MODULE VI

- **Text Editors**
  - **Overview of Editing**
  - **User Interface**
  - **Editor Structure.**
- **Debuggers**
  - **Debugging Functions and Capabilities**
  - **Relationship with other parts of the system**
  - **Debugging Methods**
    - **By Induction**
    - **Deduction**
    - **Backtracking**
- **Device drivers**
  - **Anatomy of a device driver**
  - **Character and block device drivers**
  - **General design of device drivers**

- **Text Editors**
  - Text editors are software programs that enable the user to create and edit text files.
  - Example: Notepad, Wordpad, MS Word etc.
  - A document may include objects such as Computer programs, Equations, Tables, Diagrams, Line art, Photographs etc.
  - There are two types of editors:
    - **Manuscript-oriented editor**: It is associated with characters, words, lines, sentences and paragraphs
    - **Program orient editors**: These are associated with identifiers, keywords etc.

  - **Overview of an Editing Process**
    - The document-editing process is an interactive user-computer dialogue designed to accomplish four tasks
      1. Select the part of the target document to be viewed and manipulated
      2. Determine how to format this view on-line and how to display it.
      3. Specify and execute operations that modify the target document.
      4. Update the view appropriately
    - Editing Steps:
      - TRAVELING
        - It involves traveling through the document to locate the area of interest such as next screenful , bottom and find pattern.
      - FILTERING
        - Selection of what is to be viewed and manipulated is controlled by filtering.
        - Filtering extracts the relevant subset of the target document at the point of interest such as next screenful of text or next statement.
      - FORMATING
        - Determines how the result of filtering will be seen on adisplay screen.
      - EDITING PHASE
        - Editing phase involves – insert, delete, replace, move, copy, cut, paste, etc...

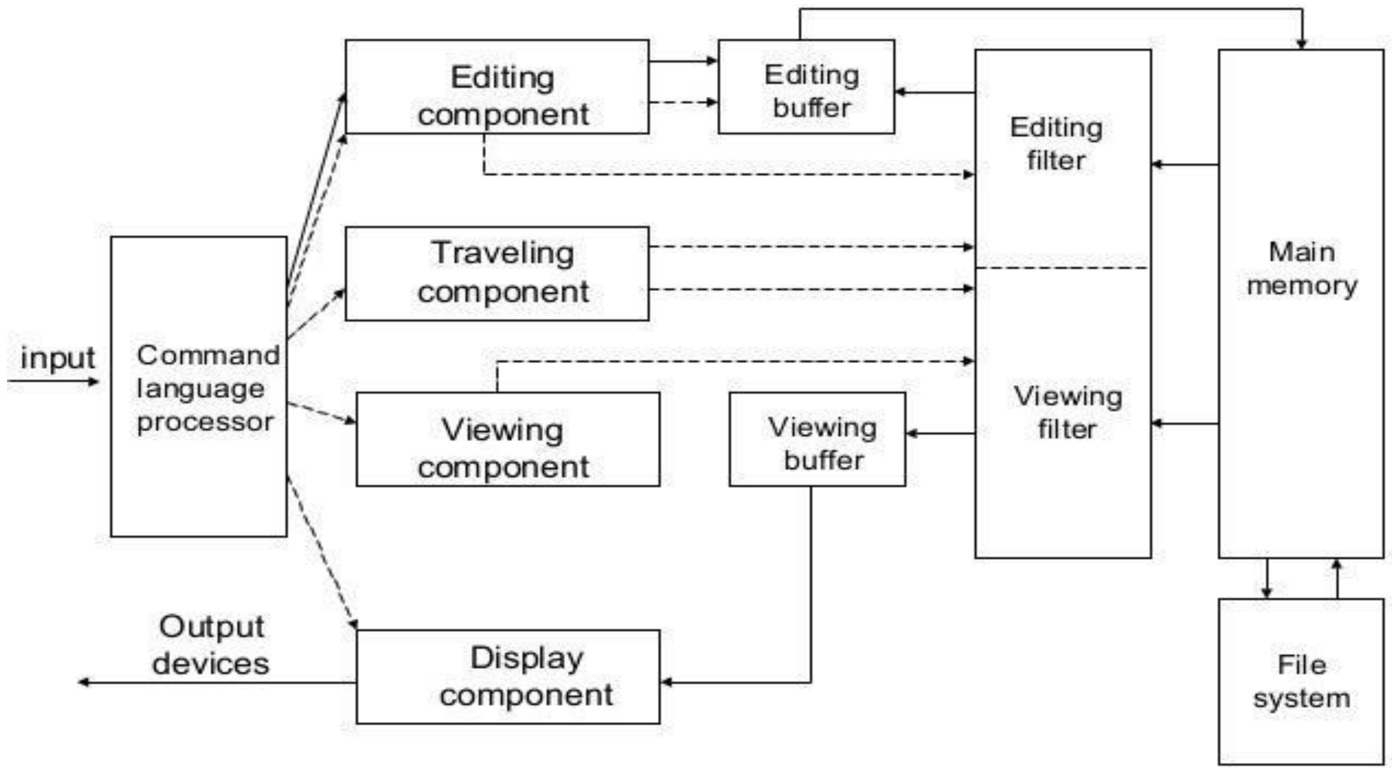
- **User Interface of a Text Editor**
  - The user of an interactive editor is presented with a conceptual model of the editing system.
  - This model is an abstract framework on which the editor and the world on which it operates are based.
  - Some of the early line editors simulated the world of keypunch. These editors allowed operations on numbered sequences of 80 character card-image lines, either with in a single line or on an integral number of lines.
  - Some modern screen editors define a world in which a document is represented as a quarter-plane of text lines, unbounded both down and to the right. The user sees through a cutout, only a rectangular subset of this plane on a multiline display terminal.
  - The user interface of a text editor is concerned with
    - Input Devices
    - Output Devices
    - Interaction Languages

