System programming

Mid (18-19)

(1) ⊡ system programming : It is the process of designing and writing computer programs that allow the hardware to interact with the user and the programmer. system programs provide an environment for developing and executing programs.

⊡ Unix kernel API : It provides a set of functions and system calls that allow user-level applications to interact with the Unix operating system's kernel. The kernel is the core component of the operating system that manages system resources, facilitates communication between hardware and software.

⊡ shell scripting : shell scripting involves writing a series of commands in a text

file that the operating system's shell can execute. It automates tasks by combining and executing commands in a sequence.

[1] File Attribute: File attributes are a type of meta-data that provides additional information about a file. They can help organize and manage file more efficiently.

(2)

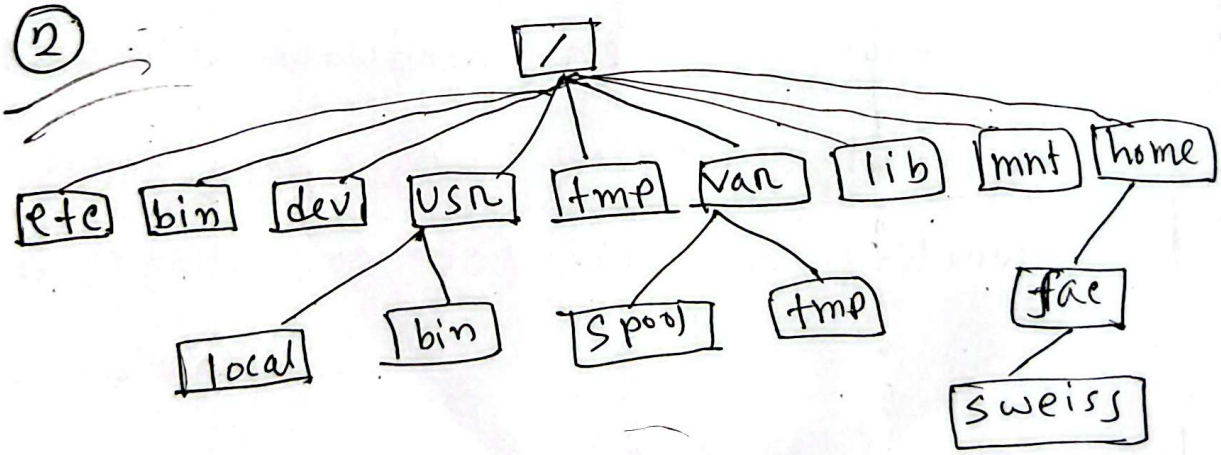


fig: The top of a typical UNIX directory hierarchy.

How ls -l work:

Working procedure:

(1) command execution: When I run the 'ls -l' command in the terminal, the shell interprets the command and invokes the 'ls' program with the '-l' flag.

(2) Reading directory: The 'ls' program reads the specified directory to retrieve a list of files and subdirectories.

(3) collecting file information: For each file and subdirectory 'ls' gathers various metadata and attributes from the file system.

(4) Formatting and display: The collected file information is then formatted according to the long listing format. where each entry is displayed on a separate line.

(5) output to terminal: once the file information is collected and formatted, 'ls' outputs the formatted data to the terminal.

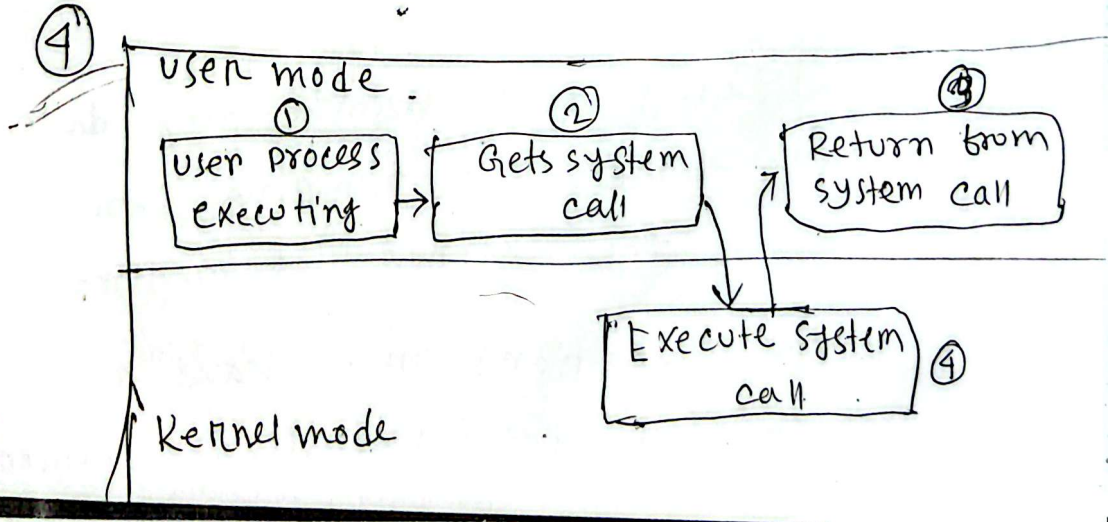
(3)

~~the~~ while using only a username as a user identifier might seem simpler, there are several reasons why systems typically use both usernames and UID's:

(i) security: UID's are unique and unambiguous, eliminating the risk of confusion. username are often publicly visible, but UID's are typically ~~to~~ hidden. This protect password from ~~being~~ being easily cracked.

