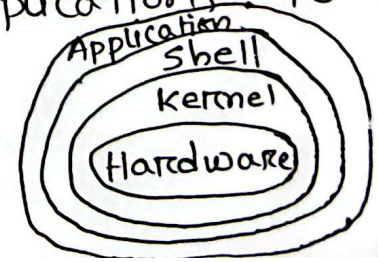
Date:

Answare to the question 1(a)

System programing: System programing involves writing and executing application software programs. system programmers use programing Language to develop software and hardware components and control computer operations in a constrained environment. system programing design and implementation of system software

Typical system programing: OS, compiler, assembler

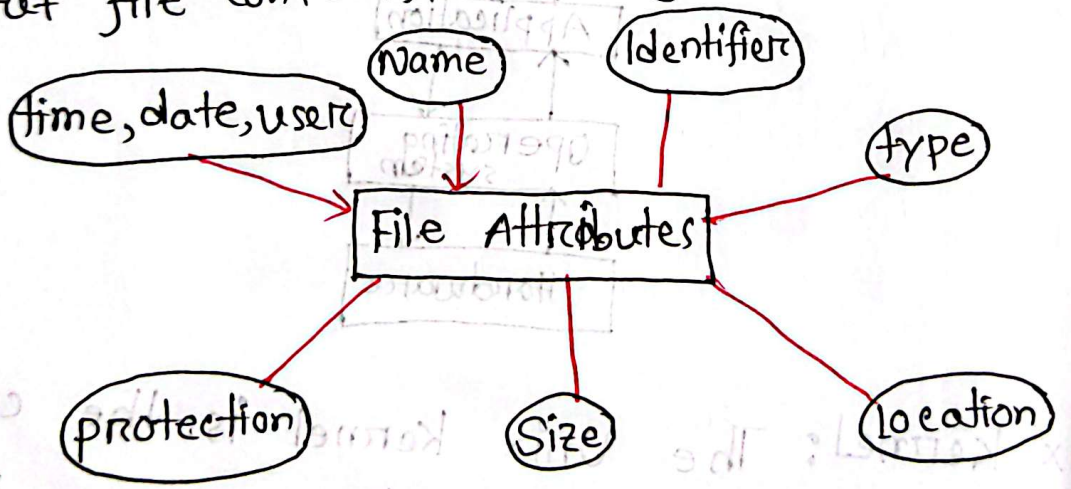
Unix kernel: The Unix kernel is the core of the Unix operating system, acting as an intermediary between hardware and software. It manages essential resource like process, memory and I/O device, providing a foundation for applications to run.



Shell scripting: Shell scripting is a text file, accept command as input from user and execute them. Shell scripting involves writing a series of commands in a text file that an operating systems shell can interpret and execute. This automates task, saving time and reducing errors.

File attributes: File attributes are metadata associated with a file that describes its properties and how it should be treated by the operating system and application. They provide information about file contents.

(basic information of data making it easier, fast use)



Date :

(b)

A file whose file descriptor is fd contains th
following sequence of bytes 3, 1, 4, 1, 5, 9, 2, 6
the following sytems calls are made

```
lseek (fd, 3, seek_set);
read (fd, &buffer, 4);
```

where the lseek call makes a seek to byte 3
the file. what does buffer contain after th
read has completed

File content

Index	0	1	2	3	4	5
Data	3	1	4	1	5	9

lseek (fd, 3, seek_set):
moves the file pointer to byte index 3
which is 4th byte in the file

Read (fd, & buffer, 4):
Reads 4 bytes starting from the index 3

Byte 3 —— 1
Byte —— 5
Byte —— 9
Byte —— 2

Subject:.....

Date:.....

Answer : After the read is completed the buffer contains 1, 5, 9, 2

(c)

before

change command

Answer to the question 2.(a)

cd [<dir>] means the change pwd to home directory, or <dir> if it is supplied. However, "changing directories" and ~~beginning~~ being in a directory" are imprecise pharases. when you cd to a directory named dir, you may think of you as being "in dir", but this is not true. what is the true meaning for cd [<dir>] in order to resolve the relative pathnames?

Relative path: Dosen't start wit a slash and is interpreted relative to the current working directory

print working directory

Subject:
Date:

updating the pwd:

- The cd command resolves the target directory based on its type and the current pwd (current working directory) whether its absolute or relative
- If successful, it updates the pwd to point to the resolved target directory.

