DESIGN AND IMPLEMENTATION OF UNIX EXPLORER

A DISSERTATION

Submitted in partial fulfilment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

AJAY KUMAR

ER & DCI NOIDA

IIT Roorkee-ER&DCI, Noida
C-56/1, "Anusandhan Bhawan"
Sector 62, Noida-201 307
FEBRUARY, 2003

Enrolment No. 019003

$\frac{621.380285}{AJA}$

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this dissertation titled **"DESIGN AND IMPLEMENTATION OF UNIX EXPLORER"**, in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Information Technology,** submitted in **IIT, Roorkee – ER&DCI Campus, Noida,** is an authentic record of my own work carried out during the period from August 2002 to February, 2003 under the guidance of **Dr. MOINUDDIN**, Senior Professor, Jamia Milia Islamia, New-Delhi.

The matter embodied in this dissertation has not been submitted by me for award of any other degree of diploma

Date: 24-2-2003
Place: Noida

[illegible]
(Ajay Kumar)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 24-2-2003
Place: Noida

[illegible]
(Dr. Moinuddin),
Professor ,
Jamia Milia Islamia,
New Delhi.

ACKNOWLEDGEMENT

I hereby take the privilege to express my deepest sense of gratitude to **Prof. Prem Vrat**, Director, Indian Institute of Technology, Roorkee, and **Mr. R.K. Verma,** Executive Director, Electronics Research & Development Center of India, Noida for providing me with this valuable opportunity to carry out this work. I am also very grateful to **Prof. A.K. Awasthi**, Programme Director and Dean, Post Graduate Studies and Research, **Prof. R.P. Agarwal,** course coordinator, IIT, Roorkee and **Mr. V.N. Shukla**, course coordinator, ER&DCI, Noida for providing the best of the facilities for the completion of this work and constant encouragement towards the goal.

I am grateful to my guide **Dr. Moinuddin**, Professor, JMI, New Delhi for his valuable guidance, advice, suggestions and constant encouragement through numerous discussions and demonstrations.

My sincere thanks to **Mr. Munish Kumar**, Project Engineer, ER&DCI, Noida for his valuable suggestions.

I owe special thanks to my best friends, all of my classmates and other friends who have helped me formulate my ideas and have been a constant support. Thanks to my parents and my brother who provided their support and enduring confidence in me during my entire life. Last but not the least, I thank almighty for being on my side from the conception of this idea to its implementation.

[illegible]
(Ajay Kumar)
Enrollment No. 019003

CONTENTS

ABSTRACT

Unix explorer is a GUI style plain-text editor. The system provides the user with a user friendly GUI and helps him in browsing his Unix account. It provides the user with a set of features like user directory traversal, editing and creation of files, moving of files between directories authentication etc.

It is based on fully graphical design, and command set based on successful Macintosh and MS Windows editors. In Unix explorer, one can do such operation that can be done with a character-based editor, in the same (or fewer) number of keystrokes, and without touching the mouse. What makes Unix explorer different is that you can use the mouse at any step in the process. With the mouse, one can manipulate the text graphically and more directly, and move quickly through menus and dialogs to navigate less familiar commands and features.

client server processing styles. Chapter 3 has to say about the complete details for design and implementation part of general graphical user interface structure, which represents the unix explorer with the support of editor. Chapter 4 specifies the some of the functional utilities used in adherence with the completion of the interface and the functionalities done for the implementation of the unix explorer. Chapter 5 describes the system development life cycle along with the strategies used during the client server process like different types of servers on the basis of state information and steps performed by them. In Chapter 6 result has been shown for the ease of the user in accordance with the explorer designed. Also it enhances the technical strategies must me taken care of regarding the future exploration of the client server application in different file transfer and techniques of compilation and execution of the code written in the editor. Chapter 7 finally includes the conclusion and difficulties along with the future work that could be done for the further improvement of the system.

LITERATURE SURVEY

The Unix Explorer is an application that allows a user to explore his account. A Server Process comes into being only when a client request is received. This process waits and listens to the client requests. These requests are essentially the various operations such as creating a new file, deleting a file, printing a file, editing a file, moving files in directories etc. that the user desires to be done in his account. Upon receiving the client's requests, the server carries out these operations in the user 's account on the unix server and sends the fresh information of the user 's account to the client program. A client process that runs on the unix machine allows the user to log into his unix account. Also provide a GUI with a windows explorer look and feel. The GUI basically, has various menu options that allow the user to create new files, delete old files, and various other operations in his account. A simple text editor is included that can be used to read, create and edit text files. Hoare[2] describes a proto-language for describing communicating sequential processes. His language is based on Dijkstra's guarded command language and yields a synchronization mechanism. In particular, langauge allows description of parallelism within a process, message passing, and single and repeated selection guarded by message receipt. His examples show minimal type checking and minimal modularity. Messages are named tuples of primitive types. The intent of his paper is to introduce useful primitive concepts and suggest a basis for notation, without explicating detailed semantics. Motif User's Guide[9] describes how to interact with Motif-based applications, including how to use and customize the Motif Window Manager, how to use the Motif widgets, and how to customize your interaction with Motif applications.Weihl[4] discusses the problems in specifying asynchronous systems by separating termination and partial correctness and using a non-deterministic guarded state machine language to describe them. That identifies interfaces and atomic messages of distributed processes. Expressions in a given state machine language are sufficient for generating a particular trace of the described system's behavior, and for use as the basis of proofs for machine equivalence (through state mapping and invariants).

Weihl notes that over-specification can cramp an implementation and his alarmingly non-deterministic example specifications support this, asserting that coarser, more abstract, specifications are simpler to understand and deal with. No entry into vastly hierarchical, modular or component-wise descriptions is made: the main focus is on abstraction functions for reasoning about a system. He gives no explicit technique for using the guarded state machine description to test a particular trace. Marco[1]is a shell/GUI with an emphasis on referring to, and manipulation of distributed objects. It is based on the DCE cell model, and is described as a visualiser tool with a manipulation metaphor similar to window managers. The analogy of windows with objects implies hierarchical control and perhaps composition, but this is not expanded upon. Walter[7] makes use of the Elvin event logger in order to display real-time or recorded event interaction for debugging/comprehension purposes. In particular, the display tool allows the user to manipulate the representation by zooming, imploding. Berry and Raymond[5] refine the ISO RM-ODP into a broad(andvaguely definitive) reference for the terminology used in Hector. Tool support for these concepts is discussed towards the end of this work, noting that a toolset for managing a type repository' would need language translation capabilities. The concepts described are echoed by Arnold and Bond[6] where the need for such an extensible framework is argued for in terms of solutions for interoperability. Again, type management is recommended to remain as flexible and unencumbered as possible for future extensions. Induslka [3] outline some of the classes of types that will appear in the type manager.

2.1 Client-Server Architecture

A server process (program) fulfills the client request by performing the task requested. Server programs generally receive requests from client programs, execute database retrieval and updates, manage data integrity and dispatch responses to client requests. Sometimes server programs execute common or complex business logic. The server-based process "may" run on another machine on the network. This server could be the host operating system or network file server; the server is then provided both file system services and application services. Or in some cases, another desktop machine provides the application services. The server process acts as a software engine that manages shared resources such as databases, printers, communication links, or high powered-processors. The server process performs the back-end tasks that are common to similar applications.

2.1.1 Two Tier Architecture

A two-tier architecture is where a client talks directly to a server, with no intervening server. It is typically used in small environments (less than 50 users).A common error in client/server development is to prototype an application in a small, two-tier environment and then scale up by simply adding more users to the server. This approach will usually result in an ineffective system, as the server becomes overwhelmed .To properly scale to hundreds or thousands of users, it is usually necessary to move to a three-tier architecture.

2.1.2 Three-Tier Architecture

A three-tier architecture introduces a server (or an "agent") between the client and the server. The role of the agent is manifold. It can provide translation services (as in adapting a legacy application on a mainframe to a client/server environment), metering services (as in acting as a transaction monitor to limit the number of simultaneous requests to a given server), or intelligent agent services (as in mapping a request to a number of different servers, collating the results, and returning a single response to the client.