(ii) Efficiency: UID's are smaller than usernames and require less storage space. This allows for faster system operation. Using UIDs for internal operations avoids the overhead of case-sensitive string comparisons, improve system performance.

(iii) Scalability: UIDs guarantee a unique identifier for every user. UIDs can be easily assigned and reassigned as needed.



TOPIC NAME

Yes, system calls are time consuming. The translation between user space and kernel space incurs overhead, and the execution of the system call itself may involve complex operations. However, the degree of time consumption varies based on the nature of the system call and the efficiency of the operating system.

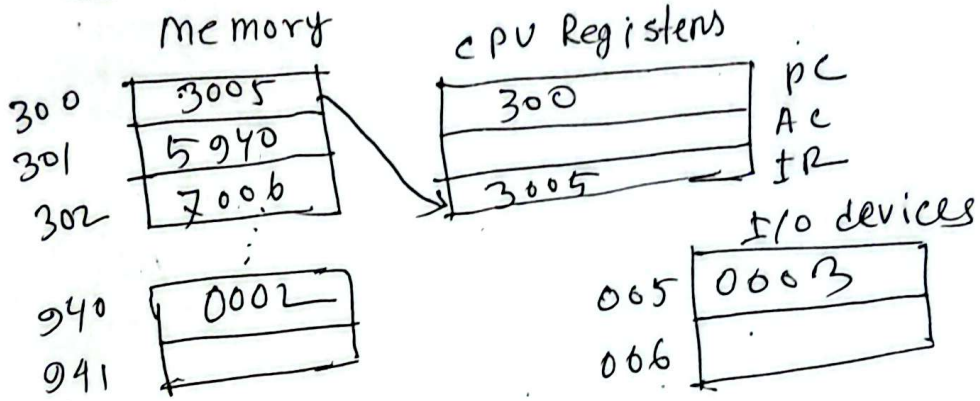
Not only does transferring data take time, but jumping into and out of the kernel takes time.

Kernel functions must have access to the disk, the terminals, the printers, and the other resources. User functions must not have access to those resources. Therefore computer actually operates in different mode.

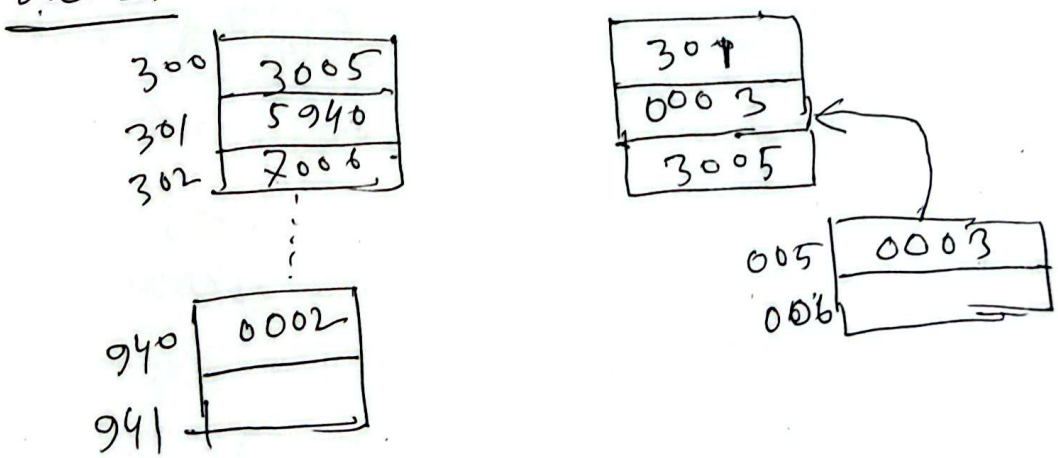
(5)

Assume that the next value retrieved from device 5 is 3 and that location 940 contains a value of 2.

