UNIT – V

Text editor

Prof. Reshma Pise

Comp Engg. Dept

Vishwakarma University

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

Types of Editors

1) Line Editors : Eg: Edlin

2) Stream Editors : vi , context oriented. Find(“abc”)

3) Screen Editors : WYSWYG Screenful of text displayed. cursor

4) Word Processor : Eg : MS-Word , advanced formatting, spell check

5) Structure editors : Specify the structure of document

Eg: Syntax directed editors

A text editor allows you to edit a text file

(create, modify etc…)

Text editors on Windows OS

- Notepad, WordPad, Microsoft Word

Text editors on UNIX OS

- vi, emacs, jed, pico, gedit

COMMON EDITING FEATURES

Moving the cursor

Deleting

Replacing

Pasting

Searching

Searching and replacing

Saving and loading

Miscellaneous(e.g. quitting)

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

Here we restrict to text editors, where character strings are the primary elements of the target text

CONTD..

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

Select the part of the target document to be viewed and manipulated

Determine how to format this view on-line and how to display it

Specify and execute operations that modify the target document

Update the view appropriately

CONTD..

- The above tasks involves traveling, filtering and formatting
  - Traveling (Navigation) – locate the area of interest
  - Filtering - extracting the relevant subset
  - Formatting – visible representation on a display screen
- Editing phase involves – insert, delete, replace, move, copy, cut, paste, etc…

CONTD..

- Manuscript-oriented editor – characters, words, lines, sentences and paragraphs
- Program-oriented editors – identifiers, keywords, statements
- User wish – what he wants - formatted

USER INTERFACE

- Conceptual model of the editing system
  - Provides an easily understood abstraction of the target document and its elements
- 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
  - Are used to enter elements of text being edited, to enter commands
- The input devices
  - Lets the user view the elements being edited and the results of the editing operations
- The interaction language
  - Communication with the editor

INPUT DEVICES DIVIDED INTO THREE CATEGORIES

Text Devices - keyboard

Button Devices – special function keys, symbols on the screen

Locator Devices – mouse, data tablet

- Voice input devices – translates spoken words to their textual equivalents

OUTPUT DEVICES

Teletypewriters - **first output devices**

Glass teletypes - **Cathode ray tube (CRT) technology**

Advanced CRT terminals

TFT Monitors - Wysiwyg

Printers – Hard-copy

INTERACTION LANGUAGE

Typing oriented or text command oriented - oldest editors, use of commands, use of function keys, control keys etc.,

Menu-oriented user interface – menu is a multiple choice set of text strings or icons. Display area for text is limited. Menus can be turned on or off.

TEXT EDITORS IN WINDOWS ENVIRONMENT

- Edit - MS-DOS editor, menu oriented, options are selected with specified Alphabets
- Notepad - A basic text editor that you can use to create simple documents. Menu oriented, use of control keys
- WordPad - We can create and edit simple text documents or documents with complex formatting and graphics, uses menu as well as icons
- Microsoft Word - A sophisticated word processor, menu as well as use of icons

TEXT EDITORS IN UNIX ENVIRONMENT

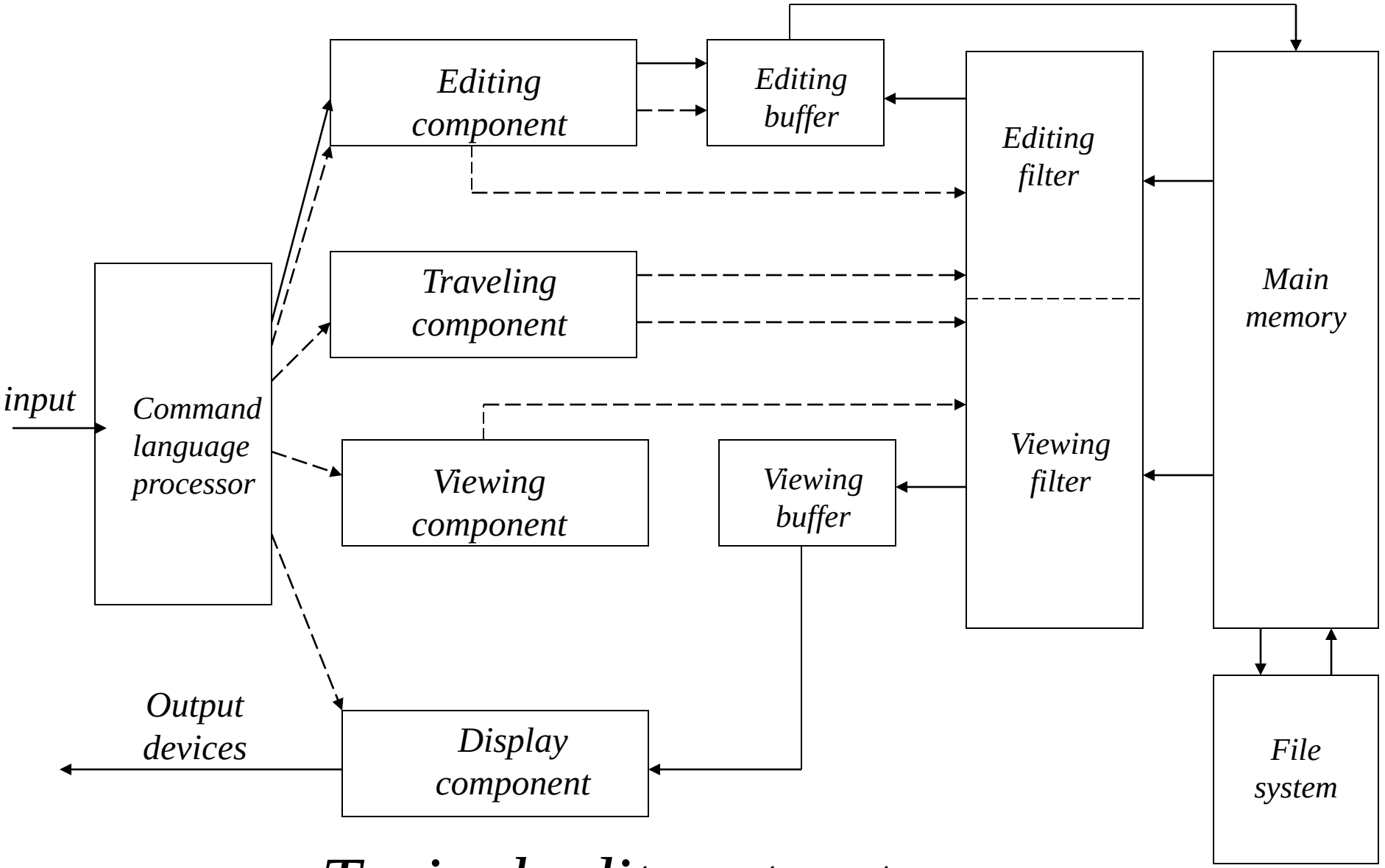
1) vi - text editor, old, reliable, present on every Unix machine, uses two modes, command mode, text mode

