CS-499 BTech Project

Final Project Report

TITLE: LINUX FOR INDIAN LANGUAGES

Guide: Dr. Rajat Moona

Submitted by: Aravind Menon (98069), Gaurav Gupta (98133)

Introduction

Objective

The objective of this project is to produce a Linux distribution that has support for a number of Indian languages.

Principal requirements

The following are the requirements that the new distribution should meet:

1. Existing X applications should be able to support Indian languages without any modification in their binaries, i.e. they should be able to correctly display ISCII [1] text.
2. There should be keyboard support for allowing user to enter Indian Language text.
3. Internationalization: The new distribution should provide Indian language support for already internationalized applications – i.e. those applications whose output, or part of output, depends on the language selected.
4. Unicode/UTF-8 support: It should be possible to correctly display Unicode/UTF-8 [3] encoded strings of Indian languages.

Approach

Since existing X applications are supposed to run unmodified in the new distribution, the support has to be provided at an underlying low-level library. We have decided to incorporate support for Indian languages at the level of 'Xlib' – the client side library of the X Windowing system [2,4].

The modified X library (`libX11.so`) provides the support for correct display of ISCII strings and Unicode/UTF-8 strings of Indian languages.

Keyboard support is incorporated using the XKB mechanism of X. This mechanism allows for defining various aspects of keyboard mapping like symbols of the keyboard, types of keys, interpretations of the symbols etc. The XKB mechanism is defined in greater detail in the Section 'Keyboard Mapping'. Actual details of the implementation can be found the 'Documentation' section.

Localization support for internationalized applications has been added using the 'locale' mechanism of glibc. This mechanism allows one to define new 'locales', which define the behaviour of internationalized applications for the 'locale'.

The rest of the report is organized as follows. In section 1 we give a brief summary of the work done in the previous semester. In section 2 we describe the XKB mechanism [6], 'locale' mechanism [7] and Unicode/UTF-8 support. In section 3 we show some snapshots of results we have obtained. The Appendix contains sections on documentation, the programmers' manual and the administrator's manual.

Section 1: Summary of previous semester's work

Work done in the previous semester mainly concentrated on the following two aspects:

1. Modifying Xlib to support display of ISCII text
2. Investigation of the mechanisms for adding keyboard support.

We now briefly describe the two

1.1 Xlib modification

The principal task in adding support for ISCII is to first convert it to glyph coded string before the display functions of X are called. This is because the display functions of X, viz `XDrawString`, `XDrawText` and `XDrawImageString` expect their input strings in glyph encoded form. For English and European languages, the character string and glyph strings are identical, whereas this conversion has to be done explicitly in the case of ISCII.

Similarly, while finding the text extents (text height, width, left and right bearings and font ascent and descent) of a string, the string has to be first converted into the glyph string in the appropriate font. This has to be done for the Xlib functions `XTextExtents`, `XTextHeight` and `XTextWidth`.

The modified Xlib first checks if the font set in the graphics context corresponds to an Indian language. In case it is, Xlib assumes that its input string is an ISCII string and converts it into the glyph string for the corresponding font. For this conversion, it uses the 'isciilib' conversion routines written by Jyotirmoy Saikia (MTech 99).

The X routines for finding text extents cannot determine the font name from the font id because they do not have a parameter for connection with the server. However, the font name is required for determining whether the current font corresponds to an Indian language. Maintaining a font cache to store the mapping between font id and font name for fonts that have been loaded solved this problem. This cache also improves the overall speed of the display functions (`XDrawString` etc) because an expensive communication between the X server and client is avoided.

1.2 Mechanisms for keyboard support

Keyboard support means that the user should be able to use the normal English keyboard for inputting ISCII text. This means that an alternate keyboard layout be provided for inputting ISCII text. The user should be able to switch between different keyboard layouts by some keypress.

Keyboard mapping in Linux is handled differently for the two modes in which the system can run:

1. Terminal mode
2. X Window system mode

In terminal mode, the keyboard mapping (i.e. conversion from scan-codes to character codes) is handled by the Linux keyboard driver, whereas in X mode, it is handled by the X server. Both these mappings are implemented by maintaining a mapping table. There are utilities available which allow a user to write his own mapping tables (in a particular syntax), these mappings cane be then loaded dynamically into the system.

They are:

1. loadkeys for terminal mapping table
2. xmodmap for X mapping table.

Section 2: This semester's work

This section discusses the following three aspects:

2.1 Keyboard support for X
2.2 Localized Indian support for internationalized applications
2.3 4-byte Unicode/UTF-8 support in Xlib

The keyboard section discusses the XKB mechanism of X, and how this mechanism is used for incorporating support for Indian languages. In the localization section, we discuss the GNU '`gettext`' mechanism, which allows for defining localized behaviour of internationalized applications. We also describe some new locales, which have been added by the TDIL group at IIT Kanpur using this mechanism. Finally, in the Unicode/UTF-8 section, we describe how the X localization mechanism has been used to incorporate Unicode/UTF-8 support for Indian languages.

2.1 Keyboard support for X

Objective

To provide three keyboard layouts on a standard keyboard:

1. Normal US English layout
2. English Phonetic layout
3. Inscript layout

Behavioural requirement

The normal US English layout behaves exactly like the normal US keyboards.

In the English phonetic mode, the user is able to enter ISCII characters by pressing their English phonetic equivalents. For example, 'k' generates the ISCII code for ' ', 'K' for ' ', etc.