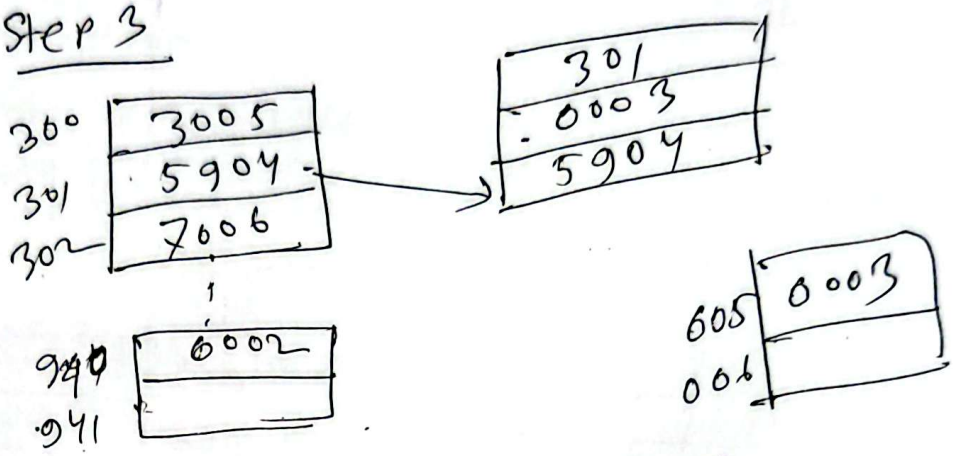
Step 1



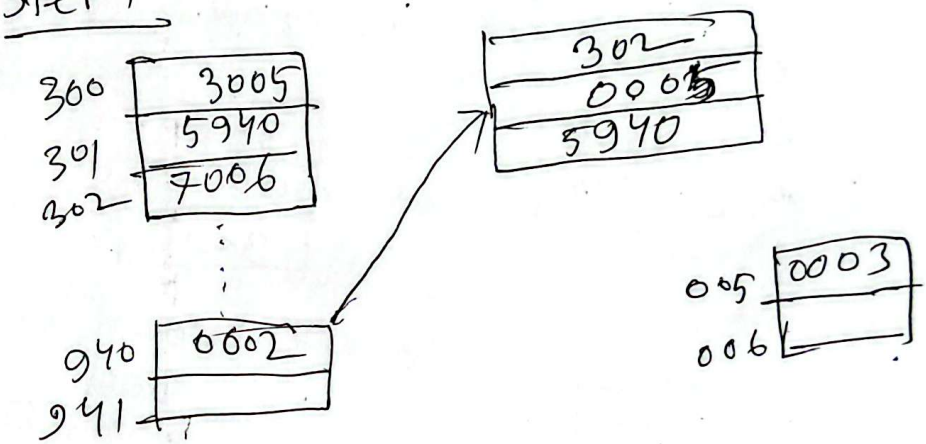
Step 2



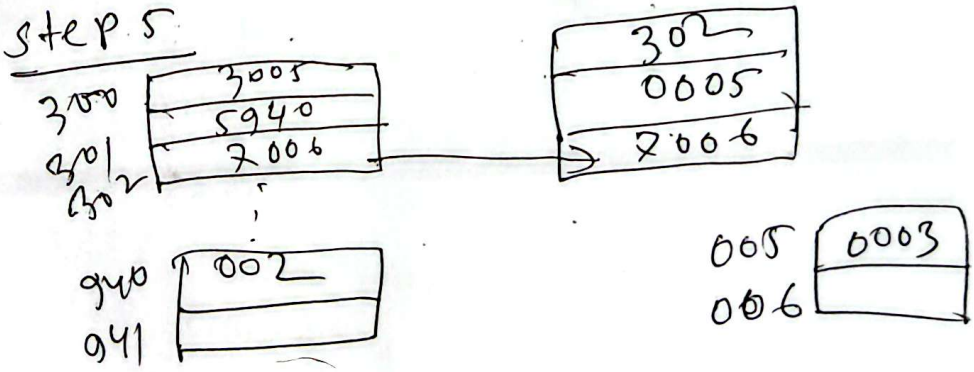
Step 3



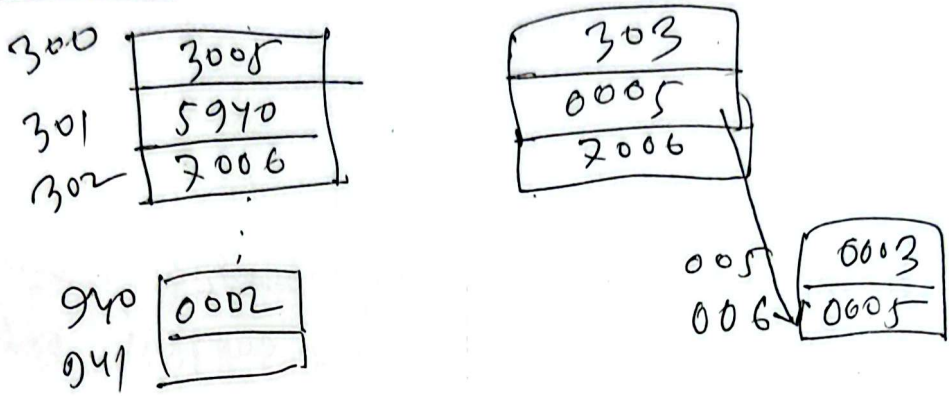
Step 4



step 5



Step 6



Final 18-19

1(b)

The contents of the file descriptor fd are: 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5

lseek(fd, 3, SEEK_SET);: This system call sets the file offset to byte 3 from the beginning of the file (SEEK_SET). So, the file pointer is moved to byte 3, which is the number 1 in the sequence.