  - **Input Devices**
    - Input devices are used to enter elements of the text being edited, to enter commands, and designate editable elements.
    - Input devices are categorized as:
      - **Text devices/String Devices**
        - These are typically typewriter like keyboards on which user presses and releases keys, sending unique code for each key.
        - Virtually all computer keyboards are of QWERTY type.
      - **Button devices/Choice Devices**
        - Special function keys on alpha numeric keyboard
        - It generates an interrupt or set a system flag, usually causing an invocation of an associated application program action.
      - **Locator devices**
        - These are two dimensional analog to digital converters that position a cursor symbol on the screen by observing the users movement of the device.
        - Ex: Mouse , Joysticks, Touch screens, Data tablets etc
      - **Voice input device**
        - It translates spoken words to their textual equivalents.

  - **Output Devices**
    - Output devices let the user view the elements being edited and the result of the editing operations.
    - The first output devices were teletypewriters, that generates output on paper
    - Next is Cathode ray tube (CRT), which uses CRT screen for display.
    - The modern professional workstations are based on personal computers with high resolution displays.

- **Interaction Languages**
  - The interaction language could be,
    - **Typing oriented/ Text command oriented**
      - Oldest editors are typing oriented
      - The user communicates with the editor by typing text strings.
      - These strings are sent to the editor and are usually echoed to the output device.
      - The users should remember the exact form of all commands.
      - In order to avoid that, we can use **function key interfaces**.
      - Each command is associated with marked key on keyboard .This eliminate much typing
        - Eg: Insert Key, Shift Key, Control key etc.
      - Disadvantage: Have too many unique keys which will result in a unwieldly keyboard.
    - **Menu-oriented user interface**
      - Menu-oriented user interface has menu with a multiple choice set of text strings or icons.
      - Menus can be turned on or off.
      - User performs actions by selecting items from the menu
      - **Drawback:**
        - When user requires many possible actions and several choices to complete an action, this is not a suitable method.
        - Display area for text is limited.