2.2 Client-Server Technical Issues

The basic characteristics of client-server architectures are:

- Combination of a client or front-end portion that interacts with the user, and a server or back-end portion that interacts with the shared resource. The client process contains solution-specific logic and provides the interface between the user and the rest of the application system[1]. The server process acts as a software engine that manages shared resources such as databases, printers, modems, or high powered processors.

- The front-end task and back-end task have fundamentally different requirements for computing resources such as processor speeds, memory, disk speeds and capacities, and input/output devices.
- The environment is typically heterogeneous and multivendor. The hardware platform and operating system of client and server are not usually the same. Client and server processes communicate through a well-defined set of standard application program interfaces (API's) and RPC's.

An important characteristic of client-server systems is scalability. They can be scaled horizontally or vertically. Horizontal scaling means adding or removing client workstations with only a slight performance impact. Vertical scaling means migrating to a larger and faster server machine or multiservers.

2.3 Servers Classification

The simplest forms of servers are disk servers and file servers. With a file server, the client passes requests for files or file records over a network to the file server. This form of data service requires large bandwidth and can slow a network with many users down considerably. Traditional LAN computing allows users to share resources, such as data files and peripheral devices, by moving them from standalone PCs onto a Networked File Server (NFS).

The more advanced form of servers are database servers, transaction server and application servers. In database servers, clients passes structured query language requests as messages to the server and the results of the query are returned over the network. The code that processes the structured query language request and the data resides on the server allowing it to use its own processing power to find the requested data, rather than pass all the records back to a client and let it find its own data as was the case for the file server. In transaction servers, clients invoke remote procedures that reside on servers, which also contains an structured query language database engine. There are procedural statements on the server to execute a group of structured query language statements (transactions) which either all succeed or fail as a unit. The applications based on transaction servers are called on-line transaction processing and tend to be missioncritical applications which require 1-3 second response time, 100% of the time and require tight controls over the security and integrity of the database.

The communication overhead in this approach is kept to a minimum as the exchange typically consists of a single request/reply (as opposed to multiple structured query language statements in database servers). Application servers are not necessarily database centered but are used to server user needs. A resource on a server allows users to share data, while security and management services, which are also based in the server, ensure data integrity and security.

2.4 Client-Server Processing Styles

The five ways of describing the different c/s styles based on how they split the three components of any application: user interface, business or application logic, data management. The five styles are distributed presentation, remote presentation, distributed function, remote data management and distributed data management.

2.4.1 Distributed Or Remote Presentation

In the mainframe IBM world, client-server is essentially distributed or remote presentation. This style maps a workstation Graphical User Interface (GUI) front end onto an existing application's text-based screen. This is also called Remote, Mapping, Front-ending or HLLAPI (High-Level Language Application Programming Interface). The mode of operation is typically intelligent workstations intercepting and interrogating text-screen data streams sent from a host for display in a windowed environment. This is "frontware" solution, where a GUI front end is added to an application and is placed on a workstation. However, most processing[2] remains on the host or server, with the exception of user interface translation logic and possibly validation logic. For example, data from an application program is sent to a screen program on the mainframe to be displayed. The merged data is sent to the workstation as a text screen data stream. The workstation interprets the data and converts it to graphical form in a window. Typically every mainframe screen used by the application has a corresponding window on the workstation and vice versa. When the user enters the data in a GUI window, it is transformed by the frontware application into a corresponding text screen data format and is sent to the host computer.

2.4.2 Remote Data Management

In remote data management, the entire application resides on the client and the data management is located on a remote server/host. Remote Data Management is relatively easily to program for because there is just one application program. The client communicates with the server using structured query language, the server then responds with data that satisfies the query. RDBMS products that offer remote data management provide a layer of software on the client to handle the communication with the DBMS server. This style represents a more traditional LAN database server or file server approach. Workstations support the presentation, functional logic and interface with the data server through the data manipulation language. Distributed data management is an extension of remote data management and uses the distributed facilities of the DBMS to access distributed data in a manner transparent to users. This is most relevant for architectures having data spread across several servers and when access to a DBMS on another server is required.

2.4.3 Function Processing

In this type of function the split occurs in the application functionality, one part going to the client, other to the server. Distributed function applications are the most complex of the three typologies since two separately compiled application programs must be developed. Developers must analyze where each function should reside and what type of dialog must occur between the either a message-based or remote procedure call (RPC) mechanism for transfer of dialog and data. However, there are many variants of this topology. One variant of the distributed function style is where data management and application functioning occur at both the client and server. In this instance, data management at the client would include referential, read-only data. Data frequently updated or accessed by many users would reside on the server.

ANALYSIS AND DESIGN

3.1 GUI Components:

The GUI consists of the following components:

A menu bar with the following options

- File→New, Open, Delete, Print, Connect, Disconnect, Exit.
- Edit→Undo, Cut, Copy, Paste
- Search→Details
- Help→About UNIX man

A bar (Panel) that shows the full path of the current highlighted file in the file tree.

3.2 Menu Bar Options:

File→ Connect

- The user is prompted with a message box asking for the host address, username and password.
- A socket is created and this information is passed onto the server side.
- Internet super server initiates our server process, which checks and authenticates this information.
- If invalid, the user is prompted with an appropriate error message.
- Else the server sends data about his home directory to the client, where it is displayed in a tree structure.
- Also all the appropriate menu options are enabled after a successful connects.

File→ Disconnect

- The user is simply logged out and the program returns to its initial state.
- Our server process is terminated and socket descriptor is closed.

File→New

- The user is asked to specify the name of the file he wants to create.

- User is then provided with a simple text editor, where he can type he wants and saves it.
- On being saved, the text is sent to the server side in a buffer and the file is saved in the specified name.

File→Open

- The currently highlighted name of the file is sent to the server.
- Its access permissions are checked and if allowed, the file is sent to the client side where contents are displayed in the fore mentioned text editor.

File→delete

- The currently highlighted name of the file is sent to the server.
- The file is removed from the directory.

File→Print

- The highlighted file is spooled to the print spooler daemon.

Edit→Cut

- The currently highlighted name of the file is sent to the server.
- The server stores the pathname along with the filename sent ti it.

Edit→Copy

- The currently highlighted name of the file is sent to the server.
- The server stores the pathname along with the filename sent to it.

Edit→Paste

- The server checks to see whether the file, which is to be pasted, has been cut or copied earlier.
- If copied, the file is pasted to the current directory.
- If cut, the file is move to the current directory.

Edit→Undo

The last change made through Copy-Paste, Cut-Paste or Delete operation is backtracked.

View→ Details

- The currently highlighted name of the file is sent to the server.

- The server executes the stat system call and collects all information pertaining to the file.
- This information is formatted into a string and is passed to the client where it is displayed.

Help→Unix man

- The user is prompted to enter the keyword.
- The keyword is passed to the server and the server executes the man command for that keyword.
- The information is passed to the client where it is displayed.

  Besides the functions being available on the menu bar, it also provides a totally interactive GUI which more or less corresponds to the Windows standard .

3.3 Single Clicked Option

Single Clicked (Directory)

- Name of the highlighted directory is sent to the server.
- The directory name received is made the current directory of the user & the information about it is send to the client application.
- On receiving the data, the file tree on the right of the screen is updated to show the files of the current highlighted directory.

Single Click(File)

- The clicked file's name is saved by the client application & is displayed at the status bar along with the available operations.

3.4 Double Click Option

Double Click(Directory)

- Name of the double clicked directory is sent to he server.
- The directory tree is expanded to show the subdirectories of the clicked directory.
- The directory name sent is made the current directory of the user & data about this directory is sent to the client application.
- The file tree on right of the screen is updated to show the files of the currently highlighted directory.

Double Click (File)

- The currently highlighted name of the file is sent to the server.
- Its access permissions are checked and if allowed the file is sent to the client side, where it's contents are displayed in the fore mentioned text editor.

3.5 Expand Icon Click (•)

- The name of the corresponding directory is sent to the UNIX server.
- The directory tree is expanded to show the subdirectories of the clicked directory.
- The directory name sent is made the current directory of the user & information about this directory is sent to the client application.
- The file tree on the right of the screen is updated to show the files of the currently highlighted directory.