read (fd, &buffer, 4); : This system call reads 4 bytes from the current file offset (byte 3) and store them in the buffer. The buffer will contain the bytes starting from the current offset : 1, 5, 9, 2

So, After the read has completed, the buffer will contain the sequence of bytes : 1, 5, 9, 2

2(a)

The cd [<dir>] command changes the current working directory to the specified directory, or to the user's home directory if no directory is given. It updates the shell's internal record of the current directory to resolve relative pathnames.

when you use cd to change to a directory named dir, you are not physically moving into that directory. Instead, the shell updates its internal record of the current working directory to the specified directory. The shell then uses this updated information to resolve relative pathnames.

Relative pathnames are pathnames that do not start with the root directory ("/") but instead are based on the current working directory.

2(b)

UNIX creates the illusion of permissions for users by implementing a robust file permission system based on three types of permissions: read (r), write (w), and execute (x). These permissions are assigned to three classes of users:

owner, group, and others.

The key mechanisms UNIX uses to create this illusion are:

User Identification: Each process running in UNIX is associated with a specific user, known as the "effective user ID" or "EUID." The EUID determines the permissions available to the process for accessing files and resources.

File ownership: Every file and directory in UNIX is associated with an owner and a group. The owner is the user who created the file, and the group is a collection of users who shares common access rights to the file.

File permission Bits: UNIX assigns specific permission bits to the owner group, and others, representing

the access rights for each class of users.

Access Control Lists (ACLs): In addition to the standard permissions, some UNIX systems support ACLs, which provide more fine-grained control over file access.

3(a)

Key word	system call	function call
privileged /kernel mode	Executes in kernel mode, accessing privileged system resources	Executes in user mode, cannot access kernel level resources
Trap instruction	Uses a trap instruction to switch from user mode to kernel mode	Doesn't use a trap instruct Control remains in user space
system dependent	system dependent; varies across operating system	system independent; behaves the same across systems.

3 (b)

Yes, setting the set-group-Id (SGID) bit on a directory has a significant effect on newly created files and subdirectories within that directory.

* when SGID is set on a directory, any new files or subdirectories created within it inherit the group ownership of the parent directory, rather than the group of the user who created them.

* This can be useful for collaboration scenarios where users need to share files within a specific group.

Benefits:

(1) Simplified group ownership management.

(ii) Controlled group access?

(iii) Streamlined collaboration

However, setting th SGID bit also has potential security implications:

* unintended access: If the directory group has wider access than intended, it can grant unintended permissions to the users in the group.

3(e)

(i) ps -ax | grep cron

Here:

ps -ax:

→ Lists all running processes on the system

→ -a: shows processes from all users.

→ -x : Includes processes not attached to a terminal.

→ | : A pipe that passes the output of the ps command as input to the next command

→ grep cron : Filters the output to show only lines containing the word "cron".

(2) ps -ax | tree processes.txt | more

→ ps -ax : lists all running processes.

→ | tree processes.txt : tree save the output to the file processes.txt, while also sending it to stdout.

→ | more : Displays the output one screen at a time, allowing you to scroll through the list of processes interactively.

4 (a)

=

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.

Free Blocks :

The file system maintains a data structure, often called the "free block bitmap" or "free block list", to track which disk blocks are available for use. This bitmap is a binary representation, where each bit corresponds to a block on the disk. A '0' bit indicates that the block is free, and a "1" bit indicates that the block is already allocated and in use. The kernel can quickly scan this bitmap to find

available free blocks.

Free Inodes:

Similarly, the file system uses a "free inode bitmap" or "free inode list" to track which inodes are free for allocation. Each inode represents a file or directory on the file system. It also uses bits to represent the availability of inodes. A "0" bit means the inode is free, and a "1" bit means it is in use.

The methods that are use to keep track of unused blocks and inodes are:

(i) Ext 2/3/4 : uses bit maps

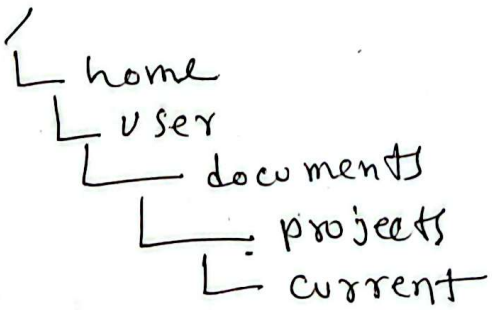
(ii) XFS : Uses extent-based allocation

for free blocks and a combination of bitmaps and free inode lists.

(iii) Btrfs: Employs a combination of extend-based allocation and free space tree for both blocks and inodes.

7(b).

(i) The pwd command displays the current working directory of the user in the shell



the pwd command will display the complete path:

/home/user/documents/projects/current.

(i) ~~Represent~~ Repetitive steps to compute the current directory using pwd:

The pwd command starts at the top of the directory tree and follows linked nodes through each subdirectory until it reaches the current directory.

