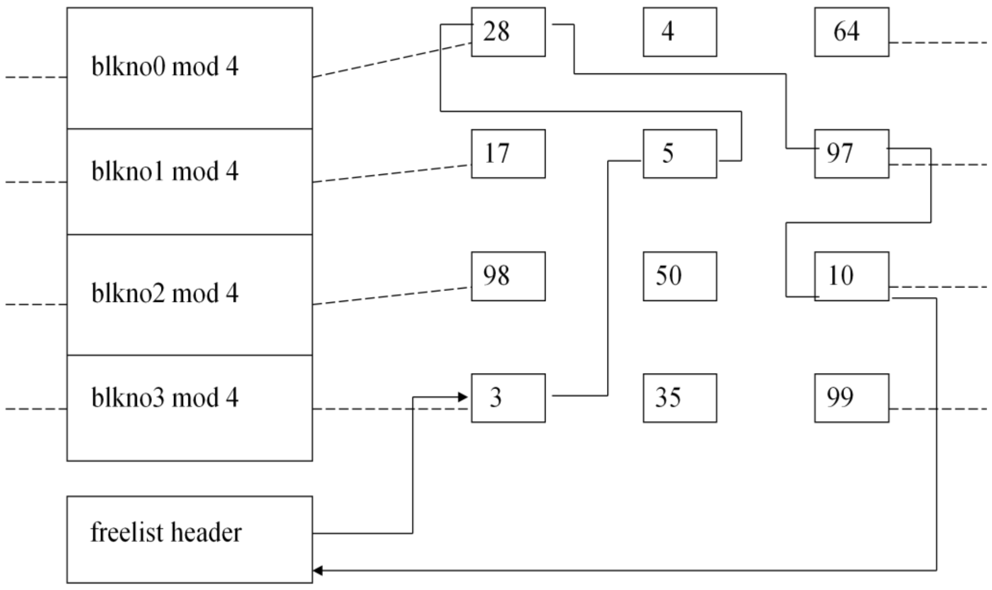
- Determine the logical device num and block num
- The algorithms for reading and writing disk blocks use the algorithm *getblk*
  - The kernel finds the block on its hash queue
    - The buffer is free.
    - The buffer is currently busy.
  - The kernel cannot find the block on the hash queue
    - The kernel allocates a buffer from the free list.
    - In attempting to allocate a buffer from the free list, finds a buffer on the free list that has been marked "delayed write".
    - The free list of buffers is empty.

- The kernel finds the block on the hash queue and its buffer is free



Search for block 4

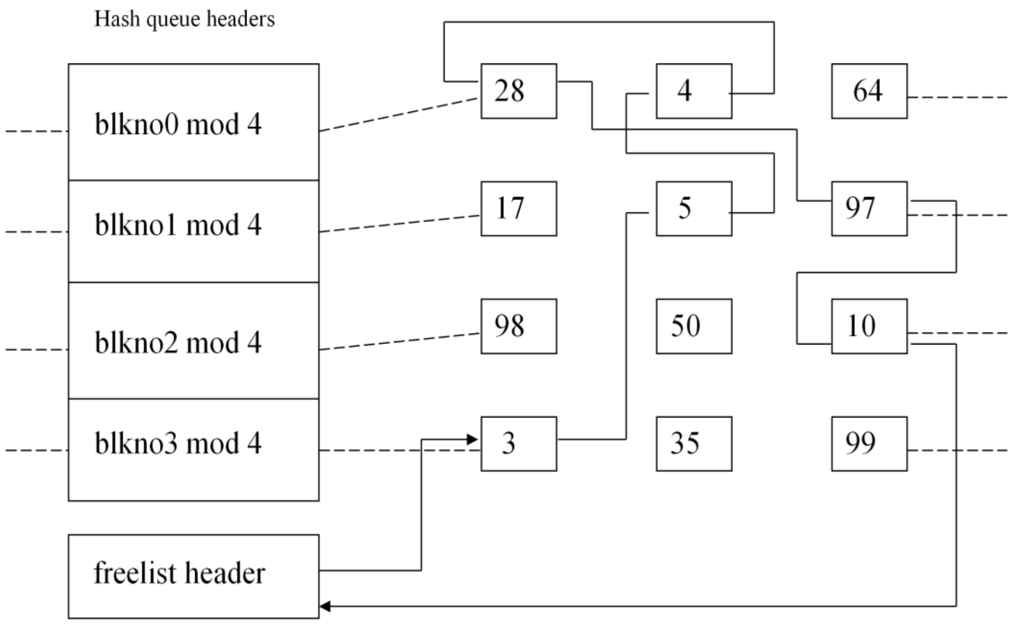
Retrieval of a Buffer:1st Scenario (b)



