Indian Language Support for the Linux Operating System

Authors: J Patricia, K Ratheesh, V S Shenoi, G Sreepriya,
Timothy A Gonsalves[1] and Hema A Murthy
TeNeT Group, Department of Computer Science and Engineering,

Indian Institute of Technology, Madras, Chennai - 600 036
E-mail: indlinux-iitm@lantana.iitm.ernet.in

1 Introduction

Almost all the widely available software today is written and documented in English, and uses English as the medium to interact with users. This has the advantage of a common language of communication between developers, maintainers and users from different countries. In a country like India, an overwhelming majority of the population does not know English. Given this fact, availability of affordable native language software will play a crucial role in the process of taking the benefits of the "information revolution" to the marginalized sections of society and to achieve appropriate social use of information technology [1].

1.1 Overview of Native Language Support

Locale

Each country or language has its own set of native attributes. These attributes could include the country's cultural conventions, language-specific scripts (fonts), format of date and time, representation of numbers, currency-symbols etc. The formal description of these attributes together with associated translations targeted to a native language, constitute the *Locale* for the particular language or country.

Internationalization and Localization

Internationalization refers to the process by which a package is made aware of and is enabled to support multiple languages. This is a *generalization process*, by which the programs are not tied to a specific language for user-interaction and other locale-specific attributes and instead, use generic ways of doing the same.

Localization refers to the process, which provides the necessary information specific to a language or country to an internationalized package. This is a *particularization* process by which generic methods

[1] Currently with Nilgiri Networks, Ooty (on leave from IIT)

already implemented in an internationalized package are customized for a particular language or country.

The retrieval of the native attributes encompassed by *Locale* and usage of the same depends on the operating system and the programming environment.

1.2 Scope of the paper

For content generation in a particular language to take place, input and display mechanisms need to be in place for that language. Encoding, font and display supports need to be provided for the particular language. This support is largely dependent on the operating system and the programming environment. This paper addresses these aspects of the process of Native Language Support, for the Linux Operating System. The choice of Linux as the operating system has been motivated by the fact that Linux is a robust and stable operating system and is free.

The Linux operating system has two interfaces, namely the console and the X Window System. The RAM requirement for the console is about 4MB whereas for a minimal X Window system, the RAM requirement is 6-8 MB. The X Window System however has a user-friendly graphical interface.

This paper deals with the various tasks involved in the development of Native Language Support for the Linux operating system, both for the console as well as for the X Window System, with mutual compatibility. Developing a native language interface at an operating system level is a better proposition compared to developing it at an application level as the former enables all the applications running on top of the operating system to inherit the interface with no or minimal modification. An application developed in the console-based environment must work without requiring any modification in the X-environment. Further, once support has been developed for a particular language, the effort to enable any other Indian Language support should require changes to the configuration only. To meet this requirement, ISCII [2] in consonance with the Inscript keyboard layout has been used, so that the keyboard and sound mappings are uniform across all the Indian languages. ISCII includes ASCII as a subset.

In the subsequent sections, we address the issues involved in providing Indian language support for the Linux operating system, for the console and the X Window environment, and also suggest solutions for the same.

2 Multilingual support on Linux

Linux offers high flexibility for customization of the keyboard-input-display pipeline. This flexibility is offered both in the console and in the X Window System. At the input level, Linux offers the flexibility to manipulate the mapping tables that specify the keycodes/actions generated by the keys of the keyboard. The character consequently displayed depends on the font that is loaded. Linux provides easy mechanisms to load fonts for the console as well as for the X Window System. The high level of flexibility offered by Linux was an encouraging factor for the choice of Linux as the platform for our effort. The detailed mechanisms for the keyboard handling and display handling for the console and the X Window System are given in the subsequent sections.

2.1 The Console

The keyboard-input mechanism at the console level in Linux is as follows: When a key is pressed, the keyboard controller sends scancodes to the kernel keyboard driver. The keyboard driver sends whatever it receives to the application program when it is in scancode mode (for example, when the X Window System runs). Otherwise, it parses the stream of scancodes into keycodes, corresponding to keyPress or keyRelease events. These keycodes are sent to the application program when it is in keycode mode. Otherwise, these keycodes are looked up in a keymap and the character or string found there is transmitted to the application, or the action described there is performed [4].

At the console level, Linux allows loading of a font into the EGA/VGA character generator, with the options of specifying the screen-font map and/or application-character set mapping. Linux allows loading of keyboard translation tables as well. Linux provides a utility, *consolechars*, for doing the former task, and *loadkeys* for the latter [3]. Thus, at the console level, Linux allows a high level of flexibility of customizing the keyboard input as well as the display.

2.2 The X Window System

The keyboard-input mechanism under the X Window System is as follows: The X server generates a keyPress event when a key is pressed and a keyRelease event when the key is released. Under X, the keyboard gets attached to the window or the sub-window, which has the focus. All the keyboard events are directed to the window, which has the focus. The

X keyboard model [10] is broken into two layers: server-specific codes (called *keycodes*) which represent the physical keys, and server-independent symbols (called *keysyms*) which represent the letters or words that appear on the keys. The keycode is an integer with value between 8 and 255 and uniquely identifies the key. The keycodes have to be mapped to ASCII characters before they can be used.

The X server decides the keycode to be generated for a specific physical key. Each key, including the modifiers, has a unique keycode. Although the keycode generated for the common alphanumeric keys may be the same for many workstations, it is not guaranteed to be so. Therefore, applications do not use the raw keycode. Instead, the server-dependent keycode is translated to meaningful characters by a two-step process:

- The first step involves translating the keycode to a symbolic name, known as keysym.
- The second step involves converting the keysym to an ASCII text string that can be used for displaying and for saving in files or buffers.