(ii) knowing the top of the tree:

when using pwd, we know ~~when~~ we have reached the top of the directory tree (the root) when there are no more parent directories above the current directory being visited.

(iii) printing directory names in the correct order:

pwd uses a stack to store the

names of visited directories. As it traverses down the tree, it pushes the names onto the stack. when it reaches the current directory, it pops the names from the stack in reverse order, giving the correct path from the root to the current directory.

5(b)

In UNIX, when a new file system is introduced the system doesn't assign a new drive letter.

process of mounting in UNIX:

(i) A new storage device (e.g., USB, hard drive, Network drive) is connected.

(ii) It is formatted with a file system

(iii) the 'mount' command is used to attach the file system to a directory path

(iv) From then on, users access the contents of that device by navigating to the mount point.

Example:

Suppose a new disk is attached, and its file system is on /dev/sdb1

* create a mount point:

```
sudo mkdir /mnt/newdisk
```

* mount the file system

```
sudo mount /dev/sdb1 /mnt/newdisk
```

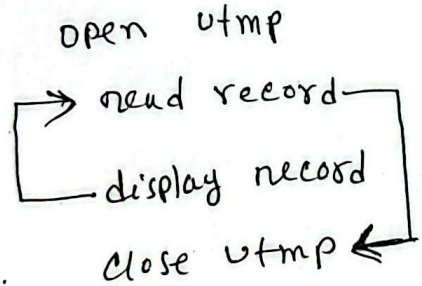
Now all file from the new disk can be accessed at /mnt/newdisk.

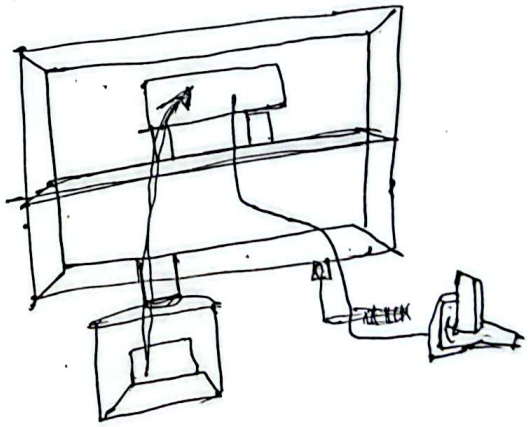
5(e) — See 11-19 - mid ques. (2) and ...

Data flow in "who" command

→ "who" reads structures from a file. The file contains one structure for each log-in session

→ The file is an array, so "who" must read the records and print out the information.





6(a)

The stat system call plays a crucial role in unix-like systems, returning valuable information about a file based on its name. Let's delve into how it works using the

directory, inode, and data model:

(1) Receiving the request: The application makes a stat system call, providing the file name. The kernel take the file name and initiates the search process.

(2) Navigating the directory tree: The kernel starts from the current working directory, traversing the directory structure to search for an entry with the matching filename.

(3) Reaching the Inode: Once the matching filename is found, the kernel retrieves the associated inode number.

(4) Filling the information: Using the retrieved inode, the kernel extracts various details about the file, including:

→ File size
→ Permission
→ Ownership
→ Modification time
→ Access time
→ And more.

(5) Delivering the response: The kernel fills the provided structure with the gathered information.

6 (b)

Designing a multithreaded program to count words in two files

(i) Create two threads, one for each file

(ii) Each thread reads its file and counts the number of words.

(iii) Then main thread waits for both threads to finish

TOPIC NAME :

4. The main thread then sums the word counts from both threads and prints the total

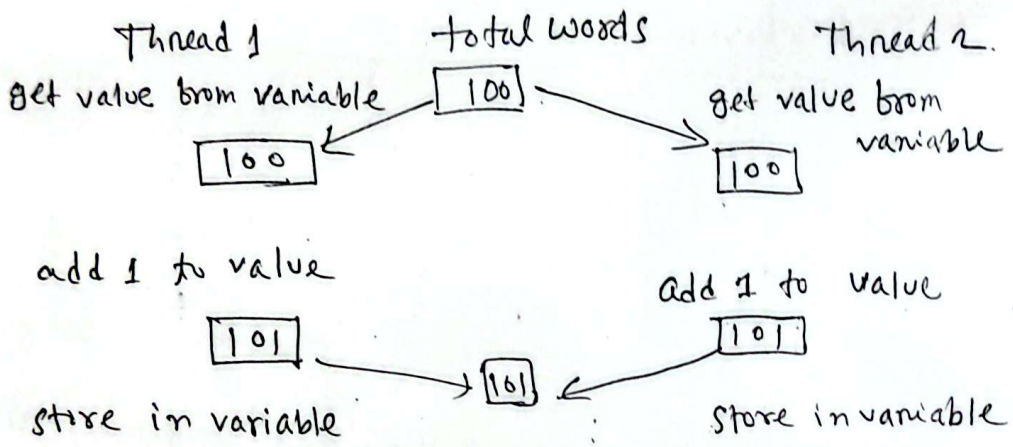


Fig: Two threads increment the same counter

☐ Use case scenarios for classification:

① Content Analysis system.

scenario: A system scans documents to determine their length or complexity.

use: The word count helps classify documents into:

i) short articles
ii) medium reports
iii) Long research papers.