Remove block 4 from free list

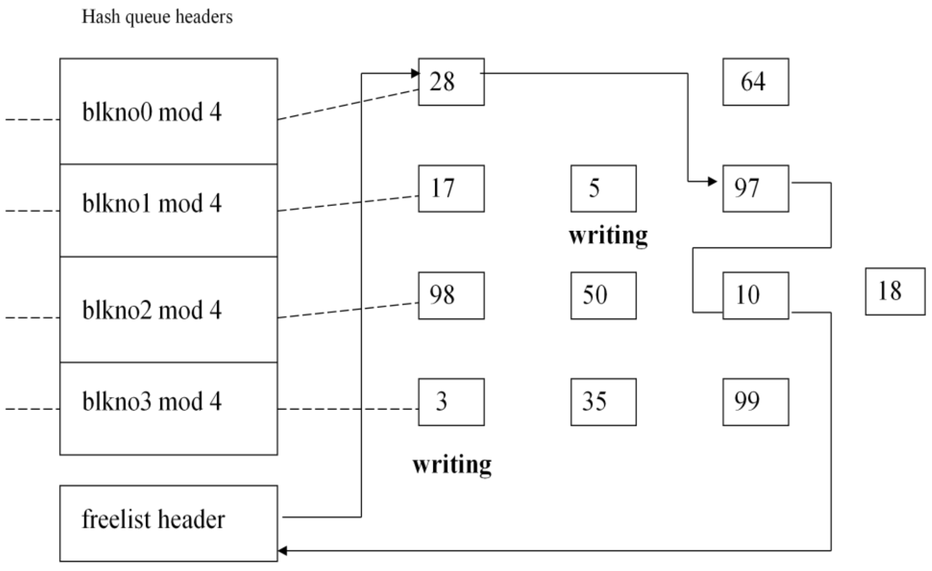
Retrieval of a Buffer: 3rd Scenario (a)

- The kernel cannot find the block on the hash queue, and finds delayed write buffers on hash queue



Search for block 18, Delayed write blocks on free list

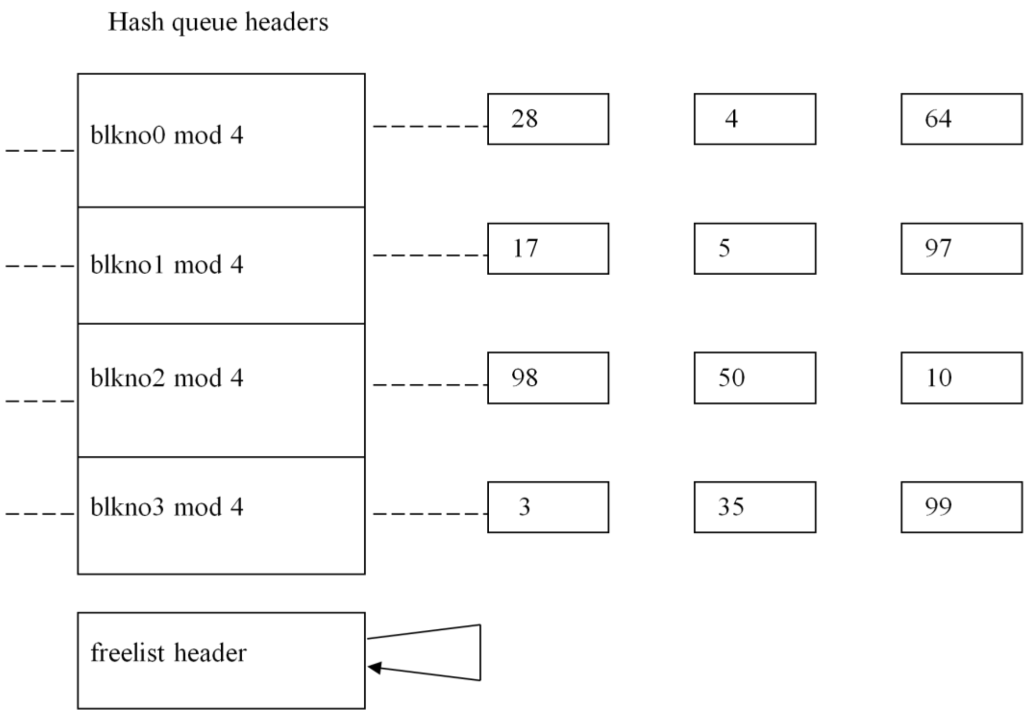
Retrieval of a Buffer: 3rd Scenario (b)



(b) Writing Blocks 3, 5, Reassign 4 to 18
Figure 3.8

Retrieval of a Buffer: 4th Scenario

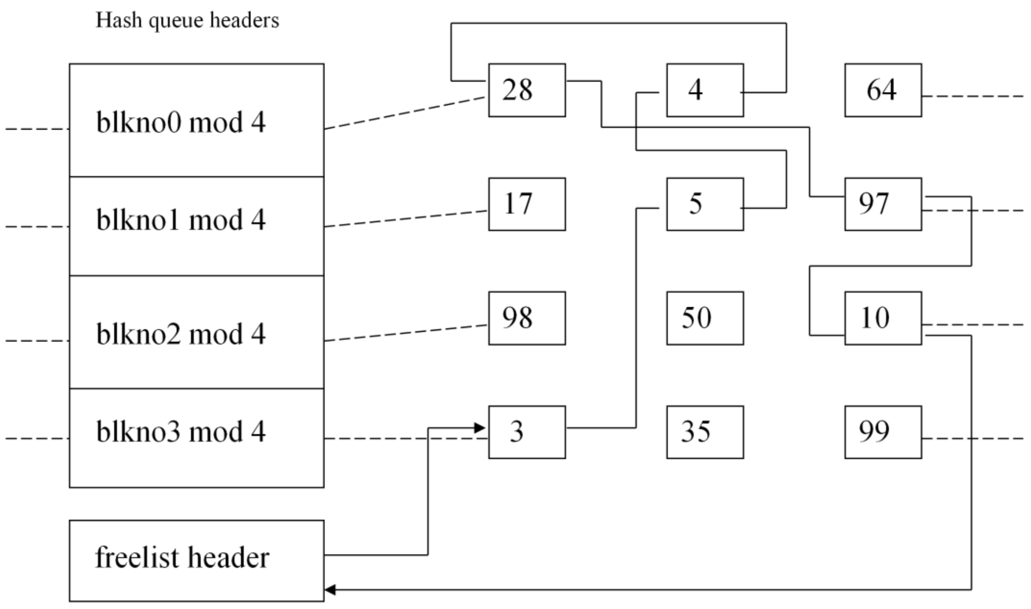
- The kernel cannot find the buffer on the hash queue, and the free list is empty