3.6 Collapse Iconclick (×)

- Name of the highlighted directory is sent to the server.
- The directory tree is collapsed to the level of clicked directory.
- This directory is made the current directory of the user & information about this directory is sent to client application.
- The file tree on the right of the screen is updated to show the files of the currently highlighted directory.

Chapter 4

IMPLEMENTATION

4.1 Selecting Text

In this Unix explorer with editor has two general types of selections, primary (highlighted text), and secondary (underlined text). Selections can cover either a simple range of text between two points in the file, or they can cover a rectangular area of the file. Rectangular selections are only useful with non-proportional (fixed spacing) fonts.

To select text for copying, deleting, or replacing, press the left mouse button with the pointer at one end of the text to select, and drag it to the other end. The text will become highlighted. To select a whole word, double click. Double clicking and then dragging the mouse will select a number of words. Similarly, it can select a whole line or a number of lines by triple clicking or triple clicking and dragging. To delete the selected text, press delete or backspace. To replace it, begin typing.

Selecting Text's Data structure:

```
/* Results passed back to the convert proc processing an INSERT_SELECTION
  request, by getInsertSelection when the selection to insert has been
  received and processed */
enum      insertResultFlags      {INSERT_WAITING,      UNSUCCESSFUL_INSERT,
SUCCESSFUL_INSERT};

/* Actions for selection notify event handler upon receiving confirmation
  of a successful convert selection request */
enum              selectNotifyActions              {UNSELECT_SECONDARY,
REMOVE_SECONDARY,EXCHANGE_SECONDARY};

/* temporary structure for passing data to the event handler for completing
  selection requests (the hard way, via xlib calls) */
typedef struct {
   int action;
   XtIntervalId timeoutProcID;
  Time timeStamp;
  Widget widget;
  char *actionText;
  int length;
} selectNotifyInfo;
```

To select a rectangle or column of text, hold the Ctrl key while dragging the mouse. Rectangular selections can be used in any context that normal selections can be used, including cutting and pasting, filling, shifting, dragging, and searching. Operations on rectangular selections automatically fill in tabs and spaces to maintain alignment of text within and to the right of the selection. Note that the interpretation of rectangular selections by fill paragraph is slightly different from that of other commands.

The middle mouse button can be used to make an additional selection (called the secondary selection). As soon as the button is released, the contents of this selection will be copied to the insert position of the window where the mouse was last clicked (the destination window). This position is marked by a caret shaped cursor when the mouse is outside of the destination window. If there is a (primary) selection, adjacent to the cursor in the window, the new text will replace the selected text. Holding the shift key while making the secondary selection will move the text, deleting it at the site of the secondary selection, rather than copying it.

Selected text can also be dragged to a new location in the file using the middle mouse button. Holding the shift key while dragging the text will copy the selected text, leaving the original text in place. Holding the control key will drag the text in overlay mode.

Normally, dragging moves text by removing it from the selected position at the start of the drag, and inserting it at a new position relative to the mouse. Dragging a block of text over existing characters displaces the characters to the end of the selection. In overlay mode, characters which are occluded by blocks of text being dragged are simply removed. When dragging non-rectangular selections, overlay mode also converts the selection to rectangular form, allowing it to be dragged outside of the bounds of the existing text.

```
/* Menu modes for SGI_CUSTOM short-menus feature */
enum menuModes {FULL, SHORT};

/*Short menu mode is a special feature for the SGI system distribution. To make toggling
short-menus mode faster (re-creating the menus was too slow), a list is kept in the
window data structure of items to be turned on and off. Initialize that list and give the
menu creation  routines a pointer to the window on which this list is kept. This is
(unfortunately) a global variable to keep the interface simple for the mainstream case.
 */
  ShortMenuWindow = window;
```

4.2 Finding And Replacing Text

The Search menu contains a number of commands for finding and replacing text. The Find... and Replace... commands present dialogs for entering text for searching and replacing. These dialogs also allow choosing whether the search to be sensitive to upper and lower case, or whether to use the standard Unix pattern matching characters (regular expressions). Searches begin at the current text insertion position.

```
/*
Checks whether a selection spans multiple lines. Used to decide on the default scope for
replace dialogs.
*/
static int selectionSpansMultipleLines(WindowInfo *window)
{
  int selStart, selEnd, isRect, rectStart, rectEnd, lineStartStart,
  lineStartEnd;
  int lineWidth;
  textDisp *textD;

  if (!BufGetSelectionPos(window->buffer, &selStart, &selEnd, &isRect,
          &rectStart, &rectEnd))
      return FALSE;
```

This is kind of tricky. The perception of a line depends on the line wrap mode being used. So in theory, we should take into account the layout of the text on the screen. However, the routine to calculate a line number for a given character position TextDPosToLineAndCol) only works for displayed lines, so we cannot use it. Therefore, we use this simple heuristic:

- If a newline is found between the start and end of the selection, it has to have a multi-line selection.

Search for "searchString" in "window", and select the matching text in the window when found (or beep or put up a dialog if not found). If "continued" is TRUE and a prior incremental search starting position is recorded, search from that original position, otherwise, search from the current cursor position.

```
*/

int SearchAndSelectIncremental(WindowInfo *window, int direction,
        const char *searchString, int searchType, int searchWrap, int continued)
/*
```

Called when user types in the incremental search line. Redoes the search for the new search string.

```
*/

static void iSearchTextValueChangedCB(Widget w, WindowInfo *window,
        XmAnyCallbackStruct *callData)

/*
```

Search and replace using previously entered search strings (from dialog or selection).

```
*/
int ReplaceFindSame(WindowInfo *window, int direction, int searchWrap)


/*
```

```
Search for string "searchString" in window "window", using algorithm "searchType"
and direction "direction", and replace it with "replaceString" Also adds the search and
replace strings to the global search history.
*/
int SearchAndReplace(WindowInfo *window, int direction, const char *searchString,
        const char *replaceString, int searchType, int searchWrap)
```

```
/* If the first/last character of `searchString' is a "normal word character" (not contained
in `delimiters', not a whitespace) then limit search to strings, who's next left/next right
character is contained in `delimiters' or is a whitespace or text begin or end. If the
first/last character of `searchString' itself is contained in delimiters or is a white space,
then the neighbour character of the first/last character will not be checked, just a simple
match will suffice in that case.
*/

static int searchLiteralWord(const char *string, const char *searchString, int caseSense,
int direction, int wrap, int beginPos, int *startPos, int *endPos,const char * delimiters);
```

Find Again and Replace Again repeat the last find or replace command without prompting for search strings. Find Again, then Replace Again if the highlighted string should be replaced, or Find again to go to the next string. Find Selection searches for the text contained in the current primary selection for Selecting Text). The selected text does not have to be in the current editor window, it may even be in another program. For example, if the word dog appears somewhere in a window on your screen, and you want to find it in the file you are editing, select the word dog by dragging the mouse across it, switch to your Editor window and choose Find Selection from the Search menu.

Find Incremental is yet another variation on searching, where every character typed triggers a new search. Incremental searching is generally the quickest way to find something in a file, because it gives you the immediate feedback of seeing how your search is progressing, so you never need to type more than the minimally sufficient search string to reach your target.

```
/*
        Attach callbacks to the incremental search bar widgets. This also fudges up the
translations on the text widget so Shift+Return will call the activate callback (along with
Return and Ctrl+Return). It does this because incremental search uses the activate
callback from the text widget to detect when the user has pressed Return to search for the
next occurrence of the search string, and Shift+Return, which is the natural command for
a reverse search does not naturally trigger this callback.
*/
void SetISearchTextCallbacks(WindowInfo *window)
```

4.3 Searching Backwards

Holding down the shift key while choosing any of the search or replace commands from the menu will search in the reverse direction. Set the search direction using the buttons in the search dialog, may find it a bit confusing that Find Again and Replace Again don't continue in the same direction as the original search.