- **Text Editor Structure**



Typical editor structure

- **Command Language processor**
  - The command language processor accepts input from the user's device, analyse the tokens and syntactic structure of the commands.
  - It may invoke semantic routines.
  - The semantic routines may involve traveling, editing, viewing and display functions.
- **Editing Component**
  - Editing operations are always specified by the user.
  - The editing component is a collection of modules dealing with the editing tasks.
  - The start of the area to be edited is determined by the **current editing pointer** maintained by the editing component.
  - The current editing pointer can be set or reset explicitly by the user with the travelling commands or implicitly by the system as a side effect of previous editing operation.
- **Editing Filter and Editing Buffer**
  - The editing commands invoke the editing filter.
  - This component filters the document to generate a new editing buffer based on the current editing pointer as well as the editing filter parameters.
  - These parameters are specified by both the user and the system.
  - They provide information such as the range of text that can be affected by an operation.
- **Viewing Component**
  - It is a collection of modules responsible for determining the next view.
  - In viewing a document, the start of the area to be viewed is determined by the current viewing pointer.
  - The pointer can be set or reset explicitly by the user with a traveling command or implicitly by the system as a result of the previous editing operation.
  - The pointer is maintained by the viewing component.
  - When the display needs to be updated, the viewing component invokes the viewing filter.
- **Viewing Filter and Viewing Buffer**
  - When the display needs to be updated, the viewing component invokes viewing filter.
  - Viewing component filters the document to generate a new viewing buffer based on the current viewing pointer as well as on the viewing filter parameters.
  - Eg: The user of a certain editor might travel to line 75, and after viewing it, decide to change all occurrences of "abc" by "pqr" in line 1 through 50 are then filtered from the document to become the editing buffer. Successive substitutions take place in this editing buffer, without corresponding updates of the view.
  - The parameters are specified by both the user and the system.
  - In line editors, the viewing buffer may contain the current line.
  - In screen editors, this may contain the rectangular cut-out of text.
  - The buffer is then passed to the display component of editor.
- **Traveling Component**
  - It performs the setting of the current editing and viewing pointers, and thus determines the point at which the viewing and editing begins.

    - **Display Component**
        - Display operations are always specified implicitly by the other three categories of operations : editing, traveling and viewing.
        - It takes the idealized view from the viewing component and maps it to a physical output device in the most efficient manner
        - It produces a display by mapping the viewing buffer to a rectangular subset of the screen, usually called a screen.
    - **Main Memory and File System**
        - Loading an entire document into main memory be infeasible.
        - So load only a part of the document into main memory.

- **<u>Debugger</u>**
    - A debugger is a computer program that is used to test and debug other programs.
    - Debugging is the activity of locating and correcting errors.
    - It can start once a failure has been detected.
    - An interactive debugging system provides programmers with facilities that aid in testing and debugging a program.
    - Debugging is a two step process that begins when you find an error as a result of a successful test case
        - Determine the exact nature and location of the suspected error within the program.
        - Fixing the error.
    - Difference between Testing and Debugging
        - Testing: process of executing a program with the aim of finding errors or bugs.
        - Debugging: correcting these errors found during testing is debugging.
    - **Types of bugs:**
        - Compile time: Usually caught with compiler
            - Eg: Syntax, Spelling, type mismatch etc.
        - Design time:
            - Flawed algorithm
            - Produce incorrect output.
        - Program Logic: Produce incorrect output.
        - Memory nonsense:
            - Eg: null pointers, array bounds, bad types etc.
            - It is a run time exception
        - Interface errors between modules, threads, programs: It is a run time exception
        - Off normal conditions:
            - Failure of some part of software of underlying machinery
            - It causes incomplete functionality.
        - Deadlocks: Multiple processes fighting for a resource.
    - **Debugging Functions and Capabilities**
        - A set of unit test functions
            - One group of such functions deals with execution sequencing(observation and control of the flow of program execution).
                - Eg:
                    - The program may be halted after a fixed number of instructions are executed.