Search for block 18, free list empty

Retrieval of a Buffer: 5th Scenario

- Kernel finds the buffer on hash queue, but it is currently busy



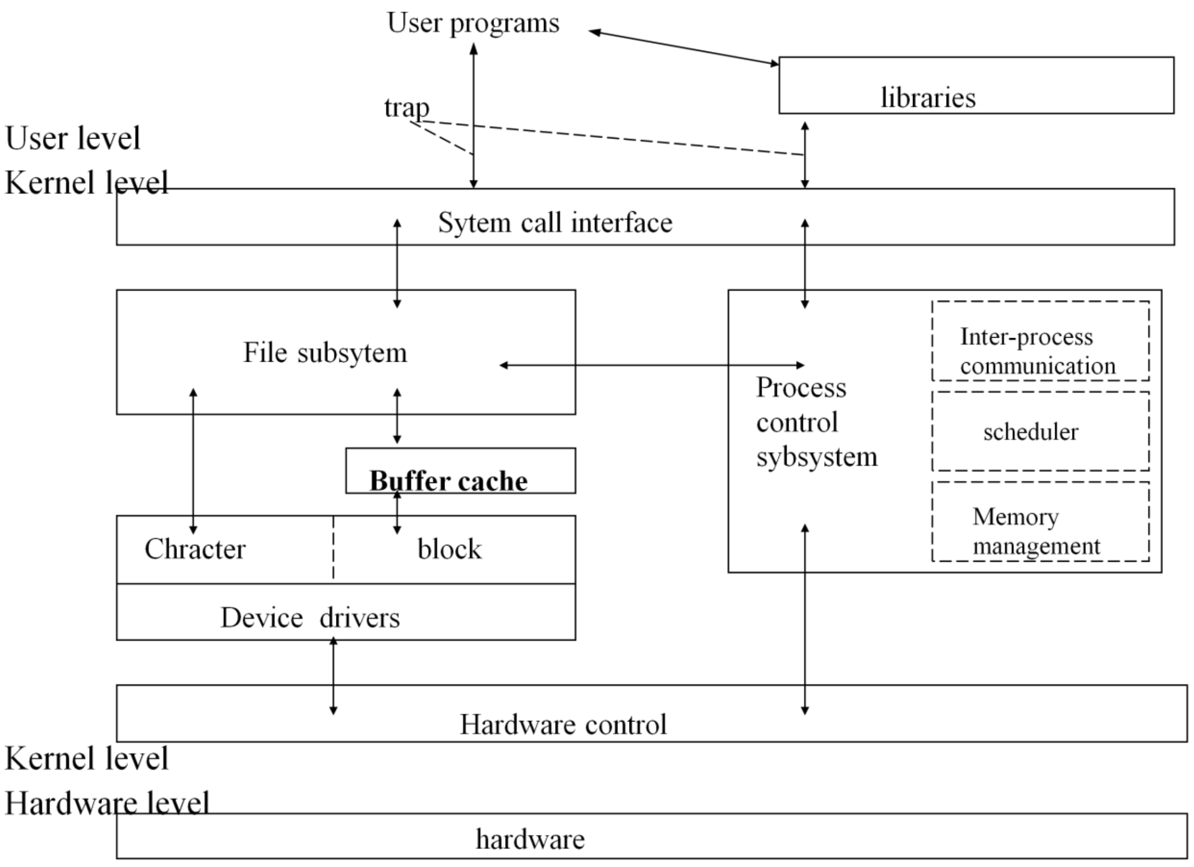
Search for block 99, block busy

5 Explain & draw block diagram of UNIX kernel .

Ans

:-

UNIX Kernel Architecture



- Three major tasks of kernel:
  - ✠ Process Management
  - ✠ Device Management
  - ✠ File Management
- Three additional Services for Kernel:
  - Virtual Memory
  - Networking
  - Network File Systems
- Experimental Kernel Features:
  - ✈ Multiprocessor support
  - ✈ Lightweight process (thread) support

Process Control Subsystem

- Process Synchronization
- Interprocess communication
- Memory management:
- Scheduler: process scheduling
  (allocate CPU to Processes)

File subsystem

- A file system is a collection of files and directories on a disk or tape in standard UNIX file system format.
- Kernel's file sybsystem regulates data flow between the kernel and secondary storage devices.

Hardware Control

- Hardware control is responsible for handling interrupts and for communicating with the machine.
- Devices such as disks or terminals may interrupt the CPU while a process is executing.
- The kernel may resume execution of the interrupted process after servicing the interrupt.

6 Explain file system layout ?

Ans:- File system layout
Xv6 lays out its file system as follows. Block 0 is unused, left available for use by the operating system boot sequence. Block 1 is called the superblock; it contains metadata about the file system. After block 1 comes a sequence of inodes blocks, each containing inode headers. After those come bitmap blocks tracking which data blocks are in use, and then the data blocks themselves.
The header fs.h (3700) contains constants and data structures describing the layout of the file system. The superblock contains three numbers: the file system size in blocks, the number of data blocks, and the number of inodes.