#Ä	#Æ	#&	+	+É	<	<Ç	=	>	@	Bà
sh+x	x	sh+;	sh+a	alt+a	alt+i	sh+alt+i	alt+u	sh+alt+u	sh+alt+r	sh+z
B	Bì	B	+Íà	+Íä	+Íè	+ÍÉ	E	JÉ	MÉ	HÉ
sh+e	sh+alt+e	sh+2	sh+`	alt+o	sh+alt+o	sh+\	k	sh+k	g	sh+g
R	SÉ	U	VÉ	ZÉ	\É	]	`	c	f	hÉ
q	c	sh+c	j	sh+j	sh+]	alt+t	sh+alt+t	alt+d	sh+alt+d	sh+n
iÉ	lÉ	n	vÉ	xÉ	xÍÃ	{É	¡	}É	¦É	¨É
t	sh+t	d	sh+d	n	alt+n	p	sh+p	b	sh+b	m
ªÉ	ªÍÃ	®	®Ã	±É	²É	²ÍÃ	´É	¶É	¹É	ºÉ
y	sh+y	r	sh+r	l	sh+l	alt+l	v	sh+s	alt+s	s
½	#É	Ì#	#Ò	#Ö	#Ú	#Þ	#à	#ä	#è	#ì
h	a	I	sh+i	u	sh+u	alt+r	z	e	sh+e	2
#Íà	#Íä	#Íè	#ÍÉ	#Â	Ã					
`	o	sh+o	\	-	]					

In the Inscript layout mode, the user is able to enter ISCII characters according to the Inscript layout, which is the recommended layout for ISCII. The following image shows the Inscript overlay for Devanagari.

The user should be able to switch cyclically through the US English, English Phonetic and Inscript layouts by pressing the ScrollLock key. The user should also be able to switch temporarily from one layout to the other by pressing the AltGr key i.e. in the US English mode, AltGr switches temporarily to the English Phonetic mode; in the English Phonetic and Inscript mode, AltGr switches temporarily to the US English mode.

The LED lights of ScrollLock and CapsLock should reflect the currently enabled layout of the keyboard. In English Phonetic mode, the ScrollLock LED should be ON; in the Inscript mode, both ScrollLock and CapsLock LEDs should be on.

Approach

The above support is incorporated using the XKB mechanism, which we describe below.

XKB is the X server's keyboard module. XKB is a highly configurable module and its behaviour is determined by a database of configuration files. These configuration files are read and compiled by the XKB module each time the X server starts. By modifying these configuration files, or adding new configuration files, we can effectively modify the X keyboard behaviour to suit our requirements.

XKB configuration database consists of the following 5 components. For a greater description of the components, please refer to the documentation in the Appendix.

keycodes

This consists of tables that define symbolic names for key scan-codes

Types

This describes what are known as key types. Each keyboard key is of a particular key type. The key type of a key determines how the character code produced by the key changes in the presence of 'modifiers' (Control, Shift and so on).

compat (abridgment from compatibility)

This describes the 'behavior' of modifiers. XKB has some internal variables that store the state of the modifiers as well as the 'group'. These internal variables, along with the pressed key determine what character code is generated.

The 'compat' files describe how these internal variables change when any modifier key is pressed.

symbols

This is the main table which specifies the mapping from scan-codes (symbolic names defined in 'keycodes') to the set of all possible values ('symbols').

geometry

This file describes keyboard geometry – key placement on the physical keyboard. Two keyboards that have different physical key placements will have two different geometry files. Thus, the geometry file specifies the configuration of keyboards of different manufacturers.

Using the XKB mechanism described above, Indian language support has been introduced by writing a new 'symbols' configuration file. Additional 'compat' and 'types' configuration files have also been written to support this 'symbols' file. A typical entry in the 'symbols' file would look like the following:

```
key <AB06> {    [],
     type[Group2]= "IND_THREE", [n, 193, 198, 199],
     type[Group3]= "IND_TWO", [n, 209, 210]
};
```

This entry defines three groups for the key corresponding to the letter 'n'.The first group is the US English group whose symbols are the same as the symbols for the US keyboard. The second group is the English Phonetic group: the key is defined to be of type "IND_TWO", and the group defines four symbols – 'n', and ISCII codes 193, 198 and 199.

IND_THREE is defined in 'types' as follows.

```
type "IND_THREE" {
        modifiers = None+Shift+Alt;
        map[None]= Level2;
        map[Shift]= Level3;
        map[Alt]= Level4;
    };
```

Thus, in the English Phonetic mode, a normal keypress produces code of 'level2', i.e. ISCII code 193, 'Shift' + <AB06> produces code of 'Level3' i.e. ISCII code 198, 'Alt'+<AB06> produces 'Level4' code i.e. ISCII code 199, and any other modifier+ <AB06> produces 'n'.

Similarly, the third group is defined for the Inscript layout.

We also require that the user should be able to switch between the three keyboard layouts. This switching between groups is done by the ScrollLock key, which has been defined as follows.

```
key <SCLK> {     [ ISO_Next_Group   ]        };
```

ISO_Next_Group is defined in 'compat' as follows:

```
interpret ISO_Next_Group {
     action= LockGroup(group=+1);
};
```

i.e. On pressing, the group is incremented by one and locked.

For displaying the current layout of the keyboard, LEDs have been programmed. Below, we show part of the code for handling this functionality

```
indicator "Scroll Lock" {
          groups=All - group1;
};
```

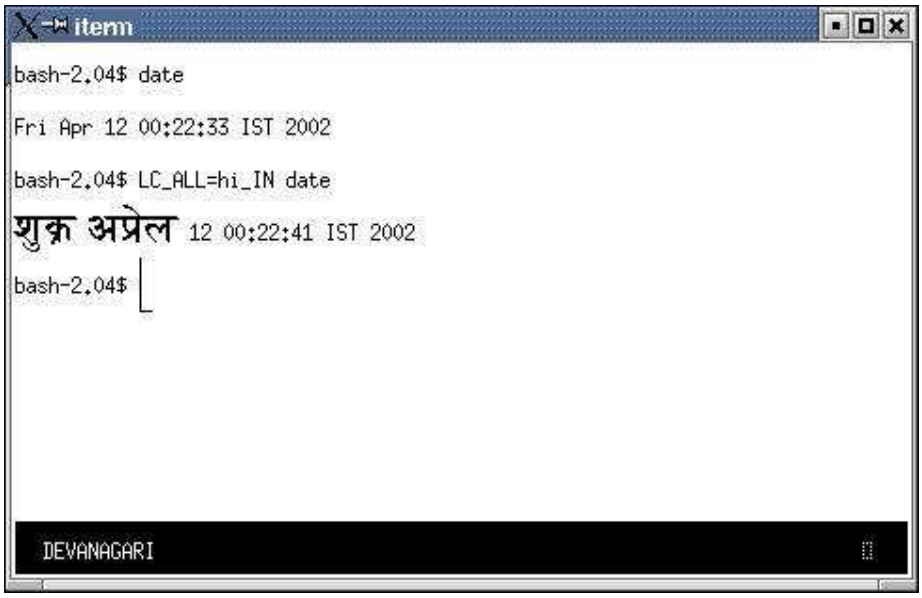
` This means that the ScrollLock LED is to be turned for groups 2 and 3, i.e. English Phonetic and Inscript keyboard.