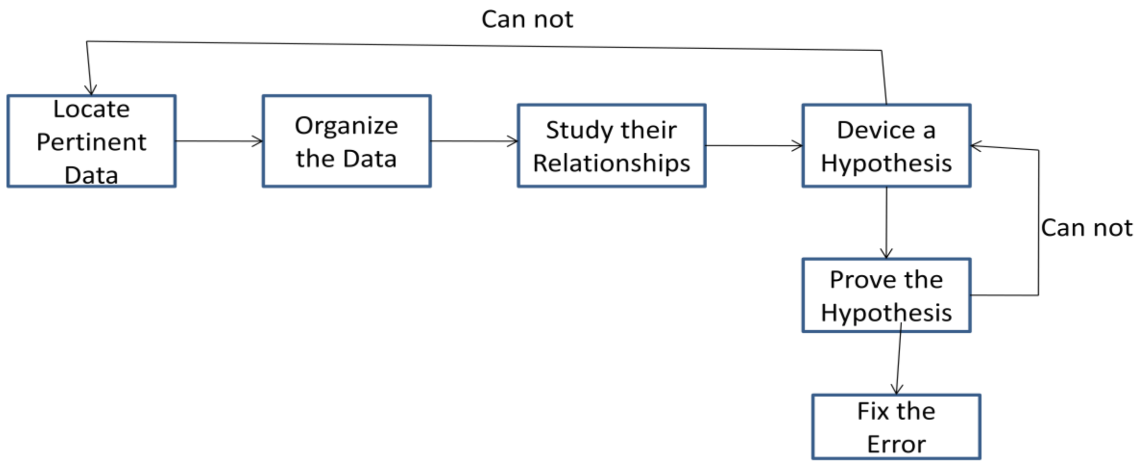
        - Breakpoints: Execution to be suspended when a specific point in the program is reached.
        - Programmer can define conditional expressions that are continually evaluated during the debugging session. Program execution is suspended when any of these conditions becomes true.
      - After execution is suspended, other debugging commands can be used to analyze the process of program and to diagnose errors detected.
- Debugging system should provide the functions such as Tracing and Traceback
  - Tracing:
    - Used to track the flow of execution logic and data modifications.
    - The control flow can be traced at different levels of detail: Procedure, branch, individual instruction, and so on.
    - It is also based on conditional expressions.
  - Traceback:
    - Shows the path by which the current statement was reached.
    - It can also show which statements have modified a given variable or parameter.
- Debugging systems should have a good program display capability.
  - It is possible to display the program being debugged, with statement numbers.
  - Should be able to control the level at which display occurs
    - The program may be displayed as it was originally written, after macro expansion, and so on.
  - The system should save all the debugging specifications (break points, display modes, etc.) across each compilation, so the programmer does not need to reissue all these debugging commands.
  - It should be possible to display or modify the content of any of the variable and constants in the program, and then resume execution.
- Capability of handling multilingual situations
  - Debugging systems can be divided in to two
    - Language Dependent: Debugging system designed for a particular programming language.
    - Language Independent:
      - Most user environment involves the use of several different programming environments.
      - It is a single debugging system designed for several programming language.
      - When the debugger receives control, the execution of the program being debugged is temporarily suspended. The debugger must then be able to determine the language in which the program is written and set its context accordingly.
      - The debugger should able to switch its context when a program written in one language calls a program written in a different language.

      - Assignment statements that change the values of variables during debugging should be processed according to the syntax and semantics of the source programming language.

                - COBOL: MOVE 3.5 TO A
                - FORTRAN: A = 3.5
            - Conditional expressions should use the notion of the source language
                - Eg: A be not equal to B
                    - COBOL: IF A NOT EQUAL TO B
                    - FORTRAN: IF(A .NE. B)
    - Able to deal with optimized code
        - Optimization involves the rearrangement of segments of code in the program.
        - Eg:
            - Eliminate loop invariant expressions
            - Separate loops can be combined in to single loop
            - Redundant expressions may be eliminated
            - Blocks of code may be rearranged to eliminate unnecessary branch instructions
        - All these optimization create problems for the debugger and should be handled carefully.
    - Able to handle storage properly
        - The compiler normally assigns a home location in main memory to each variable.
        - Variable value may be temporarily held in registers at various times to improve speed of access.
        - Statements referring these variables use the value stored in the register, instead of taking the variable value from its home location.
        - If the user changes the value of the variable in its home location while debugging, the modified value might not be used by the program.
        - Solution: Variable may be permanently assigned to a register. There may be no home location at all.