7 Write algo for block read ahead?

Ans:-

Read Ahead

```
Algorithm breada
{
    if(first block not in cache)
    {
        get buffer for first block(algorithm getblk);
        if(buffer data not valid)
            initiate disk read;
    }
    if(second block not in cache)
    {
        get buffer for second block(algorithm getblk);
        if(buffer data valid)
            release buffer(algorithm brelse);
        else
            initiate disk read;
    }
```

```
    if(first block was originally in cache)
    {
        read first block(algorithm bread);
        return buffer;
    }
    sleep(event first buffer contains valid data);
    return buffer;
}
```

8 Explain procedure for reading and writing of disk block ?

Ans:-

Reading a disk block

```
algorithm bread   /*block bread*/
input: file system block number
output: buffer containing data
{
    get buffer for block(algorithm getblk);
    if(buffer data valid)
            return buffer;
    initiate disk read;
    sleep(event disk read complete);
    return(buffer);
}
```

Writing a disk block

```
algorithm bwrite   /*block write*/
input:                  buffer
output:   none
{
     initiate disk write;
     if(I/O synchronous)
     {
              sleep(event I/O complete);
              release buffer(algorithm brelse);
     }
     else if(buffer marked for delayed write)
              mark buffer to put at head of free list;
}
```

9 Write algo for buffer allocation /

Ans:-

Algorithm for buffer allocation

```
while(buffer not found)
{
    if(block in hash queue)
    {
        if(buffer busy)          /*scenario 5*/
        {
            sleep(event buffer becomes free);
            continues; /*back to while*/
        }
        mark buffer busy;        /*scenario 1*/
        remove buffer from free list;
        return buffer;
    }

    /* Next Page… */
```

```
    else
    {
        if(there are no buffer on free list)          /*scenario 4 */
        {
            sleep(event any buffer becomes free);
            continue;                 /*back to while loop*/
        }
        remove buffer from free list;
        if(buffer marked for delayed write) {         /*scenario 3 */
            asynchronous write buffer to disk;
            continue;                 /*back to while loop*/
        }
        /* scenario 2 – found a free buffer */
        remove buffer from old hash queue;
        put buffer onto new hash queue;
        return buffer;
    }
}
```

UNIT-4

1 Define super block ?
Ans-A superblock is a record of the characteristics of a filesystem, including its size, the block size, the empty and the filled blocks and their respective counts, the size and location of the inode tables, the disk block map and usage information, and the size of the block groups.

2 what is system call give the name of some system call?
Ans-In computing, a **system call** is how a program requests a service from an operating system's kernel that it does not normally have permission to run. System calls provide the **interface between a process and the operating system. Most operations interacting with the** system require permissions not available to a user level process, e.g. I/O performed with a device present on the system, or any form of communication with other processes requires the use of system calls.

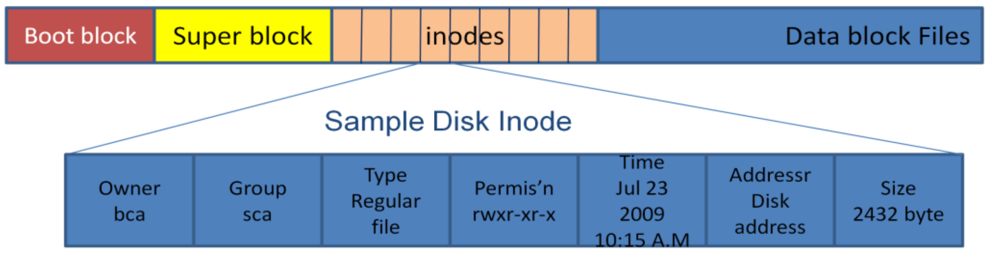
3 content of an inode entry?
Ans-Mode, uid ,gid ,atime ,ctime ,mtime ,size ,block count ,reference count ,direct blocks(10) ,single indirect,double indirect ,triple indirect

4 What is inode ? draw & explain a sample disk inode?
Ans- Inode-

- Contains permissions, owner, groups and last modification times.
- Type of file: regular, directory or special file
- If it is symbolic link, the value of the symbolic link.
- If it is a regular file or directory, contains location of its disks block

Sample Disk inode:-

- File owner identifier
- File type
- File access permission
- File access time
- Number of links to the file
- Table of contents for the disk address of data in a file
- File size



1. **File owner identifier**

   There are three types of user in the UNIX operating system such as Owner, Group users, Other users. The Owner has rights to access all files in the UNIX operating system. The owner is otherwise called as super user.

2. File type

UNIX OS has four different types of files such as Ordinary file, Directory, Special file, FIFO or pipes. File type fields specify the files belong to the which types of the file.

3. File access permissions

It specifies the access rights to the three types of user to read, write and execute the UNIX files and directories.

4. File access time

It gives the information about the last modification and last access time of the file and the last modification of the Inode.

5. Number of links to the file

It represent the number of names the file has in the Unix directory hierarchy.

6. Table of contents

It has the disk addresses of data in the file. User consider the data in a file as a logical stream of bytes, but the kernel saves the data in discontiguous disk blocks.

7. File size

Size of th file is the number of bytes from the beginning of the file. The size of the file is greater than 1 by actual size of the file.

5 Explain Open system call?

Ans- In most modern operating systems, a program that needs access to a file stored in a filesystem uses the **open** system call. This system call allocates resources associated to the file (the file descriptor), and returns a handle that the process will use to refer to that file from then on.