Thus, with these new configuration files, we are able to achieve the desired behaviour from the keyboard. However it is not possible to have keyboard input for characters that are not represented by a single ISCII code. Thus characters IÉ, jÉand YÉ cannot be entered from the keyboard developed by us.

2.2 Localized Indian support for internationalized applications

An internationalized application is one whose output strings and output format are not hard coded in the application. The output strings and format of an

internationalized application vary according to the language, i.e. in Linux according to the environment variable LANG. For example, the 'date' program is internationalized and its output is as follows (in Iterm):



Here, the output string is different for Hindi. The format may also be specified to be different.

Localization means adding support for a new language (or locale) for an already internationalized application. In our case, it would mean adding support for Indian languages for internationalized applications.

Below, we briefly describe the GNU mechanism for adding localization support. Next, we describe how this mechanism has been used for Indian language support. This implementation part of the work was done by the TDIL group at IIT Kanpur.

GNU gettext standard

The prevailing programming interface for internationalization in POSIX-compatible systems is that specified in ISO C, and extended in POSIX.2. The gettext standard defines *locales* which are specified by a name string, and six categories of localization information for which users may specify their locale of choice. These categories define different aspects of the localized behaviour of programs.

LC_CTYPE

specifies the character set. The character set defines different categories of characters for the locale e.g. lower characters, upper characters, digits etc.

LC_COLLATE

specifies the conventions for sorting order i.e. the order in which the characters of the languages are to be sorted

LC_TIME

specifies the formatting of dates and times, including the day and month names in the locale's language..

LC_NUMERIC

specifies the formatting of numbers.

LC_MONETARY

specifies the formatting of monetary quantities.

LC_MESSAGES

specifies the message strings for different applications of the language.

Programs that want to use local conventions for their output use the setlocale function to specify the locale used for each category of local conventions. The specified locale is then used by the various C library functions like `'toupper'`, `'strcoll'` etc

The gettext package functions `'localeconv'` and `'nl_langinfo'` can be used by applications to retrieve items describing local conventions from the current locale. Thus, these applications can display their output in the local conventions.

For writing locale descriptions POSIX.2 specifies a language. The `localedef` program compiles these description files into locale specification files. The language specified by POSIX.2 is for the Categories LC_CTYPE, LC_COLLATE, LC_TIME, LC_NUMERIC and LC_MONETARY. A typical entry for a category looks like the following.

Ex: (for hindi)

```
LC_NUMERIC
% This is the POSIX Locale definition for the LC_NUMERIC
category.
%
decimal_point           "<U002E>"
thousands_sep           "<U002C>"
grouping                3;2
%
END LC_NUMERIC
```

The above defines the decimal point for Hindi to be "." and "," as the separator. It defines the first delimiter to be after the 3rd digit and the subsequent delimiters to be after every 2nd digit.

The values of `<U002E>` and `<U002C>` have been mapped to corresponding ISCII strings in a `charmap` file. This `charmap` file is used by `localedef` during compilation of the locale.

The above categories for ISCII locales have been defined for the following languages:

(i) Assamese,
(ii) Bengali,
(iii) Gujarati,
(iv) Hindi,

(v) Kannada,
(vi) Malayalam,
(vii) Oriya,
(viii) Punjabi,
(ix) Tamil and
(x) Telugu

This work has been done by the TDIL group at IIT Kanpur.

Message strings for applications (LC_MESSAGES)

The LC_MESSAGES category can be used for specifying the language specific message strings of an application. This is done through the following steps:

1. The message strings appearing in the program are extracted using the `xgettext` utility. This extracts the strings into a `'po'` (portable object) file.
2. The `'po'` basically consists of pairs of `'msgid'` and `'msgstr'`. The `'msgid'` stands for the original string and `'msgstr'` stands for the translated string. For `'msgid'` strings, which are to be translated, the corresponding `'msgstr'` field has to be filled. The following is an example of part of a `po` file with a translated string:

```
#: main.cpp:36
msgid "KDE Floppy Disk utility"
msgstr "Eä.bÒ<Ç}±ÉÉ{{ÉÒÌbºE ªÉÒ{ÉÒ±ÉÉ{ÉÒ"

#: main.cpp:42
msgid "Kfloppy"
msgstr "Eä}±ÉÉ{{ÉÒ"
```

3. The `'po'` files are complied into `'mo'` (machine object) files using the utility 'msgfmt'.
4. Finally, when the application runs, the `gettext` library returns to it the translated string for the language selected.

Internationalized applications make call to gettext, `dgettext`, and `dcgettext` functions of the glibc library and get the strings to be displayed for the messages.

2.3 Unicode/UTF-8 support in X

Objective

To incorporate Unicode/UTF-8 support for Indian languages in Xlib.

Approach

Unicode/UTF-8 support has been added into the X localization framework of Xlib. This framework allows the X developer to add new 'locales' to X. We have incorporated Indian language support in the existing UTF-8 locale of X. The following subsections describe the X locale mechanism, and the additions we have made to add support for Indian languages.