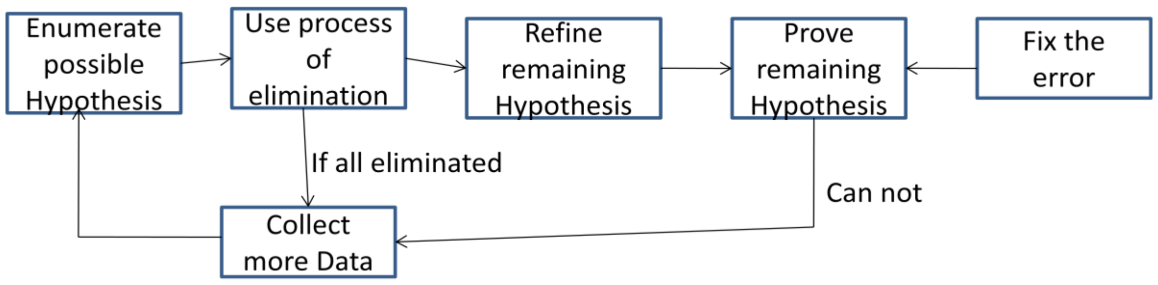
- **Relationship with other parts of the system**
    - Debugger must always be available.
        - It must appear to be a part of the run time environment and an integral part of the system.
        - When an error is discovered, immediate debugging must be possible.
        - The debugger must communicate and cooperate with other operating system components.
    - Debugging is more important in production time than it is at application development time.
        - When an application fails during a production run, work dependent on that application stops.
    - The debugger must be consistent with the security and integrity components of the system.
    - The debugger must coordinate its activities with those of existing and future language compilers and interpreters.
        - The debugging facilities in existing languages will continue to exist and be maintained.

- **Debugging Methods**
  - **Debugging By Induction(Reasoning strategy/thoughtful strategy)**
    - Many errors can be found by using a disciplined thought process without the debugger ever going near the computer.
    - One such thought process is induction which can figure out solution to a problem.
    - Here we start with our own experience and then generalize a rule
    - This strategy assumes that once the symptoms of the errors are identified, and the relationships between them are established, the errors can be easily detected by just looking at the symptoms and the relationships.



    - Locate the pertinent data:
      - Enumerate all you know about the problem.
      - Collect available information, collect symptoms, error occurrence, error conditions, effect of the failure etc.
    - Organize the data:
      - Structure the collected data and determine success conditions and the differences.
    - Device a Hypothesis:
      - Derive one or more hypothesis.
      - If multiple theories are possible, then select the most probable one first.
    - Prove the Hypothesis:
      - Hypothesis should completely explain the existence of clues.
    - Fix the error:
      - Implement the appropriate fixes based on the hypothesis.
      - Verify the new code by rerunning it for the failure case and make sure the fix corrects the error condition.

  - **Debugging By Deduction**
    - The process of deduction proceeds from some general theories or premises, using the processes of elimination and refinement, to arrive at a conclusion.

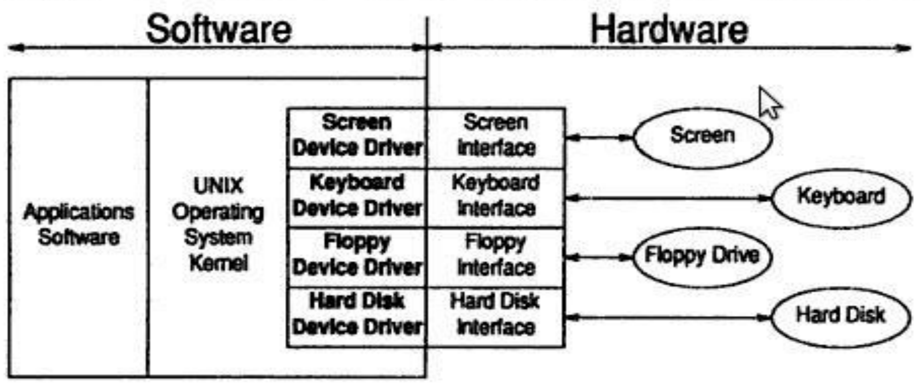

Enumerate possible Hypothesis
Use process of elimination
Refine remaining Hypothesis
Prove remaining Hypothesis
Fix the error
If all eliminated
Can not
Collect more Data

- Enumerate the possible Hypothesis
  - Develop a list of all hypotheses for the failure case.
  - They are theories to help you to structure and analyse the available data.
- Use Data to eliminate possible hypothesis
  - Carefully examine all of the data, particularly looking for contradiction, and try to eliminate all but one of the possible hypothesis.
- Refine remaining Hypothesis (Expand the remaining hypothesis)
  - The possible hypothesis at this point might be correct, but it is unlikely to be specific enough to pinpoint the error.
  - Hence the next step is to use the available clues to refine the theory.
- Prove/Disprove the hypothesis
  - Prove the reasonableness of the hypothesis