4.4 Selective Replacement

To replace only some occurrences of a string within a file, choose replace... from the Search menu, enter the string to search for and the string to substitute, and finish by pressing the find button. When the first occurrence is highlighted, use either replace again to replace it, or find again to move to the next occurrence without replacing it, and continue in such a manner through all occurrences of interest.

To replace all occurrences of a string within some range of text, select the range, choose Replace... from the Search menu, type the string to search for and the string to substitute, and press the "R. in Selection" button in the dialog Note that selecting text in the Replace... dialog will unselect the text in the window.

4.5 Cut and Paste

The easiest way to copy and move text around file or between windows is to use the clipboard, an imaginary area that temporarily stores text and data. The Cut command removes the selected text from file and places it in the clipboard. Once text is in the clipboard, the Paste command will copy it to the insert position in the current window. For example, to move some text from one place to another, select it by dragging the mouse over it, choose cut to remove it, click the pointer to move the insert point where the text is inserted, then choose Paste to insert it. Copy copies text to the clipboard without deleting it from file. It can also use the clipboard to transfer text to and from other motif programs and x-programs which make proper use of the clipboard.

There are many other methods for copying and moving text within Editor windows and between Editor and other programs. The most common method is clicking the middle mouse button to copy the primary selection (to the clicked position). Copying the selection by clicking the middle mouse button in many cases is the only way to transfer data to and from many X programs.

Holding the Shift key while clicking the middle mouse button moves the text, deleting it from its original position, rather than copying it. Other methods for transferring text include secondary selections, primary selection dragging, keyboard-based selection copying, and drag and drop.

4.6 Mouse Use

Mouse-based editing is what Editor is all about, and uses the more advanced features like secondary selections and primary selection dragging. This is done by adequately with just the left mouse button: Clicking the left button moves the cursor. Dragging with the left button makes a selection. Holding the shift key while clicking extends the existing selection, or begins a selection between the cursor and the mouse. Double or triple clicking selects a whole word or a whole line. "Selecting Text", which explains the terminology of selections, that is, what is meant by primary, secondary, rectangular, etc.

• Button And Modifier Key Summary

General mouse buttons and modifier keys:

Buttons

Button 1 (left): Cursor position and primary selection

Button 2 (middle): Secondary selections, and dragging and copying the primary selection

Button 3 (right) :Quick-access programmable menu and pan scrolling

Modifier keys

Shift : On primary selections, (left mouse button):
Extends selection to the mouse pointer
On secondary and copy operations, (middle):
Toggles between move and copy

Ctrl : Makes selection rectangular or insertion columnar

Alt* : (on release) Exchange primary and secondary selections.

LEFT MOUSE BUTTON

The left mouse button is used to position the cursor and to make primary selections.

Click:	Moves the cursor
Double Click:	Selects a whole word
Triple Click:	Selects a whole line
Quad Click:	Selects the whole file

Shift Click: Adjusts (extends or shrinks) the selection, or if there is no existing selection, begins a new selection between the cursor and the mouse.

Drag: Selects text between where the mouse was pressed and where it was released.

Ctrl + Drag: Selects rectangle between where the mouse was pressed and where it was released.

RIGHT MOUSE BUTTON

The right mouse button posts a programmable menu for frequently used commands.

Click/Drag: Pops up the background menu (programmed from preference default settings → customize menus→ window background).

Ctrl+Drag: Scrolls the window both vertically and horizontally (Pan scrolling).

MIDDLE MOUSE BUTTON

The middle mouse button is for making secondary selections, and copying and dragging the primary selection.

Click: Copies the primary selection to the clicked position.

Shift+Click : Moves the primary selection to the clicked position, deleting it from its original position.

Drag : 1) Outside of the primary selection:
Begins a secondary selection.
2) Inside of the primary selection:
Moves the selection by dragging.

Ctrl+Drag : 1) Outside of the primary selection:
Begins a rectangular secondary selection.
2) Inside of the primary selection:
Drags the selection in overlay mode.

When the mouse button is released after creating a secondary selection:

No Modifiers: If there is a primary selection, replaces it with the secondary selection. Otherwise, inserts the secondary selection at the cursor position.

Shift: Move the secondary selection, deleting it from its original position. If there is a primary selection, the move will replace the primary selection with the secondary selection. Otherwise, moves the secondary selection to the cursor position.

Alt* : Exchange the primary and secondary selections.

While moving the primary selection by dragging with the middle mouse button:

Shift: Leaves a copy of the original selection in place rather than removing it or blanking the area.

Ctrl: Changes from insert mode to overlay mode

Escape : Cancels drag in progress.

Normally in overlay mode, dragging moves text by removing it from the selected position at the start of the drag, and inserting it at a new position relative to the mouse. When you drag a block of text over existing characters, the existing characters are displaced to the end of the selection. In overlay mode, characters which are occluded by blocks of text being dragged are simply removed. When dragging non-rectangular selections, overlay mode also converts the selection to rectangular form, allowing it to be dragged outside of the bounds of the existing text.

Mouse buttons 4 and 5 are usually represented by a mouse wheel nowadays. They are used to scroll up or down in the text window.

The Alt key may be labeled meta or compose-character on some keyboards. Some window managers bind combinations of the Alt key and mouse buttons to window manager operations. In Editor, alt is only used on button release, so regardless of the window manager bindings for Alt-modified mouse buttons, it can still do the corresponding Editor operation by using the Alt key after the initial mouse press, so that Alt key is held while the release of mouse button.

• Keyboard Shortcuts

Most of the keyboard shortcuts in Editor have the pull-down menus. However, there are more. These include; dialog button shortcuts; menu and dialog mnemonics; labeled keyboard keys, such as the arrows, page-up, page-down, and home; and optional Shift modifiers on accelerator keys, like [Shift]Ctrl+F.

• Menu Accelerators

Pressing the key combinations shown on the right of the menu items is a shortcut for selecting the menu item with the mouse. Some items have the shift key enclosed in brackets, such as [Shift] Ctrl + F. This indicates that the shift key is optional. In search commands, including the shift key reverses the direction of the search. In Shift commands, it makes the command shift the selected text by a whole tab stop rather than by single characters.

• Menu Mnemonics

Pressing the Alt key in combination with one of the underlined characters in the menu bar pulls down that menu. Once the menu is pulled down, typing the underlined characters in a menu item (without the Alt key) activates that item. Menu pulled down, also use the arrow keys to select menu items, and the Space or Enter keys to activate them.

• Keyboard Shortcuts Within Dialogs

One button in a dialog is usually marked with a thick indented outline. Pressing the Return or Enter key activates this button.

All dialogs have either a Cancel or Dismiss button. This button can be activated by pressing the Escape (or Esc) key. Pressing the tab key moves the keyboard focus to the next item in a dialog. Within an associated group of buttons, the arrow keys move the focus among the buttons. Shift+Tab moves backward through the items.

Most items in dialogs have an underline under one character in their name. Pressing the Alt key along with this character, activates a button as if it had pressed with the mouse, or moves the keyboard focus to the associated text field or list. It can select items from a list by using the arrow keys to move the selection and space to select. In file selection dialogs, also it can type the beginning characters of the file name or directory in the list to select files

- **Labeled Function Keys**

The labeled function keys on standard workstation and PC keyboards, like the arrows, and page-up and page-down, are active in editor.
Holding down the control key while pressing a named key extends the scope of the action that it performs. For example, Home normally moves the insert cursor the beginning of a line.
Ctrl+Home: moves it to the beginning of the file. Backspace deletes one character, Ctrl+Backspace: deletes one word.
Holding down the shift key while pressing a named key begins or extends a selection. Combining the shift and control keys combines their actions. For example, to select a word without using the mouse, position the cursor at the beginning of the word and press.

Ctrl+Shift+RightArrow: The Alt key modifies selection commands to make the selection rectangular.
Under X and Motif, there are several levels of translation between keyboard keys and the actions they perform in a program. "X-Resources" has more information on this related to the operation performed by the Unix editor here. Because of all of it's configurability, and since keyboards and standards for the meaning of some keys vary from machine to machine, the mappings may be changed from the defaults listed resources available in the motif or under X-resources.

