MODULE 5

EDITORS AND DEBUGGING SYSTEMS

An Interactive text editor has become an important part of almost any computing environment. Text editor acts as a primary interface to the computer for all type of “knowledge workers” as they compose, organize, study, and manipulate computer-based information.

An interactive debugging system provides programmers with facilities that aid in testing and debugging of programs. Many such systems are available during these days. Our discussion is broad in scope, giving the overview of interactive debugging systems – not specific to any particular existing system.

5.1 Text Editors:

- An Interactive text editor has become an important part of almost any computing environment. Text editor acts as a primary interface to the computer for all type of “knowledge workers” as they compose, organize, study, and manipulate computer- based information.
- A text editor allows you to edit a text file (create, modify etc…). For example the Interactive text editors on Windows OS - Notepad, WordPad, Microsoft Word, and text editors on UNIX OS - vi, emacs , jed, pico.
- Normally, the common editing features associated with text editors are, Moving the cursor, Deleting, Replacing, Pasting, Searching, Searching and replacing, Saving and loading, and, Miscellaneous(e.g. quitting).

5.1.1 Overview of the editing process

- An interactive editor is a computer program that allows a user to create and revise a target document. Document includes objects such as computer diagrams, text, equations tables, diagrams, line art, and photographs. In text editors, character strings are the primary elements of the target text.

- Document-editing process in an interactive user-computer dialogue has 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

- The above task involves traveling, filtering and formatting.
    - Traveling – To locate the area of interest. This is done by operations such as next screenful, bottom and find pattern.
    - Filtering- extracts the relevant subset of the target document.
    - Formatting- How the result of filtering will be seen as a visible representation(the view) on a display screen.
    - Editing- The target document is created or altered with a set of operations such as insert, delete, replace, move and copy.
- There are two types of editors. Manuscript-oriented editor and program oriented editors. Manuscript-oriented editor is associated with characters, words, lines, sentences and paragraphs. Program-oriented editors are associated with identifiers, keywords, statements. User wish – what he wants – formatted.
- So in overall the user might travel to the end of the document. A screenful of text would be filtered, this segment would be formatted, and the view would be displayed on an output device. The user could then edit the view.

5.1.2User Interface:

- Conceptual model of the editing system provides an easily understood abstraction of the target document and its elements. For example, Line editors – simulated the world of the key punch – 80 characters, single line or an integral number of lines, Screen editors – Document is represented as a quarter-plane of text lines, unbounded both down and to the right.
- The user interface is concerned with, the input devices, the output devices and, the interaction language. The input devices are used to enter elements of text being edited, to enter commands. The output devices, lets the user view the elements being edited and the

results of the editing operations and, the interaction language provides communication with the editor.

- **Input Devices** are divided into three categories:

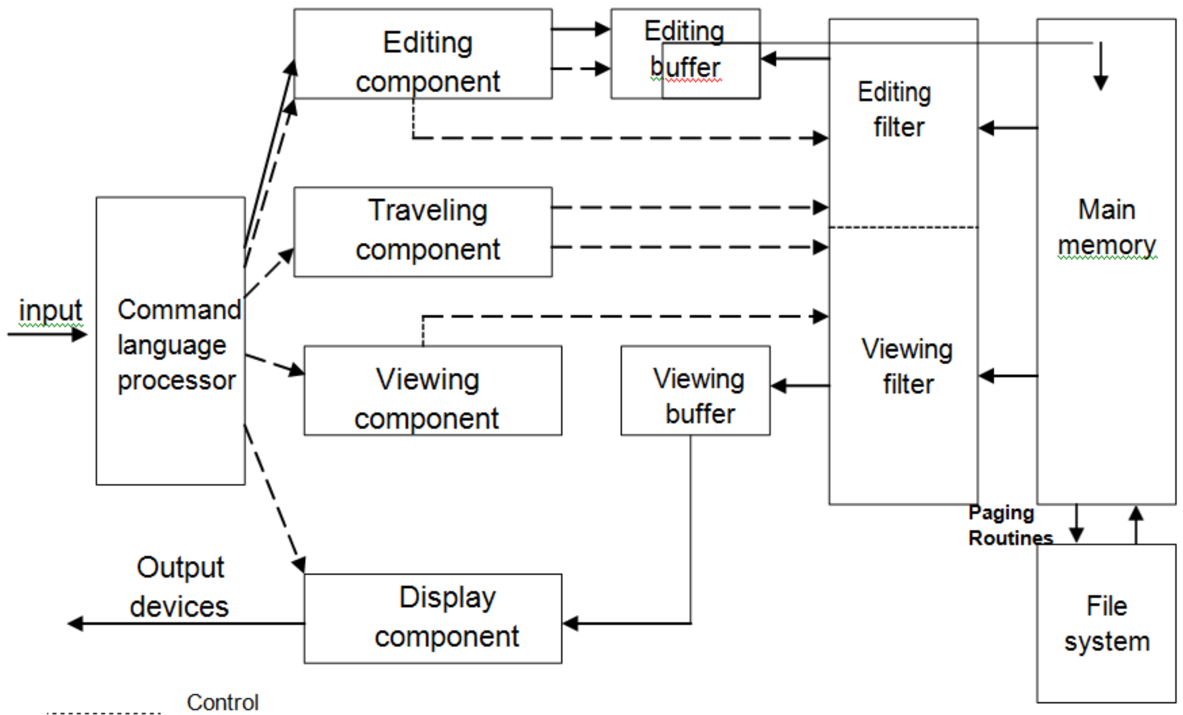
    - text devices- are type writer like key boards on which a user presses and releases keys sending a unique code for each key.
    - button or choice devices- generate an interrupt causing an invocation of an associated application program action. They include a set of function keys. Buttons can be simulated in software.
    - Locator devices – are two dimensional analog to digital converters that position a cursor symbol on the screen by observing the user's movement of the device. Eg: mouse, data tablet. Returns the coordinates of the position of the device. Text devices with arrow keys can be used as locator devices . Arrow shows left, right , up or down.
    - Voice input devices- Translates spoken words to their textual equivalent.

- **Output Devices** lets the user view the elements being edited and the results of the editing operations. CRT terminals use hardware assistance for such features as moving the cursor , inserting and deleting characters and lines etc.
- **The interaction language** is one of the common types.
    - **Typing or text command oriented-** the user communicates with the editor by typing text strings both for command names and for operands.These strings are sent to the editor and echoed to the output device.This requires the user to remember the commands.
    - **Function key oriented-** In this each command is associated with a marked key on the user's keyboard.
    - **Menu oriented systems-** A menu is a multiple choice set of text strings or icons which are graphic symbols that represent object or operations. The user can perform actions by selecting items from the menu. Some systems have the most used functions on a main command menu and have secondary menus to handle the less frequently used functions.

5.1.3Editor Structure:

Most text editors have a structure similar to that shown in the following figure. That is most text editors have a structure similar to shown in the figure regardless of features and the computers

Command language Processor accepts command, uses semantic routines – performs functions such as editing and viewing. The semantic routines involve traveling, editing, viewing and display functions.



Typical Editor Structure

- The **command language processor** accepts input from the user's input devices and analyses the tokens and syntactic structure of the commands. That is, it function like lexical and syntactic phases of a compiler. It invokes the semantic routines directly. The command language processor also produces an intermediate representation of the desired editing operations. This representation is decoded by an interpreter that invokes the appropriate semantic routines.

- **Editing Component** - In editing a document, the start of the area to be edited is determined by the current editing pointer maintained by the editing component. Editing component is a collection of modules dealing with editing tasks. Current editing pointer can be set or reset due to next paragraph, next screen, cut paragraph, paste paragraph etc..,.