X Locale mechanism

Xlib provides the locale mechanism to provide support for localization (l10n) in text display. Non-English languages like Arabic, Chinese, Japanese etc require specialized routines to handle text display. For example, Chinese text is written vertically, so a call to XDrawString won't work. Similarly, Arabic text may be written right to left. Each language - or locale, as we shall call it, therefore requires specialized routines for handling text display.

Xlib l10n framework allows us to 'define' new locales - 'New_locale' can be added by adding:

1. The loader for 'New_locale' – this is a new C file which has to be added to the X library, and X recompiled
2. The configuration file for the locale

Some definitions first:

Multibyte characters: Multibyte refers to an encoding in which each character might be encoded in a variable number of bytes. In case of X localization, a multibyte char represents text in the local encoding.

WideCharacter string: WideCharacter refers to an encoding in which each character takes the same number of bytes. In case of X localization, this refers to Unicode 4-byte encoding.

Charset: A charset refers to the set of displayable characters. The charset, is therefore, the glyph encoding, e.g.: ISO8859-1, ISO8859-2, Latin-1, etc

An Xlib locale is defined for a single character encoding – usually the local encoding of the language, for example `Jis`, `Sjis` are two different encodings for Japanese - hence two different locales have to be defined for the two.

1. Locale loader - The locale loader sets conversion routines for interconversion between character encoding, glyph encoding and wide character. The characters to glyph conversion routines are called subsequently when localized text is to be displayed. Note that conversion routines have to be defined for each charset the locale supports.

2. The configuration file defines the set of charsets that the locale supports along with the corresponding fonts for the charsets. Thus, a locale might support multiple charsets. Additionally, other properties of the font (eg whether it is a vertical script), and some properties of the charsets are defined in this file. A configuration file entry may look like the following:

```
fs1     {
        charset {
                name    ISO8859-1:GR
        }
        font    {
                primary ISO8859-1:GR
        }
}
```

The entry says that for the charset named `ISO8859-1` the fonts that can be used should end with `ISO8859-1` (case ignored). `'fs1'` in the first line specifies the fontset. `GR` specifies that the characters in the charset are unsigned.

When the application calls X functions to display the locale encoded strings, the conversion routines of the current locale convert this string into the correct glyph string. The configuration file has been parsed beforehand, so X knows which font to load for the current locale. This font is loaded and the glyph string displayed using XDrawString.

If the string consists of substrings of different charsets, X correctly converts the two substrings into different glyph encodings and displays them in the correct fonts.

Unicode/UTF-8 support using X locale

The current release of X on which we have worked (4.0.2) defines a locale for UTF-8 locale, among other locales. This locale was modified to add conversion routines for the Unicode range of Indian languages. These converters convert the UTF-8/Unicode string to ISCII strings. The modified XDrawString then handles display of ISCII strings automatically.

As a result 9 new charsets have been added in the file `lcUTF8.c` (in the source of X11). The names of the new locales are

(i) Devanagari,
(ii) Punjabi,
(iii) Bengali,
(iv) Tamil,
(v) Telugu,
(vi) Gujarati,
(vii) Malayalam,
(viii) Kannada,
(ix) Oriya.

The conversion is table driven from UTF8 to ISCII. These conversion routines have been added in the files `dv.h`, `pn.h`, `bn.h`, `ta.h`, `tl.h`, `gj.h`, `ml.h`, `kn.h` and `or.h` in the `lcUniConv` directory of the X11 source.

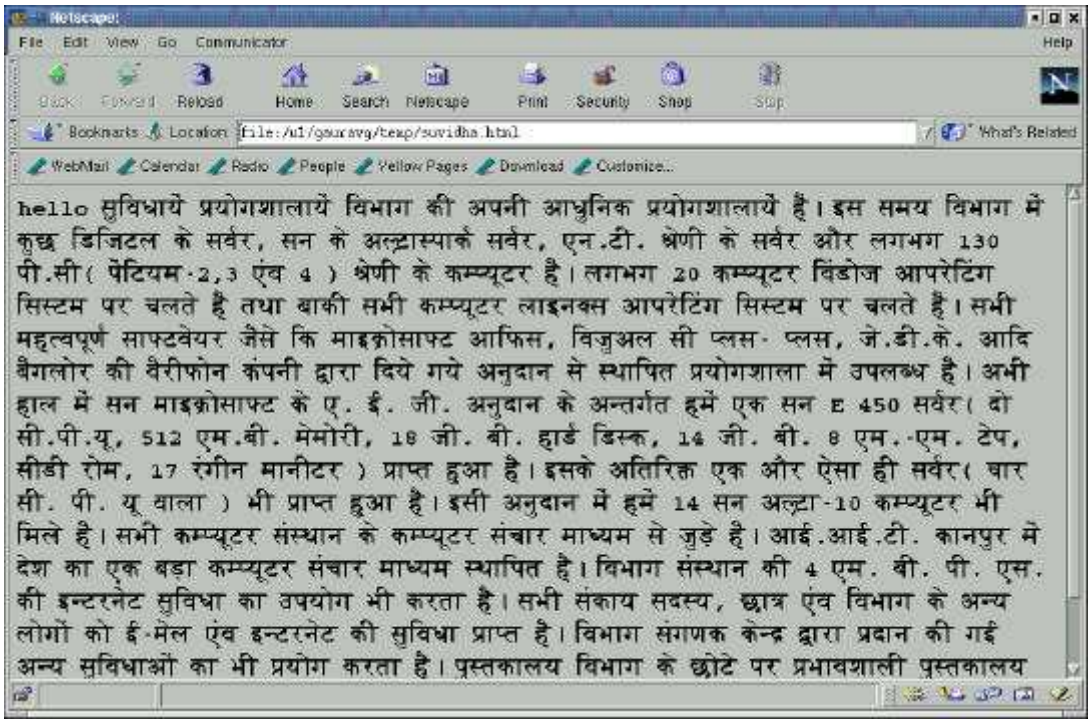
A new configuration file has been added for the UTF-8 encoded Indian languages.