The open system call can take three arguments:

1. The pathname to the file,
2. the kind of access requested on the file (read, write, append etc.),
3. the initial file permission is requested using the third argument called mode. This argument is relevant only when a new file is being created
4. **The open system call interface is standardized by the POSIX specification. A file is opened or created by calling the open function:-**
5. int open (const char *pathname, int oflag, .../*,mode_t mode */);
6. The value returned is the new file descriptor (in POSIX, file descriptors are non-negative integer values). This integer is usually an index on a table of open files for the process. The operating system keeps independent tables for different processes. The file descriptor contains, among other things, a position pointer that indicates which place of the file will be acted upon by further file operations.
7. The same filesystem file can be opened simultaneously by several processes, and even by the same process (resulting in several file descriptors for the same file). Operations on the descriptors like moving the file pointer, or closing it are independent (they do not affect other descriptors for the same physical file). However, operations of the physical file (like a write) can be seen by operations on the other descriptors (a posterior read can read the written data)
8. Alternatively, open can return a negative number indicating that the open operation failed
9. The pathname argument is the name of the file to open. It is a file path indicating in which place of the file system the file should be found (or created if that is requested)
10. There are a multiple options for this function, which are specified by the oflag argument.
11. This argument formed by OR'ing together one or more of the following constants (from <fcntl.h>)
12. **O_RDONLY**
    Open for reading only,
13. **O_WRONLY**
    Open for writing only,
14. **O_RDWR**
    Open for reading and writing,
15. Optionally, you can also use the following parameters (the following list is not exhaustive, it intends to show what kind of options can be used with open):
16. **O_APPEND**
    When using this flag, all data written will be appended to the end of the file. The file operations will ignore the position pointer inside the file descriptor and always append at the end of the file, even when other processes are also accessing the file

17. **O_CREAT**
    This flag will make open to create the file if it does not exist; otherwise the open attempt fails (setting errno to ENOENT).
18. **O_TRUNC**
    If the file already exists (and is a regular file rather than, say, a device or named pipe) then discard its previous contents, reducing it to an empty file.
19. **O_EXCL**
    When this flag is used with O_CREAT, and the file already exists, then fail, setting errno to EEXIST.
20. **O_NONBLOCK**
    Operations on the file descriptor will fail (setting errno to EWOULDBLOCK) rather than causing the calling process to block, e.g., if there is no data currently available to satisfy a read, or the kernel's buffer is full and cannot accept a write. Regular files on disk never `block' in this way, despite the fact that the process must often wait for data to arrive from the disk. This option is most useful with other kinds of files such as named pipes and devices.

6 Explain namei algo in detail ?
Ans- Algorithm for conversion of a path name to an inode :

```
Algorithm namei          /* convert path-name to inode */
Input    : path name
Output   : locked inode
{
        if (path name starts from root)
                working inode = root inode (algorithm iget);
        else
                working inode = current directory inode ( algorithm iget);
        while (there is more path name)
        {
                read next path name component from input;
                varify that working inode is of directory, access permissions OK;
                if (working inode is of root and component is "..")
                        continue;           /* loop back to while */
                read directory ( working inode ) by repeated use of alogrithms
                                bmap, bread and brelse;
                if (component matches an entry in directory (working inode))
                {
                        get inode number for matched component;
                        release working inode (algorithm iput);
                        working inode = inode of matched component (algorithm iget);
                }
                else
                        return ( no inode);
        }
        return (working inode);
}
```

7 Write algorithm for allocating disk block ?
Ans- Algorithm for Allocating Disk Block**:**

```
Algorithm ialloc         /* allocate inode */
Input    : file system
Output   : locked inode
{
        while (not done)
        {
                if (super block locked)
                {
                        sleep (event super block become free);
                        continue;                   /* while loop */
                }
```

```
                    if (inode list in super block is empty)
                    {
                              lock super block;
                              get remembered inode for free inode search;
                              search disk for free inodes until super block full,
                                 or no more free inodes (algorithms bread and brelse);
                              unlock super block;
                              wake up (event super block becomes free);
                              if (no free inodes found on disk)
                                        return (no inode);
                              set remembered inode for next free inode search;
                    }
                    /* there are inodes in super block inode list */
                    get inode numbers in super block inode list;
                    get inode (algorithm iget);
                    if (inode not free after all)          /* !!! */
                    {
                              write inode to disk;
                              release inode (algorithm iput);
                              continue;          /* while loop */
                    }
                    /* inode is free */
                    initialize inode;
                    write inode to disk;
                    decrement file system free inode count;
                    return (inode);
          }
}
```

8 Sort note on -
1 Mounting and Unmounting
2 FSTAT and STAT
3 Link and Unlink

Ans:-

Mounting and unmounting :-

The privileged mount system call is used to attach a file system to a directory of another file system; the unmount system call detaches a file system. When you mount another file system on to your directory, you are essentially splicing one directory tree onto a branch in another directory tree. The first argument to mount call is the mount point, that is , a directory in the current file naming system. The second argument is the file system to mount to that point. When you insert a cdrom to your unix system's drive, the file system in the cdrom automatically mounts to /dev/cdrom in your system.

2 Fstat and stat:-

stat() is a Unix system call that returns useful data about a file inode. The semantics of stat() vary between operating systems. As an example, the Unix command ls uses it to retrieve information on (among many others):

fstat() is a library function that retrieves the status of a file. It is identical to stat() except that the file's identity is passed as a file descriptor instead of as a filename.