2. Spam Vs. Genuine Email Detection

Scenario: An email scanner processes incoming messages from different file.

Use:

i) short, generic messages with few words might be spam.

ii) Longer, detailed messages are more likely genuine.

6(c)

1. pthread_create:

* creates a new thread of execution within the same process.

* Takes four arguments:
  1) pointer to a pthread_t variable to store the thread ID.
  2) Optional thread attributes.
  3) The function the new thread will execute
  4) An optional argument to pass to the new thread function.

* Return 0 on success, an error code otherwise.

2. pthread_join:

* waits for a specific thread to terminate.
* takes two arguments:
    1) The thread ID
    2) Optional pointer to a void pointer to store the return value of the joined thread.
* Returns 0 on success; an error code otherwise.
* Useful for ~~synchronization~~ synchronizing threads.

3. pthread_mutex_lock:

* Aquires a mutex lock
* Takes one argument:
    1) pointer to the pthread_mutex_t object representing the mutex.
* Blocks the calling thread if/ until the mutex is available
* Ensure exclusive access to shared resources and prevent race conditions.

4. pthread_mutex_unlock:

* Releases a priviously acquired mutex lock

* Takes one argument
  1) pointer to the pthread_mutex_t object representing the mutex.

* Signals other threads waiting on the mutex that it is now available.

* ~~EXOT~~ Ensures proper synchronization and avoids deadlocks

Final 17-18

1(a)

(1) users and Groups:

users:

→ ~~gndid~~ Individual accounts authorized to use a system.

→ Each users has a unique identifier (UID) and password.

→ Accessing files and resources based on their role.
→ Can belong to multiple groups.

Groups

→ collection of users sharing common access.
→ Identified by a ~~GROUP~~ Group ID

Key difference:

(1) users are individuals, groups are collections.
(11) users have ~~indidi~~ individual permissions, groups provide shared permissions.

(2) System call versus system libraries

System call:

→ Low-level functions that provide direct access to the operating system kernel.

→ Uses for tasks like managing files, memory, processes, and I/O.

→ Examples: read, write, open, fork.

System libraries:

→ collections of pre-written functions.

→ offer higher-level abstractions and easier-to-use interfaces for common tasks.

→ Generally safer than direct system calls.

→ Examples: printf, scanf, fopen, fgets.

key differences:

(i) system calls are closer to hardware, libraries offer abstractions.

(ii) system calls directly interact with kernel, libraries manages interactions.

1(e).

While both UNIX and C are used for system programming, the access criteria for I/O devices differ significantly between them.

UNIX:

(i) Device Files: In UNIX, I/O devices are represented as special files. Each device has a unique file name, for a serial port or for a disk drive.

(ii) Access permissions: Access to these device files is controlled by standard UNIX file permissions.

(iii) System calls: Once a device file is opened, programs use system calls to interact with the device.

(iv) Device drivers: The specific behavior of each device is handled by device drivers in the kernel.

(v) Kernel control: Device access in UNIX is ultimately controlled by the kernel, ensuring security and resource management.

C:

(i) No Direct Access: C itself doesn't provide direct access to I/O devices.

(ii) System interface: C programs rely on system libraries to interact with I/O devices.

(iii) Language limitations: C doesn't enforce access control or handle device-specific details.

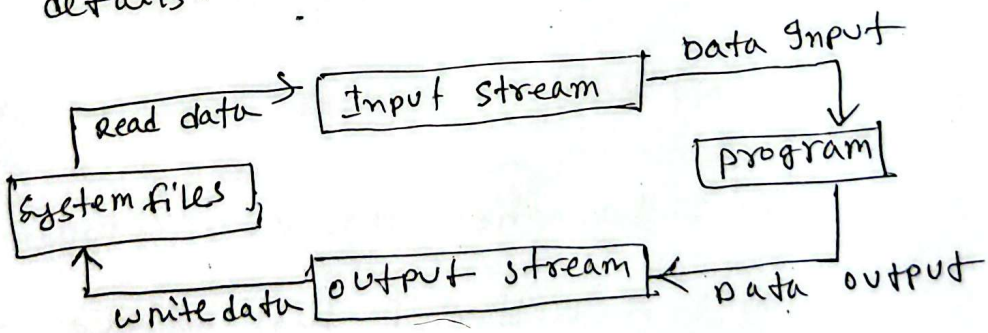


Fig: File handling in C

2 (a)

(i) ls mydir > outfile1 &

ls mydir: lists the contents of the directory mydir.

>: Redirects standard output to outfile1

&: Runs the command in the background.

Shell features used: Redirection and Background execution.