- **SPECIALTY KEYBOARDS**

On machines with different styles of keyboards, generally, text editing actions are properly matched to the labeled keys, such as Remove, Next-screen, etc. For different key bindings, it can be instantiated by the section titled "Key Binding" under the customizing heading in the Help menu of Motif / X resources.

- **Shifting And Filling**

SHIFT LEFT, SHIFT RIGHT

For shifting blocks of text it is also useful for other tasks, such as creating indented paragraphs.

To shift a block of text one tab stop to the right, select the text, then choose Shift Right from the Edit menu. Note that the accelerator keys for these menu items are Ctrl+9 and Ctrl+0, which correspond to the right and left parenthesis on most keyboards. As adjusting the text in the direction pointed to by the parenthesis character. Holding the Shift key while selecting either Shift Left or Shift Right will shift the text by one character.

It is also possible to shift blocks of text by selecting the text rectangularly, and dragging it left or right (and up or down as well). Using a rectangular selection also causes tabs within the selection to be recalculated and substituted, such that the non-whitespace characters remain stationary with respect to the selection.

- **Filling**

Text filling using the Fill Paragraph command in the Edit menu is one of the most important concepts in editor.

In plain text files, unlike word-processor files, there is no way to tell which lines are continuations of other lines, and which lines are meant to be separate, because there is no distinction in meaning between newline characters which separate lines in a paragraph, and ones which separate paragraphs from other text. This makes it impossible for a text editor to tell parts of the text which belong together as a paragraph from carefully arranged individual lines.

In continuous wrap mode[7] lines automatically wrap and unwrap themselves to line up properly at the right margin. In this mode, it simply omit the new lines within paragraphs and let editor make the line breaks as needed. Unfortunately, continuous wrap mode is not appropriate in the majority of situations, because files with extremely long lines are not common under Unix and may not be compatible with all tools, and because it can't achieve effects like indented sections, columns, or program comments, and still take advantage of the automatic wrapping.

Without continuous wrapping, paragraph filling is not entirely automatic. Auto-Newline wrapping keeps paragraphs lined up as soon as typed, but once entered, editor can no longer distinguish newlines which join wrapped text, and new lines which must be preserved. Therefore, editing in the middle of a paragraph will often leave the right margin messy and uneven.

Since editor can't act automatically to keep text lined up, there is a need to tell it explicitly where to operate, and that is what Fill Paragraph is for. It arranges lines to fill the space between two margins, wrapping the lines neatly at word boundaries. Normally, the left margin for filling is inferred from the text being filled. The first line of each paragraph is considered special, and its left indentation is maintained separately from the remaining lines (for leading indents, bullet points, numbered paragraphs, etc.). Otherwise, the left margin is determined by the furthest left non-whitespace character. The right margin is either the Wrap Margin, set in the menu (by default, the right edge of the window).

There are three ways to use Fill Paragraph. The simplest is, while typing text, and there is no selection, simply select Fill Paragraph (or type Ctrl+J), and editor will arrange the text in the paragraph adjacent to the cursor. A paragraph, in this case, means an area of text delimited by blank lines.
The second way to use Fill Paragraph is with a selection. It select a range of text and then chose Fill Paragraph, all of the text in the selection will be filled. Again, continuous text

between blank lines is interpreted as paragraphs and filled individually, respecting leading indents and blank lines.

The third, and most versatile, way to use Fill Paragraph is with a rectangular selection. Fill Paragraph treats rectangular selections differently from other commands. Instead of simply filling the text inside the rectangular selection, editor interprets the right edge of the selection as the requested wrap margin. Text to the left of the selection is not disturbed (the usual interpretation of a rectangular selection), but text to the right of the selection is included in the operation and is pulled in to the selected region. This method enables you to fill text to an arbitrary right margin, without going back and forth to the wrap-margin dialog, as well as to exclude text to the left of the selection such as comment bars or other text columns.

- **File Format**

While plain text is probably the simplest and most interchangeable file format in the computer world, there is still variation in what plain text means from system to system. Plain text files can differ in character set, line termination, and wrapping.

While character set differences are the most obvious and pose the most challenge to portability, they affect editor only indirectly via the same font and localization mechanisms common to all X applications. Editor can not display Unicode text files, or any multi-byte character set.

The primary difference between an MS DOS format file and a Unix format file,is how the lines are terminated. Unix uses a single newline character. MS DOS uses a carriage-return and a newline. Editor can read and write both file formats, but internally, it uses the single character Unix standard. Editor auto-detects MS DOS format files[6] based on the line termination at the start of the file. Files are judged to be DOS format if all of the first five line terminators, within a maximum range, are DOS-style. To change the format in which editor writes a file from DOS to Unix or visa versa, use the Save As command and check or un-check the MS DOS Format button.

Wrapping within text files can vary among individual users, as well as from system to system. Both windows and macos make frequent use of plain text files with no implicit right margin. In these files, wrapping is determined by the tool which displays them. Files of this style also exist on Unix systems, despite the fact that they are not supported by all Unix utilities. To display this kind of file properly in Editor, select the wrap style that is called continuous.

The last and most format differences is the terminating newline. Some Unix compilers and utilities require a final terminating newline on all files they read and fail in various ways on files which do not have it. Vi and approximately half of Unix editors enforce the terminating newline on all files that they write. It makes the final terminating newline optional .

Chapter 5

PROBLEM SOLUTION

5.1. System Development Life Cycle

Similar to software development, an engineering work and a systematic approach can achieve a qualified deliverable, i.e. effective and efficient scheme. To make it simpler, a three-phase model is introduced.

5.1.1. Investigation

As at the beginning, most of the users find it really difficult to work on the non-user friendly unix shell and vi editor. Thus there is a need to develop an application that provides the user with an interface and utilities like that of windows explorer, so that the user can be relieved from using the baffling vi-editor and many other utilities which are rather difficult to operate. This being the primary reason, various other reasons also influenced the choice of the project.

5.1.2. Design

The objective of design is to construct an editor having the entire scheme by the information gathered by the analysis stage. The design include several components, including input attributes, model architecture, operation procedures and different types of operation to operate the different facilities.

5.1.3. Implementation

The aim of this stage is to translate the design into a workable system. The programming is the primary activity in the implementation stage. More over at the implementation label the all information gathered during the requirement analysis is taken care of to implement the unix explorer with the editor facilities. For implementation of it, especially for the graphical user interface the motif/x resources functionality is used.

5.2 The Client Server Method

The design for the application program has it's roots on the basis Client-Server Model which is the standard model for the network application server is a process waiting for a client process so that the server can do something for the client.

A typical scenario is as follows: -

- The server process is started on some computer system. It initializes itself, and then goes to sleep waiting for the client process to contact it requesting some service.
- A client process is started either on the same system or another system that is connected to the server system through a network.
- An interactive user entering a command to a time sharing system also initiates client processes. The client process sends a request across the network to the server requesting service of some form.
- When the server process has finished providing its service to the client, the server goes back to sleep, waiting for the next request to arrive.

The server process can be further subdivided into two types: -

5.2.1. Iterative Servers

When the server in a known, short time, can handle a client's request, the server process handles the requests itself. The term iterative server is also used to describe the server implementation that process one request at a time. The iterative server implementations, which are easier to build and understand, may result in poor performance because they make the clients wait for the service. A time of day service is typically handled in an iterative fashion[7].

5.2.2. Concurrent Servers

When the amount of the time to the service a request depends on the request itself, the server typically handles the request in concurrent fashion. These are called Concurrent Server. Simply it refers to the server handling multiple requests at the same .A concurrent server invokes another process to handle each time client requests that the original server process can go back to sleep, waiting for the next request. This type of server requires an OS that allow multiples processes to run at same time. The concurrent server applications are more difficult to design and build, but provide a better performance. The server handles[4] most client requests that deal with a file information in a concurrent fashion as the amount of processing required to each request depends on the size of the file.

5.3 Servers State Information

The information that a server maintains about the ongoing interaction with the clients are called Stateful Server. The servers that do not keep any state information are called Stateless Servers. Stateless Servers are also called Context Free Servers. The main motive behind the stateless server is the efficiency.

