RESEARCH ARTICLE OPEN ACCESS

Tamil language support for the open source operating system

[1] K. Ravi Kumar, [2]P. Karthik

[1]Asst.professor, Dept.of.Computer science, Tamil University (Established by the Govt.of.Tamilnadu), Thanjavur-613010.

[2]Research Scholar, Dept.of.Computer Science, Tamil University, Thanjavur-613010.

----------**********----------

Abstract:

We propose in this world widely available Tamil software today is written and documented in English, and uses English as the medium to interact with users and company. This has the advantage of a common language of communication between developers, maintainers and users from different countries. In a country like India, an of the population does not know English. Given this fact, availability of native language software will crucial help as 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

Keywords: **operating system, key code.**

----------**********----------

Introduction:

All 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.

Anternationalization 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 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 problem analysis:

This paper 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 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.

SUPPORT ON LINUX OS:

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 key codes/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.

The keyboard driver support:

The keyboard-input mechanism at the console level in Linux is as follows: When a key is pressed, the keyboard controller sends scan codes to the kernel keyboard driver. The keyboard driver sends whatever it receives to the application program when it is in scan code mode for example, when the X Window System runs. Otherwise, it parses the stream of scan codes into key codes, display content level Tamil action

2.2 The X Window and keyboard

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 is broken into two layers: server-specific codes called *keycodes* which represent the physical keys, and serverindependent 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

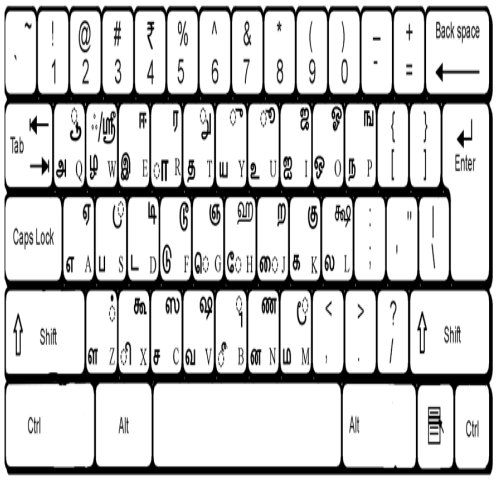
The display mechanism under the X Window System is as follows: Unlike text display terminals, X displays can show text in varying sizes and shapes. The X server retrieves the symbols from a font by indexing based on character code. There are mainly two types of fonts – Bitmap-based fonts and Outline-based 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 of Tamil Language and key action

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.

Key code font action:

Linux uses the *PC Screen Font* (PSF) format for display purposes for the console. 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.

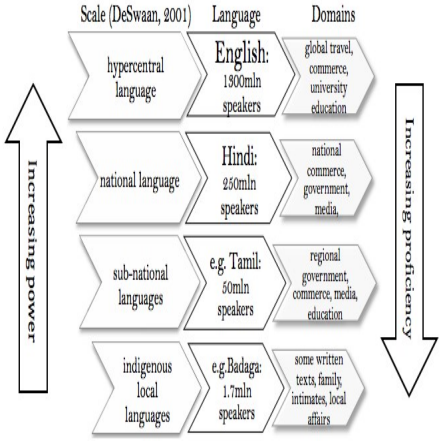
applications running in it support only English and other foreign languages, which do not demand variable width glyphs.

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.

Key code support language 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 + Ê = ÊE ; But in Tamil it takes the form: è + ¤ = A. In some cases, the consonant may et 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.



5.1 Editor

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

IV. CONCLUSIONS

Among them, the homomorphism key agreement allows authorized users get more control key and in order to achieve the purpose of file sharing. The encryption and decryption functionalities are performed at the CS that is a trusted third party in the SeDaSC methodology. The proposed methodology can be also employed to mobile cloud computing due to the fact that compute-intensive tasks are performed at the CS. We described the permissions provided by the services, their semantics, and the access-granting techniques that are used to apply these permissions to users. Additionally, a set of protocols for sharing data securely in several public storage clouds were presented. These protocols were by extending an ideal set of properties required for sharing data between users of a cloud service.

V. REFERENCES

[1] J. Wu, P. WyckoffandD. K. Panda,"PVFS over InfiniBand: Design and Performance Evaluation",In: Proceedings of 2003 International Conference on Parallel Processing, (**2003**), pp. 107-115.

[2] S. A. Weil, K. T. Pollack and S. A. Brandt, „Dynamic Metadata Management for Petabyte-Scale File Systems", In: Proceedings of the ACM/IEEE SuperComputing Conference, pp. 35-47.

[3] D. Ellard, J. Ledlie and P. Malkani, "Passive NFS Tracing of Email and Research Workloads", In: Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST'03), San Francisco, CA, (**2003**) March, pp. 203-216.

[4] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu,"Scalable secure file sharing on untrusted storage," in Proc. OfFAST, 2003, pp. 29-42.

[5] E. Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius:Securing remote untrusted storage," in 2003, pp.131-145.

[6] Google drive. https://drive.google.com/