The X server manages the translation of keycodes to keysyms. Users may use utilities such as "xmodmap" to alter the mapping of one or more keys. If required, a secondary mapping, in the form of keymap tables may be used by the client applications, to convert keycodes into keysymbols at the application level.

The display mechanism under the X Window System is as follows: Unlike text display terminals, X displays can show text in varying sizes and shapes. The X server retrieves the symbols from a font by indexing based on character code. There are mainly two types of fonts – Bitmap-based fonts and Outline-based or curve-based fonts. Outline-based fonts are becoming popular because of their scalability. True Type Fonts (TTF) are widely regarded as the best scalable fonts for low-resolution devices like displays. *Xfstt* is a freely available font server for True Type fonts and supplies fonts to the X window system display servers.

Thus, a high level of flexibility of customizing the keyboard input and display mechanisms is provided for the X Window System as well.

3 Issues in providing Native Language Support

The various issues involved in providing Native Language Support for the console and the X Window System environment of the Linux operating system are discussed in this section.

3.1 Character Width

Linux uses the *PC Screen Font* (PSF) format for display purposes for the console[3]. Neither the PSF format, nor the kernel modules implementing the display mechanism, viz., console and video drivers, supports variable width fonts. Moreover, the width of a font glyph is fixed at 8 pixels. This does not pose a problem for the English characters where even the glyph with the largest width, “m” can be represented legibly in 8 pixels. Also, the mean deviation of width of characters is very less in English, and hence the aesthetic appeal of characters is not affected because of the fixed width of the glyphs.

But, this is not the case with most of the Indian language scripts. A character Ë in Tamil or B in Malayalam cannot be legibly fit in 8-pixel width. If we reduce the font size to accommodate the widest glyph of a script into 8-pixels, then we loose out on the legibility of the narrower glyphs of the script. One option to overcome this problem would be to enable the kernel to support wider fixed-width fonts (say, all 16 pixels wide). But then, there are characters like S in Malayalam or ó in Tamil which are very narrow and it will look odd if characters with such a wide variation in width are displayed together in a screen, with the same width-allocation for all of them. So, an appropriate solution to overcome this problem would be to provide support for variable width fonts.

Similar issues arise in the context of the X Window System also. The virtual terminal (e.g. xterm) or the applications running in it support only English and other foreign languages, which do not demand variable-width glyphs.

The variable width of the glyphs has an implication in the X Window System as well, namely, the window width, and this also needs to be addressed. Normally, the virtual terminal and the applications running in it have a window width that accommodates 80 English characters. But when Indian language characters are also considered, the width of the window to accommodate any 80 Indian characters will be uncomfortably high. To overcome this problem, the width of the window should be set independent of the width of the widest character.

Further, under the X Window System, when Indian characters are involved, the *column-pixel* relation, which is uniform in the case of a window that supports only English, has to be modified to accommodate Indian scripts.

3.2 Vowel and Consonant Clusters

Another feature in Indian languages is the concept of vowel and consonant cluster formations. Consonant-vowel clusters of Indian languages result in non-trivial modified versions of the consonant.

Besides, the glyph ordering is also different in different languages. For example, consider the vowel modifier sounding like the English character *'e'*, applied to the consonant sounding: 'ka'. In Hindi, this takes the form: E + Î = ÊE ; But in Tamil it takes the form: è + ◌ = A. In some cases, the consonant may get sandwiched between the components of the vowel modifier. For example, consider the vowel modifier sounding like the English character *'O'*, applied to the consonant è (*Ka*). The resultant character is: ªè£. Editing operations also need to be taken care of while working with vowel modifiers. For example, if backspace is pressed at a character ªè£, it should give è, not ªè. A cursor-positioning request to go to the next column should place the cursor after ªè£, and not after ªè.

3.3 Internationalization of applications

If we want to really use the Native Language Support for the console or the X Window System, applications running on it, like Mail-utility, Editor, Web browser and command interpreter also need to be modified to give the user interface in a native language. This may include modification of the applications and also generation of the application-specific string tables.

3.4 Development of Font Files

At the console-level, Linux uses PC Screen Font (PSF) format, a bitmap-based font format, for the font files. PSF files for Indian Languages are not available unlike True Type Font (TTF) format files, which are widely available for most of the Indian languages. True Type font format is outline based unlike PSF, which is bitmap-based. So, we need to develop a mechanism to convert a True Type Font file for an Indian Language, to a PSF file.

4 Proposed Solution

The focus of the effort in this work has been to provide a unified approach to address these problems across all languages that require variable width font and have the concept of modifiers. The interface

allows co-existence of English and multiple native languages as well. In the effort undertaken, support has been extended for Indian languages. The 8-bit ISCII has been used as the encoding standard [2]. This facilitates the compatibility of console-based support and the X Windows support, with support for transliteration.

The requirements of an ideal solution are:

- A facility to switch between different keyboard layouts.
- Generic solution i.e., it should be language-independent. Once support is enabled for one language, it should be just a matter of altering/generating some configuration files for supporting other languages.
- The solution should address all the issues specified in Section 3 .
- Applications written for the console should work in the X Window System as well and vice-versa.
- For console, the support should be developed at the kernel level, so that all applications running on it can inherit the Native Language interface.

The ISCII standard is a choice for the encoding of characters of Indian languages. Using ISCII as the standard for the developmental effort for the console, the X Window System and the applications ensures mutual compatibility between the support provided for the console and for the X Window System. The ISCII standard in consonance with the Inscript keyboard facilitates uniform mapping and transliteration across all the major Indian languages.

4.1 Console-based solution – Kernel Modification

The display mechanism at the console is taken care by the console, the tty and the video device drivers in the kernel. Provision of variable-width font support in all respects, will require the modification of all these drivers. The solution adopted by us is to display multiple glyphs for a single character, code in order to display wider fonts. In this case, the glyphs are still of fixed size and a one-to-many mapping mechanism is introduced in the display pipeline. In this design, only the console and TTY drivers need to be changed.