Further description of the exact nature of changes is in the documentation section .

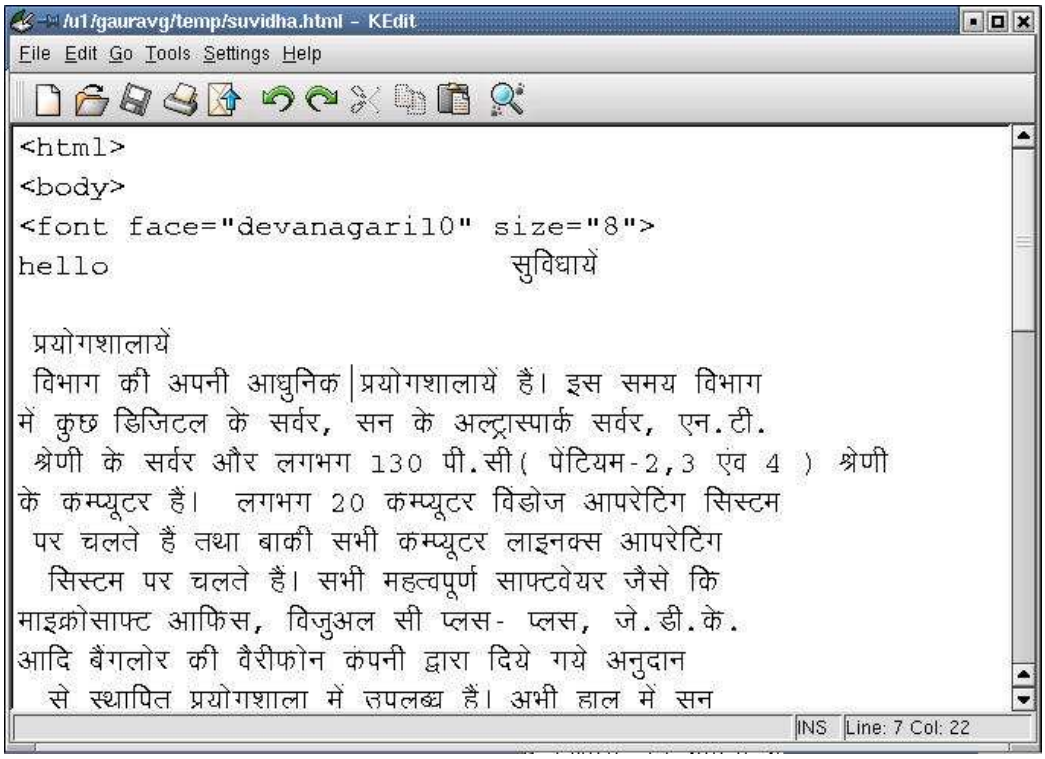
Section 3: Results

The XKB has been implemented successfully. Text input can be done in ISCII format. Scroll-lock is used to shift between the English, English phonetic and the Inscript keyboard. The right ALT key is used as a temporary shifting from one group to another. Pressing the right ALT in English keyboard gives an English phonetic keyboard. In the other two modes, pressing the right ALT takes one to the English keyboard.

All the X applications are working with ISCII once the appropriate font is set. For example, Netscape can display a mixture of English and Hindi text.



Now that both the input and display functions are properly modified, text editors can also work be used to edit a text, which is a mixture of ISCII and ASCII text. However, ATR characters are not yet supported. An example would be that text editors are also working fine with the modified X library provided an Indian font can be set as default. For example, we used kedit to make the HTML file seen in the Netscape window above.



X localization support is also complete. Thus a mixture of multiple languages in UTF8 or Unicode encoding can be displayed. We have written a small application that displays a mixture of UTF-8 Hindi and English text using X locale. This application is successfully able to display the text.

C localization has been implemented. However, we are unable to change the MenuBar etc. of KDE applications to display ISCII strings. This is probably because qtlib, the toolkit which KDE applications use, does some preprocessing of its own on the strings it has to display.

Conclusion

We have been mostly successful in our objective of providing support for Indian languages in Linux. All our stated objectives have been satisfied except for the internationalization of X window applications.

Acknowledgements

We would like to thank our guide, Dr. Rajat Moona, for his immense help and able guidance throughout the project. We would like to thank Jyotirmoy Saikia for providing 'isciilib' and for personally clarifying any doubts we had about its usage. Finally, we would like to thank the TDIL group at IIT Kanpur for their cooperation and support in the project work.

References

1. IS 13194: 1991, Indian Standard, Indian Script Code for Information Interchange – ISCII.
2. X source
3. http://www.Unicode.org - for resources on UTF-8/Unicode
4. http://www.Xfree86.org
5. http://www.Linux.org
6. http://www.tsu.ru/~pascal/en/xkb/ - xkb documentation
7. http://www.gnu.org/software/gettext/ - GNU gettext documentation

Appendix

Section 1: Documentation

Changes made to Xlib distribution

The changes made to the Xlib distribution fall into three major categories:

1. Functions pertaining to display for ISCII text
2. Functions related to display for UTF-8 text
3. Keyboard mapping

1.1 ISCII Text handling

Modifications have been made to Xlib so that it can successfully handle ISCII text, and therefore all Indian languages, which are based on the Brahmi script. The modified Xlib is to meet the following requirement: It should correctly display both pure ISCII text, and ISCII text interspersed with ASCII (English) text.