  - **Travelling component –** performs the setting of the current editing and viewing pointers and thus determines the point at which the viewing/editing filtering begins.
  - **Editing filter**- When the user issues an editing command the editing component invokes the editing filter. This component filters the document to generate a new **editing buffer** based on the current editing pointer as well as on the editing filter parameters.

  - **Filtering** consists of selection of continuous characters beginning at the current point.

  - **Viewing component**- thee start of the area to be viewed is determined by the viewing pointer. This pointer is maintained by the viewing component. When the display need to be updated the viewing component invokes the **viewing filter**. This component filters the document to generate a new **viewing buffer**.

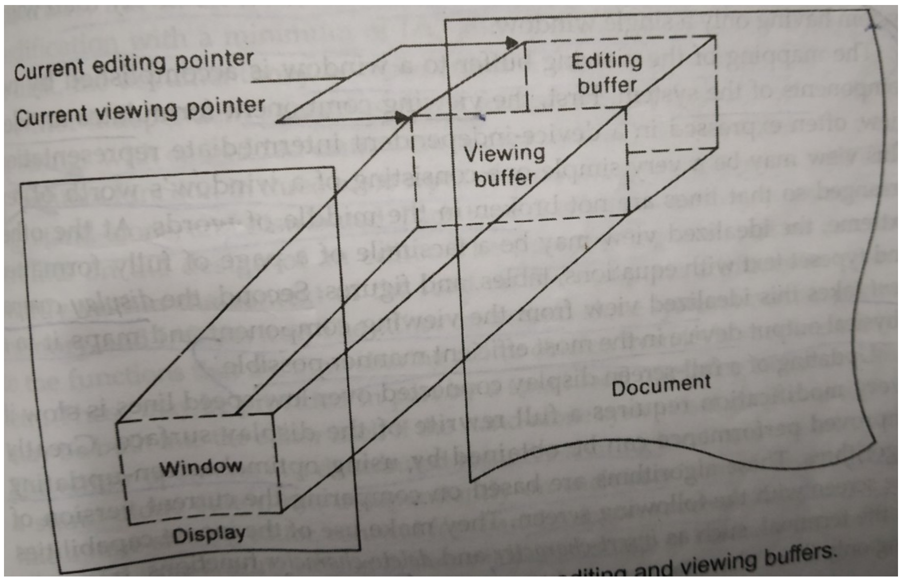
  - **Display component**- The viewing buffer is then passed to the display component which produces a display by mapping the buffer to a rectangular subset of the screen called window.

  - The editing and viewing buffers can be independent or overlapped.

  - The mapping of viewing buffer to window is accomplished by two components.

    1. Viewing component- formulates an ideal view

    2. Display component – takes this ideal view from viewing component and maps it to the output device.



Simple relationship between editing and viewing buffers

- The components of the editor deal with a user document on two levels: In main memory and in the disk file system. Loading an entire document into main memory may be infeasible – only part is loaded – demand paging is used – uses editor paging routines.
- Documents may not be stored sequentially as a string of characters. Uses separate editor data structure that allows addition, deletion, and modification with a minimum of I/O and character movement.
- Many editors use terminal control database. They can call terminal independent library routines such as scroll down, or read cursor positions.
- Types of editors based on computing environment: Editors function in three basic types of computing environments:

    1. Time sharing
    2. Stand-alone
    3. Distributed.

    Each type of environment imposes some constraints on the design of an editor.

- In time sharing environment, editor must function swiftly within the context of the load on the computer's processor, memory and I/O devices.
- In stand-alone environment, editors on stand-alone system are built with all the functions to carry out editing and viewing operations – The help of the OS may also be taken to carry out some tasks like demand paging.
- In distributed environment, editor has both functions of stand-alone editor; to run independently on each user's machine and like a time sharing editor, contend for shared resources such as files.

Interactive Debugging Systems:

An interactive debugging system provides programmers with facilities that aid in testing and debugging of programs. Many such systems are available during these days. Our discussion is broad in scope, giving the overview of interactive debugging systems – not specific to any particular existing system.