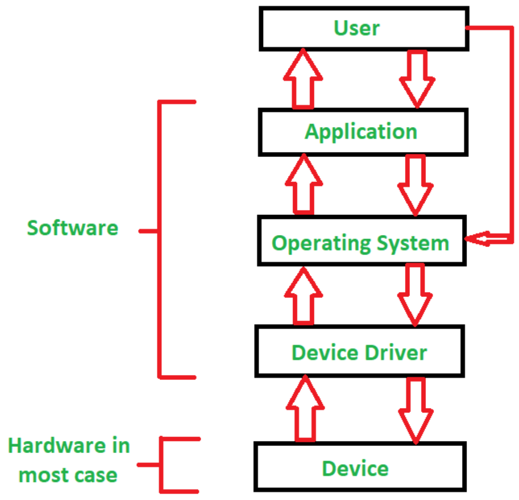
- **Debugging By Backtracking**
  - Backtrack the incorrect result through the logic of the program until you find the point where the logic went off track. That means, find the point where the program gives incorrect result.
  - For large programs, such analysis at the statement level would be too tedious to perform. What we seek is to first narrow the user's focus to a small region of the program that is likely to contain an error, and then do statement level analysis within this region.
  - The program is viewed as a sequence of logical blocks instead of individual statements.
  - One need only to think backwards at the program block level instead of the statement level
  - To perform such backward analysis using conventional interactive debuggers, one would first set a breakpoint just before the last logical block. If the program state is found to be correct at this point, it would imply that the error occurred within the last block. Otherwise another breakpoint would be set before the second-last block, and the program reexecuted. If the program state is found to be correct at that point, then we may conclude that the error resides within that second-last block. Otherwise this process of setting breakpoints in backward order and reexecuting the program continues until the erroneous block is discovered.
  - If N is the number of blocks in the program, then clearly this method of setting breakpoints successively in backwards order and reexecuting the program every time leads to an O(N^2) cost for execution and require only O(N) block executions.

- **Device Drivers**
  - A Device Driver is glue between an OS and its I/O devices.
  - Device drivers communicate directly with devices. They act as translators converting generic requests received from the operating system into commands that specific peripheral controllers can under-stand.
  - Relationship between an application software, OS and device driver is illustrated below.

FIGURE 1-1



  - The application software makes system calls to the OS requesting services. The OS analyses these requests and when necessary issues requests to the appropriate device driver. Device drivers in turn analyses the request from OS and when necessary issues command to the hardware interface to perform the operations needed to service the request

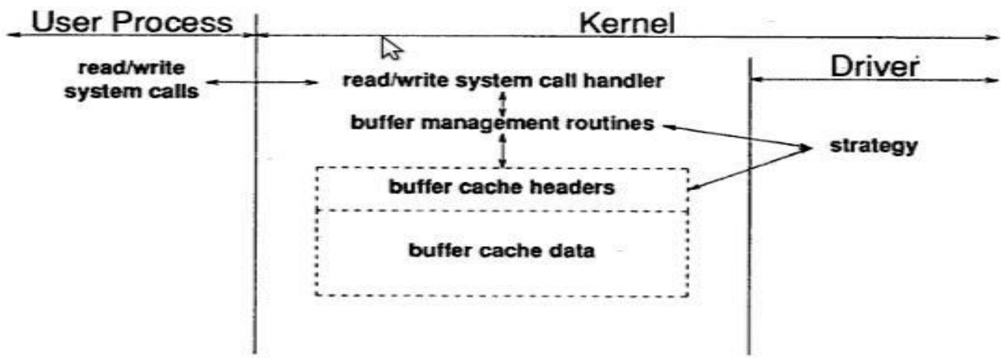


  - Device driver simplifies the design of OS. Without the device drivers the operating system would be responsible for talking directly to the hardware. This would require the OS designer to include support for all the devices that the users might want to connect to the computer. It would also mean adding support to a new device would require modifying the OS. By separating device driver's functions from OS, the designer can concentrate on the issues related to the operation of the system as a whole. Thus device driver designer does not have to worry about the general issues related to I/O management which is handled by the OS. The device driver writer can connect any I/O device to the system without having to modify the OS. Device drivers thus provide the OS with a standard interface to non-standard I/O devices.