The following functions of libX11 have been modified for this purpose:

Function name	File
a. `XDrawString`	`Text.c`
b. `XDrawText`	`PolyTxt.c`
c. `XDrawImageText`	`ImText.c`
d. `XTextExtents`	`TextExt.c`
e. `XTextWidth`	`TextExt.c`
f. `XTextHeight`	`TextExt.c`

and also

g. `XLoadFont`	`LoadFont.c`
h. `XLoadQueryFont`	`Font.c`
i. `XUnLoadFont`	`UnldFont.c`
j. `XFreeFont`	`Font.c`

Files named `font_cache.h`, `font_struct.h`, and `AltFont.c` have been added. `AltFont.c` defines the following functions

```
char * getFontName (const int fid);
XFontStruct * getFontStruct (const int fid);
XFontStruct * getAltFontStruct (const int fid);
char * queryFont (const int fid, Display *dpy);
void storeFont (const int fid, char *fontname,
                    Display *dpy);
void freeFont (const int fid);

char * getAltFont(const char *iscii_font);
```

First we describe the functionality of the modified functions:

The first six functions do tasks related to text display and finding extents of text. The modifications done to these functions are very identical in nature.

a. `XDrawString (display, d, gc, x, y, string, length);`

The parameters of relevance are `gc`, `x`, `y`, `string`, `length`.

The default functionality of XDrawString is to display `'string'` at position `x`, `y` in the font `'gc->values.font'`. The `'string'` is assumed to be the `'glyph'` string.

Modified functionality: XDrawString assumes that the `'string'` passed is ISCII + ASCII string if `'gc->values.font'` corresponds to an Indian font, ie if the `'getencodingbyname'` function call succeeds for this font. (`'getencodingbyname'` is a function of `'isciilib'`, which returns the name of the encoding file for the Indian font).

Next, assume that `'string'` is a sequence of alternating ISCII and ASCII substrings, starting with either an ISCII or ASCII substring. If the first substring is ISCII, (i.e. `'codeconversionbyencoding (string)'` is successful in parsing some characters), the font remains the Indian font, else the corresponding English font is set (using `'getAltFont'`). The first substring is then displayed at `(x, y)`. `x` is incremented by the amount `XTextWidth(first substring)`. `XDrawString` is then recursively called to display the rest of the string at the incremented x position.

b. `XDrawText (display, d, gc, x, y, TextItems, nitems);`

TextItems is an array of `'nitems'` number of `'strings'`, each of which is to be displayed successively in a DIFFERENT font.

The modified function loops over the `'nitems'` calling XDrawString for each of the `'strings'`

c. `XDrawImageString (display, d, gc, x, y, string, length);`

This is exactly identical to `XDrawString` except that the background of the display is set to some colour other than white. The changes made to this function are exactly the same as those made to `XDrawString`.

d. `XTextExtents (fontstruct, string, nchars, dir, font_ascent, font_descent, XCharStruct);`

This function finds the extents of the `'string'` of `'nchars'` number of characters for the font `'fontstruct->fid'`. The `'dir'`, `'font_ascent'` and `'font_descent'` can be found directly from the `'fontstruct'`. Other information that is returned in the `'XCharStruct'` is the total width, maximum ascent, maximum descent, and left and right bearings of the 'string'.

Here also, the modification done is of the same nature as that done to `XDrawString`. Only difference is that instead of a recursive call to `XTextExtents`, loop iteration is done.

e. `XTextWidth (fontstruct, string, count);`

This returns the width of the 'string'.

Modification: Similar to XTextExtents

f. `XTextHeight (fontstruct, string, count);`

Return the height of 'string'.
Modification: Similar to XTextExtents

The next four functions handle tasks related to font loading and unloading. These functions have been modified to cache the some data related to Indian fonts. The reason for this modification is the following:

Functions d, e and f – i.e. the functions pertaining to text extents, do not have a handle to the display. Therefore, these functions cannot determine the fontname from the font id. The fontname is required for determining whether the current text is ISCII+ASCII or pure ASCII. Therefore, a cache is maintained for Indian fonts. The cache entry of a font contains its fid, name, fontstruct and altfontstruct. 'altfontstruct' is the fontstruct for the corresponding English font.

g. `XLoadFont (display, name);`

Loads the font and returns fid.
Modification: call `storeFont (fid, fontname)` - storeFont caches the required data if fontname corresponds to an Indian font.

h. `XLoadQueryFont (display, fid);`

Loads and queries font, returns fontstruct
Modification: same as above

i. `XUnloadFont (display, fid)`

Unload font.
Modification: call `freeFont` - freeFont removes font information from cache if it was there

j. XFreeFont (display, fontstruct)

Same as above, additionally, `fontstruct` is freed.

The cache is maintained as a simple linked list. The function declarations to maintain the cache can be found in font_cache.h, definitions can be found in AltFont.c.

1.2 UTF-8/Unicode text handling

Requirement

Modified Xlib should correctly display UTF-8/Unicode text corresponding to any Indian language + English.

Approach

Adding a new locale named "in_IN.UTF-8"

Source files modified: lcUTF8.c

Files added: In directory lcUniConv
dv.h, bn.h, gj.h,
kn.h, ml.h, or.h,

pn.h, tl.h, tm.h,
iscii.h

Configuration files added:

XLC_LOCALE in new directory
/usr/X11R6/lib/X11/locale/in_IN.UTF-8

Xlib provides an elaborate mechanism to provide support for localization (l10n) in text display. Non-English languages like Arabic, Chinese, Japanese etc require specialized routines to handle text display. For example, Chinese text is written vertically, so a call to XDrawString won't work. Similarly, Arabic text may be written right to left. Each language - or locale, as we shall call it, therefore requires specialized routines for handling text display.

Xlib l10n framework allows us to 'define' new locales - 'New_locale' can be added by adding:

1. The loader for 'New_locale' - lcNew_locale.c
2. The configuration file XLC_LOCALE for the locale in
   /usr/X11R6/lib/X11/locale/New_locale directory

An Xlib locale is defined for a single character encoding – usually the local encoding of the language, for example Jis, Sjis are two different encodings for Japanese - hence two different locales have to be defined for the two.

1. Locale loader - The locale loader sets conversion routines for interconversion between character encoding, glyph encoding, UTF-8 and Unicode. The character to glyph conversion routines are called subsequently when localized text is to be displayed.

2. XLC_LOCALE file defines the set of charsets that the locale supports along with the corresponding fontname extensions. Additionally, other properties of the font (eg whether it is a vertical script), and some properties of the charsets are defined in this file.