Support for vowel–consonant clusters will require support for context-sensitive parsing at the I/O level for the console. The parsing rules would be different for different languages, and hence should be user-defined in order to make the kernel modifications to be language independent.

Thus, the kernel should be able to interpret and process the appropriate language-specific parse rules.

4.1.1 Multiglyph support

A *multiglyph* mapping will be of the general form:

- $C = G_1 G_2 G_3 \ldots G_n$

where C is a character code and G_i's are the glyph indices.

Display of characters

For displaying multiglyph characters, data structures are added in the kernel to store a one-to-many mapping of character codes to glyph indices. Device driver functions are provided to load user defined mutliglyph mappings. In the kernel display pipeline, code is inserted to index the character codes into the data structures and display the glyph codes corresponding to a character.

Deletion and Backspacing

One issue that needs to be taken care of while editing, is deletion and backspacing. When a multiglyph character is erased using the backspace key or delete key, all the glyphs corresponding to the glyph should be erased. For this, two bit-mappings are created in the kernel at the time of loading multiglyphs, and these bit-mappings assist in deciding the number of glyphs to be deleted.

Inserting a character

While a character is being inserted, all the glyphs that are displayed to the right of it need to be shifted. But the number of positions to shift depends upon the number of glyphs in the character to be inserted. Again, the bit-mappings for backspacing and deletion are used for determining the number of glyphs in the character and the shifting is done accordingly.

Processing Cursor Positioning

A function, gotoxy(), in the console driver is modified for supporting cursor positioning. Suppose that a request comes to position the cursor at coordinate <x, y>. The multigplyph patch does not affect the row numbers, but the column positioning needs to be modified. The modifications in gotoxy() refer to the delete and backspace bit-mappings and also the parserules to find out the actual column number that is required.

4.1.2 Parserule Support

A parserule will be of the general form:

- $[G_{1\text{-}1}G_{1\text{-}2}G_{1\text{-}3} \ldots G_{1\text{-}m}]\ C = G_{2\text{-}1}G_{2\text{-}2}G_{2\text{-}3} \ldots G_{2\text{-}n}$

where $G_{i\text{-}j}$'s are glyph indices and C is a character code.

The Forward and Reverse DFAs

The parse rules are used to construct two deterministic finite automata (DFA) in the kernel space. They are called the *forward_dfa* and the *reverse_dfa*.

Kernel data structures have been developed to implement the automata and device driver functions are also provided. The automata help in determining the cluster formations and related information. As an illustration, the forward and reverse DFAs for a set of four parse rules are given in Figure 1

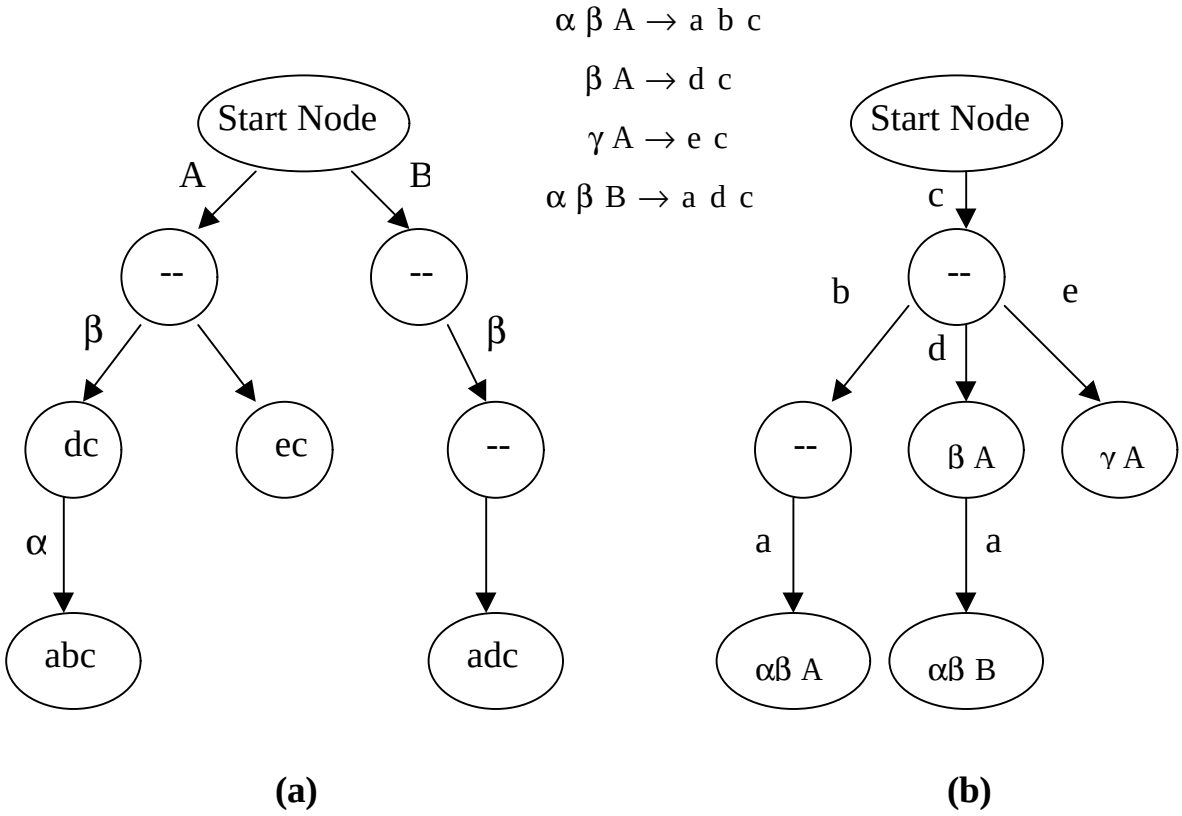

Figure 1: A set of parserules and (a): A forward DFA (b): A reverse DFA produced from these parserules using algorithm ConstructDFA.