(ii) vi *

vi: Launches the vi editor

*: A wildcard that matches all files in the current directory.

Shell features used: Filename Expansion/ wildcard matching (globbing)

(iii) ls mydir > outfile1 & date > outfile2 &

Two separate commands are run in the background:

ls mydir > outfile1 &: Lists directory contents and saves output to outfile1 in the background.

date > outfile2 &: Gets current date/time and save it to outfile2 in the background.

Shell features used: Redirection and Background execution

2(b).

(1) When we read data after the end of a file using lseek, we'll typically encounter one of two scenarios:

① End-of-File (EOF): most systems will return an EOF indicator when you attempt to read beyond the actual data in the file. This means the read operation will fail, and you won't receive any data.

② Zero-filled bytes: Some systems might return a buffer filled with zeros.

In both cases, you won't be able to access the actual data beyond the end of the file.

(ii)

Writing data after the end of a file has different consequences depending on the system:

① File extension: most systems will extend the file size to accommodate the written data.

② Truncation: some systems might truncate the data to fit within the

existing file size. This means only the bytes that fit within the original file size will be written, and any excess data will be discarded.

③ Error: In rare case, the system might return an error if writing beyond the end of the file is not allowed.

3(a)

(i) The specific system calls involved will depend on the implementation of the login process, but they typically involve:

* open: To open the password file
* read: To read the user entry from the password file
* crypt: To compare the entered password with the stored hash.
* write: To update login records if successful.

(ii) A race condition arises when the timing of these calls interacts in a specific way:

(1) User 1 opens the password file, reads their entry, and enters their password.

(2) Before user 1 finishes comparing the password, the system switches to user 2.

(3) User 2 open the password file

(4) User 2 read their entry, enters their password, and successfully compares it.

(5) User 1 compares their password, potentially failing and potentially locking their account.

(iii) Several techniques can prevent this race condition.

① Locking: Acquire a lock on the password file before opening it. only one user can hold the lock at a time

② semaphores: Use a semaphore to manage access to the password file.

③ Atomic operations: Implement login as a single atomic operation, ensuring it completes entirely before another user can start.

3(b)

(i) Execute permission for a directory:

* for a directory, the ~~ext~~ "execute" permission allows users to access and enter the directory.

* In ~~context~~ the context of a directory, execution means the ability to seach

the directory and access files and subdirectories within it.

(ii) Execute Bit for a directory

* In the permission mode of a directory, represented by the pattern drwxr-xr-x, the 'execute' bit is denoted by the 'x' in the owner, group, and others

* For example, in drwxr-xr-x, the owner has execute permission because of the 'x' in the first position after 'd'.

Why it is useful:

* Navigation and access: Without "execute" permission, you wouldn't be able to access files within the directory.
* security: By carefully controlling "execute" permissions, you can restrict access to specific directories.

7(a)

(1) Files themselves actually don't directly 'know' which directory they reside in. This information is managed by the operating system, not stored within the file itself. Each file on a unix system has associated metadata, separate from the data it contains.

The metadata includes various ~~contents~~ details like filename, size, permissions, and important an inode number. The inode number acts as a unique identifier for the file and holds additional information, including pointers to the directory blocks where the file name is stored. These directory blocks essentially act as a index, linking the file name to its location within the directory structure

A file "knows" which directory it is in because its inode contains a reference to the directory where the file reside.

(2) The pwd command in unIx stands bor "print working directory". It bigure out where you are by accessing the inbor-mation stored by the operating system about the current directory.

Each process running on the system has its own associated current working directory, stored in a process control block (PCB). When you run the "pwd" command, the operating system retrieves the current directory information brom a special variable or data structure that keeps track ob the directory hierarchy. This information typically includes the bull path ob the current directory brom the root ob the bile system.

4(b)

Tracking Large files in UNIX file systems: facts, problems, and solution.

(i) Facts:

1) Inode management: Unix file systems utilize inodes to manage file metadata, including pointers to data blocks.

2) Large File structure: Large file in Unix systems can have numerous data blocks scattered across the disk.

3) Addressing methods: Direct Addressing and Indirect addressing.

(ii) prosible problems:

① Performance: Traversing multiple levels of indirect pointers can be slower than direct addressing, impacting perbormance

② Fragmentation: scattered data blocks across the disk can result in wasted space and slower access times due to fragmentation.

③ metadata overhead: managing large number of data blocks may require additional inode space, potentially affecting performance.

Solutions:

① 64 bit Inodes: modern systems often utilize 64-bit inodes, significantly increasing the direct addressing limit and supporting larger files more efficiently

② File system optimizations: contain file system employ mechanisms such as extents and preallocation to minimize fragmentation and optimize handling of large file.

5(a)

(i) Attributes of a connection:

→ For disk files: Permissions (read, write, execute), ownership, size, timestamps.

→ For device files: Settings specific to the device, like baud rate, parity.

(ii) Examination of current attributes:

→ For disk files: use command like 'ls -l' or 'stat'

→ For device files: use tool like 'ioctl' system calls or 'lspci' / 'lsusb' commands.