The current release of X on which we have worked (4.0.2) defines a locale for UTF-8 locale, among other locales. We have modified this locale to add UTF-8/Unicode to ISCII converters for nine Indian scripts - Devanagari, Oriya, Malayalam, Punjabi, Bengali, Gujarati, Telugu, Tamil and Kannada. A separate converter was not added for Assamese because its Unicode range is the same as that of Bengali.

1.3: Keyboard mapping

Objective

To provide two alternate keyboard layouts for typing of Indian scripts:

1. Inscript keyboard
2. Phonetic keyboard

Both these layouts are keyboard overlays on a normal English keyboard.

The following is a description of the 5 components of XKB database

keycodes

This consists of tables that define symbolic names for key scan-codes For example

```
<TLDE>= 49;
<AE01> = 10;
<AE02> = 11;
<AE03> = 12;
```

Additionally, symbolic names are defined for the led indicators. Eg.

```
Indicator 3 = "Scroll Lock";
```

Types

This describes what are known as key types. Each keyboard key is of a particular key type. The key type of a key determines how the character code produced by the key changes in the presence of 'modifiers' (Control, Shift and so on).

For example the 'letter' keys are of type "ALPHABETIC" which is defined as follows:

```
type "ALPHABETIC" {
        modifiers = Shift+Lock;
        map[Shift] = Level2;
        preserve[Lock]= Lock;
        level_name[Level1] = "Base";
        level_name[Level2] = "Caps";
};
```

This is interpreted to mean that the `'Shift'` and `'Lock'` (Capslock) modifiers affect the code produced on pressing an ALPHABETIC key. The code to be generated if the `'Shift'` state is set is defined to be that of `'Level2'`. The actual code corresponding to `'Level2'` will be found in the `'symbols'` database.

compat (abridgment from compatibility)

This describes the 'behavior' of modifiers. XKB has some internal variables that store the state of the modifiers as well as the 'group'. These internal variables, along with the pressed key determine what character code is generated. The 'compat' files describe how these internal variables change when any modifier key is pressed. For example,

```
interpret Mode_switch {
     action= SetGroup(group=+1);
};
```

This means that the when the key which is mapped onto `'Mode_switch'` is pressed, the action should be to temporarily increment the group number by one. Essentially, the 'group' decides the keyboard layout. We have defined 'group' 1 to be US English layout, 'group' 2 to be Phonetic layout and 'group' 3 to be Inscript layout. These definitions are written in the 'symbols' database.

This file also describes the behavior of LED-indicators on the keyboard.

```
indicator "Caps Lock" {
     modifiers= Lock;
};
```

This means that when the Lock (Capslock) is pressed, the "Caps Lock" indicator is to be turned on.

symbols

This is the main table which specifies the mapping from scan-codes (symbolic names defined in 'keycodes') to the set of all possible values ('symbols'). A typical entry in the symbols file looks like the following:

```
key  <AE01> {[  1,    exclam ] };
```

This simple entry defines the set of symbols for the key '<AE01>'. Only one group is defined for this key. By default, this key is defined to be of type 'TWO_LEVEL', which is defined in the 'types' file as follows:

```
type "TWO_LEVEL" {
        modifiers = Shift;
        map[Shift] = Level2;
        level_name[Level1] = "Base";
        level_name[Level2] = "Shift";
    };
```

Therefore, this key, when pressed without any modifiers, will produce the symbol `'1'`. When the shift modifier is pressed, the symbol generated would be the level2 symbol i.e. `'exclam'`.

geometry

This file describes keyboard geometry – key placement on the physical keyboard. Two keyboards that have different physical key placements will have two different geometry files. Thus, the geometry file specifies the configuration of keyboards of different manufacturers.

The following is the list of files that have been added/modified for keyboard support in X. The directory is w.r.t the root directory of xkb /usr/X11R6/lib/X11/xkb

Directory	Added file/Modification
`Rules`	xfree86 modified to define new default files for 'compat' and 'ind' for 'in' layout
`Compat`	iso9995 modified to define new symbols: `ISO_Last_Group_Lock`, `ISO_First_Group_lock` and `ISO_Prev_Group_Lock`
	`Misc_in` added for leds
	`Ind` added as default configuration file for `'in'` layout

`Types`	'`indian`' file added to define new key types '`ind`' file added as default configuration file for '`in`' layout
`symbols`	'`in`' file added for defining Indian keyboard
`keymap`	'`xfree86`' modified to add an entry for Indian keymap.

Section 2: Programmer's manual for Xlib

Writing programs using modified Xlib:

- ISCII Text: ISCII text can be directly displayed using `XDrawString`.

  There is no change in the usage.

One needs to set the corresponding Hindi font and call the XDrawString function. Even if the text contains part of English text, the corresponding matching English font (found in the fonttable used by Isciilib) is loaded and the entire ISCII string gets printed properly. In case the matching font found in fonttable is could not be loaded, the English text is displayed using the Hindi font only. In case the ISCII text could not be properly parsed, then the unparsed string is printed using the font last used.

- UTF8 string: UTF8 string can be displayed using `XmbDrawString`

In this case the Text to be displayed is in UTF8 format and may be a mixture of multiple languages (depending on the UTF8 range). This involves the following steps

1. Setting the Indian locale. The Indian locale is loaded using the `setlocale()` function.
2. Creating the `fontset`: A `fontset` containing both the Indian scripts and an English script is created. `XmbDrawString` is used after creating the appropriate fontset. The extension of fonts (specifying the charset) for the Indian languages should be the same language (Ex: a font for devanagari would end with –devanagari-).
3. Call to `XmbDrawString`: `XmbDrawString` is called with the fontset and other arguments.