Forward Parserule Matching

Forward parse rule matching is used while normal characters are being displayed. Suppose that there is a parse rule α β A = a b c. Also assume that the glyphs α β are already displayed. Now, when the character code corresponding to A is pressed, the forward DFA matching is initialized. If the rules are already loaded before, then the traversal will go in the order A → β → α, matching all successive characters, finally when it encounters a non-matching character, the data element in the last DFA node (abc) is returned.

Meanwhile, a count of the number of characters backspaced is kept. Finally, these characters, which are actually the LHS of the matched parserules are erased and then the glyphs in the data element returned, which are actually the RHS of the rule, are written at that position. Always the longest matching is taken.

The necessary code for this is added in do_con_write() function of the console driver.

Reverse Parserule Matching

Reverse parserule matching is used when a backspace character is pressed. Suppose that the cursor is positioned to the right of the glyphs a b c, and the backspace key is pressed. A reverse matching is initialized, and it will match in the order c → b → a. When finally a non-matching glyph is found, it will return the data element of the longest matched rule (if any), which is α β A. Now, the glyphs a b c are erased, and then the glyphs α β and character A are displayed. When glyphs are displayed, the forward matching is suspended, else it will again result in previous values, i.e., a b c. But, when A is to be displayed, forward matching is enabled, but limited to a single character, that is to A itself. This is because, A may be having a multimap defined in the form of a rule, like A = x y which we would like to match and process. But we don't want to match more than one character because then it will match the rule completely, and will return a b c again.

The code for implementation of this is included in the do_con_write() function in the console driver.

Also, appropriate modifications are made in the behavior of insertion flag, line-wrapping information and in cursor positioning functions. A complete flow diagram of the console I/O process in the new design is given in Figure 2

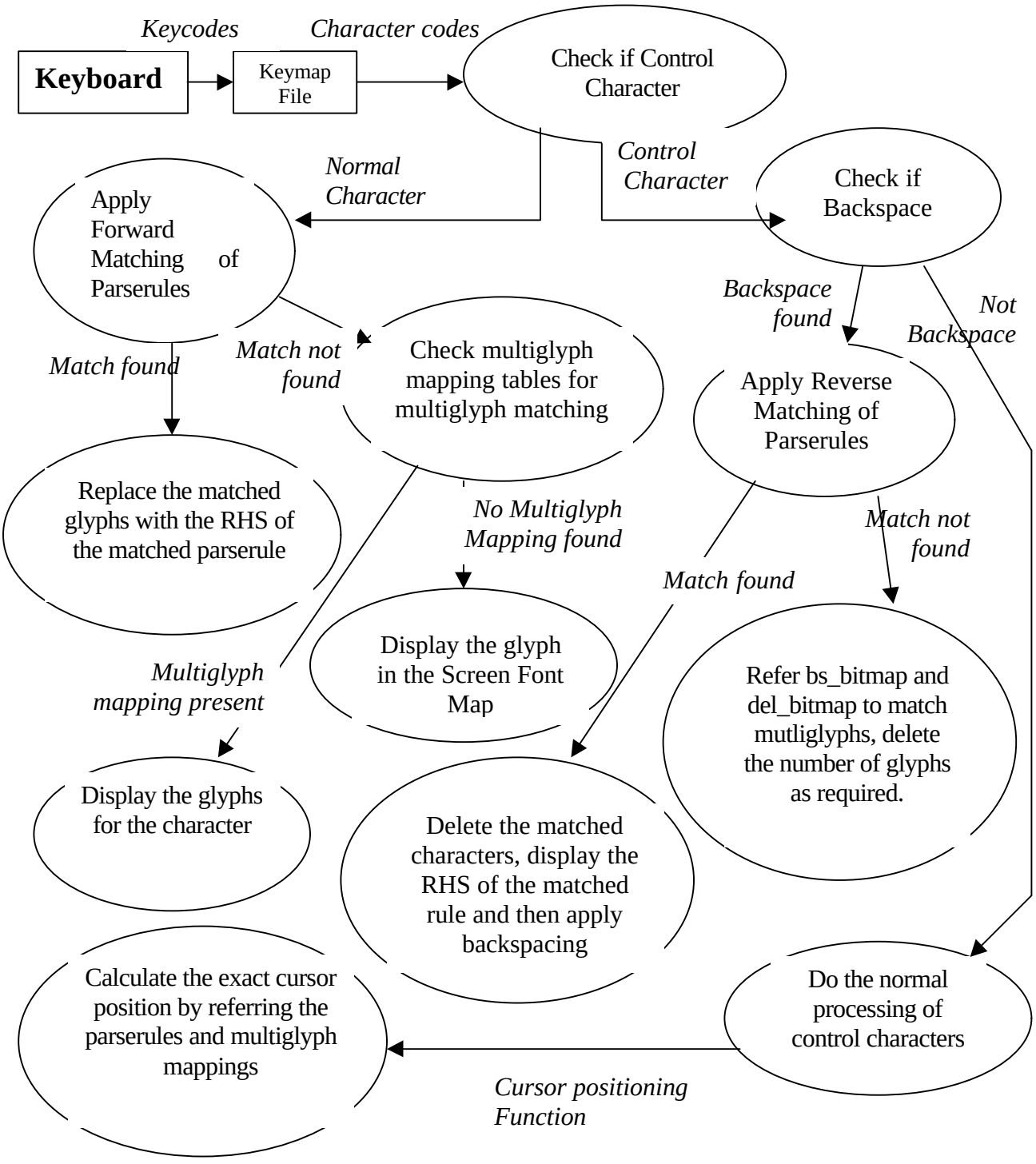

Figure 2: Console I/O Flow diagram in the new design

4.1.3 Utilities

Two utilities have been developed to enable the user to load the multimap and parserules into the kernel space. This section describes the usage of these utilities.