3 Link and Unlink:-

The **link** system call is used to create a new link in a directory that references as existing link:

```
int link(const char *src, const char *dst);
```

The program takes existing file test.txt and create new link test2.txt

```
#include <unistd.h>
int main(void) {
```

```
          if (link("test.txt", "test2.txt") == -1) {
                    printf("error\n");
          }
          return 0;
}
```

The user root is the only one who is allowed to perform links on directories.
The **unlink** system call is used to remove a directory entry and it reduce the associated i-node link count of 0 is reached, the data block, of the file are removed.

```
int unlink(const char *file);
```

If a process has an open file descriptor on a file and an unlink is performed on the file, even if the link count has reached zero, the OS will not remove the file. Only after the process terminate and the file descriptor associated with the file is closed, is sthe file data and i-node deleted.
This leads to an important technique for creating tmp files by a process which has to be guaranteed for deletion

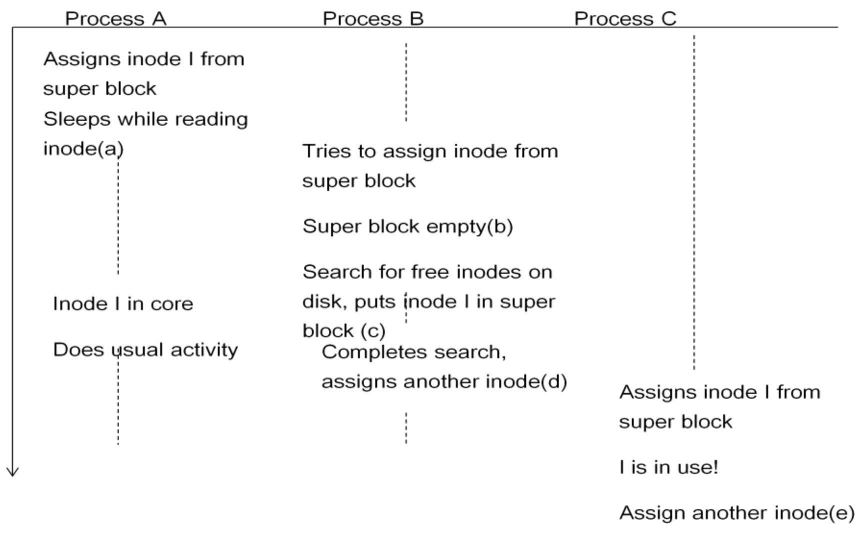
```
// tmpunlink.c
#include <unistd.h>
int main(void) {
          int fd;
          fd = open("/tmp/mytemp.txt", O_CREAT | O_WRONLY, 0777);
          unlink("/tmp/mytemp.txt");
          return 0;
}
```

9 Discuss the race condition in assigning INODES?
Ans:-Race condition-

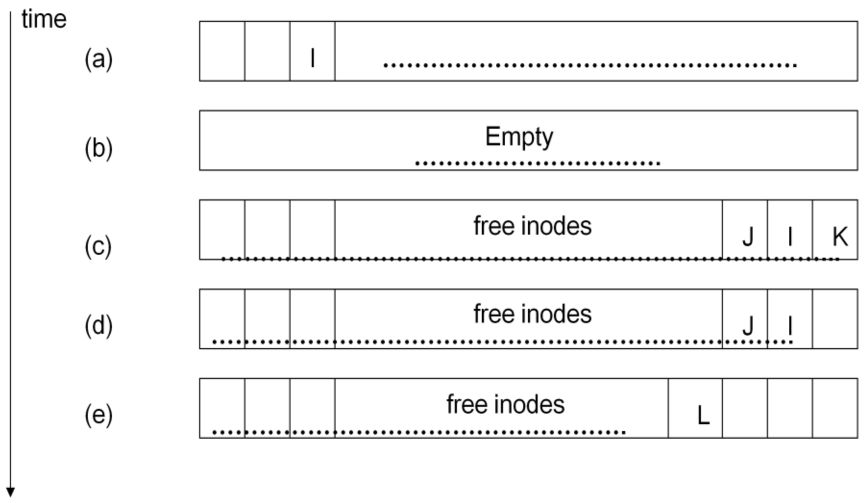
A Race Condition Scenario in Assigning Inodes

three processes A, B, and C are acting in time sequence

1. The kernel, acting on behalf of process A, assigns inode I but goes to sleep before it copies the disk inode into the in-core copy.
2. While process A is asleep, process B attempts to assign a new inode but free inode list is empty, and attempts assign free inode at an inode number lower than that of the inode that A is assigning.
3. Process C later requests an inode and happens to pick inode I from the super block free list
4.



Process A
Process B
Process C
Assigns inode I from super block
Sleeps while reading inode(a)
Tries to assign inode from super block
Super block empty(b)
Search for free inodes on disk, puts inode I in super block (c)
Inode I in core
Does usual activity
Completes search, assigns another inode(d)
Assigns inode I from super block
I is in use!
Assign another inode(e)



10. Explain link system call.
Ans : link()

The UNIX system file structure allows more than one named reference to a given file, a feature called "aliasing". Making an alias to a file means that the file has more than one name, but all names of the file refer to the same data. Since all names refer to the same data, changing the contents of one file changes the contents of all aliases to that file. Aliasing a file in the UNIX system amounts to the system creating a new directory entry that contains the alias file name and then copying the i-number of a existing file to the i-number position of this new directory entry. This action is accomplished by the link() system call. The link() system call links an existing file to a new file.