The state information allows the server to keep the information about what client request previously and to compute an incremental response as each new request arrives. However keeping the amount of information small in a server can reduce the size of the messages the client server exchange, and allow the server to respond quickly. By contrast, the motivation for the state reliability lies in the protocol reliability. State information in a server can become incorrect if the messages are lost, duplicated out of the order or if the client computer crashes or reboots. Therefore stateless servers are preferred in application, which require higher reliability.

The server program for this application suits in the category of Concurrent and Stateless Servers and his is the design employed for the server program in this case.

Now as the server and the client program have to communicate via a network so there has to be a protocol on which both of the processes should agree upon. In this case the protocol suits used is TCP/IP.

As we all know, when a client process wants to contact a server, the client must have a way of identifying the server that it wants to communicate. If the client knows the

32 -bit Internet address of the host on which the server resides can contact the host, but for identifying the particular server process, the client must know the port number of the server process. Both TCP and UDP use 16 -bit integer port number for the identification. For Example, every TCP/IP implementation that supports FTP assigns the well-known port of 21(decimal) to it TFTP assigned the UDP port of 69.

As the client program has to be on the windows system and the server on the unix, so some form of network programming should be incorporated in the basic design of the application. So, the following fig.5.3.1 represents the type of communication, which will persists between the client and the server.

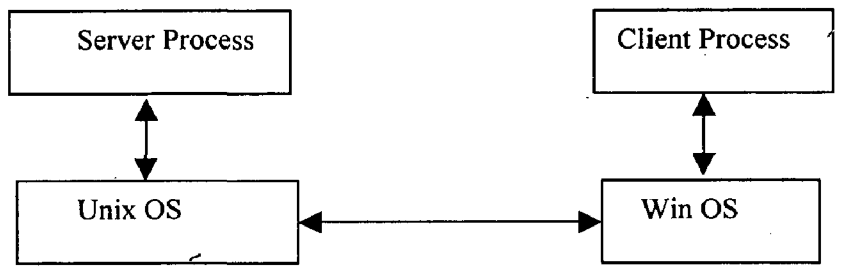


Fig: 5.3.1: (IPC between two processes on different systems.)

As the above design was formulated for this application, so for each and every client there should be a one unique server busy in fulfilling the request of that specific client only.

Also the server program should be run as soon as a request of a client arrived on the Unix system i.e., there should be a process on the Unix system waiting client's request to arrives soon as such request arrives, this process should invoke the server program so that the server program may handle the request. Another alternative is that of making the server program a daemon process running indefinitely on the Unix system. But in this case, the daemon process will be consuming a major time of the CPU and other resources, so this strategy was discarded and the former one was adopted because one such facility is provided by the unix system itself i.e.,-the inetd or the Internet Super server , a detailed description of which follows:

5.4 Internet Super server

5.4.1 Inetd Process

A server that uses either TCP or UDP can use this daemon process. It does not handle either the XNS or UNIX domain protocols.

5.4.2 Demon Process

This daemon provides two features:

- It allows single process (inetd) to be waiting to service multiple connection requests instead of one process for each potential service. This reduces the total number of processes in the system.
- It simplifies the writing of daemon processes to handle the request, since many of the startup details are handled by the inetd. This obviates the need for the actual process to execute the code to become daemon.

However, in that of inetd daemon as in fig.5.3.2 has to execute both a fork and an exec to envoke the actual server process, while a self-contained daemon that did everything itself only has to execute a fork to handle each request. This additional overhead, however, is worth the simplification of the actual server process and the reduction in the total number of processes in the system. The inetd process establishes itself as a daemon. It then reads the file /etc/inetd.conf to initialize itself. This text file specifies the services that the super server is to listen for, and what to do when a service request arrives.

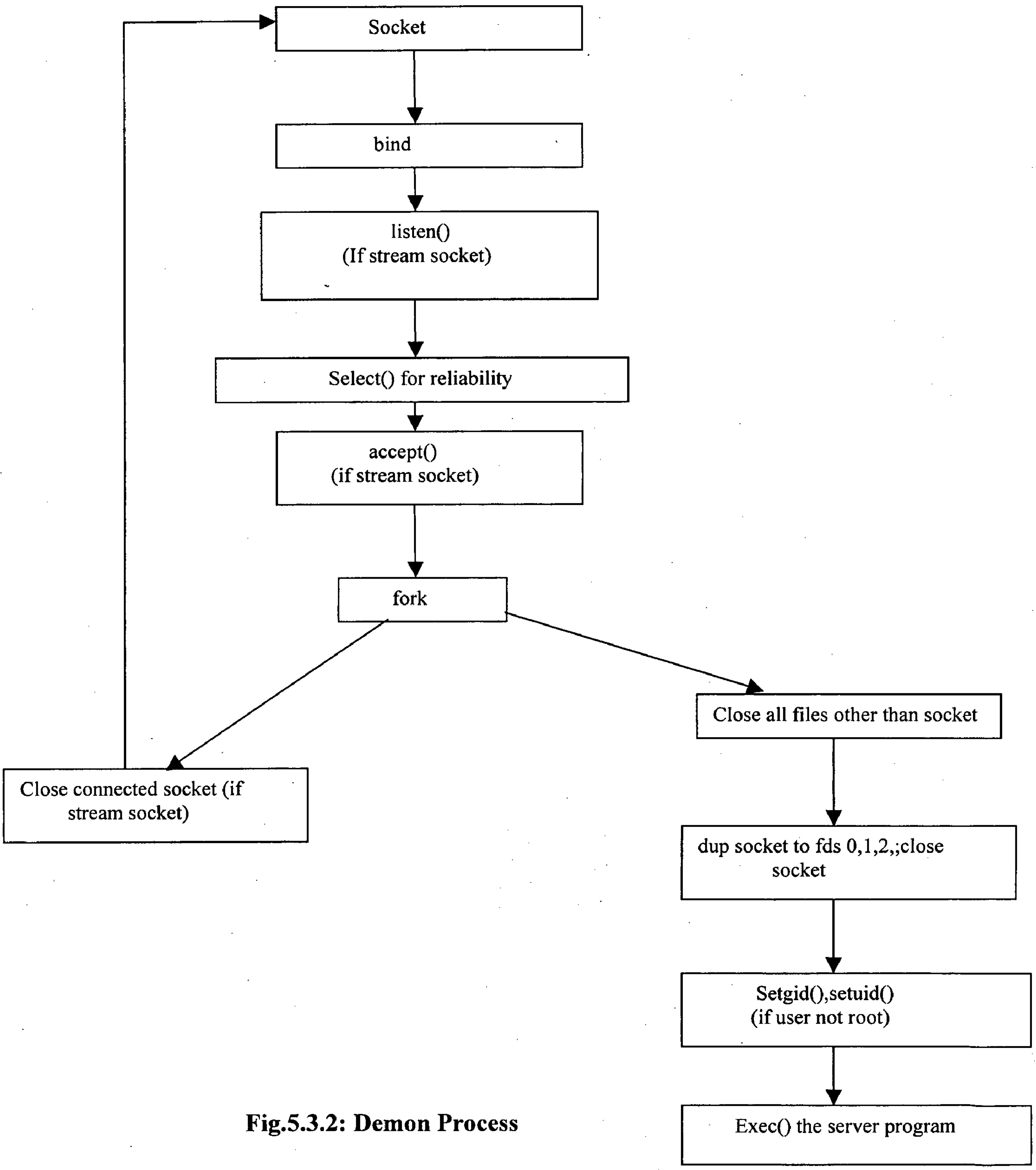

Socket
bind
listen()
(If stream socket)
Select() for reliability
accept()
(if stream socket)
fork
Close connected socket (if stream socket)
Close all files other than socket
dup socket to fds 0,1,2,;close socket
Setgid(),setuid()
(if user not root)
Exec() the server program

Fig.5.3.2: Demon Process

5.5 Steps Performed By Inetd.

- On startup it reads the /etc/inetd.conf file and creates a socket of the appropriate type (stream or datagram) for all the services specified in the file.