2) emacs - the Extensible, Customizable, Self-Documenting, Real-time Display Editor

3) gedit

EDITOR STRUCTURE

- Most text editors have a structure similar to that shown in the next slide



Typical editor structure

EDITOR STRUCTURE

- 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

Editing operations are specified explicitly by the user and display operations are specified implicitly by the editor

- *Traveling and viewing operations may be invoked either explicitly by the user or implicitly by the editing operations*

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..,

WHEN EDITING COMMAND IS ISSUED…

Editing component invokes the editing filter – generates a new editing buffer – contains part of the document to be edited from current editing pointer

Filtering and editing may be interleaved, with no explicit editor buffer being created

IN VIEWING A DOCUMENT…

The start of the area to be viewed is determined by the current viewing pointer maintained by the viewing component

Viewing component is a collection of modules responsible for determining the next view

Current viewing pointer can be set or reset as a result of previous editing operation

WHEN DISPLAY NEEDS TO BE UPDATED…

Viewing component invokes the viewing filter – generates a new viewing buffer – contains part of the document to be viewed from current viewing pointer

Line editors – viewing buffer may contain the current line

Screen editors - viewing buffer contains a rectangular cutout of the quarter plane of the text

Viewing buffer is then passed to the display component of the editor, which produces a display by mapping the buffer to a rectangular subset of the screen – called a window

The editing and viewing buffers may be identical or may be completely disjoint

Identical – user edits the text directly on the screen

Disjoint – Find and Replace

There are 150 lines of text, user is in 100th line, decides to change all occurrences of 'text editor' with 'editor'

The editing and viewing buffers can also be partially overlap, or one may be completely contained in the other

Windows typically cover entire screen or a rectangular portion of it

May show different portions of the same file or portions of different file

Inter-file editing operations are possible

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

EDITORS FUNCTION IN THREE BASIC TYPES OF COMPUTING ENVIRONMENTS:

Time sharing

Stand-alone

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

Debugging Functions and Capabilities

- Debugging process

Program-Display capabilities

Relationship with Other Parts of the System

User-Interface Criteria

DEBUGGING

- **Debugging** is a methodical process of finding and reducing the number of bugs, or defects, in a computer program
- An interactive debugging system provides programmers with facilities that aid in testing and debugging of programs
- Here we discuss
  - Introducing important functions and capabilities of IDS
  - Relationship of IDS to other parts of the system
  - The nature of the user interface for IDS

DEBUGGING FUNCTIONS AND CAPABILITIES

Debugging system should also provide functions such as tracing and trace back

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

DEBUGGING PROCESS

- Print debugging is the act of watching (live or recorded) trace statements, or print statements, that indicate the flow of execution of a process.
- In computers, debugging is the process of locating and fixing or bypassing bugs (errors) in computer program code or the engineering of a hardware device.
- Remote debugging is the process of debugging a program running on a system different than the debugger.
- Post-mortem debugging is the act of debugging the core dump of process.

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

The context being used has many different effects on the debugging interaction.
The statements are different depending on the language

Cobol - MOVE 6.5 TO X
Fortran - X = 6.5
C - X = 6.5

Examples of assignment statements

Similarly, the condition that X be unequal to Z may be expressed as

Cobol - IF X NOT EQUAL TO Z
Fortran - IF (X.NE.Z)
C - IF (X <> Z)

- It is also important that a debugging system be able to deal with optimized code. Many optimizations like

  - Invariant expressions can be removed from loops
  - Separate loops can be combined into a single loop
  - Redundant expression may be eliminated
  - Elimination of unnecessary branch instructions

```
                goto B
                .....
        A :     goto B
    .... B :    print( )
```

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.

USER-INTERFACE CRITERIA

Debugging systems should be simple in its organization and familiar in its language, closely reflect common user tasks.

The user interaction should make use of full-screen displays and windowing-systems as much as possible.

With menus and full-screen editors, the user has far less information to enter and remember.

USER-INTERFACE CRITERIA

There should be complete functional equivalence between commands and menus – user where unable to use full-screen IDSs may use commands.

The command language should have a clear, logical and simple syntax; command formats should be as flexible as possible.

Any good IDSs should have an on-line HELP facility. HELP should be accessible from any state of the debugging session.

Thank You