Not "being in" a directory:

- We don't physically move into the directory.
- cd just changes the current working directory in our shell environment
- It means only the directory context is updated, not our actual location
- This is important because future commands use this directory as a base for relative paths

Example: current directory /home/user

use command cd documents

the directory will change /home/user/documents

So, when we use ls command it will show files inside documents.

cd moves current directory to another directory

Subject:.....

Date:.....

(b)

In multi-user operating system, the operating system must protect users from each other and protect itself from users. However, while providing an operating environment for all users the operating system creats this illusion (permission for read, write, execution, etc) by creating data paths between user process and devices and files. How does Unix creats this illusion?

Unix creats the illusion of safe, private and controlled access for users by implementing several mechanism.

1. File permission: Each file and directories has read (r), write (w) and execute (x) permission. permission are defined for
   (i) owner
   (ii) group
   (iii) others

Example: rwxr-xr-- means
owner : read, write, execute
group : read, execute
others : read only.

Subject:
Date:

11. User & Group id: Every process runs with a user id & Group id

operating system check these id against file permission to allow an deny access.

3. System calls & kernel mode:

User process cannot access hardware on files directory, Instead they use system (reads write open ())

Kernel checks permission before allowing access.

4. File directories:

The os use file description to manage open files.

5. File hirearcy: Unix organize file in a hierarcical driectories structure

6. Device a file:

Subject:.....
Date:.....

(e)

Each user has a username and a number, the uid why?

*

Every user has both a username (Human-readable string) and uid (user identifier)

username human readable name that's easy to remember

on the other hand, uid is a unique number that the kernel uses internally to identify and manage user efficiently.

Numbers are faster for the kernel to process and they stay the same even if the username is changed

So we need both a friendly name for humans and permanent number for the kernel.

Date:

wouldn't it be simpler to record the username of the user as the owner of a file?

It might seem simpler to use the username directly as the owner of a file, but this approach has some problems.

username are strings, which are slower for the kernel to process and compare

Also, username can be changed, while a uid is permanent and always stay the same

why not have a single identifier for each user?

Single identifier wouldn't work well for both because:

username is for human readable
uid is for kernel.

Subject:.....

Date:.....

Answer to the question 3(a)

A system call is like a conventional function call in that it causes a jump to a subroutine followed by a return to the caller. But it is significantly different. Distinguish between system call and function call based on the following keywords: privileged/ kernel mode, trap instruction, system dependent

	System call	Function call
privileged/kernal mode:	Requires a switch from user to kernel mode to execute	Executes in user mo without changing pri
Trap Instruction:	Uses a trap to transfer control to kernel	perforems direct no trap
system dependent:	yes provide by the OS kernel, depends on the operating system	Language o compiler fea not depend on OS

(b)

Unix files have a full set of permission bits, including a set-user-ID bit and a set-group-id bit. If you turn on the set-group-ID bit for a directory, does it have any effect? If, so, what and why? If not, could you think of some use for this bit?

yes, setting the set-group Id (set-GId) bit on a directory has a special effect.

when set-GId is set on a directory all new files and subordinates created within that directory will inherit the directory's group, instead of the creating primary group

Without SGID bit: New files or directories inherit the primary of the group of the user who creats them.

With SGID bit set: new files and subordinates inherit the ownership of the directory itself, not the users primary group.

Subject:
Date:

gids helpful collaborative environments where multip
users are working in a shared directory.
Ensures group ownership

(c)

Analyze the following three commands

(i) ps - ax | grep corn :

ps - ax : Lists all runing process on the system

-a : shows process from all users

-x : includes process not attached to a terminal

| : This is pipe sysmbol. It takes ~~output of the command~~ pipes the output of ps-ax into the next command.

grep corn : Search for lines containing the word 'corn'.

(ii) ps - ax | tee process.txt

ps - ax (i) Answer

tee process.txt

(i) takes input from : ps - ax
to
• writes file process . text .

Answer to the question 4(a)

The kernel had to locate a free inode and free disk blocks when it created a new file. How does the kernel know which blocks are free? How does the kernel know which inodes are free? what method does the file system on your machine use to keep track of unused blocks and inodes?