As each socket is created, a mind () is executed for every socket, specifying the well-known address for the server. This TCP or UDP port number is obtained by logging up the service-name field from the configuration file in the /etc/services file.

- For stream socket, a listen () is executed, specifying a willingness to receive connection on the socket and the queue length for incoming connections. This step is not done for datagram sockets.

- A select () is then executed, to wait for the first socket to become ready for reading. A stream socket is considered ready for reading when a connection request arrives for the socket. A datagram socket is ready for reading when datagram arrives. At this point the inetd daemon just waits for the select () system call to return.

- When a socket is ready for reading, if it is a stream socket, an accept() system call is executed to accept the connection.

- The inetd daemon forks and the child process handles the service request. The child closes all the file descriptors other than the socket descriptor that it is handling and then calls dup2 to cause the socket to be duplicated on the file descriptors 0,1 and 2. The original socket descriptor is then closed. Doing this , the only file descriptors that are open in the child are 0,1 and 2. It then calls getpwnam () to get the password file entry for the long - name that is specified in /etc/inetd.conf file . If this entry does not have a user ID of zero (the super user) then the child becomes the specified user by executing the setid() and setuid () system calls . Since the inetd process is executing with a user ID of zero, the child process inherits this user ID across he fork (), so it is able to become any user that it chooses) .The child process now does an exex() to execute the appropriate server-program to handle the request , passing the arguments that are specified in the configuration file.

If the socket is the stream socket, the parent process must close the connected socket. The parent goes back and executes the select() system call again, waiting for the next socket to become ready for reading.
The above steps are done if the wait-flag is nowait. If wait is specified then the steps done by the parent process are:

- First, after the fork () the parent saves the process ID of the child, so it can tell later when that specific child process terminates, by looking at the value returned by the wait() system call.

- Second, the parent disables the current socket from future selects by using the FD_CLR macro to turn off the bit in the fd_set structure that it uses for the select. This means that the child process takes over the socket until it terminates. Once the child process terminates, the parent process is notified by a SIGCLD signal, and the parent process's signal handler obtains the process ID of the terminating child and re-enables the select for the corresponding socket by using the FD_SET macro. When the child process terminates, the parent process is probably waiting for its select () system call to return. When a signal handler is invoked while a process is executing a "slow" system call, when the signal handler returns, the system call (the select in this case) returns with An error indication and an error no value of EINTR,. The inetd process recognizes this and executes the select () again. But by executing the system again, it can now wait for the socket corresponding to the child process that terminated. If the occurrence of the signal did not interrupt the select(), the parent would be able to wait for the socket corresponding to the terminating child process.

Since inetd is the process that does the accept () of a stream connection, the server that is invoked by inetd has to execute the getpeername() system call to obtain the address of the client. Servers that want to verify that the client is using a reserved port do this, for example. For datagram servers, the address of the client is returned to the server when it executes one of the receive() system calls.

Chapter 6

RESULT AND DISCUSSION

6.1 GUI of Unix Explorer

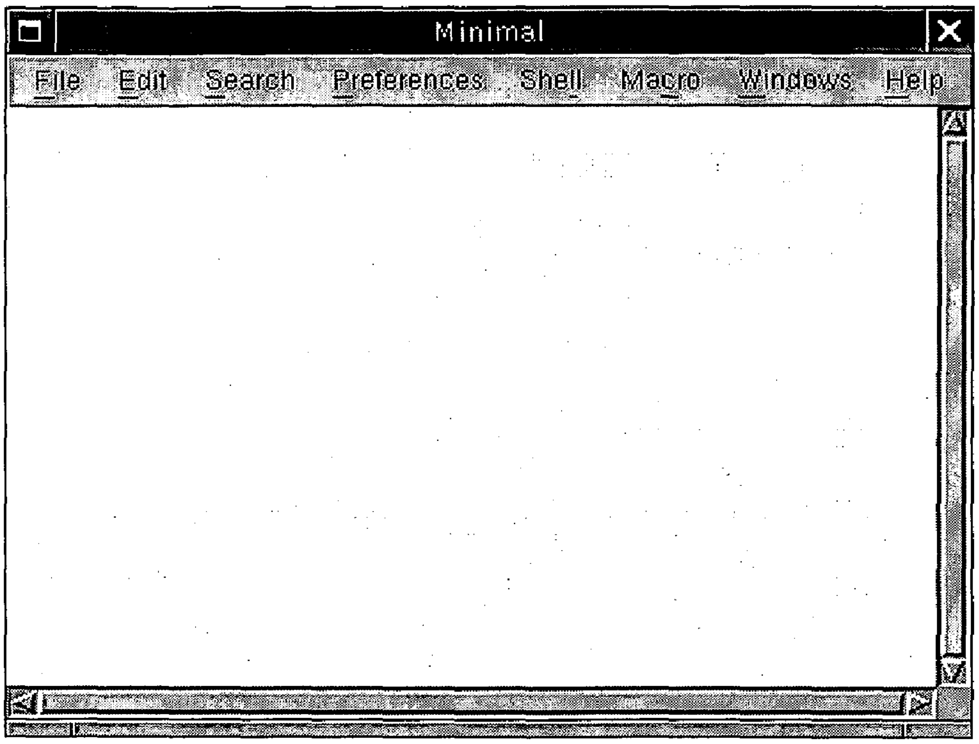
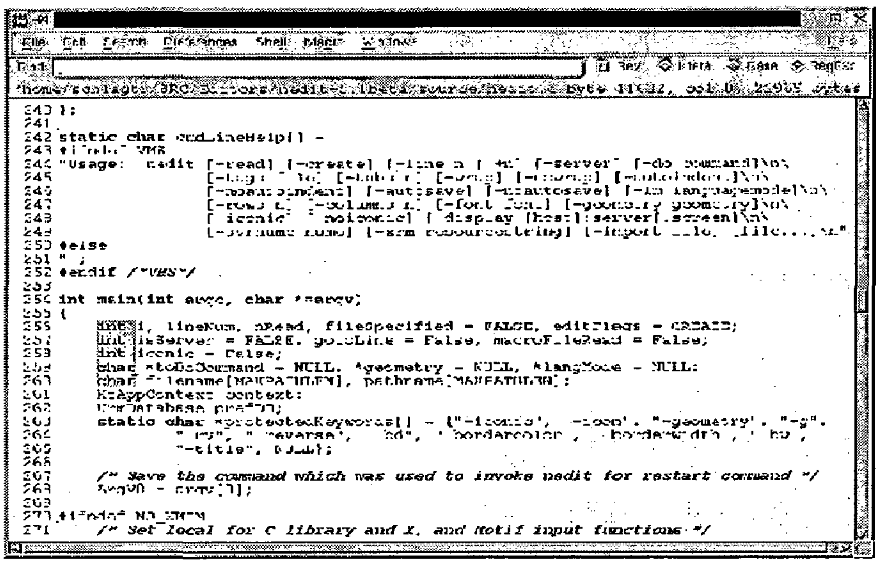


Fig.6.1.1: Graphical User Interface of Unix Explorer with Editor

Fig 6.1.1 shows the Graphical User Interface of the Unix Explorer along with editor having various operations. The Windows has the different facilities as the main menu like File, Edit, Search, Prefernces, Shell, Macro, Window and finally Help.

In this window the file menu has the operation like to create a new document and open the documents. It also has the various basic operations like, save, close, print and Exit etc. The Open option from the menu file acts as a small windows explorer for the unix account users, where the attributes of the file will be displayed.