The prototype for link() is:

```
int link(original_name, alias_name)
char *original_name, *alias_name;
```

where both original_name and alias_name are character strings that name the existing and new files respectively. link() will fail and no link will be created if any of the following conditions holds:

- a path name component is not a directory.
- a path name component does not exist.
- a path name component is off-limits.
- original_name does not exist.
- alias_name does exist.
- original_name is a directory and you are not the superuser.
- a link is attempted across file systems.
- the destination directory for alias_name is not writable.
- the destination directory is on a mounted read-only file system.

Following is a short example:

```
/*  link.c
*/

#include <stdio.h>

int main()
{
  if ((link("foo.old", "foo.new")) == -1)
    {
    perror(" ");
```

```
        exit (1);        /* return a non-zero exit code on error */
        }
      exit(0);
    }
```

11. Write and discuss algo for allocate an in-core copy of an inode.
Ans: /* release (put) access to in-core inode */

```
Input : pointer to in-core inode
Output : none
{
    lock inode if not already locked;
    decrement inode reference count;
    if (reference count == 0)
    {
        if (inode link count == 0)
        {
            free disk blocks for file (algorithm free, section 4.7);
            set file type to 0;
            free inode ( algorithm ifree, section 4.6);
        }
        if (file accessed or inode changed or file changed)
            update disk inode;
        put inode on free list;
    }
    release inode lock;
}
```

Fields of the disk inode

- Status of the in-core inode, indicating whether

    - Inode is locked
    - Process is waiting for the inode to become unlocked
    - Differ from the disk copy as a result of a change to the data in the inode
    - Differ from the disk copy as a result of a change to the file data
    - File is a mount point
    - Logical device number of the file system
    - Inode number (linear array on disk, disk inode not
    - need this field
    - Pointers to other in-core inodes
    - Reference count

Unit 5

1. Define U area.

Ans: In addition to the text data and stack segment the OS also maintains for each process a region called the u area (User Area). The u area contains information specific to the process (e.g. open files current directory signal action accounting information) and a system stack segment for process use. If the process makes a system call (e.g. the system call to ***write*** in the function in main) the stack frame information for the system is stored in the system stack segment. Again this information is kept by the OS in an area that the process doesn't normally have access to. Thus if this information is needed the process must use special system call to access it. Like the process itself the contents of the u area for the process are paged in and out bye the OS.

2. Explain init process.

Ans; Once kernel and drivers are loaded, Linux starts loading the rest of the system. This starts with the First Process, known as init and it has the process id of 1 (the kernel itself has the process id of 0, which cannot be displayed by using the "ps" command).

The init process takes control of the boot operation. The init process in turn runs /etc/rc.d/rc.sysinit, which performs a number of tasks, including network configuration, SELinux status, keyboard maps, system clock, partition mounts, and host names.

The runlevels are controlled by a configuration file which init process reads from the location /etc. The name of the init configuration file is "inittab".

The init process then determines the runlevel by looking at the initdefault directive in /etc/inittab configuration file. The following are the defined runlevels. The init process remains active as long as the system is running.

3. What are the various process system call?

Ans: **Unix Processes**

Multitasking means concurrent execution of programs. A task can be a process or a thread, depending on the operating system. As there are usually more tasks than hardware processors in a computer system, the operating system has to multiplex the resources (processor, memory and I/O) to the tasks. There are various strategies for scheduling. In the following, we will focus on preemptive multitasking, which means that a task has no influence on how long it can keep the resource

1 fork()

In Unix processes are created using the system call fork(). The first process of a program is created upon the start of the program.

2 exec()

When a process wants to execute a different program it can use any of the exec() system calls.

3 wait() & waitpid()

The system calls wait() and waitpid() are used to synchronize a parent process with the termination of its child processes.

4 kill()

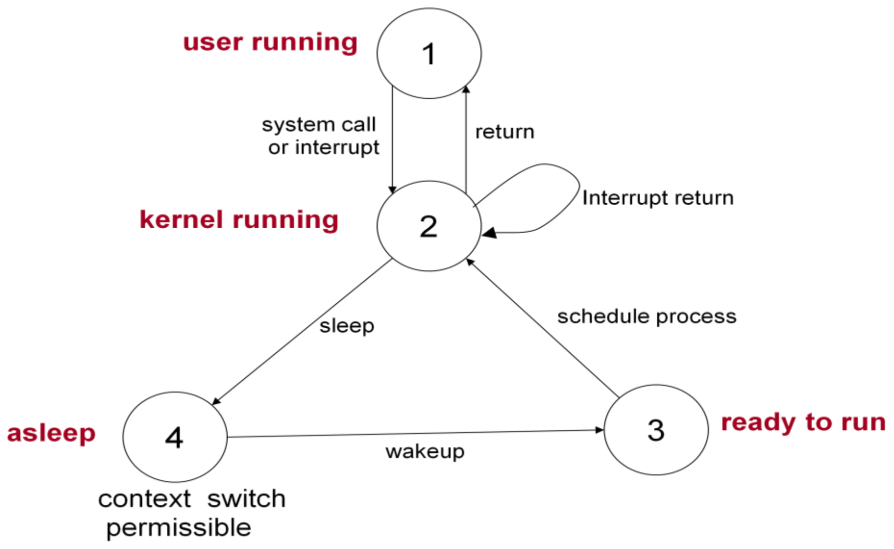
The kill() system call is used to send signals to processes.