Loadmultimap

The *loadmultimap* utility can be used to load a *multimap* file into the kernel space. The usage of this utility is as follows:

- loadmultimap <*multimap* filename>

A multimap file should be a text file where each line is of the form:

- C G_1 G_2 G_3 ... G_n

where C is the character code in the range [0,256) and G_i's are glyph indices in the range [0,512). Both of them can be in decimals or in hexadecimals (prefixed with 0x).

Loadparserules

The *loadparserules* utility can be used to load a *parserule* file into the kernel space.
The usage of this utility is as follows:

- loadparaserules <*parserule* filename>

A parserule file should be a text file where each line is of the form

- $[G_{1\text{-}1}G_{1\text{-}2}G_{1\text{-}3} \ldots G_{1\text{-}m}]$ C = $G_{2\text{-}1}G_{2\text{-}2}G_{2\text{-}3} \ldots G_{2\text{-}n}$

where G_{ij}'s are the glyph indices in the range [0,256) and C is a character code in the range [0,512). Both can be given in decimals or hexadecimals.

4.2 X Window System based solution: Virtual Terminal – "iitm-term"

The X Window System based solution is to provide a virtual terminal similar to xterm, which supports Indian scripts in addition to the normal English script. The work has been based on the design and implementation of rxvt-idev [9], which is an extension of RXVT-2.4.5. A library providing the necessary functions to handle the keyboard layout and parse rules for each supported language is maintained for this purpose. RXVT 2.6.2, a virtual terminal has been extended to support the Indian scripts.

4.2.1 Indian language support library: "Libind"

To work in Indian languages, in addition to loading the font, facility to load the necessary keyboard mapping and language dependent parse rules is required. REC Trichy, in collaboration with IIT Kanpur has provided a library that provides support for Indian languages as part of the development of rxvt-idev [9]. Bug fixing and enhancements have been made to the library and support for a minimal and an extended set of parserules has been provided. The keyboard mapping defines the ISCII codes that are to be generated for a sequence of ASCII codes from the keyboard. The parse rules define the font glyphs that are to be displayed for a sequence of ISCII codes. The library has been provided with API for all these tasks. This consists of the following files and functions, which can be used to incorporate Indian script support in an application.

Files:

- "libind.conf" : This file is used as a configuration file. This holds general information such as the path, desired default keyboard layout, alternative keyboard layout and the English font to be used. It also holds language specific settings for each language. When we want support for a new language, appropriate entries in this file have to be made.

- Fontmap files: The Libind uses the Fontmap files to map ISCII characters to their equivalent font codes. This mapping forms the parse rules for the language. It defines the ISCII character sequence along with the resultant non-trivial glyph(s). The fontmaps are specific to the font and the language. The user can create his own Fontmap and place it in the **maps** directory (Libind searches in this directory), under the name "<language>.map", where <language> is the language for which the Fontmap is meant and access it from his application by calling "indian_init" function with the language parameter set to <language>.

- Keymap files: The Libind uses the keymap files to define the keyboard layout. This maps the ASCII code of the keys to the desired ISCII characters. The keymaps for the standard inscript and the phonetic keyboard layouts are provided in the **keymap** directory (Libind searches in this directory).

Functions:

- Indian_init: This function reads a font map table which maps ISCII syllables to their equivalent font codes and returns the size of the font map table and the font for the language.

- Iscii2font: This function converts the input ISCII string into its equivalent font codes.

- Readkeymap: This function reads the keymap file, which maps an ASCII character to its ISCII equivalent.

- Ins2iscii: This function converts a string of ASCII characters to a string of equivalent ISCII characters as defined by the Inscript keymap.

- iitk2iscii: This function converts a string of ASCII characters to a string of equivalent ISCII characters as defined by the phonetic keymap.

4.2.2 Virtual Terminal: “iitm-term”

To work in the X-Windows, a virtual terminal called ``iitm-term'' has been developed, which provides support for Indian scripts. It can display two different fonts simultaneously (i.e. a normal English font as well as one of the supported Indian language fonts).

For development of iitm-term, RXVT-2.6.2 was chosen as the base terminal and it has been extended to support the ISCII data and the Indian scripts. The design is based on the design and implementation of rxvt-idev [9], which is an extension of RXVT-2.4.5. The rxvt-idev was able to display English and Indian Language characters. But it was found to have a lot of design and implementation flaws and exhibited inconsistent behaviour and frequent crashes. Our effort has been oriented towards addressing the major problems that we observed: frequent crashes, inconsistent behaviour for events like screen-refresh, backspace, fixed window-width, and some event handling problems pertaining to the X Window System.

In the following paragraphs we give an overview of the design of rxvt-idev and the modifications that have been made to it. The modified terminal is referred to as the “iitm-term”.

The major modifications have been:

- Enhancement of the data structure for the rendering information.

- Synchronization of the handling of some of the X-events taking into account the computations involved when Indian languages are used.
- Enhancement of the process of screen-refresh.
- Support for window resizing.

RXVT was chosen because it is an 8 bit clean, color xterm replacement that uses significantly less memory than a conventional xterm. The display is constructed using two sets of information: One structure called "screen" and the other called "drawn". The "drawn" structure holds the actual data that is presently displayed. The "screen" structure holds the information that is to be copied on to the "drawn" structure to get displayed finally. Each of these identical structures contains mainly the following information for each row of the display:

- 8-bit Text: stores the actual character for each column of the rows,
- 32-bit Rendering: stores the rendering information for each column of the rows, and
- Length: number of characters on each row of display.