- **Device Characteristics**
  - Identifier: A universally unique identifier that is intrinsic to the device.
  - Operational State: Indicates whether the device is mounted or unmounted.
  - LUN: Logical Unit Number (LUN) within the SCSI target. The LUN number is provided by the storage system. If a target has only one LUN, the LUN number is always zero (0).
  - Type: Type of device, for example, disk or CD-ROM.
  - Drive Type: Information about whether the device is a solid-state drive (SSD) or a regular non-SSD hard drive.
  - Capacity: Total capacity of the storage device.
- **Design Issues of Device Driver**
  - OS / Driver Communication
    - Deals with exchange information (i.e command and data) between device driver and OS
    - Also includes support functions that the kernel provides for device driver
  - Driver / Hardware Communication
    - Deals with exchange information (ie command and data) between device driver and the device it controls.
    - Deals with how the software talks to the hardware
    - Deals with how the hardware talks to the software
  - Driver Opertaions
    - Interpreting commands received from OS
    - Scheduling the requests
    - Managing read and write (ie data transfer across OS and hardware).
    - Accepting and processing H/W interrupts
    - Maintain integrity of kernel's and driver's data structures
- **Types of device driver and their anatomy**
  - Based on the differences in the way they communicate with UNIX OS the device drivers are classified into
    - Block Drivers
    - Character Drivers
    - Terminal Drivers
    - Stream Drivers

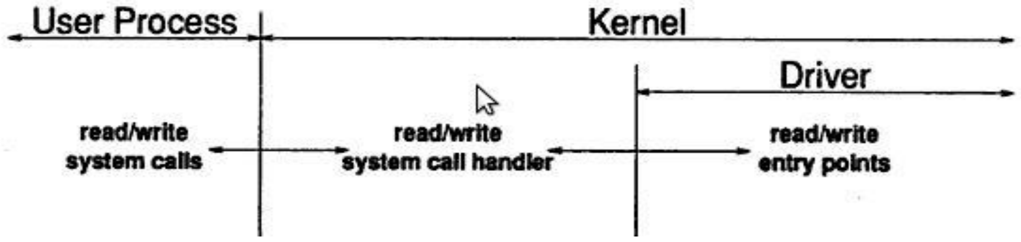
  - Block Drivers
    - It communicates with OS through a collection of fixed-sized buffers.
    - Example: Disk drives
    - The OS manages a cache of these buffers and attempts to satisfy user requests for data by accessing buffers in the cache. The driver is invoked only when the requested data is not in the cache or when buffers in the cache have been changed and must be written out.

FIGURE 1-2



- Character Drivers
    - It can handle I/O requests of arbitrary size and can be used to support almost any type of device. Mostly used devices are Printers. Usually character drivers are used for devices that either deal with data a byte at a time or work best with data in chunks smaller or larger than the fixed sized buffers used by block drivers (eg tape drives). The only relevant difference between a char device and a regular file is that you can always move back and forth in the regular file, whereas most char devices are just data channels, which you can only access sequentially.

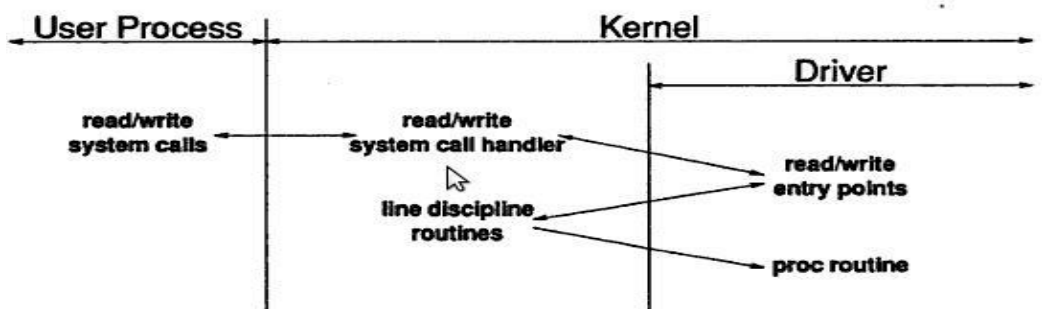
FIGURE 1-3



    - Major difference between block and character device driver is that the user processes interact with block drivers indirectly through the buffer cache but their relationship with character drivers is very direct. From figure we can see that the I/O request is passed essentially unchanged from the process to the driver and the driver is responsible for transferring the data directly to find from the user process's memory.