4. Explain and draw process state and transition diagram.

Ans: Process states are:

- The process is running in user mode
- The process is running in kernel mode
- The process is not executing, but it is ready to run as soon as the scheduler chooses it
- The process is sleeping
    - Such as waiting for I/O to complete

- The kernel allows a context switch only when a process moves from the state kernel running to the state asleep
- Process running in kernel mode cannot be preempted by other processes.



5. What are the contents of u area of a process.

Ans: In addition to the information about a process held in the process table, the user area (``u-area'') of each process contains information that the process uses when it is running. The process table is permanently resident in memory, but a process's u-area can be swapped out to disk. The form of a u-area corresponds to the **user** structure defined in <*sys/user.h*>.

When issued without arguments, the output of the **user** command contains information about the u-area of the process that is currently executing or, for a memory image, the process that was executing at the time of the panic. If the system was executing the idle loop (no process was runnable) at the time of the panic, there is no current process.

The following sections are of interest:

USER ID's

Shows the real and effective IDs of the user running the process.

PROCESS MISC

Shows miscellaneous information about the process. The fields command and psargs tell you the name of the program and the first few arguments to the command. The start time shows the actual clock time when the process was initialized.

OPEN FILES AND POFILE FLAGS

Tells you the files that the process had open, and the file descriptors involved. The file descriptors in use by the program are in shown square brackets; the F#numbers represent slot numbers in the open file table and can be used as arguments to the **file** command. For example, file descriptor 1 usually corresponds to the standard output (*stdout*) unless this was redefined. In this example, it points to slot 216 of the open file table; you can use the command **file 216** to view this entry.

SIGNAL DISPOSITION

Shows the behavior defined for all signals

6. Short note

Red hat linux

Red Hat Linux, assembled by the company Red Hat, was a popular Linux basedoperating system until its discontinuation in 2004.

Red Hat Linux 1.0 was released on November 3, 1994. It was originally called "Red Hat Commercial Linux. It was the first Linux distribution to use the RPM Package Manager as its packaging format, and over time has served as the starting point for several other distributions, such as Mandriva Linux and Yellow Dog Linux.
Since 2003, Red Hat has discontinued the Red Hat Linux line in favor of Red Hat Enterprise Linux (RHEL) for enterprise environments. Fedora, developed by the community-supported Fedora Project and sponsored by Red Hat, is the free version best suited for home use. Red Hat Linux 9, the final release, hit its official end-of-life on 2004-04-30, although updates were published for it through 2006 by the Fedora Legacy project until that shut down in early 2007.
Features

- Version 3.0.3 was one of the first Linux distributions to support Executable and Linkable Format instead of the older a.out format.
- Red Hat Linux introduced a graphical installer called Anaconda, intended to be easy to use for novices, and which has since been adopted by some other Linux distributions. It also introduced a built-in tool called Lokkit for configuring the firewall capabilities.
- In version 6 Red Hat moved to glibc 2.1, egcs-1.2, and to the 2.2 kernel. It also introduced Kudzu, a software library for automatic discovery and configuration of hardware.
- Version 7 was released in preparation for the 2.4 kernel, although the first release still used the stable 2.2 kernel. Glibc was updated to version 2.1.92, which was a beta of the upcoming version 2.2 and Red Hat used a patched version of GCC from CVS that they called "2.96" The decision to ship an unstable GCC version was due to GCC 2.95's bad performance on non-i386 platforms, especially DEC Alpha. Newer GCCs had also improved support for the C++ standard, which caused much of the existing code not to compile.
- Version 8.0 was also the second to include the Bluecurve desktop theme. It used a common theme for GNOME-2 and KDE 3.0.2 desktops, as well as OpenOffice-1.0. KDE members did not appreciate the change, claiming that it was not in the best interests of KDE.

Ubuntu Linux

Ubuntu is a computer operating system based on theDebian GNU/Linux distribution and distributed as free and open source software. It is named after the Southern African philosophy of *Ubuntu* ("humanity towards others").
With an estimated global usage of more than 12 million users. Ubuntu is designed primarily for desktop use, although netbook and server editions exist as well.Web statistics suggest that Ubuntu's share of Linux desktop usage is about 50%,[10][11]and indicate upward-trending usage as a web server.
Ubuntu is sponsored by the UK-based company Canonical Ltd., owned by South African entrepreneur Mark Shuttleworth. Canonical generates revenue by sellingtechnical support and services tied to Ubuntu, while the operating system itself is entirely free of charge.

7. Explain

Process address space

Address space is the amount of memory allocated for all possible addresses for a computational entity, such as a device, a file, a server, or a networked computer. Address space may refer to a range of either physical or virtual addresses accessible to a processoror reserved for a process. As unique identifiers of singleentities, each address specifies an entity's *location* (unit of memory that can be addressed separately). On a computer, each computer device and process is allocated address space, which is some portion of the processor's address space. A processor's address space is always limited by the width of its address bus andregisters. Address space may be differentiated as either *flat*, in which addresses are expressed as incrementally increasing integers starting at zero, or *segmented*, in which addresses are expressed as separate segments augmented by *offsets* (values added to produce secondary addresses). In some systems, address space can be converted from one format to the other through a process known as thunking.
In terms of IP address space, there has been concern that IPv4 (Internet Protocol Version 4) had not anticipated the enormous growth of the Internet, and that its 32-bit address space would not be adequate. For that reason, IPv6 has been developed with 128-bit address