The text will hold the actual ISCII/ASCII code of the character to be displayed in the columns of the each of the rows, including the scrollbar buffer. The rendering information includes the basic attributes (foreground color, background color, bold, underline etc) and special attributes (whether cursor is in the column, whether the char is ASCII or ISCII etc) as shown in Figure 3 below.

The length of a row will be the actual number of characters in the line or -1 in the case of wrapped lines. Each line for both text and rendering are allocated only on demand. The *text* and *rendering* are pairs, which are allocated or deallocated together.

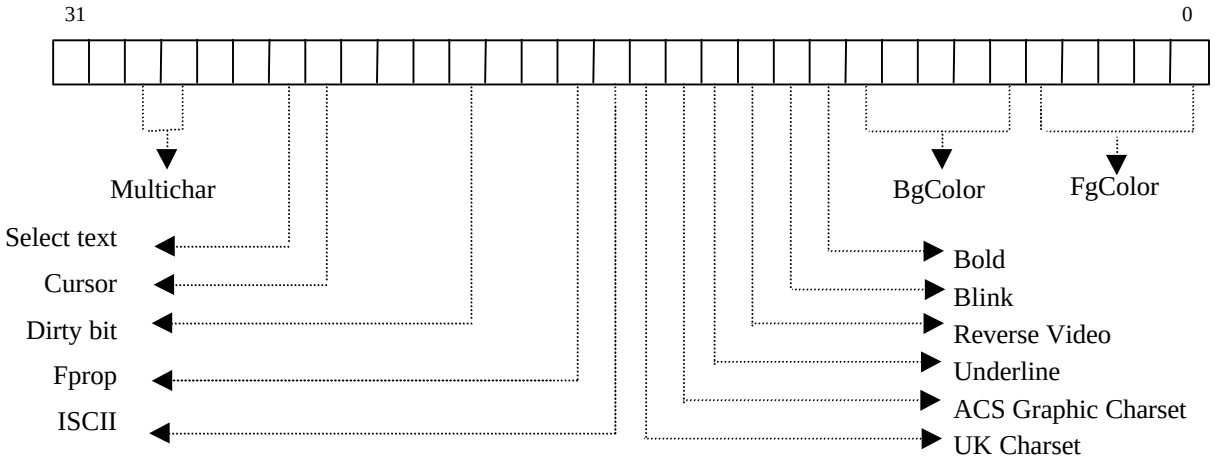

Figure 3: 32-bit Rendering information

The “screen” structure will hold the information for rows even in the scroll back region.
Unlike the normal English fonts, which are mono-spaced, the Indian script fonts are proportional with each character glyph having different width. Therefore, when accommodating Indian scripts, the column-pixel relation with respect to display has to be modified.

Let P_i denote the pixel position for the i^{th} column denoted by C_i. Let FW_i denote the width of the character glyph in the i^{th} column. In the case of English characters, the relation is:

$$P_i = C_i * FW_{fixed}$$

However, in the case of Indian characters, the relation would be:

$$P_i = \sum_j FW_j \text{ where } j = 0 \text{ to } i-1$$

When the character glyphs from both Indian and English script are involved, the computation is accordingly modified.

For displaying Indian languages, True Type Fonts (TTF), which are scalable are used. During the screen refresh, the rendering bits of the character are checked to see if it is an English character or an Indian language character. Accordingly, the fonts will be set for display.

The function key F2 has been programmed to switch among the languages on the fly, which provides immediate transliteration. A list of supported languages is maintained and on suppression of the F2 key, the next language in sequence is loaded, and the Indian characters are redrawn with the font of the new language.

To support the standard Inscript keyboard layout, a keymap file is developed, which contains the ASCII code of the keys pressed along with the desired ISCII code. Since ISCII has been used as the standard, a common keymap can be used for all the Indian languages. Similarly, another keymap file has been developed to handle the phonetic keyboard layout. The function key F1 has been programmed to switch between the two keyboard layouts. Whenever a switch among the keyboard layouts is requested, the keymap for the new layout is loaded into the memory. And when a key is pressed from the keyboard, a function ``a2i_binsearch'' searches the keymap file using binary search and retrieves the ISCII code corresponding to the ASCII code of the character from keyboard. The flow is as explained in the Figure 4 below.

In Indian languages, there is a wide range of vowel modifiers which when applied to base consonants result in modified glyphs. To handle all these consonant-vowel clusters, a map file containing all possible conjunctures and their equivalent glyphs are developed for each Indian language.

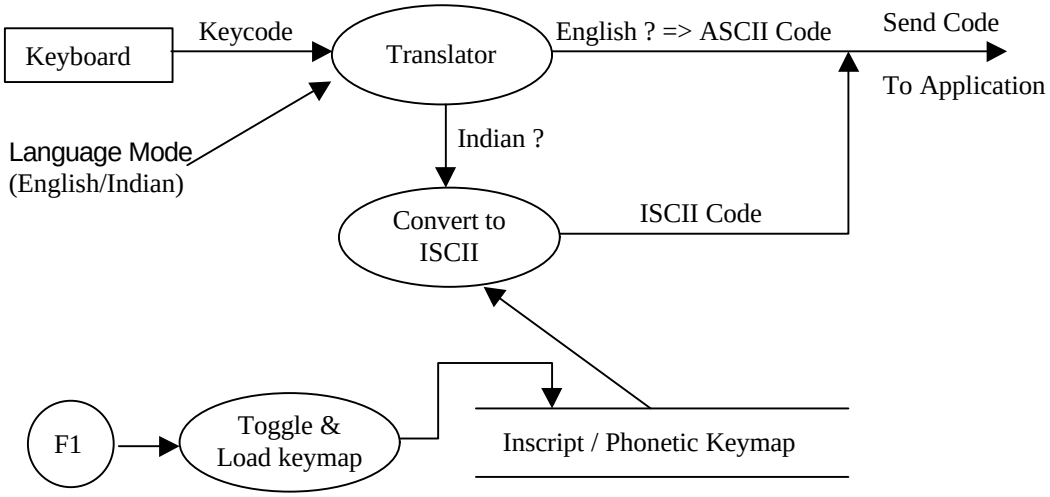

Figure 4: Keyboard translation

The desired Indian language is loaded using the function ``indian_init'' which will select the font for an Indian language from a configuration file and also load the parse rules to handle the clusters for the language.