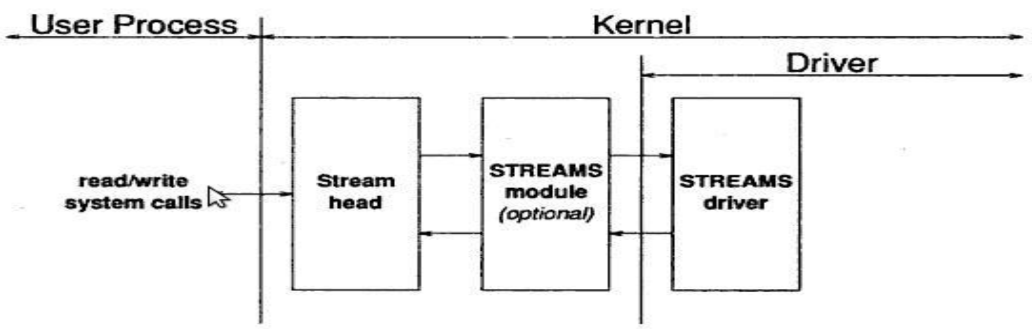
- Terminal Drivers
    - Same as character drivers but specialized to deal with communication terminals that connect users to OS. Terminal drivers are not only responsible for shipping data to and from user terminals but also for handling line editing, tab expansion and many other terminal functions that are part of the standard UNIX terminal interface. Because of this additional processing that terminal drivers must perform it is useful to consider terminal drivers as a separate type of driver.

FIGURE 1-4



- Stream Drivers(Network Device Drivers)
  - It can handle high speed communication devices such as networking adapters that deal with unusual sized chunks of data , that need to handle protocol. Network Devices use stream drivers. Network devices support layered protocol. If character drivers where used for network devices, it would require that each layer of the protocol be implemented within the single driver. This causes lack of modularity and re-usability and thus reduces the efficiency of the system. As a result stream device drivers where developed which is an extension of character drivers. It send and receive packets. They do not know about individual connections. Network drivers have unique names (e.g., eth0). They are not present in the file system. They support protocols and streams related to packet transmission (i.e., there is no read and write). Network devices are accessed via the BSD socket interface and the networking subsytems.

FIGURE 1-5



Character device driver	Block device driver
A Character ('c') Device is one with which the Driver communicates by sending and receiving single characters (bytes, octets).	A Block ('b') Device is one with which the Driver communicates by sending entire blocks of data
Sequential stream of data access.	Random data access nature
Handles comparatively lesser amount of data in the form of byte stream	Handles larger amount of data in the form of blocks.
Faster mode data transfer rate as they don't use the block access method.	Slower data transfer rate as chunks of data involved.
Operates directly with the user process of data transfer.	Designed to operate indirectly through buffer cache.
Abbreviated as ' cdev '. Eg: Serial ports, Parallel ports, Sounds	Abbreviated as ' blkdev Eg: Hard disks, USB, Cameras, Disk On Key

Previous Year University Questions

1. Explain the overview of editing process
2. Write notes on the user interface of a text editor
3. Explain the structure of a text editor with the help of a diagram
4. Explain the different types of Text Editors and User Interface
5. List out the main four tasks associated with the Document Editing Process
6. With a neat diagram show the relationship between viewing and editing buffer
7. What is a Debugger?
8. Write notes on the debugging functions and capabilities of an interactive debugging system
9. List out the criteria that should be met by the user interface of an efficient debugging system
10. Explain the different debugging methods in detail
11. Write down the situations where debugging by induction, deduction and backtracking are used, explaining each process
12. A new hardware device is plugged into a system. Which is the appropriate system software needed for the proper working of the new hardware? Give its functionalities and general architecture
13. Explain the general design of device driver
14. Explain the general design and anatomy of a device driver with the help of diagrams
15. What are the functions of device drivers?
16. Differentiate between character and block device drivers
17. What is a Device Driver? What are the major design issues of a Device Driver?