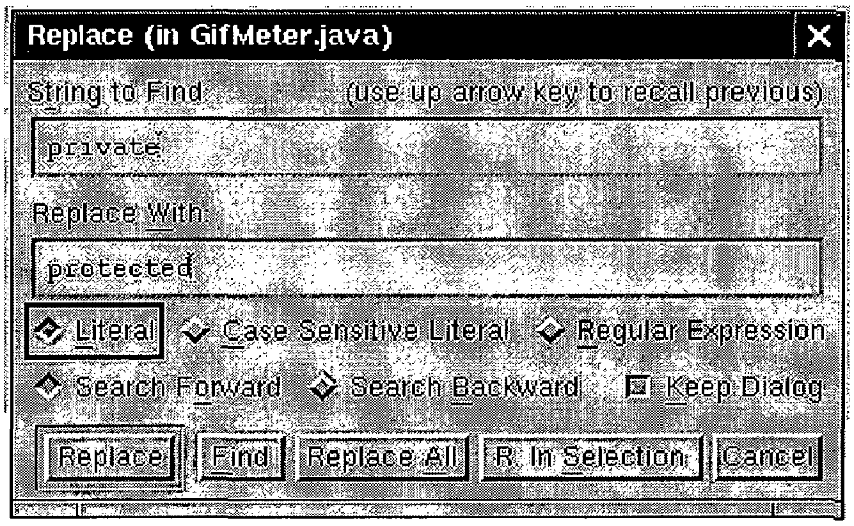
6.2 Unix explorer as an Editor



```
/home/[illegible]/SRC/[illegible]/nedit-5.1beta/source/nedit.c byte 11612, col 0, 22905 bytes
};

static char cmdLineHelp[] =
#ifdef VMS
"Usage:  nedit [-read] [-create] [-line n | +n] [-server] [-do command]\n\
               [-tags file] [-tabs n] [-wrap] [-nowrap] [-autoindent]\n\
               [-noautoindent] [-autosave] [-noautosave] [-lm languagemode]\n\
               [-rows n] [-columns n] [-font font] [-geometry geometry]\n\
               [-iconic] [-noiconic] [-display [host]:server[.screen]\n\
               [-svrname name] [-xrm resourcestring] [-import file] file...\n"
#else
" ";
#endif /*VMS*/

int main(int argc, char **argv)
{
    int i, lineNum, nRead, fileSpecified = FALSE, editFlags = CREATE;
    int isServer = FALSE, gotoLine = False, macroFileRead = False;
    int iconic = False;
    char *toDoCommand = NULL, *geometry = NULL, *langMode = NULL;
    char filename[MAXPATHLEN], pathname[MAXPATHLEN];
    XtAppContext context;
    XrmDatabase prefDB;
    static char *protectedKeywords[] = {"-iconic", "-icon", "-geometry", "-g",
            "-rv", "-reverse", "-bd", "-bordercolor", "-borderwidth", "-bw",
            "-title", NULL};

    /* Save the command which was used to invoke nedit for restart command */
    ArgV0 = argv[0];

#ifndef NO_XMIM
    /* Set local for C library and X, and Motif input functions */
```

Fig.6.2.1: Unix explorer as an Editor

In the fig.6.2.1,it shows that it can be even used to write the program also but it has been proposed for the facilities to compile and execute the specified code written.



String to Find: (use up arrow key to recall previous)

private

Replace With:

protected

Fig.6.2.2: String to Replace and Find

And the fig.6.2.2 shows and windows where one can do operation like replace and find string.

6.3 Strategies Used By the Client Program

The output of the ls-l command is received as a string and then it is passed on the basis of blank space and the new line character into the specified fields, which are then stored into a vector. A delimiter "///////" is stored into the vector when a new line character is received (because the names of the link files are broken into two or more parts separated by blank spaces). Afterwards, each string in the vector after the "///////" is checked for the starting character equal to 'd' or not. If the string starts with 'd' the corresponding entry is stored in the directory vector else it is stored in the file vector. The directory tree and file tree are constructed from the directory vector and the file vector respectively.

To make it possible for the user to edit a file, the contents of the file were retrieved from the server and then displayed to the user in a modal dialog containing a TextArea component. Any changes made by the user to the data in the TextArea were local to the file (on the client side). Then finally the file was overwritten on the server side.

6.4 Strategies Used By the Server Program

Since the Unix system stores the user password in an encrypted format, so the user authentication involves the encryption of the password entered by the user and then comparing it with the system stored encrypted password. In the Unix system the password may be encrypted by any of the several encryption algorithms chosen by their key value. The first two character of the system stored encrypted password determine the key of the encryption algorithm. This key is used to encrypt the user entered password with the help of the crypt () system call.

Remembering the events[6] in temporary buffers does backtracking of the last action performed by the user on the client side. The file which is cut or deleted is stored in /tmp/copy file for later recovery, if any. The complete pathnames, on which operation like cut, copy or delete was done, are stored in two temporary character arrays. If the user intends to undo the last action, then the above information helps in doing so.

6.5 Strategies Used For Communication

Reading single bytes from the socket-input stream until a '\0' is read does the transfer and retrieval of data on the client side. The reading of data by the client is continued until a string of length zero is read.

The server reads the data from the socket descriptor in blocking mode and reads the data byte by byte until a '\0' is read. The server writes to the socket descriptor in blocks of 1020 bytes.

Chapter 7

CONCLUSION

A GUI (Graphical User Interface) style Unix explorer having a text editor for programs and plain-text files. Rather than windows based text editors it is a familiar and comfortable environment. It provides the standard menu, dialog, editing, and mouse support, as well as all of the standard shortcuts to which the users of modern GUI based environments are accustomed. For users of older style Unix editors, here it has been also proposed to the world of mouse-based editing.

7.1 Problem and future work

One of the major drawbacks of using this is that the dialogs don't automatically get keyboard focus when they pop up. So, it's solution is that as most x-window managers allow you to choose between two categories of keyboard focus models: pointer focus, and explicit focus. Pointer focus means that as you move the mouse around the screen, the window under the mouse automatically gets the keyboard focus. Users who use this focus model should set "popups under pointer" in the default settings sub menu of the preferences menu in main GUI of the unix explorer. Users with the explicit focus model, in some cases, may have problems with certain dialogs, such as Find and replace.

The Backspace key doesn't work, or deletes forward rather than backward. While this is an X/Motif binding problem, and should be solved outside of resources in the Motif[9] virtual binding layer (or possibly xmodmap or translations). If you set the resource: nedit.remapDeleteKey to True, it will forcibly map the delete key to backspace. The default setting of this resource recently changed, so users who have been depending on this remapping will now have to set it explicitly (or fix their bindings).

The future function can be extended to this is that a command prompt can be created as a shell. Commands can be added to the shell commands menu (unix only) and don't output anything until they are finished executing. If the command output is directed to a dialog, or the input is from a selection, output is collected together and held until the command completes. And also there is a proposal for the operation like the compilation and execution facilities of the program written for the specified platforms. This can be operated on its own, or as a two-part client/server application. Client-Server mode is useful for integrating it with software development environments, mailers, and other programs.

REFERENCES:

[1].David Arnold. Marco: "A browser for open distributed systems". In DCE Developers Conference, Boston, Massachusetts, 1994.

[2].C. A. R. Hoare. "Communicating sequential processes".21(8): 666-677, August 1978.

[3].Jadwiga Indulska, Miriam Bearman, and Kerry Raymond. "A type management system for an ODP trader". In International Conference on Open Distributed processing (ICODP'93), pages 169-180, Berlin, Germany, 13-16 September 1993. North-Holland.

[4]. William E. Weihl. "Specifications of concurrent and distributed systems". chapter 3, pages 27-54. Addison-Wesley, 2nd edition, 1994.

[5]. David Arnold and Andy Bond. "An interaction framework for open distributed systems". International Conference on Distributed Platforms IDCP96, Dresden Germany, February 1996. IFIP/IEEE.

[6].Andrew Berry and Kerry Raymond. "The DSTC architecture model". http://internal.dstc.edu.au/, 1994.

[7].Andy Bond and David Arnold. "Visualizing service interaction in an open system". In first International Workshop on Services in Distributed and Networked Environments (SDNE'94), Prague, Czech Republic, 27-28 June 1994. IEEE Computer Society.

[8]. Jeffrey E. F. Friedl. "Mastering Regular Expressions" (c) 1997, O'Reilly & Associates ISBN 1-56592-257-3.s

[9].Motif User Guide ,Open Group Desktop Technology. http:// www.motifzone.net/

[10]. W.Richard Stevens.,"Unix Network Programming" Prentice-Hall India,ISBN-81-203-0749-6.