   Example code:

```
#include <locale.h>

#include<X11/Xlib.h>

void main(){
     XFontSet fs;
     char **missing_charset_list, *def_string;
     int missing_charset_count;

     Display *p_disp;
     Window Main;
     GC theGC;
     int x,y;
     //location on the screen where the text would
be printed

     char   *base_fontname_list   =   "dv_ttyogesh,
     as_ttyogesh,fixed"
```

```
/*for mixture of devanagari, Assamese and English
text */
```

```
        char *msgtext = "Hello \xe0\xa4\x85\xe0\xa4\
                        \xb0\xe0\xa4\xb5\xe0\xa4\xbf\xe0\
                       \xa4\xa8\xe0\xa5\xb0\xe0\xa4\xa6";
        char *locale;
        .
        .
        .
        locale = setlocale(LC_ALL, "in_IN.UTF-8");
        if(locale == NULL){
             printf("unable to load locale\n");
             exit(-1);
        }
        .
        .
        fs     =     XCreateFontSet     (p_disp,     \
        base_fontname_list,   &missing_charset_list,   \
        &missing_charset_count, &def_string);
        .
        .
        XmbDrawString(p_disp,Main,fs,theGC,x,y,msgtext\
        , strlen(msgtext));
        .
        .
}
```

Please note that the Indian locale has charsets defined for all the Indian scripts and for English scripts. Thus the list of missing charsets could contain the list of all those that could not be loaded. In case it contains the language that you intend to display then it means that there is no font that matched the proper extension. This needs to be handled. There is no separate Assamese range in UTF8. The Bengali range is used. Thus Assamese fonts will also have an extension of –bengali-.

- 4-byte Unicode string: Using XwcDrawString

The method is identical to printing UTF8 string except that one calls `XwcDrawString` instead on `XmbDrawString`.

For further details about programming in X, please refer to the X programming Manual.

Section 3: Administrator's Manual

3.1 Installing Xiscii

The Xiscii RPM contains the `libX11.so` library and some locale specific files. When the Xiscii RPM is installed this file in the `/usr/X11R6/lib` directory is overwritten. Also the file `locale.alias` in the directory `/usr/X11R6/lib/X11/locale` is overwritten. Thus f you are installing Xiscii for the first times using an RPM you might like to save these file so that you can revert to the unmodified X if you feel so.

To revert to the unmodified X just replace the file `libX11.so.6.2` with the file that you have saved. Also replace the `locale.alias` file. Some locale specific information might remain in the system but this would not have any affect on the functioning of X in any fashion.

3.2 Adding new Indian fonts

For the display of string using `XmbDrawString`, it is important that every font has the extension of the corresponding charset. However most fonts do not have any extension like the `ISO8859-1` for some English fonts. As a result it is important to give a different name to the font. This can be done my manually modifying the `fonts.dir` file in the directory in which the font is installed.

We suggest that you add another entry corresponding to the new name. Thus for a Devanagari font add an entry like the following:

```
#Default entries created in the font directory
DVYG0XTT.TTF -altsys-DV_TTYogesh-bold-i-normal--0-0-0-0-p-0-iso8859-1
DEV1.pfa        -unknown-shreedev001-medium-r-normal--0-0-0-0-p-0-adobe-
fontspecific

#Entries after changes are made manually
DVYG0XTT.TTF -altsys-DV_TTYogesh-bold-i-normal--0-0-0-0-p-0-iso8859-1
DVYG0XTT.TTF -altsys-DV_TTYogesh-bold-i-normal--0-0-0-0-p-0-devanagari-
DEV1.pfa        -unknown-shreedev001-medium-r-normal--0-0-0-0-p-0-adobe-
fontspecific
DEV1.pfa -unknown-shreedev001-medium-r-normal--0-0-0-0-p-0-devanagari-
```

3.3 Updating fonttable

As and when new fonts are added to the system, the `fonttable` (generally in the directory `/usr/share/fonts`) file might need some changes. A typical entry in the file is as follows:

```
   *dv_tt*        ISFOC_DEV   -adobe-courier-*-o-normal--22-
220-75-75-m-130-iso8859-1      0.75
```

The meaning of this entry is that font-names matching `*dv_tt*` have a configuration file `ISFOC_DEV`, the English font to be used in case of ASCII characters is the 3rd column and the 4th column gives the multiplication factor by which the point size of the Indian font should be multiplied to get an English font of the right point size.

If the names of the new fonts added do not match any of the existing entries in the `fonttable` you will need to make an entry so that the right matches are found by the

ISCIILIB as and when needed. If the configuration file is also different you might need to write a new configuration file depending on the font. (Please refer to the ISCIILIB manual to learn how to write a configuration file). One needs to be very careful when giving English font-names for two reasons.

1. The fontname must be complete, i.e. it must have all the fields. In case you leave a wildcard (*) for a particular field, then the corresponding field of the ISCII font is substituted. The fields of point-size, point-size ten times and average width would always be computed from the ISCII font by multiplying it with the multiplication factor. You also need to do some experimenting regarding the multiplication factor to be used.

2. Sometimes X fails to loads fonts even though one finds them listed using xlsfonts. In this case you need to again do some experimenting and see if the English text is being displayed properly. In case the matching font does not match, the English text would be displayed using ISCII font only.

3.4 Installing X keyboard

If you install the Indian keyboard using the RPM, you will also need to do some changes manually to start using the Indian keyboard. The RPM installs the Indian keyboard files at the appropriate places. The configuration file of X needs to be changed to start using this keyboard once it is installed. To do this, modify the file /etc/X11/XF86Config-4

Keyboard section as it would appear before.

```
#**************************************************
# Keyboard section
#**************************************************

Section "InputDevice"
    Identifier "Keyboard1"
    Driver      "Keyboard"
    Option "AutoRepeat"  "250 30"

    Option "XkbRules" "xfree86"
    Option "XkbModel" "pc105"
    Option "XkbLayout" "us"
EndSection
```

modify the line

```
<    Option “XkbLayout” “us”
>    Option “XkbLayout” “in”
```

Once these changes are done, you can start using the Indian keyboard when X restarts. (Typically on logging off.)