A set of *lex* rules is defined for the valid conjuncture formation. Once a word matches the rule, a binary search is made in the parse rules and the matching TTF glyphs are retrieved. These will be subsequently displayed using an ``IDrawString'' function.

The flow of transformation of the ISCII input to the screen display is explained in the Figure 5 below:

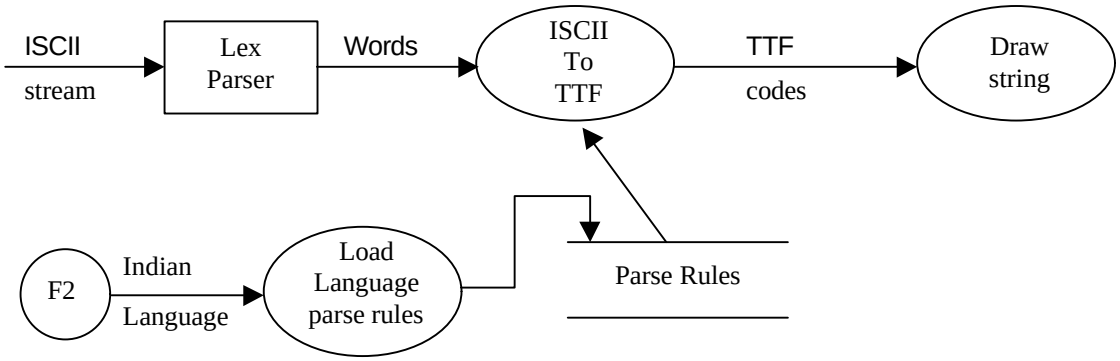

Figure 5: Consonant-Vowel cluster formation

During startup, the virtual terminal has a width to accommodate 80 English characters, which may not be sufficient to accommodate that many Indian characters. However, the window can be easily resized using the mouse.

X -Event Handling in ``iitm-term'':

The programming model in X based applications follows the event-driven approach [10]. Events are generated for various actions such as movement of a window, resizing of a window, exposure of a window, keyPress, mouse button press etc. The programming model captures these X events and processes them in sequence. The RXVT, wherein no Indian scripts were involved was able to handle these events smoothly. But when the Indian scripts are involved the iitm-term was not able to perform smoothly due to the computations involved with regard to the variable width of characters and the column-pixel relations' differences. The terminal was crashing periodically. The problems were prominent when some of these events such as *expose* and *configure notify* which require considerable computations, were getting generated very frequently.

This problem was solved in a two step approach. In the first step, the events were allowed to post their data into a shared memory location protected by a Ternary Semaphore as explained below. In the second step, the data from the shared memory was processed in an optimized way.

The shared memory locations are protected by the ternary semaphore of the following type:

```
typedef enum {Old, New, InUse} DataSemaphore;
```

The semaphore value "Old" indicates that the data in the shared buffer has been used up and can now be overwritten by the assynchronous X event. The value "New" means the data in the shared buffer is not yet processed. A situation where in the X event sees the semaphore value "New" indicates the speed mismatch in the generation of event data and consumption of the data. In this situation, we are overwriting the buffer with the new data, thereby losing the previous unprocessed data. This is acceptable in the situations such as movement of windows and resizing of windows. The semaphore value "InUse" indicates that the data in the

shared buffer is currently being used and should not be overwritten. In this case, the new data of the X event is ignored. This situation indicates the speed mismatch.

So the data posting is done on the X event, based on the semaphore and the processing is done along with a less computation intensive event. By following this approach the crashing of the terminal has been eliminated.

5 Applications and Utilities

A suite of applications was adapted to support Indian languages under console as well as iitm-term. These include - Editors (emacs, vi and pico), Mail client: (pine), Text based browser: (lynx) and Compiler (gcc). In addition, a prototype version of shell (Mash,Tash) with Indian language interface [Malayalam, Tamil] was developed. An utility, ttf2Psf has been developed to convert a TTFfile to a PSF file.

The native language support provided in the kernel can be inherited by any application running on the console or the iitm-term. Still, in order to make full use of the support, applications need to be modified so that their user interfaces and messages are also in the native language.

5.1 Editor : Emacs

To enable Emacs with support for Indian languages, the user interface and system responses need to be changed. So, the frequently used string literals in the Emacs LISP code have been made into variables in the format:

(setq <variable> <ISCII string>)

For each language, a file of these definitions is to be maintained. And depending on the desired language, these definitions are to be loaded. Further, some language-dependent files kept under subdirectories with the respective language names are loaded to enable the Indian language support. LISP functions have been defined to select the desired Indian language and load appropriate files. These are briefly explained below:

- indian-init : This is used to load a default Indian language.
- select-indian-language <language> : This function depending onthe<language> will load the "<language>Msg.el" file, which holds the definitions in the <language>. Also the language dependent files from the subdirectory <language> will be loaded.

