Chapter 4

Editors and Debugging Systems

This Chapter gives you…

- Text editors
- Interactive Debugging Systems

4.0 Introduction

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.

4.1 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).

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

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

The above task involves traveling, filtering and formatting. Editing phase involves – insert, delete, replace, move, copy, cut, paste, etc…

- Traveling – locate the area of interest
- Filtering - extracting the relevant subset
- Formatting – visible representation on a display screen

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.

4.1.2 User 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, button devices and, locator devices. Text Devices are keyboard. Button Devices are special function keys, symbols on the screen. Locator Devices are mouse, data tablet. There are voice input devices which translates spoken words to their textual equivalents.

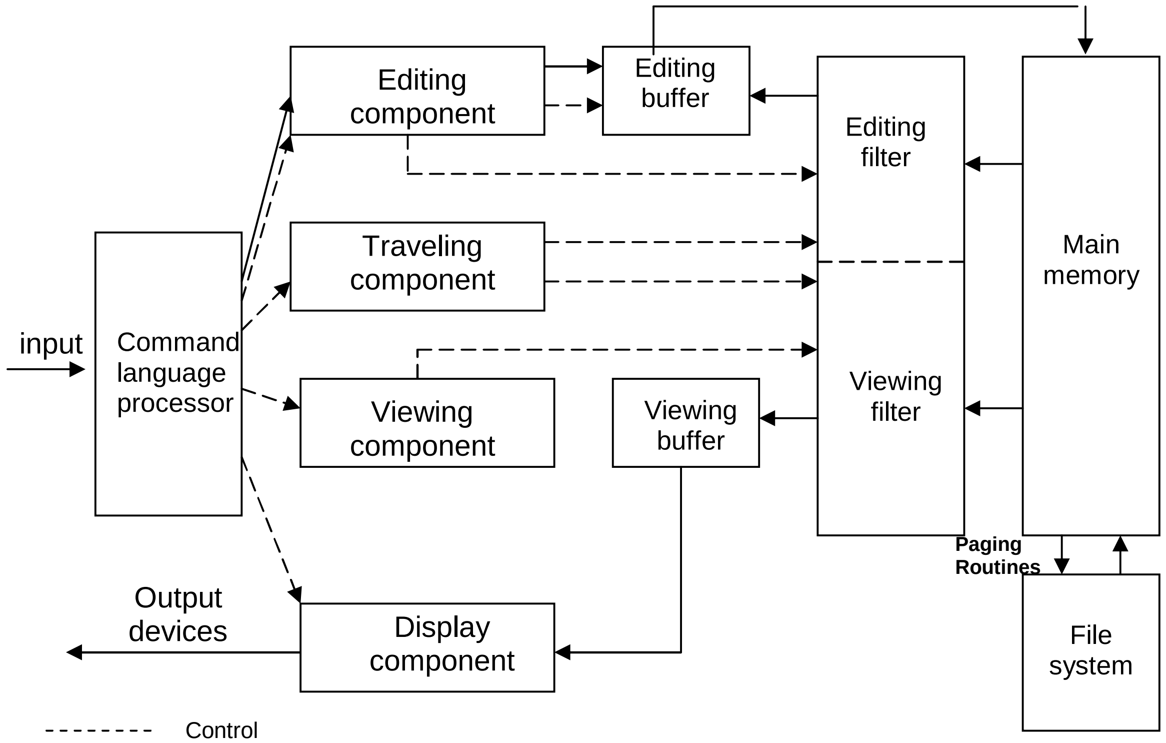
Output Devices are Teletypewriters (first output devices), Glass teletypes (Cathode ray tube (CRT) technology), Advanced CRT terminals, TFT Monitors (Wysiwyg) and Printers (Hard-copy).

The interaction language could be, typing oriented or text command oriented and menu-oriented user interface. Typing oriented or text command oriented interaction was with oldest editors, in the form of use of commands, use of function keys, control keys etc., Menu-oriented user interface has menu with a multiple choice set of text strings or icons. Display area for text is limited. Menus can be turned on or off.

4.1.3 Editor 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

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. In case of 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 (For example, 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.

4.1.4 Types of editors based on computing environment

Editors function in three basic types of computing environments: Time sharing, Stand-alone, and 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.

4.2 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
- The nature of the user interface for IDS

4.2.1 Debugging Functions and Capabilities

One important requirement of any IDS is the observation and control of the flow of program execution. Setting break points – execution is suspended, use debugging commands to analyze the progress of the program, résumé execution of the program. Setting some conditional expressions, evaluated during the debugging session, program execution is suspended, when conditions are met, analysis is made, later execution is resumed.

A Debugging system should also provide functions such as tracing and traceback. 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… Traceback 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

4.2.2 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.

To provide these functions, a debugger 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 debugger must be sensitive to the specific language being debugged.

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)
```

Similar differences exist with respect to the form of statement labels, keywords and so on…

The notation used to specify certain debugging functions varies according to the language of the program being debugged. Sometimes the language translator itself has debugger interface modules that can respond to the request for debugging by the user. The source code may be displayed by the debugger in the standard form or as specified by the user or translator.

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

Leads to rearrangement of segments of code in the program. All these optimizations create problems for the debugger, and should be handled carefully.

4.2.3 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.

4.2.4 User-Interface Criteria

Debugging systems should be simple in its organization and familiar in its language, closely reflect common user tasks. The simple organization contribute greatly to ease of training and ease of use. 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. 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.