When the kernel needs to create a new file, it must find free disk blocks and a free inode to allocate for the new file. The kernel relies on the file system to keep track of which blocks and inodes are free and available for allocation.

1. Free block tracking:

The file system maintains a data structure, often called the "free block bitmap" or "free block list", to track which disk block are available for use.

Bitmap: This is a common approach, where each bit in a bitmap represents a block.

0 bit Indicates the block is free.

1 bit Indicates the block is already allocated.

This method is simple and efficient for small filesystem but can become impractical for large ones due to memory requirements

Free block list: This maintain a linked list of free blocks. This List allows for efficient block allocation but requires traversing the list to find a suitable block.

Extent-based allocation: This method groups free blocks into Larger contiguous সাধারণভাবে blocks are placed side side regions called "extents". Tracking extents instead of individual blocks improves efficiency in larger file system.

Free Inode tracking:

- Inode Bitmap: Similar to the free block bitmap
  - 0 bit → Indicates free block
  - 1 bit → Indicates Allocated block
- Free Inode List: Maintains a linked list of free inodes, similar to the free block list
- Inode allocation group: groups multiples inodes and corresponding data blocks together. This improves allocation efficiency.

Example:

Ext2/3/4: use bitmaps for both free blocks and inodes

XFS: use extend-based allocation

Btrfs: Extend-based allocation

Subject: .....

Date: .....

(b)

A directory is just a node in a set of linked nodes. using the pwd command you can simply know the current directory. Based on the functionalities of pwd answer the following.

(i) what repetitive steps are performed to compute its current directory?

- Starting point: It starts with the current working directory name
- Read directory entry: It fetches the directory entry of the current directory, which has name such information as parent directory and name
- check if root: parent directory is called root dir
- current directory: parent directory information is used to update the current directory

Date:

(ii) How do we know when we read the top of the tree?

using pwd we will be at the root of the directory (the top)

using pwd (print working directory), we know we have reached the top the directory tree (the root directory), when it has no more parent directories than the current one.

(iii) How do we print the directory names in the correct order?

A recursive or an iterative algorithm is used to compute the order of printed directory.

Recursive: Each level of the directory tree is processed recursively, printing the current directory name before processing its parent.

Iterative: The directory is searched by iteratively but directory names are stored on a stack. upon reaching the root, the names are popped off the stack in reverse order.

Subject:.....
Date:.....

Answer to the question 5(a)

before

(b)

In unix, there is a single hierarchy and every accessible file is in this single file hierarchy, no matter how many disk are attached. Similar to windows, there is no such thing as the "C" drive" or "E" in unix. If any new file system arrive then how UNIX adapt itself? Give examples.

In UNIX, there is a single unified hierarchy starting from the root directory /. unlike windows, Unix does not use drive letters like C or D. when a new file system arrives UNIX adapts by mounting that file system onto an existing directory in the single directory tree. ~~this tree is called mound calle tree~~ This directory is called the mount point.

Mounting: A new file system is attached (add) to an existing directory in the main UNIX file system. This directory is called the mount point

file system representation: The contents of the new file system become accessible as if they were subdirectories and files within that mount point. The contents of this new file system are made available as subdirectories or files of that mount point.

Unified view: Users access the mounted file system through the usual folder structure, without needing to know about the actual physical disks or drives.

Example: suppose we have a new hard drive (/dev/sdb1) and we want to use it in our directory (/mnt/new_drive)

using mount command

mount /dev/sdb1 /mnt/new_drive

/dev/sdb1 (new hard drive)

/mnt/new_drive (directory)

Subject:.....
Date:.....

(e)

How does Ls-l work? Briefly explain the data flow in who command with figures.

In simple terms, "ls-l" is a command used in unix-like operating system (such as Linux) to list detailed information about files and and directories in a directory.

The "ls" command stand for "list", and "-l" option adds extra details, showing things Like file permissions, ownership, file size, modification date, and file or directory name.

Ls-l work:

1. Reading directory entries from the request location -
2. For each file making system calls to retrieve metadat
3. converting raw data into human readable format (permission, ownership, size)
4. Formating & printing the information in columns.

Data flow in who commands:

[user terminal] → who command execution →

↓

[who process] → reads /var/run/utmp → file →

↓

[system call] → open(), read(), system call()

↓

[kernel] → provides file access & data

↓

[who process] → parses user session record

↓

[Data formatting] → converts readable output

↓

[Terminal display] → show logged in user

Subject:.....