Here we discuss

- Introducing important functions and capabilities of IDS

- Relationship of IDS to other parts of the system

- Debugging methods

Debugging Functions and Capabilities:

- One important requirement of any IDS is unit test functions specified by the programmer. Such functions deal with execution sequencing , which is the observation and control of the flow of program execution.Eg: The program may be suspended after a fixed number of instructions are executed. The programmer can define break points. After the program is suspended debugging commands can be used to diagnose errors.
- A Debugging system should also provide functions such as tracing and trace back

- Tracing can be 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.
- Trace back can show the path by which the current statement in the program was reached. It can also show which statements have modified a given variable or parameter. The statements are displayed rather than as hexadecimal displacements.
- Program-Display capabilities. A debugger should have good program-display capabilities.
    - Program being debugged should be displayed completely with statement numbers.
    - The program may be displayed as originally written or with macro expansion.
    - Keeping track of any changes made to the programs during the debugging session. Support for symbolically displaying or modifying the contents of any of the variables and constants in the program. Resume execution – after these changes.

- A debugging system should consider the language in which the program being debugged is written. A single debugger – many programming languages – language independent. The debugger- a specific programming language– language dependent.
- The debugging system should be able to deal with optimized code. Many optimizations involve rearrangement of code in the program.Eg: Separate loops can be combined into single loop.
- Storage of variables- When a program is translated the compiler assigns a home location in memory for each variables. Variable values can be temporarily held in registers to improve speed of access. If a user changes the value of a variable in home location while debugging the modified value might not be used by the program.
- The debugging of optimized code requires cooperation from optimized compiler.

Relationship with Other Parts of the System:

- The important requirement for an interactive debugger is that it always be available. Must appear as 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 such as interactive subsystems.

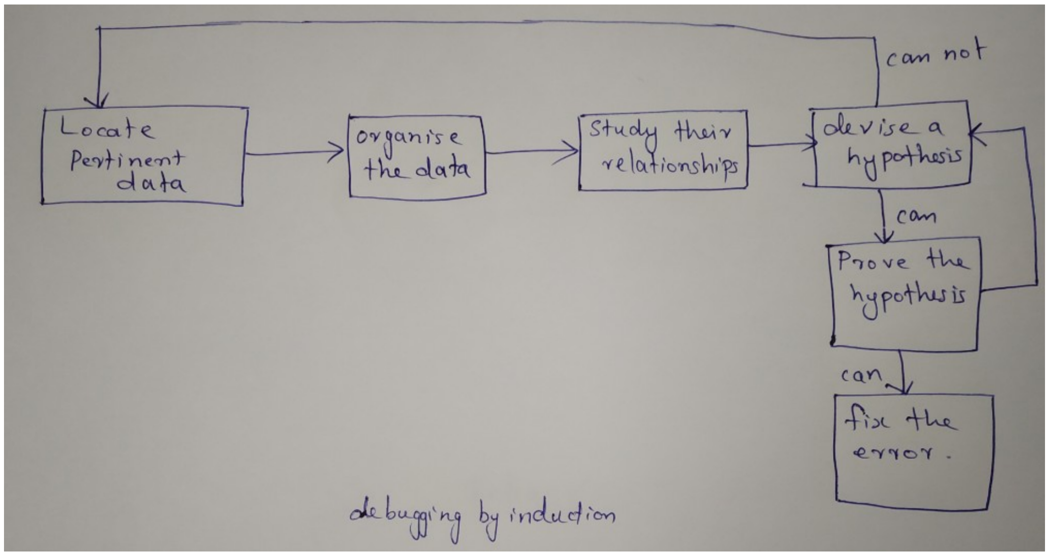
- Debugging is more important at production time than it is at application-development time. When an application fails during a production run, work dependent on that application stops.
- The debugger must also exist in a way that is consistent with the security and integrity components of the system.
- The debugger must coordinate its activities with those of existing and future language compilers and interpreters.