These plug-in functions and files are kept in the site dependent startup directory. For enabling 8-bit support, so that emacs does not discard the 8th bit from input, “(set-input-mode nil nil 1) “ is inserted in the startup file. To load an Indian language, the following code is inserted in the startup file:

- (require ‘Indian-msg) ; To use the Indian-msg.el file.
- (select-indian-language ‘Malayalam); Select the Indian language Malayalam.

A user can easily switch between the various supported Indian languages by using the “select-indian-language” function from the ".emacs” startup file in the home directory.

5.2 Shell: Tash

‘Tash’ a derivative of ‘bash’ has been developed with a Tamil interface. This has been accomplished by the usage of shell functions and variables. A subset of Linux commands with the respective options has been implemented with a Tamil interface, with the facility to make additions, if needed.

5.3 Ttf2Psf:

The issue that has been discussed in Section 3.4 prompted the development of the utility ‘Ttf2Psf’. The FreeType Library has been used to extract the bitmaps of the specified glyphs from a TTF file and generate a PSF file with the bitmaps of the specified glyphs embedded in it.

5.4 GCC Localization using GNU Gettext

The main task in localizing an application is the translation of strings given as input to *printf()* or related functions. An easy method of translation is to put all the strings used in the application as macro definitions in some header file, and include the header file wherever they are used. But this would imply recompilation of the application for each language and the necessity for different executables for different languages. The GNU ‘*gettext*’ package enables us to separate out the string tables from the actual code and allows us to supply the string resources as binary files called machine object files along with the executable. If the software is enabled with the *gettext* support, then

native language support can be easily provided for different languages by just creating the machine object files.

The GCC 'C' Compiler supports *gettext*, and this feature has been used in developing local language support for *gcc*. The machine object file has been developed for Malayalam, and this enables display of commonly encountered error messages and warnings of the compilation process in Malayalam. Also, the programmer can give comments and strings in Malayalam inside a 'C' program. The effort for providing native language support for a 'C' compiler has many advantages from a social perspective. With this support, lack of proficiency in English will no longer be a handicap to someone who wants to program in 'C'. This would encourage people who are not proficient in English to learn programming and they can become effective contributors for development of small scale software systems, which inturn could make a significant difference in the rural Indian context.

5.5 Pine and Pico

Support for Malayalam has been provided for Pine – a mail-utility and Pico- an editor. As these applications have not been designed with the *gettext* support, our approach has been to modify the source code to change the string variables and string constants in the source code to Malayalam. The new applications display a user interface in Malayalam.

5.6 ISCII Printing

The content generated with our Indian language –enabled editors (emacs, pico) can be printed using any printer with ISCII support. (For example, the TVSE- MSP 430.)

To complete the full cycle of Indian script handling for printers that do not have in-built ISCII support, an utility ``iscii2ps'' has been developed. This utility helps in printing an ISCII text file. The utility reads an ISCII text file, applies the language dependent parse rules to form the words composed of font codes of the TTF font file for the language. Then a PostScript file is generated with language tags for these words. PostScript functions are written to handle the margins, line justifications, new line and page breaks. The utility also generates a file that can be directly sent to the printer. This utility with the minimal page formatting of margins, justification, new line and page breaks help in getting a printed version of an Indian language text in the true type fonts. This support has been extended to be used on a low-end dot matrix printer also.

6 Conclusion

The iitm-term, the modifications made in the Linux kernel and the utilities that have been developed can be used to effectively provide console based support and the X Window System support for any Indian language or any language that demands variable width font and/or uses modifiers and clusters. At the console level, the support has been developed for Tamil and Malayalam. The task of providing console-based support for any Indian language reduces to that of:

- Creation of a PC Screen Font file: The **Ttf2Psf** utility can be used for this purpose
- Creation of the multi-maps and parse-rules as appropriate for the language.

The support provided can be used for the process of localization of other internationalized packages as well. Using the support that has been provided, native language support can be extended to new applications as well.

For the future, we plan to develop a full-fledged graphics browser with support for Indian languages, under the X Window System. Also, development of an Indian Language interface for *mySQL*, an RDBMS is on the anvil.

We hope that this work will go a long way to contribute in a significant manner in the context of Native Language support for software and consequently in the process of taking the benefits of "information revolution" to the marginalized sections of society and to achieve appropriate social use of information technology.

BIBLIOGRAPHY

[1]. Kenneth Kenistion, *"Politics, Culture and Software"* Economic and Political Weekly, Vol. XXXIII, No, 3, pp. 105-110, January 17, 1998

[2]. *Indian Standard, Indian Script Code for Information Interchange - ISCII* from Bureau of Indian Standards, bisind@del2.vsnl.net.in

[3]. *Consolechars documentation*, documentation included with ConsoleTools package, available at the URL: http://www.multimania.com/ydirson/en/lct/

[4]. Andries Brouwer <aeb@cwi.nl> , *Linux Keyboard and Console HOWTO Documentation*, Version 2.8, 25 February 1998 available at the URL: http://www.linux.com/howto/Keyboard-and-Console-HOWTO.html

[5]. Linux kernel source code, all versions are available at the URL: http://www.kernel.org/pub/linux/kernel/

[6]. Michael Beck, Harald Bohme, Mirko Dziadzka, Ulrich Kunitz, Robert Magnus, Dirk Verworner. *Linux Kernel Internals* Second Edition, 1998, Addison-Wesley. Chapters 1-4, 7, Appendices A,B

[7]. Alessandro Rubini , *Linux Device Drivers* , 1998, O'reilly & Associates Inc. Chapters 1-5

[8]. GNU gettext package documentation, available at the URL: www.gnu.org/software/gettext/gettext.html

[9]. Naoshad A Mehta and Rudrava Roy, "RXVT - Indian Devanagari", www.rxvt-idev.freeservers.com.

[10]. X Window System Programming, Nabajyoti Barkakati, Prentice Hall India Pvt. Ltd.