Date:.....

Answer to the question 6 (a)

The stat system call is passed the name of a file and a pointer to a struct and fills the struct information about the file. Explain, using the directory, Inode, and data model, how stat works?

The stat system call plays a crucial role in Unix-like systems, returning valuable information about a file based on its name.

Receiving the Request: The application makes a stat system call, providing the filename. The kernel takes the filename and initiates the search proc

Navigating (moving or finding) the directory tree: The kernel starts from the current working directory, traversing the directory structure. It searches for an entry with the matchi filename. This search involves comparing filenam within directory blocks.

Reaching the Inode: the matching file name is found, the kernel retrives the associated inode number. The inode acts as unique identifier and metadata store for the file.

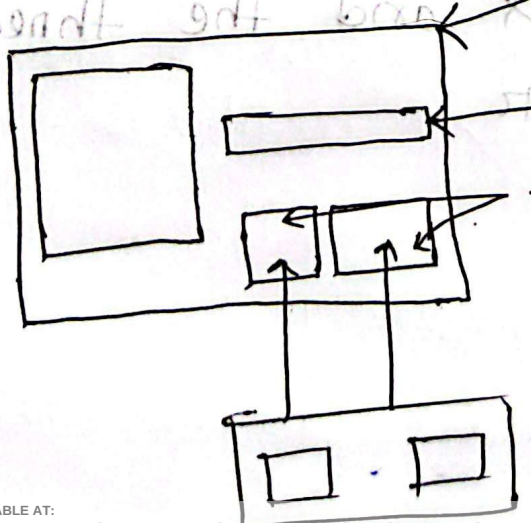
Filling the information: using the rettrieve inode, the kernel extracts various details about file

1. file size
2. permission
3. ownership
4. modification time
5. Access time

(b)

How can we design a multithread program to count and print the total number of words in three files? Give two use case scenarios for classification.

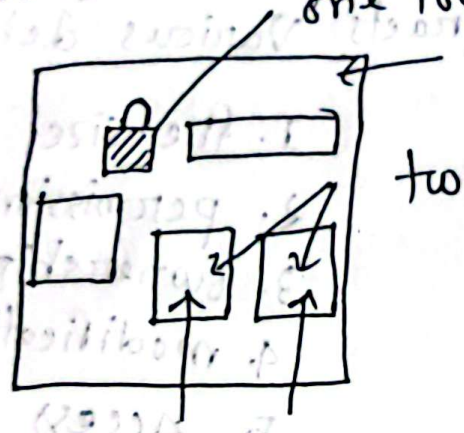
Version 1:



Subject:
Date:

our first version creates a separate thread to count each file. Both threads increment the same counter as they detect words. both threads fetch the same value, increment the register, and then store the new value. Two increments take place, but the value of the counter increase only one by one.

Version 2:



The two threads share the counter safely. If one threads calls pthread_mutex_lock when other thread has locked the mutex, that thread blocks until the mutex is unlocked. As soon as the mutex is unblocked, the call to pthread_mutex_lock unblocks and the thread can increment the counter.

Date:

the two threads share counter safely. when the other thread is locking the mutex, call pthreads_ muter_ lock, one of the threads will block until the mutex was unlocked. when the mutex is unlocked, the call pthreads_ mutex_lock on other thread unblocks and thread increment the counter.

(c)

write short notes:

(1) pthread_ create, pthread_ join

pthread_create:

creats a new thread of execution within the same process

takes four arguments

(i) store the thread ID
(ii) thread attributes
(iii) new thread will execute
(iv) Return 0 on success, an error code otherwise.

Subject:_____
Date:_____

pthread_join: waits for a specific thread to terminate

takes two arguments

1. thread ID
2. optional pointer to a void pointer to store the return value of the joined thread.
3. Return 0 success, an error code otherwise

(ii) pthread mutex_lock, pthread mutex_unlock

pthread_mutex lock:

Acquires a mutex lock

takes argument:

1. Blocks the calling thread until the mutex is available
2. ensure access to shared resource and prevent race condition.

pthread_mutex_unlock:

Release a previously acquired mutex lock

avoid deadlocks when accessing shared resource.