Debugging Methods

1. Debugging by Induction
2. Debugging by Deduction
3. Debugging by Backtracking

Debugging by Induction

- In induction one proceeds from the particulars to the whole.ie, By starting with the symptoms of the error in the result of one or more test cases and looking for relationships among the symptoms.



1. Locate the pertinent data: Consider all the available data or symptoms about the problems

2. Organise the data: Pertinent data is structured to allow one to observe patterns of particular importance and search for contradictions. One such organization structure can be a table.

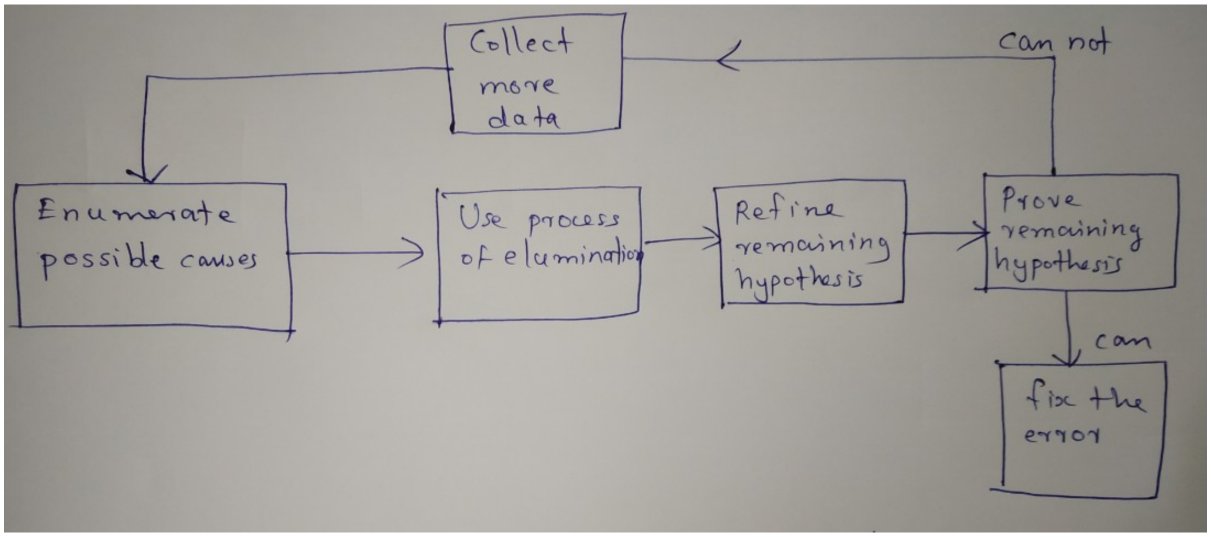
3. Devise a hypothesis: In this step study the relationship between the clues and devise using patterns, one or more hypothesis about the cause o the error.

4. Prove the hypothesis: Prove the reasonableness of the hypothesis before proceeding. A failure to this, results in the fixing of only one symptom of the problem.

Debugging by Deduction

- Is a process of proceeding from general theories or premises to arrive at a conclusion.
    1. Enumerate all possible cases- The first step is to develop all causes of the error.
    2. Use the data to eliminate possible causes- By careful analysis of data particularly by looking for contradictions attempt to eliminate all possible causes except one.
    3. Refine the remaining hypothesis- The possible causes at this point may be correct. But refine it to be more specific.

    4. Prove the remaining hypothesis.


Collect more data
can not
Enumerate possible causes
Use process of elumination
Refine remaining hypothesis
Prove remaining hypothesis
can
fix the error

Debugging by Back Tracking

- For small programs the method of backtracking is more effective to locate errors.
- To use this method start at the place in the program where an incorrect result was produced and go backwards in the program one step at a time. That is executing the program in reverse order to derive the values of all variables in the previous step. Then the error can be localized.

Device Driver

Learn from PPT