(iii) changing current attributes:

→ for disk files: modify with commands like 'chmod', 'chown', and 'touch'.

→ For device files: use 'fcntl' or 'ioctl'

system calls to interact with the device driver and make adjustments.

5 (b)

Four groups of tty driver settings:

① Input Flags

purpose: control input processing and character interpretation.

Bits:

(i) IGNBRK: Ignore break condition on input

(ii) IGNCR: Ignore carriage return.

② output Flags

purpose: control output processing and character formatting

Bits:

(i) OPOST: Enable output processing

(ii) ONLCR: Translate newline to carriage return and line feed.

③ Control Flags:

purpose: control basic terminal settings like baud rate, parity, and data flow.

Bits:

(i) CS8: 8-bit data characters

(ii) CREAD: Enable receiver

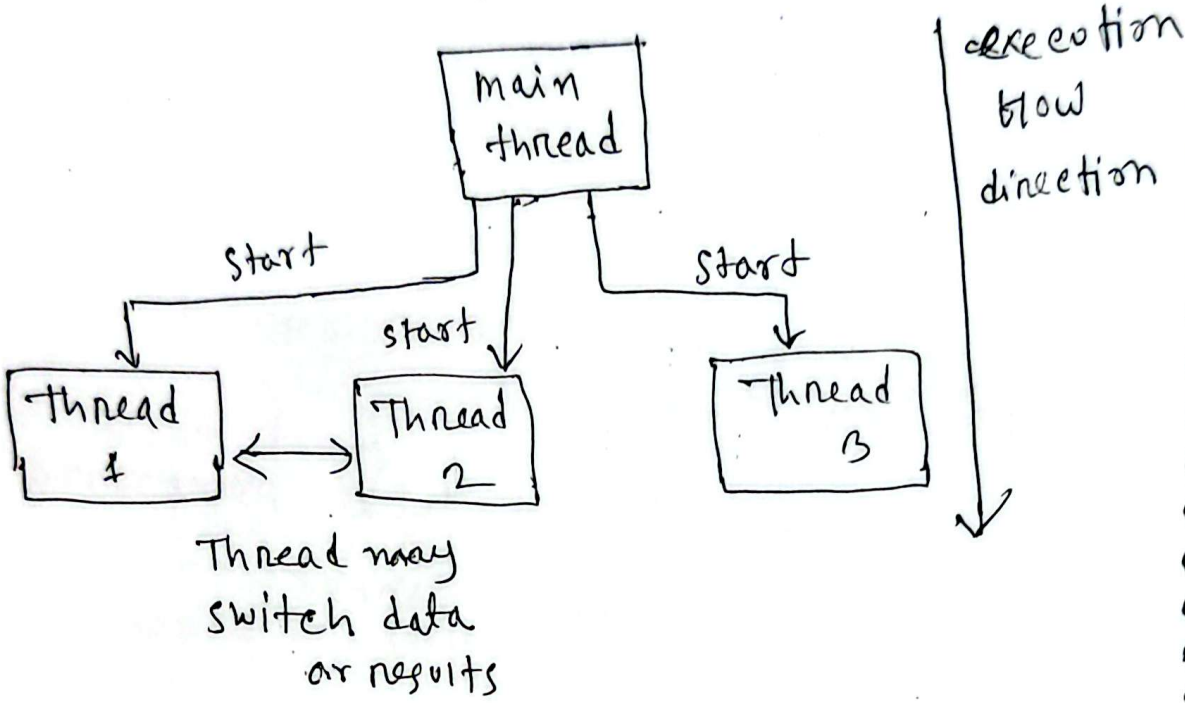
④ Local Flags

purpose: control local processing like echo and canonical input.

Bits:

(i) ICANON: Enable canonical input mode

(ii) ECHO: Echo typed characters to the screen.



Examples:

① web browser: Different threads download web pages, handle user interactions, and render content, keeping the browser responsive even while loading multiple resources.

② Game engine: multiple threads handle physics simulations, AI calculations, graphics rendering, and sound effects, creating a dynamic and immersive gaming experience.

TOPIC NAME : System programming Lec-09

DAY :

TIME : DATE : / /

* We can determine names and properties of files by ls command.

ls sorts the names in alphabetical order.

```
$ ls -l
```

(output)

```
drwxr-xr-x  2  root  root  4096  Mar 21  2002  bin
```

permission — links — owner — group — size — date — filename

ls does two things

→ Lists the contents of directories

→ Displays information about file.

① How to list the contents of a directory ?

= ls /home/user/Documents

Directory name

② How to obtain and display properties of a file.

= ls -l filename

TOPIC NAME : DAY :

TIME : DATE :

③ How to determine if a name refers to a file or a directory?

= ls -l myfile.txt

ls -l myfolder

↳ if first character '-', its a regular file
↳ " " " 'd', " " directory

-rw-r--r-- ← file

drwxr-xr-x ← directory

GOOD LUCK